On Security Notions for Multi-Party Computation

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Sven Maier

aus Stuttgart

Tag der mündlichen Prüfung: 24. Oktober 2022
Erster Gutachter: Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter: Prof. Dr. Marc Fischlin

Acknowledgments

Writing this thesis has been a lot of work, yet this phase of my life has been made easier by the many people I've met during my time as a PhD student. I would like to take this opportunity to thank them.

First and foremost, I would like to thank my supervisor, Prof. Jörn Müller-Quade, for putting enough faith in me to offer me a job and to supervise me, and for being not only a great doctorate supervisor but also a remarkable person to talk to about stuff that is unrelated to work. Next, my deepest gratitude goes to Prof. Marc Fischlin, who agreed to take the time to act as the second referee of my thesis.

During my defense, I have had the honor of being additionally examined by Prof. Bernhard Beckert, Prof. Jörg Henkel, and Prof. Hannes Hartenstein. Special thanks to them for taking their time and making something I was initially very nervous about a lot more pleasant.

During my time at KIT, I've had the honor of working with outstanding co-authors, namely, in alphabetical order, Thomas Agrikola, Nicholas-Philip Brandt, Geoffroy Couteau, Nico Döttling, Valerie Fetzer, Marcel Keller, Alexander Koch, Jeremias Mechler, Anne Müller, Tobias Müller, Jörn Müller-Quade, Markus Raiber, Andy Rupp, Rebecca Schwerdt, and Marcel Tiepelt. Thank you all for many fruitful discussions, working with you has taught me a lot.

Special thanks also goes to Andy Rupp and Geoffroy Couteau, who supported me even to the point where it feels like they were additional supervisors. Without the two of you, I genuinely believe that I wouldn't have my PhD now. Thank you so much.

Also, special thanks to Thomas Agrikola, who read through many parts of my dissertation and provided helpful feedback.

I was fortunate enough to have met some wonderful colleagues, without whom my time at KIT wouldn't have been nearly as fun as it was. In particular, I'd like to thank (in alphabetical order) Thomas Agrikola for helping me fight through the jungle of KIT regularities required for a dissertation and fixing my weird LaTeX problems as well as organizing hiking trips, Geoffroy Couteau for his infectious love for anything related to cryptography and his incredible patience during discussions, Valerie Fetzer for an amazing trip to Japan, Willi Geiselmann for always having an open ear and suggestions when things did not work out the way they should have, Simon Hanisch for organizing board game nights, Gunnar Hartung for planning and organizing the Diablo 3 videogame sessions, Killian Herveau for awesome cooking parties and his unending supply of good mood, Julia

Kastner for many hours of bouldering together and for teaching me a lot about squirrels, Michael Klooß for knowing probably any good videogame-soundtrack out there and his attention to detail if it comes to definitions, Alexander Koch for interesting philosophical discussions on Effective Altruism, Jessica Koch for still deciding to join our Mario Kart game night despite having deadlines, Carmen Manietta for keeping the institute running, Jeremias Mechler for interesting conversations on practical IT Security and ways to secure the own operating system, Markus Raiber for his endless knowledge on books, Andy Rupp for always taking his time for research discussions to fully understand a given problem even when he had other things to do, Rebecca Schwerdt for making sure I always have a sufficient supply of Tea, Mario Strefler for taking over organizational tasks to make sure I had enough time to do research and for starting the running group, Marcel Tiepelt for convincing me to join said running group and for his incredible love for food, Akin Ünal for organizing great activities in- and outside of the institute and for ensuring the institutes supply of powerful mouthwash, Bogdan Ursu for many competitive table tennis matches and for inventing Bogdan-style-bouldering, and Thomas Worsch for making sure there is always enough coffee at the institute to keep it from collapsing.

Finally, I would like to thank my friends and family for their encouragement and support.

Abstract

Most of the security notions used which are currently used originate from the 1980s. Yet a better understanding of the theory since then opens the question whether they can be refined.

A limiting factor here are so-called *impossibility proofs*, which prove mathematically which security guarantees cannot be fulfilled. These provide a limiting factor, yet their statement should not be overstated. The proof is only valid in its own setting and only covers one specific security notion. Historically, the established security notions settled for something much weaker, leaving a gap between what is practically used and what is known to be impossible.

In this thesis, we shed light on some of these gaps and investigate security notions related to Secure Multi-Party Computation (MPC) lying between those that have been established and those that are known to be impossible.

Modelling Business Models and Legal Regulations with MPC. With Secure Multi-Party Computation (MPC), parties can securely compute a function on private inputs in such a way, that no information on the other parties inputs is leaked except for the functions output. Nowadays, MPC only has a relatively small overhead over the direct computation. Yet even though data economy is incentivized in practice, MPC is not often used in practice.

We believe that one of the reasons MPC is rarely used in practice is that it ignores business models and legal requirements that require a certain amount of leakage for the data, while generic MPC aims for near-perfect privacy.

We present a novel building block that enables businesses—represented by a central *operator*—to model the desired amount of leakage required to maintain business or to fulfill the legal requirements efficiently, while letting users collect data anonymously and without being linked throughout interactions.

We model the requirements in the Universal Composability (UC) framework. This guarantees that security holds regardless which protocols are executed in parallel. Despite strong security guarantees the protocol is still efficient enough to be executed on state-of-the-art hardware, even if the user collect data on smartphones which have limited computation power.

Fetzer, Keller, Maier, Raiber, Rupp, Schwerdt, PETS 2022

An Instantiation of Stronger Commitments. A bit commitment scheme allows a sender to fix a bit towards a receiver without revealing it in the process (*hiding*), but such that the sender cannot convince the receiver that the commitment was on a different bit (*binding*).

In the quantum world, commitments are strong enough to provide MPC, so there is a strong incentive to make commitments as secure as possible; yet impossibility results state that both security properties—hiding and binding—cannot simultaneously hold unconditionally. As a consequence, modern bit commitment schemes relax one property to hold only computationally, *i.e.* based on a computational hardness assumption.

We introduce the first bit commitment protocol in the Quantum Random Oracle Model (QROM) that achieves unconditional security for the receiver (binding) and *everlasting* security for the sender (hiding) and that does not require additional hardware. Our result is based on a new assumption on the hardness of storing quantum states over a long period of time. Everlasting security models technical progress of the adversary, as a transcript that cannot be efficiently broken nowadays might be easy to extract once faster machines are available.

We prove the commitment protocol secure in the QROM under the aforementioned assumption and note that an instantiation in the standard model leads to a new attack on the everlasting hiding property.

Döttling, Koch, Maier, Mechler, Müller, Müller-Quade, Tiepelt, IN SUBMISSION

Undetectable Multi-Party Computation. Covert MPC is an extension to MPC that not only hides the inputs, but the entire presence of the computation. Parties only learn the output if all other parties followed the protocol correctly, and if the output is *favorable* for all parties. Otherwise the parties learn nothing, not even which parties tried to participate.

A single non-participant can accidentally abort the entire computation. The question arises: Can N parties perform the computation while $K > N$ parties are present, where the output only depends on the inputs of the N participants, while hiding the identity of the other participants among all individuals present? This should hold in particular if the remaining parties are unaware that a computation is in progress.

We relate this question to the theoretical feasibility of *anonymous whistleblowing*, where a single party discloses a message without revealing the own identity and without other parties having to act in any special way. Unfortunately, we show that no primitive can fulfill both correctness and anonymity with overwhelming probability in the *asymptotic* setting, even under very strong assumptions. Yet we provide a heuristic instantiation in the *fine-grained* setting with overwhelming correctness and any given target anonymity. Our results provide strong foundations for the study of the possibility of anonymous communications through authenticated channels, an intriguing goal which we believe to be of fundamental interest.

Agrikola, Couteau, Maier, TCC 2022

Zusammenfassung

Die meisten Sicherheitsbegriffe, die heutzutage benutzt werden, stammen aus den 1980ern. Doch durch ein seitdem besseres Verständnis der Theorie stellt sich die Frage, ob sie nicht weiterentwickelt werden können.

Ein begrenzender Faktor sind hierbei sogenannte *Unmöglichkeitsbeweise*, die mathematisch beweisen, welche Sicherheitsgarantien nicht erfüllt werden können. Diese liefern einen begrenzenden Faktor, ihre Aussage sollte jedoch nicht übertrieben werden. Der Beweis ist nur in seinem eigenen Setting gültig und deckt nur genau den einen Sicherheitsbegriff ab. Historisch haben sich die etablierten Sicherheitsbegriffe jedoch zu etwas deutlich schwächerem entwickelt, wodurch eine Lücke zwischen dem entstanden ist, was praktisch benutzt wird, und dem, was bekanntermaßen unmöglich ist.

In dieser Promotion zeigen wir einige dieser Lücken auf und untersuchen Sicherheitsbegriffe, die mit Sicherer Mehrparteienberechnung (MPC) zusammenhängen, und die zwischen den Etablierten und den Unmöglichen liegen.

Abbildung von Geschäftsmodellen und Gesetzlichen Regelungen in MPC. Mit Sicherer Mehrparteienberechnung (MPC) können Parteien eine Funktion über privaten Eingaben auf sichere Weise so berechnen, dass nichts über die Eingaben der anderen Parteien bekannt wird außer die Ausgabe der Funktion. Heutzutage hat MPC nur einen vergleichsweise geringen Mehraufwand im Vergleich zur direkten Berechnung. Und obwohl Datensparsamkeit in der Praxis belohnt wird, wird MPC kaum benutzt.

Wir glauben dass einer der Gründe dafür, dass MPC in Praxis kaum benutzt wird, darin liegt, dass es Geschäftsmodelle und gesetzliche Regelungen ignoriert die eine gewisse Leakage der Daten benötigen, während allgemeines MPC auf fast-perfekte Privatsphäre hinarbeitet.

Wir präsentieren einen neuen Baustein, der es Geschäften—die durch einen zentralen *Operator* repräsentiert werden—ermöglicht, effizient die gewünschte Menge an Leakage abzubilden, die benötigt wird, um das Geschäft aufrechtzuerhalten oder um gesetzliche Vorgaben zu erfüllen, während Nutzer anonym und ohne durch mehrere Interaktionen hinweg verlinkt werden können Daten sammeln.

Wir modellieren die Anforderungen im Universal Composability (UC) Framework. Dadurch wird garantiert, dass die Sicherheitsgarantien unabhängig davon halten, welche Protokolle parallel ausgeführt werden. Trotz dieser starken Sicherheitsgarantien ist das Protokoll

dabei effizient genug, um auf moderner Hardware ausgeführt zu werden, selbst wenn der Nutzer die Daten auf Smartphones mit beschränkter Rechenleistung sammeln.

Fetzer, Keller, Maier, Raiber, Rupp, Schwerdt, PETS 2022

Eine Instantiierung stärkerer Commitments. Mit einem Bit Commitment Schema kann sich ein Sender gegenüber eines Empfängers auf ein Bit festlegen, ohne das dabei zu offenbaren (*hiding*), aber auf eine Art die es dem Sender nicht erlaubt, den Empfänger später davon zu überzeugen, dass das Commitment auf ein anderes Bit festgelegt wurde (*binding*).

In der Quantenwelt sind Commitments stark genug, um MPC zu konstruieren, weswegen es einen Anreiz gibt, Commitments so sicher wie möglich zu machen; jedoch sagen Unmöglichkeitsbeweise aus, dass beide Sicherheitsbegriffe – hiding und binding – gleichzeitig nicht bedingungslos halten können. Als Konsequenz weichen moderne Bit Commitment Schemas eine Sicherheitseigenschaft auf, die dann nur noch *computationally* halten, also auf Grundlage komplexitätstheoretischer Annahmen.

Wir stellen das erste Bit Commitment Protokoll im Quantum Random Oracle Model (QROM) vor, das bedingungslose Sicherheit für den Empfänger (binding) und langfristige Sicherheit für den Sender (hiding) bietet und das dabei keine Zusatzhardware benötigt. Unser Resultat basiert auf einer neuen Annahme über die Schwierigkeit, Quantenzustände über einen langen Zeitraum zu speichern. Langfristige Sicherheit modelliert technischen Fortschritt des Angreifers, da Transkripte, die heutzutage nicht effizient gebrochen werden können, in Zukunft vielleicht einfach extrahierbar sind, sobald schnellere Maschinen verfügbar sind.

Wir beweisen die Sicherheit des Commitment Protokolls im QROM unter oben genannter Annahme und zeigen, dass eine Instantiierung im Standardmodell zu einem neuen Angriff auf die langfristige Hiding-Eigenschaft zulässt.

Döttling, Koch, Maier, Mechler, Müller, Müller-Quade, Tiepelt, IN EINREICHUNG

Undetectable Multi-Party Computation. Covert MPC ist eine Erweiterung von MPC, die nicht nur die Eingaben versteckt, sondern das gesamte Vorhandensein der Berechnung. Teilnehmer lernen nur dann die Ausgabe, wenn alle anderen Parteien das Protokoll ausgeführt haben und die Ausgabe für alle Parteien *vorteilhaft* ist. Anderenfalls lernen die Teilnehmer nichts, nicht mal, welche anderen Parteien versucht haben, an der Berechnung teilzunehmen.

Ein einzelner Nichtteilnehmer kann unabsichtlich die gesamte Berechnung abbrechen. Daher stellt sich die Frage: können N Teilnehmer eine Berechnung ausführen, während $K > N$ Parteien anwesend sind, und bei der die Ausgabe nur von den Eingaben der N Teilnehmer abhängt, während die Identität der anderen Teilnehmer unter den anwesenden Parteien versteckt wird? Dies sollte insbesondere dann gelten, wenn die restlichen Parteien nicht wissen, dass eine Berechnung im Gang ist.

Wir verknüpfen diese Frage mit der theoretischen Machbarkeit von *Anonymen Whistleblowing*, bei dem eine einzelne Partei versucht, eine Nachricht preiszugeben, ohne dabei die eigene Identität zu offenbaren und ohne dass sich die anderen Parteien auf irgendeine besondere Art verhalten müssen. Leider zeigen wir dass keine Primitive sowohl Korrektheit und Anonymität mit überwältigender Wahrscheinlichkeit im asymptotischen Setting erreichen kann, selbst unter sehr starken Annahmen. Jedoch konstruieren wir eine heuristische Instantiierung im *Fine-Grained* setting mit überwältigender Korrektheit und jeder beliebigen Ziel-Anonymität. Unsere Ergebnisse liefern starke Grundlagen für die Untersuchung der Möglichkeit von Anonymen Nachrichtentransfer durch authentifizierte Kanäle, ein faszinierendes Ziel von dem wir glauben, dass es von grundlegendem Interesse ist.

Agrikola, Couteau, Maier, TCC 2022

Own Publications

[1] Thomas Agrikola, Geoffroy Couteau, and Sven Maier. “Anonymous Whistleblowing over Authenticated Channels”. In: *TCC 2022*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Lecture Notes in Computer Science. To appear. Springer, Heidelberg, Germany, Nov. 2022.

[2] Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. *Constructing Secure Multi-Party Computation with Identifiable Abort*. Cryptology ePrint Archive, Report 2020/153. https://eprint.iacr.org/2020/153. 2020.

[3] Nico Döttling, Alexander Koch, Sven Maier, Jeremias Mechler, Anne Müller, Jörn Müller-Quade, and Marcel Tiepelt. *Towards Everlasting Bit Commitment from Quantum Decay*. Unpublished manuscript. 2022.

[4] Valerie Fetzer, Marcel Keller, Sven Maier, Markus Raiber, Andy Rupp, and Rebecca Schwerdt. “PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics”. In: *Proceedings on Privacy Enhancing Technologies* 2022.2 (Apr. 2022), pp. 447–516. DOI: 10.2478/popets-2022-0054.

Contents

Abstract . . . iii

Zusammenfassung . . . v

Own Publications . . . ix

List of Figures . . . xvii

List of Tables . . . xxi

1. Introduction . . . **1**
- 1.1. Modelling Business Models and Legal Regulations with MPC . . . 4
- 1.2. An Instantiation of Stronger Commitments . . . 5
- 1.3. Undetectable Multi-Party Computation . . . 6

2. Preliminaries . . . **9**
- 2.1. Notation . . . 9
- 2.2. Commitments . . . 10
  - 2.2.1. Definition . . . 10
  - 2.2.2. Correctness . . . 11
  - 2.2.3. Hiding Commitment Schemes . . . 11
  - 2.2.4. Binding Commitment Schemes . . . 12
- 2.3. Signature Schemes . . . 12
  - 2.3.1. Definition . . . 13
  - 2.3.2. Correctness . . . 13
  - 2.3.3. Existential Unforgeability under Chosen Message Attacks . . . 13
- 2.4. Encryption Schemes . . . 14
  - 2.4.1. Definition . . . 14
  - 2.4.2. Correctness . . . 15
  - 2.4.3. Indistinguishability under Chosen Plaintext Attacks . . . 15

I. MPC for Business Models and Legal Regulations **17**

3. Introduction . . . **19**
- 3.1. Our Contribution . . . 20
- 3.2. Related Work . . . 21

4. Preliminaries . . . **23**
4.1. Notation . . . 23
4.2. Commitments . . . 23
4.2.1. Additively Homomorphic Commitment Schemes . . . 23
4.2.2. Structure Preserving Commitment Schemes . . . 24
4.3. Signature Schemes . . . 24
4.3.1. Structure Preserving Signature Schemes . . . 24
4.4. Zero Knowledge Schemes . . . 25
4.4.1. Perfect Completeness . . . 25
4.4.2. Perfect Soundness . . . 26
4.4.3. CRS Indistinguishability . . . 26
4.4.4. Perfect $F_{pp_{\mathbb{G}}}$-Extractability . . . 26
4.4.5. Dual-Mode . . . 27
4.4.6. Dual-Mode Zero Knowledge Scheme . . . 27
4.5. The Universal Composability Framework . . . 28
4.5.1. Description . . . 28
4.5.2. UC Subfunctionalities . . . 28

5. Overview . . . **35**
5.1. Parties and Roles . . . 35
5.2. Tasks . . . 36
5.2.1. Preparatory Tasks . . . 37
5.2.2. Bookkeeping . . . 38
5.2.3. Outsourcing Analytical Computations . . . 38
5.3. Cryptographic Building Blocks . . . 40
5.4. Set-Up Assumptions . . . 41
5.5. Computation of Benign Functions f . . . 42
5.6. The User Logbook . . . 43
5.7. General Principles . . . 44
5.8. Security Guarantees . . . 45
5.8.1. Operator . . . 45
5.8.2. User . . . 47
5.9. Realizing the Individual Tasks . . . 48
5.10. Limitations of our scheme . . . 49
5.10.1. Verification by the TSA . . . 49
5.10.2. The Trouble of Aborts . . . 50

6. Ideal Functionality . . . **53**

7. Protocol . . . **59**
7.1. Benign Functions . . . 59
7.1.1. Bookkeeping . . . 59
7.1.2. Outsourced Analytics . . . 60
7.2. Robust Secret Sharing . . . 60
7.3. Anonymous Communication . . . 62

7.4. Our Protocol . . . 63

8. Security . . . 79
8.1. User Security . . . 79
8.2. Operator Security . . . 100

9. Example Applications . . . 125
9.1. Fraud Detection for Mobile Payments . . . 125
9.2. Targeted Advertisement System . . . 129

10. Implementation . . . 131
10.1. Setup . . . 131
10.2. Evaluation . . . 131
10.3. Discussion . . . 134
10.4. Performance of Fraud Detection . . . 134

11. Conclusion . . . 137

II. An Instantiation of Everlasting Secure Commitments 139

12. Introduction . . . 141
12.1. Contribution . . . 142
12.2. Related Work . . . 144

13. Preliminaries . . . 147
13.1. Notation . . . 147
13.2. Quantum Information . . . 147
13.2.1. Security Notion . . . 149
13.2.2. Conjugate Coding . . . 150
13.2.3. Quantum Bit Commitment . . . 152
13.2.4. Fully Homomorphic Encryption . . . 152
13.2.5. q-IND-CPA . . . 153
13.2.6. Quantum Fully Homomorphic Encryption . . . 154
13.3. Security Model . . . 155
13.3.1. Quantum Random Oracle Model . . . 155
13.3.2. Everlasting Security . . . 156
13.3.3. Quantum Random Oracles with Everlasting Security . . . 157

14. Impossibility Result . . . 159

15. Quantum Decay . . . 163
15.1. Motivating the Assumption . . . 163
15.2. Mathematical Description . . . 164
15.3. Simulating QROs . . . 166

16. Everlasting Quantum Commitment Protocol . . . **169**
16.1. The BCJL Protocol . . . 169
16.2. Everlasting Quantum Commitment Protocol . . . 170
16.2.1. The protocol . . . 171

17. Instantiating the Classical Commitment in the Quantum Random Oracle Model **173**
17.1. Closeness-Testing of Quantum Random Oracles . . . 173
17.2. A Classical Commitment Protocol . . . 176
17.3. Security Analysis . . . 177
17.3.1. Unconditional Binding Property . . . 177
17.3.2. Quantum-Computational Hiding Property . . . 177

18. Proof of Security of Π_{QCom} . . . **181**
18.1. Sender Security . . . 181
18.2. Receiver Security . . . 188

19. An Obfuscated Measurement Attack . . . **191**
19.1. The High-level Idea . . . 191
19.2. A Partially Homomorphic Commitment Scheme . . . 192
19.2.1. Sender Security . . . 194
19.2.2. Receiver Security . . . 196
19.3. An Obfuscated Measurement Attack . . . 197
19.3.1. Description . . . 197

20. Conclusion . . . **201**

III. Anonymous Whistleblowing **203**

21. Introduction . . . **205**
21.1. Contribution . . . 207
21.2. Related Work . . . 210

22. Preliminaries . . . **213**
22.1. Steganography . . . 213
22.2. Distribution Testing . . . 214
22.3. Covert Oblivious Transfer . . . 217
22.4. Indistinguishability from Random under Chosen Ciphertext Attacks . . . 217
22.5. Strong Existential Unforgeability under Chosen Message Attacks . . . 218
22.6. Ideal Obfuscation . . . 218

23. Anonymous Transfer . . . **221**
23.1. Network Model and Non-Participating Parties . . . 221
23.2. The Model . . . 221
23.3. Fine-grained Anonymous Transfer . . . 223
23.4. Trivial Anonymous Transfers . . . 224

23.5. Reductions Among AT Protocols 225
23.5.1. AT implies silent-receiver AT. 225
23.6. Strong Anonymous Transfer 226

24. Impossibility of Anonymous Transfer 227
24.1. The Attacker 227
24.2. Putting the Pieces Together 234
24.3. Impossibility of Anonymous Transfer for $K > 3$ 235
24.3.1. Security Analysis of Π'_{AT} 236
24.4. Extensions and Limitations 239
24.4.1. Limitations of the impossibility result. 240

25. Fine-Grained AT from Ideal Obfuscation 241
25.1. The Protocol 241
25.2. Security Analysis 244
25.2.1. Correctness 244
25.2.2. Anonymity. 245
25.2.3. Secrecy 260

26. A fine-grained Anonymous Transfer for n-bit messages 265
26.1. The Protocol 265
26.2. Security Analysis 265
26.2.1. Correctness 265
26.2.2. Anonymity 266
26.2.3. Secrecy 273

27. Undetectable Oblivious Transfer 277
27.1. Definitions for Undetectable Oblivious Transfer 277

28. Undetectable Oblivious Transfer Instantiation 281
28.1. Correctness 281
28.2. Privacy 283
28.2.1. Sender Privacy 283
28.3. Anonymity 285
28.3.1. Corrupted Sender 286
28.3.2. Corrupted Receiver 287
28.3.3. Corrupted Dummy Friend 289

29. Towards Undetectable Multi-Party Computation 293
29.1. Defining Undetectable Multi-Party Computation 293
29.2. Towards constructing Undetectable Two-Party Computation from Undetectable Oblivious Transfer 293

30. Conclusion 295

Bibliography 311

List of Figures

4.1. The functionality $\mathcal{F}_{\mathsf{CRS}}$ for setting up a Common Reference String. 29
4.2. The functionality $\mathcal{F}_{\mathsf{Reg}}$ for registering public keys. 30
4.3. The functionality $\mathcal{F}_{\mathsf{SMT}}$ for secure message transfer. 30
4.4. The functionality $\mathcal{F}_{\mathsf{ORR}}$ for Onion-Routing with Replies. 32
4.5. Procedure $\texttt{Process\_Next\_Step}(O)$ for an onion O. 33

5.1. Overview of the preparatory tasks. 37
5.2. Overview of the Bookkeeping task. 38
5.3. Overview of tasks for outsourcing computations. 39

6.1. The first part of the basic functionality $\mathcal{F}_{\mathsf{BKA}}$. 54
6.2. The second part of the basic functionality $\mathcal{F}_{\mathsf{BKA}}$. 55

7.1. The procedure $\texttt{Share}$ for the protocol Π_{BKA}. 61
7.2. The procedure $\texttt{Combine}$ for the protocol Π_{BKA}. 61
7.3. The procedure $\texttt{Verify}$ for the protocol Π_{BKA}. 62
7.4. The procedure $\texttt{Send}$ for the protocol Π_{BKA}. 62
7.5. The first part of the protocol Π_{BKA} that specifies the parties state and the initialization part of the protocol. 63
7.6. The second part of the protocol Π_{BKA} that specifies the behavior for signing function parameter. 64
7.7. The third part of the protocol Π_{BKA} that specifies the behavior for user registration. 65
7.8. Language $\Lambda_{\mathrm{UserReg}}$ used for the $\texttt{UserRegistration}$ task. 65
7.9. The fifth part of the protocol Π_{BKA} with the first part on how to compute a function f. 67
7.10. The sixth part of the protocol Π_{BKA} that specifies how to compute a function f. 68
7.11. The seventh part of the protocol Π_{BKA} that specifies how to compute a function f. 69
7.12. Language $\Lambda_{\mathrm{Compute}}$ used for the $\texttt{Compute}$ task. 69
7.13. Language $\Lambda_{\mathrm{Transfer}}$ used for the Bookkeeping and Update task. 70
7.14. The eighth part of the protocol Π_{BKA} with the first part on how to outsource a function f. 71
7.15. The nineth part of the protocol Π_{BKA} that specifies how to outsource a function f. 72
7.16. The tenth part of the protocol Π_{BKA} that specifies how to outsource a function f. 73
7.17. Language $\Lambda_{\mathrm{Outsource}}$ used for the $\texttt{Outsource}$ task. 73

7.18. The eleventh part of the protocol Π_{BKA} with the computation of an analytical function f. 74
7.19. The twelveth part of the protocol Π_{BKA} with the first part on how to apply the update of an outsourced analytical computation. 75
7.20. The thirteenth part of the protocol Π_{BKA} with the second part on how to apply the update of an outsourced analytical computation. 76
7.21. The fourteenth part of the protocol Π_{BKA} with the third part on how to apply the update of an outsourced analytical computation. 77
7.22. Language Λ_{Update} used for the `Update` task. 77

8.1. The first part of the simulator with an honest user: Defines state and set up. 80
8.2. The second part of the simulator with an honest user and corrupt operator: Defines initialization. 80
8.3. The third part of the simulator with an honest user and corrupt operator: Defines behavior for signing function parameter. 81
8.4. The fourth part of the simulator with an honest user and corrupt operator: Defines behavior for user registration. 81
8.5. The fifth part of the simulator with an honest user and corrupt operator: Defines behavior for computation of a function f. 82
8.6. The sixth part of the simulator with an honest user, honest helper and corrupt operator: Defines behavior for outsourcing of a function f. 83
8.7. The seventh part of the simulator with an honest user, corrupted helper and corrupt operator: Defines behavior for outsourcing of a function f. 84
8.8. The eighth part of the simulator with an honest helper and corrupt operator: Defines behavior for computing an outsourced function f. 85
8.9. The ninth part of the simulator with an honest user and helper and corrupt operator: Defines behavior for updating the user data after outsourcing an analytical computation. 86
8.10. The tenth part of the simulator with an honest helper and corrupt operator and user: Defines behavior for updating the user data after outsourcing an analytical computation. 87
8.11. The first part of the simulator with an honest operator: Defines state and set up. 101
8.12. The second part of the simulator with an honest operator and corrupt user: Defines initialization. 101
8.13. The third part of the simulator with an honest operator and TSA: Defines behavior for signing function parameter. 102
8.14. The fourth part of the simulator with an honest operator and honest user: Defines user registration. 102
8.15. The fifth part of the simulator with an honest operator and corrupt user: Defines behavior for user registration. 102
8.16. The sixth part of the simulator with an honest operator and corrupt user: Defines behavior for computation of a function f. 103
8.17. The seventh part of the simulator with an honest operator, user, and helper: Defines behavior for outsourcing a function f. 104

8.18. The eighth part of the simulator with an honest operator and helper and a corrupted user: Defines behavior for outsourcing a function f. 104
8.19. The ninth part of the simulator with an honest operator and user and a corrupted helper: Defines behavior for outsourcing a function f. 105
8.20. The tenth part of the simulator with an honest operator and corrupted user and helper: Defines behavior for outsourcing a function f. 106
8.21. The eleventh part of the simulator with an honest operator and helper: Defines behavior for computing an outsourced a function f. 107
8.22. The twelfth part of the simulator with an honest operator and corrupted helper: Defines behavior for computing an outsourced a function f. 108
8.23. The thirteenth part of the simulator with an honest operator and helper and a corrupted user: Defines behavior for updating the user data. 109
8.24. The fourteenth part of the simulator with an honest operator and user and a corrupted helper: Defines behavior for outsourcing a function f. 110
8.25. The fiveteenth part of the simulator with an honest operator and corrupted user and helper: Defines behavior for outsourcing a function f. 111

9.1. Instantiation of f for privacy-preserving mobile payments with fraud detection. 126
9.2. Instantiation of f for a privacy-preserving targeted advertisement system. . 130

16.1. The commitment protocol Π_{Com} from Brassard et al. [36] for committing to a given bit $b \in \{0, 1\}$. 170
16.2. The unveil protocol Π_{Unv} from Brassard et al. [36] for unveiling a commitment. 170
16.3. The everlasting commitment protocol Π_{Com} for committing to a given bit $b \in \{0, 1\}$. The shaded area shows the extension of the protocol from Fig. 16.1. 171
16.4. The everlasting unveil protocol Π_{Unv} for unveiling a commitment. 172

17.1. The classical commitment protocol Π_{Com} in the QROM for committing to a given bit $b \in \{0, 1\}$. 176
17.2. The classical unveil protocol Π_{Unv} in the QROM for unveiling a commitment. 176
17.3. Hiding games for the commitment Com. 178

18.1. Games to show independence of chosen basis vectors. 182
18.2. Depiction of the oracle selection process. $|\mathrm{cod}(R_Q)\rangle$ contains the output of the challenge oracle R_Q. $|\mathrm{dom}(R_S)\rangle$ contains the query sent to the simulated oracle R_S. 185
18.3. Quantum circuit to compare inputs. 186
18.4. Circuit-based description. 187

19.1. The classical commitment protocol Π_{Com} in the standard model for committing to a given bit $b \in \{0, 1\}$. 193
19.2. The classical unveil protocol Π_{Unv} in the standard model for unveiling a commitment. 193
19.3. Hiding games for the commitment Com*. 194
19.4. QPT part of the Obfuscated Measurement Attack to break the everlasting hiding property. 197

19.5. Unbounded part of the Obfuscated Measurement Attack to break the everlasting hiding property. 198

23.1. Definition of the anonymity experiment $\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^n},\mathcal{A},\phi}(\kappa)$. 223

24.1. Definition of the first part of the guessing algorithm, $\mathrm{A}_{guess_0}^{\Xi}$, against the δ-anonymity of the silent-receiver κ-bit AT protocol Π_{AT^κ}, parameterized by a polynomial $\Xi = \Xi(\kappa)$. 228
24.2. Definition of the second part of the guessing algorithm, $\mathrm{A}_{guess_1}^{\Xi}$, against the δ-anonymity of the silent-receiver κ-bit AT protocol Π_{AT^κ}, parameterized by a polynomial $\Xi = \Xi(\kappa)$. 229
24.3. The protocol Π'_{AT} that constructs $(K-1)$-party AT from a given K-party SRAT for K parties. 235

25.1. The protocol Π_{AT} for fine-grained Anonymous Transfer. 242
25.2. Obfuscated program P_{FG-AT} for a single-bit Anonymous Transfer in the fine-grained setting with c rounds. 243
25.3. Game to distinguish whether Bernoulli oracles follow a given distribution p or $q = p - 1/2c$. 256

26.1. The protocol Π_{AT^n} for fine-grained n-bit Anonymous Transfer. 266
26.2. Obfuscated program P_{FG-AT} for an n-bit Anonymous Transfer in the fine-grained setting with c rounds. 267
26.3. Game to determine whether c Bernoulli-oracles follow a given distribution p_χ or $q_\chi = p - 1/c$. 270

27.1. Definition of the game $\mathrm{Exp}^{\mathsf{anon\text{-}ot}}_{\Pi_{UOT},\mathcal{A},\varpi}(\kappa)$. 278

28.1. The two-round protocol Π_{UOT} for Undetectable Oblivious Transfer with given protocols for Anonymous Transfer and Covert Oblivious Transfer. 282
28.2. The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the sender. 285
28.3. The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the receiver. 287
28.4. The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the dummy friend. 289

List of Tables

10.1. Execution times in *ms* for operator and helper. $\sum$ is the total time for the operator, **ZK** the time spent on verifying the ZK proof. H is the execution time of the helper. 132
10.2. Data exchanged in *kB* in relation to the size of the UH. BK denotes a BK task with non-trivial shift and permutation. BK' denotes BK with only an incremental update. 132
10.3. Execution time in *ms* for user. U_1 uses phone 1, U_2 uses phone 2, $\sum$ is the total user execution time, of which **On** is the online running time, **ZK** the creation of the ZK proofs, **Val** the time for validating the UH, and **PC** the precomputation time. 132
10.4. Time (in seconds) and communication (in megabyte) for logistic regression with two parties. *Strong* uses c5.9xlarge instances, otherwise we use m4.large. 133

19.1. Possible quantum states $|\phi\rangle = Z^{(z)}X^{(x)}|\psi\rangle$ a QOTP encryption of $|\psi\rangle = |0\rangle$ or $|\psi\rangle = |1\rangle$ can be in. For the sake of simplicity, we assume $-|1\rangle = |1\rangle$. 199

1. Introduction

Historically, cryptography has established itself as the art of hiding information, which has been around even in ancient Greek. The task at hand seems relatively simple: How can we transfer a message from a sender to a receiver in such a way, that the receiver can read the message, but no opposing party who might capture the messenger can make sense of the message?

Although the methods of hiding and transporting the sensitive information have changed since then, cryptography was still mostly related to hiding information. This changed in the 1980s, when Yao [139] formulated and solved the problem of Secure Multi-Party Computation (MPC), which can be best explained by his Millionaires' Problem [140]:

Two millionaires P_0 and P_1 each have individual assets x_0 and x_1, respectively. They want to find out who of them has more money, without revealing any further information in the process; in particular, the computation should not enable any of the two parties to infer the difference $|x_0 - x_1|$ of their assets. It is thus infeasible for any one party to simply publish the own assets and to ask the other party to publish whether the own value is above or below the published value. Instead, Yao [139] solved the problem using cryptographic techniques like the then-new Oblivious Transfer (OT) [116, 69].

This gave rise to the exciting new research field of MPC which is still under investigation to date. The more generalized formulation replaces the two millionaires by an arbitrary number N of participants $(\mathsf{P}_1, \ldots, \mathsf{P}_N)$ and the function that essentially outputs 1 if $x_1 > x_0$ and 0 otherwise by an *arbitrary* function f. The earliest positive results of Yao [139], Beaver, Micali, and Rogaway [21], and Goldreich, Micali, and Wigderson [77] were only meant as possibility results; the method used by the latter in particular was highly improved in terms of practical efficiency since then.

Modern MPC protocols provide a relatively low overhead compared to the direct computation with the actual values, in particular when assuming that parties do not deviate from the protocol (*semi-honest*), where general-purpose solutions based on the ABY-framework [61] and special purpose solutions like GAZELLE [94] for neural networks can be used, but also if we fear that participants might deviate from the protocol in order to learn new information (*malicious*), the SPDZ-family founded by Damgård et al. [58] provides a relatively small overhead.

So all in all, it is possible for a set of N parties to compute any given function f securely, as long as f is efficiently computable. But what does it mean for a computation to be secure? Intuitively, it means that the parties are unable to learn any information from the protocol execution except for the output and anything that can be inferred from that. In particular, they learn no additional information on *any* other parties' secret input.

In practice, this is modeled by following the real-ideal-paradigm [79, 77]. To better understand this, let us imagine some parallel world that would be *ideal* for MPC. In this world, there is one party which is incorruptible, discrete and who is trusted by everyone. Every party that wants to participate has a secure connection to this trusted party and can hence interact both in an *authenticated* (in that no other party can impersonate anyone else) and *secure* (meaning that all messages can only be read by the participant and the trusted party) way.

In this idealized world, the problem of MPC can be solved as follows: Each participant sends the own input x_i to the trusted party. Once the trusted party has received input by all N parties, it computes $y \leftarrow \mathsf{f}(x_1, \ldots, x_N)$ (with fresh random coins if f is probabilistic) and outputs the result to all participants.

Unfortunately, this ideal setting is quite unrealistic for almost all scenarios where we would want to use MPC; it is quite difficult to have even small parties agree on a neutral and commonly trusted party to perform the computation, but for problems such as elections with competing (and often distrustful) parties this becomes almost impossible. Instead, we consider actual protocols that are executed in the *real* world, where N parties—who likely distrust each other—execute a protocol that, despite the mutual distrust and the absence of a single trusted party, is *as secure* as the execution in the ideal world. This implies that the computation according to the protocol yields the same result and provides *no additional information* on the other parties inputs.

This property is formalized as *indistinguishability*: We consider a protocol in the real world to be *at least as secure* as the ideal execution, if no *distinguisher* $\mathcal{D}$ (in the context of the Universal Composability (UC) framework [40, 41] often called *environment* $\mathcal{Z}$) can determine whether it is observing honest parties executing a protocol, or if the parties perform the computation in the ideal world by letting the trusted party do all the work.

Since in the ideal world, honest parties only forward their input to the trusted party (which we call *functionality*), there are no protocol messages exchanged. As such, the two worlds could be easily distinguished by simply checking whether parties communicate with each other or not. Hence, the task of proving security comes down to providing a simulator who acts in the ideal world, and who creates *transcripts* (or reports the correct messages dynamically in the context of UC) of a real-world execution that are indistinguishable from the actual real world execution of the protocol.

However, the simulator—just like the adversary in the real world—only gets quite limited notifications from the functionality which (typically) do not contain the secret inputs, but only the output of the function. As such, it is the simulators task to provide a transcript *based only on the output* of the function (and, optionally, some additional leakage explicitly specified in the description of the trusted party) in such a way, that the distinguisher cannot determine better than by guessing randomly whether it is witnessing a genuine message exchange from a real execution or a transcript provided by a simulator who does not know the secret inputs.

If indistinguishability is formally proven, then the protocol is secure. It follows that *any* attack that can be made in the real world can also be executed in the ideal world against the trusted party.

Therefore, any security analysis needs to provide a simulator and a proof that the view is indeed indistinguishable (and can be provided by the simulator).

Although the security notions came from the 1980s, with major adjustments until the early 2000s, they are in essence still widely identical. The major differences from different instantiations are with respect to parameters such as, but not limited to, the round complexity of the protocol, the abilities of the adversary, the message length, the complexity of the individual tasks, or the assumption used in the proof to show indistinguishability.

In this thesis, we ask the question if the established security notions for MPC can be improved upon. In particular, we investigate three different settings:

Business Models and Legal Regulations. Despite its practical efficiency, MPC is barely used these days. One of the reasons for that is that perfect privacy excludes mandatory checks for real world scenarios (say, to detect money laundry when transferring money) or business models which require data analysis (say, for optimization of the own services).

Can we construct Secure Multi-Party Computation that provides anonymity while still complying with regulations and business models?

Stronger Commitments. In the quantum world, commitments suffice to construct MPC [57]. A perfectly secure commitment would yield a perfectly secure MPC protocol, yet perfectly secure commitments are impossible due to Mayers [108] and Lo and Chau [105].

Can we have commitments without hardware assumptions, where the security for one party holds unconditionally, and the security for the other party is stronger than just *the computational setting current literature resorts to?*

Undetectable MPC. MPC hides the input, but every party is aware of the computation—even if the computation is aborted and yields no result. Yet there are scenarios where even the computation itself should be hidden unless the result has been computed successfully.

Can N parties hide a computation in an innocent-looking conversation between $K > N$ parties, without revealing their identity even to each other?

We will elaborate on the individual questions in the following.

1.1. Modelling Business Models and Legal Regulations with MPC

As we have already discussed, MPC is an incredibly powerful cryptographic tool for which efficient protocols exist. At the same time, only very few projects exist that use MPC in the real world; among the best known examples is the dutch sugar beet auction [29], where the market clearing price is computed using MPC. Yet despite its benefits there are still only relatively few instances where MPC is used to solve a genuine problem in the real world.

One might wonder why that is, and frankly, there could be more than one reason. The protocols are still hard to comprehend for non-cryptographers, the hardware would have to be better than the one used for computations on plaintext data, causing additional costs, the protocol would require a lot more bandwidth, just to name a few. Yet one reason we aim to tackle in this part of the thesis is the inapplicability of general MPC to business models and legal requirements. This is largely due to the fact that cryptography aims to maximize privacy: The less data is leaked the better. In contrast, many businesses are either legally bound to analyze certain parts of the data, or their business requires proper analysis of genuine user data, for which perfect privacy opens the door to use counterfeit inputs.

Thus, despite the existence and practicability of MPC, these computations are performed directly on the cleartext data in reality. Such computations, however, have the problem that data can be misused; the operator responsible for performing the computation has the technical possibility to sell the data to other companies or to perform further analytics without notifying the user, and data centers that store customer data become a target for hackers. So what ways are there to solve this problem?

We propose a solution based on an abstract framework that enables operators to easily model the required amount of leakage into a function that is working on *private* user data without revealing it. So in a way, one could argue that this makes MPC weaker as we allow (albeit in a more explicit way) to model leakage into the computation, which results in less overall privacy in general compared to the near-perfect anonymity that is common. However, we believe that this increases the overall privacy, as it enables to use MPC in areas, where previously all computations were performed on plaintext data. Thereby, the operator can no longer *use* the user data for a computation without *interacting* with the owner of the data. This implicitly requires some form of agreement by the user for the data to be used in this computation, which can be seen as a way to technically enforce some requirements of the General Data Protection Regulation (GDPR).

So interestingly, we arguably *weaken* the security of MPC in an effort to *strengthen* the overall security, by providing an MPC definition to widen the scope where MPC can be applied.

1.2. An Instantiation of Stronger Commitments

While not directly related to MPC in the classic world (*i.e.*, the world without quantum computers), in our second contribution we investigate *commitments*. Commitments provide a quite strong building block, which is classically believed to be insufficient for constructing MPC. This changes, however, if we move into the quantum world; when both parties have a quantum computer then Crépeau [57] showed that commitments suffice to construct Oblivious Transfer (OT), from which MPC can be constructed [99, 90].

In proving security we separate between different adversarial strength. A protocol is *unconditionally* or *statistically* secure if the adversary has unlimited runtime and still is unable to break the security property. In contrast, if a protocol is *computationally* secure, breaking the security property is provably *at least as hard* as solving a complexity theoretic problem that is conjectured to be not efficiently solvable. So while the security property can be broken in practice, the task may take even top-tier computers several years, which provides a sufficient amount of security for many applications.

The protocol provided by Crépeau [57] is statistically secure, yet uses the commitments as *black-box* and assumes idealized, perfectly secure instantiations. Using OT as building block it follows from Kilian [99] and Ishai, Prabhakaran, and Sahai [90] that N-party MPC can be constructed. They, too, make black-box use of an idealized building block, and assuming with an idealized instantiation of that building block, the resulting MPC protocol is statistically secure.

So in the quantum world, if we could *somehow* construct statistically secure commitments, we would obtain statistically secure OT [57], hence statistically secure MPC [99, 90].

Such commitments could be provided using dedicated hardware (*e.g.* by using a heavily guarded safe where the committer writes the message to-be-committed on a piece of paper and stores it inside the safe), which would then have to be used somewhat excessively in order to get statistically secure MPC out of it.

Given this information, it is highly desirable to construct statistically secure commitments that work without special hardware. It is folklore knowledge that this is impossible in classical systems. Yet the classical impossibility result is *not* applicable in the quantum setting, meaning that in the early 1990s in particular, the cryptographic community was eager to investigate the possibility of perfectly secure commitments in the quantum setting. However, it was shown by Mayers [108] and Lo and Chau [105] that this too is impossible. This results in an asymmetry of security properties where usually, the security of one party is based on computational assumptions only. Yet the open question still stands how *strong* the weaker security property can get. This question was already investigated by Unruh [129], who provides a protocol that is unconditionally binding and everlasting hiding; that is, the adversary is computationally bounded during the protocol execution but drops all runtime restrictions after protocol termination. However, Unruh [129] again relied on additional hardware. We investigate the same setting, namely that of unconditional security for one party and everlasting security for the other, only that—except for the quantum computer—the participants do not require any additional hardware. We provide a positive answer to this question when we assume Quantum Decay (QD)—stating that quantum information cannot be stored indefinitely—and a Quantum Random Oracle Model (QROM) which models a quantumly accessible ideal hash function. The question remains whether the latter is indeed necessary; can there be an instantiation *without* the QROM? While we do not prove directly that the QROM is really necessary, we do provide evidence that this might be the case. In particular, we introduce a new type of attack called Obfuscated Measurement Attack (OMA) on the everlasting hiding property and show how it can be applied to our protocol when instantiating it in the standard model (*i.e.* without a QROM).

1.3. Undetectable Multi-Party Computation

We already pointed out that MPC is a strong tool that hides the input of each individual party beyond what can be learned from the jointly computed output. This means that it can be used in any scenario where a number of N parties agree to compute a given function, as long as the desire to compute that function does not automatically leak sensitive information. Yet, as was already stated by von Ahn, Hopper, and Langford [133], there are scenarios where the desire to even start a computation reveals too much, and where we quickly run into a bootstrapping problem. One of the examples brought fourth by the authors was the classical motivation for MPC, where Alice and Bob want to find out if they both fancy each other, but neither of them wants the respective other party to know they fancy them if the feelings are not mutual. In the classical literature, MPC is presented as the solution to this problem, as Alice and Bob can compute a logical AND on their secret binary inputs to obtain the correct output. In reality, however, this requires either of the two parties to convince the respective other party to start this computation, which already reveals that parties' interest in the other party; computing a logical AND in MPC when the own input is 0 is a waste of time, after all. So the question investigated by the authors was how two people can compute such sensitive functions without revealing their intent directly. At first glance, it might seem like a viable solution to use a technique similar to *paroles* used in spy movies to recognize an ally: If Alice wants to find out if Bob is interested in her, she starts a conversation by using a recognized sentence that sounds innocent (*e.g.* asking how much the bread costs in the bakery on the other side of the street) but that was previously agreed upon to start the protocol, and if Bob answers in a specific way (*e.g.* recommending a special type of cake from that bakery) then it means that the feelings are mutual. However, this does *not* solve the problem sufficiently well. Either the parole is very generic, in which case the false positive rate is too high because Alice might really just want to know how much the bread costs, or the parole is very specific, in which case Bob already knows that Alice is interested without even having done anything. Likewise, any bystander who is aware of the protocol immediately knows what is going on.

So this naïve method does not work. Instead, the authors drew inspiration from their previous work in the field of steganography and provided a two-party protocol that computes the function and provides output to both parties, but only if both parties actively follow the protocol until the end *and* if the output is *favorable*; for the example, favorable would mean that parties only receive output if they are both interested in each other. Yet if parties do not get any output, it is indistinguishable whether the other party simply did not follow any protocol and just acted normally or if the output was unfavorable.

This solved the problem for two parties, but what if there are more than two parties who want to covertly compute a function? A first step in this direction was taken by Chandran et al. [48], yet at the cost of requiring *all* parties to follow the protocol; if an additional non-participant is present and just acts normally, the entire computation fails. So the question remains: If we know in advance that there are N parties present who want to compute the function, but $K > N$ parties are participating in the conversation, is it possible to compute the function *only* on the inputs of the N actual participants? And is it possible that despite obtaining output, the participants do not know which other parties were actively participating and which were just present and acted naturally? Performing CMPC with all $\binom{K}{N}$ subsets would solve the former, but upon receiving output, each participant knows exactly which other parties were participating and which were not. Furthermore, this method would drastically limit the number of present parties.

We propose a different solution that extends the notion to K parties, of which only N parties are computing a function based *only* on the inputs of the N active participants. We call this notion Undetectable Multi-Party Computation (UMPC) and investigate whether or not this works. UMPC itself can be reduced to a novel primitive called Anonymous Transfer (AT), which allows a sender to *anonymously* publish a message either to a fixed receiver or to the general public. We show that *if* this primitive exists with overwhelming *correctness* and *anonymity*, then we can instantiate UMPC. So the question on the feasibility of UMPC comes down to the possibility of AT. Yet, we show that no protocol can achieve overwhelming correctness and anonymity even against a semi-honest adversary with asymptotic runtime. However, with relaxed requirements, a (non-trivial) protocol under non-standard assumptions exists. We analyze this protocol and prove its security in the fine-grained setting.

2. Preliminaries

2.1. Notation

We write bit strings using lowercase roman letters, *e.g.* x or y. For $i \in \mathbb{N}$ we denote by $x[i]$ the i-th bit of x. We write vectors as $\vec{x}$ using lowercase roman letters and matrices as $\vec{X}$ using uppercase roman letters. Similar to bitstrings, for a given vector $\vec{x}$ and some $i \in \mathbb{N}$ we denote by $\vec{x}[i]$ the i-th element of $\vec{x}$. Given two bit vectors $\vec{x}$ and $\vec{y}$ of the same length, we denote by $\vec{x} \oplus \vec{y}$ the bit-wise XOR operation between $\vec{x}$ and $\vec{y}$.

For a natural number n we denote by $[n]$ the set $\{1, \ldots, n\}$ of all natural numbers from 1 to n and for two natural numbers n_0 and $n_1 \in \mathbb{N}$ for $n_0 < n_1$ we write $[n_0, n_1]$ for the set $\{n_0, n_0 + 1, \ldots, n_1\}$ of all natural numbers between n_0 and n_1. We denote the security parameter by κ. Even without writing it every time, every algorithm and protocol obtains 1^κ as input, that is, a unary encoding of the security parameter.

We write functions in roman letters, *e.g.* f. By $\mathrm{poly}(\kappa)$ we denote the set of functions that are *polynomial* in κ, that is, functions for which a constant c and a vector $\vec{a} \in \mathbb{R}^c$ exists such that the function can be represented as $\sum_{i=0}^{c} \vec{a}[i]\kappa^i$. By $\mathrm{negl}(\kappa)$ we denote the set of *negligible* functions, that is, functions that asymptotically shrink faster than any inverse function in $\mathrm{poly}(\kappa)$. More formally, a function f is negligible in the security parameter (written $\mathrm{f} \in \mathrm{negl}(\kappa)$) if the following condition holds:

$$\forall c \in \mathbb{N}\ \exists n_c \in \mathbb{N}\ \forall n > n_c \colon |\mathrm{f}(n)| \leq \frac{1}{n^c} \tag{2.1}$$

Likewise, a function f is *overwhelming* in the security parameter (written $\mathrm{f} \in \mathrm{owhl}(\kappa)$) if $(1 - \mathrm{f}) \in \mathrm{negl}(\kappa)$.

Let Ω be the set of all outcomes of a probability experiment. We denote events by capital roman letters such as $A, B \subseteq \Omega$. The complementary event of A is denoted as $\bar{A} = \Omega \setminus A$. The conditional probability of A happening conditioned on B is denoted $\Pr\left[\ A \ \middle|\ B\ \right]$.

We write general participants of protocols in sans serif fonts, *e.g.* P for an arbitrary party or S for a sender. Each party P has its own random tape which we denote as T_{P}. We use N to denote the number of parties in any execution. Unless explicitly mentioned these are Interactive Turing Machines (ITMs) which run in Probabilistic Polyonmial time (PPT). The only party we write in a different font are parties relevant for the security model. These are written in calligraphic font, *e.g.* $\mathcal{A}$ for an adversary or $\mathcal{C}$ for a challenger.

Schemes are written in small caps, *e.g.* $\textsc{Com}$ for a commitment scheme or $\textsc{Sig}$ for a signature scheme. The respective methods are written in roman letters. When using a method, we generally write the scheme before the method, *e.g.* $\textsc{Com}.\mathrm{Com}$ for the commitment method Com as part of the commitment scheme $\textsc{Com}$.

Let $\Sigma \in \{0,1\}^n$ be an n-bit string. We denote by $\overline{\Sigma}$ the complementary bitstring where each bit is flipped. For example, if $\Sigma = (10110)$ then $\overline{\Sigma} = (01001)$.

We write sets by italic fonts, *e.g.* dom to denote a (finite) domain. We write $x \sim \mathit{dom}$ to denote uniform sampling from a given set.

If stmt is a statement, we denote by $[\![\mathit{stmt}]\!]$ the boolean outcome of that statement: $[\![\mathit{stmt}]\!] = 1 \iff \mathit{stmt} = \mathit{true}$. For example, $[\![5 > 3]\!]$ is 1, and $[\![5 < 3]\!]$ is 0.

For probability distributions p and q we write $p^{\otimes t}$ as the distribution arising from taking t samples from p, and $p \circ q$ as the distribution obtained by sampling one time from p and one time from q. By $\|p\|_1$ we mean the L_1 norm of p, that is, the sum of the absolute values of all possible outcomes.

For assigning values we use the following notation: If the assignment is *deterministic*, we use the symbol $:=$, *e.g.* for $x := 5$. If the assignment is *probabilistic* (*i.e.* determined by an algorithm that uses random coins) we write $\leftarrow$, *e.g.* $\mathtt{com}_x \leftarrow \textsc{Com}.\mathrm{Com}(x)$. If the algorithm itself involves sampling, we use the notation $\overset{\$}{\leftarrow}$, *e.g.* $x \overset{\$}{\leftarrow} \{1, \ldots, n\}$.

2.2. Commitments

In the following, we provide definitions for commitment schemes. We focus on definitions for bit commitment schemes for $\mathit{msg} \in \{0,1\}$ as those can be expanded canonically to string-commitments for $\mathit{msg} \in \{0,1\}^n$. Informally, commitment schemes provide a two-phase technique that enables *fixing* a bit without *revealing* it in a first phase (*commitment phase*), and revealing *the same* bit in a second phase (*unveil phase*), while guaranteeing that fixing the bit does not leak information on the bit (hiding) and that the bit remains fixed and cannot be changed in phase two (binding).

2.2.1. Definition

We provide a general definition of a commitment scheme following [76]:

Definition 2.2.1 (Commitment Scheme). *A commitment scheme* $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$ *over a single bit is defined as a tuple of two methods* $(\mathrm{Com}, \mathrm{Unv})$ *executed between a committer* C *and a receiver* R*. The methods are defined as follows:*

$\mathrm{Com}(1^\kappa, b)$ *takes as input a unary encoding of the security parameter* 1^κ *and a bit* b *and returns two bitstrings* $(\mathtt{com}_b, \mathtt{unv}_b)$ *to* C *and the bitstring* $(\mathtt{com}_b)$ *to* R.

$\mathrm{Unv}(1^\kappa, \mathtt{com}_b, \mathtt{unv}_b, b)$ *takes as input a unary encoding of the security parameter* 1^κ*, the commitment string* $\mathtt{com}_b$*, the unveil information* $\mathtt{unv}_b$ *and the bit* b*, and returns* 1 *to all parties if the unveil information* $\mathtt{unv}_b$ *opens the commitment* $\mathtt{com}_b$ *to* b *and* 0 *otherwise.*

2.2.2. Correctness

A commitment protocol is *correct* if an honestly generated commitment gets rejected only with negligible probability, where the probability is over the random coins used by $\textsc{Com}.\mathrm{Com}$, that is, if for all $\kappa \in \mathbb{N}$ and all $b \in \{0,1\}$, the following probability is *overwhelming* in κ:

$$\Pr\left[\textsc{Com}.\mathrm{Unv}(\mathtt{com}_b, \mathtt{unv}_b, b) = 1 \;\middle|\; \begin{array}{l} b \sim \{0,1\}, \\ (\mathtt{com}_b, \mathtt{unv}_b) \leftarrow \\ \quad \textsc{Com}.\mathrm{Com}(1^\kappa, b) \end{array} \right] \tag{2.2}$$

2.2.3. Hiding Commitment Schemes

We call a commitment scheme *computationally* hiding if the commitment message cannot be extracted by any PPT adversary $\mathcal{A}$. This means:

Definition 2.2.2 (Computationally Hiding Bit Commitment Scheme)**.** *A bit commitment scheme* $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$ *is* computationally hiding *if for any PPT adversary* $\mathcal{A}$ *and any security parameter* $\kappa \in \mathbb{N}$*, the following probability is negligible in* κ*:*

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{l} b \overset{\$}{\leftarrow} \{0,1\}, \\ \tau_b \leftarrow \textsc{Com}.\mathrm{Com}(1^\kappa, b), \\ b' \leftarrow \mathcal{A}(1^\kappa, \tau_b) \end{array} \right] - 1/2 \right| \tag{2.3}$$

where τ_b *is the* transcript *of an honest execution for a commitment on* b*. In this case* τ_b *corresponds to the entire view of a receiver during the commitment-phase.*

Note that this property implies that a transcript originating from an honestly created commitment on b cannot be distinguished from a transcript of a commitment on $\overline{b}$ by any PPT adversary. This property will be used later in this thesis to prove the hiding property of commitment schemes.

A commitment scheme is *unconditionally* hiding if the commitment message contains *absolutely* no information on the committed bit. That is, we consider a bit-commitment scheme as hiding if the distribution of *zero*-commitments is indistinguishable to the distribution of *one*-commitments and cannot be distinguished by *any* algorithm with noticeable advantage over guessing. More formally [76]:

Definition 2.2.3 (Unconditionally Hiding Bit Commitment Scheme). *A bit commitment scheme* $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$ *is* unconditionally hiding *if the following condition holds:*

$$\{\textsc{Com}.\mathrm{Com}(0)\} \approx \{\textsc{Com}.\mathrm{Com}(1)\} \tag{2.4}$$

where $\approx$ *denotes* information-theoretical *indistinguishability.*

2.2.4. Binding Commitment Schemes

The binding property implies that a commitment on some bit b cannot be efficiently opened to a commitment on $\overline{b}$. More formally [76]:

Definition 2.2.4 (Computationally Binding Bit Commitment Schemes). *A bit commitment scheme* $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$ *is computationally binding if for every PPT-adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and security parameter* κ *it holds that the following probability is negligible in the security parameter* κ*:*

$$\left| \Pr\left[\textsc{Com}.\mathrm{Unv}(\mathtt{com}_b, \mathtt{unv}_b, b) = 1 \;\middle|\; \begin{array}{l} (st, \mathtt{com}) \leftarrow \mathcal{A}(1^\kappa), \\ b \xleftarrow{\$} \{0,1\}, \\ \mathtt{unv} \leftarrow \mathcal{A}_2(\kappa, st) \end{array} \right] - \frac{1}{2} \right| \tag{2.5}$$

If $\mathcal{A}_1$ and $\mathcal{A}_2$ have no runtime restrictions, then the scheme is called *unconditionally binding*.

Definition 2.2.5 (Unconditionally Binding Bit Commitment Schemes). *A bit commitment scheme* $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$ *is unconditionally binding if for every* unbounded*-adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the following probability is negligible in the security parameter* κ*:*

$$\left| \Pr\left[\textsc{Com}.\mathrm{Unv}(\mathtt{com}, \mathtt{unv}_b, b) = 1 \;\middle|\; \begin{array}{l} (st, \mathtt{com}) \leftarrow \mathcal{A}(1^\kappa), \\ b \xleftarrow{\$} \{0,1\}, \\ \mathtt{unv}_{msg} \leftarrow \mathcal{A}(st, b) \end{array} \right] - \frac{1}{2} \right| \tag{2.6}$$

2.3. Signature Schemes

Signature schemes aim to provide authenticity of messages.

2.3.1. Definition

We provide a general definition of a signature scheme as it was defined by [75]:

Definition 2.3.1 (Signature Scheme). *A signature scheme* $\mathcal{S}\textsc{ig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *with message length* n *is defined as a tuple of three methods* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *executed between a sender* S *and a receiver* R*. The methods are defined as follows:*

$\mathsf{KeyGen}(1^\kappa)$ *takes as input a unary encoding of the security parameter* 1^κ *and returns two bitstrings* $(\mathsf{k}, \mathsf{vk})$ *to* S *and* vk *to* R*.*

$\mathsf{Sign}_{(\mathsf{k})}(1^\kappa, msg)$ *takes as input a unary encoding of the security parameter* 1^κ*, a message* $msg \in \{0,1\}^n$*, and a signing key* k*, and returns a signature* σ_{msg} *to* S *and* R*.*

$\mathsf{Vfy}(1^\kappa, \sigma_{msg}, \mathsf{vk}, msg)$ *takes as input a unary encoding of the security parameter* 1^κ*, a signature* σ_{msg}*, a verification key* vk*, and a message* $msg \in \{0,1\}^n$*, and returns* 1 *if the signature* σ_{msg} *successfully verifies the message* msg *under the key* vk*, and* 0 *otherwise.*

2.3.2. Correctness

Intuitively, we say that a signature scheme is correct if a signature that was created by following the $\mathsf{Sign}()$ method is accepted by the Vfy method. More formally [75]:

Definition 2.3.2 (Correctness for Signature Schemes). *Let* $\mathcal{S}\textsc{ig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a signature scheme.* $\mathcal{S}\textsc{ig}$ *is correct, if the following probability is overwhelming in the security parameter* κ*:*

$$\Pr\left[\mathcal{S}\textsc{ig}.\mathsf{Vfy}(\sigma_{msg}, \mathsf{vk}, msg) = 1 \;\middle|\; \begin{array}{l} (\mathsf{k}, \mathsf{vk}) \leftarrow \mathcal{S}\textsc{ig}.\mathsf{KeyGen}(1^\kappa), \\ msg \xleftarrow{\$} \{0,1\}^n, \\ \sigma_{msg} \leftarrow \mathcal{S}\textsc{ig}.\mathsf{Sign}_{(\mathsf{k})}(msg) \end{array} \right] \tag{2.7}$$

where the probability is taken over the randomness involved in KeyGen *and* Sign *and the implicit cointoss to create the message.*

2.3.3. Existential Unforgeability under Chosen Message Attacks

We follow the outline for describing Existential Unforgeability under Chosen Message Attacks (EUF-CMA) security set by [75]:

Definition 2.3.3 (EUF-CMA Secure Signature Scheme). *A signature scheme* $\mathcal{S}\textsc{ig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *is* EUF-CMA secure *if for every PPT-adversary* $\mathcal{A}$ *with access to a signing oracle* $O_{\mathsf{Sign}_{(\mathsf{k})}(\cdot)}$ *the following probability is negligible in the security parameter* κ*:*

$$\Pr\left[\mathcal{S}\textsc{ig}.\mathsf{Vfy}(\sigma^*, \mathsf{vk}, msg^*) = 1 \;\middle|\; \begin{array}{l} (\mathsf{k}, \mathsf{vk}) \leftarrow \mathcal{S}\textsc{ig}.\mathsf{KeyGen}(1^\kappa), \\ (\sigma^*, msg^*) \leftarrow \mathcal{A}^{O_{\mathsf{Sign}_{(\mathsf{k})}(\cdot)}}(\mathsf{vk}, 1^\kappa) \end{array} \right] \tag{2.8}$$

where msg^* *was never sent from* $\mathcal{A}$ *to* $O_{\mathsf{Sign}_{(\mathsf{k})}(\cdot)}$*.*

If we additionally require the *signature* to be unique, we get a stronger notion called Strong Existential Unforgeability under Chosen Message Attacks (sEUF-CMA) security:

Definition 2.3.4 (sEUF-CMA Secure Signature Scheme). *A signature scheme* $\textsc{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *is* sEUF-CMA secure *if for every PPT-adversary* $\mathcal{A}$ *with access to a signing oracle* $O_{\mathsf{Sign}_{(\mathsf{k})}}(\cdot)$ *the following probability is negligible in the security parameter* κ*:*

$$\Pr\left[\ \textsc{Sig}.\mathsf{Vfy}(\sigma^*, \mathsf{vk}, msg^*) = 1 \ \middle|\ \begin{array}{l}(\mathsf{k}, \mathsf{vk}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa),\\ (\sigma^*, msg^*) \leftarrow \mathcal{A}^{O_{\mathsf{Sign}(\mathsf{k})}(\cdot)}(\mathsf{vk}, 1^\kappa)\end{array}\right] \tag{2.9}$$

where msg^* *was never sent from* $\mathcal{A}$ *to* $O_{\mathsf{Sign}_{(\mathsf{k})}}(\cdot)$ *and* σ^* *was never returned from* $O_{\mathsf{Sign}_{(\mathsf{k})}}(\cdot)$ *to* $\mathcal{A}$*.*

2.4. Encryption Schemes

Even though the general definitions for symmetric and asymmetric encryption schemes are quite similar, we provide individual definitions for both of them.

2.4.1. Definition

We follow the definitions from Goldreich [75] for symmetric and asymmetric encryption schemes:

Definition 2.4.1 (Symmetric Encryption Scheme). *A symmetric encryption scheme* $\textsc{Ske} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is defined as a tuple of three methods* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *executed between a sender* S *and a receiver* R*. The methods are defined as follows:*

$\mathsf{KeyGen}(1^\kappa)$ *takes as input a unary encoding of the security parameter* 1^κ *and returns a shared key* sk *to* S *and* R*.*

$\mathsf{Enc}(1^\kappa, \mathsf{sk}, msg)$ *takes as input a unary encoding of the security parameter* 1^κ*, a symmetric key* sk*, and a message* $msg \in n$*, and returns a ciphertext* ct *to* S*.*

$\mathsf{Dec}(1^\kappa, \mathsf{sk}, ct)$ *takes as input a unary encoding of the security parameter* 1^κ*, a symmetric key* sk*, and a cipher text* ct*, and returns a plaintext message* msg *to* R*.*

Definition 2.4.2 (Asymmetric Encryption Scheme). *An asymmetric (or* public-key*) encryption scheme* $\textsc{Pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is defined as a tuple of three methods* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *executed between a sender* S *and a receiver* R*. The methods are defined as follows:*

$\mathsf{KeyGen}(1^\kappa)$ *takes as input a unary encoding of the security parameter* 1^κ *and returns two bitstrings* $(\mathsf{sk}, \mathsf{pk})$ *to* S *and* pk *to* R*.*

$\mathsf{Enc}(1^\kappa, \mathsf{pk}, msg)$ *takes as input a unary encoding of the security parameter* 1^κ*, a public key* pk*, and a message* $msg \in n$*, and returns a ciphertext* ct *to* S*.*

$\mathsf{Dec}(1^\kappa, \mathsf{sk}, ct)$ *takes as input a unary encoding of the security parameter* 1^κ*, a secret key* sk*, and a cipher text* ct*, and returns a plaintext message* msg *to* R*.*

2.4.2. Correctness

We define correctness for asymmetric schemes only, but stress that the definition for symmetric schemes is analogous. In essence, both require that the correct message can be extracted from the ciphertext, meaning that encrypting and then decrypting a message results in the same message. More formally, correctness states the following [75]:

Definition 2.4.3 (Correctness for Asymmetric Encryption Schemes)**.** *Let* $\mathcal{P}\textsc{ke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an asymmetric encryption scheme.* $\mathcal{P}\textsc{ke}$ is correct *if the following probability is* overwhelming *in the security parameter* κ*:*

$$\Pr\left[\textsc{Pke}.\mathsf{Dec}(\mathsf{sk}, ct) = msg \;\middle|\; \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa), \\ msg \overset{\$}{\leftarrow} \{0,1\}^n, \\ ct \leftarrow \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}, msg) \end{array} \right] \tag{2.10}$$

2.4.3. Indistinguishability under Chosen Plaintext Attacks

Security for both symmetric and asymmetric encryption schemes follows the indistinguishability paradigm. The Indistinguishability under Chosen Plaintext Attacks (IND-CPA) security of symmetric schemes is defined as follows [75]:

Definition 2.4.4 (IND-CPA Secure Symmetric Encryption)**.** *A symmetric encryption scheme* $\textsc{Ske} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA-secure *if for every PPT-adversary* $\mathcal{A}$ *and every* $x, y \in \{0,1\}^{\mathrm{poly}(\kappa)}$ *such that* $|x| = |y|$*, the following probability is negligible in the security parameter* κ*:*

$$\begin{aligned} &|\Pr[\mathcal{A}(\textsc{Ske}.\mathsf{Enc}_{\mathsf{sk}}(x)) = 1 | \mathsf{sk} \leftarrow \textsc{Ske}.\mathsf{KeyGen}(1^\kappa)] \\ &- \Pr[\mathcal{A}(\textsc{Ske}.\mathsf{Enc}_{\mathsf{sk}}(y)) = 1 | \mathsf{sk} \leftarrow \textsc{Ske}.\mathsf{KeyGen}(1^\kappa)]| \end{aligned} \tag{2.11}$$

Definition 2.4.5 (IND-CPA Secure Asymmetric Encryption)**.** *An asymmetric encryption scheme* $\textsc{Pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA-secure *if for every PPT-adversary* $\mathcal{A}$ *and every* $x, y \in \{0,1\}^{\mathrm{poly}(\kappa)}$ *such that* $|x| = |y|$*, the following probability is negligible in the security parameter* κ*:*

$$\begin{aligned} &|\Pr[\mathcal{A}(\mathsf{pk}, \textsc{Pke}.\mathsf{Enc}_{\mathsf{pk}}(x)) = 1 | (\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Pke}.\mathsf{KeyGen}(1^\kappa)] \\ &- \Pr[\mathcal{A}(\mathsf{pk}, \textsc{Pke}.\mathsf{Enc}_{\mathsf{pk}}(y)) = 1 | (\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Pke}.\mathsf{KeyGen}(1^\kappa)]| \end{aligned} \tag{2.12}$$

Part I.

MPC for Business Models and Legal Regulations

3. Introduction

Secure Multi-Party Computation (MPC) allows a set of N parties to securely compute a function f on private inputs $(x_1, \ldots, x_N)$ in such a way, that all participants only learn the correct value of $\mathsf{f}(x_1, \ldots, x_N)$ without leaking any further information about their input to the other parties. With recent advancements in MPC protocols can be used with a relatively slow overhead, in particular when assuming passive adversaries [61, 94] but also with security against participants who actively deviate from the protocol [90, 58]. At the same time, regulations like the GDPR of the EU incentivize data economy. So it would be natural to assume that MPC is now used in a variety of user-centric settings such as pay-as-you-drive insurance, transportation systems, web search or mobile payments. However, it is still very rare to find MPC in practical applications.

There are many reasons for that, but despite additional costs for development, deployment and maintenance and the inherent complexity for non-cryptographers one reason for the rare usage of MPC in such scenarios is a discrepancy between anonymity requirements appreciated by the cryptographic community and the real-world requirements coming from both business models and legal regulations, which do not work in settings that provide perfect anonymity.

Let us consider loyalty programs for grocery shopping. There, a company rewards loyalty of customers; if customers buy in cooperating stores sufficiently often they get rewards. Of course, this business model is not funded by stores that want to convince people to buy there more often, a huge part of the funding is based on analyzing the data of what people buy in what quantities and finding correlations between the purchase behavior for different products.

So while it would be possible to model such a loyalty program in MPC, it would come at the cost of being able to analyze the purchase behavior of customers; funding would have to be based *only* on the fee that participating stores pay to provide customers with a motivation to continue shopping there.

So the research question we investigate in this part of this theses is as follows:

Can we provide a generic framework for MPC that enables efficient modeling of leakage required for business models and legal requirements?

3.1. Our Contribution

We provide a positive answer to the research question stated above by introducing an abstract privacy- and authenticity-preserving bookkeeping and analytics framework that works on user data. That is, we call **users** the parties in our framework who can collect and manipulate data and provide that data anonymously for analytical computations. Throughout this part of this thesis we will assume that the users have only limited computational power when compared to the other parties, as in a real instantiation a user would probably use a smartphone to interact. We further assume that the framework is managed and maintained by a central **operator**. Users can only manipulate their collected data by interacting with the operator. At the same time, the operator is unable to link two interactions to the same user. That is, the user is always hidden among all existing users: In every interaction, all users are equally likely to be the participant. On the other hand, users always have the guarantee that they are communicating with the operator. So in essence, we require that one dedicated party (the operator in our case) sends only authenticated messages, whereas the other party (in this case the user) is only guaranteed to communicate with a party from a given set of parties (here the set of registered users). We show later how this can be constructed using the Tor [62] network—yet we conjecture that it also works with general mix-nets [50]—alongside with a Public-Key Infrastructure (PKI) and an identification scheme.

The framework we construct consists of several individual protocols which enable users to provide their collected data for *computations*. Depending on the scenario, the functions to-be-computed have to be kept private—be it because they involve trade secrets that should not be revealed to the user, or because they only work when kept private, as is the case with fraud detection schemes[1]. We thus provide a mechanism that only leaks a very general structure of the computed function to the user, without leaking the function details itself. This, however, enables a trivial attack on the users anonymity; an operator who can use any function it wants can simply compute the identity function that unveils the entire user data to the operator, which then makes the entire overhead of using MPC for the computation itself pointless and breaks anonymity.

We thus require a mechanism that disables such attacks and that forces the operator to only use *privacy-friendly* functions. The method we use to achieve this requires an additional party we call the **Trusted Signing Authority (TSA)**. Before being able to use *any* functions for computation the operator needs the TSA to *verify* them. If the function can be used then the TSA *signs* the function; it is then checked as mutually visible part of the MPC that the signature verifies the function under the verification key of the TSA.

As we specified earlier we assume the user to interact using a standard computer or even a smartphone with relatively weak hardware when compared to the operator, which we expect to run on much stronger servers. Unfortunately, this restriction also means that

[1] Once a fraud detection scheme that detects money laundry or misuse becomes public knowledge, adversaries can perform the computations locally and search for a set of inputs where the highest possible amount of cash is being laundered without raising suspicion.

many functions that require a lot of hardware even when computed directly (*i.e.*, without using MPC) cannot be efficiently computed on a smartphone. We thus introduce a new class of parties we refer to as **helpers**. Helpers provide the computational power of a server and perform computations on behalf of a user, without learning any of the user data and without being able to manipulate the results unnoticed. Furthermore, helpers coordinate computations where more than one user is involved.

We model, analyze and prove the security of our protocols in the Universal Composability (UC) framework [40, 41], which ensures that all security properties hold *regardless* which protocols are executed in parallel. Given that most protocols these days are executed on computers or smartphones, where the operating system is responsible for scheduling, we consider this strong type of composition to be necessary.

Despite the strong guarantees from the UC framework we aimed at practical efficiency. While not part of this thesis directly, the protocols were implemented in the conference version of our paper [70] and executed using only *smartphones* as user devices.

Also part of the conference version but not directly part of this dissertation are two *instantiations* that display how the tasks of *targeted advertisement* for grocery stores and *fraud detection* for payment systems can be implemented in this framework.

3.2. Related Work

While our work is the first that provides an abstract privacy-preserving bookkeeping and analytics building block that can be instantiated with different user-centric applications, some parts of our contribution have been considered before by other papers.

Privacy-Preserving Point Collection. If we omit the analytical part of our framework and only focus on *collecting* points we use similar techniques as the Black-Box Accumulators (BBA+) by Jager and Rupp [92] and Hartung et al. [85]. Using those it is possible to collect and redeem points, but not to provide the points for analytical computations.

A slightly different approach was taken by Blömer et al. [27] who introduced updatable anonymous credentials. Those provide an update function that a user can apply to the attributes of the credentials in order to manipulate them in a weaker sense than we do: The update function updates the value to a new value that is provided and known only by the user, without providing the possibility to manipulate the attributes based on their previous contents.

Both the works of Jager and Rupp [92], Hartung et al. [85], and Blömer et al. [27] only consider *collecting* points and do not provide any methods towards *using* points for computations. Furthermore, both use weaker security models which does not provide the strong composability guarantees we consider essential for any such scheme. In contrast, our method provides UC security.

Analytics on Authenticated Data. The scheme from Kolesnikov, Kumaresan, and Shikfa [101] provides *input authentication* by ensuring that outputs of one computation are correctly transferred into the next computation and cannot be changed in between. Their method identifies users in every interaction and only considers malicious users and assume the operator to be at most passively corrupted. In contrast, our method provides unlinkability and works even if the operator is actively deviating from the protocol.

There are a lot of methods that provide MPC for dedicated structures without considering many computations on similar data and where parties can freely choose their inputs. These can be grouped into protocols that work for special *functions* (such as neural networks or logistic regression) only, such as the work from Asharov et al. [16], Cetin et al. [46], and Juvekar, Vaikuntanathan, and Chandrakasan [94], or propose techniques directly for solving a given analytical problem [141, 121, 138, 64].

Applications. Our first application provides targeted advertisements for loyalty programs. Before our work, targeted advertisement has only been analyzed in the online setting to provide a privacy-preserving alternative to services such as Google Analytics, where a users device should only display advertisements based on its interests without revealing the user profile to the advertisement network. Those are mostly informal ideas, yet Backes et al. [17] provide a formal model and proof. In contrast, we consider loyalty programs where advertisements are based on the users purchases in actual stores.

Those methods differ between client-based ad selection, where the users device decides which advertisements to display [127, 17, 82], and proxy-based selection [84] where the system requires a Trusted Third Party to deliver the advertisements. The former means that the algorithm needs to be public, which we consider unrealistic for commercial advertisement companies, and that it is harder to keep track which advertisements were delivered, thus making payment from the advertising companies much harder. The latter only works as long as the trusted party is both *available* and *trusted* both by the users and the advertisement network.

Our second application provides a digital payment scenario that includes necessary detection mechanisms for fraud detection and money laundry. This is a relatively new topic with barely any previous work to relate with. The only work we are aware of that is related to ours is that of Canillas et al. [45]. The authors evaluate the performance of classifiers based on Fully Homomorphic Encryption that classifies transaction data as fraudulent or not. The method they use only assumes *some* inputs and does not provide any guarantees on whether these reflect the actual transactions.

4. Preliminaries

4.1. Notation

We write by $msg \in \{0,1\}^n$ a message (or an arbitrary bitstring) of length n. A commitment on msg is written $\mathtt{com}_{msg}$, the unveil information as $\mathtt{unv}_{msg}$ (see Section 4.2). When there are old *and* new commitments on the same values, we mark the old commitment with an asterisk, *i.e.* $\mathtt{com}_{msg^*}$. We denote by x the (unauthenticated) inputs, and by y the (unauthenticated) outputs.

For ZK (see Section 4.4) we write by Λ the language. In the figures showing a language, we use gray color to denote the witness.

4.2. Commitments

In addition to the properties of commitments we defined in Section 2.2 we state here some additional definitions for commitments.

4.2.1. Additively Homomorphic Commitment Schemes

A bit commitment scheme is *additively homomorphic* if commitments expose a desired form of malleability such that arithmetic operations on the commitment yield a desired effect on the underlying values.

Definition 4.2.1 (Additively Homomorphic Commitment Scheme). *A bit commitment scheme* $\textsc{Com} = (\mathrm{Setup}, \mathrm{Com}, \mathrm{Unv}, \mathrm{CAdd}, \mathrm{UAdd})$ *is* additively homomorphic *if the following probability is overwhelming in the security parameter* κ*:*

$$\Pr\left[\begin{array}{l|l} \textsc{Com}.\mathrm{Unv}(\mathtt{com}_{b\oplus b'}, & (b,b')\xleftarrow{\$}\{0,1\}, \\ \qquad\qquad \mathtt{unv}_{b\oplus b'}, b\oplus b') = 1 & pp_{\mathbb{G}} \leftarrow \mathrm{Setup}, \\ & (\mathtt{com}_b, \mathtt{unv}_b) \leftarrow \textsc{Com}.\mathrm{Com}_{pp_{\mathbb{G}}}(b), \\ & (\mathtt{com}_{b'}, \mathtt{unv}_{b'}) \leftarrow \textsc{Com}.\mathrm{Com}_{pp_{\mathbb{G}}}(b'), \\ & \mathtt{com}_{b\oplus b'} \leftarrow \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_b, \mathtt{com}_{b'}), \\ & \mathtt{unv}_{b\oplus b'} \leftarrow \textsc{Com}.\mathrm{UAdd}(\mathtt{unv}_b, \mathtt{unv}_{b'}) \end{array}\right] \tag{4.1}$$

4.2.2. Structure Preserving Commitment Schemes

If a commitment scheme is defined over some pairing group then an important property for enabling ZK proofs is that it preserves the structure. This means [2]

Definition 4.2.2 (Structure-Preserving Commitment Scheme)**.** *A commitment scheme* $\textsc{Com} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Unv})$ *is called structure-preserving with respect to a bilinear group generator* $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ *if the following conditions are all satisfied.*

1) *Common parameter* $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ *consists of a group description generated by* g *and constants* $a_{ij} \in \mathbb{Z}_o$.
2) *Commitment and unveil messages consist of group elements in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.
3) *Opening algorithm* Unv *consists only of evaluating membership in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *and relations described by pairing product equations.*

4.3. Signature Schemes

Again we extend the definitions given in Section 2.3 and provide some additional properties a signature scheme can have.

4.3.1. Structure Preserving Signature Schemes

The definition of structure preserving signature schemes is similar to that for commitments. We again use the definition from Abe et al. [2]:

Definition 4.3.1 (Structure-Preserving Signature Scheme)**.** *A digital signature scheme* $\textsc{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *is called structure-preserving with respect to a bilinear group generator* $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ *if the following conditions are all satisfied.*

1) *The group is uniquely determined by a group description* $pp_{\mathbb{G}}$ *generated by* g *and constants* $a_{ij} \in \mathbb{Z}_o$.
2) *Verification key* vk *consists of group elements in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *other than* $pp_{\mathbb{G}}$.
3) *Messages and signatures consist of group elements in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.
4) *Verification algorithm* Vfy *consists only of evaluating membership in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *and relations described by pairing product equations.*

4.4. Zero Knowledge Schemes

Let Rel be a witness relation for some NP-language $\Lambda = \{\mathit{stmt} | \exists \mathit{wit} \colon (\mathit{stmt}, \mathit{wit}) \in \mathit{Rel}\}$. A Zero Knowledge (ZK) scheme contains of two parties, a *Sender* S (often referred to as *prover*) and a *Receiver* R (which is often called *verifier* in the literature). The prover tries to convince the verifier that for a given statement stmt it holds that $\mathit{stmt} \in \Lambda$ without leaking any other information. If this requires only a single message sent from S to R we call the scheme Non-Interactive Zero Knowledge.

For efficiency reasons we work with *group-based* constructions, which generally follow the following definition:

Definition 4.4.1 (Group-Based NIZK Schemes). *A Zero Knowledge scheme* $\mathcal{Z}\kappa = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Proof}, \mathsf{Verify})$ *is* group*-based if each verifiable relation contains* triplets $(\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}, \mathit{stmt}, \mathit{wit})$ *for a group parameter* $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$*. Further, the algorithms behave as follows:*

Gen *takes as input a security parameter* 1^κ *and outputs public parameters* $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$.

Setup *takes as input* $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ *and outputs a (public) Common Reference String* crs.

Proof *takes as input the Common Reference String* crs*, the group parameters* $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$*, a statement* stmt *and a witness* wit *such that* $(\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}, \mathit{stmt}, \mathit{wit}) \in \mathit{Rel}$*, and outputs a proof* π.

Verify *takes as input the Common Reference String* crs*, the group parameter* $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$*, a statement* stmt*, and a proof* π*, and outputs* 1 *if the proof is valid and* 0 *otherwise.*

For the sake of simplicity we generally omit the security parameter 1^κ, the group parameter $\mathit{pp}_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ and the CRS crs.

The following definitions are implicitly with respect to group-based NIZKs.

4.4.1. Perfect Completeness

A ZK scheme is perfectly complete if a proof regarding a true relation will *always* be accepted. More formally [83]

Definition 4.4.2 (Perfect Completeness). *A Zero Knowledge scheme* $\mathcal{Z}\kappa = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Proof}, \mathsf{Verify})$ *is* perfectly complete *if for all adversary* $\mathcal{A}$ *we have that the following probability is* exactly *one:*

$$\Pr\left[\begin{array}{l} (\mathit{pp}_{\mathbb{G}}, \mathit{stmt}, \mathit{wit}) \in R \implies \\ \mathcal{Z}\kappa.\mathsf{Verify}(\mathit{pp}_{\mathbb{G}}, \mathit{crs}, \mathit{stmt}, \pi) = 1 \end{array} \middle| \begin{array}{l} (\mathit{pp}_{\mathbb{G}}, \mathsf{sk}) \leftarrow \mathcal{Z}\kappa.\mathsf{Gen}(1^\kappa), \\ \mathit{crs} \leftarrow \mathcal{Z}\kappa.\mathsf{Setup}(\mathit{pp}_{\mathbb{G}}), \\ (\mathit{stmt}, \mathit{wit}) \leftarrow \mathcal{A}(\mathit{pp}_{\mathbb{G}}, \mathit{crs}), \\ \pi \leftarrow \mathcal{Z}\kappa.\mathsf{Proof}(\mathit{pp}_{\mathbb{G}}, \mathit{crs}, \mathit{stmt}, \mathit{wit}) \end{array}\right] \qquad (4.2)$$

4.4.2. Perfect Soundness

Perfect soundness is completeness in the other direction; it formally states that it should be *impossible* to convince a verifier of a false statement [83].

Definition 4.4.3 (Perfect Soundness). *A Zero Knowledge scheme $\mathcal{Z}\kappa$ = (Gen, Setup, Proof, Verify) is* perfectly sound *if for all adversaries $\mathcal{A}$ we have that the following probability is* exactly *zero:*

$$\Pr\left[\begin{array}{l} stmt \notin \Lambda \implies \\ \mathcal{Z}\kappa.\mathsf{Verify}(pp_{\mathbb{G}}, crs, stmt, \pi) \end{array} \middle| \begin{array}{l} (pp_{\mathbb{G}}, \mathsf{sk}) \leftarrow \mathcal{Z}\kappa.\mathsf{Gen}(1^{\kappa}), \\ crs \leftarrow \mathcal{Z}\kappa.\mathsf{Setup}(pp_{\mathbb{G}}), \\ (stmt, \pi) \leftarrow \mathcal{A}(pp_{\mathbb{G}}, crs) \end{array}\right] \tag{4.3}$$

4.4.3. CRS Indistinguishability

CRS Indistinguishability intuitively means that there is some different way to compute the CRS such that

(1) the new CRS cannot be distinguished from one that was honestly created, and

(2) the new CRS can be constructed with a backdoor.

More formally:

Definition 4.4.4 (CRS Indistinguishability). *A Zero Knowledge scheme $\mathcal{Z}\kappa$ = (Gen, SetupExt, SetupSim, Proof, Verify, ExtractWit, SimZK) provides* computationally indistinguishable Common Reference Strings *if for every PPT-adversary $\mathcal{A}$ it holds that the following probability is negligible in the security parameter κ:*

$$\left|\begin{array}{r} \Pr\left[\mathcal{A}(crs) = 1 \middle| \begin{array}{l} (pp_{\mathbb{G}}, \mathsf{sk}) \leftarrow \mathcal{Z}\kappa.\mathsf{Gen}(1^{\kappa}), \\ (crs, td_{ext}) \leftarrow \mathcal{Z}\kappa.\mathsf{SetupExt}(pp_{\mathbb{G}}, \mathsf{sk}) \end{array}\right] \\ - \Pr\left[\mathcal{A}(crs) = 1 \middle| \begin{array}{l} (pp_{\mathbb{G}}, \mathsf{sk}) \leftarrow \mathcal{Z}\kappa.\mathsf{Gen}(1^{\kappa}), \\ (crs, td_{sim}) \leftarrow \mathcal{Z}\kappa.\mathsf{SetupSim}(pp_{\mathbb{G}}, \mathsf{sk}) \end{array}\right] \end{array}\right| \tag{4.4}$$

4.4.4. Perfect $F_{pp_{\mathbb{G}}}$-Extractability

This property implies a (limited) way of extracting information from the witness. While it is not possible to extract the witness directly, its exponentiation can be extracted efficiently.

Definition 4.4.5 (Perfect $F_{pp_{\mathbb{G}}}$-Extractability). *A Zero Knowledge scheme $\mathcal{Z}\kappa$ = (Gen, Setup, SetupExt, Proof, Verify, ExtractWit) is* perfectly $F_{pp_{\mathbb{G}}}$-extractable *if $\mathcal{Z}\kappa$ follows CRS indistinguishability (Definition 4.4.4) and for all adversaries $\mathcal{A}$ the following probability is* exactly *one:*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(crs, stmt, \pi) = 1 \implies \\ \exists_{wit} F_{gp}(wit) = wit' \end{array} \middle| \begin{array}{l} pp_{\mathbb{G}} \leftarrow \mathsf{Gen}(1^{\kappa}), \\ (crs, td_{ext}) \leftarrow \mathsf{SetupExt}(pp_{\mathbb{G}}), \\ (stmt, \pi) \leftarrow \mathcal{A}(crs, td_{ext}), \\ wit' \leftarrow \mathsf{ExtractWit}(crs, td_{ext}, stmt, \pi) \end{array}\right] \tag{4.5}$$

4.4.5. Dual-Mode

Dual-Mode Zero Knowledge generally depends on the Common Reference String *crs*. Such schemes provide two different distributions on how the CRS is created which follow CRS indistinguishability (Definition 4.4.4).

We furthermore require $F_{pp_{\mathbb{G}}}$*-extractability* in the mode that uses SetupExt to set up the CRS [83].

The second mode that uses SetupSim is supposed to provide *zero-knowledge*:

Definition 4.4.6 (Statistical Zero Knowledge)**.** *A Zero Knowledge scheme* $\mathcal{ZK} = (\mathsf{Gen}, \mathsf{SetupExt}, \mathsf{SetupSim}, \mathsf{Proof}, \mathsf{Verify}, \mathsf{ExtractWit}, \mathsf{SimZK})$ *provides* Statistical Simulatability *if for every adversary $\mathcal{A}$ the following probability is overwhelming in the security parameter κ:*

$$\Pr\left[\mathcal{ZK}.\mathsf{Verify}(crs, stmt, \pi) = 1 \;\middle|\; \begin{array}{l} (pp_{\mathbb{G}}, \mathsf{sk}) \leftarrow \mathcal{ZK}.\mathsf{Gen}(1^{\kappa}), \\ (crs, td_{sim}) \leftarrow \mathcal{ZK}.\mathsf{SetupSim}(pp_{\mathbb{G}}, \mathsf{sk}), \\ (stmt, wit) \leftarrow \mathcal{A}(pp_{\mathbb{G}}, crs), \\ \pi \leftarrow \mathcal{ZK}.\mathsf{SimZK}(stmt, crs, td_{sim}) \end{array}\right] \tag{4.6}$$

4.4.6. Dual-Mode Zero Knowledge Scheme

We now have all the tools required to define a dual-mode Zero Knowledge scheme.

Definition 4.4.7 (Dual-Mode Zero Knowledge Scheme)**.** *A Zero Knowledge scheme* $\mathcal{ZK} = (\mathsf{Gen}, \mathsf{SetupExt}, \mathsf{SetupSim}, \mathsf{Proof}, \mathsf{Verify}, \mathsf{ExtractWit}, \mathsf{SimZK})$ *is called* Dual-Mode *Zero Knowledge scheme if all of the following conditions are fulfilled:*

1) *$\mathcal{ZK}$ has* CRS indistinguishability.
2) *$\mathcal{ZK}$, when set up with* $(crs, td_{ext}) \leftarrow \mathcal{ZK}.\mathsf{SetupExt}(\cdot)$, *has* perfect completeness.
3) *$\mathcal{ZK}$, when set up with* $(crs, td_{sim}) \leftarrow \mathcal{ZK}.\mathsf{SetupSim}(\cdot)$, *has* perfect completeness.
4) *$\mathcal{ZK}$, when set up with* $(crs, td_{ext}) \leftarrow \mathcal{ZK}.\mathsf{SetupExt}(\cdot)$, *has* perfect soundness.
5) *$\mathcal{ZK}$, when set up with* $(crs, td_{ext}) \leftarrow \mathcal{ZK}.\mathsf{SetupExt}(\cdot)$, *has* $F_{pp_{\mathbb{G}}}$-extractability.
6) *$\mathcal{ZK}$, when set up with* $(crs, td_{sim}) \leftarrow \mathcal{ZK}.\mathsf{SetupSim}(\cdot)$ *has* statistical Zero Knowledge.

4.5. The Universal Composability Framework

4.5.1. Description

We perform our analysis in the Universal Composability (UC) framework [40, 41], which is a strong version of simulation-based security [79, 77]. The key idea there is to compare a real protocol execution between mutually distrustful parties to an idealized execution, where a trusted party performs the computation based on the participants inputs. The behavior of the trusted party is specified by an *ideal functionality* $\mathcal{F}$. In the real world, all parties execute a protocol Π, which is said to *realize* the functionality $\mathcal{F}$ (written as $\Pi \geq \mathcal{F}$) if it can be shown to be indistinguishable from the ideal world. On a formal level, we want that for all environments $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ (that can adaptively depend on the behavior of $\mathcal{Z}$) such that the view in the real world is *indistinguishable* from the view in the ideal world that the simulator provides. This simulator creates a transcript of an execution without knowing the parties' inputs. More precisely, the transcripts of both worlds must be indistinguishable for any non-participant, even those who know the parties' secret inputs. Indistinguishability of the two worlds implies that the real adversary cannot learn anything from the real protocol execution that the simulator cannot contrive without knowing the private inputs.

The UC framework provides much stronger security guarantees than the standalone model, but comes with some restrictions; without a trusted setup, no protocol Π can UC-realize functionalities such as commitments [43], while computational constructions in the standalone model exist. Constructions in the UC framework also hold in the standalone model and, conversely, impossibilities in the standalone model extend to the UC framework.

Regarding communication there is a distinction between the *synchronous* and *asynchronous* communication network. In an asynchronous model, messages are handled by the adversary who can drop all messages [52, 23], thus causing (anonymous) abort. In the synchronous model, the ability to drop messages is taken from the adversary and we implicitly assume that any message sent by a sender S also reaches the receiver R. Thus any protocol in the synchronous setting can be implicitly considered to consist of a set of *rounds*, where at the end of each round, any party can can send any message to any other party, and at the beginning of the next round, each party receives *every* message sent to it by other parties. The adversary still gets notified of the transfer but cannot suppress it.

We generally assume that the simulator gets notified whenever any party passes input to any functionality. The simulator doesn't learn anything regarding the parties secret inputs (except its length when applicable). It only learns that input was provided.

4.5.2. UC Subfunctionalities

Soon after the UC framework was introduced by Canetti [40, 41], it was shown by Canetti and Fischlin [43] that it is too restrictive to construct even MPC-incomplete functionalities

Functionality $\mathcal{F}_{\mathsf{CRS}}$

$\mathcal{F}_{\mathsf{CRS}}$ sets up a Common Reference String according to a given distribution. It is parameterized by a distribution QD. It is running with a set of N parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$.

On input $(\mathtt{Value}, sid)$ by some party P_i, if this is the first activation, sample a value $crs \overset{\$}{\leftarrow} \mathrm{QD}$ and return crs to P_i.

On input $(\mathtt{Value}, sid)$ by some any party P_i for any further invocations, directly return crs to P_i.

Figure 4.1.: The functionality $\mathcal{F}_{\mathsf{CRS}}$ for setting up a Common Reference String.

such as commitments—which were possible without any additional assumptions in the standalone model. As a result, constructions in the UC framework require subfunctionalities. Those are functionalities which fulfill a certain task. They can then be instantiated based on hardware assumptions [81], computational assumptions [7, 117], or simply by letting it execute by a trusted party.

An additional benefit of subfunctionalities is the fact that it enables a more *modular* approach to protocol design. If the overall protocol uses some mechanism for, say, key exchange, but works independently of the actual protocol (that is, in a *black-box* way), then it makes sense to model a protocol with a key exchange subfunctionality.

In this section we introduce the subfunctionalities we use for this part of the thesis.

4.5.2.1. Common Reference String

The Common Reference String (CRS) is a publicly accessible string that is sampled at the beginning of a protocol from a distribution QD that is known to all parties.

In Fig. 4.1 we show the UC functionality for setting up and maintaining a CRS as it was described by Canetti and Fischlin [43]. On its first activation it samples the CRS from the distribution QD. On any further activations it returns that same CRS that was sampled during the first activation.

4.5.2.2. Registration

The functionality $\mathcal{F}_{\mathsf{Reg}}$ for registering public keys was introduced as *Certification Authority* by Canetti [42] and is often referred to as *Bulletin Board* in the literature (*e.g.* in [137, 70]).

Functionality $\mathcal{F}_{\mathsf{Reg}}$

$\mathcal{F}_{\mathsf{Reg}}$ lets parties register public keys. It is running with a set of N parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$.

On input $(\mathtt{Register}, v)$ by some party P_i, leak $(\mathtt{Register}, v)$ to the adversary and await the message Ok by the adversary.

Then, abort if this is not the first request by P_i, and otherwise store the pair (P_i, v).

On input $(\mathtt{Fetch}, \mathit{pid}_{\mathsf{P}})$ by some any party P_i, send $(\mathtt{Fetch}, \mathsf{P}, \mathsf{P}_i)$ to the adversary and await response $\mathtt{Ok}$. Then abort if there is no pair $(\mathit{pid}_{\mathsf{P}}, v)$, and otherwise load v and return (v) to P_i.

Figure 4.2.: The functionality $\mathcal{F}_{\mathsf{Reg}}$ for registering public keys.

Functionality $\mathcal{F}_{\mathsf{SMT}}$

$\mathcal{F}_{\mathsf{SMT}}$ enables confidential and authenticated communication. It is running with a sender S and a receiver R.

On input $(\mathtt{Send}, \mathsf{R}, \mathit{msg})$ by S with pid $\mathit{pid}_{\mathsf{S}}$, leak $(\mathtt{Sent}, \mathsf{S}, \mathsf{R}, |\mathit{msg}|)$ to the adversary.

On input $(\mathtt{Ok}, \mathit{msg}^*, \mathsf{R}^*)$ by $\mathcal{A}$, output $(\mathtt{Sent}, \mathit{pid}_{\mathsf{S}}, \mathit{msg})$ to R if S is honest and $(\mathtt{Sent}, \mathit{pid}_{\mathsf{S}}, \mathit{msg}^*)$ to R^* if S is corrupted.

Figure 4.3.: The functionality $\mathcal{F}_{\mathsf{SMT}}$ for secure message transfer.

The functionality basically acts as a database, where each party can store a message. In the original paper this message was assumed to be a public key, yet the functionality does not enforce this and only expects some binary string v. However, since the main body of this part uses the functionality only to register and fetch key material, we stick with the notion of a Registration functionality.

4.5.2.3. Secure Message Transfer

The classical communication model in UC has very little to offer with respect to security. Any party can send a message to any other party, but the network through which messages are sent is controlled by the adversary. As such, the adversary can read every message sent from any party to any other party—even between two honest parties. Furthermore, the adversary can freely change the content of the message *and* the reported sender before delivery.

To enable communication that fulfills both *confidentiality*—the adversary only learns the length of the messages, that is, the amount of bits sent through the respective channels, but no longer learns the messages content—and *authenticity*—the message received should be clearly recognizable as coming from the supposed sender and should contain exactly the massage that the sender sent—a common abstraction in the UC framework is to assume Secure Message Transfer (SMT).

The UC subfunctionality from Canetti [40] is given in Fig. 4.3. It models sending a message from a sender S to a receiver R in such a way that it leaks only the sender and the receiver and the *length* of the message to the adversary. The adversary can then insert a different message and receiver, and this will only be considered if the sender is corrupted. Otherwise, the original input from the sender will be sent to the receiver determined by the senders input.

4.5.2.4. Onion-Routing with Replies

Onion-Routing with Replies (ORR) is an extension to classical Onion Routing services like TOR [62]. Onion Routing in general aims to provide anonymity for clients against servers by routing the messages through a set of *mix-servers* (sometimes also called *proxy* servers in the literature). In TOR, there are three such servers between each client and server: an *entrance node*, an *intermediate node* and an *exit node*. The user picks one for each from a public list of available mix-servers alongside their public keys and instead of sending a message *msg* to the server directly, the user now first encrypts *msg* alongside the recipient with the key of the *exit node*, then adds the IP of the exit node and encrypts that with the key of the *intermediate node*, and finally encrypts the resulting ciphertext alongside the IP of the intermediate node with the public key of the *entrance node*. The result is then handed to the entrance node.

Each node then decrypts the cipher text and forwards the result to the target address[1]. The exit node then forwards the message to the final receiver. This way, assuming non-colluding servers, the entrance node only knows that the sender is communicating over TOR, and the exit node can see the message and the receiver but is unaware of the actual sender. Using encryption on top ensures that also the message is hidden from the exit node.

TOR solves the problem of letting the user send messages to a server, but not the problem of letting the server *respond* to a given message. To solve that as well Ando and Lysyanskaya [11] introduced Onion-Routing with Replies (ORR) (which was later extended by Kuhn et al. [104]), where the sender also implicitly specifies a *return path* for the message.

[1] Hence the name Onion Routing, as this process is reminiscent of peeling of layers from an onion.

Functionality $\mathcal{F}_{\mathsf{ORR}}$

$\mathcal{F}_{\mathsf{ORR}}$ provides anonymous communication using Onion-Routing with Replies. It is running with mix-servers $\mathsf{M}_1, \ldots, \mathsf{M}_{N_\mathsf{M}}$ and parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$ we refer to as sender S and receiver R.

Init: Initialize a list $L = \emptyset$ of onions processed by corrupted mix-servers, for each mix-server a list $B_i = \emptyset$ of onions held by M_i, an empty mapping *Back* from the temporary id ϑ of an onion to their path and forward id, and an empty mapping ID_{fwd} from backward id to forward id.

On input $(\texttt{Process\_Onion}, \mathsf{R}, msg, P^{\rightarrow}, P^{\leftarrow})$ from S, abort if $|P| > \Psi$. Otherwise, sample sid and create an onion as $O \leftarrow (sid, \mathsf{S}, \mathsf{R}, msg, P^{\rightarrow}, P^{\leftarrow}, 0, fw)$. If S is corrupted, leak $(0, sid, \mathsf{R}, msg, P^{\rightarrow}, P^{\leftarrow}, fw)$ to the adversary. Handle the message according to $\texttt{Process\_Next\_Step}(O)$ from Fig. 4.5.

On input $(\texttt{Process\_Back\_Onion}, msg, \vartheta)$ from R, abort if $Back(\vartheta) = \perp$. Otherwise, set $(\mathsf{S}, P^{\rightarrow}, P^{\leftarrow}, \mathsf{R}, sid') := Back(\vartheta)$. Sample sid and create an onion as $O \leftarrow (sid, \mathsf{S}, \mathsf{R}, msg, P^{\leftarrow}, P^{\rightarrow}, 0, bw)$. If S is corrupted, leak $(0, sid, \mathsf{R}, msg, P^{\rightarrow}, P^{\leftarrow}, fw, sid')$ to the adversary where sid' is the forwards id of the same onion. Handle the message according to $\texttt{Process\_Next\_Step}(O)$.

On input $(\texttt{Deliver\_Message}, \vartheta)$ from $\mathcal{A}$, pick (O, j) from L according to ϑ.

If $j < |P|+1$, sample a new temporary id ϑ' uniformly at random, send ϑ' *received* to M_j, and store (ϑ', O) in B_j.

Otherwise, abort if $msg = \perp$ and else send $(MESSAGE, msg, \vartheta, d)$ to R. Then, if $P^{\leftarrow} \neq \emptyset$ and $d = fw$, send (fw) to R and set $Back(\vartheta) := (\mathsf{S}, P^{\rightarrow}, P^{\leftarrow}, \mathsf{R}, sid)$.

On input $(\texttt{Forward\_Onion}(\vartheta'))$ from M_i, abort if $(\vartheta', \cdot) \notin B_i$ and otherwise load and remove (ϑ, O) from B_i and handle the retrieved onion according to $\texttt{Process\_Next\_Step}(O)$.

Figure 4.4.: The functionality $\mathcal{F}_{\mathsf{ORR}}$ for Onion-Routing with Replies.

In Fig. 4.4 we show the ideal functionality presented by Kuhn et al. [104, Full Version, Algorithms 1 and 2]. A sender S can insert a new onion using the `Process_Onion` task, where a dedicated receiver and the message are inserted alongside a forward path $P^{\rightarrow}$ and a backwards path $P^{\leftarrow}$. Both paths contain a set of *mix-servers* through which the message is scheduled. Each step is simulated through the `Process_Next_Step` procedure from Fig. 4.5. This method simulates forwarding the message, but groups the steps that involve only corrupted mix servers as those are all controlled by the adversary anyways.

```
**Procedure** Process_Next_Step($O = (sid, S, R, msg, P^{\rightarrow}, P^{\leftarrow}, i, d)$)
Subprocedure Process_Next_Step for the ideal functionality of Onion-Routing with Replies in Fig. 4.4.
.................................................................................
if $(M_{i+1}, \ldots, M_n, R)$ are all corrupted then
  Leak $(\vartheta, d, M_i, R, msg, (M_{i+1}, \ldots, M_n))$ to the adversary.
  if $d = fw$ then
    $Back(\vartheta) := (S, P^{\rightarrow}, P^{\leftarrow}, R, sid)$
    Let $j^*$ be the biggest $j^*$ such that $(M_1, \ldots, M_{j^*}) \in P^{\leftarrow}$ are all corrupted. Leak $(M_1, \ldots, M_{j^*+1})$ from $P^{\leftarrow}$ to the adversary.
    If S is corrupted, leak $(\vartheta, (sid, R, msg, P^{\rightarrow}, P^{\leftarrow}, fw))$ to the adversary.
  elseif $d = bw$ then
    If S is corrupted, leak $(\vartheta, (sid, R, msg, P^{\rightarrow}, P^{\leftarrow}, bw, sid'))$ to the adversary where $sid'$ is the forwards id of the same onion.
  fi
elseif $\exists j > i$: $M_j$ is not corrupted then
  $j^* = \min_j j$ such that $M_j$ is not corrupted
  Sample random id $\vartheta$ and leak $(\vartheta, M_i, (M_{i+1}, \ldots, M_{j^*-1}))$ to the adversary.
  $L = L \cup (\vartheta, O, j^*)$
  if $d = fw$ then
    If S is corrupted, leak $(\vartheta, (sid, R, msg, P^{\rightarrow}, P^{\leftarrow}, fw))$ to the adversary.
  elseif $d = bw$
    If S is corrupted, leak $(\vartheta, (sid, R, msg, P^{\rightarrow}, P^{\leftarrow}, bw, sid'))$ to the adversary where $sid'$ is the forwards id of the same onion.
    If additionally $i = 0$ leak $(\vartheta, sid)$ to the adversary.
    fi
  fi
fi
```

Figure 4.5.: Procedure Process_Next_Step(O) for an onion O.

At its destination the functionality outputs the message to the operator; assuming that a sufficient number of mix servers can be trusted and that a sufficient number of users have provided input, the adversary cannot link the output to an input.

5. Overview

This chapter provides an overview to the parties and tasks involved in this part of the thesis. It is heavily based on the conference publication [70, Sections 2 and 3] this part of the thesis is based on and thus contains many similar parts.

5.1. Parties and Roles

We consider five different types of parties:

Users U**.** There are arbitrarily many users participating at the same time. Each user collects data inside a personal logbook λ. The main part of this data is called User History (UH) *UH*. This is the part of the data users want to provide for privacy-preserving computations. The UH is authenticated and cannot be changed by the user at will. The only way to update the data is by interacting with the operator in private bookkeeping or analytics tasks. The private data is represented by a *vector* of *slots*. Each slot contains a $\mathbb{Z}_o$-element. Users can be corrupted by the adversary and maliciously collude with other corrupted parties. We assume that users participate using relatively weak hardware. This is due to the fact that many real-world applications would be executed on smartphones.

The Operator O**.** For any given instance of we there is exactly one operator. This party is the central entity managing the system. The operator has an interest in evaluating analytical functions on the data collected by users. In certain scenarios, the precise details of these functions, however, may be subject to trade secrets and should not be leaked to any other party. We thus assume that the operator can *hide* all sensitive information inside Function Parameters (FPs) *fp* while the publicly known function f itself only provides the general structure. In real applications, FPs correspond to transition matrices of neural networks [94] or weights and features for linear regression. Being the operator's input to analytics computations they achieve the same level of privacy as the users' private data. Relying on FPs is without loss of generality [131] regarding the class of computable functions. The operator may be corrupted and maliciously collude with other corrupted parties apart from helpers without affecting the user's privacy.

The Trusted Signing Authority T. Allowing arbitrary private FPs precludes any meaningful level of privacy for the users. To prevent malicious operators from using FPs which trivially undermine the users' privacy, we assume the existence of a Trusted Signing Authority (TSA): Before using FPs for computations, the operator has to let them be certified by the TSA to not violate privacy requirements. We elaborate on the difficulties of this task in Section 5.10.1. We enforce that only FPs certified by the TSA can be used during computations on the users' data. The TSA is an external entity and has to be trusted by the users and the operator. We consider the TSA to be a privacy-defending donation-funded Non-Governmental-Organization (like the Electronic Frontier Foundation) or the Federal Data Protection Officer.

Helpers H. To help the users with analytical computations there are arbitrarily many helpers. Helpers are used for computing analytical functions and to coordinate computations that require data from multiple users. We assume helpers to provide the computational power and bandwidth of a regular server. Helpers may be corrupted and collude with corrupted users. As long as helpers do not collude with the operator, all user data is hidden from both parties. Any user may potentially set up their own helper, yet it is also possible that Non-Governmental-Organizations provide donation-funded helper servers.

Mix-Servers M. Mix-Servers only appear in ORR (see Section 4.5.2.4). It is their job to provide an anonymous channel between the user and the operator or helper. Normal communication in UC leaks the unique pid to the receiver, in reality, the IP address can be used to determine who the sender of a message is—or even, in case of static IPs, to group different interactions together. To circumvent such attack we require Mix Servers which *relay* the traffic of a user. The mix servers in ORR obtain an encrypted message which they decrypt, then extract the new recipient from the result and forward the dedicated message to them. That way, users only interact with mix-servers, mix servers only interact with mix-servers, and mix-servers only interact with the operator or helper; but (assuming that at least one of the intermediate mix servers is trusted) there is no direct link between the user and the operator or helper.

5.2. Tasks

In this section we introduce all the tasks we consider. These are distributed into *preparatory* tasks which we introduce in Section 5.2.1, simple two-party *bookkeeping* tasks which we present in Section 5.2.2, and *outsourced* tasks for complex analytical function which we describe in Section 5.2.3.

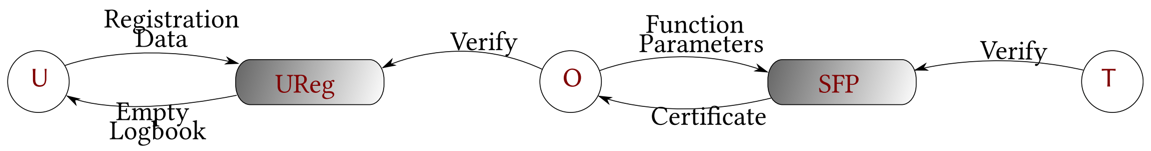


Figure 5.1.: Overview of the preparatory tasks.

5.2.1. Preparatory Tasks

Before users can participate in the system they have to register with the operator. Similarly, the operator has to have the FPs signed by the TSA before they can be used for computations. Both tasks have to be conducted only once (per user/FP, resp.) before they are eligible for any other computations. An overview of these tasks is provided in Fig. 5.1.

Sign Function Parameter. Any function the operator wants to compute is split in two parts: A generic and publicly known function representation f (*e.g.*, a neural network with padded neurons) and a corresponding private set of FPs which determine the specifics (*e.g.*, the transition matrices). These FPs constitute sensitive data held by the operator. Before the operator can use any FPs for computations, the TSA has to verify that the resulting function does not violate the required privacy standards. To that end, the TSA verifies the FPs according (but not limited) to the following criteria: 1) Is the output sufficiently general to not leak confidential user information? We assume a public catalog of requirements FPs have to fulfill. This also provides feedback for the users on the expected level of privacy. 2) Does this function use the slots of the UH according to their specification? We generally assume that the specification of the UH—the semantic interpretation of the individual slots—to be public knowledge. 3) Are there any additional leaks when combining these FPs with any of the previously certified FPs? We model the verification function such that the TSA can input all previously accepted FPs as auxiliary input. The actual Sign Function Parameter (SFP) task is depicted in Fig. 5.1: The operator inputs FPs which are then checked by the TSA against the privacy guidelines. If the FPs comply with the guidelines, the TSA provides the operator with some form of certificate. Our protocol uses signatures on the FPs. This certificate is a required input to any computation task that uses these FPs.

User Registration. Users store their private User History inside a personal logbook λ. Providing the user with an "empty" logbook which contains an authenticated initial UH is the purpose of the User Registration (UReg) task. The task is depicted in Fig. 5.1 and consists of three phases. In the first phase the user is identified. The operator then verifies that the user is not registered already—we require that each user can only have *at most* one logbook. Lastly both parties jointly compute the initial logbook containing an authenticated UH. The UH always starts out empty. The logbook λ is a requirement for participation in any of the other tasks. The UReg task is the *only* task identifying the user and only performed once.

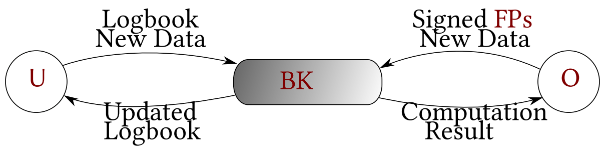


Figure 5.2.: Overview of the Bookkeeping task.

5.2.2. Bookkeeping

The privacy-preserving bookkeeping enables the operator and a user to manipulate data with certain guarantees for both parties. An overview of this task can be found in Fig. 5.2.

Bookkeeping. The Bookkeeping (BK) task manipulates a single user's data. While the main purpose of this task is to provide an efficient targeted manipulation of the UH in an authenticated yet privacy-preserving way (such as adding or resetting individual values), the task optionally performs *lightweight* direct analytics of the user data according to an application-specific function f and the certified FPs. This enables manipulations of the UH based on its current values, which may be required for scenarios such as fraud detection where the decision (to be recorded) whether a transaction is granted or not depends on a risk level stored in UH. The main part—authenticated manipulation of the UH—is done without leaking the authenticated and private data stored in the UH to the operator. The UH is updated using Update Information $UI = (\vec{\alpha}, \vec{s}, \vec{a})$. These contain three consecutive operations defined by three maps which are output by f. The first map $\vec{\alpha}$ defines a *permutation* of the contents inside the UH. The second map $\vec{s}$ determines which values of the UH are set to new values directly, we call this *direct update* throughout this part of the thesis. The final map $\vec{a}$ is defined by a vector of additive updates which will be applied to the values of UH, we call this vector *additive increment*. The additive increment is hidden from the operator. In contrast, the permutation and direct updates are learned by both parties. As such they are subject to privacy considerations during auditing with the TSA to ensure that they leak no personal information about the user. We stress that permutations and direct updates are optional; they can be skipped during the BK task in favor of more efficient updates using only the additive increment. We chose this type of update mechanism as a tradeoff between what is usually required for (basic) bookkeeping applications and what can be efficiently implemented (using zero-knowledge proofs and homomorphic commitments).

The UH remains authenticated because (1) the operator knows that the input was authenticated, and (2) the correctness properties ensure that the data was manipulated correctly.

5.2.3. Outsourcing Analytical Computations

We assume that users have relatively weak hardware incapable of securely computing analytical functions like neural networks efficiently. To still enable these costly computations we involve an additional party—the helper—which provides the computational

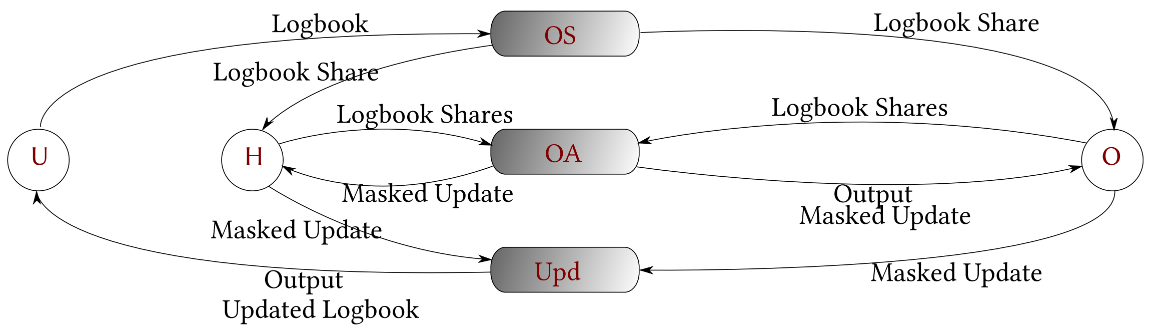


Figure 5.3.: Overview of tasks for outsourcing computations.

power and bandwidth required to perform such computations. As long as the helper does not collude with the operator it is not possible for the helper to learn any secrets or even analytical results from participation. A corrupted helper also poses no risk for the operator's privacy.

Involving a helper also enables *synchronization* of analytical tasks which require private data from several users. The basic workflow for outsourcing analytical tasks is shown in Fig. 5.3; note that the three different tasks involved in this cannot be scheduled arbitrarily but have to be conducted top to bottom. The general workflow first lets the user *distribute* the logbook containing their current UH to a helper of its choice and the operator using the Outsource task. These shares are then used by helper and operator for computing the analytical function using the Outsourced Analytics task. This can potentially cause updates on the UH which follow the same three basic operations (permutation, direct update, additive increment) that were also used for the Bookkeeping task. Again, we want to hide the additive vector applied to the User History and thus use a similar mechanism as during the BK task, only that the values relevant for the user are temporarily stored by the helper and hence masked with One Time Pads. The update can later be applied by the user by means of the Update task which also re-enables the Outsource task for that user.

Outsource. To prepare the computation of an analytical task the Outsource (OS) task is conducted between user, helper and operator as shown in Fig. 5.3. The user's logbook, which contains the latest authenticated User History, is shared between the operator and helper for use in the computation. The user receives a new logbook which is identical to the old one except that it is now marked as having been outsourced.

Outsourced Analytics. Using the shares obtained from the OS tasks, helper and operator can now conduct the actual computation with the Outsourced Analytics (OA) task. This is depicted in Fig. 5.3 as well. The function is computed on the private user data using any secret-sharing based MPC framework. The operator learns the desired analytical result directly. Information relevant for the user is masked and sent as output directly to the helper; the users fetch the data and reconstruct it at their convenience but the helper does not learn the data. Computation again follows the function f and requires validated FPs from the operator.

Update. As mentioned before, the results of OA can be used to update the user data. To apply these to the latest UH the user and operator can use the Update (Upd) task (see Fig. 5.3). This also re-enables the Outsource task for their logbook. The analytical result destined for the user is forwarded to them. Both the operator and helper input the share they obtained from the Outsourced Analytics task. These shares contain the updates for the UH, the computation results relevant for the user and additional information to detect tampering of the shares. The user inputs the latest UH and obtains a new authenticated UH which was updated using the same mechanism as used in the BK task.

The triplet from Fig. 5.3 is *non-blocking* regarding further Bookkeeping tasks: A user can outsource the latest User History in an Outsource task, which prepares an Outsourced Analytics task in the "background", and then update the UH using the Bookkeeping task arbitrarily often *before* participating in the Update task. This is a desired behavior in scenarios such as targeted advertising. We stress, however, that the triplet is blocking with respect to further analytical tasks (amongst others, to hamper denial of service attacks on helpers and operator): After executing the OS task the user has to successfully perform the Upd task before it can call Outsource again. This implies that the User History input to the Update task has potentially been changed via successive Bookkeeping tasks and significantly differs from the UH used during Outsource. Special care has to be taken during function design to ensure that the update is still meaningful even after any number of Bookkeeping tasks manipulating the UH in the meantime.

5.3. Cryptographic Building Blocks

We make use of the following building blocks.

A Non-Interactive Zero-Knowledge Proof of Knowledge scheme. This scheme is used by the user to prove that certain operations have been performed correctly. It needs to be extractable and zero-knowledge. Groth-Sahai proofs [83] satisfy our requirements.

A pairing group. We make use of an asymmetric pairing group $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \mathrm{e}, o, g_1, g_2)$. The groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$ of prime order o are cyclic groups with generators g_1 and g_2. Identification of a user relies on the hardness of the Co-CDH problem [31], which asks to compute g_2^x given g_1, g_2, g_1^x. The Co-DH assumption is implied by the SXDH assumption [83] we use to instantiate Groth-Sahai proofs.

An additively homomorphic, structure-preserving commitment scheme. The logbook contains the user data, *i.e.* the User History, alongside additional information alongside their commitments; we provide more information on the logbook in Section 5.6. These commitments are from an additively homomorphic, unconditionally hiding and computationally binding commitment scheme. To ensure compatibility with our zero knowledge proof system the commitments are additionally structure-preserving. A scheme that fulfills all those requirements is the commitment scheme from Abe et al. [2].

A structure-preserving EUF-CMA-secure signature scheme. To ensure the integrity of the logbook, we use an EUF-CMA-secure signature scheme which is compatible (structure-preserving) with our zero-knowledge proof system. The signature scheme by Abe et al. [1] provides these requirements.

A Robust Secret Sharing scheme. We use a robust secret sharing scheme, which lets a party create shares of a secret in such a way that (1) the recipients can verify the integrity of the received shares, and (2) tampering with the shares can be detected during reconstruction. Unlike *verifiable* secret sharing which only protects against a malicious dealer, *robust* secret sharing also protects against recipients trying to manipulate their shares in order to change the reconstructed output.

5.4. Set-Up Assumptions

We formally conduct our investigation in the asynchronous UC framework with anonymous abort against static corruption of an arbitrary subset of parties that does not include both the helper *and* the operator at the same time. Following strong impossibility results [43] regarding constructions in the UC-framework, most instantiations require set-up assumptions: Building blocks with a pre-defined behavior that are generally used in a black-box way. This means that those too can be interpreted as functionalities which are controlled entirely by the simulator—providing an advantage over real-world adversaries. The set up assumptions can be instantiated by any protocol realizing this functionality or even by using trusted hardware.

We require the following set-up assumptions:

$\mathcal{F}_{\mathrm{CRS}}$: **Common Reference String (CRS)** A CRS is a string visible to all parties, sampled from a publicly known distribution.

$\mathcal{F}_{\mathrm{Reg}}$: **Registration (Reg)** Reg is a common abstraction to model a Public-Key Infrastructure (PKI). Parties can register their own public keys and fetch keys from other parties. Unlike real PKIs, $\mathcal{F}_{\mathrm{Reg}}$ allow users to register a key *exactly once* and stored keys do not expire.

$\mathcal{F}_{\mathrm{MPC}}$: **Secure Multi-Party Computation (MPC)** Using MPC the parties can compute the function f on their private inputs in such a way, that no additional information on the individual inputs is leaked except for the output.

$\mathcal{F}_{\text{SMT}}$: **Secure Message Transfer (SMT)** SMT (*cf.* Section 4.5.2.3) provides a secure channel between any two parties. The messages are both authenticated and confidential. Unless we explicitly state so in our protocol, we assume that all messages are sent via SMT.

$\mathcal{F}_{\text{ORR}}$: **Onion-Routing with Replies (ORR)** ORR (*cf.* Section 4.5.2.4) provides onion routing in the sense of Dingledine, Mathewson, and Syverson [62], but in UC secure and with the ability of the sender to provide a return path. The receiver thus only communicates with mix servers, and not with the user directly.

5.5. Computation of Benign Functions f

Our contribution lies in the construction and analysis of a privacy-preserving bookkeeping mechanism built around *any* existing MPC framework. While the MPC building block we use enables computation of *any* function f, we only compute *benign* functions. We will elaborate in Section 7.1 but stress here that the restriction (1) is necessary as we require specific pre- and post-processing steps, and (2) does not restrict the basic set of computable functions any further, as any function can be transformed into a benign function by adding these pre- and post-processing steps.

As functions can be used for BK and OA let's briefly discuss the pre- and post-processing steps required for each task.

Bookkeeping. As a *pre-processing* step the function needs to verify that the operator used the same FPs for the MPC that were also verified earlier. To that end we require the operator to input a *signature* on these FPs that was signed by the TSA. A similar trick is used to ensure that the user uses the same UH that was verified before the computation. The *post-processing* step ensures that the additive increment of the update is not be learned by the operator. To that end, f computes a commitment and only outputs that to the operator (alongside the permutation and the direct update as clear vectors) while the user additionally obtains the clear text and the unveil information.

Outsourced Analytics. During the OS task the user created and distributed robust secret shares of the UH and proved that these were correctly distributed. Our *pre-processing* phase then verifies that the shares are reconstructed correctly, *i.e.*, that no tampering has been detected. Then the FPs are verified just as in BK. *Post-processing* then proceeds as in BK and computes a commitment on the additive increment. It additionally applies the OTPs on the permutation, the direct update, the increment, the unveil information on the increment, and the unauthenticated output, as those are routed through the helper.

5.6. The User Logbook

We denote by λ the logbook containing the data stored by a user:

$$\lambda = \begin{pmatrix} UH & ser & lin & \mathsf{sk}_{\mathsf{U}} & \\ \mathtt{com}_{UH}, & \mathtt{com}_{ser}, & \mathtt{com}_{lin}, & \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, & \sigma_{UH} \\ \mathtt{unv}_{UH} & \mathtt{unv}_{ser} & \mathtt{unv}_{lin} & \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}} & \end{pmatrix} \tag{5.1}$$

The logbook contains all the data required to maintain the User History and to anonymously interact with the operator.

The User History UH**.** The key component of the logbook is the User History. It is a vector of $\mathbb{Z}_o$ elements that represents the authenticated data collected by the user.

The Serial Number ser**.** The serial number is a single $\mathbb{Z}_o$ element that uniquely determines a revision of a logbook. It is unique in that with overwhelming probability there are no two different logbooks (neither belonging to different users nor to the same user) with the same serial number. This means that once a serial number has been used, it will never be used again.

The Linking Number lin**.** We require linkability *inside* the triplet for outsourcing computations: consecutive tasks of Outsource, Outsourced Analytics and Update have to be linked to the same user as otherwise the data outsourced during the Outsource task cannot be used during Outsourced Analytics and the changes of the resulting update cannot be applied to the correct User History during the Update task. We thus use an additional $\mathbb{Z}_o$ element that links these executions for all three parties involved in outsourcing computations, which the user stores inside the logbook. If no data has been outsourced since the latest update then the linking number is 0.

The Secret Key sk_{U}**.** The user has a fixed private key. On a technical level, this key is a random $\mathbb{Z}_o$ element. It is randomly chosen during User Registration and is never directly revealed to anybody. Instead, users only prove knowledge of sk_{U}.

The Commitment Information $\mathtt{com}$**,** $\mathtt{unv}$ **and Signature** σ_{UH}**.** To ensure authenticity the operator generally signs all values inside the logbook. However, the signature is not on the values directly—as this would conflict with the users privacy requirement—but on *commitments* thereof. This is why the user not only stores the values inside the logbook but also the commitments that were used by the operator to compute the signature. As those are part of the witness to generate zero-knowledge proofs, the user also stores the corresponding unveil information.

The final value in the logbook is a signature by the operator that ensures integrity of the commitments on the UH, the serial number, the linking number and the identity.

5.7. General Principles

For every task involving the user, our protocol begins and ends with the same two mechanisms: The authenticated input mechanism ensures the user enters a fresh and authenticated logbook into the interaction while the updating mechanism provides the user with a new valid logbook when the task is finished.

Authenticated Input Mechanism. At the start of each task the user owns a valid logbook λ containing the data described in Eq. (5.1). To prove validity of the logbook to the operator the user first *rerandomizes* all old commitments $\mathtt{com}^*_{msg}$ to commitments $\mathtt{com}_{msg}$ of the same value. Note that homomorphic commitments are trivially rerandomizable as homomorphically adding a commitment $\mathtt{com}_0$ on 0 to a given commitment changes its internal randomness, but not its value. In the second step the user computes a Zero Knowledge proof π showing that they know (1) commitments $\mathtt{com}^*$ on the same values as the $\mathtt{com}$ and (2) a signature σ_{UH} that authenticates the original commitments under the verification key of the operator. The rerandomized commitments $\mathtt{com}$ and proof π are sent to the operator for validation. The above process is only conducted for values the operator is not supposed to learn, usually $(UH, lin, \mathsf{sk}_\mathsf{U})$. In case the user wants to fetch updates from an outsourced analytical computation or wants to start one and needs to show the logbook contains $lin = 0$, the hidden values are $(UH, \mathsf{sk}_\mathsf{U})$ only. The serial number ser is always revealed at the start of a task and checked by the operator (using a database lookup) to make sure the user does not try to use an outdated logbook for a new task.

Updating Mechanism. At the end of each task the user and operator jointly compute a new valid logbook λ^{new} to be used in the next task. To do that, the operator needs to reliably learn commitments $(\mathtt{com}^{\text{new}}_{UH}, \mathtt{com}^{\text{new}}_{ser}, \mathtt{com}^{\text{new}}_{lin}, \mathtt{com}^{\text{new}}_{\mathsf{sk}_\mathsf{U}})$ to all new values $(UH^{\text{new}}, lin^{\text{new}}, ser^{\text{new}}, \mathsf{sk}^{\text{new}}_\mathsf{U})$ and sign them for the user without learning the values themselves. We explain how commitments to each of the values are obtained by the operator.

For the update of the User History we considered two different options. For the permutations and updates we considered two different options. The first one lets the operator do everything, yet only works with *individual* commitments for each value in the UH. As the operator, has access to all the individual commitments from the UH, the permutation can be applied directly by shuffling the individual commitments, whereas the direct update is applied by computing *new* commitments on the updated values and sending them alongside their unveil information to the user. However, having individual commitments has a very poor impact on the size of the logbook; so instead we chose a method compatible with shrinking commitments: The permutation and direct updates are performed the user, who hands the new UH alongside a ZK proof that the operations were applied correctly to the operator. The function f additionally provides *commitments* on the additive increments.

The operator then applies the additive increments by using the additive homomorphism of the commitment scheme.

In case there are only additive updates, the first step is skipped and only the additive increment is homomorphically applied to the commitment.

A new serial number is constructed using a coin-toss protocol similar to the protocol by Blum [28], but without the third round and by exploiting the homomorphic properties of the commitment scheme. In essence, the operator and user execute the following protocol: (1) The user picks a random share and sends a commitment of that share to the operator. (2) The operator picks a random share, computes a commitment of that share and homomorphically adds it to the commitment received by the user. That way, the operator has a commitment on the sum of both shares, which suffices for the signature creation. Finally, the operator sends its own serial number share alongside the commitment and unveil information to the user.

The linking number of the logbook stays the same for most interactions. Whenever the linking number is changed (in Outsource and Update) the operator knows the new linking number so the creation of the commitments is up to the operator. In the remaining tasks the linking number should not change due to the task execution, so the user sends a rerandomized commitment on the linking number and proves correct rerandomization.

The secret key is chosen once during registration of a user and then is never changed. Similar to the linking number, the operator only learns a rerandomized commitment and a proof of correct rerandomization.

The operator signs all the aforementioned commitments sends the to the user. The user verifies the signature and stores it in the new logbook.

5.8. Security Guarantees

This section discusses the security requirements in an informal fashion based on which we designed an ideal functionality in the UC framework in Chapter 6.

5.8.1. Operator

The *operator* expects authenticated inputs and correct analytical results, thereby hiding potential trade secrets. In particular, we desire the following security requirements for the operator:

Owner Binding. Users can only use *their own* User History. Even corrupted users cannot efficiently steal an honest user's UH. Generating a logbook on behalf of another user requires showing ownership of its public key; extracting this would violate the co-CDH assumption. Furthermore, the adversary cannot successfully steal another user's logbook as users never interact with each other, and communication uses confidential channels. Pretending the adversary's logbook belongs to a different user is also prevented: Given the (perfect) extractability of the NIZKPoK this would result in a different witness, which in turn means that the adversary (1) used different unveil information for the commitment, breaking the binding property, or (2) forged a signature on a new commitment, breaking EUF-CMA security.

History Freshness. No user can use an outdated User History for any of the Bookkeeping, Outsource or Update task: The same UH can never be used twice without the operator noticing. To achieve this goal we use an online-check of serial numbers. A user trying to re-use an old UH has only three options: (1) Lie during the ZK proof that the rerandomization of com_{ser} was correctly performed, thus breaking the soundness-property, or (2) open their coin toss commitment $\mathsf{com}^{(\mathsf{U})}$ during the creation of the new serial number to a different value, which would break the binding property, or (3) compute a new signature σ_{UH} on a changed commitment com'_{ser} that verifies under the operator's verification key, which would break the unforgeability of the signature scheme.

History Unforgeability. The User History can only be changed through task executions. It is computationally infeasible for a user to create a User History with arbitrary values that will be accepted by the operator. This hampers *Model Extraction Attacks* [128] in which the user uses targeted inputs to steal the FPs. The logbook entries are only used as witnesses for ZK proofs, but each of them comes with a commitment signed by the operator. Thus history unforgeability intuitively holds for the following reasons: (1) The soundness property of the ZK scheme ensures that proofs containing forged entries will be rejected with overwhelming probability. (2) The binding property of the commitment scheme and the unforgeability of the signature scheme further disable attacks where the commitments on the entries are opened to different values or where the signature on a manipulated entry is forged.

Uniqueness. Each user can have *at most* a single logbook. It is not efficiently possible for a user to own two valid logbooks at the same time. This property is enforced as follows: During the task for User Registration the operator fetches the public key of the user from $\mathcal{F}_{\mathsf{Reg}}$, which models a Public-Key Infrastructure (PKI) in UC but with the difference that each user can only register *one* key. If the public key the user tries to use during User Registration has already been used the operator aborts. Thus uniqueness is reduced to the security properties of the PKI.

Function Privacy. The Function Parameters input by the operator remain private: only the TSA is allowed to learn them. Other parties only learn the output of the function computed with these FPs. This property follows from the security properties of the MPC framework. In the surrounding protocol the FPs are only ever sent as commitments; the hiding property of the commitment scheme hence ensures that this leaks no information about the actual FPs.

5.8.2. User

Any *user* interacting with we expects privacy of the collected data throughout different interactions:

Unlinkability. The leakage is limited to information that enables identifying the user during the User Registration task as well as coupling a consecutive tuple of Outsource, Outsourced Analytics and Update to the same anonymous user. Other than that, it is not possible to link to executions to the same user; any two interactions are equally likely to have been made with any registered user. To achieve that goal we use the Zero Knowledge property of the NIZKPoK scheme and the hiding and rerandomization property of the commitment scheme: Any data that could be used to link a user to a previous interaction is only used as witness for ZK proofs and the operator only sees commitments thereof. As the commitments are rerandomizable they do not leak information regarding their previous use. To provide a secure channel that does not enable easy linkability we use Onion-Routing with Replies (*cf.* Section 7.3).

Input Privacy. The user does not reveal anything about the UH that cannot be derived from the result of the computation. During computation of f this automatically follows from the invocation of $\mathcal{F}_{\mathsf{MPC}}$. In the surrounding protocol users only use UH in three settings: (1) Inside commitments, where the hiding property ensures that this leaks no information. (2) As part of the witness in ZKs proofs, where input privacy follows from the Zero Knowledge property. (3) For secret sharing during the Outsource task, which is information-theoretically secure as long as the two recipients—the operator and the helper—do not collude.

Function Parameter Binding. Computations can *only* be performed on FPs which were previously certified by the TSA. It is not efficiently possible for the operator to use uncertified FPs. For using FPs which were not verified by the TSA a malicious operator would have to: (1) forge a signature σ_{fp} in the name of the TSA, which would break the unforgeability of the signature scheme, or (2) open a valid commitment $\mathtt{com}_{fp}$ to new (invalid) FPs fp' by breaking the binding property of the commitment scheme.

5.9. Realizing the Individual Tasks

Let us explain how we realize the central tasks of Bookkeeping, Outsource, Outsourced Analytics and Update.

Bookkeeping. The Bookkeeping task is almost completely covered by the authenticated input and updating mechanisms explained above. Between the two mechanisms, the user verifies that the operator uses correctly signed Function Parameters (see Section 5.2.1) for the task and both of them jointly compute the update information and additional outputs according to f.

Outsourcing Analytical Computations. As mentioned in Section 5.2.3 outsourcing an analytical computation consists of three consecutive tasks, namely the Outsource, Outsourced Analytics and Update task.

For the OS task the user opens the commitment on the linking number *lin* to show that it is zero, meaning that no other outsourced computation is in progress. Additionally, the user uses the additive robust secret sharing scheme to share the UH and the auxiliary input x_{U} which are to be used for the Outsourced Analytics task, and of several one-time pads for the helper and operator, respectively. The one-time pads hide the outputs of the computation that are relevant for the user; the function only outputs *masked* values which only the user can unmask. The secret shares are also proven to be distributed correctly. Helper and operator check the values using the robust secret sharing scheme and, using the values obtained from the helper in this process, the operator verifies the zero-knowledge proof to ensure that the shares were created correctly. Helper and operator then use the modified coin toss protocol to compute a new linking number for the OS/OA/Upd triple. Again, as the last step the operator provides the user with the necessary information to create the new logbook, the contents of which have not changed except for the serial and linking number.

As the OA task does not include the user, the general principles from Section 5.7 do not apply to this task. Instead, the helper only verifies that the operator uses correctly signed FPs for the computation. To that end the operator sends a *commitment* on the FPs and a *signature* on the commitment to the helper. The helper then verifies the signature under the verification key of the TSA. Afterwards both jointly compute f according to the MPC setup. Note that—apart from the Update Information the operator is supposed to learn—the Update Information and auxiliary outputs for the respective users are one-time padded to retain privacy from both helper and operator.

The Update task again follows the general principles: The user proves validity of the used logbook and reveals the linking number so that helper and operator can find the correct Update Information. The user then unmasks the one-time padded information from the helper and operator, respectively, checking them for consistency. The remainder of the task then consists of the user and operator conducting the logbook updating mechanism.

The linking number is set to zero again in this process so the user will be able to outsource a new analytical task.

5.10. Limitations of our scheme

In this section we discuss some of the limitations and open problems and provide ideas on how to solve them.

5.10.1. Verification by the TSA

A common problem of functions constructed using deep machine learning (such as neural networks) is a lack of transparency regarding their behavior. Our framework suffers from the same problem which even persists if we ignore function privacy for the operator; a user who has to compute a neural network on private data does not automatically know what the network computes and how the output is to be interpreted. A function that maps, say, purchases of a user to some abstract class of advertisements relevant for that user is hard to distinguish from one that maps purchases to an encoding that reflects the individual purchases on a fine-grained level.

In our setting, the problem is even harder as we additionally require function privacy for the operator; only a Trusted Signing Authority is there to ensure that the operator only uses valid Function Parameters which provide a sufficient level of privacy for the user. The TSA has the same problems mentioned above: While it is straightforward to check whether a given machine learning model indeed classifies as specified for randomly chosen inputs, a sufficiently complex model can be used to hide *backdoors* [51] in the form of special inputs provided by the operator which would break unlinkability and input privacy for any user. As it is highly unlikely that the TSA finds this backdoor by using random testing, the model behaves normally for all inputs chosen by the TSA with high probability and could even get certified. While we generally assume that the output of the function is a discrete set of much smaller size than the input space—as is the case for both applications we propose—we do not restrict to only those functions; using arbitrary output for the operator requires special attention during the verification step.

While it is possible to implant a backdoor into the model given a sufficiently large output space and a sufficiently complex model we stress that there are several different ways to detect—and even to remove, although at the cost of overall accuracy—a backdoor. A survey on the scenario itself alongside mechanisms to detect and remove a backdoor is given in [72].

Our generality lets the operator create a model with a backdoor and submit it for the Sign Function Parameter task, yet increasing progress in the field of backdoor detection [72, 134] given only the final model makes it unlikely that these Function Parameters will get a certificate. We hence require the TSA to perform a number of such tests in order for verification mechanism to be sufficiently daunting for an operator that tries to use backdoors.

5.10.2. The Trouble of Aborts

Aborts are a common problem in MPC: If a party looses connection during a computation or refuses to answer entirely then the computation cannot be finished. This is also modeled into most security frameworks. For example, we use the the asynchronous UC model. There, the entire communication is managed by the adversary; parties can ask the adversary to transfer a given message to an other party but the adversary is free to change any part of the message, to drop the message entirely or to report messages in the name of any honest party. Using authenticated channels removes the adversaries capability to *change* the message or to *create* new messages in the name of honest parties, and secure channels additionally take the adversaries ability to *read* the message. Yet even with these precautions the adversary is still able to *drop* messages at will.

For normal computations an abort only means that the parties do not get any output. But in our setting this means that the user is at worst left with no valid logbook if the abort happened after the old logbook has been invalidated at the start of a task but before the new one has been created and sent to the user. An additional case to consider is if the abort happens during an Outsourced Analytics task. This leaves the user incapable of ever outsourcing data again as the linking number will never be reset.

Yet we stress that dealing with aborts is straightforward and could easily be incorporated into the protocol, albeit at the cost of a longer functionality and protocol description and a much more complicated security proof. But for completeness reasons we sketch here how the protocol can be secured against aborts.

In total there are four tasks where the user is directly involved and where aborting in between means that there is no logbook that the user can use and one task where an abort implies that the user cannot outsource anymore. The four tasks where the user is directly involved in, namely User Registration, Bookkeeping, Outsource and Update, we have to ensure that the mechanism cannot be abused to let a malicious user obtain two different logbooks. Thus we require that the same messages that were sent before the abort will be sent in the next interaction again to ensure that the reconstructed logbook will end up with the same serial number. So essentially the reconstruction only finishes the previously started task using the state both parties had right before the abort.

The situation only becomes complicated if an abort occurs during an Outsourced Analytics task. Without a reconstruction mechanism the user would be unable to outsource ever again, as resetting the linking number lin to $\vec{0}$ is only possible in the Update task which requires a completed Outsourced Analytics task. Yet again we stress that a slight modification suffices to deal with this case:

If an abort occurs during an Outsourced Analytics task then this abort only matters if no output has been provided to the operator as parties generally get notified of the abort[1]. Hence the operator is aware that the computation involving data from a given linking number lin has failed. The reconstruction task basically consists of the update task but with all three manipulation vectors corresponding to $\perp$. That is, the permutation $\vec{\alpha}$ is the identity, the direct update $\vec{s}$ is $\perp$ everywhere, and the additive increment is $\vec{a} = 0$. This resets the linking number lin stored inside the users logbook to $\vec{0}$ and thus enables future Outsource tasks for that user.

Note that this reconstruction mechanism can be used to restore a broken logbook; yet until the reconstruction has been performed the user is essentially locked from any further interactions, with the exception of aborts during Outsourced Analytics where the user can still perform Bookkeeping tasks.

[1] If the abort is happening in the real world, then we can assume standard techniques such as timeouts can be used to determine that the message will likely never arrive.

6. Ideal Functionality

In this section we provide the full description of our ideal functionality $\mathcal{F}_{\mathsf{BKA}}$. We use the standard UC model [41], and assume that the simulator is activated by $\mathcal{F}_{\mathsf{BKA}}$ whenever any party provides any input.

All inputs to the functionality have the form (Task name, List of secret inputs). The task name uniquely determines the task to be executed.

The notifications $\mathcal{S}$ obtains after $\mathcal{F}_{\mathsf{BKA}}$ obtained input from any party depends on the respective party providing the input: On inputs from T, O, or H, $\mathcal{F}_{\mathsf{BKA}}$ activates $\mathcal{S}$ with input (Task name, *pid*), where *pid* is the Party Identifier of T, O or H, respectively.

For users, however, we want unlinkability in all tasks except for User Registration. During UReg the functionality explicitly leaks pid_{U} of the calling U to the adversary. For all other tasks we model unlinkability by having the functionality only revealing the *role* of a user after a call, and *not* the *pid*; for example, a user calling the OS task yields to a notification of the form (`Outsource`, User) for the adversary.

With the exception of User Registration the pid of a user is never revealed to anyone. This implies that the user can interact anonymously.

Additionally, each party treats the Subsession Identifier (ssid) of the current task in the same way it treats the Session Identifier, in that they are implicitly sent as input and cannot be changed.

The stateful functionality. Our functionality is *stateful*. After interaction with any party it updates its state. The state contains of different type of data. First of all, $\mathcal{F}_{\mathsf{BKA}}$ implicitly stores the pids of all registered users, alongside their latest UH. The latter ensures that the only way to change the contents of the UH is by using the provided tasks; it is not possible for any party to change the UH without interacting with the functionality.

For OA we only allow each user to only outsource one computation at a time. To that end, the functionality stores a boolean value for each registered user indicating whether the user has an OA computation is in progress or not. Furthermore, the functionality stores the UI during OA until the user fetches it in the Upd task.

Functionality $\mathcal{F}_{\mathsf{BKA}}$

$\mathcal{F}_{\mathsf{BKA}}$ enables secure Bookkeeping and Analytics. It is running with an operator O, a sign party T, and a number of N user U.

`Initialize`: On input (`Initialize`) by O and T, respond to other tasks and output (Ok) to O and T.

`SignFP`: On input (`SignFP`, fp, f, x_{O}) from O and (`SignFP`, x_{T}) from T, abort if fp when used with f violates privacy standards or if fp have been used with f before. Otherwise let ℓ be the number of function parameters stored for f, remember fp for its use for f and leak (f, ℓ) to the adversary. Finally, output (Ok) to both O and T.

`UserRegistration`: On input (`UserRegistration`) from U and O, abort if there is already some User History associated to the user $\mathit{pid}_{\mathsf{U}}$. Create a new User History UH as all-zero vector and associate this with $\mathit{pid}_{\mathsf{U}}$. Output (Ok) to U and O.

`Compute`: On input (`Compute`, f, x_{O}, fp) by O, abort if fp was never verified for f during `SignFP`. Otherwise, look up the index ℓ under which fp is stored for usage by f and leak ℓ to the adversary.

When additionally receiving input (`Compute`, f, x_{U}) by U, look up the latest User History UH that belongs to $\mathit{pid}_{\mathsf{U}}$. Compute $\mathsf{f}(\mathit{fp}, \mathit{UH}, x_{\mathsf{U}}, x_{\mathsf{O}})$ to obtain outputs $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{O}}, y_{\mathsf{U}})$.

Set $\mathit{UH}^{(1)} := \vec{\alpha}(\mathit{UH})$, apply $\vec{s}$ to $\mathit{UH}^{(1)}$ to get $\mathit{UH}^{(2)}$ and set $\mathit{UH}^{\text{new}} := \mathit{UH}^{(2)} + \vec{a}$ and store the new User History UH^{new}. Output $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$ to U and $(\vec{\alpha}, \vec{s}, y_{\mathsf{O}})$ to O.

`Outsource`: On input (`Outsource`, f) by O and H and (`Outsource`, f, x_{U}) by U, load the ssid ssid from the current task. If both U and O are corrupted, store $(\mathit{ssid}, \perp, \perp, \perp)$ for OA with the helper that corresponds to $\mathit{pid}_{\mathsf{H}}$ and return (Ok) to U, O, and H. Otherwise, abort if the user is marked for OA. Mark the user U for OA. Load the users current UH UH and store $(\mathit{ssid}, \mathit{pid}_{\mathsf{U}}, \mathit{UH}, x_{\mathsf{U}})$ for OA. Return (Ok) to U, O, and H.

Figure 6.1.: The first part of the basic functionality $\mathcal{F}_{\mathsf{BKA}}$.

Finally, the functionality stores all certified Function Parameters for a given function f. The only way to update this list is by using the SFP task with the Trusted Signing Authority T. This ensures that the operator can only perform computations of functions which were verified before.

Functionality $\mathcal{F}_{\mathsf{BKA}}$

$\mathcal{F}_{\mathsf{BKA}}$ enables secure Bookkeeping and Analytics. It is running with an operator O, a sign party T, and a number of N user U.

`Analytics:` On input $(\texttt{Analytics}, fp, x_{\mathsf{O}})$ by O, abort if fp was never verified for f during `SignFP`. Otherwise, look up the index ℓ under which fp has been stored for usage by f and leak ℓ to the adversary.

When additionally receiving input $(\texttt{Analytics}, \mathsf{f})$ from H, load the Z_{f} oldest entries $(ssid_\zeta, pid_\zeta, UH_\zeta, x_\zeta)$ for OA with the helper that belongs to pid_{H}.

If O is corrupted, output $\{(\zeta)\}$ for all ζ for which pid_ζ belongs to a corrupted user to the adversary, await input (UH_ζ, x_ζ) from the adversary for each of them and use that instead of the stored ones.

If H is corrupted, output $\{(\zeta)\}$ for all ζ for which pid_ζ belongs to a corrupted user to the adversary, await input (x_ζ) from the adversary for each of them and use that instead of the stored ones.

Compute $(\{(\vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta, y_\zeta)\}_{\zeta=1}^{Z}, y_{\mathsf{O}})$ as $\mathsf{f}(fp, \{(UH_\zeta, x_\zeta)\}_{\zeta=1}^{Z}, x_{\mathsf{O}})$ and store $(ssid_\zeta, \vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta, y_\zeta)$ for Upd

Leak $\{(\zeta), \vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta\}$ for each ζ for which pid_ζ belongs to a corrupted user to the adversary and output (Ok) to H and $(\{\vec{\alpha}_\zeta, \vec{s}_\zeta\}_{\zeta=1}^{Z}, y_{\mathsf{O}})$ to O.

`Update:` If U and O are corrupted, output (Ok) to all parties. Otherwise, load the UI $(ssid, \vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$ that was stored during OA and the latest UH UH for the user with pid pid_{U}.

Set $UH^{(1)} := \vec{\alpha}(UH)$, apply $\vec{s}$ to $UH^{(1)}$ to get $UH^{(2)}$ and set $UH^{\text{new}} := UH^{(2)} + \vec{a}$ and store the new User History UH^{new}.

If O is corrupted or if O and U are honest and H is corrupted, leak $ssid$ to the adversary.

Unmark the user U for OA and remove the stored UI.

Finally, output (Ok) to H, $(\vec{\alpha}, \vec{s})$ to O, and $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$ to U.

Figure 6.2.: The second part of the basic functionality $\mathcal{F}_{\mathsf{BKA}}$.

The Init-task. The initializing task has to be called before anything else. The task only contains the operator and the TSA and essentially starts the whole process. Before calling init all other calls are ignored. Once it has been called the functionality responds to the remaining tasks.

Sign Function Parameters. In this task the operator inputs Function Parameters fp which are to be used for a given function f. We assume that for any function f there is a public catalog of requirements that any suitable FPs have to fulfill. All the checks are performed inside the functionality. If the checks succeed then the functionality remembers that these FPs can be used for f. The functionality leaks the new amount of FPs it has for the function f to the adversary; we require this information for our security proof.

User Registration. With the User Registration task a user can register for participation. In this task a users pid is stored inside the functionality alongside an initial UH. This is the only way a new user can get a UH; without a valid UH stored the functionality aborts any other task that involves a user. Thus this models implicitly that the only way to get a UH is via this task. The task leaves the user with an empty UH which can be used for further participation.

Bookkeeping. For the BK task the functionality fetches the current UH. This ensures that the latest UH is used. The operator inputs FPs which define the function to be computed; the identifier of which is leaked to the adversary as this is required for simulation. The functionality aborts if the inserted FPs were never registered before.

The computation of function f yields a result $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathrm{U}}, y_{\mathrm{O}})$. Relevant for the update of the UH are only $(\vec{\alpha}, \vec{s}, \vec{a})$. The remaining outputs, y_{U} and y_{O}, are respective analytical outputs for the user and the operator. For our security proof we further require that the outputs of corrupted users are leaked to the adversary. Yet we stress that this is not in conflict with our confidentiality guarantees as we do not enforce privacy for corrupted users.

Afterwards, the User History gets updated by the functionality. The user and operator only learn which operations were applied (though the operator does not learn the incremental vector), but neither of them gets to actively change any values other than by applying these updates.

Outsource. The outsource-task requires the user U to provide the data, and the operator O and a helper H to prepare for the computation. The functionality only continues if the user has no previous outsourced computation in progress, which is represented by a boolean flag it stores for each user. If this flag is not set, meaning that the user who provided the input is not marked, then the functionality remembers its *current state* of the UH for later use in OA. To allow linkability with the next Outsourced Analytics and Update tasks, the functionality also stores the ssid of this task. Note that until the actual OA task is executed the user can still perform any number of BK tasks which change the UH, but the snapshot that will be used for the OA task is unaffected by these changes.

Otherwise, if the user has an OA in progress already at the time the OS task is called, the functionality aborts.

Outsourced Analytics. In this task the helper (on behalf of Z users) and the operator want to compute the function f. To that end, the functionality fetches the Z oldest values it stored during OS *with the same helper* H from its state to get the input from the users.

Similar to a normal computation the operator provides FPs which can only be used if they were previously certified by T during the SFP task—as otherwise they cannot be stored inside the functionality.

The functionality contains leaks in case of corruptions. For one, it is not efficiently possible for a simulator to extract the whole auxiliary input x_{U} during OS so the functionality asks for new inputs for all corrupted users. Furthermore, we stress that a corrupted operator can create arbitrary users which means that for real protocols, it might be the case that some users who called OS never registered but still have a valid UH—as the UReg task was executed with a corrupted user and operator and hence the functionality was never notified. For those, the simulator needs to equivocate the data. Yet we do not consider this to be in conflict with our security guarantees from Section 5.8 as the data of honest users is not affected by this.

The functionality performs the computation (thus ensuring correctness) and stores the outputs relevant for the user. As the user is not present at this task and the computation can output data that manipulates the users data stored in the UH, this data is stored so that the user can later fetch it during Upd.

Update. The update task applies the UI from the OA task to the users data. Note that while the computation during OA used a *snapshot* of the UH the user had while the OS task was called, the update is applied to the *latest* UH. This might differ as further BK tasks were possible between the time where the user called OS and the time that Upd is called.

The functionality thus fetches the latest UH and applies the updates from OA onto it. Furthermore, it unmarks the user so that the OS task is possible again.

7. Protocol

In this chapter we introduce our protocol. We start by describing what we mean by *benign* functions (which was briefly discussed in Section 5.5) in Section 7.1.

We then describe some helping procedures: In Section 7.2 we introduce a procedure for Robust Secret Sharing that we use for the Outsource task to let the user share the latest UH between the operator and helper, such that as long as only one of them is corrupted, the (benign) analytical function will not compute anything if the data has been manipulated. In the same section we also show how to *verify* that the data was shared correctly. This in itself does not really help, but combined with the ZK proof—where the user proves that the shares add up to valid values—this takes away the users ability to outsource invalid data.

Then in Section 7.3 we proceed to describe how the user uses the ORR setup to anonymously send a message to the operator (and the helper); this is merely a convenience as it makes reading the protocol descriptions easier.

7.1. Benign Functions

Our protocol requires certain pre- and post-processing steps before the actual MPC-protocol can be executed. These additional steps have to be fulfilled by any computable function f which we call *benign* function.

Functions can be executed during the two tasks BK and OA. In the following we will describe the necessary pre- and post-processing steps in more detail.

7.1.1. Bookkeeping

Pre-processing. In the pre-processing step the additional code verifies the integrity of the used Function Parameters before using them with f. This works by letting the user insert a *commitment* of the FPs (which the user only does if the operator has provided a valid signature that verifies under the verification key of the TSA) while the operator inserts the FPs and the unveil information directly. A similar mechanism is used for the verification of the user inputs: The user inserts the latest UH unveil information of the latest commitment, while the commitment itself is inserted by O. The function then only continues if both commitments verify, meaning that the check is performed by $\mathcal{F}_{\mathsf{MPC}}$.

Post-processing. Instead of providing outputs directly (namely the updated UH and the individual outputs for both parties) the post-processing step separates the updates into *permutations* $\vec{\alpha}$, *direct updates* $\vec{s}$ and *additive updates* $\vec{a}$. Furthermore, f computes a *commitment* $(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}})$ on $\vec{a}$ and outputs $\mathtt{com}_{\vec{a}}$ to the operator and $(\vec{a}, \mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}})$ to the user, whereas $\vec{\alpha}$ and $\vec{s}$ are output to both parties.

7.1.2. Outsourced Analytics

Pre-processing. In the pre-processing phase the FPs provided by the operator are verified in the same way as in the BK task. Then the function f proceeds to reconstruct the robust shares for each user by using the method described in Fig. 7.2 to extract the actual inputs of each user.

Post-processing. Post-processing starts similar to the BK task by providing updates as permutation $\vec{\alpha}$, direct update $\vec{s}$, and additive update $\vec{a}$. The first two are output to the operator, but the third one should not be learned by either of the operator or helper.

To hide these (and other sensitive information that should not be seen by the helper) the user provided as additional input a OTP $otp = otp_{\vec{\alpha}} \| otp_{\vec{a}} \| otp_y \| otp_{\mathrm{unv}}$. The function uses the respective bits of the OTP to *mask* the user outputs that are given to the helper. As $\mathcal{F}_{\mathsf{MPC}}$ leaks no intermediate results this means that the output seen by the helper only contains *cryptographically protected* outputs for each participating user. Note that in addition to the add-vector, the user also learns the *unveil information* of the commitment given to the operator; this is an inverse check to the pre-processing step to ensure that the values given to the user are correct. As the commitment is only given to the operator and the unveil information is later obtained (in an encrypted form) by the helper, the user can verify that the increment was not changed by either of the two parties.

7.2. Robust Secret Sharing

Both our protocol and the benign functions make use of a *sharing* protocol and its corresponding *combine* protocol. Those are required so that the user can *share* information during the OS task with H and O in such a way, that no party—neither H nor O—can change the shares unnoticed. The protocols are shown in Fig. 7.1.

Essentially, the sharing part comes down to *additive secret sharing*. The main difference is that the dealer does not only send the additive share of a value to each party, but also adds a *commitment* on the respective *other* parties share. Additionally, each party obtains unveil information on its own commitment.

```
**Procedure** $\texttt{Share}(msg)$
Subprocedure $\texttt{Share}$ for the protocol $\Pi_{BKA}$. It holds that $msg \in \{0,1\}^m$.
.............................................................................
Sample $msg^{(0)} \xleftarrow{\$} \{0,1\}^m$.
Set $msg^{(1)} := msg - msg^{(0)}$.
$(\texttt{com}_{msg^{(0)}}, \texttt{unv}_{msg^{(0)}}) \leftarrow \textsc{Com}.\mathrm{Com}(msg^{(0)})$.
$(\texttt{com}_{msg^{(1)}}, \texttt{unv}_{msg^{(1)}}) \leftarrow \textsc{Com}.\mathrm{Com}(msg^{(1)})$.
$rs_{msg}^{(0)} := (msg^{(0)}, \texttt{com}_{msg^{(1)}}, \texttt{unv}_{msg^{(0)}})$.
$rs_{msg}^{(1)} := (msg^{(1)}, \texttt{com}_{msg^{(0)}}, \texttt{unv}_{msg^{(1)}})$.
**return** $(rs_{msg}^{(0)}, rs_{msg}^{(1)})$.
```

Figure 7.1.: The procedure $\texttt{Share}$ for the protocol Π_{BKA}.

```
**Procedure** $\texttt{Combine}(rs_{msg}^{(0)}, rs_{msg}^{(1)})$
Subprocedure $\texttt{Combine}$ for the protocol $\Pi_{BKA}$. It holds that $rs_{msg}^{(0)}$ and $rs_{msg}^{(1)}$ are robust shares.
.............................................................................
Parse $rs_{msg}^{(0)}$ as $(msg^{(0)}, \texttt{com}_{msg^{(1)}}, \texttt{unv}_{msg^{(0)}})$.
Parse $rs_{msg}^{(1)}$ as $(msg^{(1)}, \texttt{com}_{msg^{(0)}}, \texttt{unv}_{msg^{(1)}})$.
Abort if $\textsc{Com}.\mathrm{Unv}(\texttt{com}_{msg^{(0)}}, \texttt{unv}_{msg^{(0)}}, msg^{(0)}) = 0$.
Abort if $\textsc{Com}.\mathrm{Unv}(\texttt{com}_{msg^{(1)}}, \texttt{unv}_{msg^{(1)}}, msg^{(1)}) = 0$.
$msg := msg^{(0)} + msg^{(1)}$.
**return** $msg$.
```

Figure 7.2.: The procedure $\texttt{Combine}$ for the protocol Π_{BKA}.

The combine protocol from Fig. 7.2 is only executed in $\mathcal{F}_{\mathsf{MPC}}$ as part of the computation for f. It reconstructs the additive secret shares, but *only* if the commitments are valid. As those are cross-entered (the commitment on the share for the helper is entered by the operator and vice versa) this ensures that both parties indeed use the correct shares.

The verify procedure from Fig. 7.3 lets both recipients verify that the data was shared correctly, *without* requiring access to the original data. As each robust share has a *commitment* on the other parties share attached this protocol requires both parties to *exchange* their commitments and then locally *verify* that they were correct. This check is to ensure that the user provided correct *commitments*; combined with the ZK proof that shows that the shares belong to a valid UH (which requires commitments of all shares) this step provides security against a potentially lying user, whereas the check from Fig. 7.2 inside f provides security against malicious helper/operator who try to insert wrong data.

Procedure $\texttt{Verify}(rs_{UH}^{(\mathsf{P})}, \mathtt{com}_{UH}^{(\overline{\mathsf{P}})}, rs_{x_\mathsf{U}}^{(\mathsf{P})}, \mathtt{com}_{x_\mathsf{U}^{(\overline{\mathsf{P}})}}, rs_{otp}^{(\mathsf{P})}, \mathtt{com}_{otp})$

Subprocedure $\texttt{Verify}$ for the protocol Π_{BKA}. It holds that all the entries $rs_{\cdot}^{(\mathsf{P})}$ are robust shares and $\mathsf{P}, \overline{\mathsf{P}} \in \{\mathsf{O}, \mathsf{H}\}$ such that $\mathsf{P} \neq \overline{\mathsf{P}}$.
The code is executed by a helper H or the operator O.

```
if $\textsc{Com}.\mathrm{Unv}(UH, \mathtt{com}_{UH}, \mathtt{unv}_{UH}) = 0 \lor$
   $\textsc{Com}.\mathrm{Unv}(x_\mathsf{U}, \mathtt{com}_{x_\mathsf{U}}, \mathtt{unv}_{x_\mathsf{U}}) = 0 \lor$
   $\textsc{Com}.\mathrm{Unv}(otp, \mathtt{com}_{otp}, \mathtt{unv}_{otp}) = 0$ then
   return 0
else
   return 1
fi
```

Figure 7.3.: The procedure $\texttt{Verify}$ for the protocol Π_{BKA}.

Procedure $\texttt{Send}(msg, \mathsf{R})$

Subprocedure $\texttt{Send}$ for the protocol Π_{BKA}. It is running with a set of parties $(\mathsf{U}_0, \ldots, \mathsf{U}_{N_\mathsf{U}-1}, \mathsf{H}_0, \ldots, \mathsf{H}_{N_\mathsf{H}-1}, \mathsf{M}_0, \ldots, \mathsf{M}_{N_\mathsf{M}-1}, \mathsf{O})$. It holds that $msg \in \{0,1\}^m$ and $\mathsf{R} \in \{\mathsf{O}, \mathsf{H}_0, \ldots, \mathsf{H}_{N_\mathsf{H}-1}\}$.
It is running in the $\mathcal{F}_{\mathsf{ORR}}$-hybrid model.
The code is executed by a user U_i.

```
Sample $P^{\rightarrow} \xleftarrow{\$} \{\mathsf{M}_0, \ldots, \mathsf{M}_{N_\mathsf{M}-1}\}^{\Psi}$
Sample $P^{\leftarrow} \xleftarrow{\$} \{\mathsf{M}_0, \ldots, \mathsf{M}_{N_\mathsf{M}-1}\}^{\Psi}$
Send $(\texttt{Process\_Onion}, \mathsf{R}, msg, P^{\rightarrow}, P^{\leftarrow})$ to $\mathcal{F}_{\mathsf{ORR}}$.
```

Figure 7.4.: The procedure $\texttt{Send}$ for the protocol Π_{BKA}.

7.3. Anonymous Communication

In Fig. 7.4 we introduce a procedure that is used by the sender to send a message to a dedicated receiver. We extracted it to its own procedure in order to make the overall protocol more readable; whenever the user sends a message to either the helper or operator, the user has to sample both a *forward path* $P^{\rightarrow}$ and a *backward path* $P^{\leftarrow}$ of mix-servers, and then send input to $\mathcal{F}_{\mathsf{ORR}}$.

Note that $\mathcal{F}_{\mathsf{ORR}}$ hides the senders identity (among the group of potential senders) so it is trivially required that the first message is always sent by the *user*. Alongside the message, the receiver obtains a dedicated *return information* ϑ. This is require to answer to that message.

Protocol Π_{BKA}

Protocol Π_{BKA} in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{SMT}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{ORR}})$ hybrid model.

Each user U stores:

- User logbook λ.
- Key pair $(\mathsf{sk}_{\mathsf{U}}, \mathsf{pk}_{\mathsf{U}})$.
- Verification keys $(\mathsf{vk}_{\mathsf{O}}, \mathsf{vk}_{\mathsf{T}})$ of the operator and the sign party.

The operator O stores:

- Signature key pair $(\mathsf{vk}_{\mathsf{O}}, \mathsf{sk}_{\mathsf{O}})$.
- List of known serial numbers ser.
- List of known user public keys pk_{U}.
- List of tuples $(\mathsf{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$.
- List of tuples $(\mathsf{H}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})})$ for OA.
- List of tuples $(lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_{\mathsf{U}}})$ for Upd.

The Helper H stores:

- List of tuples $(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$ for OA.
- List of tuples $(lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}})$ for Upd.

The Sign Party T stores:

- Signature key pair $(\mathsf{vk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}})$.
- List of tuples (fp, f).

$(\mathtt{Initialize})$ On input $(\mathtt{Initialize})$, O computes a signature key pair $(\mathsf{vk}_{\mathsf{O}}, \mathsf{sk}_{\mathsf{O}}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$, sends $(\mathtt{Register}, \mathsf{vk}_{\mathsf{O}})$ to $\mathcal{F}_{\mathsf{Reg}}$, and outputs $(\mathtt{Ok})$.

On input $(\mathtt{Initialize})$, T computes a signature key pair $(\mathsf{vk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$, sends $(\mathtt{Register}, \mathsf{vk}_{\mathsf{T}})$ to $\mathcal{F}_{\mathsf{Reg}}$ and outputs $(\mathtt{Ok})$.

Figure 7.5.: The first part of the protocol Π_{BKA} that specifies the parties state and the initialization part of the protocol.

7.4. Our Protocol

In this section we introduce our protocol. Since the protocol itself is quite long we introduce each task individually.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the task `SignFP` to sign the function parameters.

On input $(\texttt{SignFP}, fp, \mathsf{f}, x_{\mathsf{O}})$, O computes commitments $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}) \leftarrow \textsc{Com}.\mathrm{Com}(fp)$ and sends $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \mathsf{f}, x_{\mathsf{O}})$ to T.

On input $(\texttt{SignFP}, x_{\mathsf{T}})$ and after receiving the message from O, T aborts if fp does not comply with the privacy-requirements of f, if the same fp were used previously for f or if $\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) = 0$. Otherwise, T computes a signature $\sigma_{fp} \leftarrow \textsc{Sig}.\mathrm{Sign}_{(\mathsf{k}_{\mathsf{T}})}(\mathsf{f}, \mathtt{com}_{fp})$, sends σ_{fp} to O and outputs $(\mathtt{Ok})$.

When receiving (σ_{fp}) from T, O stores a new tuple $(\mathsf{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ and outputs $(\mathtt{Ok})$.

Figure 7.6.: The second part of the protocol Π_{BKA} that specifies the behavior for signing function parameter.

Init. Before any other task can be executed the operator and the TSA have to run the Init task. Even though this is modeled as two-party task we stress that the two parties never actually interact with each other. They only individually create their signing keys and register them at $\mathcal{F}_{\mathrm{Reg}}$.

Sign Function Parameter. As described in Section 5.2.1 this task is intended for the TSA to *certify* the FPs that an operator wants to use, to ensure that these fulfill a given set of privacy requirement. The protocol for the Sign Function Parameter task lets the operator input some FPs which are to be used for computation of some function f which is used either in the BK or the OA task. The TSA verifies the FPs to ensure that they match the required privacy standards. We assume those privacy standards to be public knowledge.

If a given set of FPs verifies the operator obtains a signature that verifies the commitment $\mathtt{com}_{fp}$ of the FPs and the function f where it can be used under the verification key of the TSA. That way, the signature legitimates the use of the FPs which were committed to. The unforgeability of the signature scheme ensures that the operator can only use FPs for which the unveil information to the commitment are known, and the binding property of the commitment scheme ensures that the only unveil information the operator is able to find open the commitment to the original FPs that were verified by the TSA.

User Registration. The protocol for User Registration is shown in Fig. 7.7. One of our requirements is that each user has at most a single logbook. We use a special *PKI* infrastructure in $\mathcal{F}_{\mathrm{Reg}}$ to ensure that no user can register twice: $\mathcal{F}_{\mathrm{Reg}}$ lets the user register a

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the task `UserRegistration`.

On input (`UserRegistration`), U sends (`Fetch`, pid_{O}) and (`Fetch`, pid_{T}) to $\mathcal{F}_{\mathrm{Reg}}$ to obtain and store vk_{O} and vk_{T}. Then the user picks a uniformly random secret key $\mathsf{sk}_{\mathsf{U}} \xleftarrow{\$} \mathbb{Z}_o$, computes the public keys as $\mathsf{pk}_{\mathsf{U}} := g_1^{\mathsf{sk}_{\mathsf{U}}}$ in $\mathbb{G}_1$ and $\mathsf{pk}^*_{\mathsf{U}} := g_2^{\mathsf{sk}_{\mathsf{U}}}$ in $\mathbb{G}_2$, and sends (`Register`, pk_{U}) to $\mathcal{F}_{\mathrm{Reg}}$.

Then U computes $(\mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{unv}_{\mathsf{sk}_{\mathsf{U}}}) \leftarrow \mathrm{Com.Com}(\mathsf{sk}_{\mathsf{U}})$, sets $stmt := (\mathsf{pk}_{\mathsf{U}}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{e}, pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t})$ and $wit := (\mathsf{sk}_{\mathsf{U}}, \mathsf{unv}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{pk}^*_{\mathsf{U}})$ and computes $\pi \leftarrow \mathrm{Zk.Proof}(stmt, wit, \Lambda_{\mathrm{UserReg}})$ as in Fig. 7.8. The user then draws a random share $ser_0 \xleftarrow{\$} \mathbb{Z}_o$, computes $(\mathsf{com}_{ser_0}, \mathsf{unv}_{ser_0}) \leftarrow \mathrm{Com.Com}(ser_0)$ and sends $(\pi, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{com}_{ser_0})$ to O.

On input (`UserRegistration`) and after receiving the first message from U, O sends (`Fetch`, pid_{U}) to $\mathcal{F}_{\mathrm{Reg}}$ to obtain pk_{U} and aborts if pk_{U} is used already. O stores pk_{U} and sets $stmt := (\mathsf{pk}_{\mathsf{U}}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}})$ and aborts if $\mathrm{Zk.Verify}(\pi, stmt, \Lambda_{\mathrm{UserReg}}) = 0$. Otherwise, O creates an empty User History UH and computes the commitments $(\mathsf{com}_{UH}, \mathsf{unv}_{UH}) \leftarrow \mathrm{Com.Com}(UH)$ and $(\mathsf{com}_{lin}, \mathsf{unv}_{lin}) \leftarrow \mathrm{Com.Com}(0)$. Then O picks a random $ser_1 \xleftarrow{\$} \mathbb{Z}_o$ and computes $(\mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}) \leftarrow \mathrm{Com.Com}(ser_1)$ and the commitment on the final nonce as $\mathsf{com}_{ser} := \mathrm{Com.CAdd}(\mathsf{com}_{ser_0}, \mathsf{com}_{ser_1})$. Finally, O computes the signature $\sigma_{UH} \leftarrow \mathrm{Sig.Sign}_{(\mathsf{k}_{\mathsf{O}})}(\mathsf{com}_{UH}, \mathsf{com}_{ser}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}})$, sends $(ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \mathsf{com}_{UH}, \mathsf{unv}_{UH}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \sigma_{UH})$ to U and outputs (`Ok`).

When receiving the message from O, U computes the serial as $ser_0 + ser_1 \bmod o$ and adjusts $\mathsf{com}_{ser} := \mathrm{Com.CAdd}(\mathsf{com}_{ser_0}, \mathsf{com}_{ser_1})$ and $\mathsf{unv}_{ser} := \mathrm{Com.UAdd}(\mathsf{unv}_{ser_0}, \mathsf{unv}_{ser_1})$. Then U aborts if the signature or the commitment is invalid. Otherwise, it stores $\lambda := (UH, \mathsf{com}_{UH}, \mathsf{unv}_{UH}, ser, \mathsf{com}_{ser}, \mathsf{unv}_{ser}, \mathsf{sk}_{\mathsf{U}}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH})$ and outputs (`Ok`).

Figure 7.7.: The third part of the protocol Π_{BKA} that specifies the behavior for user registration.

Language $\Lambda_{\mathrm{UserReg}}(\mathsf{pk}_{\mathsf{U}}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{e}, pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t})$

$\mathsf{sk}_{\mathsf{U}}, \mathsf{unv}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{pk}^*_{\mathsf{U}}:$

$\mathsf{e}(\mathsf{pk}_{\mathsf{U}}, g_2) = \mathsf{e}(g_1, \mathsf{pk}^*_{\mathsf{U}}) \wedge$

$g_1^{\mathsf{sk}_{\mathsf{U}}} = \mathsf{pk}_{\mathsf{U}} \wedge$

$\mathrm{Com.Unv}\big(\mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{unv}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{sk}_{\mathsf{U}} = 1\big)$

Figure 7.8.: Language $\Lambda_{\mathrm{UserReg}}$ used for the `UserRegistration` task.

key *exactly once*. Our protocol requires the user to publish a *fresh* public key to the PKI. To that end we use a naive identification protocol based on a *pairing group* $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$. The user broadcasts a public key using $\mathcal{F}_{\mathsf{Reg}}$ but has to prove knowledge of the corresponding *secret key*. To that end the user uses the ZK proof from Fig. 7.8.

This proves knowledge of a *different* key $\mathsf{pk}_{\mathsf{U}}^{*}$ which contains the same secret as the original public key but has the secret applied to g_2. In the same proof, the user proves that the commitment on the secret key is indeed on the same secret key that was used for the creation of the public key. The operator then only accepts if this proof is valid and if the public key has not been used before. If this succeeds then the user gets an empty; by empty we mean that the initial UH is zero on each spot. While there might be applications where the user does *not* start with an all-empty logbook, a successive BK task can be used to set up the correct initial values; yet since all the applications we considered (not only here, but also in the full version [70]) started out with an empty logbook we decided in favor of using an empty logbook here.

With this empty UH the two parties create the initial logbook together. The principle is fairly similar in this task to all the following tasks: To ensure history freshness the user has to carry a serial number which is invalidated during the next interaction with the operator. As the serial number would allow tracking of the same user through different tasks we decided to take inspiration from the Blum coin flipping protocol [28] over $\mathbb{Z}_o$ where only the user learns the outcome of the coin toss and the operator only learns that the outcome is randomly distributed over $\mathbb{Z}_o$. This is to ensure that no party picks a malicious serial number such as one that contains tracking information. The homomorphism of the commitment scheme enables the operator to create a commitment on the *actual* serial number based only on the commitment of the users share and the commitment on the own share.

The linking number is initially 0 so the operator creates valid zero-commitments and sends them alongside the unveil information to the user. With all this information at hand, the operator computes the initial *signature* of the logbook. This is created by signing the commitments on the UH and the linking number (both of which are zero-commitments created by the operator), the commitment on the serial number (which was homomorphically computed by the operator), and the commitment on the users public key (which was created and sent by the user). The operator then sends all the information to the user and the user then verifies the data and stores the initial logbook.

Note that all the communication happening here is *identifying* on *both* sides.

Bookkeeping. The protocol for the BK task that directly updates the UH is given in Figs. 7.9 to 7.11.

Essentially the protocol serves as a wrapper around $\mathcal{F}_{\mathsf{MPC}}$: It starts by letting the user prove to the operator that the latest input is used and the operator proving to the user that it will input valid FPs that were signed by the TSA. For the former we provide a ZK

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the first part of $\texttt{Compute}$ for a function f.

On input $(\texttt{Compute}, \mathsf{f}, x_{\mathsf{U}})$, U computes commitments $(\mathtt{com}_{UH}, \mathtt{unv}_{UH}) \leftarrow \textsc{Com}.\mathsf{Com}(UH)$, $(\mathtt{com}_{lin}, \mathtt{unv}_{lin}) \leftarrow \textsc{Com}.\mathsf{Com}(lin)$ and $(\mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}) \leftarrow \textsc{Com}.\mathsf{Com}(\mathsf{sk}_{\mathsf{U}})$ and loads the corresponding old commitments $(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*})$, $(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*})$ and $(\mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}})$ alongside the old serial ser^* and its commitments $(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*})$ from the logbook λ. Now U proves correct rerandomization according to Fig. 7.12 by setting $stmt := (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, ser^*, \mathtt{com}_{ser}, \mathsf{vk}_{\mathsf{O}})$ and $wit := (UH, \mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{unv}_{ser}, lin, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, \mathtt{unv}_{lin}, \mathsf{sk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH^*})$ and computing $\pi \leftarrow \textsc{Zk}.\mathsf{Proof}(stmt, wit, \Lambda_{\mathsf{Compute}})$. Finally, U follows $\texttt{Send}(\mathsf{O}, (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, ser^*, \mathtt{com}_{ser}, \pi))$ from Fig. 7.4.

On input $(\texttt{Compute}, \mathsf{f}, x_{\mathsf{O}}, fp)$ and after receiving the message and return information from $\mathcal{F}_{\mathsf{ORR}}$, O looks up a stored tuple $(\mathsf{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ and aborts if there is either no such tuple or more than one for the given (f, fp) pair.

O aborts if ser^* has been used before and otherwise stores ser^*. Then it sets $stmt := (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, ser^*, \mathsf{vk}_{\mathsf{O}})$ and aborts if $\textsc{Zk}.\mathsf{Verify}(\pi, stmt, \Lambda_{\mathsf{Compute}})$ fails.

Otherwise, O inputs $(\texttt{Compute}, \mathsf{f}, (fp, \mathtt{com}_{UH}, \mathtt{unv}_{fp}, x_{\mathsf{O}}))$ into $\mathcal{F}_{\mathsf{MPC}}$, and $(\texttt{Process\_Back\_Onion}, (\mathtt{com}_{fp}, \sigma_{fp}), \vartheta)$ to $\mathcal{F}_{\mathsf{ORR}}$.

Continued on Fig. 7.10.

Figure 7.9.: The fifth part of the protocol Π_{BKA} with the first part on how to compute a function f.

language in Fig. 7.12. The user computes new commitments for the User History, the serial number, the linking number, and the secret key, and proves that the rerandomizations were performed correctly—namely that they contain the same values as the commitments that were signed by the operator during the last interaction. For the latter—namely letting the operator prove that the FPs used for the computation are validly signed—we use an interactive protocol: The operator sends the commitment and the signature to the user. This can be done directly as we do not hide from the user *which* FPs will be used—the user learns whether the same FPs have been used in previous computations—but only what those FPs *are*. Hence we do not require any form of rerandomization or ZK here. Instead, the operator sends the commitment on the FPs alongside the signature on the commitment and the function directly to the user. The user then verifies the signature using the verification key of the TSA and only continues if this signature is valid. If the commitment is valid the user sends input to $\mathcal{F}_{\mathsf{MPC}}$. This input contains the commitment on the FPs.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the second part of $\texttt{Compute}$ for a function f.

Continuation of Fig. 7.9.

When receiving the message from $\mathcal{F}_{\mathsf{ORR}}$, U aborts if $\textsc{Sig}.\mathsf{Vfy}_{\mathsf{vk_T}}(\sigma_{fp}, \mathtt{com}_{fp})$ fails. Otherwise, U picks a random $ser_0 \xleftarrow{\$} \mathbb{Z}_o$ as serial share and computes $(\mathtt{com}_{ser_0}, \mathtt{unv}_{ser_0}) \leftarrow \textsc{Com}.\mathsf{Com}(ser_0)$.

Finally, U sends $(\texttt{Compute}, \mathsf{f}, (UH, \mathtt{com}_{fp}, \mathtt{unv}_{UH}, x_{\mathsf{U}}))$ to $\mathcal{F}_{\mathsf{MPC}}$ and follows $\texttt{Send}(\mathsf{O}, (\mathtt{com}_{ser_0}))$ from Fig. 7.4.

When receiving $(\texttt{Output}, (\vec{\alpha}, \vec{s}, \vec{a}, \mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, y_{\mathsf{U}}))$ from $\mathcal{F}_{\mathsf{MPC}}$, U acts as follows: If $\vec{\alpha}$ and $\vec{s}$ are not empty, U sets $UH^{(1)} := \vec{\alpha}(UH)$, computes $UH^{(2)}$ by replacing entries from $UH^{(1)}$ with values from $\vec{s}$ wherever there are non-empty values in $\vec{s}$, and sets $UH^{\mathrm{new}} := UH^{(2)} + \vec{a}$. Then U computes the corresponding commitments $(\mathtt{com}_{UH^{(1)}}, \mathtt{unv}_{UH^{(1)}}) \leftarrow \textsc{Com}.\mathsf{Com}(UH^{(1)})$ and $(\mathtt{com}_{UH^{(2)}}, \mathtt{unv}_{UH^{(2)}}) \leftarrow \textsc{Com}.\mathsf{Com}(UH^{(2)})$.

Now U proves correct transfer of $\vec{\alpha}$ and $\vec{s}$ by setting $stmt^* := (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and $wit^* := (UH, \mathtt{unv}_{UH}, UH^{(1)}, \mathtt{unv}_{UH^{(1)}}, UH^{(2)}, \mathtt{unv}_{UH^{(2)}})$ and computing $\pi^* := \textsc{Zk}.\mathsf{Proof}(stmt^*, wit^*, \Lambda_{\mathrm{Transfer}})$ from Fig. 7.13. Otherwise, if $\vec{\alpha}$ and $\vec{s}$ are empty, set $UH^{\mathrm{new}} := UH + \vec{a}$, $\mathtt{com}_{UH^{(1)}} = \mathtt{com}_{UH}$, $\mathtt{com}_{UH^{(2)}} = \mathtt{com}_{UH}$ and $\pi^* = \bot$.

Finally, U computes $\mathtt{com}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathsf{CAdd}(\mathtt{com}_{UH^{(2)}}, \mathtt{com}_{\vec{a}})$ and $\mathtt{unv}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathsf{UAdd}(\mathtt{unv}_{UH^{(2)}}, \mathtt{unv}_{\vec{a}})$ and follows $\texttt{Send}(\mathsf{O}, (\mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \pi^*))$ from Fig. 7.4.

Continued on Fig. 7.11

Figure 7.10.: The sixth part of the protocol Π_{BKA} that specifies how to compute a function f.

The operator inputs the corresponding opening information and the clear values. Since f is a *benign* function (*cf.* Section 7.1), computation only happens if the operator's unveil information successfully opens the user's commitment to the FPs inserted by the operator. As long as the signature scheme is unforgeable and the commitment scheme is binding this ensures that no malicious operator can input uncertified Function Parameters; this would either require a forged signature on a new commitment created without knowing the signing key of the TSA or different opening information for the existing commitment.

After receiving output from $\mathcal{F}_{\mathsf{MPC}}$ the user updates the UH. We already described the three-stage update mechanism of the UH in Section 5.2.2. The user obtains the UI consisting of $(\vec{\alpha}, \vec{s}, \vec{a})$, the operator only $\vec{\alpha}$ and $\vec{s}$. Unless they are empty, the first two maps, $\vec{\alpha}$ and $\vec{s}$, are applied directly to the elements in the User History by the user. We refer to $UH^{(1)}$ as the

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the third part of `Compute` for a function f.

Continuation of Fig. 7.10.

When receiving $(\texttt{Output}, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, y_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{MPC}}$ and both messages from $\mathcal{F}_{\mathsf{ORR}}$, O acts as follows: If $\vec{\alpha}$ and $\vec{s}$ are non-empty, set $stmt^* := (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and abort if $\textsc{Zk}.\mathrm{Verify}(\pi^*, stmt^*, \Lambda_{\mathrm{Transfer}}) \neq 1$.

Regardless of $\vec{\alpha}$ or $\vec{s}$, O sets $\mathtt{com}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_{UH^{(2)}}, \mathtt{com}_{\vec{a}})$, picks a random ser_1 and computes $(\mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}) \leftarrow \textsc{Com}.\mathrm{Com}(ser_1)$. Then O computes the commitment on the serial $\mathtt{com}_{ser} := \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_{ser_0}, \mathtt{com}_{ser_1})$ and creates the signature $\sigma_{UH^{\mathrm{new}}} \leftarrow \textsc{Sig}.\mathrm{Sign}_{(\mathsf{k_O})}(\mathtt{com}_{UH^{\mathrm{new}}}, \mathtt{com}_{lin}, \mathtt{com}_{ser}, \mathtt{com}_{\mathsf{sk_U}})$. Finally, O sends $(\texttt{Process\_Back\_Onion}, (ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \sigma_{UH^{\mathrm{new}}}), \vartheta)$ to $\mathcal{F}_{\mathsf{ORR}}$ and outputs (y_{O}).

When receiving the message from $\mathcal{F}_{\mathsf{ORR}}$, U sets $ser := ser_0 + ser_1 \mod o$ and homomorphically updates $\mathtt{com}_{ser} := \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_{ser_0}, \mathtt{com}_{ser_1})$ and $\mathtt{unv}_{ser} := \textsc{Com}.\mathrm{UAdd}(\mathtt{unv}_{ser_0}, \mathtt{unv}_{ser_1})$.

Finally, U aborts if $\sigma_{UH^{\mathrm{new}}}$ fails to verify and otherwise stores $\lambda := (UH^{\mathrm{new}}, \mathtt{com}_{UH^{\mathrm{new}}}, \mathtt{unv}_{UH^{\mathrm{new}}}, ser, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, \mathsf{sk_U}, \mathtt{com}_{\mathsf{sk_U}}, \mathtt{unv}_{\mathsf{sk_U}}, \sigma_{UH^{\mathrm{new}}})$ and outputs $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$.

Figure 7.11.: The seventh part of the protocol Π_{BKA} that specifies how to compute a function f.

Language $\Lambda_{\mathrm{Compute}}(\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{\mathsf{sk_U}}, ser, \mathtt{com}_{ser}, \mathsf{vk_O})$

$UH, \mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{unv}_{ser}, lin, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, \mathtt{unv}_{lin}, \mathsf{sk_U}, \mathtt{com}_{\mathsf{sk_U^*}}, \mathtt{unv}_{\mathsf{sk_U^*}}, \mathtt{unv}_{\mathsf{sk_U}}, \sigma_{UH}:$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH}, \mathtt{unv}_{UH}, UH) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, UH) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, lin) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin}, \mathtt{unv}_{lin}, lin) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, ser) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{ser}, \mathtt{unv}_{ser}, ser) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk_U}}, \mathtt{unv}_{\mathsf{sk_U}}, \mathsf{sk_U}) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk_U^*}}, \mathtt{unv}_{\mathsf{sk_U^*}}, \mathsf{sk_U}) = 1 \wedge$

$\textsc{Sig}.\mathrm{Vfy}(\sigma_{UH}, (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{ser}, \mathtt{com}_{\mathsf{sk_U}}), \mathsf{vk_O}) = 1$

Figure 7.12.: Language $\Lambda_{\mathrm{Compute}}$ used for the `Compute` task.

Language $\Lambda_{\text{Transfer}}(\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$

$$
\begin{aligned}
& UH, \mathtt{unv}_{UH}, UH^{(1)}, \mathtt{unv}_{UH^{(1)}}, UH^{(2)}, \mathtt{unv}_{UH^{(2)}} : \\
& \textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH}, \mathtt{unv}_{UH}, UH) = 1 \wedge \\
& \textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH^{(1)}}, \mathtt{unv}_{UH^{(1)}}, UH^{(1)}) = 1 \wedge \\
& \textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH^{(2)}}, \mathtt{unv}_{UH^{(2)}}, UH^{(2)}) = 1 \wedge \\
& UH^{(1)} = \vec{\alpha}(UH) \wedge \\
& \forall_{i=1}^{|UH|}((\vec{s}[i] = \bot) \implies (UH^{(2)}[i] = UH^{(1)}[i])) \wedge \\
& \forall_{i=1}^{|UH|}((\vec{s}[i] \neq \bot) \implies (UH^{(2)}[i] = \vec{s}[i]))
\end{aligned}
$$

Figure 7.13.: Language $\Lambda_{\text{Transfer}}$ used for the Bookkeeping and Update task.

UH created by applying the permutation $\vec{\alpha}$ to the old UH and by $UH^{(2)}$ as the UH created by updating the values from $UH^{(1)}$ as stated in $\vec{s}$. The user then proves to the operator that the updates were applied correctly. This is what the language from Fig. 7.13 does. The operator never learns the actual contents of the intermediate UHs but only *commitments* thereof; the ZK proof then ensures that the commitments were created correctly.

The user sends both proofs alongside the commitments on the new UH to the operator. Note that if both the permutation and the update vector are trivial[1] then the new UH corresponds to the old one and the proof of correct transfer is trivial and hence left empty.

The operator verifies both proofs (if necessary) and then homomorphically computes the commitment of the final UH using the commitment on the permuted and updated UH from the user and the commitment on the addition vector $\vec{a}$ obtained from $\mathcal{F}_{\text{MPC}}$. The same technique is used to update the serial number homomorphically. The commitments are then signed by the operator. All of this information is then sent back to the user, who then updates the logbook as in the UReg task.

Outsource. The goal of the Outsource task is to *distribute* the data of the user between the operator and a helper. The data is shared using the RSS protocol from Fig. 7.1. The two recipients are the helper and operator; each of them gets an additive share of the data alongside a *commitment* of the *other parties* share and unveil information for the own commitment.

The user uses the RSS-protocol to create robust shares of the UH, the unauthenticated input, and a One Time Pad that is long enough to mask the permutation, the direct update, the additive update, the unveil information thereof and the unauthenticated output from the computation. Due to the additional requirements for the function f (*cf.* Section 7.1)

[1] We refer to the update vectors as trivial if the permutation is the identity and the direct update does not change any values.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the first part of $\mathtt{Outsource}$ for a function f.

On input $(\mathtt{Outsource}, \mathsf{f}, x_{\mathsf{U}})$, U draws one-time pads otp_y, otp_{unv}, $otp_{\vec{\alpha}}$, $otp_{\vec{s}}$ and $otp_{\vec{a}}$ of appropriate size, sets $otp := otp_y \| otp_{\mathrm{unv}} \| otp_{\vec{\alpha}} \| otp_{\vec{s}} \| otp_{\vec{a}}$, and follows $\mathtt{Share}(msg)$ from Fig. 7.1 for $msg \in \{UH, x_{\mathsf{U}}, otp\}$ to obtain robust shares $rs^{(\mathsf{H})}_{UH}$, $rs^{(\mathsf{H})}_{x_{\mathsf{U}}}$, $rs^{(\mathsf{H})}_{otp}$, $rs^{(\mathsf{O})}_{UH}$, $rs^{(\mathsf{O})}_{x_{\mathsf{U}}}$, and $rs^{(\mathsf{O})}_{otp}$.

Now U loads the old commitments $(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*})$, $(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*})$ and $(\mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}})$ alongside the old serial ser^* and its commitments $(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*})$ from the logbook λ, draws a random $ser_0 \overset{\$}{\leftarrow} \mathbb{Z}_o$, computes a commitment $(\mathtt{com}_{ser_0}, \mathtt{unv}_{ser_0}) \leftarrow \textsc{Com}.\mathrm{Com}(ser_0)$ and proves correct sharing according to Fig. 7.17 by setting $stmt := (rs^{(\mathsf{O})}_{UH}, \mathtt{com}_{UH^{(\mathsf{O})}}, ser^*, \mathtt{com}_{ser}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{vk}_{\mathsf{O}})$ and $wit := (\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}^{(\mathsf{H})}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{unv}_{ser}, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, \mathsf{sk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH^*})$ and computes $\pi \leftarrow \textsc{Zk}.\mathrm{Proof}(stmt, wit, \Lambda_{\mathrm{Outsource}})$. Finally, U follows $\mathtt{Send}(\mathsf{O}, (rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_{\mathsf{U}}}, rs^{(\mathsf{O})}_{otp}, ser^*, \mathtt{com}_{ser}, \mathtt{com}_{ser_0}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \pi))$ and $\mathtt{Send}(\mathsf{H}, (rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_{\mathsf{U}}}, rs^{(\mathsf{H})}_{otp}))$ and $\mathtt{Send}(\mathsf{H}, (rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_{\mathsf{U}}}, rs^{(\mathsf{H})}_{otp}))$ from Fig. 7.4.

When receiving the message from $\mathcal{F}_{\mathsf{ORR}}$, O aborts if ser was used before and otherwise remembers ser.

Then O picks $lin^{(\mathsf{O})} \overset{\$}{\leftarrow} \mathbb{Z}_o$ and computes commitments $(\mathtt{com}_{lin^{(\mathsf{O})}}, \mathtt{unv}_{lin^{(\mathsf{O})}}) \leftarrow \textsc{Com}.\mathrm{Com}(lin^{(\mathsf{O})})$.

Finally, O sends $(\mathtt{com}_{lin^{(\mathsf{O})}}, \mathtt{com}^{(\mathsf{H})}_{UH}, \mathtt{com}^{(\mathsf{H})}_{x_{\mathsf{U}}}, \mathtt{com}^{(\mathsf{H})}_{otp})$ to H, where the commitments (except for lin) were taken from the respective robust shares.

Continued on Fig. 7.15.

Figure 7.14.: The eighth part of the protocol Π_{BKA} with the first part on how to outsource a function f.

those are applied to the output given to the helper, as the helper itself does not provide inputs nor gets dedicated outputs but only *stores* the data until the user fetches it. Since the helper should not see the outputs they are masked using the OTP.

The double spending detection and history freshness are similar to the BK task; the language from Fig. 7.17 is similar to the one from Fig. 7.12 with the main difference being that here the user still proves correct rerandomization of the linking number, but as the protocol already contains a check that the linking number is 0 (to ensure that the user can only outsource exactly once) the real linking number is not part of the witness anymore; this means that the user does not have to rerandomize the commitment on the linking number, and instead only proves that the old linking number is 0.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the second part of `Outsource` for a function f.

Continuation of Fig. 7.14.

When receiving the messages from $\mathcal{F}_{\mathsf{ORR}}$ and O, H follows $\mathtt{Verify}(rs_{UH}^{(\mathsf{H})}, \mathsf{com}_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, \mathsf{com}_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}, \mathsf{com}_{otp}^{(\mathsf{H})})$ from Fig. 7.3 and aborts if the output is 0.

Otherwise, H draws $lin^{(\mathsf{H})} \overset{\$}{\leftarrow} \mathbb{Z}_o$ and sends $(lin^{(\mathsf{H})}, \mathsf{com}_{UH^{(\mathsf{O})}}, \mathsf{com}_{x_\mathsf{U}^{(\mathsf{O})}}, \mathsf{com}_{otp^{(\mathsf{O})}})$ to O, where the commitments were taken from the respective robust shares.

When receiving the message from H, O sets $stmt := (rs_{UH}^{(\mathsf{O})}, \mathsf{com}_{UH^{(\mathsf{O})}}, ser^*, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{vk}_\mathsf{O})$, follows $\mathtt{Verify}(rs_{UH}^{(\mathsf{O})}, \mathsf{com}_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, \mathsf{com}_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}, \mathsf{com}_{otp}^{(\mathsf{H})})$ from Fig. 7.3 and aborts if the output is 0 or if $\textsc{Zk}.\mathrm{Vfy}(\pi, stmt, \Lambda_{\mathrm{Outsource}}) = 0$.

Otherwise, O sets $lin := lin^{(\mathsf{O})} + lin^{(\mathsf{H})}$ and stores the tuple $(\mathsf{H}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_\mathsf{U}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})})$ for Outsourced Analytics.

Now O draws a random $ser_1 \overset{\$}{\leftarrow} \mathbb{Z}_o$, computes $(\mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}) \leftarrow \textsc{Com}.\mathrm{Com}(ser_1)$, and homomorphically computes $\mathsf{com}_{UH} := \textsc{Com}.\mathrm{CAdd}(\mathsf{com}_{UH^\mathsf{H}}, \mathsf{com}_{UH^\mathsf{O}})$ and $\mathsf{com}_{ser} := \textsc{Com}.\mathrm{CAdd}(\mathsf{com}_{ser_0}, \mathsf{com}_{ser_1})$. Then O computes a signature $\sigma_{UH} \leftarrow \textsc{Sig}.\mathrm{Sign}_{\mathsf{k}_\mathsf{O}}(\mathsf{com}_{UH}, \mathsf{com}_{ser}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}})$.

Finally, O sends $(\mathtt{Process\_Back\_Onion}, (ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \sigma_{UH}), \vartheta)$ to $\mathcal{F}_{\mathsf{ORR}}$ and $(lin^{(\mathsf{O})}, \mathsf{unv}_{lin^{(\mathsf{O})}})$ to H and outputs Ok.

Continued on Fig. 7.16

Figure 7.15.: The nineth part of the protocol Π_{BKA} that specifies how to outsource a function f.

Additionally, the language lets the user prove that the UH was shared correctly. That is, the user proves that the two values inside the commitments sent to helper and operator on the shares *add up* to the UH that was signed by the operator during the last interaction.

The creation of a new serial number is similar to the method in BK; the user draws a share, sends a commitment thereof to the operator, who then draws a second share, computes commitments, and homomorphically computes a commitment on the final serial number. And the final part of the protocol contains the creation of a new logbook. This is similar to the method from the BK task, with the main exception that the helper and operator compute a *linking number*, to which the operator then computes a commitment which is sent alongside the unveil information and the clear value to the user.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the third part of `Outsource` for a function f.

Continuation of Fig. 7.15.

When receiving the message from O, H aborts if $\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin^{(\mathsf{O})}}, \mathtt{unv}_{lin^{(\mathsf{O})}}, lin^{(\mathsf{O})}) = 0$. Otherwise, H stores the tuple $(lin, rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_{\mathsf{U}}}, rs^{(\mathsf{H})}_{otp})$ for Outsourced Analytics, sends $(\texttt{Process\_Back\_Onion}, (lin), \vartheta)$ to $\mathcal{F}_{\mathsf{ORR}}$ and outputs $\texttt{Ok}$.

When receiving both responses from $\mathcal{F}_{\mathsf{ORR}}$, U aborts if σ_{UH} fails to verify and otherwise stores $\lambda := (UH, \mathtt{com}_{UH}, \mathtt{unv}_{UH}, ser, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, \mathsf{sk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH})$ and outputs $\texttt{Ok}$.

Figure 7.16.: The tenth part of the protocol Π_{BKA} that specifies how to outsource a function f.

Language $\Lambda_{\text{Outsource}}(rs^{(\mathsf{O})}_{UH}, \mathtt{com}^{(\mathsf{O})}_{UH}, ser, \mathtt{com}_{ser}\mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{vk}_{\mathsf{O}})$

$\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}^{(\mathsf{H})}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{unv}_{ser}, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*},$
$\mathsf{sk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH^*}:$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, UH) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\textsc{Com}.\mathrm{CAdd}(\mathtt{com}^{(\mathsf{H})}_{UH}, \mathtt{com}^{(\mathsf{O})}_{UH}), \textsc{Com}.\mathrm{UAdd}(\mathtt{unv}^{(\mathsf{H})}_{UH}, \mathtt{unv}^{(\mathsf{O})}_{UH}),$
$UH) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, ser) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{ser}, \mathtt{unv}_{ser}, ser) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, 0) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathsf{sk}_{\mathsf{U}}) = 1 \wedge$
$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{sk}_{\mathsf{U}}) = 1 \wedge$
$\textsc{Sig}.\mathrm{Vfy}(\sigma_{UH^*}, (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{ser}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{vk}_{\mathsf{O}})) = 1$

Figure 7.17.: Language $\Lambda_{\text{Outsource}}$ used for the `Outsource` task.

Outsourced Analytics. The protocol for performing analytical tasks lets both parties fetch the values stored during the OS task earlier and input them into $\mathcal{F}_{\mathsf{MPC}}$. We denote by Z_{f} the number of user whose data is considered in the computation of f. The helper just takes the oldest Z_{f} stored entries, the operator fetches the oldest Z_{f} entries that were stored *for the given helper* H. Before using $\mathcal{F}_{\mathsf{MPC}}$ to compute f on the given data we again need to verify that the FPs entered by the operator are valid. The technique is the same as for the BK task but this time, the verification is done with the helper. If the data verifies

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the first part of $\texttt{Analytics}$ for a function f.

On input $(\texttt{Analytics}, \mathsf{f})$, H loads the first Z_f entries $\{(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z_\mathsf{f}}$ and removes them from the database.

On input $(\texttt{Analytics}, \mathsf{f}, fp, x_\mathsf{O})$, O loads the first Z_f entries $\{(\mathsf{H}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_\mathsf{U}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})})\}$ that correspond to the given helper H and removes them from the database.

Then O looks up the stored tuple $(\mathsf{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ for the given (f, fp) pair and sends $(\mathtt{com}_{fp}, \sigma_{fp})$ to H and $(\texttt{Compute}, \mathsf{f}, (fp, \mathtt{unv}_{fp}, \{rs_{UH}^{(\mathsf{O})}, rs_{x_\mathsf{U}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}\}_{\zeta=1}^{Z_\mathsf{f}}))$ to $\mathcal{F}_{\mathsf{MPC}}$.

When receiving the message by O, H aborts if $\mathrm{Sig.Vfy}(\sigma_{fp}, \mathsf{vk}_\mathsf{T}, (\mathsf{f}, \mathtt{com}_{fp})) = 0$ and otherwise sends $(\texttt{Compute}, \mathsf{f}, (\mathtt{com}_{fp}, \{(rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z}))$ into $\mathcal{F}_{\mathsf{MPC}}$.

On output $(\{(ct_{\vec{\alpha}_\zeta}, ct_{\vec{s}_\zeta}, ct_{\vec{a}_\zeta}, ct_{\mathrm{unv}_{\vec{a}_\zeta}}, ct_{y_\mathsf{U}})_\zeta\}_{\zeta=1}^{Z_\mathsf{f}})$ by $\mathcal{F}_{\mathsf{MPC}}$, H stores a tuple $((lin, ct_{\vec{\alpha}_\zeta}, ct_{\vec{s}_\zeta}, ct_{\vec{a}_\zeta}, ct_{\mathrm{unv}_{\vec{a}_\zeta}}, ct_{y_\mathsf{U}})_\zeta)$ for Update for each $\zeta \in [Z_\mathsf{f}]$ and output Ok.

On output $(\{(\vec{\alpha}_\zeta, \vec{s}_\zeta, \mathtt{com}_{\vec{a}_\zeta}, ct_{y_\mathsf{U}})_\zeta\}_{\zeta=1}^{Z_\mathsf{f}})$ by $\mathcal{F}_{\mathsf{MPC}}$, O stores a tuple $((lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_\mathsf{U}})_\zeta)$ for Update for each $\zeta \in [Z_\mathsf{f}]$ and output y_O.

Figure 7.18.: The eleventh part of the protocol Π_{BKA} with the computation of an analytical function f.

both parties provide input for $\mathcal{F}_{\mathsf{MPC}}$. These inputs contain the FPs and their validation (*i.e.* a commitment by the helper and unveil information by the operator) and the robust shares of each of the users whose data is used for this computation. As f is *benign* it follows that the data is correctly reconstructed inside $\mathcal{F}_{\mathsf{MPC}}$ and the results for the users are masked.

After obtaining output both parties store the data they require for the subsequent Upd tasks, using the *linking number* to link the data to the respective OS and Upd calls.

Update. The Update protocol from Figs. 7.19 to 7.21 contains two steps which do not necessarily have to be executed at once. In the first step the user only requests the data from the helper by sending the linking number. The helper then looks up the data stored for that linking number during the OA task, sends them back directly to the user and

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the first part of `Update`.

On input (`Update`), U loads the linking number lin and follows `Send`(H, (lin)) from Fig. 7.4.

On input (`Update`) and after receiving the message and return information from $\mathcal{F}_{\mathsf{ORR}}$, H looks up a stored tuple $(lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}})$ for the given linking number lin and aborts if there is no such tuple stored. Otherwise, H removes the stored entry from the database, sends (`Process_Back_Onion`, $(ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}), \vartheta$) to $\mathcal{F}_{\mathsf{ORR}}$ and outputs Ok.

When receiving the message from $\mathcal{F}_{\mathsf{ORR}}$, U computes commitments $(\mathtt{com}_{UH}, \mathtt{unv}_{UH}) \leftarrow \textsc{Com}.\mathsf{Com}(UH)$ and $(\mathtt{com}_{\mathsf{sk_U}}, \mathtt{unv}_{\mathsf{sk_U}}) \leftarrow \textsc{Com}.\mathsf{Com}(\mathsf{sk_U})$ and loads the corresponding old commitments $(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*})$, $(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*})$ and $(\mathtt{com}_{\mathsf{sk}^*_\mathsf{U}}, \mathtt{unv}_{\mathsf{sk}^*_\mathsf{U}})$ alongside the old serial ser^* and its commitments $(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*})$ from the logbook λ.

Then U draws a random $ser_0 \overset{\$}{\leftarrow} \mathbb{Z}_{\mathsf{O}}$ and commitments $(\mathtt{com}_{ser_0}, \mathtt{unv}_{ser_0}) \leftarrow \textsc{Com}.\mathsf{Com}(ser_0)$ and reconstructs $\vec{\alpha}, \vec{s}, \vec{a}, \mathtt{unv}_{\vec{a}}$ and y_U by applying the respective parts of otp to the received masked values.

Now U proves correct rerandomization by setting $stmt := (\mathtt{com}_{UH}, ser, lin, \mathtt{com}_{\mathsf{sk_U}}, \mathsf{vk_O})$ and $wit := (UH, \mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, \mathsf{pk_U}, \mathtt{com}_{\mathsf{sk}^*_\mathsf{U}}, \mathtt{unv}_{\mathsf{sk}^*_\mathsf{U}}, \mathtt{unv}_{\mathsf{sk_U}}, \sigma_{UH})$ and computing $\pi \leftarrow \textsc{Zk}.\mathsf{Proof}(stmt, wit, \Lambda_{\mathsf{Update}})$ from Fig. 7.22.

If $\vec{\alpha}$ and $\vec{s}$ are not empty, U sets $UH^{(1)} := \vec{\alpha}(UH)$, computes $UH^{(2)}$ by replacing entries from $UH^{(1)}$ with values from $\vec{s}$ wherever there are non-empty values in $\vec{s}$, and sets $UH^{\mathrm{new}} := UH^{(2)} + \vec{a}$.

Continued on Fig. 7.20.

Figure 7.19.: The twelveth part of the protocol Π_{BKA} with the first part on how to apply the update of an outsourced analytical computation.

deletes them; which concludes the task for the helper. In the second step the user requests the data from O. In that same message the user also proves validity of the user history as was done in the BK task; to that end the user uses the ZK language from Fig. 7.22. The language is quite similar to the one used for the OS task in Fig. 7.17 but instead of proving that the linking number is 0, the user proves that the linking number stored in the commitment that was last signed contains the number sent to the operator. And since there is no secret sharing involved, that part is not contained in Fig. 7.22.

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the second part of `Update`.

Continuation of Fig. 7.19.

U computes the corresponding commitments $(\mathsf{com}_{UH^{(1)}}, \mathsf{unv}_{UH^{(1)}}) \leftarrow \textsc{Com}.\mathsf{Com}(UH^{(1)})$ and $(\mathsf{com}_{UH^{(2)}}, \mathsf{unv}_{UH^{(2)}}) \leftarrow \textsc{Com}.\mathsf{Com}(UH^{(2)})$. Then U proves correct transfer of $\vec{\alpha}$ and $\vec{s}$ by setting $stmt^* := (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and $wit^* := (UH, \mathsf{unv}_{UH}, UH^{(1)}, \mathsf{unv}_{UH^{(1)}}, UH^{(2)}, \mathsf{unv}_{UH^{(2)}})$ and computing $\pi^* := \textsc{Zk}.\mathsf{Proof}(stmt^*, wit^*, \Lambda_{\mathrm{Transfer}})$ from Fig. 7.13. Otherwise, if $\vec{\alpha}$ and $\vec{s}$ are empty, set $UH^{\mathrm{new}} := UH + \vec{a}$, $\mathsf{com}_{UH^{(1)}} = \mathsf{com}_{UH}$, $\mathsf{com}_{UH^{(2)}} = \mathsf{com}_{UH}$ and $\pi^* = \perp$.

Finally, U follows $\mathtt{Send}(\mathsf{O}, (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, ser, \mathsf{com}_{ser_0}, lin, \pi, \pi^*))$ from Fig. 7.4.

When receiving the message and return information from $\mathcal{F}_{\mathsf{ORR}}$, O aborts if ser was used before and otherwise marks ser as used. Then O loads $(lin, \vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{a}}, ct_{y_\mathsf{U}})$ and removes it from the database; if no such entry exists for the given linking number lin then O aborts. Otherwise, O verifies the proof by setting $stmt := (\mathsf{com}_{UH}, ser, lin, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{vk}_\mathsf{O})$ and aborts if $\textsc{Zk}.\mathsf{Verify}(\pi, stmt, \Lambda_{\mathrm{Update}}) = 0$.

If $\vec{\alpha}$ and $\vec{s}$ are non-empty, O sets $stmt^* := (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and abort if $\textsc{Zk}.\mathsf{Verify}(\pi^*, stmt^*, \Lambda_{\mathrm{Transfer}}) \neq 1$.

Regardless of $\vec{\alpha}$ or $\vec{s}$, O sets $\mathsf{com}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathsf{CAdd}(\mathsf{com}_{UH^{(2)}}, \mathsf{com}_{\vec{a}})$, picks a random ser_1 and computes $(\mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}) \leftarrow \textsc{Com}.\mathsf{Com}(ser_1)$. Then O sets $lin = 0$, computes the commitment on the serial $\mathsf{com}_{ser} := \textsc{Com}.\mathsf{CAdd}(\mathsf{com}_{ser_0}, \mathsf{com}_{ser_1})$ and on the linking number $(\mathsf{com}_{lin}, \mathsf{unv}_{lin}) \leftarrow \textsc{Com}.\mathsf{Com}(lin)$ and creates the signature $\sigma_{UH^{\mathrm{new}}} \leftarrow \textsc{Sig}.\mathsf{Sign}_{(\mathsf{k}_\mathsf{O})}(\mathsf{com}_{UH^{\mathrm{new}}}, \mathsf{com}_{lin}, \mathsf{com}_{ser}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}})$. Finally, O sends $(\mathtt{Process\_Back\_Onion}, (ct_{y_\mathsf{U}}, \mathsf{com}_{\vec{a}}, ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \sigma_{UH^{\mathrm{new}}}), \vartheta)$ to $\mathcal{F}_{\mathsf{ORR}}$ and outputs $(\vec{\alpha}, \vec{s})$.

Continued on Fig. 7.21

Figure 7.20.: The thirteenth part of the protocol Π_{BKA} with the second part on how to apply the update of an outsourced analytical computation.

Similar to the BK task the user proves that both the permutation and the direct update were applied correctly to the UH if necessary. The technique—and even the language from Fig. 7.13—are identical to the one used in the BK task.

The user then sends everything to the operator, who verifies the proofs. Alongside the necessary information for creating the new logbook the operator sends the encrypted

Protocol Π_{BKA}

Protocol Π_{BKA} for handling the third part of $\mathtt{Update}$ for a function f.

Continuation of Fig. 7.20.

When receiving the message from $\mathcal{F}_{\mathsf{ORR}}$, U sets $ser := ser_0 + ser_1 \mod o$ and homomorphically updates $\mathtt{com}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_{UH^{(2)}}, \mathtt{com}_{\vec{a}})$, $\mathtt{unv}_{UH^{\mathrm{new}}} := \textsc{Com}.\mathrm{UAdd}(\mathtt{unv}_{UH^{(2)}}, \mathtt{unv}_{\vec{a}})$, $\mathtt{com}_{ser} := \textsc{Com}.\mathrm{CAdd}(\mathtt{com}_{ser_0}, \mathtt{com}_{ser_1})$ and $\mathtt{unv}_{ser} := \textsc{Com}.\mathrm{UAdd}(\mathtt{unv}_{ser_0}, \mathtt{unv}_{ser_1})$.

Finally, U aborts if $\sigma_{UH^{\mathrm{new}}}$ fails to verify, if $ct_{y_{\mathsf{U}}}$ received from O is different to the one received from H, or if $\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, \vec{a}) \neq 1$, and otherwise stores $\lambda := (UH^{\mathrm{new}}, \mathtt{com}_{UH^{\mathrm{new}}}, \mathtt{unv}_{UH^{\mathrm{new}}}, ser, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, \mathsf{sk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH^{\mathrm{new}}})$ and outputs $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$.

Figure 7.21.: The fourteenth part of the protocol Π_{BKA} with the third part on how to apply the update of an outsourced analytical computation.

Language $\Lambda_{\mathrm{Update}}(\mathtt{com}_{UH}, ser, lin, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{vk}_{\mathsf{O}})$

$UH, \mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, \mathtt{unv}_{UH}, \mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, \mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, \mathsf{pk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \sigma_{UH}$:

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH}, \mathtt{unv}_{UH}, UH) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{UH^*}, \mathtt{unv}_{UH^*}, UH) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{ser^*}, \mathtt{unv}_{ser^*}, ser) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin^*}, \mathtt{unv}_{lin^*}, lin) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}^*_{\mathsf{U}}}, \mathsf{sk}_{\mathsf{U}}) = 1 \wedge$

$\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{unv}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{sk}_{\mathsf{U}}) = 1 \wedge$

$\textsc{Sig}.\mathrm{Vfy}(\sigma^*_{UH}, (\mathtt{com}_{UH}, \mathtt{com}_{lin}, \mathtt{com}_{ser}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}), \mathsf{vk}_{\mathsf{O}}) = 1$

Figure 7.22.: Language $\Lambda_{\mathrm{Update}}$ used for the $\mathtt{Update}$ task.

user output and the commitment of the additive upgrade (which was not contained in the helper's message) to the user. The construction of the new logbook then uses the same techniques that were used before in the BK and OS task.

8. Security

In this chapter we analyze the security of our system. That is, we show that the protocol Π_{BKA} is *at least* as secure as our ideal functionality $\mathcal{F}_{\mathsf{BKA}}$, without relying on a trusted party to execute $\mathcal{F}_{\mathsf{BKA}}$ on all parties inputs. To that end, we provide a simulator that simulates the protocol messages of honest parties without knowing the parties secret input, and prove that those simulated messages cannot be differentiated by any efficient environment $\mathcal{Z}$.

For technical reasons, we have to restrict our adversary to corrupting only *either* the helper H, or the operator O. We split our simulator up in two parts. Section 8.1 contains the simulator for all corruption scenarios related to the security of an honest user U, even in the presence of other malicious users. Section 8.2 contains the simulator for all corruption scenarios regarding the security of an honest operator O. Combined, those two simulators cover all corruption scenarios, in which either H or O are honest.

8.1. User Security

In this section, we investigate the security of our system in scenarios that relate to the security of an honest user U. To that end, we prove the following theorem:

Theorem 8.1.1 (User Security). *If instantiated with the building blocks introduced in Section 5.3, it holds that*

$$\Pi_{BKA}^{\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{Reg}},\mathcal{F}_{\mathsf{SMT}},\mathcal{F}_{\mathsf{MPC}},\mathcal{F}_{\mathsf{ORR}}} \geq \mathcal{F}_{\mathsf{BKA}}$$

against all PPT-adversaries $\mathcal{B}$ that have statically corrupted the operator O and a subset of users U.

We use the UC-framework [41] and provide a simulator $\mathcal{S}$ for this case. The simulator provides a view for any PPT-environment $\mathcal{Z}$ (that is restricted to not corrupting any helpers) that is consistent with a *real* protocol execution.

The simulator is given in Figs. 8.1 to 8.10.

We now introduce a series of hybrid games $\textsc{Game}_i(\kappa)$ and corresponding simulators $\mathcal{S}_i$ for protocols $\Pi_{BKA}^{(i)}$. Formally, given security parameter κ, each hybrid has the following form:

$$\textsc{Game}_i(\kappa) := \mathtt{view}_{\Pi_{BKA}^{(i)},\mathcal{S}_i,\mathcal{Z}}(1^\kappa)$$

Simulator $\mathcal{S}_{BKA}$

State of a simulator $\mathcal{S}_{BKA}$ and setup against a corrupted operator.

The simulator stores:

- td_{sim}: Trapdoor for the zero knowledge scheme.
- vk_{O}: Verification key of the operator
- $(\mathsf{vk}_{\mathsf{T}}, \mathsf{k}_{\mathsf{T}})$: Signature key pair of the TSA.
- List of tuples $\{\mathsf{f}, fp, \mathtt{com}_{fp}, \sigma_{fp}, \ell\}$ of function parameters that can be used for a given function f.
- List of tuples $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}\}$.
- List of tuples $\{\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}, \mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, otp, ct_{y_{\mathsf{U}}}\}$.
- List of tuples $\{ssid, lin\}$.

On input `Setup`, $\mathcal{S}_{BKA}$ sets up the Common Reference String using $(crs, td_{sim}) \leftarrow \textsc{Zk}.\text{SetupSim}(1^{\kappa})$ and stores the simulation trapdoor td_{sim}. Now $\mathcal{S}_{BKA}$ starts simulating all the hybrid functionalities.

Figure 8.1.: The first part of the simulator with an honest user: Defines state and set up.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the initialization against a corrupted operator.

On input $(\mathtt{Initialize}, pid_{\mathsf{T}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ generates and stores a signature key pair $(\mathsf{k}_{\mathsf{T}}, \mathsf{vk}_{\mathsf{T}}) \leftarrow \textsc{Sig}.\text{KeyGen}(1^{\kappa})$ and follows the protocol of T. The verification key vk_{T} is used for the simulation of $\mathcal{F}_{\mathsf{Reg}}$.

On input $(\mathtt{Register}, \mathsf{vk}_{\mathsf{O}}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{Reg}}$, $\mathcal{S}_{BKA}$ aborts if a key for pid_{O} exists and otherwise calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of O with input $(\mathtt{Initialize})$.

On output (Ok) from $\mathcal{F}_{\mathsf{BKA}}$ to O, $\mathcal{S}_{BKA}$ reports output (Ok) from $\mathcal{F}_{\mathsf{Reg}}$ to O.

Figure 8.2.: The second part of the simulator with an honest user and corrupt operator: Defines initialization.

We then show for each pair of consecutive hybrids $\textsc{Game}_i(\kappa)$ and $\textsc{Game}_{i+1}(\kappa)$, that, given our underlying assumptions, no distinguisher can distinguish the two games better than by guessing.

For our proof, we consider the following hybrid games $\textsc{Game}_i(\kappa)$:

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `SignFP` task against a corrupted operator.

On input $(\mathtt{SignFP}, pid_{\mathsf{T}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after receiving $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \mathsf{f}, x_{\mathsf{O}})$ from O to T, $\mathcal{S}_{BKA}$ aborts if $\mathsf{Com.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) \neq 1$. Otherwise, $\mathcal{S}_{BKA}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of O with input $(\mathtt{SignFP}, fp, \mathsf{f}, x_{\mathsf{O}})$.

On output (Ok) from $\mathcal{F}_{\mathsf{BKA}}$ to O, $\mathcal{S}_{BKA}$ follows the protocol of T.

Figure 8.3.: The third part of the simulator with an honest user and corrupt operator: Defines behavior for signing function parameter.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `UserRegistration` task against a corrupted operator.

On input $(\mathtt{UserRegistration}, pid_{\mathsf{U}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the honest protocol of the user to create and send the first message $(\pi, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{com}_{ser_0})$ to O.

When receiving $(ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathtt{com}_{UH}, \mathtt{unv}_{UH}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH})$ from O to U, $\mathcal{S}_{BKA}$ inputs $(\mathtt{UserRegistration})$ in the name of the operator O into $\mathcal{F}_{\mathsf{BKA}}$.

When receiving (Ok) from $\mathcal{F}_{\mathsf{BKA}}$ to O, $\mathcal{S}_{BKA}$ follows the remaining honest protocol of the user and, upon successful termination of the simulated user, stores $\{pid_{\mathsf{U}}, \lambda\}$.

Figure 8.4.: The fourth part of the simulator with an honest user and corrupt operator: Defines behavior for user registration.

GAME$_1(\kappa)$: The first game is equivalent to the real experiment. That is,

$$\text{GAME}_1(\kappa) := \mathtt{view}_{\Pi_{BKA}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{S}_1, \mathcal{Z}}(1^\kappa)$$

This means that all parties execute the real protocol.

GAME$_2(\kappa)$: All hybrid functionalities, namely $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{SMT}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{ORR}})$, are now executed by $\mathcal{S}_2$.

Lemma 8.1.2. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\text{GAME}_1(\kappa)$ from $\text{GAME}_2(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\text{GAME}_1(\kappa)) = 1] - \Pr[\mathcal{Z}(\text{GAME}_2(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We are using the UC composition theorems, here. Letting $\mathcal{S}_2$ emulate all of $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{SMT}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{ORR}})$ is possible as all of them are UC-functionalities.

Simulator $\mathcal{S}_{BKA}$

The $\texttt{Compute}$ task for a function f against a corrupted operator.

On input $(\texttt{Compute}, \mathsf{f}, \mathsf{User})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ creates the commitments $(\mathsf{com}_{UH}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{com}_{ser}, \mathsf{com}_{lin})$ as zero-commitments $\textsc{Com}.\mathsf{Com}(\vec{0})$ for a zero-vector of appropriate size and samples a uniformly random serial $ser^* \overset{\$}{\leftarrow} \mathbb{Z}_{\mathsf{O}}$.

Then $\mathcal{S}_{BKA}$ forges a proof of correct rerandomization by setting $stmt := (\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, ser^*, \mathsf{com}_{ser}, \mathsf{vk}_\mathsf{O})$ and creating the proof as $\pi \leftarrow \textsc{Zk}.\mathsf{SimZK}(stmt, td_{sim}, \Lambda_{\mathsf{Compute}})$. Then $\mathcal{S}_{BKA}$ simulates $\texttt{Send}(\mathsf{O}, (\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{com}_{ser}, ser^*, \pi))$ to Fig. 7.4 and remembers ϑ.

On input $(\texttt{Process\_Back\_Onion}, (\mathsf{com}_{fp}, \sigma_{fp}), \vartheta)$ from O to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ aborts if there is no stored tuple $(\mathsf{f}, fp, \mathsf{com}_{fp}, \sigma_{fp}, \ell)$. Otherwise, $\mathcal{S}_{BKA}$ follows the protocol to execute $\texttt{Send}(\mathsf{O}, \mathsf{com}_{ser_0})$ from Fig. 7.4 and remembers ϑ and ℓ.

On input $(\texttt{Compute}, \mathsf{f}, (fp, \mathsf{com}_{UH}, \mathsf{unv}_{fp}, x_\mathsf{O}))$ from O to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{BKA}$ aborts if $\textsc{Com}.\mathsf{Unv}(\mathsf{com}_{UH}, \mathsf{unv}_{UH}, UH) \neq 1$ or $\textsc{Com}.\mathsf{Unv}(\mathsf{com}_{fp}, \mathsf{unv}_{fp}, fp) \neq 1$. Otherwise, $\mathcal{S}_{BKA}$ sends input $(\texttt{Compute}, \mathsf{f}, x_\mathsf{O}, fp)$ to $\mathcal{F}_{\mathsf{BKA}}$ in the name of O.

When receiving the leak (ℓ^*) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ aborts if $\ell^* \neq \ell$.

On output $(\vec{\alpha}, \vec{s}, y_\mathsf{O})$ from $\mathcal{F}_{\mathsf{BKA}}$ to O, $\mathcal{S}_{BKA}$ sets $(\mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{0})$ and reports $(\texttt{Output}, (\vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{a}}, y_\mathsf{O}))$ as output from $\mathcal{F}_{\mathsf{MPC}}$ to O.

Then $\mathcal{S}_{BKA}$ computes $\mathsf{com}_{UH^{(1)}}$ and $\mathsf{com}_{UH^{(2)}}$ as $\textsc{Com}.\mathsf{Com}(0)$ each, sets $stmt^* := (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and forges a proof $\pi^* \leftarrow \textsc{Zk}.\mathsf{SimZK}(stmt^*, td_{sim}, \Lambda_{\mathsf{Transfer}})$. Then $\mathcal{S}_{BKA}$ follows $\texttt{Send}(\mathsf{O}, (\mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \pi^*))$ from Fig. 7.4 and remembers ϑ.

On input $(\texttt{Process\_Back\_Onion}, (ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \sigma_{UH^{\mathrm{new}}}), \vartheta)$ from O to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ follows the protocol to construct ser and to verify the signature.

Figure 8.5.: The fifth part of the simulator with an honest user and corrupt operator: Defines behavior for computation of a function f.

Essentially all that happens here is a re-encapsulation on who executes what; as this regrouping is only cosmetically and all the hybrids can be executed in PPT this change cannot be detected. □

GAME$_3(\kappa)$: The simulator $\mathcal{S}_3$ now maintains a state in which it stores exactly the same data that an honest operator stores for the UI of the UH during an OA task. During simulation of $\mathcal{F}_{\mathsf{MPC}}$ for OA, *i.e.* after O inserted $(\texttt{Compute}, \mathsf{f}, (fp, \mathsf{unv}_{fp}, \{(rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{otp})_\zeta\}_{\zeta=1}^{Z_\mathsf{f}}))$ and H inserted $(\texttt{Compute}, \mathsf{f}, (\mathsf{com}_{fp}, \{(rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_\mathsf{U}}, rs^{(\mathsf{H})}_{otp})_\zeta\}_{\zeta=1}^{Z}))$ into

Simulator $\mathcal{S}_{BKA}$

The $\mathtt{Outsource}$ task for outsourcing a function f against a corrupted operator and an honest user and helper.

On input $(\mathtt{Outsource}, \mathsf{f}, \mathsf{User})$ and $(\mathtt{Outsource}, \mathsf{f}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ samples $ser \xleftarrow{\$} \mathbb{Z}_o$, and creates robust shares $(rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{O})}_{UH}), (rs^{(\mathsf{H})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{x_\mathsf{U}}), (rs^{(\mathsf{H})}_{otp}, rs^{(\mathsf{O})}_{otp})$ by following $\mathtt{Share}(\vec{0})$ from Fig. 7.1 and $\mathtt{com}_{\mathsf{sk}_\mathsf{U}}$, $\mathtt{com}_{ser_0}$ and $\mathtt{com}_{ser}$ as zero-commitments $\textsc{Com}.\mathrm{Com}(\vec{0})$ for respective zero-vectors of appropriate size. Then $\mathcal{S}_{BKA}$ simulates the proof of correct sharing by setting $stmt := (rs^{(\mathsf{O})}_{UH}, \mathtt{com}_{UH^{(\mathsf{O})}}, ser^*, \mathtt{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{vk}_\mathsf{O})$ and setting $\pi \leftarrow \textsc{Zk}.\mathrm{SimZK}(stmt, td_{sim}, \Lambda_{\mathrm{Outsource}})$. Now $\mathcal{S}_{BKA}$ follows $\mathtt{Send}(\mathsf{O}, (rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{otp}, ser^*, \mathtt{com}_{ser}, \mathtt{com}_{ser_0}, \mathtt{com}_{\mathsf{sk}_\mathsf{U}}, \pi))$ from Fig. 7.4 and remembers ϑ.

On input $(\mathtt{com}_{lin^{(\mathsf{O})}}, \mathtt{com}^{(\mathsf{H})}_{UH}, \mathtt{com}^{(\mathsf{H})}_{x_\mathsf{U}}, \mathtt{com}^{(\mathsf{H})}_{otp})$ from O to H, $\mathcal{S}_{BKA}$ aborts if any of the commitments differ from the ones created earlier. Otherwise, $\mathcal{S}_{BKA}$ sets $lin^{(\mathsf{H})} \xleftarrow{\$} \mathbb{Z}_o$ and reports a message $(lin^{(\mathsf{H})}, \mathtt{com}_{UH^{(\mathsf{O})}}, \mathtt{com}_{x_\mathsf{U}^{(\mathsf{O})}}, \mathtt{com}_{otp^{(\mathsf{O})}})$ from H to O.

When receiving $(lin^{(\mathsf{O})}, \mathtt{unv}_{lin^{(\mathsf{O})}})$ from O to H, $\mathcal{S}_{BKA}$ aborts if $\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{lin^{(\mathsf{O})}}, \mathtt{unv}_{lin^{(\mathsf{O})}}, lin^{(\mathsf{O})}) \neq 1$ and otherwise reconstructs lin, stores $(pid_\mathsf{H}, \mathsf{f}, lin, rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_\mathsf{U}}, rs^{(\mathsf{H})}_{otp})$ for OA, loads the ssid and stores (ssid, lin).

On input $(\mathtt{Process\_Back\_Onion}, (ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH}), \vartheta)$ from O to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ follows the protocol to reconstruct ser and to verify the signature. If $\mathcal{S}_{BKA}$ did not abort it sends input $(\mathtt{Outsource}, \mathsf{f})$ to $\mathcal{F}_{\mathsf{BKA}}$ in the name of O and awaits output (Ok).

Figure 8.6.: The sixth part of the simulator with an honest user, honest helper and corrupt operator: Defines behavior for outsourcing of a function f.

$\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_3$ computes f honestly (with fresh random coins, if necessary), based on the two inputs. $\mathcal{S}_3$ uses $\mathtt{Combine}$ from Fig. 7.2 on the inputs to reconstruct $(UH, x_\mathsf{U}, otp)_\zeta$ for each user $\zeta \in [Z]$. If reconstruction on any of the shares fails, $\mathcal{S}_3$ aborts. Otherwise, $\mathcal{S}_3$ computes for each $\zeta \in [Z_\mathsf{f}]$ the commitment $(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}) \leftarrow \mathrm{Com}(\vec{a})$ and the masked output $ct_{y_\mathsf{U}} := y_\mathsf{U} + otp_y$. $\mathcal{S}_3$ then stores a new entry $\{\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}, \mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, otp, ct_{y_\mathsf{U}}\}$ for the UI.

Lemma 8.1.3. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_2(\kappa)$ from $\textsc{Game}_3(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_2(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_3(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Simulator $\mathcal{S}_{BKA}$

The $\texttt{Outsource}$ task for outsourcing a function f against a corrupted operator and corrupted user.

On input $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $(\texttt{Process\_Onion}, \mathsf{H}, (rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, and after receiving $(\mathsf{com}_{lin^{(\mathsf{O})}}, \mathsf{com}_{UH}^{(\mathsf{H})}, \mathsf{com}_{x_{\mathsf{U}}}^{(\mathsf{H})}, \mathsf{com}_{otp}^{(\mathsf{H})})$ from O to H, $\mathcal{S}_{BKA}$ follows the protocol and aborts if $\texttt{Verify}(rs_{UH}^{(\mathsf{H})}, \mathsf{com}_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, \mathsf{com}_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}, \mathsf{com}_{otp}^{(\mathsf{H})})$ from Fig. 7.3 fails. Then $\mathcal{S}_{BKA}$ draws a random $lin^{(\mathsf{H})}$ and reports a message $(lin^{(\mathsf{H})}, \mathsf{com}_{UH^{(\mathsf{O})}}, \mathsf{com}_{x_{\mathsf{U}}^{(\mathsf{O})}}, \mathsf{com}_{otp^{(\mathsf{O})}})$ from H to O.

On input $(lin^{(\mathsf{O})}, \mathsf{unv}_{lin^{(\mathsf{O})}})$ from O to H, $\mathcal{S}_{BKA}$ follows the protocol, aborts if the commitment on $lin^{(\mathsf{O})}$ is invalid and otherwise reconstructs lin and stores $(pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$ for OA.

Then $\mathcal{S}_{BKA}$ loads the ssid $ssid$ and stores $(ssid, lin)$. Now $\mathcal{S}_{BKA}$ simulates sending (lin) through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$.

Finally, $\mathcal{S}_{BKA}$ sends $(\texttt{Outsource}, \mathsf{f}, \perp)$ in the name of U to $\mathcal{F}_{\mathsf{BKA}}$ and $(\texttt{Outsource}, \mathsf{f})$ in the name of O to $\mathcal{F}_{\mathsf{BKA}}$ and awaits response $(\texttt{Ok})$ for both.

Figure 8.7.: The seventh part of the simulator with an honest user, corrupted helper and corrupt operator: Defines behavior for outsourcing of a function f.

Proof. The behavior of the two simulators is exactly identical, only that $\mathcal{S}_3$ has more information; namely the update information. Since none of the messages depend on this information, indistinguishability trivially follows.

However, there is a now abort-criteria for the simulator. We also have show that the abort by $\mathcal{S}_3$ in $\textsc{Game}_3(\kappa)$ only occurs iff H aborts in $\textsc{Game}_2(\kappa)$. Assume that H and O in $\textsc{Game}_2(\kappa)$ have respective shares $rs_{\cdot}^{(\mathsf{H})}$ and $rs_{\cdot}^{(\mathsf{O})}$. Without loss of generality, assume that the share that caused the abort of $\mathcal{F}_{\mathsf{MPC}}$ in $\textsc{Game}_2(\kappa)$ be that of the UH UH, as the cases for x_{U} and otp are analogous. In $\textsc{Game}_2(\kappa)$ the two shares are handed over to the subfunctionality $\mathcal{F}_{\mathsf{MPC}}$, where they are merged with $\texttt{Combine}$. The procedure aborts if the verification of the shares fails.

The simulator $\mathcal{S}_3$ in $\textsc{Game}_3(\kappa)$ does exactly the same steps; it aborts iff verification in $\texttt{Combine}$ fails. Hence, the abort criteria are identical as the same code is executed only by two different machines. So no environment $\mathcal{Z}$ can distinguish the two games better than by guessing. □

$\textsc{Game}_4(\kappa)$: This game is as $\textsc{Game}_3(\kappa)$ with the one change that during setup, instead of honestly sampling a CRS, $\mathcal{S}_4$ computes $(crs, td_{sim}) \leftarrow \mathsf{SetupSim}$, publishes $crs := crs$

Simulator $\mathcal{S}_{BKA}$

$\texttt{Analytics}$ computing an analytical function f against a corrupted operator and honest helper.

On input $(\texttt{Analytics}, \mathsf{f}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after receiving $(\mathsf{com}_{fp}, \sigma_{fp})$ from O to H, $\mathcal{S}_{BKA}$ aborts if no entry $\{\mathsf{f}, fp^*, \mathsf{com}_{fp}, \sigma_{fp}, \ell^*\}$ was stored for the given $(\mathsf{f}, \mathsf{com}_{fp}, \sigma_{fp})$ and otherwise remember ℓ^* and fp^* for later.

On input $(\texttt{Compute}, \mathsf{f}, (fp, \mathsf{unv}_{fp}, \{rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_{\mathsf{U}}}, rs^{(\mathsf{O})}_{otp}\}^{Z_{\mathsf{f}}}_{\zeta=1}))$ from O to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{BKA}$ aborts if $fp \neq fp^*$. Otherwise, $\mathcal{S}_{BKA}$ follows the protocol of the helper and loads the first Z_{f} entries $\{(lin, rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_{\mathsf{U}}}, rs^{(\mathsf{H})}_{otp})_\zeta\}^{Z_{\mathsf{f}}}_{\zeta=1}$ and uses $\texttt{Combine}$ from Fig. 7.2 to reconstruct all $(UH, x_{\mathsf{U}}, otp)_\zeta$ tuples from the robust shares.

Otherwise, $\mathcal{S}_{BKA}$ inputs $(\texttt{Analytics}, fp, x_{\mathsf{O}})$ in the name of O into $\mathcal{F}_{\mathsf{BKA}}$.

When receiving the leak (ℓ) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ aborts if $\ell^* \neq \ell$.

When receiving a list $\{(\zeta)\}$ of indices where the data belongs to corrupted users, $\mathcal{S}_{BKA}$ inserts the previously reconstructed inputs $\{(UH, x_{\mathsf{U}})_\zeta\}$ for all corrupted users $\zeta \in \{(\zeta)\}$ into $\mathcal{F}_{\mathsf{BKA}}$.

When receiving the leak $\{(\zeta), \vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta\}$ for the data of all corrupted users from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ does the following: For each $\zeta^* \in [Z_{\mathsf{f}}]$, if ζ^* belongs to a corrupted user (*i.e.* if it was contained in the previous leak of $\{(\zeta)\}$), $\mathcal{S}_{BKA}$ sets $\vec{a}$ and y_{U} as the value contained in the leak. Otherwise, if ζ^* belongs to an honest user, $\mathcal{S}_{BKA}$ sets $\vec{a} := \vec{0}$ and draws a random y_{U}.

Now $\mathcal{S}_{BKA}$ computes $(\mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{a})$, extracts otp_{unv} from otp and sets $ct_{y_{\mathsf{U}}} := y_{\mathsf{U}} + otp_{\mathsf{unv}}$.

On output $(\{\vec{\alpha}_\zeta, \vec{s}_\zeta\}^{Z}_{\zeta=1}, y_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$ to O, $\mathcal{S}_{BKA}$ adds entries $\{(\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}, \mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}, otp, ct_{y_{\mathsf{U}}})_\zeta\}$ for each $\zeta \in [Z_{\mathsf{f}}]$ and reports $(\{(\vec{\alpha}_\zeta, \vec{s}_\zeta, \mathsf{com}_{\vec{a}_\zeta}, ct_{y_{\mathsf{U}}})_\zeta\}^{Z_{\mathsf{f}}}_{\zeta=1})$ as output from $\mathcal{F}_{\mathsf{MPC}}$ to O.

Figure 8.8.: The eighth part of the simulator with an honest helper and corrupt operator: Defines behavior for computing an outsourced function f.

as CRS and stores td_{sim}. Also, the simulator stores the verification key vk_{O} of the operator. This is obtained by simulating $\mathcal{F}_{\mathsf{Reg}}$ during the initialization.

Lemma 8.1.4. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_3(\kappa)$ from $\textsc{Game}_4(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_3(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_4(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Simulator $\mathcal{S}_{BKA}$

$\texttt{Update}$ for updating the user data after an outsourced computation against a corrupted operator and honest user and helper.

On input $(\texttt{Update}, \text{User})$ and $(\texttt{Update}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ inserts $(\texttt{Update})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of O.

When receiving the leak $(ssid)$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ looks up and removes $(ssid, lin)$ and abort if no such tuple is stored. Then $\mathcal{S}_{BKA}$ looks up and removes $\{\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}^*, \mathtt{com}_{\vec{a}^*}, \mathtt{unv}_{\vec{a}^*}, otp, ct^*_{y_{\mathsf{U}}}\}$ for the given H and lin and aborts if no such tuple is stored.

Otherwise $\mathcal{S}_{BKA}$ computes fresh zero-commitments $\text{Com}.\text{Com}(\vec{0})$ for $\mathtt{com}_{UH}$, $\mathtt{com}_{ser_0}$ and $\mathtt{com}_{\mathsf{sk_U}}$, draws a random $ser^* \xleftarrow{\$} \mathbb{Z}_o$, and proves correct rerandomization by setting $stmt := (\mathtt{com}_{UH}, ser, lin, \mathtt{com}_{\mathsf{sk_U}}, \mathsf{vk_O})$ and computing $\pi \leftarrow \text{Zk}.\text{SimZK}(stmt, td_{sim}, \Lambda_{\text{Update}})$.

If $\vec{\alpha}$ and $\vec{s}$ contain non-trivial values, $\mathcal{S}_{BKA}$ computes $\mathtt{com}_{UH^{(1)}}$ and $\mathtt{com}_{UH^{(2)}}$ as $\text{Com}.\text{Com}(\vec{0})$ each, sets $stmt^* := (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and forges a proof $\pi^* \leftarrow \text{Zk}.\text{SimZK}(stmt^*, td_{sim}, \Lambda_{\text{Transfer}})$. Otherwise, $\mathcal{S}_{BKA}$ sets $\mathtt{com}_{UH^{(1)}} = \mathtt{com}_{UH^{(2)}} := \mathtt{com}_{UH}$ and $\pi^* := \bot$.

Then $\mathcal{S}_{BKA}$ follows $\texttt{Send}(\mathsf{O}, (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \mathtt{com}_{\mathsf{sk_U}}, ser, \mathtt{com}_{ser_0}, lin, \pi, \pi^*))$ from Fig. 7.4 and remembers ϑ.

On input $(\texttt{Process\_Back\_Onion}, (ct_{y_{\mathsf{U}}}, \mathtt{com}_{\vec{a}}, ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH^{\text{new}}}), \vartheta)$ from O to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ aborts if $\mathtt{com}_{\vec{a}} \neq \mathtt{com}_{\vec{a}^*}$ or $ct_{y_{\mathsf{U}}} \neq ct^*_{y_{\mathsf{U}}}$. Otherwise, $\mathcal{S}_{BKA}$ follows the protocol of the user to verify the logbook and signature and aborts if it is invalid.

Figure 8.9.: The ninth part of the simulator with an honest user and helper and corrupt operator: Defines behavior for updating the user data after outsourcing an analytical computation.

Proof. Indistinguishability trivially follows from the trapdoor-nature of Com and Zk. If any environment $\mathcal{Z}$ could distinguish the execution of the protocol when using crs created by $crs \leftarrow \text{Setup}$ from crs created by $(crs, td_{sim}) \leftarrow \text{SetupSim}$ with probability $\frac{1}{2} + \alpha$, we can build a PPT-environment $\mathcal{Z}'$ that breaks the indistinguishability of the dual-mode property of Zk by having $\mathcal{Z}'$ execute the code of all parties in its head. This leads to the same success probability of $\frac{1}{2} + \alpha$, thus causing $\alpha \in \text{negl}(\kappa)$ by requirement of the chosen Zk-scheme. □

Game$_5(\kappa)$: Replaces all zero-knowledge proofs of honest parties by simulated proofs (using td_{sim}) created by the simulator. Note that the simulated proofs can be created *independently* from (thus without knowing) the actual witness.

Simulator $\mathcal{S}_{BKA}$

`Update` for updating the user data after an outsourced computation against a corrupted operator, corrupted user and honest helper.

On input $(\texttt{Update}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after receiving $(\texttt{Process\_Onion}, \mathsf{H}, (lin), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ looks up and removes $\{\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}^{*}, \mathtt{com}_{\vec{a}^{*}}, \mathtt{unv}_{\vec{a}^{*}}, otp, ct^{*}_{y_{\mathsf{U}}}\}$ for the given H and lin and aborts if no such tuple is stored. Otherwise, $\mathcal{S}_{BKA}$ extracts $(otp_{\vec{\alpha}}, otp_{\vec{s}}, otp_{\vec{a}}, otp_{\mathtt{unv}}, otp_{y})$ from otp, sets $ct_{\vec{\alpha}} := \vec{\alpha} + otp_{\vec{\alpha}}$, $ct_{\vec{s}} := \vec{s} + otp_{\vec{s}}$, $ct_{\vec{a}} := \vec{a} + otp_{\vec{a}}$, $ct_{\mathtt{unv}_{\vec{a}}} := \mathtt{unv}_{\vec{a}} + otp_{\mathtt{unv}}$, and $ct_{y_{\mathsf{U}}} := y_{\mathsf{U}} + otp_{y}$.

Then $\mathcal{S}_{BKA}$ simulates sending $(ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct\mathtt{unv}_{\vec{a}}, cty_{\mathsf{U}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$ and inserts $(\texttt{Update})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of O and a randomly chosen corrupted U and awaits output $(\mathtt{Ok})$ for both.

Figure 8.10.: The tenth part of the simulator with an honest helper and corrupt operator and user: Defines behavior for updating the user data after outsourcing an analytical computation.

Lemma 8.1.5. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_4(\kappa)$ from $\textsc{Game}_5(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_4(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_5(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Any PPT-environment $\mathcal{Z}$ that could distinguish those two games would trivially be able to successfully break the dual-mode property of $\textsc{Zk}$, which is not possible by assumption. □

$\textsc{Game}_6(\kappa)$: This game changes $\textsc{Game}_5(\kappa)$ in the simulation of the BK task: Instead of computing the addition vector $\vec{a}$ alongside its commitment and decommitment information honestly according to the (benign) function f the simulator uses $\vec{a} = \vec{0}$ for a sufficiently big zero-vector. As commitment- and unveil-information $\mathcal{S}_6$ uses $(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{0})$. Note that the values for $\vec{a}$ and $\mathtt{unv}_{\vec{a}}$ are not needed for simulation since $\textsc{Game}_5(\kappa)$ so the only remaining value visible to the environment is $\mathtt{com}_{\vec{a}}$.

Lemma 8.1.6. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_5(\kappa)$ from $\textsc{Game}_6(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_5(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_6(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. As already stated in the game description the only used value that is visible to the PPT environment is the commitment $\mathtt{com}_{\vec{a}}$. The values $\vec{a}$ and $\mathtt{unv}_{\vec{a}}$ are only used as part of the witness in the Zero Knowledge proof in the original protocol and are not used since $\textsc{Game}_5(\kappa)$. Hence, the environment can only see the commitment $\mathtt{com}_{\vec{a}}$.

For the PPT-environment $\mathcal{Z}$ distinguishing $\textsc{Game}_5(\kappa)$ from $\textsc{Game}_6(\kappa)$ comes down to breaking the hiding-property of the commitment scheme $\textsc{Com}$ which is not possible (for the PPT-environment) by requirement. □

$\textsc{Game}_7(\kappa)$: All commitments that honest players create in the real protocol (*i.e.* those on *lin*, *ser*, *UH* and sk_U) are now created by the simulator as $\mathsf{com}_{\vec{0}}$, *i.e.* commitments to the zero-vector of appropriate size. Also, whenever any user U is supposed to send a serial number, $\mathcal{S}_7$ samples a new value $ser \xleftarrow{\$} \mathbb{Z}_o$ and sends this instead of a real serial number.

Lemma 8.1.7. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_6(\kappa)$ from $\textsc{Game}_7(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_6(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_7(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. First, note that since $\textsc{Game}_5(\kappa)$ the Zero Knowledge proofs π are simulated using the simulation trapdoor td_{sim} instead of proving actual properties of the commitments. Hence, there the change from *meaningful* commitments to zero-commitments cannot be noticed. Other than that, the commitments are only ever used for homomorphic addition; the environment $\mathcal{Z}$ only sees committed values. Any PPT-environment $\mathcal{Z}$ that could distinguish the two games $\textsc{Game}_6(\kappa)$ and $\textsc{Game}_7(\kappa)$ would be able to successfully break the hiding property of the commitment scheme $\textsc{Com}$, which by assumption is only possible with negligible advantage. □

$\textsc{Game}_8(\kappa)$: All helpers H are replaced by a equivalent machines H'. These behave similar, except for the OS task: There, H' sends to the simulator $\mathcal{S}_8$ the shares $(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$.

Lemma 8.1.8. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_7(\kappa)$ from $\textsc{Game}_8(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_7(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_8(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The helper sends the same messages in both games. None of the messages that $\mathcal{S}_8$ sends depend in any way on the leak provided by H'. The leak is hidden from the environment. Hence, the distributions for both games are trivially equivalent. □

$\textsc{Game}_9(\kappa)$: This game extends the state of the simulator $\mathcal{S}_9$ by letting it also store the data stored by the helper for OA. Namely, the additional data stored is $\{pid_\mathsf{H}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}\}$.

The map is only updated during OS-tasks. After $\mathcal{S}_9$ received the leak $(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$ from H', $\mathcal{S}_9$ adds that data directly to the list of tuples.

Lemma 8.1.9. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_8(\kappa)$ from $\textsc{Game}_9(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_8(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_9(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. In both games, the same messages are sent. The simulator also behaves equivalently, as none of the messages that $\mathcal{S}_9$ sends depend on the stored information. Hence, no (PPT) environment $\mathcal{Z}$ can differentiate these two games. □

GAME$_{10}(\kappa)$: Extends the state of the simulator $\mathcal{S}_{10}$ even further by adding a list of tuples $(ssid, lin)$ that link the ssid of the OS task with the linking number computed during that task. The map is only updated during OS-tasks. After receiving the leak $(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$ from H', the simulator $\mathcal{S}_{10}$ loads the current ssid $ssid$ and adds an entry $(ssid, lin)$.

Lemma 8.1.10. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_9(\kappa)$ from GAME$_{10}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_9(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{10}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Again, the only difference is that the simulator obtains and stores additional information. As no messages depend on the additional information, the environment $\mathcal{Z}$ is unable to differentiate the two games. □

GAME$_{11}(\kappa)$: All calls from honest parties of the form `Share`(msg) for an arbitrary $msg \in \mathbb{Z}_o^n$ during the protocol execution are replaced by calls `Share`$(\vec{0})$ from the simulator, where $\vec{0} = 0^n$ is the all-zero vector of appropriate size. Furthermore, the simulator takes the role of the helper H during the computation of the linking number during OS. Thus the helper no longer leaks $(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})$ to $\mathcal{S}_{11}$.

During simulation of OS-tasks, linking numbers are now created by the simulator and the operator. The simulator $\mathcal{S}_{11}$ still stores $\{pid_\mathsf{H}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}\}$, but acquires the data differently. The linking number lin is known to $\mathcal{S}_{11}$ due to its participation in the coin toss. The robust shares are known regardless of the user's corruption. If the user is honest, the shares are created by $\mathcal{S}_{11}$ in the first place and can be stored directly. If the user is corrupted, the environment sends the shares to the helper in the name of the user; as this happens via input to $\mathcal{F}_{\mathsf{ORR}}$ this message is visible to $\mathcal{S}_{11}$ due to the change induced in GAME$_2(\kappa)$.

Lemma 8.1.11. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{10}(\kappa)$ from GAME$_{11}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{10}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{11}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Through `Share`, the user creates OTP encrypted inputs for H and O; the environment $\mathcal{Z}$ only ever sees the share of the corrupted operator, not that of the helper. Thus $\mathcal{Z}$ only has a partial view, which information-theoretically hides the value that is to be shared due to properties of the OTP. The share itself is not used directly in any further arithmetic computations—only as input to the setup functionality $\mathcal{F}_{\mathsf{MPC}}$ during the OA-task. There, the simulator works independently of the shares and instead uses the actual values to simulate $\mathcal{F}_{\mathsf{MPC}}$ by computing f.

Hence, differentiation between shares of msg and shares of $\vec{0}$ is not possible for any PPT-environment $\mathcal{Z}$ without breaking the information-theoretic security of OTP encryption.

The computation of the linking number happens via Blum coin toss, where no secrets are involved; thus $\mathcal{S}_{11}$ can execute this part on behalf of the helper by following the protocol. This change is hence only cosmetically; the same code is executed on a different machine. Hence, since the simulator performs the honest computation and does exactly the same as H' would, the distributions for both games regarding computation of lin are equivalent.

Since the shares are created by the simulator, $\mathcal{S}_{11}$ does not have to rely on leaks by H' and can store them directly, having the same information afterwards. □

GAME$_{12}(\kappa)$: All honest user U are replaced by machines U' that run a similar code as U except for the Upd task: Here, the user only sends the linking number lin to the simulator $\mathcal{S}_{12}$. The remaining part of the honest user's protocol for the Upd task is played by the simulator.

After receiving the leak, $\mathcal{S}_{12}$ fetches the entry $\{\mathsf{H}, lin, \vec{\alpha}, \vec{s}, \vec{a}, \mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}, otp, ct_{y_\mathsf{U}}\}$ for Upd for the given (H, lin) and aborts if no such entry is stored. Otherwise, $\mathcal{S}_{12}$ follows the protocol of the user from GAME$_{11}(\kappa)$.

Lemma 8.1.12. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{11}(\kappa)$ from GAME$_{12}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{11}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{12}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. First, note that honest user leaking information does not change anything in the distribution of sent messages, which makes it impossible for the environment $\mathcal{Z}$ to distinguish based on that change. So the only change that *could* be detected by the environment is the simulation of the Upd task based only on the data accessible to the simulator and not on the entire data stored by each user. To that end, we need to show that the simulator can report the same messages in GAME$_{12}(\kappa)$ that the user would have sent in GAME$_{11}(\kappa)$.

During the Upd-tasks with an honest user in game GAME$_{11}(\kappa)$ the following tasks are now performed by $\mathcal{S}_{12}$:

- Sending lin to the helper and operator. This is easily possible as lin is leaked by the user.
- Proving correct rerandomization using of the UH using Λ_{Update} from Fig. 7.22. As the ZK proof is forged since GAME$_5(\kappa)$ this is trivially possible.
- Finding out whether $\vec{\alpha}$ and $\vec{s}$ contain non trivial values and then either proving correct transfer of both results to the UH using the ZK language $\Lambda_{\text{Transfer}}$ from Fig. 7.13 and setting $\mathsf{com}_{UH^{(1)}}$ and $\mathsf{com}_{UH^{(2)}}$ to be zero-commitments, or setting the proof and both of $\mathsf{com}_{UH^{(1)}}$ and $\mathsf{com}_{UH^{(2)}}$ to be $\perp$.

As both $\vec{\alpha}$ and $\vec{s}$ are stored (since $\textsc{Game}_3(\kappa)$ and correctly fetched the check can be performed by the simulator.

- Provide commitments for $(\mathsf{com}_{UH}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{com}_{ser_0})$. Those are all commitments on the zero-vector since Section 8.1 and as such can be created without knowing the users data. Note that this is no problem for the ZK proofs as they are also simulated due to $\textsc{Game}_7(\kappa)$.
- Provide the old serial number ser. Note here that the old serial number is *independent* of the proof (as that is simulated) and can be drawn uniformly at random by the simulator (since $\textsc{Game}_7(\kappa)$) without the environment noticing.
- Creation of the logbook:

  After the simulator received the message with the UI from the operator and has compared them with the additional data stored since $\textsc{Game}_3(\kappa)$ $\mathcal{S}_{12}$ can follow the honest users protocol and abort whenever the honest user would. So we have to show that $\mathcal{S}_{12}$ aborts in $\textsc{Game}_{12}(\kappa)$ iff U aborts in $\textsc{Game}_{11}(\kappa)$.

  Since in both $\textsc{Game}_{11}(\kappa)$ and $\textsc{Game}_{12}(\kappa)$, the helper H is assumed to be honest[1], the values sent to the user are correct. Those values are exactly the same as the ones that $\mathcal{S}_{12}$ has stored. Hence, the simulator already has the user's view on those values. Additionally, the simulator sees messages exchanged between parties and as such has access to ct_{y_U}. Since the simulator essentially follows the honest protocol with the same data from here on, the abort-criteria remain equivalent and cannot be used to distinguish.

□

$\textsc{Game}_{13}(\kappa)$: Introduces incorruptible entity $\mathcal{F}_{\mathsf{BKA}}$ that follows the specification from Figs. 6.1 and 6.2 into the experiment. The entity is only accessible by honest participants and the simulator through subroutine input/output tapes.

Lemma 8.1.13. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{12}(\kappa)$ from $\textsc{Game}_{13}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{12}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{13}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Since there is no direct link between $\mathcal{Z}$ and $\mathcal{F}_{\mathsf{BKA}}$ there is no way for corrupted parties (which are controlled by $\mathcal{Z}$) to notice the change. Honest parties still only act according to the protocol from $\textsc{Game}_{12}(\kappa)$, which does not contain any interaction with $\mathcal{F}_{\mathsf{BKA}}$. Hence, there is no way for $\mathcal{Z}$ to distinguish the two games; every action any honest party takes in $\textsc{Game}_{12}(\kappa)$ is equivalent to their actions in $\textsc{Game}_{13}(\kappa)$ and the corrupted parties act entirely independent of $\mathcal{F}_{\mathsf{BKA}}$. □

[1] As we assume the operator to be corrupted and both cannot be corrupted at once.

$\textsc{Game}_{14}(\kappa)$: Replaces the TSA T with a semi-dummy party T'. The party immediately forwards its input to the ideal functionality $\mathcal{F}_{\mathsf{BKA}}$ but also leaks x_{T} to $\mathcal{S}_{14}$ after receiving it. All interactions of T are then simulated by $\mathcal{S}_{14}$ by following the original protocol.

Lemma 8.1.14. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{13}(\kappa)$ from $\textsc{Game}_{14}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{13}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{14}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability easily follows from the fact that the TSA T leaks the relevant part of its input to the simulator who performs the same code; hence, the simulator can execute the protocol of T perfectly. Concretely, during the Init-task, the only input to T is $(\mathtt{Initialize})$. As the key generation algorithm for the signature scheme is public knowledge this can be performed by $\mathcal{S}_{14}$. For the Sign Function Parameter-task, the only inputs are $(\mathtt{SignFP}, x_{\mathsf{T}})$, where x_{T} is leaked to the simulator. The rest of the protocol can be simulated when knowing k_{T} (which $\mathcal{S}_{14}$ does from simulation of the Init-task).

Thus, the distribution visible by $\mathcal{Z}$ is identical and the game hop is purely cosmetically. Our claim follows. □

$\textsc{Game}_{15}(\kappa)$: This hybrid game expands the state of the simulator $\mathcal{S}_{15}$ by a list of tuples $\{\mathsf{f}, fp, \mathsf{com}_{fp}, \sigma_{fp}, \ell\}$, where an entry means that fp can be used for f with a signature σ_{fp} on $(\mathsf{com}_{fp}, \mathsf{f})$ and they were entered as the ℓ-th FP tuple. During simulation of the task SFP the simulator stores a new tuple in case an input fp can be used for a function f. In that case, the simulator computes the signature σ_{fp} on $(\mathsf{f}, \mathsf{com}_{fp})$ and stores $(\mathsf{f}, fp, \mathsf{com}_{fp}, \sigma_{fp}, \ell)$.

The user U is replaced by a new user U', which skips verification of the signature on the FP and instead asks the simulator $\mathcal{S}_{15}$ if the given values for $(\mathsf{f}, fp, \mathsf{com}_{fp}, \sigma_{fp}, \ell)$ are valid. Instead of manually verifying the signature, $\mathcal{S}_{15}$ verifies that there is an entry $(\mathsf{f}, fp, \mathsf{com}_{fp}, \sigma_{fp}, \ell)$ and aborts if there is none.

Lemma 8.1.15. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{14}(\kappa)$ from $\textsc{Game}_{15}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{14}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{15}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability follows from the EUF-CMA security of the used signature scheme: Let $\mathcal{Z}$ be an environment that distinguishes between $\textsc{Game}_{14}(\kappa)$ and $\textsc{Game}_{15}(\kappa)$ with probability $1/2 + \alpha$ with *non-negligible* advantage α. From $\mathcal{Z}$, we construct an adversary $\mathcal{A}$ on the EUF-CMA property of the signature scheme $\textsc{Sig}$. Let C be the EUF-CMA challenger. C provides a signature oracle to $\mathcal{A}$. $\mathcal{A}$ flips a random coin and on heads simulates $\textsc{Game}_{14}(\kappa)$ and on tails $\textsc{Game}_{15}(\kappa)$. During simulation, $\mathcal{A}$ creates all secrets honestly, except for k_{T} and vk_{T}. On every execution of the SFP-task, whenever $\mathcal{A}$ is supposed to sign $(\mathsf{f}, \mathsf{com}_{fp})$ for O using k_{T}, $\mathcal{A}$ sends $(\mathsf{f}, \mathsf{com}_{fp})$ to the signature-oracle and uses the result as signature.

Now note that we assume that $\mathcal{Z}$ successfully distinguishes the two games $\textsc{Game}_{14}(\kappa)$ and $\textsc{Game}_{15}(\kappa)$ notably better than by guessing. Since the only difference is in the way signatures are handled, any distinguishing attack would require $\mathcal{Z}$ to input some distinguishing commitment $\mathtt{com}_{fp}$ on some FPs (fp) into the game that can be used to determine which game it is playing.

First, note that if the signature on $\mathtt{com}_{fp}$ is accepted in $\textsc{Game}_{15}(\kappa)$, it is also accepted in $\textsc{Game}_{14}(\kappa)$, as a stored entry $(\mathsf{f}, \mathtt{com}_{fp}, \sigma_{fp})$ implies that the SFP task has been called with input $(fp, \mathtt{com}_{fp})$ successfully and yielded signature σ_{fp}. However, the other way is not as clear; the only differing behavior that can be caused (and used by $\mathcal{Z}$ to detect the change) is by preparing some tuple $(\mathtt{com}_{fp}, \sigma_{fp})$ that is rejected in $\textsc{Game}_{15}(\kappa)$ but accepted in $\textsc{Game}_{14}(\kappa)$. Clearly, the latter implies that the signature σ_{fp} on $(\mathsf{f}, \mathtt{com}_{fp})$ is valid. The former, however, implies that $\mathcal{Z}$ never called the Sign Function Parameter-task on fp for function f in the name of O. This can only happen if (1) $\mathcal{A}$ never called the challenge oracle on input $(\mathsf{f}, \mathtt{com}_{fp})$, and (2) the signature provided by $\mathcal{Z}$ on $(\mathsf{f}, \mathtt{com}_{fp})$ still verifies under the verification key of the TSA. Taking both together makes this a valid forgery, with which $\mathcal{A}$ can break the EUF-CMA property of the signature scheme. As this is not possible by requirement our claim follows. □

Game$_{16}(\kappa)$**:** This hybrid game ensures that the operator calls the init task for $\mathcal{F}_{\mathsf{BKA}}$ by using the information extracted from the $\mathcal{F}_{\mathsf{Reg}}$ setup functionality. During the Init task the operator is supposed to insert a message $(\mathtt{Register}, \mathsf{vk_O})$ that implicitly contains the pid pid_{O} of the operator. As this is simulated by $\mathcal{S}_{16}$ the simulator follows the procedure of $\mathcal{F}_{\mathsf{Reg}}$ correctly and, if it succeeded, calls $\mathcal{F}_{\mathsf{BKA}}$ with input $(\mathtt{Initialize})$ in the name of O.

Lemma 8.1.16. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{15}(\kappa)$ from $\textsc{Game}_{16}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{15}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{16}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The change induced in this game only *activates* the functionality $\mathcal{F}_{\mathsf{BKA}}$. Since, at this point, it is not accessed by any honest party and neither the environment, nor the dummy adversary can access $\mathcal{F}_{\mathsf{BKA}}$, indistinguishability between the two games trivially follows. □

Game$_{17}(\kappa)$**:** Replaces all honest user U^* by dummy parties that immediately forward their input to the ideal functionality $\mathcal{F}_{\mathsf{BKA}}$ with the additional property, that they still leak the linking number lin during the Update-task (see $\textsc{Game}_{12}(\kappa)$) and still call $\mathcal{F}_{\mathsf{MPC}}$ with honest inputs when demanded by the protocol; the remaining protocol parts are executed just as specified in the protocol by the simulator $\mathcal{S}_{17}$.

$\mathcal{S}_{17}$ also controls input to $\mathcal{F}_{\mathsf{BKA}}$ for corrupted users during simulation of the tasks OS and Upd. During the Outsource-task, $\mathcal{S}_{17}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of U with empty input after the successful exchange of the linking number lin. In the Update-task, $\mathcal{S}_{17}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of U with input $(\mathtt{Update})$ after receiving the first message.

Lemma 8.1.17. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{16}(\kappa)$ from $\textsc{Game}_{17}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{16}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{17}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. First, note that honest user forwarding input to $\mathcal{F}_{\mathsf{BKA}}$ cannot be detected by and environment as the subfunctionality input and output tapes are inaccessible to the environment. Hence, this change is impossible to detect for any environment $\mathcal{Z}$.

We claim indistinguishability of the remaining changes based on the fact that the simulator $\mathcal{S}_{17}$ sends exactly the same messages that an honest user would in $\textsc{Game}_{16}(\kappa)$. Using the leaks, the simulator in $\textsc{Game}_{17}(\kappa)$ can create every message that the user would have sent in $\textsc{Game}_{16}(\kappa)$, as we will show now:

User Registration. Here, the user has no secret input; the simulator can safely draw a random key similar to the real user and respond to calls to $\mathcal{F}_{\mathsf{Reg}}$ with pk_{U}. The verification step is only based on the messages sent in this interaction, which are accessible to the simulator due to simulation of $\mathcal{F}_{\mathsf{ORR}}$. Thus, $\mathcal{S}_{17}$ can abort in $\textsc{Game}_{17}(\kappa)$ whenever U aborts in $\textsc{Game}_{16}(\kappa)$.

Bookkeeping. The first message that $\mathcal{S}_{17}$ has to send here in the name of U is $(\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, ser^{*}, \mathsf{com}_{ser}, \pi)$.

Since $\textsc{Game}_{7}(\kappa)$ the commitments are commitments on the all-zero vector and hence independent of the users secret inputs. As such they can be created by $\mathcal{S}_{17}$. Since $\textsc{Game}_{5}(\kappa)$ the ZK proofs are simulated using the simulation-trapdoor td_{sim} instead of the actual witness, meaning that $\mathcal{S}_{17}$ can forge the proof as long as it knows the statement; which is $(\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}, ser, \mathsf{com}_{ser}, \mathsf{vk}_{\mathsf{O}})$. The commitments were chosen by the simulator as zero-commitments, the old serial number is independent of both com_{UH} and π and hence was drawn uniformly at random, and the verification key vk_{O} is known due to simulation of $\mathcal{F}_{\mathsf{Reg}}$.

The second proof in case of a non-trivial permutation $\vec{\alpha}$ or update $\vec{s}$ is returned from the simulation of $\mathcal{F}_{\mathsf{MPC}}$ for the function f can be simulated similarly: Both $\mathsf{com}_{UH^{(1)}}$ and $\mathsf{com}_{UH^{(2)}}$ are zero-commitments, and π^{*} is created once again by using the simulation trapdoor. The statement here contains $(\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ which are all known to the operator; the commitments are zero-commitments and $\vec{\alpha}$ and $\vec{s}$ are known due to the simulation of $\mathcal{F}_{\mathsf{MPC}}$. And in case of empty $\vec{\alpha}$ and $\vec{s}$ the simulator sets all three to $\perp$. So the second message, $(\mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \pi^{*})$, can be simulated as well. The latter is obtained by simulation of $\mathcal{F}_{\mathsf{MPC}}$ and hence consistent with the operators view. Hence $\mathcal{S}_{17}$ can reconstruct the statement of the second proof according to $\Lambda_{\text{Transfer}}$ which proves that the output of $\mathcal{F}_{\mathsf{MPC}}$ has been transferred to the UH accordingly.

Finally, the simulator has to perform the verification step. $\mathcal{S}_{17}$ always used zero-vectors for shares and can hence verify the values received from the operator. This is done by executing the honest protocol, as all commitments and the signature are known to $\mathcal{S}_{17}$.

Outsource. Since $\textsc{Game}_{11}(\kappa)$ the shares for the OS-task are created by the simulator. Since $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_7(\kappa)$ the simulator also creates the commitments and the zero-knowledge proof. The serial is again drawn at random since $\textsc{Game}_7(\kappa)$ as it is independent of $\mathtt{com}_{ser^*}(= \mathtt{com}_{\vec{0}})$. Hence, the first message, $(rs_{UH}^{(\mathrm{O})}, rs_{x_{\mathrm{U}}}^{(\mathrm{O})}, rs_{otp}^{(\mathrm{O})}, ser^*, \mathtt{com}_{ser}, \mathtt{com}_{ser_0}, \mathtt{com}_{\mathsf{sk}_{\mathrm{U}}}, \pi)$, is indistinguishable from the one in $\textsc{Game}_{16}(\kappa)$.

Verification, again, is only dependent on what the simulator knows already and hence can be simulated by executing the honest protocol.

Update. The Upd-task has been simulated already since $\textsc{Game}_{12}(\kappa)$.

For security against a corrupted user U our definition of $\mathcal{F}_{\mathsf{BKA}}$ requires $\mathcal{S}_{17}$ to call $\mathcal{F}_{\mathsf{BKA}}$ in the name of U only during the Outsource-task and not during any of the other tasks:

- In **UReg**, all that would change is that $\mathcal{F}_{\mathsf{BKA}}$ adds U to the list of known users, which is never checked against corrupted users. Interaction with $\mathcal{F}_{\mathsf{Reg}}$ might still take place, though, but that one is simulated since $\textsc{Game}_2(\kappa)$.
- In the **BK**-task the environment essentially talks to itself. Updates to the UH can be done by $\mathcal{Z}$ without access to $\mathcal{F}_{\mathsf{BKA}}$ as the operator can sign any UH and thus create a valid new logbook λ. Simulation of $\mathcal{F}_{\mathsf{MPC}}$ occurs outside of the actual protocol since $\mathcal{S}_{17}$ plays $\mathcal{F}_{\mathsf{MPC}}$ since $\textsc{Game}_2(\kappa)$.
- For **OS** $\mathcal{F}_{\mathsf{BKA}}$ stores the corrupted user's information in a list, alongside that of honest users, and which are used later for the Outsourced Analytics-task. There, it does not make a difference if the corrupted user's UH is input to $\mathcal{F}_{\mathsf{BKA}}$ directly, or if it is equivocated during simulation of the subsequent OA task.
- During **OA** the simulator receives input from O to $\mathcal{F}_{\mathsf{MPC}}$ which contains the shares the corrupted user U prepared for the operator: $(\mathtt{Compute}, \mathsf{f}, (fp, \mathtt{unv}_{fp}, \{rs_{UH}^{(\mathrm{O})}, rs_{x_{\mathrm{U}}}^{(\mathrm{O})}, rs_{otp}^{(\mathrm{O})}\}_{\zeta=1}^{Z_{\mathsf{f}}}))$ The respective shares of the helper were already stored since $\textsc{Game}_9(\kappa)$ for the given helper during simulation of the OS-task. Hence all shares can be reconstructed by $\mathcal{S}_{17}$ using the protocol $\mathtt{Combine}$. This value is equivalent to the input corrupted users that would have input to $\mathcal{F}_{\mathsf{BKA}}$. So $\mathcal{S}_{17}$ can provide $\mathcal{F}_{\mathsf{BKA}}$ with the correct tuples (UH, x_{U}) for corrupted users.

  Correctness trivially follows as the simulator in $\textsc{Game}_{17}(\kappa)$ has, at this point, exactly the same values that are shared between H and O in $\textsc{Game}_{16}(\kappa)$. Hence, any attempt by $\mathcal{Z}$ to manipulate data that would have worked in $\textsc{Game}_{16}(\kappa)$ also works in $\textsc{Game}_{17}(\kappa)$ and vice versa, making it impossible for any PPT-environment $\mathcal{Z}$ to distinguish.
- During the **Upd**-task corrupted users only obtain masked values from the helper. This step does not have any consequences for further interaction as $\mathcal{F}_{\mathsf{BKA}}$ only loads and returns data that is not accessed at any further point throughout

the lifetime of the system. Hence, it suffices to call $\mathcal{F}_{\mathsf{BKA}}$ in the name of *any* corrupted user to have it deliver output to the helper.

□

GAME$_{18}(\kappa)$: Replaces the helpers H with dummy parties that forward their input to the ideal functionality $\mathcal{F}_{\mathsf{BKA}}$. The remaining messages that the helpers sent will be simulated by the simulator by honestly following the protocol of H from GAME$_{17}(\kappa)$.

Lemma 8.1.18. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{17}(\kappa)$ from GAME$_{18}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{17}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{18}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Intuitively, the helper can be easily simulated as it does not have any secret inputs. It only obtains information via robust shares of the user. At this point, the simulator $\mathcal{S}_{18}$ already simulates all the messages sent by the user and hence can act as the honest helper would by following the protocol from GAME$_{17}(\kappa)$.

In more detail, the situation for an *honest* user is as follows:

Outsource. Since the helper is now played by the simulator who also sends all messages on behalf of the user, all messages from user to helper and vice versa can be ignored. This automatically resolves most of the Outsource-task; the only interaction between the helper and the operator that has to be simulated is during a Blum coin toss to create *lin*, which does not depend on any secret inputs at all. This can honestly be executed by $\mathcal{S}_{18}$.

Managing the stored data for OA is also trivially possible; all the values stored there come from the user, who is played by the simulator at this point, anyway. Correctness of the values follows from the fact that $\mathcal{S}_{18}$ only stores information there in GAME$_{18}(\kappa)$ when the honest helper H in GAME$_{17}(\kappa)$ would.

Outsourced Analytics. In GAME$_{17}(\kappa)$, the helper loads the first Z_f entries stored for OA during OS before calling $\mathcal{F}_{\mathsf{MPC}}$. We already argued for the Outsource-task that the information $\mathcal{S}_{18}$ stores for OA in GAME$_{18}(\kappa)$ is equivalent to what H stores during GAME$_{17}(\kappa)$; hence, $\mathcal{S}_{18}$ can simulate by following the honest protocol.

Storing the UI for Upd after the simulation of $\mathcal{F}_{\mathsf{MPC}}$ is not required anymore, as it only contains information that the simulator can infer from the own state.

Update. The protocol for the Upd-task basically consists of two mostly disjoint parts: the interaction between the user and the helper and the interaction between the user and the operator.

The latter is independent of anything the helper does. The former is independent of anything the environment $\mathcal{Z}$ (playing the operator) does and has not to be simulated, as $\mathcal{Z}$ is unable to read messages exchanged between honest parties; in this case the environment only sees ciphertexts which do not need to contain valid cleartext data.

If the user is corrupted the behavior for the tasks for OS and Upd change as follows:

Outsource. In this task, everything works by following the protocol of the helper honestly. The simulator receives the shares, which are sent from U to H via $\mathcal{F}_{\mathsf{ORR}}$ (and hence visible to the simulator). Those can be stored by the simulator for OA.

Since the simulator now also performs the Blum coin toss, $\mathcal{S}_{18}$ knows lin and can send it to the user directly.

Update. The only interaction that has to be simulated is the part where the helper, after receiving lin from a user via $\mathcal{F}_{\mathsf{ORR}}$, responds to that user via $\mathcal{F}_{\mathsf{ORR}}$ with $(ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathrm{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}})$ via $\mathcal{F}_{\mathsf{ORR}}$.

During simulation of the task for OA the simulator stored the clear values for Upd alongside the OTP. From that $\mathcal{S}_{18}$ can compute the OTP-encryptions and use them for the message. Since lin is obtained from the user through the first message, the helper in $\text{Game}_{17}(\kappa)$ will send exactly the same message as the simulator in $\text{Game}_{18}(\kappa)$.

This shows that the two games are indistinguishable for all PPT-environments $\mathcal{Z}$. □

Game$_{19}(\kappa)$: The simulator now enforces that for every task, $\mathcal{F}_{\mathsf{BKA}}$ is called by the operator with the correct inputs. This causes $\mathcal{F}_{\mathsf{BKA}}$ to have input from all parties, which means that it behaves according to its definition and provides leaks and output to the parties and $\mathcal{S}_{19}$, which the simulator can use. Thus, $\mathcal{S}_{19}$ changes the simulation of $\mathcal{F}_{\mathsf{MPC}}$: Instead of executing f with the appropriate inputs the simulator now uses the inputs received from operator to $\mathcal{F}_{\mathsf{MPC}}$ in order to call $\mathcal{F}_{\mathsf{BKA}}$ in the name of O. The leaks obtained by $\mathcal{F}_{\mathsf{BKA}}$ are then used as output of $\mathcal{F}_{\mathsf{MPC}}$.

When executing `Combine` with an honest user and executing the OA-task with arbitrary user corruption, the simulation of $\mathcal{F}_{\mathsf{MPC}}$ also includes a consistency check for the operator's input: If reconstruction with share via `Combine` fails, the simulator aborts. The operator shares are obtained via input to $\mathcal{F}_{\mathsf{MPC}}$, the helper shares are fetched from the values stored during OS for OA.

When $\mathcal{F}_{\mathsf{BKA}}$ asks $\mathcal{S}_{19}$ for updated inputs $(UH, x_{\mathsf{U}})_{\zeta}$ for corrupted users during the OA-task, $\mathcal{S}_{19}$ uses `Combine` to reconstruct those values using the shares that the user created during Outsource. The helper-shares, $(rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})})$, were stored already. The operator-shares, $(rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})})$, are taken from the input that O sent to $\mathcal{F}_{\mathsf{MPC}}$.

The inputs of the operator to both the OS and Upd task do not contain any secrets, so $\mathcal{S}_{19}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of the operator after seeing the first message of an OS task and immediately after the start of the simulation of the Upd task.

Finally, the TSA is replaced by a *real* dummy party that does not leak anything to the simulator. Instead of letting the simulator compute the function that states whether the FPs are valid or not, the computation is now done by the ideal functionality based on the input of both parties; the simulator only gets the result.

Lemma 8.1.19. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing* $\text{GAME}_{18}(\kappa)$ *from* $\text{GAME}_{19}(\kappa)$ *fulfills:*

$$|\Pr[\mathcal{Z}(\textit{GAME}_{18}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textit{GAME}_{19}(\kappa)) = 1]| \in \text{negl}(\kappa)$$

Proof. Indistinguishability of the first change, namely that computations of f are replaced by leaks from $\mathcal{F}_{\mathsf{BKA}}$, follows from the following facts:

1. In both hybrid games, the result of the computation is based on the output of f. The simulator performs this computation in $\text{GAME}_{18}(\kappa)$. The functionality $\mathcal{F}_{\mathsf{BKA}}$ is, by UC-conventions, modeled as an incorruptible entity and hence performs the same honest computation. Hence, both parties compute f honestly.

2. Both parties, $\mathcal{F}_{\mathsf{BKA}}$ in $\text{GAME}_{19}(\kappa)$ and the simulator in $\text{GAME}_{18}(\kappa)$, use exactly the same input. Until the point where the interaction takes place, the simulators in both $\text{GAME}_{18}(\kappa)$ and $\text{GAME}_{19}(\kappa)$ behave equivalently, leading to an identical view for $\mathcal{Z}$. From there on, the simulator in $\text{GAME}_{18}(\kappa)$ performs the computation of $\mathsf{f}(\cdot)$ directly and obtains the outputs for $\mathcal{F}_{\mathsf{MPC}}$. In $\text{GAME}_{19}(\kappa)$, $\mathcal{S}_{19}$ forwards *the same* input to $\mathcal{F}_{\mathsf{BKA}}$ (which is possible because both have the same interface). The functionality then uses this input in order to compute f. The outputs are leaked to $\mathcal{S}_{19}$ via functionality output, who now has exactly the same values as in $\text{GAME}_{18}(\kappa)$ and can continue equivalently.

Hence, the parts involving interaction with $\mathcal{F}_{\mathsf{MPC}}$ remain equivalent, as essentially the output data that $\mathcal{S}_{19}$ obtains in $\text{GAME}_{19}(\kappa)$ via leakage has been created similarly to the output that the simulator computed in $\text{GAME}_{18}(\kappa)$.

We now have to show that the abort-criteria remain equivalent. In $\text{GAME}_{19}(\kappa)$, $\mathcal{S}_{19}$ aborts if the input of the operator to $\mathcal{F}_{\mathsf{MPC}}$ differs from the shares that the user sent to the operator during the OS-task. In $\text{GAME}_{18}(\kappa)$, the user aborts via $\mathcal{F}_{\mathsf{MPC}}$, as shares that were changed by the operator would be recognized as forgery due to `Verify` with overwhelming probability.

Next, we have to prove indistinguishability of the equivocation step for corrupted users. To that end, we claim that the updated input shares that $\mathcal{S}_{19}$ inputs to $\mathcal{F}_{\mathsf{BKA}}$ are the same that were shared during the OS-task and that would have been used by the helper and operator in a real execution. The helper-shares were treated similar to a real execution, as the simulator stored the message that was received during the OS-task for OA. The operator shares were taken from the input that the operator sent to $\mathcal{F}_{\mathsf{MPC}}$ and are hence also visible to the simulator. Hence, the same values that would have been taken as input for f in $\text{GAME}_{18}(\kappa)$ are also taken by $\mathcal{F}_{\mathsf{BKA}}$ in $\text{GAME}_{19}(\kappa)$. As mentioned above, $\mathcal{F}_{\mathsf{BKA}}$ computes f just as was done in previous games, so the outputs stay the same and hence no environment can distinguish.

Finally, note that the change regarding SFP is only cosmetically as once again, the same code is executed on the same data but by different parties which both are guaranteed to stick to the protocol. □

$\textsc{Game}_{20}(\kappa)$: Instead of relying on the leaked input x_{T} and the received data from the operator to compute whether or not the FP *fp* should be accepted during the SFP task, $\mathcal{S}_{20}$ accepts the FPs only if the functionality returns $(\mathtt{Ok})$ to the operator after both parties sent their inputs. The Trusted Signing Authority is replaced with a genuine dummy party which does not leak x_{T} to the simulator anymore.

Lemma 8.1.20. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{19}(\kappa)$ from $\textsc{Game}_{20}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{19}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{20}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Again this gamehop is only cosmetically as the same code still is executed on the same inputs, but from a different machine. In $\textsc{Game}_{19}(\kappa)$ the simulator obtains x_{T} from the (semi-)dummy TSA T^* and $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \mathsf{f}, x_{\mathsf{O}})$ from the corrupted operator and then performs some consistency checks regarding the FPs before evaluating f. In $\textsc{Game}_{20}(\kappa)$ the simulator only obtains $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \mathsf{f}, x_{\mathsf{O}})$ from the corrupted operator and then performs the same consistency checks regarding the FPs before inputting $(\mathtt{SignFP}, fp, \mathsf{f}, x_{\mathsf{O}})$ to $\mathcal{F}_{\mathsf{BKA}}$. The input from the TSA—namely x_{T}—was already forwarded by the honest dummy party to $\mathcal{F}_{\mathsf{BKA}}$ so the inputs are the same.

Indistinguishability thus follows directly. □

$\textsc{Game}_{21}(\kappa)$: Instead of relying on the leak of *lin* send by the semi-dummy user U during simulation of the Upd task for honest users, $\mathcal{S}_{21}$ now uses the leaks on the *ssid* provided by $\mathcal{F}_{\mathsf{BKA}}$ to infer the correct linking number. The honest users are replaced by dummy users, *i.e.* they only forward their input obtained by $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{BKA}}$. The remaining interactions with the operator are simulated using the simulator's knowledge. Therefore, the simulator looks up the list entry $(ssid, lin)$ when receiving the ssid to get the correct linking number *lin* during the Upd-task.

Lemma 8.1.21. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{20}(\kappa)$ from $\textsc{Game}_{21}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{20}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{21}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The situation only changes during Upd-tasks with an honest user, where the simulator $\mathcal{S}_{21}$ in the honest-user setting does not receive the leak *lin* from the user, but instead the ssid *ssid* of the respective OS-instance from $\mathcal{F}_{\mathsf{BKA}}$. Since the simulator updated the list with tuples $(ssid, lin)$ correctly during the simulation of the OS-task, $\mathcal{S}_{21}$ can obtain the same linking number *lin* in $\textsc{Game}_{21}(\kappa)$ that the user has sent in $\textsc{Game}_{20}(\kappa)$.

Given that $\mathcal{F}_{\mathsf{BKA}}$ is, by definition, incorruptible, it will always send the correct ssid to the simulator. During the OS-task, the linking number *lin* was honestly created by the simulator $\mathcal{S}_{21}$ in an interaction with the operator. This number is used by all parties as linking number and is stored just as honest parties would store it. During the Upd-task, an honest user would look up this linking number, which can be simulated by following the program code of the user and fetching it from the list of ssid and linking number tuples.

Hence, there is no way that the linking number an honest user would store during an OS and later reveal during an Upd task in $\textsc{Game}_{20}(\kappa)$ would differ from the linking number that the simulator stores (and later reveals) during the simulation of $\textsc{Game}_{21}(\kappa)$. □

The final game, 8.1, corresponds to our ideal world. Since we have shown that no (PPT) environment $\mathcal{Z}$ can differentiate this from the real execution in $\textsc{Game}_1(\kappa)$, our corollary follows:

Corollary 8.1.22 (User Security)**.** *For all environments $\mathcal{Z}$ who statically corrupted the operator, it follows that*

$$\Pi_{BKA}^{\mathcal{F}_{\mathrm{CRS}},\mathcal{F}_{\mathrm{Reg}},\mathcal{F}_{\mathrm{SMT}},\mathcal{F}_{\mathrm{MPC}},\mathcal{F}_{\mathrm{ORR}}} \geq \mathcal{F}_{\mathrm{BKA}}$$

when using building blocks as described in Section 5.3.

We have shown in Lemma 8.1.2 to Lemma 8.1.21, that under static corruption of the operator, the simulator $\mathcal{S}$ acting in the ideal world can provide a view for $\mathcal{Z}$ that is indistinguishable from a real execution of the protocol:

$$\mathtt{view}_{\mathcal{Z},\mathcal{A},\Pi_{BKA}} \approx^c \mathtt{view}_{\mathcal{Z},\mathcal{S},\mathcal{F}_{\mathrm{BKA}}}$$

8.2. Operator Security

This section contains an investigation of the remaining corruption scenarios, namely the ones that are relevant to maintain privacy of an honest *operator*. That is, we consider scenarios where the any subset of users and helpers can be corrupted and present a simulator, which provides a view in the ideal world that cannot be distinguished from a real-world execution. The simulator is given in Figs. 8.11 to 8.25.

For our proof, we consider the following hybrid games $\textsc{Game}_i(\kappa)$:

$\textsc{Game}_1(\kappa)$: The first hybrid is equivalent to the real experiment. That is,

$$\textsc{Game}_1(\kappa) \coloneqq \mathtt{view}_{\Pi_{BKA},\mathcal{S}_1,\mathcal{Z}}(1^\kappa)$$

This means that all parties execute the real protocol.

$\textsc{Game}_2(\kappa)$: All calls to the setup functionalities $(\mathcal{F}_{\mathrm{CRS}},\mathcal{F}_{\mathrm{Reg}},\mathcal{F}_{\mathrm{SMT}},\mathcal{F}_{\mathrm{MPC}},\mathcal{F}_{\mathrm{ORR}})$ are replaced by calls to $\mathcal{S}_2$, who simulates their behavior by following the description of the ideal functionality.

Lemma 8.2.1. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_1(\kappa)$ from $\textsc{Game}_2(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_1(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_2(\kappa)) = 1]| \in \mathrm{negl}(\kappa)$$

Simulator $\mathcal{S}_{BKA}$

State of a simulator $\mathcal{S}_{BKA}$ and setup against a corrupted user.

The simulator stores:

- td_{ext}: Trapdoor for the zero knowledge scheme.
- $(\mathsf{vk}_\mathsf{O}, \mathsf{k}_\mathsf{O})$: Signature key-pair of the operator
- $(\mathsf{vk}_\mathsf{T}, \mathsf{k}_\mathsf{T})$: Signature key-pair of the sign party
- List of observed serials $\{ser\}$
- List of tuples $\{\mathsf{f}, \ell, \mathtt{com}_{fp}, \sigma_{fp}\}$ of functions, indices for function parameters and commitment and signature to be used for a given function f.
- List of tuples $(\mathsf{pk}_\mathsf{U}, pid_\mathsf{U})$.
- List of tuples $\{pid_\mathsf{H}, \mathsf{f}, lin, rs^{(\mathsf{H})}_{UH}, rs^{(\mathsf{H})}_{x_\mathsf{U}}, rs^{(\mathsf{H})}_{otp}\}$.
- List of tuples $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathrm{unv}_{\vec{a}}}, ct_{y_\mathsf{U}}\}$.
- List of tuples $\{pid_\mathsf{H}, \mathsf{f}, lin, rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{otp}\}$.
- List of tuples $\{lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_\mathsf{U}}\}$.
- List of tuples $\{ssid, lin\}$.

On input `Setup`, $\mathcal{S}_{BKA}$ sets up the Common Reference String using $(crs, td_{ext}) \leftarrow \textsc{Zk}.\mathrm{SetupExt}(1^\kappa)$ and stores the extraction trapdoor td_{ext}. Now $\mathcal{S}_{BKA}$ starts simulating all the hybrid functionalities.

Figure 8.11.: The first part of the simulator with an honest operator: Defines state and set up.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the initialization against a corrupted user.

On input $(\mathtt{Initialize}, pid_\mathsf{O})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the honest protocol of O and creates and stores a signature key pair $(\mathsf{k}_\mathsf{O}, \mathsf{vk}_\mathsf{O}) \leftarrow \textsc{Sig}.\mathrm{KeyGen}(1^\kappa)$.

On input $(\mathtt{Initialize}, pid_\mathsf{T})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ generates a signature key pair $(\mathsf{k}_\mathsf{T}, \mathsf{vk}_\mathsf{T}) \leftarrow \textsc{Sig}.\mathrm{KeyGen}(1^\kappa)$ and follows the protocol of T.

Figure 8.12.: The second part of the simulator with an honest operator and corrupt user: Defines initialization.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\mathtt{SignFP}$ task against an honest operator and TSA.

On input $(\mathtt{SignFP}, pid_{\mathsf{O}})$ and $(\mathtt{SignFP}, pid_{\mathsf{T}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after having received leak (f, ℓ) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ computes zero-commitments $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}) \leftarrow \textsf{Com}.\mathsf{Com}(\vec{0})$ and a signature $\sigma_{fp} \leftarrow \textsf{Sig}.\mathsf{Sign}_{(\mathsf{k}_{\mathsf{T}})}(\mathsf{f}, \mathtt{com}_{fp})$. Then $\mathcal{S}_{BKA}$ stores $(\mathsf{f}, \ell, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$.

Figure 8.13.: The third part of the simulator with an honest operator and TSA: Defines behavior for signing function parameter.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\mathtt{UserRegistration}$ task against an honest user and operator.

On input $(\mathtt{UserRegistration}, pid_{\mathsf{U}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and $(\mathtt{UserRegistration}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the protocol of an honest user to create the public key pk_{U} in order to use that to respond to calls of the form $(\mathtt{Fetch}, pid_{\mathsf{U}})$ for $\mathcal{F}_{\mathsf{Reg}}$.

Figure 8.14.: The fourth part of the simulator with an honest operator and honest user: Defines user registration.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\mathtt{UserRegistration}$ task against a corrupted user.

On input $(\mathtt{UserRegistration}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and $(\pi, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathtt{com}_{ser_0})$ from U to O, $\mathcal{S}_{BKA}$ follows the protocol of O: It fetches the public key pk_{U} from the simulated $\mathcal{F}_{\mathsf{Reg}}$, sets $stmt := (\mathsf{pk}_{\mathsf{U}}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}})$ and aborts if there is no key pk_{U} stored or if $\textsf{Zk}.\mathsf{Verify}(\pi, stmt, \Lambda_{\mathrm{UserReg}}) \neq 1$. Additionally, $\mathcal{S}_{BKA}$ aborts if an entry $(\mathsf{pk}_{\mathsf{U}}, \cdot)$ has been stored before and otherwise stores an entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$. Now $\mathcal{S}_{BKA}$ sends input $(\mathtt{UserRegistration})$ in the name of U corresponding to pid_{U} to $\mathcal{F}_{\mathsf{BKA}}$.

When receiving (Ok) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the remaining protocol specification of the operator to create the commitments $\mathtt{com}_{ser_1}$ and $\mathtt{com}_{ser}$ and to create and send $(v_1, \mathtt{com}_{v_1}, \mathtt{unv}_{v_1}, \mathtt{com}_{UH}, \mathtt{unv}_{UH}, \sigma_{UH})$ to U.

Figure 8.15.: The fifth part of the simulator with an honest operator and corrupt user: Defines behavior for user registration.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\texttt{Compute}$ task for a function f against a corrupted user.

On input $(\texttt{Compute}, \mathsf{f}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, after having received input $(\texttt{Process\_Onion}, \mathsf{O}, (\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, ser^*, \mathsf{com}_{ser}, \pi), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, and after having received the leak (ℓ) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the protocol of the operator: it aborts if ser^* has been used before and otherwise stores ser^*, sets $stmt := (\mathsf{com}_{UH}, \mathsf{com}_{lin}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, ser^*, \mathsf{vk}_\mathsf{O})$ and aborts if $\mathrm{Z\scriptstyle K}.\mathrm{Verify}(\pi, stmt, \Lambda_{\mathrm{Compute}})$ fails. Then it aborts if there is no (unique) tuple $(\mathsf{f}, \ell, \mathsf{com}_{fp}, \sigma_{fp})$ for the given (f, ℓ) and otherwise simulates sending $(\mathsf{com}_{fp}, \sigma_{fp})$ through $P^{\leftarrow}$ to $\mathcal{F}_{\mathsf{ORR}}$.

On input $(\texttt{Compute}, \mathsf{f}, (UH, \mathsf{com}_{fp}, \mathsf{unv}_{UH}, x_\mathsf{U}))$ from U to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{BKA}$ aborts if $\mathrm{C\scriptstyle OM}.\mathrm{Unv}(\mathsf{com}_{UH}, \mathsf{unv}_{UH}, UH)$ or $\mathrm{C\scriptstyle OM}.\mathrm{Unv}(\mathsf{com}_{fp}, \mathsf{unv}_{fp}, fp)$ fail due to the simulation of $\mathcal{F}_{\mathsf{MPC}}$.

Then, $\mathcal{S}_{BKA}$ extracts $\mathsf{pk}_\mathsf{U}(= g_1^{\mathsf{sk}_\mathsf{U}})$ from the witness of π using $\mathrm{Z\scriptstyle K}.\mathrm{ExtractWit}(td_{ext}, \pi, stmt, \Lambda_{\mathrm{Compute}})$ and aborts if there is no stored entry $(\mathsf{pk}_\mathsf{U}, pid_{\mathsf{U}^*})$. Otherwise $\mathcal{S}_{BKA}$ sends input $(\texttt{Compute}, \mathsf{f}, x_\mathsf{U})$ to $\mathcal{F}_{\mathsf{BKA}}$ in the name of the extracted U^* who belongs to pid_{U^*}.

On output $(\vec{\alpha}, \vec{s}, \vec{a}, y_\mathsf{U})$ from $\mathcal{F}_{\mathsf{BKA}}$ to U, $\mathcal{S}_{BKA}$ sets $(\mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}) \leftarrow \mathrm{C\scriptstyle OM}.\mathrm{Com}(\vec{a})$ and reports output $(\vec{\alpha}, \vec{s}, \vec{a}, \mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}, y_\mathsf{U})$ from $\mathcal{F}_{\mathsf{MPC}}$ to U.

On input $(\texttt{Process\_Onion}, \mathsf{O}, (\mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \pi^*), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ checks if $\vec{\alpha} \neq \perp$ or $\vec{s} \neq \perp$ and in that case sets $stmt^* := (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and aborts if $\mathrm{Z\scriptstyle K}.\mathrm{Verify}(\pi^*, stmt^*, \Lambda_{\mathrm{Transfer}}) \neq 1$.

Regardless of $\vec{\alpha}$ and $\vec{s}$, $\mathcal{S}_{BKA}$ follows the protocol of the operator to construct the new logbook and final message and simulates sending $(ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \sigma_{UH^{\mathrm{new}}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$.

Figure 8.16.: The sixth part of the simulator with an honest operator and corrupt user: Defines behavior for computation of a function f.

Proof. The setups $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{SMT}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{ORR}})$ are considered to be *hybrid functionalities*, which by UC-conventions can be simulated by the simulator doing what the functionality dictates without any other party noticing. □

GAME$_3(\kappa)$: The simulator now maintains the list of serial numbers in which all the serials that were opened by a user are stored. That is, whenever a user successfully proves that the used logbook λ is *fresh* and hasn't been used before by sending a serial number ser together with a proof π during any of the tasks for BK, OA or Upd, $\mathcal{S}_3$ verifies the proof and aborts, if either the proof fails to verify or if the serial was

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\texttt{Outsource}$ task for a function f where everyone is honest.

On input $(\texttt{Outsource}, \mathsf{f}, \mathsf{User})$, $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{H}})$, and $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ sets $lin := \bot$, $rs_{UH}^{(\mathsf{H})} := \bot$, $rs_{UH}^{(\mathsf{O})} := \bot$, $rs_{x_{\mathsf{U}}}^{(\mathsf{H})} := \bot$, $rs_{x_{\mathsf{U}}}^{(\mathsf{O})} := \bot$, $rs_{otp}^{(\mathsf{H})} := \bot$, and $rs_{otp}^{(\mathsf{O})} := \bot$, and stores $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}\}$ and $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}\}$.

Figure 8.17.: The seventh part of the simulator with an honest operator, user, and helper: Defines behavior for outsourcing a function f.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\texttt{Outsource}$ task for a function f against a corrupted user and honest operator and helper.

On input $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{O}})$ and $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$, after having received input $(\texttt{Process\_Onion}, \mathsf{O}, (rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}, ser^*, \mathtt{com}_{ser}, \mathtt{com}_{ser_0}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \pi), P^{\rightarrow}, P^{\leftarrow})$ and $(\texttt{Process\_Onion}, \mathsf{H}, (rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}), P^{\rightarrow *}, P^{\leftarrow *})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ follows the protocol of the operator by aborting if π fails to verify or if ser is a duplicate. Then $\mathcal{S}_{BKA}$ proceeds to reconstruct UH, x_{U} and otp using the robust shares and the $\texttt{Combine}$ procedure from Fig. 7.2.

Now $\mathcal{S}_{BKA}$ draws a random $lin \overset{\$}{\leftarrow} \mathbb{Z}_o$ and stores $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}\}$ and $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}\}$.

Then $\mathcal{S}_{BKA}$ extracts pk_{U} from π using $\mathrm{Z\kappa.ExtractWit}(td_{ext}, \pi, stmt, \Lambda_{\mathrm{Outsource}})$ and aborts if there is no stored entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}^*})$.

Otherwise, $\mathcal{S}_{BKA}$ loads the ssid $ssid$, stores $(ssid, lin)$ and calls $\mathcal{F}_{\mathsf{BKA}}$ with input $(\texttt{Outsource}, \mathsf{f}, x_{\mathsf{U}})$ in the name of U^* that corresponds to pid_{U^*} whom the extracted pk_{U} belongs to.

On output (Ok) from $\mathcal{F}_{\mathsf{BKA}}$ to U^*, $\mathcal{S}_{BKA}$ follows the protocol of the operator to construct the new logbook and final message and simulates sending $(ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH^{\mathrm{new}}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$ and (lin) through $P^{\leftarrow *}$ in $\mathcal{F}_{\mathsf{ORR}}$.

Figure 8.18.: The eighth part of the simulator with an honest operator and helper and a corrupted user: Defines behavior for outsourcing a function f.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `Outsource` task for a function f against a corrupted helper and honest user and operator.

On input $(\texttt{Outsource}, \mathsf{f}, \mathsf{User})$ and $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ creates robust shares $(rs_{UH}^{(\mathsf{H})}, rs_{UH}^{(\mathsf{O})})$, $(rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})})$ and $(rs_{otp}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{O})})$ by following $\texttt{Share}(\vec{0})$ from Fig. 7.1 for zero-vectors of appropriate size. Then $\mathcal{S}_{BKA}$ simulates $\texttt{Send}(\mathsf{H}, (rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}))$ from Fig. 7.4 and remembers ϑ and $\texttt{Send}(\mathsf{O}, (r))$ for a random vector r of sufficient size.

Now $\mathcal{S}_{BKA}$ inserts $(\texttt{Outsource}, \mathsf{f})$ in the name of H into $\mathcal{F}_{\mathsf{BKA}}$.

On output $(\texttt{Ok})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ follows the honest protocol of the operator: It draws a share $lin^{(\mathsf{O})} \overset{\$}{\leftarrow} \mathbb{Z}_o$ for the new linking number and computes $(\texttt{com}_{lin^{(\mathsf{O})}}, \texttt{unv}_{lin^{(\mathsf{O})}}) \leftarrow \textsc{Com}.\mathsf{Com}(lin^{(\mathsf{O})})$ and reports a message $(\texttt{com}_{lin^{(\mathsf{O})}}, \texttt{com}_{UH}^{(\mathsf{H})}, \texttt{com}_{x_{\mathsf{U}}}^{(\mathsf{H})}, \texttt{com}_{otp}^{(\mathsf{H})})$ to H, where the commitments (except for lin) were taken from the respective robust shares.

When receiving $(lin^{(\mathsf{H})}, \texttt{com}_{UH^{(\mathsf{O})}}, \texttt{com}_{x_{\mathsf{U}}^{(\mathsf{O})}}, \texttt{com}_{otp^{(\mathsf{O})}})$ from H to O, $\mathcal{S}_{BKA}$ follows the protocol of the operator any aborts if $\texttt{Verify}(\ldots)$ from Fig. 7.3 fails.

Then $\mathcal{S}_{BKA}$ reports a message $(lin^{(\mathsf{O})}, \texttt{unv}_{lin^{(\mathsf{O})}})$ from O to H.

On input $(\texttt{Process\_Back\_Onion}, (lin), \vartheta)$ from H to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ aborts if $lin \neq lin^{(\mathsf{O})} + lin^{(\mathsf{H})}$. Otherwise, $\mathcal{S}_{BKA}$ loads the current ssid $ssid$ and stores an entry $(ssid, lin)$ and an entry $\{pid_{\mathsf{H}}, \mathsf{f}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}\}$.

Figure 8.19.: The ninth part of the simulator with an honest operator and user and a corrupted helper: Defines behavior for outsourcing a function f.

already contained in the list. If no abort happened, $\mathcal{S}_3$ adds ser to the list. The code of the operator is changed in such a way, that the checks that the simulator does now are not repeated by the operator.

Lemma 8.2.2. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing* $\textsc{Game}_2(\kappa)$ *from* $\textsc{Game}_3(\kappa)$ *fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_2(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_3(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The simulator mimics the behavior of an honest operator. Since none of the code regarding the list of serial numbers is in any way dependent on the secret input $\mathcal{Z}$ gives to the user or the operator, this can be done without any loss of generality. In particular, assume that any party (that is, either the operator or $\mathcal{S}_3$) aborts due to a duplicate serial number in any game. In either case, this would mean that there was a previous interaction of either the task for BK, OS, or Upd, where the user opened a

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `Outsource` task for a function f against an honest operator and corrupted user and helper.

On input $(\texttt{Outsource}, \mathsf{f}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$ after having received input $(\texttt{Process\_Onion}, \mathsf{O}, (rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{otp}, ser^*, \mathsf{com}_{ser}, \mathsf{com}_{ser_0}, \mathsf{com}_{\mathsf{sk_U}}, \pi), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ reconstructs UH, x_U and otp using the robust shares and the `Combine` procedure from Fig. 7.2.

Now $\mathcal{S}_{BKA}$ draws a random share $lin^{(\mathsf{O})} \overset{\$}{\leftarrow} \mathbb{Z}_o$, computes a commitment $(\mathsf{com}_{lin^{(\mathsf{O})}}, \mathsf{unv}_{lin^{(\mathsf{O})}}) \leftarrow \textsc{Com}.\mathrm{Com}(lin^{(\mathsf{O})})$ and reports a message $(\mathsf{com}_{lin^{(\mathsf{O})}}, \mathsf{com}^{(\mathsf{H})}_{UH}, \mathsf{com}^{(\mathsf{H})}_{x_\mathsf{U}}, \mathsf{com}^{(\mathsf{H})}_{otp})$ from O to H, where the commitments (except for lin) were taken from the respective robust shares.

When receiving message $(lin^{(\mathsf{H})}, \mathsf{com}_{UH^{(\mathsf{O})}}, \mathsf{com}_{x_\mathsf{U}^{(\mathsf{O})}}, \mathsf{com}_{otp^{(\mathsf{O})}})$ from H to O, O sets $stmt := (rs^{(\mathsf{O})}_{UH}, \mathsf{com}_{UH^{(\mathsf{O})}}, ser^*, \mathsf{com}_{\mathsf{sk_U}}, \mathsf{vk_O})$ and aborts if $\textsc{Zk}.\mathrm{Vfy}(\pi, stmt, \Lambda_{\mathrm{Outsource}}) \neq 1$ or if any of the commitments on $UH^{(\mathsf{O})}$, $x^{(\mathsf{O})}_\mathsf{U}$, $otp^{(\mathsf{O})}$ fail to verify.

Now $\mathcal{S}_{BKA}$ extracts $\mathsf{pk_U}$ from π using $\textsc{Zk}.\mathrm{ExtractWit}(td_{ext}, \pi, stmt, \Lambda_{\mathrm{Outsource}})$ and aborts if there is no stored entry $(\mathsf{pk_U}, pid_{\mathsf{U}^*})$.

Otherwise, $\mathcal{S}_{BKA}$ calls $\mathcal{F}_{\mathsf{BKA}}$ with input $(\texttt{Outsource}, \mathsf{f}, x_\mathsf{U})$ in the name of U^* that corresponds to pid_{U^*} whom the extracted $\mathsf{pk_U}$ belongs to and with input $(\texttt{Outsource}, \mathsf{f})$ in the name of O.

On output (Ok) from $\mathcal{F}_{\mathsf{BKA}}$ to U^* and O, $\mathcal{S}_{BKA}$ sets $lin := lin^{(\mathsf{O})} + lin^{(\mathsf{H})}$ and stores $\{pid_\mathsf{H}, \mathsf{f}, lin, rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_\mathsf{U}}, rs^{(\mathsf{O})}_{otp}\}$ for OA.

Then $\mathcal{S}_{BKA}$ follows the protocol of the operator to construct the new logbook and final message and simulates sending $(ser_1, \mathsf{com}_{ser_1}, \mathsf{unv}_{ser_1}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \sigma_{UH^{\mathrm{new}}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$ and reports a message $(lin^{(\mathsf{O})}, \mathsf{unv}_{lin^{(\mathsf{O})}})$ from O to H.

Figure 8.20.: The tenth part of the simulator with an honest operator and corrupted user and helper: Defines behavior for outsourcing a function f.

value of com_{ser^*} to the same ser that is now seen by the respective party. Since the different parties execute the same code, their abort-criteria is equivalent. The same is true for the verification of the proof π. □

Game$_4(\kappa)$: This game lets the operator leak all the linking numbers $\{lin_\zeta\}$ to the simulator during the OA task right after loading them and before providing input for $\mathcal{F}_{\mathsf{MPC}}$. It furthermore extends the state of the simulator by two lists of tuples. The first list is based on what an honest operator would store for the updates after the OA-task, where the simulator stores $\{lin, \vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{\alpha}}, ct_{y_\mathsf{U}}\}$, the second list is based on the data

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\texttt{Analytics}$ task for a function f against an honest operator and helper.

On input $(\texttt{Analytics}, \mathsf{f}, pid_{\mathsf{H}})$ and $(\texttt{Analytics}, \mathsf{f}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ loads the first Z_{f} entries of $\{(\mathsf{H}, lin, rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})})\}$ and $\{(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z_{\mathsf{f}}}$ and removes them from the database.

Then, for each entry where $lin \neq \bot$, $\mathcal{S}_{BKA}$ proceeds to reconstruct UH, x_{U} and otp using the robust shares and the $\texttt{Combine}$ procedure from Fig. 7.2 and aborts if reconstruction fails.

When receiving leak $\{(\zeta), \vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta\}$ for ζ that belong to corrupted users from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ sets for each ζ from that leak: $(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{a})$ extracts $(otp_{\vec{\alpha}}, otp_{\vec{s}}, otp_{\vec{a}}, otp_{\mathtt{unv}}, otp_y)$ from otp, sets $ct_{\vec{\alpha}} := \vec{\alpha} + otp_{\vec{\alpha}}$, $ct_{\vec{s}} := \vec{s} + otp_{\vec{s}}$, $ct_{\vec{a}} := \vec{a} + otp_{\vec{a}}$, $ct_{\mathtt{unv}_{\vec{a}}} := \mathtt{unv}_{\vec{a}} + otp_{\mathtt{unv}}$, and $ct_{y_{\mathsf{U}}} := y_{\mathsf{U}} + otp_y$, and stores $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}\}$ and $\{lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_{\mathsf{U}}}\}$.

Figure 8.21.: The eleventh part of the simulator with an honest operator and helper: Defines behavior for computing an outsourced a function f.

stored by each helper, for which the simulator stores $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}\}$. During simulation of $\mathcal{F}_{\mathsf{MPC}}$ for OA-tasks, *i.e.* after the operator inserted the message $(\texttt{Compute}, \mathsf{f}, (fp, \mathtt{unv}_{fp}, \{rs_{UH}^{(\mathsf{O})}, rs_{x_{\mathsf{U}}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}\}_{\zeta=1}^{Z_{\mathsf{f}}}))$ into $\mathcal{F}_{\mathsf{MPC}}$ and the helper has input $(\texttt{Compute}, \mathsf{f}, (\mathtt{com}_{fp}, \{(rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z}))$ to $\mathcal{F}_{\mathsf{MPC}}$, the simulator $\mathcal{S}_4$ computes f honestly (with fresh coins, if necessary), based on the two inputs. $\mathcal{S}_4$ uses the protocol $\texttt{Combine}$ on both received shares to reconstruct $(UH, x_{\mathsf{U}}, otp)_\zeta$ for each user $\zeta \in [Z]$. If reconstruction on any of the shares fails, $\mathcal{S}_4$ aborts. With this and the extracted individual OTPs $(otp_{\vec{\alpha}}, otp_{\vec{s}}, otp_{\vec{a}}, otp_{\mathtt{unv}}, otp_y)$ from otp, $\mathcal{S}_4$ computes $ct_{\vec{\alpha}_\zeta} := \vec{\alpha}_\zeta + otp_{\vec{\alpha}\zeta}$, $ct_{\vec{s}_\zeta} := \vec{s}_\zeta + otp_{\vec{s}\zeta}$, $ct_{\vec{a}_\zeta} := \vec{a}_\zeta + otp_{\vec{a}\zeta}$, $(\mathtt{com}_{\vec{a}_\zeta}, \mathtt{unv}_{\vec{a}_\zeta}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{a}_\zeta)$, $ct_{\mathtt{unv}_{\vec{a}_\zeta}} := \mathtt{unv}_{\vec{a}_\zeta} + otp_{\mathtt{unv}}$ and $ct_{y_\zeta} := y_\zeta + otp_{y_\zeta}$. $\mathcal{S}_4$ then adds a new entry $\{lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_{\mathsf{U}}}\}$ for the operators view and $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}\}$ for the helper, where the linking number lin is taken from the leak sent by the operator.

During the Upd task, $\mathcal{S}_4$ uses the data stored for the helper in order to verify that a corrupted helper sent the correct values back to an honest user. This replaces the check the user performs with the OTPs.

Lemma 8.2.3. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_3(\kappa)$ from $\textsc{Game}_4(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_3(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_4(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the $\texttt{Analytics}$ task for a function f against an honest operator and corrupted helper.

On input $(\texttt{Analytics}, pid_{\mathsf{O}})$ and after receiving the leak (ℓ) from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ aborts if there is no tuple $\{\mathsf{f}, \ell, \mathsf{com}_{fp}, \sigma_{fp}\}$ stored for the given (f, ℓ) and otherwise loads the remaining values and reports a message $(\mathsf{com}_{fp}, \sigma_{fp})$ from O to H.

On input $(\texttt{Compute}, \mathsf{f}, (\mathsf{com}_{fp}, \{(rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z}))$ from H to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{BKA}$ aborts if (com_{fp}) entered here differs from the one sent to H earlier. Otherwise, $\mathcal{S}_{BKA}$ loads the first Z_{f} entries of $\{(lin, rs_{UH}^{(\mathsf{H})}, rs_{x_{\mathsf{U}}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z_{\mathsf{f}}}$ and removes them from the database. Then $\mathcal{S}_{BKA}$ proceeds to reconstruct UH, x_{U} and otp using the robust shares and the $\texttt{Combine}$ procedure from Fig. 7.2 and inserts $(\texttt{Analytics}, \mathsf{f})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of H.

When receiving leak $\{(\zeta)\}$ as indices for all inputs that belong to a corrupted user, $\mathcal{S}_{BKA}$ sends the previously reconstructed (x_{U}) for all indices that were contained in the leak to $\mathcal{F}_{\mathsf{BKA}}$.

When receiving leak $\{(\zeta), \vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta\}$ for ζ that belong to corrupted users from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ sets for each ζ from that leak: $(\mathsf{com}_{\vec{a}}, \mathsf{unv}_{\vec{a}}) \leftarrow \mathrm{COM.Com}(\vec{a})$ extracts $(otp_{\vec{\alpha}}, otp_{\vec{s}}, otp_{\vec{a}}, otp_{\mathrm{unv}}, otp_{y})$ from otp, sets $ct_{\vec{\alpha}} := \vec{\alpha} + otp_{\vec{\alpha}}$, $ct_{\vec{s}} := \vec{s} + otp_{\vec{s}}$, $ct_{\vec{a}} := \vec{a} + otp_{\vec{a}}$, $ct_{\mathrm{unv}_{\vec{a}}} := \mathsf{unv}_{\vec{a}} + otp_{\mathrm{unv}}$, and $ct_{y_{\mathsf{U}}} := y_{\mathsf{U}} + otp_{y}$, and stores $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathrm{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}\}$ and $\{lin, \vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{a}}, ct_{y_{\mathsf{U}}}\}$. For all the other ζ (*i.e.* the ones that are not contained in the leak and hence correspond to the data from honest users) $\mathcal{S}_{BKA}$ draws random ciphers $ct_{\vec{\alpha}}$, $ct_{\vec{s}}$, $ct_{\vec{a}}$, $ct_{\mathrm{unv}_{\vec{a}}}$, and $ct_{y_{\mathsf{U}}}$ and stores $\{lin, \vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{a}}, ct_{y_{\mathsf{U}}}\}$.

Finally, $\mathcal{S}_{BKA}$ reports $(\{(ct_{\vec{\alpha}_\zeta}, ct_{\vec{s}_\zeta}, ct_{\vec{a}_\zeta}, ct_{\mathrm{unv}_{\vec{a}_\zeta}}, ct_{y_{\mathsf{U}}})_\zeta\}_{\zeta=1}^{Z_{\mathsf{f}}})$ as output from $\mathcal{F}_{\mathsf{MPC}}$ to H.

Figure 8.22.: The twelfth part of the simulator with an honest operator and corrupted helper: Defines behavior for computing an outsourced a function f.

Proof. Indistinguishability here holds for the same reason that it held in Theorem 8.2.2. The leaked linking number by the operator is undetectable for any environment yet contains the correct linking number, and the contents of both lists only depend on *messages* which $\mathcal{S}_4$ can access via simulation of $\mathcal{F}_{\mathsf{MPC}}$, and not (directly) on *secret input*. Hence, we only have a new encapsulation, where (PPT-)code that depends only on previous messages was executed by the operator or helper in $\mathrm{GAME}_3(\kappa)$ is now executed by $\mathcal{S}_4$ in $\mathrm{GAME}_4(\kappa)$.

The equivalence of their contents easily follows from the same argument. The contents themselves depend on the message that the user sent to the resp. parties,

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `Update` task against a corrupted user and honest operator and helper.

On input $(\texttt{Update}, pid_{\mathsf{H}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after having received input $(\texttt{Process\_Onion}, \mathsf{H}, (lin), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ checks if there is a stored entry $\{\mathsf{H}, lin, ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}\}$ for the given (H, lin). If such an entry exists, $\mathcal{S}_{BKA}$ removes them and simulates sending simulates sending $(ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathtt{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}})$ through $P^{\leftarrow}$ to $\mathcal{F}_{\mathsf{ORR}}$. Otherwise, $\mathcal{S}_{BKA}$ continues without sending a message.

On input $(\texttt{Update}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$, and after having received input $(\texttt{Process\_Onion}, \mathsf{O}, (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, ser, \mathtt{com}_{ser_0}, lin, \pi, \pi^*), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ looks up $\{lin, \vec{\alpha}, \vec{s}, \mathtt{com}_{\vec{a}}, ct_{y_{\mathsf{U}}}\}$ for the given lin, aborts if no such entry exists or if ser was used before and otherwise removes the entries from the database and remembers ser.

Then $\mathcal{S}_{BKA}$ sets $stmt := (\mathtt{com}_{UH}, ser, lin, \mathtt{com}_{\mathsf{sk}_{\mathsf{U}}}, \mathsf{vk}_{\mathsf{O}})$ and aborts if $\textsc{Zk}.\mathsf{Verify}(\pi, stmt, \Lambda_{\mathrm{Update}}) = 0$. Otherwise, $\mathcal{S}_{BKA}$ follows the protocol of the operator: If $\vec{\alpha} \neq \bot$ or $\vec{s} \neq \bot$, $\mathcal{S}_{BKA}$ sets $stmt^* := (\mathtt{com}_{UH}, \mathtt{com}_{UH^{(1)}}, \mathtt{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and aborts if $\textsc{Zk}.\mathsf{Verify}(\pi^*, stmt^*, \Lambda_{\mathrm{Transfer}}) \neq 1$.

Regardless of $\vec{\alpha}$ and $\vec{s}$, $\mathcal{S}_{BKA}$ extracts pk_{U} from π using $\textsc{Zk}.\mathsf{ExtractWit}(td_{ext}, \pi, stmt, \Lambda_{\mathrm{Update}})$ and aborts if there is no stored entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}^*})$. Otherwise, $\mathcal{S}_{BKA}$ inserts $(\texttt{Update})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of the extracted U^* who belongs to pid_{U^*} and follows the protocol of the operator to construct the new logbook and final message and simulates sending $(ct_{y_{\mathsf{U}}}, \mathtt{com}_{\vec{a}}, ser_1, \mathtt{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH^{\mathrm{new}}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$.

Figure 8.23.: The thirteenth part of the simulator with an honest operator and helper and a corrupted user: Defines behavior for updating the user data.

the order depends on the (adversarially chosen) scheduling mechanism, which, given any environment $\mathcal{Z}$ that tries to distinguish the two games, is equivalent for both games.

The final change contains the check in the corruption scenario where the helper is corrupted, but the user and operator are honest. Here, in $\textsc{Game}_3(\kappa)$, U aborts if either $\mathsf{Unv}(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, \vec{a}) \neq 1$, or if $ct_{y_{\mathsf{U}}}$ differs from the value the honest operator sent.

Equivalence for the latter is straightforward. If a user aborts due to the former condition ($\mathsf{Unv}(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, \vec{a}) \neq 1$), then this means that some value was tampered with. $\mathtt{com}_{\vec{a}}$ was received by the operator, who, in this scenario, is honest, hence

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `Update` task against a corrupted helper and honest user and operator.

On input $(\texttt{Update}, pid_{\mathsf{O}})$ and $(\texttt{Update}, \mathsf{User})$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ inserts $(\texttt{Update})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of H.

When receiving leak $(ssid)$ from $\mathcal{F}_{\mathsf{BKA}}$, $\mathcal{S}_{BKA}$ aborts if no entry $(ssid, lin)$ is stored and otherwise simulates $\texttt{Send}(\mathsf{H}, (lin))$ from Fig. 7.4 and remembers ϑ.

On input $(\texttt{Process\_Back\_Onion}, (ct_{\vec{\alpha}}, ct_{\vec{s}}, ct_{\vec{a}}, ct_{\mathrm{unv}_{\vec{a}}}, ct_{y_{\mathsf{U}}}), \vartheta)$ from H to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ loads and removes $\{\mathsf{H}, lin, ct^*_{\vec{\alpha}}, ct^*_{\vec{s}}, ct^*_{\vec{a}}, ct^*_{\mathrm{unv}_{\vec{a}}}, ct^*_{y_{\mathsf{U}}}\}$ for the given (H, lin) and aborts if they do not exist or differ from the values sent by H.

Figure 8.24.: The fourteenth part of the simulator with an honest operator and user and a corrupted helper: Defines behavior for outsourcing a function f.

the correct value will be sent to the user. The OTPs $otp_{\vec{a}}, otp_{\mathrm{unv}}$ used to mask $\vec{a}$ and $\mathrm{unv}_{\vec{a}}$ were created by the user during the OS task and hence are also correct. Thus, the only values that *could* be tampered with are $ct_{\vec{a}}$ and $ct_{\mathrm{unv}_{\vec{a}}}$, which $\mathcal{S}_4$ has seen during the OS task and hence, for which a consistency check suffices. $\square$

GAME$_5(\kappa)$: During setup, the Common Reference String crs is now created by $(crs, td_{ext}) \leftarrow$ SetupExt instead of following the honest Setup-protocol. The simulator stores the extraction trapdoor td_{ext}. Also, the operator now leaks the signature key pair $(\mathsf{vk}_{\mathsf{O}}, \mathsf{k}_{\mathsf{O}})$ to $\mathcal{S}_5$ during the Init-task, which is also stored by the simulator.

Lemma 8.2.4. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_4(\kappa)$ from GAME$_5(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_4(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_5(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability here follows from the CRS indistinguishability (*cf.* Section 4.4.3) of the zero-knowledge scheme Zk. If any environment $\mathcal{Z}$ could distinguish the execution of the protocol when using crs created by $crs \leftarrow$ Setup from crs created by $(crs, td_{ext}) \leftarrow$ SetupExt with probability $\frac{1}{2} + \alpha$, we can build a PPT-environment $\mathcal{Z}'$ that breaks the indistinguishability of the dual-mode property of Zk, by having $\mathcal{Z}'$ execute the code of all parties in its head. This leads to the same success probability of $\frac{1}{2} + \alpha$, thus causing $\alpha \in \mathsf{negl}(\kappa)$ by requirement of the chosen Zk-scheme. $\square$

GAME$_6(\kappa)$: Introduces a new list for the simulator of tuples $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$. Each entry means that the user's public keys pk_{U} belongs to the user with pid pid_{U}.

During simulation of $\mathcal{F}_{\mathsf{Reg}}$, after $\mathcal{Z}$ gave instructions to send a message $(\texttt{Register}, \mathsf{pk}_{\mathsf{U}})$ from the user with pid pid_{U} for a corrupted user U to $\mathcal{F}_{\mathsf{Reg}}$ and simulation succeeded (*i.e.* did not abort), $\mathcal{S}_6$ adds a new entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$.

Simulator $\mathcal{S}_{BKA}$

Simulator $\mathcal{S}_{BKA}$ for the `Update` task against an honest operator and corrupted user and helper.

On input $(\texttt{Update}, pid_{\mathsf{O}})$ from $\mathcal{F}_{\mathsf{BKA}}$ and after having received input $(\texttt{Process\_Onion}, \mathsf{O}, (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, ser, \mathsf{com}_{ser_0}, lin, \pi, \pi^*), P^{\rightarrow}, P^{\leftarrow})$ from U to $\mathcal{F}_{\mathsf{ORR}}$, $\mathcal{S}_{BKA}$ looks up $\{lin, \vec{\alpha}, \vec{s}, \mathsf{com}_{\vec{a}}, ct_{y_\mathsf{U}}\}$ for the given lin, aborts if no such entry exists or if ser was used before and otherwise removes the entries from the database and remembers ser.

Then $\mathcal{S}_{BKA}$ sets $stmt := (\mathsf{com}_{UH}, ser, lin, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{vk}_\mathsf{O})$ and aborts if $\text{Z}\textsc{k}.\text{Verify}(\pi, stmt, \Lambda_{\text{Update}}) = 0$.

Otherwise, $\mathcal{S}_{BKA}$ follows the protocol of the operator: If $\vec{\alpha} \neq \bot$ or $\vec{s} \neq \bot$, $\mathcal{S}_{BKA}$ sets $stmt^* := (\mathsf{com}_{UH}, \mathsf{com}_{UH^{(1)}}, \mathsf{com}_{UH^{(2)}}, \vec{\alpha}, \vec{s})$ and aborts if $\text{Z}\textsc{k}.\text{Verify}(\pi^*, stmt^*, \Lambda_{\text{Transfer}}) \neq 1$.

Regardless of $\vec{\alpha}$ and $\vec{s}$, $\mathcal{S}_{BKA}$ extracts pk_U from π using $\text{Z}\textsc{k}.\text{ExtractWit}(td_{ext}, \pi, stmt, \Lambda_{\text{Update}})$ and aborts if there is no stored entry $(\mathsf{pk}_\mathsf{U}, pid_{\mathsf{U}^*})$. Otherwise, $\mathcal{S}_{BKA}$ inserts $(\texttt{Update})$ into $\mathcal{F}_{\mathsf{BKA}}$ in the name of the extracted U^* who belongs to pid_{U^*} and H.

On output $(\texttt{Ok})$ from $\mathcal{F}_{\mathsf{BKA}}$ to H and $(\vec{\alpha}, \vec{s}, \vec{a}, y_\mathsf{U})$ from $\mathcal{F}_{\mathsf{BKA}}$ to U^*, $\mathcal{S}_{BKA}$ follows the protocol of the operator to construct the new logbook and final message and simulates sending $(ct_{y_\mathsf{U}}, \mathsf{com}_{\vec{a}}, ser_1, \mathsf{com}_{ser_1}, \mathtt{unv}_{ser_1}, \mathsf{com}_{lin}, \mathtt{unv}_{lin}, \sigma_{UH^{\text{new}}})$ through $P^{\leftarrow}$ in $\mathcal{F}_{\mathsf{ORR}}$.

Figure 8.25.: The fiveteenth part of the simulator with an honest operator and corrupted user and helper: Defines behavior for outsourcing a function f.

During simulation of the task for User Registration, after $\mathcal{Z}$ gave instructions to send a message $(\pi, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \mathsf{com}_{ser_0})$ from a corrupted user to the operator, $\mathcal{S}_6$ takes pk'_U from π (since it is contained in the statement of Λ_{UserReg}) and aborts if either no entry $(\mathsf{pk}'_\mathsf{U}, pid_\mathsf{U})$ is stored or if a user with public key pk'_U is already registered.

Lemma 8.2.5. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_5(\kappa)$ from $\textsc{Game}_6(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_5(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_6(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The only difference between $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_6(\kappa)$ is that $\mathcal{S}_6$ stores additional information in $\textsc{Game}_6(\kappa)$ that was accessible even in $\textsc{Game}_5(\kappa)$. Additionally, this game introduces a new abort-criteria.

To show indistinguishability of $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_6(\kappa)$ we thus have to show that those two criteria are equivalent. In $\textsc{Game}_5(\kappa)$ the operator fetches the key pk_U

belonging to pid_{U} from $\mathcal{F}_{\mathsf{Reg}}$, thus effectively asking (since $\textsc{Game}_2(\kappa)$) $\mathcal{S}_6$ for the key that the user with pid pid_{U} registered there. Thus, instead of $\mathcal{S}_6$ simulating $\mathcal{F}_{\mathsf{Reg}}$ and giving the operator the key pk_{U} so O can verify that pk_{U} is the one that was used in π, $\mathcal{S}_6$ now follows these steps. Only that due to the simulation of $\mathcal{F}_{\mathsf{Reg}}$, no further interaction is required to obtain pk_{U}. Hence, aborts in $\textsc{Game}_5(\kappa)$ due to a duplicate or mismatching pk_{U} occur if and only if aborts in $\textsc{Game}_6(\kappa)$ happen due to a duplicate or mismatching pk_{U}. Thus, both distributions from $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_6(\kappa)$ are equivalent. □

$\textsc{Game}_7(\kappa)$: Whenever a corrupted user enters a message containing a ZK proof π into $\mathcal{F}_{\mathsf{ORR}}$ during any of the tasks for BK, OS or Upd, $\mathcal{S}_7$ uses the trapdoor td_{ext} to extract the witness wit from π. The simulator only uses the extracted public key pk_{U} in order to get the pid pid_{U} by looking for an entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$ in the list. If no such entry exists for the given public key the simulator aborts.

Lemma 8.2.6. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_6(\kappa)$ from $\textsc{Game}_7(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_6(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_7(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Extraction is possible due to the different crs introduced in $\textsc{Game}_5(\kappa)$ and the fact that the interesting part of the witness only has elements from the target group; the ZK scheme provides $F_{pp_{\mathbb{G}}}$-extractability (*cf.* Section 4.4.4) and the witness contains the secret key sk_{U}, so $F_{pp_{\mathbb{G}}}$-extractability suffices to extract $g_1^{\mathsf{sk}_{\mathsf{U}}} = \mathsf{pk}_{\mathsf{U}}$. As the secret key sk_{U} is contained in all three relevant languages as a witness (see Figs. 7.12, 7.17 and 7.22) it can always be extracted. The user found by searching an entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$ is the correct user due to the completeness of the ZK scheme and the setup $\mathcal{F}_{\mathsf{Reg}}$.

Assume for the sake of contradiction that $\mathcal{S}_7$ aborts in $\textsc{Game}_7(\kappa)$ because $pid_{\mathsf{U}} = \perp$, but $\mathcal{S}_6$ would successfully terminate in $\textsc{Game}_6(\kappa)$. The following behavior by the user U could have caused this:

U proved a faulty statement. The correctness property of $\textsc{Zk}$ would cause this to be detected in $\textsc{Game}_6(\kappa)$ by O with overwhelming probability, thus causing an abort.

U changed the commitment $\mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}$ in the statement. Then the proof of equivalence between $\mathsf{com}^*_{\mathsf{sk}_{\mathsf{U}}}$ and the rerandomized $\mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}$ would have to be forged, thus breaking the statement of π. This would also cause an abort in $\textsc{Game}_7(\kappa)$, due to the correctness of $\textsc{Zk}$.

U opened $\mathsf{com}_{\mathsf{sk}_{\mathsf{U}}}$ to to a different $\mathsf{pk}'_{\mathsf{U}}$. Here, the binding-property of the commitment scheme $\textsc{Com}$ would be violated. Since we assumed $\textsc{Com}$ to be unconditionally binding, this cannot occur.

U created its own signature. In case the user U created a signature σ_{UH} on a new logbook λ' containing a public key $\mathsf{pk}'_{\mathsf{U}} \neq \mathsf{pk}_{\mathsf{U}}$ without knowing the operator's secret key k_{O}, U would break the unforgeability of $\textsc{Sig}$. By requirement on $\textsc{Sig}$, this is possible only with probability negligible in κ.

The reverse is also true; if the operator discards the proof π in $\textsc{Game}_6(\kappa)$, then the same happens in $\textsc{Game}_7(\kappa)$, as this part has not been changed. □

$\textsc{Game}_8(\kappa)$: Introduces an incorruptible entity $\mathcal{F}_{\mathsf{BKA}}$ that follows the specification from Figs. 6.1 and 6.2 into the experiment. The entity is only accessible by honest parties and the simulator through subroutine input/output tapes.

Lemma 8.2.7. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_7(\kappa)$ from $\textsc{Game}_8(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_7(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_8(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability of those two games trivially follows; the new machine is only accessible by *honest* parties, who follow the protocol. Their protocol description in $\textsc{Game}_8(\kappa)$ does not include any access to $\mathcal{F}_{\mathsf{BKA}}$. The simulator doesn't access $\mathcal{F}_{\mathsf{BKA}}$ either, so the two distributions of $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$ are statistically close and hence, the best any PPT-environment $\mathcal{Z}$ can do is guessing. □

$\textsc{Game}_9(\kappa)$: Replaces the Trusted Signing Authority T with a dummy party that immediately forwards its input to the ideal functionality $\mathcal{F}_{\mathsf{BKA}}$ and to the simulator $\mathcal{S}_9$. All interactions of T are simulated by $\mathcal{S}_9$ by following the original protocol, who also stores the relevant information for the TSA.

Lemma 8.2.8. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_8(\kappa)$ from $\textsc{Game}_9(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_8(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_9(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability of the two games directly follows from the facts that (1) both tasks for Init and SFP that contain the TSA T are executed with an honest operator O, and (2) the input used by the simulator is the same that would have been used by the TSA. As such, this gamehop is only a cosmetic change where the same code is executed on the same values, only by a different party. This implies indistinguishability based on the fact that in the view of $\mathcal{Z}$ the two games are equivalent. □

$\textsc{Game}_{10}(\kappa)$: This game extends the state of the simulator by an additional list of tuples $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ which contains information on the FPs which are eligible for the computations of functions in BK and OA.

Replaces the operator O with a new operator O' that acts like the original one, but has a few minor changes. After having confirmation that FPs fp can be used for a function f during Sign Function Parameter, O' sends (fp, f) to $\mathcal{S}_{10}$. The simulator then computes $\mathtt{com}_{fp}$ as commitment on the actual FPs and σ_{fp} as corresponding signature using the signing key of the TSA and stores the tuple $(\mathrm{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$. Furthermore, the new operator sends the FPs to $\mathcal{S}_{10}$ during BK and OA tasks and uses the $(\mathtt{com}_{fp}, \sigma_{fp})$ obtained from $\mathcal{S}_{10}$, which the simulator obtains by looking if an entry $(fp, \cdot, \cdot, \cdot)$ has been stored.

Also, during simulation of $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{10}$ uses the stored unveil information to verify the commitment $\mathtt{com}_{fp}$ the user or helper input into $\mathcal{F}_{\mathsf{MPC}}$.

Lemma 8.2.9. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\text{GAME}_9(\kappa)$ from $\text{GAME}_{10}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\text{GAME}_9(\kappa)) = 1] - \Pr[\mathcal{Z}(\text{GAME}_{10}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The rewriting is purely cosmetically, as essentially the same code is executed on different machines. In $\text{GAME}_9(\kappa)$, the commitment $\mathtt{com}_{fp}$ is computed by the operator, the signature σ_{fp} is computed by the TSA, and the resp. values are fetched by the operator prior to a computation with $\mathcal{F}_{\mathsf{MPC}}$. In $\text{GAME}_{10}(\kappa)$, all these steps are done by the simulator. Since the TSA is honest and its key has been created by $\mathcal{S}_{10}$ (since $\text{GAME}_9(\kappa)$) and the relevant information for $\mathtt{com}_{fp}$ and σ_{fp} are leaked by O, the honest code can be executed directly.

Finally, the last change induced is the replaced check with the unveil information. Since this is essentially a rewriting, as the same values are now fetched by the simulator instead of the operator, this change is purely cosmetically and hence undetectable. □

$\textbf{GAME}_{11}(\kappa)$: Replaces the operator O' by a dummy party, which, when receiving input by $\mathcal{Z}$, forwards the input immediately to $\mathcal{F}_{\mathsf{BKA}}$. During the tasks for BK and OA, the new operator O also leaks the input x_{O} and fp to $\mathcal{S}_{11}$. All messages that were sent by the operator in $\text{GAME}_{10}(\kappa)$ are now created by $\mathcal{S}_{11}$ in $\text{GAME}_{11}(\kappa)$ and sent in the name of O.

Lemma 8.2.10. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\text{GAME}_{10}(\kappa)$ from $\text{GAME}_{11}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\text{GAME}_{10}(\kappa)) = 1] - \Pr[\mathcal{Z}(\text{GAME}_{11}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. O is a PPT-machine, which executes code based on the secret input from $\mathcal{Z}$. Knowing this input from the leak, $\mathcal{S}_{11}$ can execute the code of the honest O by following the protocol. This trivially leads to indistinguishability. □

$\textbf{GAME}_{12}(\kappa)$: Instead of storing $(\mathrm{f}, fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ for FPs (*cf.* $\text{GAME}_{10}(\kappa)$) the simulator now stores only $(\mathrm{f}, \ell, \mathtt{com}_{fp}, \sigma_{fp})$, *i.e.* the simulator no longer stores $\mathtt{unv}_{fp}$ and only stores an *identifier* ℓ instead of the actual FPs.

Instead of relying on the leaked FPs for a mapping, $\mathcal{S}_{12}$ uses the leak ℓ obtained from $\mathcal{F}_{\mathsf{BKA}}$ to obtain a consistent $(\mathtt{com}_{fp}, \sigma_{fp})$ tuple. Furthermore, instead of creating these tuples honestly, $\mathcal{S}_{12}$ uses the leaked (f, ℓ) obtained during the Sign Function

Parameter task to sample a new $(\mathsf{com}_{\vec{0}}, \mathsf{unv}_{\vec{0}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{0})$ on the all-zero vector $\vec{0}$ of appropriate size (which is fixed given an instantiation of f) instead of a genuine commitment com_{fp} on fp, and uses the signing key k_T to sign $\mathsf{com}_{\vec{0}}$ and the given function f. The resulting tuple $(\mathsf{com}_{\vec{0}}, \sigma_{fp})$ is then stored as tuple $(\mathsf{f}, \ell, \mathsf{com}_{fp}, \sigma_{fp})$ and used whenever $\mathcal{F}_{\mathsf{BKA}}$ leaks the index ℓ during BK or OA.

Lemma 8.2.11. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{11}(\kappa)$ from $\textsc{Game}_{12}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{11}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{12}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The major change induced in this game hop has the simulator reporting valid zero-commitments instead of valid commitments on the FPs. Let $\mathcal{Z}$ be an environment that distinguishes the two games from $\textsc{Game}_{11}(\kappa)$ and $\textsc{Game}_{12}(\kappa)$. We construct an adversary $\mathcal{A}$ that breaks the hiding property of the commitment scheme $\textsc{Com}$, which we model similar to IND-CPA for encryption schemes. We adapt the LR-view, stating that the challenger C on the hiding game provides us with an oracle that accepts two different inputs, but outputs a valid commitment on a fixed one of them.

The adversary $\mathcal{A}$ can thus create the transcript from $\textsc{Game}_{11}(\kappa)$, but whenever a commitment com_{fp} on the FPs fp is required, $\mathcal{A}$ sends the two messages $(\vec{0}, fp)$ to C and obtains a commitment on one of them.

Note that if the commitment always uses the former entry, $\mathcal{A}$ perfectly simulates $\textsc{Game}_{12}(\kappa)$, and if the commitment always uses the latter entry, $\mathcal{A}$ perfectly simulates $\textsc{Game}_{11}(\kappa)$.

It thus follows that the success probability of $\mathcal{Z}$ in detecting this game hop is limited by the probability of $\mathcal{A}$ to break the hiding game, which is negligible. □

$\textsc{Game}_{13}(\kappa)$: Replaces all honest users U by dummy parties, which, when receiving input by $\mathcal{Z}$, forward the input immediately to $\mathcal{F}_{\mathsf{BKA}}$. During the tasks for BK and OA the new user machines U also leak the input x_U to $\mathcal{S}_{13}$. All messages that were sent by honest user U in $\textsc{Game}_{12}(\kappa)$ are now created by $\mathcal{S}_{13}$ in $\textsc{Game}_{13}(\kappa)$ and simulated via $\mathcal{F}_{\mathsf{ORR}}$ in the name of U.

Note that all leaks by $\mathcal{F}_{\mathsf{BKA}}$ are ignored by $\mathcal{S}_{13}$.

Lemma 8.2.12. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{12}(\kappa)$ from $\textsc{Game}_{13}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{12}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{13}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The case here is similar to that from Lemma 8.2.10. $\mathcal{S}_{13}$ can execute the code of any honest user U since honest user reveal their identity to $\mathcal{S}_{13}$; by leaking the secret input, $\mathcal{S}_{13}$ can follow the protocol of U from $\textsc{Game}_{12}(\kappa)$. Indistinguishability follows. □

GAME$_{14}(\kappa)$: Replaces all helpers H with dummy parties, which, when receiving input by $\mathcal{Z}$, forward the input directly to $\mathcal{F}_{\mathsf{BKA}}$. All messages that were sent by honest helpers H in GAME$_{13}(\kappa)$ are now created by $\mathcal{S}_{14}$ in GAME$_{14}(\kappa)$ and send in the name of H.

Note that (for now) all leaks by $\mathcal{F}_{\mathsf{BKA}}$ are ignored by $\mathcal{S}_{14}$.

Lemma 8.2.13. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{13}(\kappa)$ from GAME$_{14}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textit{GAME}_{13}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textit{GAME}_{14}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. The helper itself has no secrets, so no leaks are required here. Hence, all messages of H depend only on messages it has seen before. Since $\mathcal{S}_{14}$ can see those messages as well and H is a PPT-machine, $\mathcal{S}_{14}$ can execute the code of honest helpers, resulting in an indistinguishable distribution from GAME$_{13}(\kappa)$. □

GAME$_{15}(\kappa)$: $\mathcal{S}_{15}$ now calls the ideal functionality $\mathcal{F}_{\mathsf{BKA}}$ in the name of the corrupted parties with the correct input. This causes $\mathcal{F}_{\mathsf{BKA}}$ to fully perform as defined by its specification, as all inputs are provided. Hence, instead of computing the function f on the inputs in order to simulate $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{15}$ now relies on the leaks provided by $\mathcal{F}_{\mathsf{BKA}}$.

This game also extends the state of the simulator by a list of tuples (*ssid*, *lin*) that groups the ssid of the OS task with the negotiated *linking number lin*.

$\mathcal{S}_{15}$ obtains the input for the corrupted parties as follows:

User Registration, U corrupted, O honest. For UReg there is no secret input, so $\mathcal{S}_{15}$ only calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of the user U with input (`UserRegistration`) after storing the entry $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$.

Bookkeeping, U corrupted, O honest. After $\mathcal{Z}$ gave instructions to send (`Compute`, f, $(UH, \mathtt{com}_{fp}, \mathtt{unv}_{UH}, x_{\mathsf{U}})$) in the name of a user U to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{15}$ uses the extracted pid_{U} (see GAME$_7(\kappa)$) to obtain the user U* who registered for the used public key pk_{U}. After verifying that the commitments on the UH and FPs are valid (see GAME$_{12}(\kappa)$), $\mathcal{S}_{15}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of U* with input (`Compute`, f, x_{U}) where x_{U} has been extracted from the input of the user to $\mathcal{F}_{\mathsf{MPC}}$.

Since $\mathcal{F}_{\mathsf{BKA}}$ now has complete input, $\mathcal{S}_{15}$ obtains leaks. Hence, instead of computing f in his head, $\mathcal{S}_{15}$ uses the output $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}})$ from $\mathcal{F}_{\mathsf{BKA}}$ to U in order generate the output of $\mathcal{F}_{\mathsf{MPC}}$ to U by first computing valid a commitment $(\mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}) \leftarrow \textsc{Com}.\mathsf{Com}(\vec{a})$ and then reporting message $(\vec{\alpha}, \vec{s}, \vec{a}, \mathtt{com}_{\vec{a}}, \mathtt{unv}_{\vec{a}}, y_{\mathsf{U}})$ as output from $\mathcal{F}_{\mathsf{MPC}}$ to the user U.

Outsource, U corrupted, O and H honest. $\mathcal{S}_{15}$ can extract the input of the user by combining the shares that the user sent to the operator and helper, respectively. As those are visible to the simulator due to simulation of $\mathcal{F}_{\mathsf{ORR}}$ $\mathcal{S}_{15}$ can use `Combine` on the shares to reconstruct the actual values completely.

Once $\mathcal{Z}$ has instructed to follow $\texttt{Send}(\mathsf{H}, (rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})}))$ and $\texttt{Send}(\mathsf{O}, (rs_{UH}^{(\mathsf{O})}, rs_{x_\mathsf{U}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})}, ser^*, \mathsf{com}_{ser}, \mathsf{com}_{ser_0}, \mathsf{com}_{\mathsf{sk}_\mathsf{U}}, \pi))$, $\mathcal{S}_{15}$ uses the extracted shares from both the operator O and the helper H to reconstruct $x_\mathsf{U} := \texttt{Combine}(rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{O})})$; if this fails, $\mathcal{S}_{15}$ sets $x_\mathsf{U} := \perp$[2]. $\mathcal{S}_{15}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of $\mathsf{U}_{pid_\mathsf{U}}$, where pid_U corresponds to the user whose identity was extracted from π, using input $(\texttt{Outsource}, \mathsf{f}, x_\mathsf{U})$.

Outsource, U honest, O honest, H corrupted. H is designed to neither learn secrets, nor to have secrets itself. Simulator $\mathcal{S}_{15}$ calls $\mathcal{F}_{\mathsf{BKA}}$ in the name of H with input $(\texttt{Outsource}, \mathsf{f})$. Also, $\mathcal{S}_{15}$ stores a new entry $(ssid, lin)$ after negotiating the linking number.

Outsource, U and H corrupted, O honest. $\mathcal{S}_{15}$ calls $\mathcal{F}_{\mathsf{BKA}}$ for both the helper (who has no secret input whatsoever) and the user U^* (who was identified using pk_U extracted from π since $\textsc{Game}_7(\kappa)$), but using $x_\mathsf{U} := \perp$.

Outsourced Analytics, H corrupted, O honest. Since $\textsc{Game}_{11}(\kappa)$, $\mathcal{S}_{15}$ maintains the list for the outsourced information that is to-be-used for OA that an honest operator would store. As $\mathcal{S}_{15}$ follows the operators code (since $\textsc{Game}_{11}(\kappa)$) it stores all the tuples $(lin^{\text{new}}, rs_{UH}^{(\mathsf{O})}, rs_{x_\mathsf{U}}^{(\mathsf{O})}, rs_{otp}^{(\mathsf{O})})$ during the Outsource task; given the additional inputs from simulation of $\mathcal{F}_{\mathsf{MPC}}$ $\mathcal{S}_{15}$ now has a complete view of the used shares.

After the corrupted helper has inserted $(\texttt{Compute}, \mathsf{f}, (\mathsf{com}_{fp}, \{(rs_{UH}^{(\mathsf{H})}, rs_{x_\mathsf{U}}^{(\mathsf{H})}, rs_{otp}^{(\mathsf{H})})_\zeta\}_{\zeta=1}^{Z}))$ to $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{S}_{15}$ restores each input using the information stored in for the helper during OS and aborts if the reconstruction via $\texttt{Combine}$ fails. When asked by $\mathcal{F}_{\mathsf{BKA}}$ for inputs for an index set $\{Z\} \subset [Z_\mathsf{f}]$ of corrupted users $\{\mathsf{U}_\zeta\}$, $\mathcal{S}_{15}$ enters the inputs x_U from the respective indices.

Reconstruction of the output now only happens for parties ζ in the corrupted party set; information related to honest parties $(ct_{\vec{\alpha}_\zeta}, ct_{\vec{s}_\zeta}, ct_{\vec{a}_\zeta}, ct_{\mathrm{unv}_{\vec{a}_\zeta}}, ct_{y_{\mathsf{U}_\zeta}})$ is drawn at random and put to the list of UI maintained by the simulator for the helper.

Update, any corrupted party. The inputs to the Update-task contain no secrets. $\mathcal{S}_{15}$ can call $\mathcal{F}_{\mathsf{BKA}}$ in the name of any corrupted party with input $(\texttt{Update})$ as soon as the party has shown to be active by sending a message. For corrupted users, $\mathcal{S}_{15}$ awaits the proof π to extract the correct user. For corrupted helpers, $\mathcal{S}_{15}$ awaits their first message. Also, if *only* the helper is corrupted, $\mathcal{S}_{15}$ awaits the leaked ssid $ssid$ from $\mathcal{F}_{\mathsf{BKA}}$ to look up if a tuple $(ssid, lin)$ has been stored. This tuple is then used to construct the first message from the user to the helper.

[2] Note that the simulator does *not* abort here since in a real execution, the helper and operator would not notice that the robust shares are invalid until they are reconstructed in OA and we have to follow that as otherwise this would be a way to distinguish the two games.

Lemma 8.2.14. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{14}(\kappa)$ from $\textsc{Game}_{15}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{14}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{15}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability of $\textsc{Game}_{14}(\kappa)$ and $\textsc{Game}_{15}(\kappa)$ follows from the fact that in both cases the messages depend on exactly the same values. Essentially, the same code is executed, only by different machines; the game hop is only a cosmetically one.

In more detail, the situation looks as follows:

User Registration, U corrupted, O honest. Here, the definition of $\mathcal{F}_{\mathsf{BKA}}$ and $\mathcal{S}_{15}$, who calls $\mathcal{F}_{\mathsf{BKA}}$ with the correct inputs x_{U}, implies correct behavior. The input x_{U} is correct due to the extractability and the pid is correct as a wrong pid would require breaking the security of the ZK scheme or that of the implicit identification scheme by computing the secret key sk_{U} given only the public key pk_{U} stored in $\mathcal{F}_{\mathsf{Reg}}$.

Bookkeeping, U corrupted, O honest. In $\textsc{Game}_{14}(\kappa)$, each user maintains the logbook λ with the corresponding User History UH directly. In $\textsc{Game}_{15}(\kappa)$, $\mathcal{F}_{\mathsf{BKA}}$ executes the same function f on the inputs to $\mathcal{F}_{\mathsf{BKA}}$, that the simulator computed in $\textsc{Game}_{14}(\kappa)$. In $\textsc{Game}_{15}(\kappa)$, the correct—and latest—input is used by definition of the ideal functionality. The transfer values $(\vec{\alpha}, \vec{s}, \vec{a})$ are output to U and hence visible to $\mathcal{S}_{15}$ and the commitment and decommitment on $\vec{a}$ is computed directly by $\mathcal{S}_{15}$. If this value would have been the same in $\textsc{Game}_{14}(\kappa)$ then indistinguishability for any execution of the BK-task trivially follows; the best any PPT-environment $\mathcal{Z}$ could do here is to guess.

So assume that there is some set of inputs that enables $\mathcal{Z}$ can differentiate between $\textsc{Game}_{14}(\kappa)$ and $\textsc{Game}_{15}(\kappa)$ notably better than guessing based on the BK task. Since the computation performed by the simulator in $\textsc{Game}_{14}(\kappa)$ is exactly the same as the one $\mathcal{F}_{\mathsf{BKA}}$ does in $\textsc{Game}_{15}(\kappa)$ and $\mathcal{Z}$ cannot lie about x_{U} (as it is input into $\mathcal{F}_{\mathsf{MPC}}$), the only way $\mathcal{Z}$ could try to win here is by providing different input UH. There are different ways $\mathcal{Z}$ could achieve that:

- $\mathcal{Z}$ provides a wrong proof regarding σ_{UH}. Then, we could build an environment that either forges a signature σ_{UH} and uses an honest witness, or fakes a proof π and uses a false witness. The former would contradict our EUF-CMA requirement for $\textsc{Sig}$, the latter would contradict the soundness-property required for $\textsc{Zk}$.
- $\mathcal{Z}$ provides the correct information of a different corrupted user U′. We assumed $\textsc{Sig}$ to be EUF-CMA-secure, thus assuring unforgeability of the signature σ_{UH} on λ. Hence, $\mathcal{S}_{15}$ would extract the public key pk_{U} of U′ and look up a tuple $(\mathsf{pk}_{\mathsf{U}}, pid_{\mathsf{U}})$ and provide input to $\mathcal{F}_{\mathsf{BKA}}$ in the name of U′ who corresponds to the pid pid_{U}. Thus, $\mathcal{F}_{\mathsf{BKA}}$ uses the same UH that would have been used by the simulator in $\textsc{Game}_{14}(\kappa)$.

Hence, assuming that $\mathcal{F}_{\mathsf{BKA}}$ internally updates UH correctly (which we will show for the other tasks as well), this change cannot be used to increase the chance of $\mathcal{Z}$ to differentiate.

Outsource, U corrupted, O and H honest. Here, too, indistinguishability trivially follows from the correctness of inputs; we merely copied the reconstruction of shares for this scenario from OA to OS. Those are input directly into $\mathcal{F}_{\mathsf{BKA}}$, where they are used later. The shares $rs_{x_{\mathsf{U}}}^{(\mathsf{H})}$ and $rs_{x_{\mathsf{U}}}^{(\mathsf{O})}$ are either correct, in which case $\mathcal{F}_{\mathsf{BKA}}$ will use them accordingly during OA. Or they are not, in which case $\mathcal{S}_{15}$ sets $x_{\mathsf{U}} \coloneqq \bot$; the reconstruction $\textsc{Game}_{14}(\kappa)$ would fail during the OS-task, which also happens in $\textsc{Game}_{15}(\kappa)$. With the two parties performing OA being honest, no further problems arise during the subsequent execution. The extraction of pk_{U} further removes the ability of $\mathcal{Z}$ to cheat by letting $\mathcal{S}_{15}$ send input to $\mathcal{F}_{\mathsf{BKA}}$ in the name of the wrong user.

Outsource, U and O honest, H corrupted. In this case, it is not possible to cheat without being detected. By knowing the input $\mathcal{Z}$ would have given to H, $\mathcal{S}_{15}$ can mimic the behavior of an honest dummy helper and forward it into $\mathcal{F}_{\mathsf{BKA}}$.

Outsource, U and H corrupted, O honest. The correct user U can be determined via extraction of π, so no environment $\mathcal{Z}$ cannot use two different users for the two games here. However, in this task, $\mathcal{S}_{15}$ information-theoretically cannot determine the correct input x_{U}, as it only sees one part $rs_{x_{\mathsf{U}}}^{(\mathsf{H})}$ of the additive sharing—the second part, $rs_{x_{\mathsf{U}}}^{(\mathsf{O})}$, is sent between two corrupted parties and hence not visible for $\mathcal{S}_{15}$. However, in the subsequent OA execution, $\mathcal{S}_{15}$ learns the shares that the user sent to the helper through the inputs that the helper inserts into $\mathcal{F}_{\mathsf{MPC}}$. There, $\mathcal{S}_{15}$ can reconstruct x_{U}. Note that there is no difference between $\mathcal{S}_{15}$ learning the input during the OS task and inserting it into $\mathcal{F}_{\mathsf{BKA}}$ right away and $\mathcal{S}_{15}$ learning it during the OA task, since $\mathcal{F}_{\mathsf{BKA}}$ allows $\mathcal{S}_{15}$ to update the input x_{U} for all corrupted users before starting the computation during Outsourced Analytics. Thus, the same input x_{U} is used in both games.

Outsourced Analytics, H corrupted, O honest. Here, $\mathcal{S}_{15}$ still uses the same shares for the operator in both games and receives (and verifies) all helper-shares input into $\mathcal{F}_{\mathsf{MPC}}$ in both games, there is no direct change here. The only new thing is the equivocation of x_{U} for corrupted users. This change was already discussed above; the remaining protocol remains equivalent, since $\mathcal{S}_{15}$ only does in $\textsc{Game}_{15}(\kappa)$ what honest parties would do in $\textsc{Game}_{14}(\kappa)$.

Update, U corrupted, H and O honest. The values that determine the messages are still equivalent in both games. The values that the simulator stores for Upd during OA for the helper and operator are the same, so both just follow the same protocol. As nothing is done with the output of $\mathcal{F}_{\mathsf{BKA}}$, distinguishing here is not possible.

Update, U and O honest, H corrupted. The linking number that was sent as leak by the user to the simulator in $\textsc{Game}_{14}(\kappa)$ trivially equals the linking number that

$\mathcal{S}_{15}$ obtains by looking up $(\mathit{ssid}, \mathit{lin})$ for the leaked ssid. During the OS task, assuming lin is sampled from a sufficiently large space, the probability that the entry $(\mathit{ssid}, \mathit{lin})$ being unique becomes overwhelming. During the Upd task, this does not change. If this wouldn't be the case, $\mathcal{Z}$ would have to create a duplicate lin, which, with Blum coin toss, is possible only with negligible probability. Even then, $\mathcal{S}_{15}$ in $\textsc{Game}_{15}(\kappa)$ would use the correct linking number lin. Note further that $\mathcal{Z}$ has no way on how to lie about ssid.

Update, U and H corrupted, O honest. Here, the simulator only has to ensure that the interaction between the operator and the user is canonical. Again, nothing here depends on the output of $\mathcal{F}_{\mathrm{BKA}}$, it is only called to keep the functionality in a consistent space. Hence, the changes induced here provide no advantage to $\mathcal{Z}$ in distinguishing the two games.

□

$\textsc{Game}_{16}(\kappa)$: Extends the state of the simulator by two additional lists of tuples which represent what the honest operator would store during OS and what the helpers would store for OA. For the operator the simulator stores tuples $(\mathit{pid}_{\mathrm{H}}, \mathrm{f}, \mathit{lin}, \mathit{rs}^{(\mathrm{O})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{O})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{O})}_{\mathit{otp}})$ and for the helpers, $\mathcal{S}_{16}$ stores tuples $(\mathit{pid}_{\mathrm{H}}, \mathrm{f}, \mathit{lin}, \mathit{rs}^{(\mathrm{H})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{H})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{H})}_{\mathit{otp}})$.

The map is used during the tasks for OS and OA:

Outsource, U, H and O honest. $\mathcal{S}_{16}$ adds a vector of empty entries $(\bot, \ldots, \bot)$ to both lists.

Outsource, U corrupted, H and O honest. $\mathit{lin}^{\mathrm{new}}$ is now randomly sampled by $\mathcal{S}_{16}$, which replaces the coin-toss of O and H.

After $\mathcal{Z}$ instructed a corrupted party to send $(\mathit{rs}^{(\mathrm{O})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{O})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{O})}_{\mathit{otp}}, \mathit{ser}^*, \mathtt{com}_{\mathit{ser}}, \mathtt{com}_{\mathit{ser}_0}, \mathtt{com}_{\mathsf{sk}_{\mathrm{U}}}, \pi)$ to the operator via $\mathcal{F}_{\mathrm{ORR}}$, $\mathcal{S}_{16}$ adds a new entry $(\mathit{pid}_{\mathrm{H}}, \mathrm{f}, \mathit{lin}, \mathit{rs}^{(\mathrm{O})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{O})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{O})}_{\mathit{otp}})$ to the list corresponding to the view of the operator, and after receiving instructions from $\mathcal{Z}$ to send a message $(\mathit{rs}^{(\mathrm{H})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{H})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{H})}_{\mathit{otp}})$ from the user to the helper, $\mathcal{S}_{16}$ adds a new entry $(\mathit{pid}_{\mathrm{H}}, \mathrm{f}, \mathit{lin}, \mathit{rs}^{(\mathrm{H})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{H})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{H})}_{\mathit{otp}})$ to the view of the helper.

Outsource, U and O honest, H corrupted. When the honest user is supposed to create shares of UH, x_{U} and otp, $\mathcal{S}_{16}$ creates shares of the zero-vector using $\mathtt{Share}(\vec{0})$ and uses those in the same way the user uses the actual shares in the protocol. After having the linking number lin created honestly, $\mathcal{S}_{16}$ stores the values for the operator only and does not store any additional values for the view of the helper.

Outsource, U and H corrupted, O honest. $\mathcal{S}_{16}$ follows the protocol of O regarding the stored information for OA. That is, after $\mathcal{Z}$ instructed a corrupted party to send $(\mathit{rs}^{(\mathrm{O})}_{\mathit{UH}}, \mathit{rs}^{(\mathrm{O})}_{x_{\mathrm{U}}}, \mathit{rs}^{(\mathrm{O})}_{\mathit{otp}}, \mathit{ser}^*, \mathtt{com}_{\mathit{ser}}, \mathtt{com}_{\mathit{ser}_0}, \mathtt{com}_{\mathsf{sk}_{\mathrm{U}}}, \pi)$ to the operator via $\mathcal{F}_{\mathrm{ORR}}$, and after $\mathcal{S}_{16}$ took the role of the operator in honestly computing the linking number, the simulator adds $(pid_{\mathsf{H}}, \mathsf{f}, lin, rs^{(\mathsf{O})}_{UH}, rs^{(\mathsf{O})}_{x_{\mathsf{U}}}, rs^{(\mathsf{O})}_{otp})$ to its view of the operator.

Lemma 8.2.15. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{15}(\kappa)$ from $\textsc{Game}_{16}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{15}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{16}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We claim indistinguishability based on the fact that the two distributions from $\textsc{Game}_{15}(\kappa)$ and $\textsc{Game}_{16}(\kappa)$ are indistinguishable.

First, notice that the binary outcome of a Blum coin toss and a uniformly random bit cannot be differentiated better that by guessing, so this change doesn't provide any distinguishing advantage.

To support our claim also regarding the new lists of outsourced information for OA, let's consider all possible corruption cases:

H honest, O honest. In case the user is also honest, the functionality $\mathcal{F}_{\mathsf{BKA}}$ directly obtains input from the respective parties during the Outsource task, which causes $\mathcal{F}_{\mathsf{BKA}}$ to load the correct UH UH and to use the input x_{U} provided by that user. Hence, the simulator has to only remember *that* honest parties provided input, not *what* these inputs were. Thus, inserting the special symbol $\perp$ to keep the list size consistent suffices, since those values are never used again.

In case the user is corrupted the environment $\mathcal{Z}$ has to send shares $rs^{(\mathsf{H})}_{(\cdot)}$ and $rs^{(\mathsf{O})}_{(\cdot)}$ in the name of the user to the helper and the operator, respectively. Neither of them are corrupted, so $\mathcal{S}_{16}$ can see both messages. Since neither the operator nor the helper have secret inputs here, $\mathcal{S}_{16}$ can follow the honest protocols, thus producing the same distribution.

H corrupted, O honest. If the user is honest then $\mathcal{S}_{16}$ has to send messages containing valid shares of Us input, without actually knowing the input. Thus, $\mathcal{S}_{16}$ distributes zero-shares; the environment $\mathcal{Z}$ can see only the part sent to the helper, not the ones sent to the operator. Obviously, no environment $\mathcal{Z}$ can distinguish its part of the zero-sharing obtained in $\textsc{Game}_{16}(\kappa)$ from a valid sharing of the respective user input from $\textsc{Game}_{15}(\kappa)$ better than by randomly guessing; the additive shares information-theoretically hide the original message and the commitments to the other share are hiding so that they don't reveal the respective other share. Hence, those distributions look equivalent. The simulator follows the protocol of the operator regarding the stored information honestly, which causes no difference in the distributions.

Against a *corrupted* user U, $\mathcal{S}_{16}$ directly follows the protocol of the operator, thus causing the same distribution.

Note that the final corruption case where both the helper and the operator are corrupted is excluded because we assume that both cannot be corrupted at the same time. Since all possible cases cause indistinguishable distributions, our claim follows. □

GAME$_{17}(\kappa)$**:** All remaining honest parties are replaced by dummy parties, which, upon receiving input by $\mathcal{Z}$, only forward their input into $\mathcal{F}_{\mathsf{BKA}}$.

Lemma 8.2.16. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{16}(\kappa)$ from GAME$_{17}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\mathit{GAME}_{16}(\kappa)) = 1] - \Pr[\mathcal{Z}(\mathit{GAME}_{17}(\kappa)) = 1]| \in \mathrm{negl}(\kappa)$$

Proof. Again, indistinguishability follows from the fact that the messages sent are still the same. All the leaks sent by honest parties to the simulator in GAME$_{16}(\kappa)$ are ignored, the messages are independent of any leaks still sent by the semi-dummy parties: the only change between GAME$_{16}(\kappa)$ and GAME$_{17}(\kappa)$ is with respect to the auxiliary inputs x_{O} and x_{U}. Both are input to $\mathcal{F}_{\mathsf{BKA}}$ directly by the respective dummy party and used there accordingly. Hence, the change induced by GAME$_{17}(\kappa)$ doesn't change the view of $\mathcal{Z}$ at all, which makes indistinguishability trivial. □

GAME$_{18}(\kappa)$**:** All messages between honest parties are simulated by having $\mathcal{S}_{18}$ report messages of correct length, but which only contain zero-bits. Consequently, all operations that do not result in messages are removed.

Lemma 8.2.17. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing GAME$_{17}(\kappa)$ from GAME$_{18}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\mathit{GAME}_{17}(\kappa)) = 1] - \Pr[\mathcal{Z}(\mathit{GAME}_{18}(\kappa)) = 1]| \in \mathrm{negl}(\kappa)$$

Proof. As messages between the operator and helper are sent via $\mathcal{F}_{\mathsf{SMT}}$ (as are messages between user and operator during UReg) those messages are secure, and changing them to the all-zero string cannot be detected by definition as the adversary only learns the messages length (see Section 4.5.2.3). For *anonymous* messages all messages are sent through $\mathcal{F}_{\mathsf{ORR}}$ (see Section 4.5.2.4) where the respective functionality is designed such that leaks that contain the *message* only occur if: (1) at any point, all remaining mix-servers *and* the receiver are corrupted, or (2) the sender is corrupted. Otherwise, the adversary only learns a random *id* ϑ alongside the next consecutive group of corrupted mix servers until the next honest server.

Fortunately, we can rule out both cases that leak the message by requirement. We only replace messages for honest-honest communication, *i.e.* communication wherein both parties are honest. Hence, neither is the sender corrupted, nor the receiver.

Thus the message is never leaked by $\mathcal{F}_{\mathsf{ORR}}$ to the adversary, which makes the change from a *protocol* message to a *zero* message of appropriate size undetectable for any environment $\mathcal{Z}$.

□

$\textsc{Game}_{19}(\kappa)$: During the UReg-task with an honest user, instead of honestly creating a user secret key sk_U and computing a public key from it, $\mathcal{S}_{19}$ samples a random public key $\mathsf{pk}_\mathsf{U} \overset{\$}{\leftarrow} G_1$ directly.

Lemma 8.2.18. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{18}(\kappa)$ from $\textsc{Game}_{19}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{18}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{19}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. By assumption on the group $pp_{\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_t}$ a public key is uniformly distributed in $\mathbb{G}_1$. Any environment $\mathcal{Z}$ distinguishing $\textsc{Game}_{18}(\kappa)$ and $\textsc{Game}_{19}(\kappa)$ based on pk_U would violate this assumption. The simulator now lacks knowledge of the corresponding secret key sk_U. However, note that sk_U is never used in $\textsc{Game}_{18}(\kappa)$, so $\mathcal{S}_{19}$ can create a similar distribution in $\textsc{Game}_{19}(\kappa)$ independently of sk_U. □

$\textsc{Game}_{20}(\kappa)$: If the helper is corrupted and the user and operator are honest during the execution of the OS task, $\mathcal{S}_{20}$ changes its behavior. Instead of verifying the entire logbook by checking the commitments and signature, $\mathcal{S}_{20}$ only verifies that the linking number lin received by the helper confirms with the two shares $lin^{(\mathsf{O})} + lin^{(\mathsf{H})}$.

Lemma 8.2.19. *For all PPT environments $\mathcal{Z}$ the advantage for distinguishing $\textsc{Game}_{19}(\kappa)$ from $\textsc{Game}_{20}(\kappa)$ fulfills:*

$$|\Pr[\mathcal{Z}(\textsc{Game}_{19}(\kappa)) = 1] - \Pr[\mathcal{Z}(\textsc{Game}_{20}(\kappa)) = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability of the two games easily follows from the fact that in $\textsc{Game}_{19}(\kappa)$, the only corrupted party is the helper and the only value depending on the helper is lin. Hence, verification only fails iff the helper sent a wrong linking number lin. This is still the case in $\textsc{Game}_{20}(\kappa)$, thus making indistinguishability trivial. □

Thus, with $\textsc{Game}_{1}(\kappa)$ being the real world and $\textsc{Game}_{20}(\kappa)$ corresponding to the ideal world with a simulator acting as described above, we have proven the protocol Π_{BKA} to be as secure as the functionality $\mathcal{F}_{\mathsf{BKA}}$.

Thus, we can now finally prove our final security statement:

Corollary 8.2.20 (System Security)**.** *For all environments $\mathcal{Z}$ who statically corrupted a subset $\mathsf{U}' \subseteq \mathsf{U}$ and the helper H, it follows that if the building blocks are instantiated as required in Section 5.3:*

$$\Pi_{BKA}^{\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{Reg}},\mathcal{F}_{\mathsf{SMT}},\mathcal{F}_{\mathsf{MPC}},\mathcal{F}_{\mathsf{ORR}}} \geq \mathcal{F}_{\mathsf{BKA}}$$

We have shown in Lemma 8.2.1 to Lemma 8.2.19 that for an honest operator, the simulator $\mathcal{S}$ acting in the ideal world can provide a view for $\mathcal{Z}$ that is indistinguishable from a real execution of the protocol:

$$\mathtt{view}_{\mathcal{Z},\mathcal{A},\Pi_{BKA}} \approx^{c} \mathtt{view}_{\mathcal{Z},\mathcal{S},\mathcal{F}_{\mathsf{BKA}}}$$

Thus, by combining Corollaries 8.1.22 and 8.2.20, our main claim follows:

Corollary 8.2.21 (Security)**.** *Assuming the building blocks from Section 5.3, then for all environments $\mathcal{Z}$ who do not corrupt the operator* O and *the helper* H *at the same time, it holds that*

$$\Pi_{BKA}^{\mathcal{F}_{\text{CRS}},\mathcal{F}_{\text{Reg}},\mathcal{F}_{\text{SMT}},\mathcal{F}_{\text{MPC}},\mathcal{F}_{\text{ORR}}} \geq \mathcal{F}_{\text{BKA}}$$

We thus have proven our protocol Π_{BKA} to be at least as secure as the ideal functionality $\mathcal{F}_{\text{BKA}}$.

9. Example Applications

In this section we sketch two privacy-preserving applications, namely fraud detection for mobile payments and a targeted advertisement network, and show how they can be instantiated as a function for $\mathcal{F}_{\mathsf{BKA}}$.

However, we only copied this section from the paper of Fetzer et al. [70, Section 4 and Appendix F] with a few minor adjustments to compensate for the changes applied to the protocol and functionality in this dissertation compared to the conference paper. So we stress that this chapter does not count as contribution of this dissertation and is only included to provide a better intuition on how instantiations of the function f can look like.

9.1. Fraud Detection for Mobile Payments

In the European Central Bank's latest report on card fraud [19] published in 2018, the total value of fraudulent transactions at points-of-sale in 2016 amounted to about 342 Million Euro. This number sounds high, yet it only amounts to 0.008% of the overall card transaction value. This small ratio is achieved mainly thanks to the use of "strong authentication" methods like Chip&Pin as well as fraud detection mechanisms as required by the 2nd European Payment Services Directive (PSD2).

We define a privacy-preserving mobile payment system including fraud detection capabilities. The operator in this system is the bank that offers the mobile payments service to its customers. These are the users in our system who interact using a smartphone App. Fraud detection consists in monitoring a customer's transactions for anomalies or typical fraud patterns. This can be done based on simple rules or sophisticated machine learning algorithms. Due to the real-time requirements, the combination of fraud detection with privacy for mobile payments is particularly challenging. To this end, we consider the following two-tier mechanism: We force the user to perform a more complex machine learning based fraud detection with the operator if some threshold of payment transactions has been reached, resulting in some risk level, and a simple but faster rule-based fraud detection during each payment at a point-of-sale. The latter takes the risk level into account and decides whether the current transaction is accepted or declined.

Function $\mathsf{f}_{\text{mpayment}}$

```
Initialize($fp := \bot, x_{\mathsf{U}} := (rsk, rem, max), x_{\mathsf{O}} := (rsk, rem, max)$)
if $x_{\mathsf{U}} \neq x_{\mathsf{O}}$ then abort
$UH := (\bot, \ldots, \bot, rsk, rem, max, 0)$
return $(UH, y_{\mathsf{U}} := \mathsf{Ok}, y_{\mathsf{O}} := \mathsf{Ok})$

Top-Up($fp := \bot, UH, x_{\mathsf{U}} := val, x_{\mathsf{O}} := val$)
if $x_{\mathsf{U}} \neq x_{\mathsf{O}}$ then abort
return $\left(\vec{\alpha} := \begin{pmatrix} 0 & \ldots & num_e-1 \\ 0 & \ldots & num_e-1 \end{pmatrix}, \vec{s} := (\bot, \ldots, \bot), \vec{a} := (0, \ldots, 0, val), y_{\mathsf{U}} := \mathsf{Ok}, y_{\mathsf{O}} := \mathsf{Ok}\right)$

Payment($fp := \bot, UH, x_{\mathsf{U}} := (ts, loc, type, tval), x_{\mathsf{O}} := (ts, loc, type, tval)$)
if $x_{\mathsf{U}} \neq x_{\mathsf{O}}$ then abort
$(t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal) := UH$
if $bal < tval$ then
return $\left(\vec{\alpha} := \begin{pmatrix} 0 & \ldots & num_e-1 \\ 0 & \ldots & num_e-1 \end{pmatrix}, \vec{s} := (\bot, \ldots, \bot), \vec{a} := (0, \ldots, 0), y_{\mathsf{U}} := \bot, y_{\mathsf{O}} := \bot\right)$
$acc := \mathsf{f}_{\text{fdsimple}}(UH, (ts, loc, type, tval))$
$(t_1^{\text{new}}, \ldots, t_5^{\text{new}}) := (acc, ts, loc, type, tval)$
if $acc = 1$ then $bal^{\text{diff}} := tval$
else $bal^{\text{diff}} := 0$fi
$\vec{\alpha} := \begin{pmatrix} 0 & \ldots & 4 & 5 & \ldots & 9 & \ldots & 5T-5 & \ldots & 5T-1 & 5T & \ldots & 5T+3 \\ 5T-5 & \ldots & 5T-1 & 0 & \ldots & 4 & \ldots & 5T-10 & \ldots & 5T-6 & 5T & \ldots & 5T+3 \end{pmatrix}$
$\vec{s} := (t_1^{\text{new}}, \ldots, t_5^{\text{new}}, \bot, \ldots, \bot)$
$\vec{a} := (0, \ldots, 0, 0, -1, 0, -bal^{\text{diff}})$
return $(\vec{\alpha}, \vec{s}, \vec{a}, y_{\mathsf{U}} := acc, y_{\mathsf{O}} := acc)$

Risk Calculation($fp, (UH, x_{\mathsf{U}} := \bot), x_{\mathsf{O}} := \bot$)
$(t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal) := UH$
$(rsk^{\text{new}}, rem^{\text{new}}, max^{\text{new}}) := \mathsf{f}_{\text{fdcomp}}(fp, UH)$
$\vec{s} := (\bot, \ldots, \bot, rsk^{\text{new}}, \bot, max^{\text{new}}, \bot)$
$\vec{a} := (0, \ldots, 0, 0, -rem + rem^{\text{new}}, 0, 0)$
return $\left(\vec{\alpha} := \begin{pmatrix} 0 & \ldots & num_e-1 \\ 0 & \ldots & num_e-1 \end{pmatrix}, \vec{s}, \vec{a}, y_{\mathsf{U}} := \mathsf{Ok}\right), y_{\mathsf{O}} := \mathsf{Ok}$
```

Figure 9.1.: Instantiation of f for privacy-preserving mobile payments with fraud detection.

Each time the user conducts a payment at a point-of-sale, a new transaction is created and stored in the UH. We assume that a transaction record t consists of the following data encoded as vector of $\mathbb{Z}_o$ elements:

$$t := (acc, ts, loc, type, tval),$$

where acc is one iff the transaction has been accepted, ts is a timestamp, loc indicates the geographic location the transaction took place, $type$ describes the type of shop (*e.g.*, grocery store, jewelry store, etc.), and $tval$ is the transaction value. We stress that this is only an example, adding other attributes is pretty straightforward.

The UH contains the latest T transaction records, the user's balance bal and some important additional information to support the fraud detection mechanisms. These additional information include the account's current risk level rsk, a maximum value max a single transaction can have, and a limit rem on the number of payment transactions a user can perform before the complex fraud detection mechanism has to be run. Note that the latter two values depend on the current risk level.

Thus, the UH has the following form:

$$UH := (t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal)$$

The first $5T$ slots store the last T transactions (each transaction t^i requires 5 slots), t^1 is the most recent transaction. The UH has a total length of $5T + 4$ entries.

We now describe the individual tasks a user can perform. The details of the function $\mathsf{f}_{\text{mpayment}}$ can be found in Fig. 9.1.

Registration. During UReg the user obtains an empty UH with an empty balance and no stored transactions using the User Registration task. We assume that right after that, the user and the operator immediately call the BK task with the initial registration (thus making it linkable, but since the values are known this level of leakage is not a problem since all following interactions are anonymous). Both parties input the initial values for the risk level, the remaining number of transactions and the maximum transaction value, *i.e.*, (rsk, rem, max). The values rem and max can either be fixed constants or depend on the risk level. Alongside empty transactions, those are written into the new UH.

Top-Up. The user can top-up the balance of his account, which is internally realized with a BK task. As the mobile payments service is prepaid-based it is important that a user can top-up the balance. We propose a general method where user and bank agree on the amount that should be topped up and leave the actual transfer of money to the implementation method, such as anonymously depositing money at an ATM or making a transfer from the normal bank account (which would be identifying). The user invokes a BK task where both parties input the amount to be deposited, which is added to the user's balance. In our example, a top-up transaction is not recorded in the user's transaction history (although this might be reasonable). Note that only an addition is needed to update the UH here.

Payment (with simple fraud detection). Before a payment is conducted, it is first checked whether the payment is allowed or the threshold of payment transactions has been reached and a risk calculation has to be performed first. Then, a light-weight fraud detection mechanism is executed (based on the risk level). Depending on the result the payment is either accepted or denied. To issue the actual payment, the user communicates with a point-of-sale that has a communication channel with the bank and forwards all messages between the user and the bank. The payment is performed by using the BK task where both parties input all transaction details excluding the acceptance bit, *i.e.*, timestamp, location, type of shop and transaction value. The function aborts if the transaction value exceeds the account's balance. Otherwise, the simple rule-based fraud detection mechanism $\mathsf{f}_{\mathsf{fdsimple}}$ is executed to decide whether the transaction is accepted or not based on the last T transactions, the risk level *rsk*, the remaining number of transactions *rem*, the maximum transaction value *max*, and the details of the current transaction. In our example implementation in Section 10.4 we verify that the following conditions are all satisfied: 1. $tval \leq max$ (the transaction value does not exceed the allowed amount) 2. $rem > 0$ (the number of payment transactions the user can perform before the complex fraud detection mechanism has to be run has not exceeded its limit). Of course, additional checks could be included: The transaction could be denied if the risk level is medium but there are more than three transactions within a 10 minute period, or if two consecutive transactions differ in their location so much that no user could have possibly traveled that far in such a short time period.

The simple mechanism already provides a limited fraud protection which is lightweight enough to be computed by the user's resource-constrained device. As those simple mechanisms can be public, the user can evaluate the rule-based fraud detection by itself and provide the point-of-sale with a ZK proof that it evaluated the mechanism correctly based on its logbook. This can significantly speed up the payment process compared to an MPC-based computation, assuming the rules are simple and efficiently compatible with the ZK proof system.

Only if the transaction is accepted it is physically executed and the user's balance gets updated accordingly. More specifically, the three output maps of f look as follows: The permutation shifts all past transactions to the right to make room for the new transaction, which is written into the UH with the direct update. This is done because the UH only stores the last T transactions to hide the total number of transactions. The additive increment then subtracts one from the number of remaining transactions *rem* and subtracts the transaction value *tval* from the balance *bal* iff the transaction was accepted. The risk level and the maximum transaction value stay the same.

More precisely, the new transaction is assembled as $(t_1^{\text{new}}, \ldots, t_5^{\text{new}}) = (acc, ts, loc, type, tval)$ and the new UH then looks as follows after applying the three maps: $UH^{\text{new}} := (t_1^{\text{new}}, \ldots, t_5^{\text{new}}, t_1^1, \ldots, t_5^1, \ \ldots\ , t_1^{T-1}, \ldots, t_5^{T-1}, rsk, rem - 1, max, bal^{\text{new}})$.

Risk Calculation (with complex fraud detection). Each time the user executes a payment transaction, the counter for the number of remaining transactions decreases by one. When this counter reaches zero, it forces the user to participate in a more complex fraud detection algorithm, where the risk level gets updated and a more sophisticated fraud detection algorithm is executed. We assume this complex fraud detection mechanism to be based on machine learning (*e.g.*, logistic regression, as suggested by [98, 91]). Since this is a computationally expensive operation, this task is realized with the OA task. The fraud detection mechanism takes all transactions in the UH as well as the current risk level into account and computes a new risk level.

By choosing a suitable value for the initial value of that counter, we can ensure that users regularly participate in the complex fraud detection mechanism. The operator inputs its FPs *fp* into OA. The fraud detection mechanism then computes the user's new risk level *rsk* along with a new maximum number of transactions *rem*, and maximum transaction value *max*. These new values are then stored in the UH. More specifically, only the direct update and additive increment are needed. The direct update sets the new values for the risk level and the new maximum transaction value at the corresponding slots and overwrites the old values in the process. Since the outsourcing triple is non-blocking regarding Bookkeeping operations, we have to take into account that the value of the remaining number of transaction may have changed since Outsource was called. Therefore, the additive increment adds the difference between the old remaining number of transactions (from the point when Outsource was called) and the new value to the corresponding slot.

9.2. Targeted Advertisement System

We now briefly sketch a targeted mobile advertisement system which can optionally be used as an extension for loyalty systems. For their cooperation, users are rewarded with vouchers targeted at their purchase behavior. The central idea is that the user's purchases are stored in the UH. From time to time, users submit their purchase history to the operator, who analyzes it together with the histories of several other users. The user is rewarded with a voucher targeted at the user's probable interests and is displayed alongside a suitable ad in the smartphone app.

We now assume that the operator acts as a conglomeration of supermarket chains and further participating shops. In the following, we describe how to use our framework in this scenario. The function is depicted in Fig. 9.2.

Registration. Upon registration with the UReg task the user obtains an empty UH. Each slot in the UH represents a product category, *e.g.*, "vegetables", "candy", or "fast food". The UH tracks the amount of money spent in each category.

```
**Function** $\mathsf{f}_{\text{advertising}}$
Checkout($fp := \perp, UH, x_{\mathsf{U}} := \{prod_j\}_{j=1}^{num_i}, x_{\mathsf{O}} := \{prod_j\}_{j=1}^{num_i}$)
**if** $x_{\mathsf{U}} \neq x_{\mathsf{O}}$ **then abort**
$\vec{a} := (\vec{a}[0], \ldots, \vec{a}[num_c - 1])$ with $\vec{a}[i] := \sum_{j \in I} price_j$
where $I$ is the set containing all indices $j$ with $pos(categ_j) = i$.
$\vec{\alpha} := \begin{pmatrix} 0 & \ldots & num_c - 1 \\ 0 & \ldots & num_c - 1 \end{pmatrix}$
$\vec{s} := (\perp, \ldots, \perp)$
**return** $(\vec{\alpha}, \vec{s}, \vec{a}, categ := \perp, y_{\mathsf{U}} := \mathsf{Ok}, y_{\mathsf{O}} := \mathsf{Ok})$

Analytics($fp, \{(UH_\zeta, x_{\mathsf{U}_\zeta} := \perp)\}_{\zeta \in [Z_{\mathsf{f}}]}, x_{\mathsf{O}} = \perp\}$)
$\{categ_\zeta\}_{\zeta \in [Z_{\mathsf{f}}]} := \mathsf{f}_{\text{advertising}}(fp, \{UH_\zeta\}_{\zeta \in [Z_{\mathsf{f}}]})$
**for** $\zeta \in \{1, \ldots, Z\}$ **do**
    $\vec{\alpha}_\zeta := \left\{ \begin{matrix} 0 & \ldots & num_c - 1 \\ 0 & \ldots & num_c - 1 \end{matrix} \right\}$,
    $\vec{s}_\zeta := (\perp, \ldots, \perp)$,
    $\vec{a}_\zeta := (0, \ldots, 0)$,
**done**
**return** $(\{(\vec{\alpha}_\zeta, \vec{s}_\zeta, \vec{a}_\zeta, y_{\mathsf{U}_\zeta} := (\mathsf{Ok}, categ_\zeta))\}_{\zeta \in [Z_{\mathsf{f}}]}, y_{\mathsf{O}} := \mathsf{Ok})$
```

Figure 9.2.: Instantiation of f for a privacy-preserving targeted advertisement system.

Checkout. When purchasing goods at a participating store, the user updates the purchase history using the BK task. The amount of money spent in each category is calculated and added to the corresponding slots in the UH. This only requires additive updates, meaning that the steps required for permutation and direct updates can be skipped.

Analytics. The Outsourced Analytics task lets the user provide data for analytical purposes. We assume the operator has an analytical function (for example for marketing analyses) which takes some FPs and multiple UHs as input and assigns to each UH a class that describes the most likely interests of the corresponding user. The user is rewarded with a voucher that matches this class and that can be redeemed at a participating shop. Additionally, the user gets a matching advertisement. For example, if the analysis reveals that the user likes chocolate, the user obtains advertisements for a new kind of chocolate and a voucher for a 10% discount on chocolate. The UHs of the participating users remain unchanged.

If the targeted advertisement system is interconnected with a loyalty system, the user could also earn loyalty points instead of vouchers.

10. Implementation

We evaluated the practicality of our protocol by measuring execution times of a practical implementation. The implementation is (1) not implemented by the author of this thesis and hence *not* a contribution for this dissertation, and (2) based on the conference version from Fetzer et al. [70], which has a few changes in comparison to this part of this thesis, mostly in regards to the communication model. Since network communication depends on various external factors, the communication times have been ignored in the evaluation.

10.1. Setup

Evaluation of the user side has been done on a Nexus 5X smartphone released in 2015 featuring a Snapdragon 808 with two cores with 1.8GHz and four cores with 1.4GHz. The phone was running with Android 8.1.0. Throughout this chapter we refer to the Nexus smartphone as *Phone 1*. The second phone is a Galaxy S8 smartphone released 2017 featuring an Exynos 8895 with four cores running with 2.3GHz and four cores with 1.7GHz running Android 9. We will refer to this phone as *Phone 2* from now on. We executed the code for operator and helpers on much more powerful servers, equipped with an AMD Ryzen 9 3900X with 12 cores with 3.8GHz each. In all cases our implementation makes use of 6 threads to speed up cryptographic operations.

The protocol has been implemented our protocol in C++17, employing the open-source library RELIC toolkit v0.5.0 [13] for group operations. The required building blocks were instantiated as suggested in Section 5.3: For *signatures* we use the scheme from [1], for *commitments* we implemented [2], and for the NIZKPoK scheme we use the method from [83, 67]. Our building blocks are instantiated over the pairing-friendly Barreto–Naehrig Curves Fp254BNb and Fp254n2BNb [14, 20].

10.2. Evaluation

User Registration, Bookkeeping, Outsource, and Update. We averaged over 50 executions of each protocol task for UH sizes of 10, 100, 200, 400, 600, 800 and 1000 in order to get representative results. For BK we implemented both the three-stage update comprising a permutation of the UH, setting values, and adding values, as well as a simplified version where no permutation and direct update are done. The concrete choice of permutation has no impact on performance, whereas performance improves slightly the more entries

$\lvert UH\rvert$	**BK**		**OS**			**Upd**		
	$\sum$	ZK	$\sum$	ZK	H	$\sum$	ZK	H
10	22	16	20	10	4	20	12	< 1
100	57	51	30	10	15	54	44	< 1
200	96	90	42	10	26	93	80	< 1
400	176	170	60	10	44	172	152	< 1
600	254	248	79	10	65	251	225	< 1
800	332	327	97	10	84	331	299	< 1
1000	411	405	117	10	105	408	370	< 1

Table 10.1.: Execution times in *ms* for operator and helper. $\sum$ is the total time for the operator, **ZK** the time spent on verifying the ZK proof. H is the execution time of the helper.

$\lvert UH\rvert$	BK	BK'	OS	Upd
10	5.8	4.1	8.9	7.0
100	11.8	4.1	34.1	22.6
200	18.4	4.1	62.0	40.0
400	31.6	4.1	118.0	74.8
600	44.8	4.1	174.0	109.6
800	58.0	4.1	229.9	144.4
1000	71.2	4.1	285.9	170.2

Table 10.2.: Data exchanged in *kB* in relation to the size of the UH. BK denotes a BK task with non-trivial shift and permutation. BK' denotes BK with only an incremental update.

are set. Thus to provide a good lower bound, we performed a cyclic shift and then set a single entry.

We present in Table 10.1 the results using a server for operator and helper.

$\lvert UH\rvert$	U	User Registration				Bookkeeping				Outsource					Update			
		$\sum$	**On**	**ZK**	**Val**	$\sum$	**On**	**ZK**	**Val**	$\sum$	**PC**	**On**	**ZK**	**Val**	$\sum$	**On**	**ZK**	**Val**
10	U_1	154	97	78	57	616	556	494	60	551	189	298	193	64	528	461	400	67
	U_2	181	99	73	81	969	881	788	87	749	244	409	272	87	809	719	622	90
100	U_1	155	100	78	56	1559	1501	1432	58	1247	818	369	181	63	1541	1439	1319	107
	U_2	176	96	71	80	2200	2116	2033	84	1750	1180	477	240	89	2102	2008	1842	92
200	U_1	223	116	86	84	2725	2616	2542	87	2088	1540	499	204	92	2868	2723	2484	124
	U_2	175	96	70	79	3465	3382	3300	82	2959	2242	625	235	86	3353	3265	3011	90
400	U_1	241	122	91	118	5649	5530	5441	119	4309	3373	804	221	128	5633	5502	5015	130
	U_2	178	96	71	82	6302	6217	6136	85	5603	4508	986	249	88	6422	6331	5902	93
600	U_1	240	120	88	120	9226	9103	9003	121	7073	5773	1182	252	126	9210	9079	8406	129
	U_2	178	96	70	84	9695	9612	9519	84	8228	6801	1336	265	88	9978	9885	9269	92
800	U_1	242	121	90	122	11899	11775	11672	125	9386	7763	1460	253	129	12100	11966	11104	131
	U_2	186	99	74	87	13130	13046	12950	85	10778	9020	1656	265	90	13402	13311	12462	93
1000	U_1	245	121	90	123	14667	14539	14443	129	11440	9594	1708	256	131	14940	14807	13763	130
	U_2	189	100	74	92	16383	16290	16213	91	13447	11377	1987	269	91	16642	16538	15471	94

Table 10.3.: Execution time in *ms* for user. U_1 uses phone 1, U_2 uses phone 2, $\sum$ is the total user execution time, of which **On** is the online running time, **ZK** the creation of the ZK proofs, **Val** the time for validating the UH, and **PC** the precomputation time.

$\lvert UH \rvert$	Precise sigmoid				Approximate sigmoid			
	Strong LAN	LAN	WAN	MB	Strong LAN	LAN	WAN	MB
10	3.60	5.06	94.86	591.74	0.32	0.59	11.61	69.71
20	3.62	5.07	95.12	593.29	0.33	0.60	11.93	71.26
50	3.65	5.11	95.84	597.94	0.35	0.64	12.58	75.91
100	3.68	5.19	96.90	605.68	0.38	0.71	13.53	83.66
1000	4.28	6.47	116.10	745.10	0.97	1.87	32.82	223.07

Table 10.4.: Time (in seconds) and communication (in megabyte) for logistic regression with two parties. *Strong* uses c5.9xlarge instances, otherwise we use m4.large.

When only performing addition during BK task and no permutation or direct update of entries we need to communicate $\approx 4kB$ of data regardless of the number of entries in the UH. This task takes between $\approx 310ms$ and $\approx 440ms$ on Phone 1 and $\approx 460ms$ on Phone 2, while the operator needs to compute for only about $14ms$.

Overall, even for UH sizes where computation time on a smartphone exceeds $10s$, we require less than $300kB$ of communication. Thus, even when using mobile data, communication times will mostly depend on network latency and in general be relatively short.

Outsourced Analytics. We implemented logistic regression inference using MP-SPDZ by Keller [95], which allows for benchmarking across a range of security models and protocols. As the cleartext modulus of the used curve is not compatible with the implementation of homomorphic encryption in MP-SPDZ, we restrict ourselves to protocols based on oblivious transfer with malicious security. For this we use MASCOT by Keller, Orsini, and Scholl [96]. As MP-SPDZ already implements logistic regression, we only had to choose the number of features and whether to approximate the sigmoid function for faster computation. For the former, we ran inference for 10, 20, 50, 100, and 1.000 features, and for the latter benchmarked both the established sigmoid function and the three-piece approximation of Mohassel and Rindal [111]. The latter has been found to deliver good results while being much simpler to compute in the context of secure computation. This is because the restrictions to three linear pieces only requires two comparisons and oblivious selections instead of exponentiation and logarithm. To define the computation domain, we used the order of the 254-bit prime field Weierstrass curve. This allows for a smooth integration with our zero-knowledge proof.

We show our end-to-end timings when running on AWS c5.9xlarge and m4.large instances in Table 10.4. The LAN times refers to the collocated setting. We simulate a WAN setting by adding a $100ms$ roundtrip delay and a bandwidth restriction of $10MB/s$. We only use one thread and about $300MB$ RAM for malicious security.

At the time of writing, the spot price in US East was \$0.10 and \$1.53 for m4.large and c5.9xlarge, respectively. This results in a cost per computation ranging from \$0.000016 to \$0.0032.

10.3. Discussion

Our results show that, even on weak hardware compared to modern smartphones, for moderately sized UHs our protocol runs fast enough for a smooth user experience: A BK task—which will be executed most frequently—runs in less than 3s even including typical network latency for UHs with 100 entries. When BK can be performed without the need of permuting or setting UH entries, it runs in well under 1s and can even support much larger UH sizes. OS needs to transmit more data but we expect this task to be used less frequently, and a large part of the necessary computation (*i.e.* creating commitments to shares of one-time pads and the random shares for one of the two parties) is independent of the current UH state and can thus be done in the background in advance. This allows OS to have very short online running time even for large UH sizes. Upd on the other hand takes about the same time as BK. Thus, the main limiting factor for the size of the UH is the time needed for the BK task. This can be partially mitigated if the need of permuting/setting values arises only sparsely or if only parts of the UH are affected. Then the whole UH can be split into multiple parts where permuting/setting is done to some parts only and a simple additive update is performed on the remaining parts. Our evaluation also shows that Phone 2 was consistently slower than Phone 1 even though the stronger processor would suggest otherwise. We are not certain about the cause of this. Possible explanations are the introduction of new battery saving mechanisms in Android 9, or the fact that Phone 1 has the stock google version of Android whereas Phone 2 has vendor-specific modifications.

10.4. Performance of Fraud Detection

We evaluated our fraud detection application from Section 9.1 with a UH containing 100 elements, corresponding to a scenario where the last $T = 19$ transactions are taken into account for analysis (5 entries per transaction plus previous risk level, number of remaining transactions, transaction limit and balance).

We are no domain experts for fraud detection in mobile payments, and real-world parameters and implementation details for use cases like fraud detection are not easily obtainable. The simplified instantiations we benchmarked in this chapter should therefore be viewed as educated guess how real-world systems could be parameterized. Yet let us try to justify our parameter choices: On average, we have 0.3-0.8 credit card transactions per day per person in Europe [18]. So 20 transactions per logbook (UH-size of 100) would cover about a month, while 200 transactions per logbook (UH-size of 1000) would cover at least 8 months in average. Older transactions are also implicitly taken into account by making the new risk level depend on the old risk level (and the new transactions). The simple fraud detection mechanism we implemented is most likely simpler than mechanisms that are used in practice. One could of course extend the simple fraud detection we implemented with additional rules. Some ideas on how the simple fraud detection mechanism could be extended are: (1) Consider the location and time difference between transactions and deny the transaction if it is not physically possible to travel the distance in that amount of time. (2) If the transaction amount is a lot higher than the average transaction amount, the number of remaining transactions until the next calculation gets decreased by more than one. (3) Implement daily limits: a limit on the number of transactions that are allowed per day or a limit on the total transaction value that is allowed per day or both. It would be interesting to study how good our system models fraud detection systems used in practice, but that would be a research line of its own.

Registration only needs to initialize the default risk level, number of remaining transactions and transaction limit. As such it is independent of the UH size. Similarly, the top-up task does not require any special computation and thus the results for BK with only addition holds for this task (*i.e.*, $330ms - 450ms$ for Phone 1 and $\approx 490ms$ for Phone 2 combined user plus operator computation time excluding communication time and $4.1kB$ of data).

Fraud detection happens in two stages: A more complex machine learning based risk assessment is performed regularly, During each payment, the simple fraud detection based on the result of the risk assessment are checked in our implementation by the following two rules: (1) The transaction value is less or equal than the maximum allowed transaction amount, and (2) the number of payment transactions the user can perform before the complex fraud detection mechanism has to be run has not exceeded its limit. Additionally we verify that the current balance is sufficient for the transaction. To that end, we use bulletproof rangeproofs [39] to let the user prove that the simple rules were evaluated correctly. These additional proofs take $\approx 200ms$ on Phone 1 and $\approx 185ms$ on Phone 2, and $17ms$ for the operator, independent of the UH size. Thus, for 100 entries (corresponding to the last $T = 19$ transactions), this task takes a total of $1.85s$ for phone 1 and $2.5s$ for phone 2 on the user side and $80ms$ on the operator side (excluding the communication time to transmit $22kB$ of data) for 100 entries. Thus, when storing 19 previous transactions, a transaction can be performed in under $3s$ even when taking the communication time into account. Increasing the number of previous transactions increases the computation time on average by $\approx 1.5s$ for phone 1 and $\approx 1.6s$ for phone 2 and $40ms$ for the operator per 20 additional transactions.

For the outsourced risk calculation we used logistic regression on 100 features using approximate sigmoid calculation and active security for the complex fraud detection mechanism. Our results suggest that the whole outsourced risk calculation process can be completed in well under $5s$: $\approx 0.7s$ for outsourcing, $\approx 0.4s$ for the logistic regression and $\approx 1.3s$ for the update, plus communication time between the user and the operator or helper.

11. Conclusion

We investigated the feasibility of a framework that cannot only be used to efficiently model the leakage required for business models and legal regulations with MPC, but that can also be executed in the real world, on smartphones as user devices, regardless which other protocols are running in parallel. We consider this type of security, UC, to be a minimal requirement for such protocols that might be executed on a smartphone. While it might be feasible to assume that in server farms, one server handles the entire interaction of one party without being interrupted because the server was deliberately rented for this one purpose, it is unrealistic to assume that a modern-day smartphone executes an entire protocol without being interrupted even once.

We introduced our requirements for such a framework in Section 5.8 and modeled all of them into an ideal functionality in Chapter 6. We then proceeded to introduce a protocol in Chapter 7 which we claim to fulfill the ideal functionality. To support our claim, we prove security of the protocol in Chapter 8. In UC, proving security requires a *simulator* that interacts with the *environment* on the one side and exchanges and reports sent messages in such a way, that the environment does not realize that the messages do not originate from the honest parties, and with the *functionality* on the other side to provide inputs for corrupted parties (as those are controlled by the environment).

UC security is quite strong, and that usually comes at the cost of performance. However, it was shown by Fetzer et al. [70] that the protocols still are practically efficient; the section was cloned in Chapter 10 for convenience, even though the implementation does not count as contribution to this thesis.

So in conclusion, we provided a practically efficient yet still UC secure framework which enables efficient modeling of leakage for functions. In Fetzer et al. [70] (and copied in Chapter 9, though again, this chapter is not a part of this thesis) we show how instantiations of functions look like in this framework by introducing to applications for fraud detection for mobile payments and for a targeted advertisement system.

We believe that this solves one (of arguably many) reasons why MPC is rarely used in practice, despite its feasibility and the advantages with respect to data economy—which is important not only in a legal sense, but also to provide more trust for the user that collected data is not misused.

Part II.

An Instantiation of Everlasting Secure Commitments

12. Introduction

This part of the thesis is dedicated to commitments. In particular, we study the question on how *secure* commitments can become. In particular, we investigate commitments in the *quantum* setting where both parties have access to a quantum computer and can exchange quantum states. These are particularly interesting as quantum bit commitments are known to be sufficient to construct Quantum Oblivious Transfer (QOT) [57], which in term suffices to construct Secure Multi-Party Computation (MPC) [99, 90].

While commitments cannot be unconditionally secure for *both* parties (we elaborate on that in Chapter 14), modern-day quantum bit commitment protocols limit the power of one party—either the sender or the receiver—to hold only against *computationally bounded* adversaries: Common schemes are either unconditionally hiding and computationally binding [102], or unconditionally binding and computationally hiding [65, 4]. The impossibility results disable unconditional security for both parties, but the research question investigated in this part of the thesis is as follows:

Can we hope for unconditional security for one party and a security guarantee that is stronger than computational but weaker than unconditional security for the other party?

In particular, we investigate the setting of everlasting security in the quantum case. A positive answer to this research question was given by Alléaume et al. [9] for quantum key distribution using authenticated quantum states, which was later formalized and proven to be secure by Unruh [129]. Their construction uses signature cards (which could also be instantiated by a post-quantum secure public key infrastructure) to achieve this goal.

A signature card is a trusted device accessible to a party. Using the device, the party can sign arbitrary messages without access to the signing key. The corresponding verification key is available to all parties. Constructing secure commitments exploits the hardness of forging a signature and of extracting the secret key from the physical device. If the signature card uses an EUF-CMA secure signature scheme it is not efficiently possible to forge a signature.

Protocols that use signature cards exploit this fact to force a party to fix their values, which in the case of Unruh [129] forces the receiver to perform a measurement and thus to collapse the quantum state. Intuitively, parties have to *respond* to a message before being able to forge a signature. Thus they have to query the signature card, which only accepts classical inputs.

The major shortcomings of the protocol by Alléaume et al. [9] and Unruh [129] is that signature cards are hard to justify for real protocols, as it is infeasible to expect each party to have a new signature card for each individual commitment they perform. As such, we focus our research question to the setting where no additional hardware is needed and focus this part of the thesis on the following research question:

Can we hope for unconditional security for one party and everlasting security for the other party without relying on additional hardware?

12.1. Contribution

In this part of the thesis we investigate a compromise between unconditionally secure and computationally secure commitments by introducing a bit commitment scheme that provides unconditional security for one party and *everlasting* security for the second party. Everlasting security models the setting where hardness assumptions need to hold only during execution of the protocol.

We expand on the field of study for Quantum Decay [100] as a physical assumption and obtain a security model which provides protection against adversaries whose computational power is limited (*i.e.*, computationally bounded) during the protocol execution but may become very powerful (*i.e.*, unbounded) at a later point in time. Our study provides two interconnected results:

In our first contribution we instantiate a quantum bit commitment that provides unconditional security for the receiver (*binding*) and *everlasting* security for the committer (*hiding*). The construction is practically efficient under plausible assumptions, particularly that consider decoherence in a generic setting. We prove this commitment to be secure in the Quantum Random Oracle Model (QROM).

The second contribution provides a framework for a new way to attack quantum bit commitments. The findings show that our security of the above construction breaks when instantiating the commitment in the QROM with any unconditionally hiding and computationally binding commitment scheme.

This part of the thesis is organized as follows: First, we motivate and formally introduce a new adversarial model based on a physical assumption we refer to as *Quantum Decay*. It is based on the observation that quantum storage is subject to decay and is rigorously motivated in Chapter 15. Informally, the assumption states that any quantum information (*e.g.*, stored during the execution of a protocol) decays to classical information after polynomially many time[1] steps. We review the assumption in light of everlasting security, *i.e.*, where in the first stage an adversarial receiver with Quantum Polynomial time (QPT)

[1] We use the terms *time steps* and *computational steps* synonymously as after polynomially many time steps it is not possible to perform super-polynomially many computational steps and vice versa, polynomially many operations on a Turing machine can be executed in polynomially many time steps.

runtime interacts with an honest sender during the commitment phase. Then, after a polynomial number of time steps, a (classically) unbounded adversary gets access to the view of the QPT-adversary as well as any classical information that the QPT-adversary may extract from the quantum state. To evaluate the security of our scheme the decaying process may be formally modeled as a measurement performed by the bounded adversary. However, we note that a measurement is not formally required.

Quantum Bit Commitment. We present an unconditionally binding and everlasting hiding quantum bit commitment protocol which extends a quantum bit commitment protocol by Brassard et al. [36] by a classical component.

In the two-stage protocol of Brassard et al. [36] the sender commits to a classical bit by fixing two classical $\mathscr{N}$-bit strings such that the to committed value is the inner product of these two strings. One of the strings is then encoded into a quantum state vector by randomly picking from a basis set and then preparing the respective quantum state. The second string is sent as classical bitstring directly alongside the quantum state to the receiver. The (honest) receiver measures the quantum state upon receiving in random bases. To unveil a bit, the sender uncovers the bit as well as the encoding bases, allowing the receiver to verify the commitment. In order to make it harder to cheat the authors require that the $\mathscr{N}$-bit string that is encoded into the quantum state is chosen as a $(\mathscr{N}, \mathscr{K}, \mathscr{D})$-code $\mathscr{C}$.

Mayers [107] showed that the commitment scheme above is unconditionally hiding, but at most *computationally* binding. To achieve everlasting security we extend the protocol by adding a (QROM-based) classical commitment to encode the bases of the sender. The classical commitment is unconditionally binding but only computationally hiding.

The details of our protocol, denoted Π_{QCom}, are presented in Chapter 16. We provide a complete security proof in Chapter 18.

Security Proof. We deploy a classical commitment protocol in Chapter 17 that uses a Quantum Random Oracle (QRO) and prove its unconditional binding property based on the collision resistance of the QRO. We prove a new and simple lemma that limits the advantage of a quantum adversary trying to determine after t queries whether the given value table is that from the oracle or if it differs on k inputs. Then, in Chapter 18, we prove that when assuming Quantum Decay and the classical commitment from Chapter 17, our quantum bit commitment is everlasting hiding and unconditionally binding. The everlasting hiding property is shown to hold based on the independence of *optimal* measurement basis chosen by the adversary from the actual basis used to encode the quantum information. This property is based on the difficulty of the adversary to extract a sufficient amount of information from the classical commitment before the Quantum Decay renders the quantum state unusable. The unconditional binding property follows from statistical arguments. This proves our first main contribution, Theorem 12.1.1.

Theorem 12.1.1 (Informal)**.** *In the Quantum Random Oracle Model and assuming that Quantum Decay holds, the quantum bit commitment protocol Π_{QCom} is unconditionally binding and everlasting hiding.*

Obfuscated Measurement Attack. We demonstrate that Quantum Decay may not be sufficient to achieve everlasting security when instantiating the Quantum Random Oracle based commitment with any unconditionally binding and computationally hiding commitment scheme. To prove our claim we present a novel technique to attack the everlasting hiding property of quantum bit commitments. To that end we introduce a new type of attack against the everlasting hiding property of quantum commitments in Chapter 19.

Intuitively, we replace the QRO-based commitment scheme with a commitment scheme that allows a certain type of malleability: the adversary can perform homomorphic operations on the hidden value and the quantum state. This allows the bounded adversary to change the encoding of the quantum state in such a way that the bounded adversary itself actually obtains *less* information from the corresponding measurement, but such that the encoding can be *classically* undone given the measurement outcome and the additional information computed by the bounded quantum adversary. Thus an unbounded adversary can later break the classical commitment and use the extracted information to breach the everlasting hiding property of the quantum commitment. Due to the involved hiding measurement refer to this approach as Obfuscated Measurement Attack (OMA).

This does not prove that Quantum Decay in insufficient to provide everlasting secure commitments. However, it suggests that additional assumptions may be required to achieve this security notion. We summarize this in Theorem 12.1.2 as our second main contribution:

Theorem 12.1.2 (Informal)**.** *The classical commitment scheme from the protocol Π_{QCom} needs to have additional properties beyond unconditionally binding and computationally hiding for Π_{QCom} to be everlasting hiding in the setting of Quantum Decay.*

12.2. Related Work

A prominent first example on how to circumvent the impossibility result of Mayers [108] was provided by Kent [97]: He proved that perfectly secure protocols can exist if we assume relativistic assumptions such as the impossibility of faster-than-light communication. However, the resulting protocol is highly impractical. It requires the participants to be apart by an astronomical distance and even completely fails in case of transmission errors.

A different approach was taken by Koenig, Wehner, and Wullschleger [100] who also initiated the study on using Quantum Decay to construct secure commitment protocols. The authors exploit this in their design of a (highly theoretical) quantum commitment protocol in a different way than we do; for each protocol message, the protocol by Koenig, Wehner, and Wullschleger [100] has hard-coded *delays* in the protocol execution to ensure a certain decoherence of the quantum state. They model decoherence as continuous *quantum noise*. This models a *gradual* decay of the quantum state. While the underlying assumption is the same, we model the decay differently. Essentially, we only require the state to be completely decayed within a polynomial number of time steps, whereas they require the quantum computer to fulfill a (worst case) quantum noise function where the quantum state decays with each additional timestep. In their protocol the authors use this by requiring both parties to delay their communication during the commit phase to a point in time where the quantum state is sufficiently collapsed.

In particular, their model considers an adversarial quantum storage which is both bounded and noisy. At any time t the quantum state either has a minimal trace distance to the original state that depends on a noise function, or the quantum storage has already been measured.

As such we believe that our assumption building on decay after a polynomial number of computational steps to be more natural than the work in [100]. Our modeling enables protocols that do not require delays during execution. Instead, our assumption only requires that the adversary is computationally bounded for some time after initiating the protocol. We briefly recall their idea of quantum decay as well as their scheme, and argue both that our setup differs, and that our assumptions are orthogonal to theirs.

While this comparison was not made by the author of this thesis but by a co-author working on the same paper, we consider it highly relevant to distance ourselves from the work of Koenig, Wehner, and Wullschleger [100] and hence take the comparison into our thesis.

The authors model the decay as a function that is applied to all quantum states after every step, resulting in a loss of certain quantum information. If the function were the identity then quantum information could be stored indefinitely, hence the scheme would be entirely quantum and the unconditional security would not hold due to Mayers [108]. For unconditional security to hold, the authors show that the function must eventually eliminate all quantum information, thus making it impossible for the sender to modify the shared state. The advantage of the adversary is then quantified over all (POV) measurements as the probability to extract information from the quantum state about a classical value (*i.e.*, the value committed to). In contrast, we model decay as the complete collapse of quantum information rather than a function that is instantiated to achieve different levels of security. We do not need to specify the point in time where the decay happens.

Their commitment protocol consists of two parts. In the first part [100, Protocol 1] the sender encodes a random string using a set of random bases and sends the resulting quantum state to the receiver. The receiver measures the state in random bases. Both parties then *wait* for a time δt to ensure that the quantum state has decayed sufficiently to not be susceptible to quantum computations. After the delay, the decay function is applied to the quantum state of both parties. In the second part [100, Protocol 2a], the sender encodes the random string from the first part using a linear code and sends the hash of the encoding and a random vector to the receiver.

Instead, we only have to assume that decay occurs after polynomially many computational steps. Therefore, our protocol is independent of the current technological level that dictates the time it takes the quantum state to decay and does not require any delays during runtime. In particular, we only require that the adversary is unable to perform a superpolynomial (in the security parameter) number of queries to the QROM before the Quantum Decay occurs. If the decay never occurs, our protocol still remains computationally secure.

We consider the following toy example: Assume that quantum states can be kept coherent for at most three weeks. Then the protocol of Koenig, Wehner, and Wullschleger [100] requires to halt the interaction for three weeks to cause decay, during which the commit-phase is still unfinished. Only after the delay, when the sender executes the second part of the commitment, unconditional security for the committed bit is achieved.

In our case, the protocol execution only takes a relatively small amount of time (in the order of seconds or at worst minutes). The delay time only comes into play if the receiver tries to cheat: Then, unless the receiver can perform super-polynomially many computational steps within those three weeks, the committed bit remains hidden forever (even against unbounded adversaries).

13. Preliminaries

13.1. Notation

To discriminate quantum states from classical values we use the canonical *Dirac notation*, *e.g.* the state vector $|\psi\rangle$ and its dual $\langle\psi|$.

Throughout this part of the thesis we use two adversaries $\mathcal{B}$ and $\mathcal{A}$ and always denote by $\mathcal{B}$ the QPT adversary and by $\mathcal{A}$ the unbounded adversary. Furthermore, $\mathcal{A}$ has access to the final state st of $\mathcal{B}$, which contains the view, *i.e.* the complete transcript of the interactions of $\mathcal{B}$ as well as the random tape which we denote by $T_{\mathcal{B}}$.

For an adversary $\mathcal{B}$ and some oracle O, $\mathcal{B}^O$ denotes that $\mathcal{B}$ is given black-box access to the oracle O. We refer to $\mathcal{B}^O$ as *oracle algorithm*. The definition for $\mathcal{A}$ is analogous.

In this part of this thesis we will use codes. Anything related to codes will be written in script typestyle, such as $\mathscr{C}$ for the code, $\mathscr{c}$ for a codeword or $\mathscr{N}$ for the codeword length.

We write quantum gates in sans serif font, *e.g.* H for the Hadamard-gate.

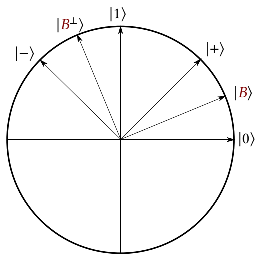
The hamming distance between two n-bitstrings x and y is denoted by $\mathrm{d}_{\mathrm{Ham}}(x) := \sum_{i=1}^{n} [\![x[i] \neq y[i]]\!]$.

13.2. Quantum Information

Each qubit is represented as an element in a two-dimensional Hilbert space H^2. The inner product is written as $\langle\psi, \psi\rangle$. If not otherwise stated, the state is represented in the computational basis, which we denote as $+$, consisting of the orthogonal vectors $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$ for a two dimensional Hilbert space. Additionally we use the Wiesner (or *diagonal*) basis $|+\rangle = \frac{1}{\sqrt{2}}\binom{1}{1}$ and $|-\rangle = \frac{1}{\sqrt{2}}\binom{1}{-1}$, which we denote as $\times$, to encode quantum states.

The set of possible single-qubit quantum states can be represented by a unit circle in two-dimensional Hilbert space: The geometrical representation on a unit circle, where the vector pointing to the top corresponds to the value $|0\rangle$ and the vector pointing to the right to the value $|1\rangle$. Then a state can be represented as $|\phi\rangle = \cos(\psi)|0\rangle + \sin(\psi)|1\rangle$.

Each point on the above circle represents a valid quantum state. The circle additionally contains descriptions of the computational bases $+ = \{|0\rangle, |1\rangle\}$, the diagonal or Wiesner



basis $\times = \{|+\rangle, |-\rangle\}$ which were first introduced by Wiesner [136] and the Breidbard bases $B = \{|B\rangle, |B\rangle^{\perp}\}$.

Formally, they are given as follows:

Definition 13.2.1 (Computational Basis). $+ = \{|0\rangle, |1\rangle\}$

Definition 13.2.2 (Wiesner or Diagonal Basis). $\times = \{|+\rangle, |-\rangle\}$

$$\begin{aligned} |+\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |-\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{13.1}$$

Definition 13.2.3 (Breidbard Basis).

$$\begin{aligned} |B\rangle &= \quad \cos\left(\frac{\pi}{8}\right)|0\rangle + \sin\left(\frac{\pi}{8}\right)|1\rangle = \frac{1}{2}|0\rangle + \frac{1}{2}|+\rangle, \\ |B^{\perp}\rangle &= -\sin\left(\frac{\pi}{8}\right)|0\rangle + \cos\left(\frac{\pi}{8}\right)|1\rangle = \frac{1}{2}|1\rangle + \frac{1}{2}|-\rangle \end{aligned} \tag{13.2}$$

An equivalent description of a quantum state is given by the density operator or matrix $\rho \equiv \sum_i p_i |\psi_i\rangle\langle\psi_i|$, which omits the information about a particular basis. It may be noted that two states that admit the same density matrix cannot be distinguished by any measurement.

By $\vec{b}$, we denote a string of the above bases. We denote the measurement operator with respect to the basis $\vec{b}$ as $M_{\vec{b}}$. A quantum state $|\psi\rangle^P$ denotes that $|\psi\rangle$ is in possession of or associated with a party P.

The distance of two states can be measured using the fidelity function F, *i.e.*, for two quantum states represented by the density matrices ρ_1, ρ_2, the fidelity is defined as follows:

$$\mathrm{F}(\rho_1, \rho_2) = \mathrm{tr}\left((\sqrt{\rho_1}\rho_2\sqrt{\rho_1})^{\frac{1}{2}}\right) \tag{13.3}$$

The trace of a square matrix M is defined as $\mathrm{tr}(M) = \sum_i M_{ii}$.

It can be shown that for any two density matrices ρ_0 and ρ_1, the fidelity is $0 \leq \mathrm{F}(\rho_1, \rho_2) \leq 1$. The fidelity increases if the states become less distinguishable. Intuitively, the fidelity $\mathrm{F}(\rho_1, \rho_2)$ of two quantum states represented by their respective density matrices ρ_1 and

ρ_2 describes the probability that the ρ_1 is falsely identified as ρ_2 upon measurement. For example, the fidelity of two identical states is one, $\mathrm{F}(\rho_1, \rho_1) = 1$ and of two orthogonal states is zero, $\mathrm{F}(\rho, \rho^\perp) = 0$. Additional details and proofs can be found in Nielsen and Chuang [114, Section 9.2.2].

A quantum system is the physical entity which is described by a quantum state. If the state is exactly known, then it is considered pure and described by a single state vector $|\psi\rangle$. Otherwise the system is fully described by an ensemble of (pure) states $\{p_i, |\psi_i\rangle\}$, where p_i is the probability to be in state $|\psi_i\rangle$.

Throughout this part of the thesis we work with both quantum algorithms described by functions and circuits using gates and wires: The notation for single and multi-qubit gates follows the standard notions, which can be found in Nielsen and Chuang [114, Sec. 1.3].

For *controlled* gates we use the notation $\mathrm{H}^{(\theta)}$ for an application of a *controlled* gate (in this case Hadamard H; the definition for other gates is equivalent), where H is only applied if θ is 1. As such we denote the Toffoli gate (*Controlled Controlled Not*) by $\mathrm{X}^{(\theta_1,\theta_2)}$. If the index is a classical integer x, $\mathrm{X}^{(x)}$ denotes applying x iterations of the quantum gate (here X) successively. In particular, for a bit $b \in \{0, 1\}$, $\mathrm{X}^{(b)}$ denotes applying the gate only if $b = 1$.

Finally, throughout this paper, we refer to protocols/primitives/parties that are not considered over quantum state or do not have quantum computational power as *classical*.

13.2.1. Security Notion

We consider the notion of unconditional (or information-theoretic) security where a scheme is unconditionally secure if an (even unbounded) adversary $\mathcal{A}$ attacking the scheme has only negligible success probability. When talking about security notions, we use the terms "unconditionally" and "information-theoretic" interchangeably. A scheme is computationally secure if an adversary's success probability for breaking the security guarantees is negligible in the security parameter κ for any $\mathrm{poly}(\kappa)$ bounded adversarial (quantum) algorithm.

Definition 13.2.4 (Code)**.** *Let $o \in \mathbb{N}$ be the order of a group. Let $\mathbb{F}_o^{\mathcal{N}}$ be a vector space. A subspace $\mathcal{C} \subseteq \mathbb{F}_o^{\mathcal{N}}$ is called a $[\mathcal{N}, \mathcal{K}, \mathcal{D}]$-code if its codewords form a $\mathcal{K}$-dimensional vector subspace of $\mathbb{F}_o^{\mathcal{N}}$.*

We additionally require closedness from a linear code, which requires that any linear combination of codewords is a codeword itself.

Definition 13.2.5 (Closeness)**.** *Let $\mathcal{C}$ be a $[\mathcal{N}, \mathcal{K}, \mathcal{D}]$-code. $\mathcal{C}$ is* closed *if for each $c_1, c_2 \in \mathcal{C}$ we have $c_1 + c_2 \in \mathcal{C}$.*

Definition 13.2.6 (Linear Code)**.** *Let $\mathscr{C}$ be a $[\mathscr{N}, \mathscr{K}, \mathscr{D}]$-code. $\mathscr{C}$ is a linear code if $\vec{0}^{\mathscr{N}} \in \mathscr{C}$ and if $\mathscr{C}$ is closed.*

The parameter $\mathscr{D}$ denotes the minimal distance, *i.e.* the minimal number of positions of any two distinct codewords $c_1, c_2 \in \mathscr{C}$ that differ:

$$\mathscr{D} := \min\{\#\{i \mid c_1[i] \neq c_2[i]\} \quad | \quad c_1, c_2 \in \mathscr{C}; c_1 \neq c_2\} \tag{13.4}$$

If t denotes the number of errors the code can correct, then $\mathscr{D}$ is at least $2t + 1$. We define a linear code by its *generator matrix* $\vec{G} \in \mathbb{F}_o^{\mathscr{N} \times \mathscr{K}}$. The generator matrix spans the code such that each codeword $c \in \mathscr{C}$ can be represented as $\vec{m}\vec{G}$ for a vector $\vec{m} \in \mathbb{F}_o^{\mathscr{K}}$.

13.2.2. Conjugate Coding

We describe the basic idea behind conjugate coding [136] (or Wiesner encoding) to encode classical information into a quantum state $|\psi\rangle$. It uses a function $\mathsf{encode}_{\vec{b}} \colon \{0,1\}^n \rightarrow H^n$. The function uniformly samples a string $\vec{b} \in \{+, \times\}^n$ of basis vectors for each qubit and encodes the input $\vec{x} \in \{0,1\}^n$ to a quantum string, *i.e.* for each bit position i it performs the following operation:

If $\vec{b}[i] = +$ it encodes 0 as $|0\rangle$ and 1 as $|1\rangle$, and if $\vec{b}[i] = \times$ it encodes 0 as $|+\rangle$ and 1 as $|-\rangle$.

$$|b_i\rangle_{\vec{b}[i]} = \begin{cases} |0\rangle & \text{if } \vec{b}[i] = +, b = 0 \\ |1\rangle & \text{if } \vec{b}[i] = +, b = 1 \\ |+\rangle & \text{if } \vec{b}[i] = \times, b = 0 \\ |-\rangle & \text{if } \vec{b}[i] = \times, b = 1 \end{cases}. \tag{13.5}$$

Codes and Quantum Computers. Linear codes in the quantum setting were thoroughly investigated by Brassard et al. [36]. In this section we recall some of their results in this setting. We stress that while their main theorem—the security of their proposed commitment protocol—was disproven by Mayers [107] the discovered flaw was due to their definition of the binding property and unrelated to the proven theorems regarding codewords.

All the following results are with respect to linear $[\mathscr{N}, \mathscr{K}, \mathscr{D}]$ codes with $\mathscr{N} \in \mathsf{poly}(\kappa)$ and $\frac{\mathscr{K}}{\mathscr{N}} = 0.52$, with $\mathscr{D} > \gamma\mathscr{N}$ for $\gamma = \mathrm{H}^{-1}(1/2)$ for H being the *entropy* and with $\frac{\mathscr{D}}{\mathscr{N}} > 10\xi$ for some $0.01 > \xi > 0$ being the noise of the quantum channel.

Lemma 13.2.7 (Brassard et al. [36, Thm. 3.5])**.** *Let $\mathsf{Gen} \in \mathbb{F}^{\mathscr{N} \times \mathscr{K}}$ be a randomly chosen generator matrix defining an $[\mathscr{N}, \mathscr{K}, \mathscr{D}]$ code. With overwhelming probability, the number of codewords (c, c') with Hamming distance $\mathrm{d}_{\mathrm{Ham}}(c', c) = \mathscr{D}$ is greater than $\frac{2^{(\mathscr{K} - \frac{\mathscr{N}}{2})}}{\sqrt{\mathscr{N}}}$.*

As we assume $\frac{K}{N} > \frac{1}{2}$, this lemma states for the codes considered here that each codeword has an exponentially large amount of direct neighbors except with negligible probability $2^{-\alpha\mathcal{N}}$ for some $\alpha > 0$.

Lemma 13.2.8 (Brassard et al. [36, Thm. 3.6] and Bennett, Brassard, and Robert [24, Thm. 8])**.** *Let E denote the set of all possible codewords with distance $\leq \mathcal{D}$ around the c' which was measured by the receiver, and let $|E| \geq \frac{2^{\left(\mathcal{K}-\frac{\mathcal{N}}{2}\right)}}{\sqrt{\mathcal{N}}}$. Let $\mathcal{A}$ be an adversary with access to all $|E|$ codewords. The amount of information $\mathcal{A}$ has about the parity $\langle c', otp\rangle$ is negligible in κ.*

Wiesner encoded codewords. The next lemmas are more specific. In particular, they limit the success probability of an adversary trying to reconstruct a codeword that was encoded into a quantum state using Wiesner's encoding technique. If the adversary has to reconstruct the codeword without knowing the used bases $\vec{b}$ then Brassard et al. [36] show that the information an adversary can extract from the quantum state is limited. The first lemma states that the optimal strategy for any adversary is to measure in *Breidbard-bases*.

Lemma 13.2.9 (Brassard et al. [36, Thm. 3.1])**.** *Let $\vec{b}$ be a string of bases chosen uniformly at random. Let $|\psi\rangle$ be a quantum state encoding any information c using Wiesner's encoding with bases $\vec{b}$. Let $\mathcal{B}$ be the receiver of $|\psi\rangle$ without knowledge of $\vec{b}$. The most information $\mathcal{B}$ can extract from $|\psi\rangle$ about c is by measurement in Breidbard-bases B.*

The next lemma then limits the amount of information that the adversary can extract using the optimal measurement strategy.

Lemma 13.2.10 (Brassard et al. [36, Lemma 3.2])**.** *Let $|\psi\rangle$ be a quantum state that encodes a word w of length $\mathcal{N}$ using Wiesner encoding bases $\vec{b}$. Let $\mathcal{B}$ be an adversary without access to $\vec{b}$, trying to extract the maximal number of bits of the word w. With overwhelming probability, the number of bits $\mathcal{B}$ can extract is less than $0.89\mathcal{N}$.*

This means that with overwhelming probability, every measurement result of $\mathcal{B}$ has a hamming-distance to the original codeword of at least $0.1100279\mathcal{N}$.

This enables limiting the information on the original codeword:

Lemma 13.2.11 (Brassard et al. [36, Thm. 3.4])**.** *Let $c \leftarrow \mathscr{C}$ be a codeword. Let c' be a codeword with distance to c at least $\gamma\mathcal{N}$. Let $h := \mathrm{d}_{\mathrm{Ham}}(c, c')$. Let $\mathcal{A}$ be an adversary with access to h and c' trying to guess c. Then the probability of $\mathcal{A}$ guessing the correct codeword c is negligible in the security parameter.*

13.2.3. Quantum Bit Commitment

The definitions for quantum bit commitments closely follow the definitions provided in Section 2.2 but with the minor change that all parties have access to quantum computers and can exchange quantum information, but the commitment is still on a *classical* bit. This means that for computational definitions of binding and hiding, the adversary is now in QPT instead of PPT. Furthermore, the commitment message itself can consist of quantum states.

While the definition from Definition 2.2.5 is just as for the classical case[1], we provide formal definition for the quantum-computational hiding property.

Definition 13.2.12 (Quantum-Computationally Hiding Bit Commitment Scheme)**.** *A bit commitment scheme* $\textsc{Com} = (\mathsf{Com}, \mathsf{Unv})$ *is* quantum-computationally hiding *if for any QPT adversary* $\mathcal{B}$ *and any security parameter* $\kappa \in \mathbb{N}$*, the following probability is negligible in* κ*:*

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\}, \\ \tau_b \leftarrow \textsc{Com}.\mathsf{Com}(1^\kappa, b), \\ b' \leftarrow \mathcal{B}(1^\kappa, \tau_b) \end{array} \right] - 1/2 \right| \tag{13.6}$$

where τ_b *is the* transcript *of an honest execution for a commitment on* b*. In this case this corresponds to the entire view of a receiver during the commitment-phase.*

13.2.4. Fully Homomorphic Encryption

The idea of Fully Homomorphic Encryption (FHE) was first put forth by Rivest, Adleman, and Dertouzos [118] and first instantiated more than 30 years later by Gentry [73]. The key idea is to provide an encryption scheme that allows to *transform* ciphertexts into valid new ciphertexts that fulfill a given relation R on the respective plaintext, without decrypting the ciphertext. This allows for numerous applications as Secure Function Evaluation can trivially be achieved by letting the client encrypt the inputs and the server executing the function on the ciphertext.

Definition 13.2.13 (Fully Homomorphic Encryption (taken from the full version of [33, 34, Section 3.1]))**.** *A (public-key) Fully Homomorphic Encryption (FHE) scheme* $\textsc{Fhe} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is a quadruple of PPT algorithms as follows.*

Key Generation. *The algorithm* $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$ *takes a unary representation of the security parameter and outputs a public encryption key* pk*, a public evaluation key* ek *and a secret decryption key* sk*.*

[1] We deliberately chose this definition as opposed to the classical definition of an adversary having to provide unveil information for *both* bits under a fixed commitment com as this definition does not hold in the quantum setting [107].

Encryption. *The algorithm* $ct \leftarrow \textsc{Fhe}.\mathrm{Enc}_{\mathsf{pk}}(b)$ *takes the public key* pk *and a single bit message* $b \in \{0,1\}$ *and outputs a ciphertext* ct.

Decryption. *The algorithm* $b^* \leftarrow \textsc{Fhe}.\mathrm{Dec}_{\mathsf{sk}}(ct)$ *takes the secret key* sk *and a ciphertext* ct *and outputs a message* $b^* \in \{0,1\}$.

Homomorphic Evaluation. *The algorithm* $ct_{\mathsf{f}} \leftarrow \textsc{Fhe}.\mathrm{Eval}_{\mathsf{ek}}(\mathsf{f}, ct_1, \dots, ct_n)$ *is parameterized by the evaluation key* ek, *a function* $\mathsf{f}\colon \{0,1\}^n \to \{0,1\}$ *and a set of* n *ciphertexts* $ct_1, \dots, ct_n$ *and outputs a ciphertext* ct_{f}.

A special case of FHE is called *Leveled* FHE, which is correct for all circuits up to a given level (depth). By a standard transformation, assuming circular security[2], any leveled FHE scheme can be transformed into a FHE scheme for arbitrary functions f [73].

Note also that modern schemes such as [74] work without an evaluation key. Hence we ignore the evaluation key throughout this part of this thesis as we only work with such FHE schemes that do not require them.

13.2.5. q-IND-CPA

For classical FHE schemes we require a definition of what it means for them to be secure in the presence of quantum adversaries. We thus use the definition of Quantum Indistinguishability under Chosen Plaintext Attacks from [38, Definition 1]:

Definition 13.2.14 (Quantum Indistinguishability under Chosen Plaintext Attacks). *Let* $\textsc{Fhe} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$ *be a classical Fully Homomorphic Encryption scheme.* $\textsc{Fhe}$ *fulfills Quantum Indistinguishability under Chosen Plaintext Attacks (q-IND-CPA) security if for any QPT adversary* $\mathcal{B}$, *there exists a negligible function* $\mathrm{negl}(\kappa)$ *such that for all* $\kappa \in \mathbb{N}$ *and* $(\mathsf{pk}, \mathsf{ek}, \mathsf{sk}) \leftarrow \textsc{Fhe}.\mathrm{KeyGen}(1^\kappa)$, *it holds that:*

$$\begin{aligned} &|\Pr[\mathcal{B}(\mathsf{pk}, \mathsf{ek}, \textsc{Fhe}.\mathrm{Enc}(\mathsf{pk}, 0)) = 1] \\ &- \Pr[\mathcal{B}(\mathsf{pk}, \mathsf{ek}, \textsc{Fhe}.\mathrm{Enc}(\mathsf{pk}, 1)) = 1]| \le \mathrm{negl}(\kappa), \end{aligned} \tag{13.7}$$

where the probability is taken over the random coins of the adversary and the coins used for generation of pk, ek *and* sk.

That is, we require that no QPT adversary $\mathcal{B}$ has a non-negligible advantage to distinguish encryptions of 0 from encryptions of 1, even when given the evaluation key ek and the public key pk.

[2] The assumption states that encryptions of the secret key under the public key do not violate the security guarantees.

13.2.6. Quantum Fully Homomorphic Encryption

Quantum Fully Homomorphic Encryption (QFHE) follows the formula of classical FHE, but enables computations on an encrypted *quantum* state. The encryption of the state is information-theoretically and uses a Quantum One Time Pad (QOTP), but the keys (x, z) are only computationally hidden using a classical scheme.

Definition 13.2.15 (Quantum Fully Homomorphic Encryption (taken verbatim from [38, Def 2]))**.** *A (symmetric) Quantum Fully Homomorphic Encryption (QFHE) scheme is a 4-tuple of quantum algorithms* $\mathit{QFHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$*:*

Key Generation $\mathsf{KeyGen}\colon 1^\kappa \to (\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$. *This algorithm takes a unary representation of the security parameter as input and outputs a classical public encryption key* pk*, a classical secret decryption key* sk *and a quantum evaluation key* $\mathsf{ek} \in \mathsf{cod}(\mathit{ES})$.

Encryption $\mathsf{Enc}_{\mathsf{pk}}\colon \mathsf{cod}(\mathit{MS}) \to \mathsf{cod}(\mathit{CS})$. *For every possible* pk*, the quantum channel* $\mathsf{Enc}_{\mathsf{pk}}$ *maps a state in the message space* MS *to a state (the* cipher state*) in the cipherspace* CS.

Homomorphic Evaluation $\mathsf{Eval}^C\colon \mathsf{cod}(\mathit{ES} \times \mathit{CS}^{\otimes n}) \to \mathit{CS}'^{\otimes n}$. *For every quantum circuit* C*, with induced channel* $\varphi_C\colon \mathsf{cod}(\mathit{MS}^{\otimes n}) \to \mathsf{cod}(\mathit{MS}^{\otimes m})$*, we define a channel* Eval^C *that maps an* n*-fold cipher state to an* m*-fold cipher state, consuming the evaluation key in the process.*

Decryption $\mathsf{Dec}_{\mathsf{sk}}\colon \mathsf{cod}(\mathit{CS}') \to \mathsf{cod}(\mathit{MS})$. *For every possible* sk*,* $\mathsf{Dec}_{\mathsf{sk}}$ *is a quantum channel that maps the state in* $\mathsf{cod}\mathit{CS}'$ *to a quantum state in* $\mathsf{cod}(\mathit{MS})$.

The definition for *symmetric* QFHE schemes is analogous only with $\mathsf{pk} = \mathsf{sk}$. Note that later schemes [32, 106] do not rely on an evaluation key ek for quantum operations. As we only work with these schemes we omit ek throughout this part of the thesis.

Most of the common schemes follow the same basic principle: The evaluator is given the QOTP-secured quantum state alongside *classical* Fully Homomorphic Encryption ciphertexts of the respective keys x and z. Whenever the server performs an operation on the quantum state, it *homomorphically adjusts* the QOTP keys accordingly.

Common QFHE schemes such as [38, 8, 106, 32] follow the prepare-and-measure principle, where the client only has to be capable of performing (alongside classical PPT operations) relatively simple tasks such as encoding a single qubit and later measuring it.

For our work, we require a QFHE Scheme which has a special property, which is fulfilled *e.g.* by the scheme of Brakerski [32]. We call schemes that fulfill these properties Quantum Fully Homomorphic Encryption Tuples:

Definition 13.2.16 (Quantum Fully Homomorphic Encryption Tuple)**.** *Let* FHE *be a classical Fully Homomorphic Encryption scheme. Let* QFHE *be a secure symmetric Quantum Fully Homomorphic Encryption-scheme. The tuple* $(\mathit{QFHE}, \mathit{FHE})$ *is called Quantum Fully Homomorphic Encryption Tuple (QFHET) iff* FHE *is q-IND-CPA secure and for all* n*-qubit* *quantum states $|\psi\rangle$, there are QPT algorithms where the two following probabilities are both overwhelming in the security parameter κ:*

$$\Pr\left[\,|\psi\rangle' = (\mathrm{H}|\psi\rangle) \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathcal{QFHE}.\mathrm{KeyGen}(1^\kappa), \\ |\phi\rangle := \mathcal{QFHE}.\mathrm{Enc}_{\mathsf{pk}}(|\psi\rangle), \\ |\phi\rangle' := \mathcal{QFHE}.\mathrm{Eval}_{\mathsf{ek}}(|\phi\rangle, \mathrm{H}), \\ |\psi\rangle' := \mathcal{QFHE}.\mathrm{Dec}_{\mathsf{sk}}(|\phi\rangle') \end{array} \right] \tag{13.8}$$

$$\Pr\left[\,|\psi\rangle' = \left(\mathrm{X}^{(cb_1,cb_2)}|\psi\rangle\right) \;\middle|\; \begin{array}{l} (\mathsf{pk}^*, \mathsf{ek}^*, \mathsf{sk}^*) \leftarrow \mathcal{FHE}.\mathrm{KeyGen}(1^\kappa), \\ (\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathcal{QFHE}.\mathrm{KeyGen}(1^\kappa), \\ |\phi\rangle := \mathcal{QFHE}.\mathrm{Enc}_{\mathsf{pk}}(|\psi\rangle), \\ (cb_1, cb_2) \xleftarrow{\$} \{0,1\}^2, \\ cct_1 := \mathcal{FHE}.\mathrm{Enc}_{\mathsf{pk}^*}(cb_1), \\ cct_2 := \mathcal{FHE}.\mathrm{Enc}_{\mathsf{pk}^*}(cb_2), \\ |\phi\rangle' := \mathcal{QFHE}.\mathrm{Eval}_{\mathsf{ek}}\left(|\phi\rangle, \mathrm{X}^{(cct_1,cct_2)}\right), \\ |\psi\rangle' := \mathcal{QFHE}.\mathrm{Dec}_{\mathsf{sk}}(|\phi\rangle') \end{array} \right] \tag{13.9}$$

That is, a QFHET has to be capable of *homomorphically* performing *Hadamard* gates, as well as *Toffoli* gates over *classically* encrypted control bits. While the former has been possible ever since the seminal paper of Broadbent and Jeffery [38], the latter has only been possible since the work of Brakerski [32].

13.3. Security Model

13.3.1. Quantum Random Oracle Model

The (classical) Random Oracle Model (ROM) was designed by Bellare and Rogaway [22] as a way to proof security of efficient, classical protocols. It models an *ideal hash function* R_H which returns *perfectly random*, but for each session *deterministic* bitstrings of length κ. So an adversary $\mathcal{B}$ has only *negligible advantage* to guess $R_H(x) \in \{0,1\}^\kappa$ given a (previously untested) input $x \in \{0,1\}^*$. However, during the same session, the outcome $R_H(x)$ will always remain the same.

The QROM [30] extends the classical Random Oracle Model. It still models an *ideal hash function* but gives the adversary $\mathcal{B}$ *quantum* access to a random oracle (which we generally denote as R_Q), thus allowing oracle queries in superposition. This disables many tricks used in the classical Random Oracle Model, as neither input awareness nor targeted programmability are possible.

Zhandry [142] introduced so-called *compressed oracles*, which add two main features to Quantum Random Oracles that are vital for our proofs: (1) the simulator can at a later point find out at which positions the oracle has been queried by the adversary and what the result of that query was, without allowing the adversary to detect that the query was recorded, and (2) the simulator can *efficiently* simulate a Quantum Random Oracle as the runtime of the simulator to provide an indistinguishable view is polynomial in the number of queries t sent by the adversary to the oracle. This allows proving post-quantum security of different constructions such as Merkle-Damgård [142]. The most basic idea of Compressed Oracles that separates them from previous approaches [143] is that instead of implicitly fixing the input-output-tuples in advance by responding according to a certain function, the approach from Zhandry [142] stores a superposition over all input-output mappings. While this initially contains each mapping with equal amplitudes, every query *entangles* the adversary's state with the oracle and any measurement outcome of the adversary *fixes* the QRO such that the superposition is only over mappings that respect the adversary's previous measurements. Zhandry [142] provides a way to efficiently simulate Compressed Oracles: using a compression mechanism the whole oracle requires qubits in the order of t, the number of queries, and provides a QPT circuit to respond to oracle queries.

13.3.2. Everlasting Security

Everlasting or *Long-Term* Security [112] allows to model future technological progress by assuming that computational problems might be efficiently breakable in the future. The model splits the security experiment into two phases: A polynomially bounded phase executing the protocol, and an unbounded phase where the adversary tries to extract information from the view of the former. The framework has previously been considered in the quantum setting by Unruh [129], that is, with quantum adversaries, but in the framework of Universal Composability (UC) [41]; resulting in an Everlasting UC framework.

Our standalone setting is less restrictive but does not provide universal composition. Our execution is split into two phases:

Phase one is executed against a QPT adversary $\mathcal{B}$. In this phase, a protocol Π is executed and the adversary $\mathcal{B}$ may perform computations. Upon termination of $\mathcal{B}$ his final state st is stored. The state contains all classical and quantum registers as well as the random tape.

Phase two starts after $\mathcal{B}$ has terminated (which happens after polynomially many time steps). In this phase, the unbounded adversary $\mathcal{A}$ gets activated with the state st of $\mathcal{B}$.

$$st \leftarrow \mathcal{B}(\Pi) \xrightarrow{\quad st \quad} \mathcal{A}$$

13.3.3. Quantum Random Oracles with Everlasting Security

We use QROs in the Everlasting setting. We build upon the model established by Unruh [130]. We assume that QROs can be efficiently simulated using techniques from Zhandry [142].

We model the QPT-adversary as oracle algorithm $\mathcal{B}^{R_Q}$ who can query R_Q a polynomial number (we generally denote this number as t) of times. After termination of $\mathcal{B}$, the unbounded adversary $\mathcal{A}$ obtains access to the QRO. Since $\mathcal{A}$ can query R_Q arbitrarily often, we model this by giving $\mathcal{A}$ direct access to the complete value table.

With this addition, our model looks as follows:

$$st \leftarrow \mathcal{B}^{R_Q}(\Pi) \xrightarrow{\qquad st \qquad} \mathcal{A}(R_Q)$$

Again, we stress that $\mathcal{A}(R_Q)$ means that $\mathcal{A}$ has access to the full value table of R_Q; thus upon activation of $\mathcal{A}$ the entire oracle has to be fixed.

14. Impossibility Result

It is folklore knowledge that classical (that is, schemes that do not use a quantum computer) bit commitment schemes cannot be unconditionally secure. The reason for that is that an unconditionally *hiding* commitment scheme implies a level of *ambiguity* regarding the bit committed to: After the commitment phase, the adversary $\mathcal{A}$ on the hiding property obtains a commitment $\mathtt{com}_b$ on some bit $b \in \{0,1\}$ from an honest committer. The unconditional hiding property states that even if $\mathcal{A}$ has no runtime restrictions, it cannot extract b from $\mathtt{com}_b$ with better advantage than by guessing. This implies in particular that the following attack does not work:

We denote by $\textsc{Com} = (\mathsf{Com}, \mathsf{Unv})$ the commitment scheme. The commitment function Com is a PPT protocol, which means that on input $1^\kappa, b$ it has some randomness applied to it. However, out of Com we can construct a *deterministic* commitment protocol Com' that takes as input $(1^\kappa, b, T_\mathrm{C})$, where T_C is the *random tape* used by the committer in the protocol. We replace all invocations of random sampling with n bit entropy by using the n next random bits of T_C and adjust distributions where necessary.

It is easy to see that (1) given fixed inputs, the protocol is *deterministic*, and (2) when sampling T_C of appropriate length uniformly at random before executing the protocol, the behavior is the same as in the original Com.

As Com' is deterministic and the adversary $\mathcal{A}$ has no runtime restrictions, the adversary $\mathcal{A}$ can perform a *complete search* over all choices of b and T_C. The adversary can do that for all possible inputs and when the output $\mathtt{com}_b$ matches that obtained from the real committer, $\mathcal{A}$ stores this input (and in particular the bit b that $\mathtt{com}_b$ commits to).

After having tested all possible inputs, the adversary checks the list if all inputs that resulted in $\mathtt{com}_b$ used the same bit b; if so it outputs b, if not it outputs a uniformly random bit.

As stated before, the requirement that the commitment scheme is unconditionally hiding means that this attack fails to provide a non-negligible advantage. As we assume that $\mathtt{com}_b$ was created honestly and that the commitment scheme $\textsc{Com}$ is correct, it follows that there is *at least* one input (with the correct bit b and the actual random tape T_C used by the committer) that resulted in $\mathtt{com}_b$. Yet the attack failing implies that there is also a different input $(1^\kappa, \bar{b}, T'_\mathrm{C})$ which commits to the complementary bit.

An unbounded adversary on the binding property can perform the exact same attack in order to later unveil $\mathtt{com}_b$ to either 0 or 1 as unveil information for both can be extracted from the attack above. This breaks the unconditional binding property and thus implies that no classical commitment scheme can be both unconditional binding *and* hiding.

The attack sketched above does work for any classical bit commitment scheme where $\mathtt{com}_b$ is a bitstring of finite length. However, performing the above attack if $\mathtt{com}_b$ is a *quantum state* is not possible: the direct search involves a comparison between the newly created state and the state received after the commit phase.

As such it was widely believed that quantum commitments can be both unconditionally binding and hiding, with Brassard et al. [36] providing a candidate construction they claimed (and proved) to be unconditionally secure for both parties.

A flaw in their proof alongside an attack was pointed out three years later by Mayers [107]. On a very high level, the flaw in the binding proof of Brassard et al. [36] was in their definition of the binding property. The authors proved that it is not possible to provide *two different* unveil informations $\mathtt{unv}_0$ and $\mathtt{unv}_1$ for a given commitment $\mathtt{com}_b$ that open the commitment to 0 and 1, respectively. While this makes sense in the classic setting, in the quantum world we do not need *both* unveil informations to break the binding property, it suffices if the adversary can come up with the one needed for this particular instance: If the adversary can commit in such a way, that he can perform a *targeted collapse* on the unveil information before starting the unveil phase, he can commit to a generic state and decide at a later point to which bit he wants to open the commitment.

The collapse of the quantum state then renders it impossible for the adversary to come up with unveil information for the complementary bit, yet this is not required as the binding property was already broken.

This attack was generalized to work on arbitrary quantum bit commitment schemes by Mayers [108] and, independently, by Lo and Chau [105]. Both consider only commitment protocols where $\mathtt{com}_b$ is a quantum state and the only measurement is performed right before termination, which is without loss of generality: Any hybrid protocol that uses classical *and* quantum information can be transformed into a protocol that exchanges *only* quantum information by encoding the classical bits into a quantum state in computational bases. And by using deferred measurement, the restriction that measurements are only possible at the end of the execution is justified.

The key idea of their proof is the following: In order for the quantum commitment to be unconditionally hiding, the quantum states for $\{\mathtt{com}_0\}$ and $\{\mathtt{com}_1\}$ need to have a "similar" description; on a formal level, this means that they have to be encoded by the same density matrix as otherwise a distinguishing attack works with non-negligible probability. This in term implies that the theorem by Hughston, Jozsa, and Wootters [88] can be applied which states that two quantum systems with similar decomposition can be transformed into each other.

This holds in particular for the two quantum states $\mathtt{com}_0$ and $\mathtt{com}_1$: The adversary can construct a quantum state containing two entangled registers, where the second register holds the quantum commitment. The second register is sent as quantum commitment to the receiver. Before the unveil phase, the adversary decides on the bit to unveil the commitment to, and performs a unitary transformation *only* to his own register which transforms the commitment to be on the desired bit b. By measuring the first register (the one that was not sent to the receiver) the adversary can extract the correct unveil information that is consistent with the receivers view so far. As such, the unveil information is accepted by the receiver and the binding property is broken.

15. Quantum Decay

15.1. Motivating the Assumption

This section motivates our first contribution, namely the assumption of Quantum Decay (QD) in a general model and provides an introduction into its formal treatment. We took this section verbatim from Döttling et al. [63, Section 2.5] and stress that it not written by the author of this thesis and is does *not* count to our contribution, but is here due to its relevance in justifying our assumption.

Our model assumes that it is impossible to preserve any quantum information after an arbitrary point in time $t \in \mathrm{poly}(\kappa)$; its decay caused the state to collapse to classical information. We use this model as one key ingredient to achieve everlasting security. The unintrigued reader may skip the motivation and jump to Definition 15.2.2.

The theoretical basis of quantum computation is the model of quantum circuits, in which one often assumes the existence of logical qubits that are perfectly separated from the outside world. However, every known implementation of a quantum device is an open system, *i.e.* the corresponding quantum states eventually couple to their environment resulting in a partial loss of coherence—"decoherence". This process is also known as quantum decay.

A more realistic approach is known as fault-tolerant quantum computing using error correcting codes (for example using surface codes [125, 37, 71]) allowing to perform any algorithmic sequence on a quantum computer as long as the physical error per gate is sufficiently low: Shor [123] showed how to construct an efficient quantum circuit that tolerates an inaccuracy of $O(\frac{1}{\log t})$, where t is number of gates of the circuit, suggesting that (fault-tolerant) quantum computing can be performed if the error induced by the computation is below a logarithmic threshold. The result was later improved to a constant inaccuracy by Aharonov and Ben-Or [6]. Therefore the model assumes that partial decay can be counteracted using error correction if the state is refreshed in additional ancilla states.

In general, decoherence obstructs the realization of *efficient* quantum circuits by causing a deviation between the anticipated and the actual state. This process may be quantified by a (discrete) time it takes a quantum state to deviate from an initial state ψ_{init} to an erroneous state ψ_{error}.

Another possibility is to consider the fidelity of the two states $\mathrm{F}(\psi_{\text{init}}, \psi_{\text{error}})$, *i.e.* the probability that the state ψ_{error} passes as the state ψ_{init} when measuring. In general, the quantification is expressed as inaccuracy from either of the two methods.

To stay below this threshold, both physical and theoretical countermeasures are deployed: Quantum machines are shielded in nearly closed environments, *e.g.* placed in an almost perfect vacuum colder than space [66]. However, these safeguards do not prevent but merely slow the process of decay. A recent result by Zhang et al. [144] shows a fidelity between a prepared state and the expected state at a magnitude of 10^{-3} for a single qubit system, or about 10^{-5} for two qubit systems. The famous result on quantum supremacy by Arute et al. [15, Supl, Sec. B] shows that the fidelity gap, thus the probability of error, grows exponentially in the number of qubits, suggesting that storing larger number of qubits results in larger failure probabilities.

Employing these countermeasures, state-of-the-art implementations such as the *IBM Q* series reach coherence times of about 90 microseconds, *Google*'s Bristlestone quantum computer [110, 80] about 35 microseconds (see Tannu and Qureshi [126] for a detailed survey). It may be noted that quantum computers add additional noise from interacting with qubits, suggesting that simply storing state may be easier than manipulating it. Recent results demonstrate coherence times of 10 minutes [135] to 6 hours [145] for single qubits; or at most 39 minutes [120] for multiple qubits.

Advancing technology allows to increase coherence timings and thus reduce the impact of decay allowing to compute for a short amount of time. Therefore, *practical* but not fully fault-tolerant quantum computing may become possible in the close future: In particular, this work considers security against quantum computers that keep states coherent for a limited amount of time, for example, enough time to perform any efficient protocol (*e.g.* minutes, hours, days, weeks, ...) but that do not achieve complete fault-tolerance, *i.e.*, can preserve a quantum state for an indefinite amount of time.

For our model, we do not assume that decoherence may increase over time. Instead we only consider the moment where decay is complete and assume that an adversary can perform any BQP computation prior to that on a fully coherent state.

15.2. Mathematical Description

This section was taken verbatim from Döttling et al. [63] and considers the mathematical modeling of everlasting security. This is *not* a contribution of this thesis.

The loss of information in an imperfectly closed system due to environmental noise can be modeled as a transformation of the respective quantum state. Such an irreversible transformation is considered to be decoherent, *i.e.* it is named after the loss of coherence, thus the loss of relations between multiple states; the outcome of which is a state similar to a measurement result.

We follow the description of Hornberger [87] and Brandt [35], who describe decoherence as the coupling of a quantum system $|\psi_q\rangle$ to the environment $|\psi_e\rangle$ which is unobservable and uncontrollable due to a large number of degrees of freedom. The environmental state can be described in the *decoherence basis* where $|e_0\rangle$ and $|e_1\rangle$ correspond to the orthogonal basis vectors. Furthermore, let $|\psi\rangle = |\psi_q\rangle \otimes |\psi_e\rangle$ describe the complete system. For the sake of simplicity, consider the two-state quantum system $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ with the density matrix ρ. Assume that the initial state is pure, thus having the density matrix representation $\rho_0 = \begin{pmatrix} |\alpha_0|^2 & \alpha_0\alpha_1^* \\ \alpha_0^*\alpha_1 & |\alpha_1|^2 \end{pmatrix}$, where the diagonal elements characterize the basis states, and the off-diagonal elements correspond to relaying basis states to each other.

It is assumed that the quantum state eventually couples to the environment, resulting in the density matrix $\rho_1 := |\psi_q\rangle|\psi_e\rangle\langle\psi_e|\langle\psi_q|$. The corresponding view of $|\psi_q\rangle$ can be obtained by tracing out the environment:

$$\begin{aligned} \mathrm{tr}_e(\rho_1) &= \alpha_0^2|0\rangle\langle 0|\mathrm{tr}(|e_0\rangle\langle e_0|) + \alpha_0\alpha_1^*|0\rangle\langle 1|\mathrm{tr}(|e_0\rangle\langle e_1|) \\ &\quad + \alpha_0^*\alpha_0|1\rangle\langle 0|\mathrm{tr}(|e_1\rangle\langle e_0|) + \alpha_1^2|1\rangle\langle 1|\mathrm{tr}(|e_1\rangle\langle e_1|)\,. \end{aligned} \tag{15.1}$$

This results in a state where the outcome is dependent on the trace of the decoherence basis. Let us consider an example where the decoherence basis of the environment is the computational basis such that $e_0 = |0\rangle$ and $e_1 = |1\rangle$. Then the partial trace results in $\mathrm{tr}_e(\rho_1) = \begin{pmatrix} |\alpha_0|^2 & 0 \\ 0 & |\alpha_1|^2 \end{pmatrix}$. Hence, the decoherence would be equivalent to an indirect measurement process, which maps the state $|\psi\rangle$ to a probability distribution over basis states, thus destroying any superposition information. This example setup corresponds to full decoherence.

However, the decoherence bases $|e_0\rangle, |e_1\rangle$ are not orthogonal in general, hence the off-diagonal elements may not vanish completely in a single time step. Nevertheless, the system is exposed to decoherence over a continuous time scale, such that coupling may occur multiple times, resulting in an continuous vanishing of the off-diagonal elements over time.

We consider quantum states over time $(t_0, t_1, \ldots, t_{\mathrm{poly}(\kappa)})$ during the execution of an (efficient) protocol. Throughout the protocol we assume the state to be coherent, such that any party, including the adversary, can perform arbitrary transformations. At the time step $t_{\mathrm{poly}(\kappa)}$, the quantum state couples with the environment, causing full decoherence and collapsing the superposition to a probability distribution of classical information.

When considering an adversarial setting, the attacker may not necessarily control when the decoherence occurs. Moreover, he may measure at any time or not at all. However, we stress that an adversary who terminates with unmeasured registers can be treated equivalently to one who measures all registers before termination. Thus performing a measurement does not yield a weaker adversary than collapsing by a random measurement.

Definition 15.2.1 (Quantum Decay Assumption)**.** *Every quantum state $|\psi\rangle$ decays after $t \in \mathrm{poly}(\kappa)$ steps. We denote by time (t_0) the time where a quantum state $|\psi\rangle$ is written into the quantum register. At time $(t_0 + t)$ the register contains a state that is independent from $|\psi\rangle$.*

We combine everlasting security from Section 13.3.2 and Quantum Decay to obtain the adversarial structure from Definition 15.2.2: The computationally bounded QPT-adversary $\mathcal{B}$ executes the (quantum) protocol with access to a quantum state. Upon termination, the Quantum Decay collapses all quantum registers and hence the unlimited adversary $\mathcal{A}$ only gets access to the classical part of $\mathcal{B}$s state. Therefore the Quantum Decay model only allows QPT operations on the quantum state.

Definition 15.2.2 (Quantum Decoherence in the Everlasting Setting)**.** *Let $\mathcal{B}$ be a QPT-adversary. Let st be the final state of $\mathcal{B}$. Let Π be a protocol. Let $\mathcal{A}$ be an unbounded adversary. The final state of $\mathcal{B}$ visible to $\mathcal{A}$ contains only classical information.*

$$(st) \leftarrow \mathcal{B}(\Pi) \xrightarrow{\quad st \quad} \mathcal{A}$$

Definition 15.2.3 (Quantum Decoherence with a QROM in the Everlasting Setting)**.** *Let R_Q be a quantum random oracle. A protocol Π is additionally set in the Quantum Random Oracle Model if the definitions from Definition 15.2.2 are fulfilled and in addition if the bounded adversary $\mathcal{B}^{(R_Q)}$ has oracle-access to R_Q. Additionally the unbounded classical adversary $\mathcal{A}$ gets access to the full value table of R_Q.*

$$st \leftarrow \mathcal{B}^{(R_Q)}(\Pi) \xrightarrow{st,\, msg \,\in\, \{0,1\}^{\mathrm{poly}(\kappa)}} \mathcal{A}$$

15.3. Simulating QROs

Simulating a Quantum Random Oracle using compressed oracles [142] fits surprisingly well to the everlasting setting. Essentially, during the QPT-phase, the simulator uses the lazy sampling method described by Zhandry [142] to answer queries. Since the adversary is also everlasting, the number of queries from the QPT-adversary are limited by a polynomial number $t \in \mathrm{poly}(\kappa)$. After termination of the QPT-adversary, execution is continued by a purely classical (unbounded) adversary which has no access to the qubits stored by the QPT-adversary. Thus, we can treat the quantum information owned by the QPT-adversary as if it has been decayed and measure the database which is then entirely classical. Any collapse will not be noted since the adversary no longer holds the quantum states; either the QPT-adversary measured them before termination, in which case measuring the database changes nothing, or the QPT-adversary kept them in superposition, in which case the measurement from the QPT simulator causes a collapse which is unnoticeable since the unbounded adversary has no access to the registers that store the quantum information. Hence, the QPT simulator can read the database in the Fourier domain and fix the reported oracle values at those spots. Those values are then accessible by the unbounded simulator.

This unbounded simulator *forges* an indistinguishable value table by copying the in- and outputs recorded by the QPT simulator to the value table, and filling all the remaining inputs with additional fresh randomness. This value table is then sent to the unbounded adversary.

16. Everlasting Quantum Commitment Protocol

This section contains our commitment protocol Π_{QCom} which exploits properties of Quantum Decay. We prove the security of Π_{QCom}—and hence Theorem 12.1.1—in the following sections. Our protocol extends the work of Brassard et al. [36]; we first recall their commitment protocol. Then we present our extensions, which result in the protocol Π_{QCom}: It lets a sender commit to a single (classical) bit and is *information theoretically* binding and everlasting hiding in the QROM.

In the sections to come we prove that once the sender committed to a bit b he cannot convince the receiver that the commitment was on $\bar{b}$, and no *feasible* measurement of the receiver allows to break the hiding property before the unveil-phase.

16.1. The BCJL Protocol

The BCJL protocol deploys a Privacy Amplification Function which is based on a linear $[\mathcal{N}, \mathcal{K}, \mathcal{D}]$ code $\mathcal{C}_{\vec{G}}$ with generator matrix $\vec{G}$—initially constructed by the receiver—and a secret one-time-pad $otp \in \mathbb{F}_2^{\mathcal{N}}$ which is chosen by the sender. To commit to a bit b the sender picks a codeword $c \in \mathcal{C}_{\vec{G}}$ and a string otp such that $b = \langle c, otp \rangle$. The encoding function $\text{encode}_{\vec{b}}$ maps the codeword c to a quantum state $|\psi\rangle$ using a random vector of Wiesner bases $\vec{b} \in \{+, \times\}^{\mathcal{N}}$. The resulting state $|\psi\rangle$ is key to the commitment. The sender sends the state $|\psi\rangle$ and the OTP otp to the receiver, who measures the state in uniformly random Wiesner bases, yielding a codeword c'.

During the unveil phase, the sender sends the committed message b, the codeword c and the encoding bases $\vec{b}$. The receiver checks if his measurements are consistent with the received values and if the the inner product $\langle c, otp \rangle$ matches the message b, and accepts or rejects accordingly.

The full protocol can be found in detail in Figs. 16.1 and 16.2.

Protocol Π_{Com}

The commitment protocol Π_{Com} by Brassard et al. [36] for committing to a given bit $b \in \{0, 1\}$. It is running with a set of 2 parties (C, R) where C is the committer and R is the receiver.

Upon activation, R samples a uniformly random generator matrix $\vec{G} \sim \mathbb{F}_2^{\mathcal{K}\times\mathcal{N}}$ and sendes $\vec{G}$ to C.

On input (b) and after $(\vec{G})$ has been received from R, C samples $otp \sim \{0, 1\}^{\mathcal{N}}$ and $\vec{b} \sim \{+, \times\}^{\mathcal{N}}$ and samples a codeword $\mathscr{c} \in \mathscr{C}_{\vec{G}}$ *s.t.* $\langle \mathscr{c}, otp\rangle = b$. Then C prepares a quantum state $|\psi\rangle \leftarrow \mathsf{encode}_{\vec{b}}(\mathscr{c})$ using Wiesner's encoding with bases $\vec{b}$ and sends $(|\psi\rangle, otp)$ to R.

On input $(|\psi\rangle, otp)$ by C, R samples uniformly random $\vec{b}' \sim \{+, \times\}^{\mathcal{N}}$ and measures and stores $\mathscr{c}' \leftarrow \mathsf{measure}_{\vec{b}'}(|\psi\rangle)$ with bases $\vec{b}'$.

Figure 16.1.: The commitment protocol Π_{Com} from Brassard et al. [36] for committing to a given bit $b \in \{0, 1\}$.

Protocol Π_{Unv}

The unveil protocol Π_{Unv} by Brassard et al. [36] for unveiling a commitment. We denote by ξ the noise of the channel.

Upon activation, C sends $(\mathscr{c}, \vec{b}, b)$ to R.

On input $\left(\mathscr{c}, \vec{b}, b\right)$ from C, R computes the error rate as $\epsilon := \sum_{i|\vec{b}[i]=\vec{b}'[i]} \frac{\mathscr{c}[i]\oplus\mathscr{c}'[i]}{\mathcal{N}/2}$ and rejects if any of the three conditions are not fulfilled: (1) $\epsilon < 1.4\xi$, (2) $\mathscr{c} \in \mathscr{C}_{\vec{G}}$, and (3) $\langle \mathscr{c}, otp\rangle = b$. Otherwise, R accepts.

Figure 16.2.: The unveil protocol Π_{Unv} from Brassard et al. [36] for unveiling a commitment.

16.2. Everlasting Quantum Commitment Protocol

In this section we present a bit commitment scheme which we claim to be unconditionally binding and *everlasting* hiding. We prove its security in the QROM under the assumption of Quantum Decay. In particular, we claim that the inevitable Quantum Decay naturally occurring in quantum states enables us to prove the security of our scheme: Remember that the attack of Mayers [108] requires the adversary to store the quantum state for an arbitrary amount of time. If this does not hold, *e.g.* if the receivers quantum state becomes unusable after polynomially many time steps due to the Quantum Decay assumption, then

Protocol Π_{Com}

The everlasting commitment protocol Π_{Com} for committing to a given bit $b \in \{0,1\}$. It is running with a set of 2 parties (C, R) where C is the committer and R is the receiver. It is parameterized by an unconditionally binding and computationally hiding commitment scheme $\textsc{Com} = (\mathrm{Com}, \mathrm{Unv})$.

Upon activation, R samples a uniformly random generator matrix $\vec{G} \sim \mathbb{F}_2^{\mathcal{K} \times \mathcal{N}}$ and sendes $\vec{G}$ to C.

On input (b) and after $(\vec{G})$ has been received by R, C samples $otp \sim \{0,1\}^{\mathcal{N}}$ and $\vec{b} \sim \{+, \times\}^{\mathcal{N}}$ and samples a codeword $c \in \mathcal{C}_{\vec{G}} s.t. \langle c, otp\rangle = b$. Then C prepares a quantum state $|\psi\rangle \leftarrow \mathrm{encode}_{\vec{b}}(c)$ using Wiesner's encoding with bases $\vec{b}$, commits to the bases using $(\mathrm{com}_{\vec{b}}, \mathrm{unv}_{\vec{b}}) \leftarrow \textsc{Com}.\mathrm{Com}(\vec{b})$ and sends $(|\psi\rangle, otp, \mathrm{com}_{\vec{b}})$ to R.

On input $\left(|\psi\rangle, otp, \mathrm{com}_{\vec{b}}\right)$ by C, R samples uniformly random $\vec{b}' \sim \{+, \times\}^{\mathcal{N}}$ and measures and stores $c' \leftarrow \mathrm{measure}_{\vec{b}'}(|\psi\rangle)$ with bases $\vec{b}'$.

Figure 16.3.: The everlasting commitment protocol Π_{Com} for committing to a given bit $b \in \{0,1\}$. The shaded area shows the extension of the protocol from Fig. 16.1.

the measurement of the commitment collapses the quantum state and prevents a malicious sender from mounting the attack after the commitment phase.

16.2.1. The protocol

In order to exploit the Quantum Decay property, we encode data in a quantum state and send additional data regarding the state as classical information; the classical information is computationally hidden and only contains information that is useless after the collapse of the quantum state. Yet with the information encoded in the quantum state breaking the commitment becomes trivial. Without it, not even an unbounded adversary can extract the secret from the commitment. It may be noted that with the subsequent instantiation in the QROM the commitment will be unconditionally binding and computationally hiding.

Figs. 16.3 and 16.4 shows the commitment protocol Π_{QCom} containing the two methods Com and Unv. We use the same linear code, the same encoding in the Wiesner bases and the same mechanism for the privacy amplification as Brassard et al. [36]. The main difference is that we additionally use a *classical* commitment scheme to commit to the bases $\vec{b}$ in order to circumvent the attack by Mayers [107] by exploiting Quantum Decay.

Protocol Π_{Unv}

The everlasting unveil protocol Π_{Unv} for unveiling a commitment. We denote by ξ the noise of the channel.

Upon activation, C sends $(c, \vec{b}, b, \mathtt{unv}_{\vec{b}})$ to R.

On input $\left(c, \vec{b}, b, \mathtt{unv}_{\vec{b}}\right)$ from C, R computes the error rate as $\epsilon := \sum_{i|\vec{b}[i]=\vec{b}'[i]} \frac{c[i]\oplus c'[i]}{\mathcal{N}/2}$ and rejects if any of the three conditions are not fulfilled: (1) $\epsilon < 1.4\xi$, (2) $c \in \mathscr{C}_{\vec{G}}$, (3) $\langle c, otp\rangle = b$, and (4) $\textsc{Com}.\mathrm{Unv}(\mathtt{com}_{\vec{b}}, \mathtt{unv}_{\vec{b}}, \vec{b})$. Otherwise, R accepts.

Figure 16.4.: The everlasting unveil protocol Π_{Unv} for unveiling a commitment.

Similar to the verification performed in [36], the receiver has to check the bit values of c on those spots where the bases $\vec{b}'$ were guessed correctly and verifies that these match with his measurement c' and the unveil information $\mathtt{unv}_{\vec{b}}$ for validity against the commitment $\mathtt{com}_{\vec{b}}$ on $\vec{b}$. Based on this protocol, we state our main theorem. We will describe the instantiation in Chapter 17 and use this to prove the theorem in Chapter 18.

Theorem 16.2.1 (Everlasting Quantum Bit Commitment in the QROM)**.** *Let Π_{QCom} be the commitment protocol from Figs. 16.3 and 16.4. Assuming Quantum Decay and the QROM, there exists an instantiation of the classical commitment* $\textsc{Com}$ *for which Π_{QCom} is everlasting hiding and unconditionally binding.*

17. Instantiating the Classical Commitment in the Quantum Random Oracle Model

In this section we describe an instantiation of the classical commitment in the Quantum Random Oracle Model. Instantiating the quantum bit commitment protocol Π_{QCom} from Figs. 16.3 and 16.4 with this classical commitment yields a provably secure protocol.

For our proof we need to change some values of a Quantum Random Oracle after the protocol execution. In a sense, this could be considered as a very weak form of *reprogramming* the random oracle *statically*, where the new $(x, R_Q(x))$ pairs need to be chosen *in advance*, *i.e.* before the adversary sends the first query. However, a slightly different way to look at this problem is by considering a *distinguishing* problem similar to *closeness testing* for discrete distributions [47]. There, the adversary needs to differentiate between an oracle that follows a distribution p, or any distribution p' that is *at least* s off, *i.e.* where $\|p' - p\| \geq s$ for a fixed slack s. The difference for us is that (1) we consider a *quantum adversary* in the QROM, and (2) we investigate the setting where the truth table is either *exactly* that of the oracle, or differs in *exactly* k spots.

17.1. Closeness-Testing of Quantum Random Oracles

In this section we consider a special case of closeness-testing, namely one where a quantum adversary tries to decide after t queries, whether the oracle responded according to a given truth table, or if the real truth table varies on exactly k spots from the one provided to the adversary after the t queries.

To the best of our knowledge, this question has never been investigated before. A somewhat related question was under thorough investigation by Ambainis, Rosmanis, and Unruh [10] and Unruh [130], where the goal of the quantum adversary was to distinguish an *all-zero* oracle from a very *sparse* oracle that only returns 1 only with probability γ with a limited number of queries. While this would suffice as foundation for our setting, we will use a different and much simpler proof method based on the framework from Chung et al. [53] where analysis in the QROM can be made using *classical* arguments only.

In this section we will prove the following lemma:

Lemma 17.1.1 (Closeness-Testing in Quantum Random Oracles). *Let R_Q and $\widetilde{R_Q}$ be two Quantum Random Oracles that map $\{0,1\}^\kappa \to \{0,1\}^{\mathrm{poly}(\kappa)}$. Let R_Q and $\widetilde{R_Q}$ be equivalent except for a set of k uniformly random spots $(w_1, \ldots, w_k) \in (\{0,1\}^\kappa)^k$, where $R_Q(w_i) \neq \widetilde{R_Q}(w_i)$. Let $(C_1, \ldots, C_k) := (R_Q(w_1), \ldots, R_Q(w_k))$ be the images of the w_i in R_Q.*

Let $\mathcal{B}$ be an oracle algorithm that makes at most $t \in \mathrm{poly}(\kappa)$ oracle queries and has access to $\{C_i\}_{i=1}^k$, and let $\mathcal{A}$ be a classical algorithm that can access the (classical part of the) final state st of $\mathcal{B}$, we have

$$\begin{aligned} &|\Pr\left[b = 1 \;\middle|\; \begin{array}{l} st \leftarrow \mathcal{B}^{R_Q}(\{C_i\}_{i=1}^k), \\ b \leftarrow \mathcal{A}(st, R_Q) \end{array} \right] \\ &- \Pr\left[b = 1 \;\middle|\; \begin{array}{l} st \leftarrow \mathcal{B}^{\widetilde{R_Q}}(\{C_i\}_{i=1}^k), \\ b \leftarrow \mathcal{A}(st, R_Q) \end{array} \right] | \in \mathrm{negl}(\kappa) \end{aligned} \tag{17.1}$$

Proof. As mentioned before, we perform our proof in the framework provided by Chung et al. [53], wherein a proof against a *quantum* adversary in the QROM can be made using purely classical arguments. *W.l.o.g.* we use a non-parallel version of the QROM, *i.e.* one where each query contains one single quantum state[1]. On a high level, our claim holds because either the adversary manages to query a point where the two oracles *differ* from each other—in which case the adversary can check against the truth tables—or all t queries were made on spots where both oracles return the same values—meaning that the adversary can only guess. Since the adversary has to send queries *before* knowing which spots differ (as the challenges do not provide any advantage as was shown by Chung et al. [53, Example 2]), the probability of hitting one spot where the two oracles differ is negligible in the security parameter. So with overwhelming probability, the adversary ends up with queries that don't help to distinguish the truth tables, where the best option is to guess, and only with negligible probability the queries actually contain usable information.

On a formal level, we define a property to determine closeness, called *CLOS*, for the compressed oracles:

$$CLOS := \{R_Q | \exists_x s.t. R_Q(x) \in \{C_1, \ldots C_k\}\} \tag{17.2}$$

Depending on *CLOS*, the probability to successfully distinguish is given as:

$$\Pr[\mathcal{A} \text{ guesses correctly}] \leq \Pr[CLOS] \cdot 1 + \Pr[\neg CLOS] \cdot \frac{1}{2} \tag{17.3}$$

Since $\Pr[CLOS] + \Pr[\neg CLOS] = 1$ it suffices to show $\Pr[CLOS] \in \mathrm{negl}(\kappa)$ to prove our claim.

[1] For parallel access with the same total number of queries the success probability would be worse.

The classical case. Following Chung et al. [53, Lemma 4], we can upper-bound the probability that after t queries, the random oracle is in *CLOS*:

$$\begin{aligned}\Pr[{R_Q}^{(t)} \in \mathit{CLOS}] &\leq \sum_{i=1}^{t} \Pr[{R_Q}^{(i)} \in \mathit{CLOS} | {R_Q}^{(i-1)} \notin \mathit{CLOS}] \\ &\leq t \cdot [\neg\mathit{CLOS} \longrightarrow \mathit{CLOS}]\end{aligned} \tag{17.4}$$

where the latter is called *transition capacity* by Chung et al. [53] and holds as long as the queries are answered independently (*i.e.* the returned values of $R_Q(x)$ are independent of the other values of the oracle), which is the case in our setting.

So to bound the probability we need to get accurate values for the transition capacity. For that, note that a single query returns one $\mathrm{poly}(\kappa)$-bit string, $k \in \mathrm{poly}(\kappa)$ of which would be in the set that transitions the database to fulfill *CLOS*. Thus,

$$[\neg\mathit{CLOS} \longrightarrow \mathit{CLOS}] \leq \frac{k}{2^{\mathrm{poly}(\kappa)}} \tag{17.5}$$

Thus, by putting Eq. (17.5) into Eq. (17.4) we get that:

$$\Pr[{R_Q}^{(t)} \in \mathit{CLOS}] \leq t \cdot \frac{k}{2^{\mathrm{poly}(\kappa)}} \in \mathrm{negl}(\kappa) \tag{17.6}$$

where the negligible property comes from the fact that we assume both the number of queries, t, and the number of points where R_Q and ${R_Q}'$ differ, k, to be polynomial in the security parameter.

The quantum setting. The above holds for the classic setting, yet our lemma is with respect to a quantum adversary who has access to a QROM. Using the framework by Chung et al. [53] we can *recycle* parts from the classic proof from above, yet some values have to be adjusted. Most notably, Chung et al. [53, Theorem 1] states that:

$$\sqrt{\Pr[{R_Q}^{(t)} \in \mathit{CLOS}]} \leq \sum_{i=1}^{t} \Pr[{R_Q}^{(i)} \in \mathit{CLOS} | {R_Q}^{(i-1)} \notin \mathit{CLOS}], \tag{17.7}$$

and due to Chung et al. [53, Theorem 2] it holds that

$$[\neg\mathit{CLOS} \longrightarrow \mathit{CLOS}] \leq \sqrt{10 \Pr[{R_Q}^{(t)} \in \mathit{CLOS}]}. \tag{17.8}$$

So, by using the rule from Eq. (17.8) on the formula from Eq. (17.4), it now holds for the transition capacity that:

$$\begin{aligned}[\neg\mathit{CLOS} \longrightarrow \mathit{CLOS}] &\leq \sqrt{10 \Pr[{R_Q}^{(t)} \in \mathit{CLOS}]} \\ &\leq \sqrt{\frac{10k}{2^{\mathrm{poly}(\kappa)}}}\end{aligned} \tag{17.9}$$

Protocol Π_{Com}

The classical commitment protocol Π_{Com} for committing to a given bit $b \in \{0,1\}$ in the Quantum Random Oracle Model. It is running with a set of 2 parties (C, R) where C is the committer and R is the receiver.
We denote by R_Q the Quantum Random Oracle.

On input b, C samples a uniformly random nonce $v \sim \{0,1\}^\kappa$ and computes $\mathtt{com}_b := R_Q(b \| v)$. C sends $\mathtt{com}_b$ to R.

Figure 17.1.: The classical commitment protocol Π_{Com} in the QROM for committing to a given bit $b \in \{0,1\}$.

Protocol Π_{Unv}

The classical unveil protocol Π_{Unv} for unveiling a commitment in the Quantum Random Oracle Model.

Upon activation, C sends $\mathtt{unv}_b := (b, v)$ to R.

On input $(\mathtt{unv}_b := (b, v))$ from C, R computes $\mathtt{com}'_b := R_Q(b \| v)$ and aborts if $\mathtt{com}'_b \neq \mathtt{com}_b$. Otherwise, R accepts.

Figure 17.2.: The classical unveil protocol Π_{Unv} in the QROM for unveiling a commitment.

And thus, due to Eq. (17.7):

$$\sqrt{\Pr[{R_Q}^{(t)} \in CLOS]} \leq t \cdot \sqrt{\frac{10k}{2^{\mathrm{poly}(\kappa)}}} \tag{17.10}$$

Finally, by squaring both sides, we get that

$$\Pr[{R_Q}^{(t)} \in CLOS] \leq \frac{10t^2k}{2^{\mathrm{poly}(\kappa)}} \in \mathrm{negl}(\kappa) \tag{17.11}$$

as again, t and k are both polynomial in the security parameter κ. This concludes our proof. □

17.2. A Classical Commitment Protocol

We present a protocol construction in the Quantum Random Oracle Model (QROM) for a classical commitment protocol in Figs. 17.1 and 17.2. The protocol is fairly canonical; to commit to a bit $b \in \{0,1\}$, the sender picks a nonce v of length κ uniformly at random and sends the bit b together with v to the QRO R_Q. The returned value is used as commitment on b. In the unveil phase, the sender sends both the nonce v and the bit b to the receiver, who then verifies that $R_Q(b \| v)$ matches the commitment. The hiding property follows from the randomness and one-wayness of R_Q, the binding property comes from the hardness of finding a collision in R_Q. The description of the protocol based on the QRO is sufficient to follow our proof of the quantum bit commitment Π_{QCom}. However, for completeness we provide a security proof of the classical commitment COM.

17.3. Security Analysis

17.3.1. Unconditional Binding Property

In this section we formally investigate the receivers security of this protocol. Intuitively, the binding property of COM follows from the injective nature of R_Q. More formally:

Lemma 17.3.1 (Unconditional Binding Property). *Assuming that* $R_Q \colon \{0,1\}^{\kappa+1} \to \{0,1\}^{\text{poly}(\kappa)}$ *is a Quantum Random Oracle, then the classical commitment protocol from Figs. 17.1 and 17.2 is* unconditionally *binding.*

Proof. A given commitment $\mathtt{com}_b$ has *at most* one pre-image in R_Q with overwhelming probability as R_Q maps strings of length $\kappa + 1$ to length $\text{poly}(\kappa)$. Hence a collision exists with only negligible probability. An adversary $\mathcal{A}$ who breaks the binding property has to send a commitment $\mathtt{com}_b = R_Q(b\|v)$ and find some v' such that $R_Q(\bar{b}\|v') = R_Q(b\|v)$, which corresponds to a *collision* in R_Q. □

17.3.2. Quantum-Computational Hiding Property

To show the hiding property, we show by consecutive games, each of which is not efficiently distinguishable by any QPT distinguisher, that the distribution of commitments to 0 is indistinguishable from the distribution of commitments to 1.

We formally introduce the four games. We start from a distribution of COM.Com(0), and end at a distribution of COM.Com(1). The games are formally shown in Fig. 17.3:

GAME$_1(\kappa)$: This game follows the honest protocol, where the commitment is to $b = 0$.

GAME$_2(\kappa)$: This game is as GAME$_1(\kappa)$, but after obtaining the original output of the Quantum Random Oracle on input $0\|v$, R_Q responds to input $0\|v$ with output r for an independently sampled $r \xleftarrow{\$} \{0,1\}^{\text{poly}(\kappa)}$.

```
$\textsc{Game}_1(\kappa)$
1 : $v \xleftarrow{\$} \{0,1\}^\kappa$
2 : $-$
3 : $\mathtt{com}_b := R_Q(0\|v)$
4 : $-$
5 : return $\mathtt{com}_b$
```

```
$\textsc{Game}_2(\kappa)$
1 : $v \xleftarrow{\$} \{0,1\}^\kappa$
2 : $r \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(\kappa)}$
3 : $\mathtt{com}_b := R_Q(0\|v)$
4 : $R_Q(0\|v) := r$
5 : return $\mathtt{com}_b$
```

```
$\textsc{Game}_3(\kappa)$
1 : $v \xleftarrow{\$} \{0,1\}^\kappa$
2 : $r \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(\kappa)}$
3 : $\mathtt{com}_b := R_Q(1\|v)$
4 : $R_Q(1\|v) := r$
5 : return $\mathtt{com}_b$
```

```
$\textsc{Game}_4(\kappa)$
1 : $v \xleftarrow{\$} \{0,1\}^\kappa$
2 : $-$
3 : $\mathtt{com}_b := R_Q(1\|v)$
4 : $-$
5 : return $\mathtt{com}_b$
```

Figure 17.3.: Hiding games for the commitment $\textsc{Com}$.

Lemma 17.3.2. *Let $\textsc{Com}$ be given as in Figs. 17.1 and 17.2. Let $\mathcal{B}'$ be a QPT distinguisher which distinguishes $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$ with advantage α. By Lemma 17.1.1, it follows that $\alpha \in \mathrm{negl}(\kappa)$.*

Proof. Indistinguishability can be shown by applying Lemma 17.1.1 with $k = 1$: $\textsc{Game}_1(\kappa)$ uses the original QRO that provides $R_Q(0\|v)$, and in $\textsc{Game}_2(\kappa)$, the oracle differs on one random pre-image, such that with overwhelming probability $\mathtt{com}_b$ does not have a pre-image.

From a QPT distinguisher $\mathcal{B}'$ between $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$, we can build a QPT reduction algorithm $\mathcal{B}$ that can decide the closeness-testing problem in Lemma 17.1.1:

The challenger C_{PreImg} for Lemma 17.1.1 provides $\mathcal{B}$ with a QRO $R_Q\colon \{0,1\}^\kappa \mapsto \{0,1\}^{\mathrm{poly}(\kappa)}$, and a message C. We let $\mathcal{B}$ simulate a compressed oracle $R_C\colon \{0,1\}^\kappa \mapsto \{0,1\}^{\mathrm{poly}(\kappa)}$.

Before providing oracle access for $\mathcal{B}'$, $\mathcal{B}$ reports the commitment $\mathtt{com}_b = C$ to α' and provides a QRO $R_S\colon \{0,1\}^{\kappa+1} \mapsto \{0,1\}^{\mathrm{poly}(\kappa)}$.

When α' queries the oracle on input $|x\rangle$, $\mathcal{B}$ interprets $|x\rangle$ as $|b\rangle \otimes |x'\rangle$: the first register is considered a (superposition over a) bit b and the remaining κ qubits are used as additional input. $\mathcal{B}$ forwards $|x'\rangle$ to R_Q and executes the following circuit on the result: It uses $|b\rangle$ for a *controlled* circuit that uses CNOT gates to copy the output of only one register to the output registers for the adversary's oracle query: conditioned on $b = 0$, the output registers contain the result of the challenge oracle R_Q, and conditioned on $b = 1$, the output registers hold the result of the simulated oracle $R_C(|x'\rangle)$. The resulting state is returned by $\mathcal{B}$ as response $R_S(|x\rangle)$.

After t queries, $\mathcal{B}'$ provides output. If $\mathcal{B}'$ assumes playing $\textsc{Game}_1(\kappa)$, $\mathcal{B}$ reports to C_{PreImg} that the value table corresponds to the oracle; and if $\mathcal{B}'$ guesses $\textsc{Game}_2(\kappa)$, $\mathcal{B}$ reports that k inputs of the value tables differ from the ones that were provided by R_Q.

With overwhelming probability there is a pre-image of C in R_S only if the challenge R_Q was used. Our intermediate circuit ensures it has a leading 0. The probability that the compressed oracle has a pre-image on C is negligible due to the sparse image domain. Hence, a pre-image of C in R_S corresponds to a pre-image of C in R_Q with overwhelming probability.

There are two important cases. The one case happens with overwhelming probability due to the sparsity of the domain and in it, the challenge C has no pre-image in the compressed oracle R_C. In this case, the following holds:

If the value table differs for k inputs, the situation resembles *exactly* that of $\textsc{Game}_2(\kappa)$ and hence, the advantage of $\mathcal{B}'$ for this case is directly inherited by $\mathcal{B}$.

Whereas if the value table is exactly that from R_Q, then there is a unique pre-image of r in R_Q that has a leading 0 in the oracle accessible by $\mathcal{B}'$. Thus in this case the situation is exactly that of $\textsc{Game}_1(\kappa)$ and $\mathcal{B}$ again inherits the advantage of $\mathcal{B}'$.

The second case is the one where the compressed oracle R_C has a pre-image for the challenge C. In this case the simulation cannot successfully reconstruct the expected view of the respective games and we can make no guarantees with respect to the advantage. However, the sparse domain of the compressed oracle R_C implies that this case occurs with at most negligible probability.

From $\mathcal{B}' \in QPT$ it follows that $t \in \mathrm{poly}(\kappa)$, hence Lemma 17.1.1 implies that $\mathcal{B}'$ has at most a negligible advantage α of $\mathcal{B}$; as $\mathcal{B}$ has the same advantage as $\mathcal{B}'$ with overwhelming probability it follows that $\mathcal{B}'$ cannot have non-negligible advantage in distinguishing the two games. □

$\textsc{Game}_3(\kappa)$: This game is as $\textsc{Game}_2(\kappa)$, but instead of equivocating a value with a leading 0, $\textsc{Game}_3(\kappa)$ picks a random pre-image starting with a 1: Both games pick a random OTP $v \xleftarrow{\$} \{0,1\}^\kappa$, and $\textsc{Game}_2(\kappa)$ uses $(0\|v)$ as initial input, whereas $\textsc{Game}_3(\kappa)$ uses $(1\|v)$ as input to the QRO before re-writing.

Lemma 17.3.3. *Let* $\textsc{Com}$ *be given as in Figs. 17.1 and 17.2. Let* $\mathcal{B}'$ *be a QPT distinguisher which distinguishes* $\textsc{Game}_2(\kappa)$ *and* $\textsc{Game}_3(\kappa)$ *with advantage* α. *Then* $\alpha \in \mathrm{negl}(\kappa)$.

Proof. Indistinguishability trivially holds against any QPT distinguisher $\mathcal{B}'$. We simulate the Quantum Random Oracle using the compressed oracle technique which is indistinguishable from a real random oracle. This implies that for each possible input $w \in \{0,1\}^{\kappa+1}$, the output $R_Q(w) \in \{0,1\}^{\mathrm{poly}(\kappa)}$ is uniformly distributed. Before the distinguisher obtains access to the Quantum Random Oracle, one of the uniformly sampled values from the image domain is replaced by a *different* uniformly sampled value, and the distinguisher $\mathcal{B}'$ has to differentiate whether this rewriting occurred on a pre-image starting with a zero or a one. This is trivially impossible to detect better than by guessing in a real Quantum Random Oracle, and the compressed oracle technique ensures that fixing a uniformly random value in advance cannot be detected either. With the pre-image being undetectable it also follows that the first bit of the pre-image where the rewriting occurred is undetectable. It thus follows that $\alpha = 0$ which proves our claim. □

$\textsc{Game}_4(\kappa)$: This game is as $\textsc{Game}_3(\kappa)$, but directly returns the output of R_Q on input $1\|v$. This resembles an honest commitment on 1.

Lemma 17.3.4. *Let* $\textsc{Com}$ *be given as in Figs. 17.1 and 17.2. Let* $\mathcal{B}'$ *be a QPT distinguisher which distinguishes* $\textsc{Game}_3(\kappa)$ *and* $\textsc{Game}_4(\kappa)$ *with advantage* α*. By Lemma 17.1.1, it follows that* $\alpha \in \mathrm{negl}(\kappa)$*.*

Proof. This proof is analogous to the indistinguishability of $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$, which was proven in Lemma 17.3.2 only with a flipped control bit such that a compressed oracle is invoked on oracle queries with a leading 0 and R_Q outputs are only forwarded to $\mathcal{B}'$ on inputs starting with 1. □

It follows that the two distributions for $\textsc{Com}.\mathrm{Com}(0)$ and for $\textsc{Com}.\mathrm{Com}(1)$ are not distinguishable by any QPT adversary, and thus the quantum-computational hiding property follows. From Lemmas 17.3.2 to 17.3.4, alongside the fact that $\textsc{Game}_1(\kappa)$ is exactly $\textsc{Com}.\mathrm{Com}(0)$ and $\textsc{Game}_4(\kappa)$ corresponds to $\textsc{Game}_4(\kappa)$, we can conclude that Lemma 17.3.5 follows (restated for convenience):

Lemma 17.3.5 (Quantum-Computational Hiding Property)**.** *Let* R_Q *be a Quantum Random Oracle* $R_Q\colon \{0,1\}^{\kappa+1} \longrightarrow \{0,1\}^{\mathrm{poly}(\kappa)}$ *as used for* $\textsc{Com}$*. No QPT adversary* $\mathcal{B}$ *can break the hiding property of* $\textsc{Com}$ *from Figs. 17.1 and 17.2 with non-negligible advantage* α *over guessing.*

18. Proof of Security of Π_{QCom}

In this section we prove the *everlasting hiding* and *unconditional binding* properties of the quantum bit commitment protocol Π_{QCom} from Figs. 16.3 and 16.4 when instantiating it with the classical commitment protocol in the QROM from Figs. 17.1 and 17.2. Therefore, we show that the the binding property holds against any unbounded adversary $\mathcal{A}$ and the hiding property holds against any everlasting adversary $(\mathcal{B}, \mathcal{A})$.

First, we investigate the everlasting hiding property in the QROM with quantum decay. To that end we start by showing that the bases chosen by the receiver are *independent* of the bases chosen by the sender, despite the classical commitment on the chosen basis vector.

Second, we prove the unconditional binding property by analyzing the two degrees of freedom within the unveil message susceptible by the adversary: The committed value b and the codeword $\mathcal{c}$. The other degrees of freedom—namely the basis vectors—are fixed by the unconditionally binding property of the classical commitment scheme. The sender may attempt to unveil a different codeword $\tilde{\mathcal{c}}$ than initially hidden in the quantum state. However, since the receiver measures the quantum state using uniformly random bases we can show that the probability that the receiver's codeword $\mathcal{c}'$ is sufficiently close to pass the tests in the unveil function is negligible, and that $\tilde{\mathcal{c}}$ will be rejected. This results in an unconditional binding property.

18.1. Sender Security

We start by presenting our game based proof that any measurement performed by an adversary is independent of value in the classical commitment. Then, without loss of generality, we assume that $\mathcal{B}$ performs the *best* measurement, that is, the measurement from which $\mathcal{A}$ can derive the largest amount of information.

Recall from Lemmas 13.2.9 and 13.2.10 that any measurement bases $\vec{b}'$ used by the receiver $\mathcal{B}$ result in at least $0.11\mathcal{N}$ incorrect bits. This results in an exponentially large number of possible codewords with the respective distance to the measured $\mathcal{c}'$, without further information helping the adversary to differentiate the correct codeword from incorrect ones. We can then apply Lemma 13.2.11 to argue that the amount of information available to even an unbounded adversary is negligible. It then follows from Lemma 13.2.8 that the adversary has only a negligible advantage in breaking the hiding property.

```
$\textsc{Game}_1(\kappa)$

$\mathsf{S}(\vec{G}, b)$
    $otp \xleftarrow{\$} \{0,1\}^{\mathcal{N}}$, $\vec{b} \xleftarrow{\$} \{+,\times\}^{\mathcal{N}}$
    $c_b \sim \mathscr{C}_{\vec{G}}$ s.t. $\langle c_b, otp \rangle = b$
    $|\psi\rangle \leftarrow \mathrm{encode}_{\vec{b}}(c)$
    foreach $i \in [\mathcal{N}]$ do
        $v[i] \xleftarrow{\$} \{0,1\}^{\kappa}$
        $-$
        $\mathtt{com}_{\vec{b}[i]} := R_Q(\vec{b}[i] \| v[i])$
    done
     Send $(|\psi\rangle, otp, \mathtt{com}_{\vec{b}})$ to $\mathcal{B}$
$\mathcal{B}(|\psi\rangle, otp, \mathtt{com}_{\vec{b}})$ :
    $c' \leftarrow Measure_{\vec{b}'}(|\psi\rangle)$
$R_Q(\vec{b})$ :
    $-$
    $-$
```

```
$\textsc{Game}_2(\kappa)$

$\mathsf{S}(\vec{G}, b)$ :
    $otp \xleftarrow{\$} \{0,1\}^{\mathcal{N}}$, $\vec{b} \xleftarrow{\$} \{+,\times\}^{\mathcal{N}}$
    $c_b \sim \mathscr{C}_{\vec{G}}$ s.t. $\langle c_b, otp \rangle = b$
    $|\psi\rangle \leftarrow \mathrm{encode}_{\vec{b}}(c)$
    foreach $i \in [\mathcal{N}]$ do
    $-$
    $r_i \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(\kappa)}$
    $\mathtt{com}_{\vec{b}[i]} := r_i$
    done
     Send $(|\psi\rangle, otp, \mathtt{com}_{\vec{b}})$ to $\mathcal{B}$
$\mathcal{B}(|\psi\rangle, otp, \mathtt{com}_{\vec{b}})$ :
    $c' \leftarrow Measure_{\vec{b}'}(|\psi\rangle)$
$R_Q(\vec{b})$ :
    $\forall_{i \in [\mathcal{N}]} v[i] \xleftarrow{\$} \{0,1\}^{\kappa}$
    $\forall_{i \in [\mathcal{N}]} R_Q(\vec{b}[i] \| v[i]) := r_i$
```

Figure 18.1.: Games to show independence of chosen basis vectors.

So before we can use Lemma 13.2.10 to limit the amount of information the adversary can extract from the commitment, we have to show that our commitment scheme fulfills the requirements stated by that lemma. Recall that their lemma was made with respect to a protocol where *no classical commitment* of the basis vectors is sent to the receiver; since our protocol contains a classical commitment on the basis vector we need to prove that the commitment yields no information that could help the adversary, which we show using the two game-hops depicted in Fig. 18.1. More formally, the games are defined as follows:

$\textsc{Game}_1(\kappa)$: This game follows the original commitment protocol from Fig. 16.3, where the sender commits to a given and fixed bit b.

$\textsc{Game}_2(\kappa)$: This game is as $\textsc{Game}_1(\kappa)$, only that instead of providing *actual* commitments $\mathtt{com}_{\vec{b}[i]}$ on the basis vectors, the reported commitments are drawn as uniformly random $\mathrm{poly}(\kappa)$-bit strings drawn independently of the basis vector used.

Lemma 18.1.1. *If an everlasting distinguisher $(\mathcal{B}', \mathcal{A}')$ can distinguish $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$ from Fig. 18.1 with non-negligible advantage α, then there exists an everlasting adversary $(\mathcal{B}, \mathcal{A})$ that can distinguish which random oracle was used in Lemma 17.1.1 with the same advantage α.*

Proof. We want to reduce a distinguisher $(\mathcal{B}', \mathcal{A}')$ of Section 18.1 and $\text{Game}_2(\kappa)$ to an adversary $(\mathcal{B}, \mathcal{A})$ on Lemma 17.1.1, who is provided with challenges C_i by the challenger C_{PreImg}.

The naïve approach for reduction adversary $(\mathcal{B}, \mathcal{A})$ would be to forward the challenges C_i as commitments $\mathtt{com}_{\vec{b}[i]}$ to the adversary $\mathcal{B}'$. Then, any oracle query from the adversary $\mathcal{B}'$ may be send to the oracle maintained by the challenger C_{PreImg} and respectively the oracle output back to the adversary. However, there might be a unknown dependency (for example hidden in a superposition query) between the preimage of the values sent to the oracle and the quantum state $|\psi\rangle$ resulting from the bases that the preimages define. All outputs of the simulated oracle have to reflect such dependencies, which means that the reduction algorithm has to ensure that all oracle queries in correspondence with C_i are mapped to the initial quantum state, and all other queries are answered consistently. The parties interact with a total of three oracles:

Challenge oracle $R_Q\colon \{0,1\}^\kappa \longrightarrow \{0,1\}^{\text{poly}(\kappa)}$. The oracle is provided by C_{PreImg} for the reduction adversary.

Compressed oracle $R_C\colon \{0,1\}^\kappa \longrightarrow \{0,1\}^{\text{poly}(\kappa)}$. The oracle is simulated by the reduction algorithm using the approach described by Zhandry [142].

Simulated oracle $R_S\colon \{0,1\}^{\kappa+1} \longrightarrow \{0,1\}^{\text{poly}(\kappa)}$, which is provided by the reduction algorithm to the distinguisher and is used for the classical commitment component. The oracle maps inputs to either the output of the challenge oracle R_Q, or to the output of a compressed oracle R_C.

We assume that the each oracle O can be queried in superposition through a circuit that works on a state $|\varphi_O\rangle := (|\text{dom}(O)\rangle, |\text{cod}(O)\rangle)$ where $|\text{dom}(O)\rangle$ is the input register and $|\text{cod}(O)\rangle$ is the output register. The circuit maps $(|\text{dom}(O)\rangle, |\text{cod}(O)\rangle)$ to $(|\text{dom}(O)\rangle, |\text{cod}(O) \oplus O(\text{dom}(O))\rangle)$; by assuming an initially empty output register we have the output $|O(\text{dom}(O))\rangle$ stored directly inside the output register. Note here that the circuit does not change the input register $|\text{dom}(O)\rangle$.

Challenge. The reduction algorithm starts by drawing a random $b \overset{\$}{\leftarrow} \{0,1\}$ and computes the codeword c and nonce otp honestly. For the creation of the quantum state $|\psi\rangle$ the adversary draws uniformly random basis vectors $\vec{b} \in \{+, \times\}^{\mathcal{N}}$ and encodes c as $|\psi\rangle$ in those bases. Instead of computing honest commitments $\mathtt{com}_{\vec{b}}$, $\mathcal{B}$ embeds the challenge $\{C_i\}_{i=1}^{\mathcal{N}}$ from C_{PreImg}. Thus the first message that $\mathcal{B}$ reports to $\mathcal{B}'$ in the name of the sender is given as $(|\psi\rangle, otp, \{C_i\}_{i=1}^{\mathcal{N}})$.

Simulated Oracle Behaviour. The reduction algorithm has to provide a view in which the pre-images of the challenges are valid commitments on the basis vectors $\vec{b}$ without knowing the actual pre-image values. With knowledge of the respective image one can compare the oracle output with the challenges and either return the challenges or a value provided by the compressed oracle. Therefore, we define the simulation oracle to provide an extra bit (compared to the challenge oracle). Given this extra bit, we can split the oracle results for values representing pre-image challenges and other values, depending on the basis. In particular the simulated oracle should behave as depicted in Fig. 18.2.

To that end, note that in our construction the classical commitment scheme in the QROM from Figs. 17.1 and 17.2, the basis vector for the i-th qubit is determined only by the *first* bit of the pre-image from C_i. The remaining qubits of the pre-image are independent of the commitment value. We exploit this property by letting the reduction algorithm take care of the first qubit of the query:

Let $|\mathrm{dom}(R_S)\rangle$ contain the distinguishers query. The reduction algorithm copies[1] the last κ qubits of the register into the query register $|\mathrm{dom}(R_Q)\rangle$, and queries the oracle provided by the challenger with all but the first qubit:

$$|\mathrm{cod}(R_Q)\rangle|\mathrm{dom}(R_Q)\rangle[1\ldots\kappa] \leftarrow R_Q(|0\rangle|\mathrm{dom}(R_Q)\rangle[1\ldots\kappa]) \tag{18.1}$$

The same procedure is performed for the compressed oracle:

$$|\mathrm{cod}(R_C)\rangle|\mathrm{dom}(R_Q)\rangle \leftarrow R_C(|0\rangle|\mathrm{dom}(R_C)\rangle[1\ldots\kappa]) \tag{18.2}$$

To ensure that the first bit of the pre-image to any of the reported C_i values matches the basis $\vec{b}[i]$ used for the creation of $|\psi\rangle$ the values $\{C_i\}_{i=1}^{\mathcal{N}}$ and (the single-bit encoding of) $\vec{b}[i]$ are encoded into quantum registers (using computational bases), the QPT circuit works on registers containing the above alongside the original query register $|\mathrm{dom}(R_S)\rangle$ which enforces the behavior depicted in Fig. 18.2.

This allows to select which of the two oracle outputs are used as result for the simulated query. The result is then written into the register $|\mathrm{cod}(R_S)\rangle$ containing the output of the query for the simulated oracle. The quantum state $(|\mathrm{dom}(R_S)\rangle, |\mathrm{cod}(R_S)\rangle)$ is then returned to $\mathcal{B}'$ as response to his query. The circuit ensures that R_S behaves like a valid random oracle in that the queries resulting in any output C_i look like valid commitments for $\vec{b}[i]$ and the same input throughout different queries yields the same output.

A detailed description of the quantum circuit implementing this behavior is given at the end of the section.

Oracle Simulation. Let t be the number of oracle queries that the QPT part of the distinguisher, $\mathcal{B}'$, performs on the simulated oracle R_S. $\mathcal{B}$ applies the simulation circuit for all t queries performed by the adversary $\mathcal{B}'$. After termination of $\mathcal{B}'$, the QPT-part of the reduction algorithm, $\mathcal{B}$, measures the database D for input-output tuples seen by $\mathcal{B}'$, and then terminates by sending those values to the classically

[1] For the sake of simplicity we refer to the cloning of non-independent states using CNOT gates as copying.

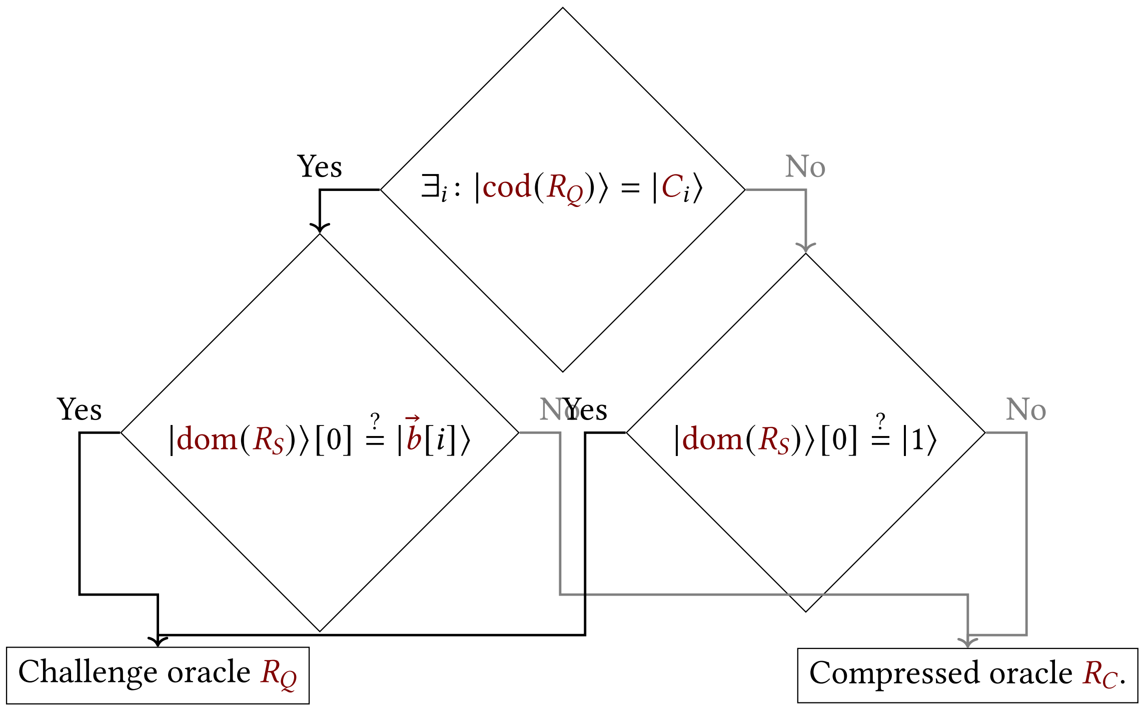
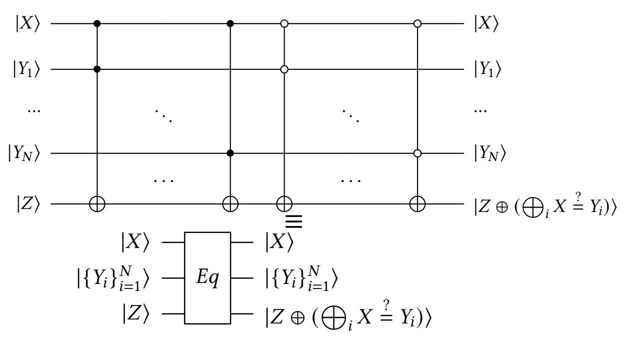

Yes
$\exists_i : |\mathrm{cod}(R_Q)\rangle = |C_i\rangle$
No
Yes
$|\mathrm{dom}(R_S)\rangle[0] \overset{?}{=} |\vec{b}[i]\rangle$
No
Yes
$|\mathrm{dom}(R_S)\rangle[0] \overset{?}{=} |1\rangle$
No
Challenge oracle R_Q
Compressed oracle R_C.

Figure 18.2.: Depiction of the oracle selection process. $|\mathrm{cod}(R_Q)\rangle$ contains the output of the challenge oracle R_Q. $|\mathrm{dom}(R_S)\rangle$ contains the query sent to the simulated oracle R_S.

unlimited part of the reduction, $\mathcal{A}$. From C_{PreImg}, $\mathcal{A}$ obtains the value table of the challenge oracle R_Q. By definition this value table is guaranteed to have pre-images for all $\mathcal{N}$ challenges C_i. Using that alongside the value table of the compressed oracle R_C, the reduction algorithm prepares the value table that is to be handed over to the classically unbounded distinguishing adversary $\mathcal{A}'$: $\mathcal{A}$ goes through every value $x \in \{0,1\}^\kappa$ of the value table for R_Q, and if $R_Q(x)$ is none of the C_i, it adds to its own value table two entries $(0\|x) \mapsto R_Q(x)$ and $(1\|x) \mapsto R_C(x)$. If there is some i such that $R_Q(x) = C_i$, then $\mathcal{A}$ adds two entries $(\vec{b}[i]\|x) \mapsto R_Q(x)(= C_i)$ and $((1 - \vec{b}[i])\|x) \mapsto R_C(x)$. This lets $\mathcal{A}$ effectively execute the circuit from Fig. 18.2 *classically* for each possible input and provide the output that is expected to be the same output that was seen by the distinguisher earlier. We stress here that this value table is consistent up to the factor that the pre-images of the challenges $\{C_i\}_{i=1}^{\mathcal{N}}$ might have resulted in different outcomes, depending on whether the value table is exactly that of R_Q or if k values differ.

Yet the view of the distinguishing algorithms $(\mathcal{B}', \mathcal{A}')$ can be perfectly simulated: If the value table was not changed by C_{PreImg} then the challenges $\{C_i\}_{i=1}^{\mathcal{N}}$ were valid commitments on the basis vectors $\vec{b}$ used for encoding the code word c into the quantum state $|\psi\rangle$, as is the case in $\textsc{Game}_1(\kappa)$. Note also that with overwhelming



$|X\rangle$
$|Y_1\rangle$
$\cdots$
$|Y_N\rangle$
$|Z\rangle$
$|X\rangle$
$|Y_1\rangle$
$\cdots$
$|Y_N\rangle$
$|Z \oplus (\bigoplus_i X \stackrel{?}{=} Y_i)\rangle$
$\equiv$
$|X\rangle$
$|\{Y_i\}_{i=1}^N\rangle$
$|Z\rangle$
Eq
$|X\rangle$
$|\{Y_i\}_{i=1}^N\rangle$
$|Z \oplus (\bigoplus_i X \stackrel{?}{=} Y_i)\rangle$

Figure 18.3.: Quantum circuit to compare inputs.

probability[2], none of the C_i are in the image domain of the compressed oracle R_C. Yet if C_{PreImg} changed values of the value table then the random oracle then the challenge is entirely independent of the basis vectors used for creating the quantum state $|\psi\rangle$, which resembles the case of $\text{GAME}_2(\kappa)$.

So if the distinguisher $(\mathcal{B}', \mathcal{A}')$ can correctly distinguish with non-negligible advantage α then the reduction $(\mathcal{B}, \mathcal{A})$ inherits this advantage and can differentiate in Lemma 17.1.1 with advantage α.

Hence, an answer of the distinguisher $(\mathcal{B}', \mathcal{A}')$ can be translated to a response the reduction algorithm $(\mathcal{B}, \mathcal{A})$ sends to the challenger C_{PreImg}.

Due to Lemma 17.1.1 the advantage α of $(\mathcal{B}, \mathcal{A})$ is negligible and thus no distinguisher $(\mathcal{B}', \mathcal{A}')$ with non-negligible advantage exists.

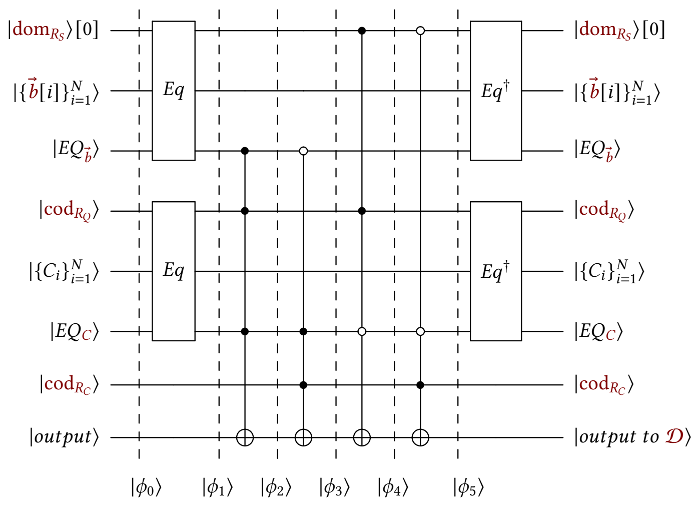
Constructing the Circuit. To conclude our prove we need to show that the circuit deployed by $\mathcal{B}$ that decides which oracle output is returned to the distinguisher $\mathcal{B}'$ actually exists. The circuit takes as inputs the challenges $\{C_i\}_{i=1}^{\mathcal{N}}$, the output $|\text{cod}(R_C)\rangle$ of the compressed oracle, the oracle query $|\text{dom}(R_S)\rangle$ from $\mathcal{B}'$, the set of all basis vectors $|\{\vec{b}[i]\}_{i=1}^{\mathcal{N}}\rangle$ used in the oracle and the output $|\text{cod}(R_Q)\rangle$ of the oracle provided by C_{PreImg}.

It compares each value in register $|\text{cod}(R_Q)\rangle$ to the challenges C_i and outputs either the image of the random oracle (if the basis has been chosen for a commitment), or otherwise the image of the compressed oracle.

Fig. 18.4 shows the respective quantum circuit which deploys the subroutine in Fig. 18.3 to compare the different values.

In the following we review the transformations performed in the circuit in more detail. We consider the steps from the initial state $|\phi_0\rangle$ to the final state $|\phi_5\rangle$ (without

[2] Due to the exponentially larger span of R_Q compared to its input domain



$|\mathrm{dom}_{R_S}\rangle[0]$
$|\{\vec{b}[i]\}_{i=1}^{N}\rangle$
$|EQ_{\vec{b}}\rangle$
$|\mathrm{cod}_{R_Q}\rangle$
$|\{C_i\}_{i=1}^{N}\rangle$
$|EQ_C\rangle$
$|\mathrm{cod}_{R_C}\rangle$
$|output\rangle$
Eq
Eq
$Eq^\dagger$
$Eq^\dagger$
$|output\ to\ \mathcal{D}\rangle$
$|\phi_0\rangle$
$|\phi_1\rangle$
$|\phi_2\rangle$
$|\phi_3\rangle$
$|\phi_4\rangle$
$|\phi_5\rangle$

Figure 18.4.: Circuit-based description.

uncomputation). However, since the input registers $\{C_i\}_{i=1}^{\mathcal{N}}$, $|\mathrm{cod}(R_C)\rangle$, $|\mathrm{dom}(R_S)\rangle$, $|\{\vec{b}[i]\}_{i=1}^{\mathcal{N}}\rangle$ are not changed after circuit execution, we only consider the initial state:

$$|\phi_0\rangle = |\mathrm{dom}(R_S)[0]\rangle|EQ_{\vec{b}} := 0^{\mathcal{N}}\rangle|EQ_C := 0^{\mathcal{N}}\rangle|output := 0\rangle$$

Note that at most one of the oracle outputs $|\{\vec{b}[i]\}_{i=1}^{\mathcal{N}}\rangle$ can match the value in $|\mathrm{dom}(R_S)\rangle$, and the same applies to $|\mathrm{cod}(R_C)\rangle$ and the values $\{C_i\}_{i=1}^{\mathcal{N}}$, resulting in the state:

$$|\phi_1\rangle = |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i \overset{?}{=} \mathrm{cod}(R_Q)\rangle|0\rangle$$

In each subsequent step one of the oracle outputs, either from the challenge oracle or the compressed oracle, are written into the respective subspace of the output register. For the sake of simplicity we omit variables to denote the unknown amplitudes of each sub-space, since the exact values are not relevant to provide a consistent view.

$$\begin{aligned}
|\phi_2\rangle = {} & |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] = \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle \\
& + |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i \overset{?}{=} \mathrm{cod}(R_Q)\rangle|0\rangle \\
|\phi_3\rangle = {} & |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] = \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle \\
& + |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \neq \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_C)\rangle \\
& + |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)\rangle|\nexists_i : C_i = \mathrm{cod}(R_Q)\rangle|0\rangle \\
|\phi_4\rangle = {} & |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] = \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle \\
& + |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \neq \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_C)\rangle \\
& + |\mathrm{dom}(R_S)[0] \neq 0\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)\rangle|\nexists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle \\
& + |\mathrm{dom}(R_S)[0] = 0\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)\rangle|\nexists_i : C_i = \mathrm{cod}(R_Q)\rangle|0\rangle
\end{aligned}$$

$$\begin{aligned}|\phi_5\rangle = &|\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] = \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle\\ &+ |\mathrm{dom}(R_S)[0]\rangle|\vec{b}[i] \neq \mathrm{dom}(R_S)[0]\rangle|\exists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_C)\rangle\\ &+ |\mathrm{dom}(R_S)[0] \neq 0\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)\rangle|\nexists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_Q)\rangle\\ &+ |\mathrm{dom}(R_S)[0] = 0\rangle|\vec{b}[i] \overset{?}{=} \mathrm{dom}(R_S)\rangle|\nexists_i : C_i = \mathrm{cod}(R_Q)\rangle|\mathrm{cod}(R_C)\rangle\end{aligned}$$

□

Note that in $\text{GAME}_2(\kappa)$, all values that the receiver obtains are *independent* of the used basis vectors:

Corollary 18.1.2. *When instantiating the classical commitment scheme from Π_{QCom} with the commitment protocol from Figs. 17.1 and 17.2, then the bases chosen by the receiver are* independent *from the bases chosen by the sender. Moreover, there is no operation that the QPT-receiver can do that depends on the real bases $\vec{b}$ chosen by the committer.*

Thus, we can directly apply Lemma 13.2.9 to deduce that the best measurement of a malicious receiver is in the *Breidbard*-bases, which yields at least $0.11\mathcal{N}$ wrong bits. And due to Lemma 13.2.11 there are exponentially many *possible* code words given any measurement in those best bases, which due to Lemma 13.2.8 results in a situation where the adversary has only a negligible advantage in breaking the hiding property.

Thus, our conclusion holds despite the fact that the correct basisvectors are contained as a classical commitment:

Corollary 18.1.3. *The commitment protocol from Figs. 16.3 and 16.4 is an* everlasting *hiding commitment scheme.*

18.2. Receiver Security

Next, we will consider the *binding* properties of our protocol which is formalized in Lemma 18.2.1:

Lemma 18.2.1 (Unconditional Binding)**.** *When instantiating the classical component of Π_{QCom} from Figs. 16.3 and 16.4 with any unconditionally binding commitment scheme then Π_{QCom}* is unconditionally binding*: Any unbounded adversary $\mathcal{A}$ has only a negligible chance to open this commitment to $\bar{b}$ instead of b, that is, the following probability is negligible in the security parameter κ:*

$$\left| \Pr\left[\text{COM}.\mathrm{Vfy}(\mathrm{com}, \mathrm{unv}_b, b) = 1 \;\middle|\; \begin{array}{l} (st, \mathrm{com}) \leftarrow \mathcal{A}(1^\kappa), \\ b \xleftarrow{\$} \{0,1\}, \\ \mathrm{unv}_b \leftarrow \mathcal{A}(st, b) \end{array} \right] - 1/2 \right| \tag{18.3}$$

where the probability is taken over the random coins.

Proof. The unveil information consists of the codeword $\mathcal{c}$, the basis $\vec{b}$ and the committed value b. First we recall that the classical commitment COM is unconditionally binding, such that not even an unbounded adversary can unveil different bases $\vec{b}$.

Therefore, in order to unveil a different value $\bar{b}$, only the codeword $\mathcal{c}^*$ may be changed: First, note that our code was constructed to have $\mathcal{D} > \mathrm{H}^{-1}(1/2)\mathcal{N} \approx 0.1100279\mathcal{N}$, so two valid codewords differ in at least $0.11\mathcal{N}$ bits. In order to break the binding property, $\mathcal{A}$ has to change at least $\frac{\mathcal{D}}{2} = 0.055\mathcal{N}$ positions in the codeword $\mathcal{c}^*$; this is due to the fact that the adversary can commit to an *invalid* codeword, where half of the difference is taken from a codeword that commits to 0 and half from a codeword that commits to 1, and then later tries to equivocate the half that was taken from the codeword that commits to the respective other bit.

This change can only be induced in those positions where the sender committed to a different vector $\vec{b}[i]$ than the $\vec{b}'[i]$ that the receiver chose, since matching bases would let the receiver detect the change.

However, the $\mathcal{A}$ does not know the positions where the bases match. There are at least $\binom{\mathcal{N}}{0.055\mathcal{N}}$ combinations, thus exponentially many possibilities for the positions. Therefore, even an unbounded adversary has only negligible chance of guessing the correct combination and thus opening the commitment to a different value.

□

19. An Obfuscated Measurement Attack

We have seen in Chapters 17 and 18 that the commitment scheme in the QROM yields a secure protocol Π_{QCom} from Figs. 16.3 and 16.4. In this section we want to demonstrate the significance of the QROM for the classical commitment scheme by presenting an alternative construction in the plain model for the classical commitment scheme and subsequently describe an attack where the adversary's measurement depends on the classical value. We show that an adversary can break the hiding property in our toy example.

It is therefore conceivable that additional *new* properties are required to achieve everlasting security for (quantum) bit commitments which were fulfilled by the QROM jointly with Quantum Decay, but not by our latter instantiation.

19.1. The High-level Idea

We present an attack on the hiding property of our quantum bit commitment scheme where the classical component of Π_{QCom} is instantiated with an unconditionally binding and computationally hiding commitment scheme $\textsc{Com}^*$ which does not rely on the QROM.

First we construct such a commitment scheme, then we show how it can be used by $\mathcal{B}$ to perform operations which are dependent on the value hidden by the commitment.

Perhaps surprisingly, the operations performed by $\mathcal{B}$ are insufficient for any QPT adversary to break the hiding property; instead, they prepare the commitment in such a way, that the unbounded adversary $\mathcal{A}$ can use them to break the hiding property, which is not possible without the preprocessing step performed by $\mathcal{B}$. In our case, the actions performed by $\mathcal{B}$ transform the Wiesner-encoded quantum state $|\psi\rangle$ into a One Time Pad encrypted quantum state $|\phi\rangle$ where the keys of the OTP are only computationally hidden. This suggests that that we cannot solely rely only on the computational hiding property of the classical commitment.

Our attack is based on a Quantum Fully Homomorphic Encryption Tuple (QFHET) $(\textsc{QFhe}, \textsc{Fhe})$ (*cf.* Definition 13.2.16). Furthermore, we assume that the QPT adversary $\mathcal{B}$ has access to valid *encryptions* ct_1 under $\textsc{Fhe}$ of bits $cb = [\![\vec{b} = \times]\!]$ indicating which bases were used; we will show later why this assumption is justified. Without loss of generality, we assume that $cb = 1$ iff the qubit was encoded in bases $\times$ and $cb = 0$ indicates an encoding in bases $+$.

We show that while the information on cb provided by Com* does not suffice for $\mathcal{B}$ to extract cb, it does suffice to *homomorphically transform* the Wiesner encoding of the quantum state the sender prepared into a QOTP ciphertext of the same value. The QFHE evaluation then additionally yields classical encryptions of the QOTP keys.

Unlike Wiesner's encoding, a QOTP-encrypted quantum state does not irreversibly destroy information upon measurement in the computational basis. A phase-flip (Z) does not change the measurement outcome at all, and a negation (X) can be performed classically on the measurement outcome. Hence, the QPT adversary can then measure in the computational basis to obtain both a (classical) OTP-encrypted codeword $\hat{c}$ and an encryption of the corresponding OTP.

The latter can be extracted by the unbounded adversary $\mathcal{A}$ and then applied on the former to obtain the original codeword c used by the sender. Alongside the one-time pad otp sent by the sender during the commitment phase this suffices to reconstruct the committed bit b.

We first provide a formal description of the attack.

Definition 19.1.1 (Obfuscated Measurement Attack)**.** *In the everlasting setting with Quantum Decay, let* Com *be a classical commitment scheme with unconditional binding and computational hiding properties. Let Π_{QCom} be given as in Figs. 16.3 and 16.4. An Obfuscated Measurement Attack (OMA) breaks the everlasting hiding property of Π_{QCom}. It contains two algorithms $\mathcal{B}$ and $\mathcal{A}$:*

$\mathcal{B}$ *is a QPT algorithm transforming the quantum state $|\psi\rangle$ obtained from the sender during Π_{QCom} from Wiesner bases into a state that quantum computationally hides the same information in* computational *bases* $+$.

$\mathcal{A}$ *is an unbounded classical algorithm, which extracts the secret hidden in the quantum state from the output of $\mathcal{B}$.*

19.2. A Partially Homomorphic Commitment Scheme

We now present a classical commitment scheme and prove its unconditional binding and computational hiding properties. The commitment contains additional values, namely valid ciphertexts under Fhe, which we will use for the OMA.

Our construction combines an existing unconditionally binding and computationally hiding commitment scheme Com$'$ with a Fully Homomorphic Encryption scheme Fhe which additionally encrypts the bit under a fresh key.

Using Com$'$ and Fhe we construct a new commitment scheme Com* which is given in Figs. 19.1 and 19.2. The left protocol shows the commit-stage which first creates a key pair (sk, pk) according to Fhe.KeyGen and uses Com$'$ to commit to both the actual bit b and

Protocol Π_{Com}

The classical commitment protocol Π_{Com} for committing to a given bit $b \in \{0,1\}$ in the standard model. It is running with a set of 2 parties (C, R) where C is the committer and R is the receiver.
It is parameterized by a Quantum Fully Homomorphic Encryption Tuple $(\textsc{QFhe}, \textsc{Fhe})$ and an unconditionally binding and computationally hiding commitment scheme $\textsc{Com}' = (\mathrm{Com}, \mathrm{Unv})$.

On input b, C samples keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathrm{KeyGen}(1^\kappa)$. Then C commits to b by computing $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathrm{Com}(b)$ and to the secret key as $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathrm{Com}(\mathsf{sk})$. Finally, C computes $ct_1 \leftarrow \textsc{Fhe}.\mathrm{Enc}_{\mathsf{pk}}(b)$ and $ct_2 \leftarrow \textsc{Fhe}.\mathrm{Enc}_{\mathsf{pk}}((0,0))$ and sends $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$ to R.

Figure 19.1.: The classical commitment protocol Π_{Com} in the standard model for committing to a given bit $b \in \{0,1\}$.

Protocol Π_{Unv}

The classical unveil protocol Π_{Unv} for unveiling a commitment in the standard model.

Upon activation, C sends $\mathtt{unv}_b := (b, \mathtt{unv}'_b, \mathsf{sk}, \mathtt{unv}'_{\mathsf{sk}})$ to R.

On input $\left(\mathtt{unv}_b := (b, \mathtt{unv}'_b, \mathsf{sk}, \mathtt{unv}'_{\mathsf{sk}})\right)$ from C, R rejects if any of the following conditions does not hold: (1) $\textsc{Com}'.\mathrm{Unv}(\mathtt{com}'_b, \mathtt{unv}'_b, b)$, (2) $\textsc{Com}'.\mathrm{Unv}(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}, \mathsf{sk})$, (3) $\textsc{Fhe}.\mathrm{Dec}_{\mathsf{sk}}(ct_1) = b$, and (4) $\textsc{Fhe}.\mathrm{Dec}_{\mathsf{sk}}(ct_2) = (0,0)$. Otherwise, R accepts.

Figure 19.2.: The classical unveil protocol Π_{Unv} in the standard model for unveiling a commitment.

the secret key sk. It also sends encryptions of the message b and of a zero-vector $(0,0)$ to the receiver.

To unveil a given commitment, the sender sends the message b and the secret key sk of $\textsc{Fhe}$ alongside their corresponding unveil information from $\textsc{Com}'$ to the receiver. The verification step lets the receiver check that the unveil information of $\textsc{Com}'$ are valid and that the secret key decrypts the cipher texts accordingly.

```
$\textsc{Game}_1(\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$
    $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathsf{Com}(0)$
    $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\mathsf{sk})$
    $(ct_1) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, 0)$
    $(ct_2) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, (0, 0))$
    return $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$
```

```
$\textsc{Game}_2(\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$
    $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathsf{Com}(0)$
    $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\vec{0})$
    $(ct_1) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, 0)$
    $(ct_2) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, (0, 0))$
    return $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$
```

```
$\textsc{Game}_3(\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$
    $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathsf{Com}(0)$
    $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\vec{0})$
    $(ct_1) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, 1)$
    $(ct_2) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, (0, 0))$
    return $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$
```

```
$\textsc{Game}_4(\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$
    $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathsf{Com}(1)$
    $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\vec{0})$
    $(ct_1) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, 1)$
    $(ct_2) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, (0, 0))$
    return $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$
```

```
$\textsc{Game}_5(\kappa)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$
    $(\mathtt{com}'_b, \mathtt{unv}'_b) \leftarrow \textsc{Com}'.\mathsf{Com}(1)$
    $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\mathsf{sk})$
    $(ct_1) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, 1)$
    $(ct_2) \leftarrow \textsc{Fhe}.\mathsf{Enc}(\mathsf{pk}, (0, 0))$
    return $(\mathtt{com}'_b, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$
```

Figure 19.3.: Hiding games for the commitment $\textsc{Com}^*$.

19.2.1. Sender Security

We start by investigating the quantum-computational hiding property. To that end, we introduce five games in Fig. 19.3 which jump from a valid commitment on $b = 0$ to a valid commitment on $b = 1$.

More formally, we have the following games:

$\textsc{Game}_1(\kappa)$: This game follows the original protocol on $\textsc{Com}^*.\mathsf{Com}(0)$.

$\textsc{Game}_2(\kappa)$: This game is as $\textsc{Game}_1(\kappa)$, but instead of an honest commitment using $\textsc{Com}'$ on sk this game reports $(\mathtt{com}'_{\mathsf{sk}}, \mathtt{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\vec{0})$ as zero-commitment of appropriate length.

Lemma 19.2.1 (Indistinguishability of $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$)**.** *If $\textsc{Fhe}$ is a q-IND-CPA-secure encryption scheme, then any QPT-distinguisher $\mathcal{B}$ can distinguish $\textsc{Game}_1(\kappa)$ from $\textsc{Game}_2(\kappa)$ only with negligible advantage $\alpha \in \mathsf{negl}(\kappa)$.*

Proof. Assume for the sake of contradiction that a QPT distinguisher $\mathcal{B}'$ can distinguish the two games with probability $1/2 + \alpha$ with non-negligible advantage α. Then there exists an adversary $\mathcal{B}$ on the (computational) hiding property of $\textsc{Com}'$ with advantage α.

The reduction algorithm $\mathcal{B}$ first creates honest keys $(\mathsf{sk}, \mathsf{pk})$ for $\textsc{Fhe}$ and then queries a commitment on 0 which it uses as $\mathtt{com}_b$. The encryptions of 0 and $(0, 0)$ for ct_1 and ct_2, respectively, are also honestly created using the public key pk. The adversary now sends the two messages $\vec{0}$ and sk to the challenger of the hiding game and obtains a challenge commitment $\mathtt{com}$. This is used as commitment $\mathtt{com}'_{\mathsf{sk}}$ on the secret key.

The full message is sent to the distinguisher $\mathcal{B}'$. If the response is $\textsc{Game}_1(\kappa)$, then the reduction adversary reports a commitment on sk and if the response is $\textsc{Game}_2(\kappa)$, then the reduction adversary reports that the commitment was on $\vec{0}$.

It is easy to see that this game simulates the view for the respective games correctly and that hence $\mathcal{B}$ has the same advantage in breaking the hiding property as $\mathcal{B}'$ has in distinguishing this game hop. Thus the claim follows. □

$\textsc{Game}_3(\kappa)$: This game is as $\textsc{Game}_2(\kappa)$, but instead of $ct_1 \leftarrow \textsc{Fhe}.\mathrm{Enc}(\mathsf{pk}, 0)$ this game reports $ct_1 \leftarrow \textsc{Fhe}.\mathrm{Enc}(\mathsf{pk}, 1)$.

Lemma 19.2.2 (Indistinguishability of $\textsc{Game}_2(\kappa)$ and $\textsc{Game}_3(\kappa)$)**.** *If $\textsc{Fhe}$ is a q-IND-CPA-secure encryption scheme, then any QPT-distinguisher $\mathcal{B}$ can distinguish $\textsc{Game}_2(\kappa)$ from $\textsc{Game}_3(\kappa)$ only with negligible advantage $\alpha \in \mathrm{negl}(\kappa)$.*

Proof. Assume for the sake of contradiction that a QPT distinguisher $\mathcal{B}'$ can distinguish the two games with probability $1/2 + \alpha$ for some $\alpha \notin \mathrm{negl}(\kappa)$. This would imply an adversary $\mathcal{B}$ on the q-IND-CPA property of $\textsc{Fhe}$ with non-negligible success probability as follows: $\mathcal{B}$ computes commitments $\mathtt{com}'_0 \leftarrow \textsc{Com}'.\mathrm{Com}(0)$ and asks for the oracle provided by $C_{q-IND-CPA}$ for an encryption ct_2 on $(0, 0)$. $\mathcal{B}$ sends as challenge two messages to $C_{q-IND-CPA}$, $b_0 = 0$ and $b_1 = 1$, to obtain the challenge encryption ct_1.

$\mathcal{B}$ sends $(\mathtt{com}'_0, \mathtt{com}'_{\mathsf{sk}}, ct_1, ct_2)$ to $\mathcal{B}'$ and obtains bit 0 for $\textsc{Game}_2(\kappa)$ and 1 for $\textsc{Game}_3(\kappa)$. This is forwarded directly to $C_{q-IND-CPA}$.

$C_{q-IND-CPA}$ either encrypts $b_0 = 0$, in which case the message from $\mathcal{B}$ corresponds to $\textsc{Game}_2(\kappa)$, or $b_1 = 1$, which resembles the message from $\textsc{Game}_3(\kappa)$; so the views from the respective games can be simulated correctly.

It thus follows from the q-IND-CPA requirement of $\textsc{Fhe}$ that $\mathcal{B}'$ has only negligible advantage in distinguishing the two games.

□

$\textsc{Game}_4(\kappa)$: This game is as $\textsc{Game}_3(\kappa)$, but lets the sender send $\textsc{Com}'.\mathrm{Com}(1)$ instead of $\textsc{Com}'.\mathrm{Com}(0)$.

Lemma 19.2.3 (Indistinguishability of $\textsc{Game}_3(\kappa)$ and $\textsc{Game}_4(\kappa)$). *If $\textsc{Com}'$ is quantum-computationally hiding, then any QPT distinguisher $\mathcal{B}$ trying to distinguish $\textsc{Game}_3(\kappa)$ from $\textsc{Game}_4(\kappa)$ has only negligible advantage $\alpha \in \mathsf{negl}(\kappa)$.*

Proof. Let $\mathcal{B}'$ be a QPT distinguisher between $\textsc{Game}_3(\kappa)$ and $\textsc{Game}_4(\kappa)$. We can create an adversary $\mathcal{B}$ that uses $\mathcal{B}'$ to attack the *hiding* property of $\textsc{Com}'$.

The challenger C_{Hid} on the hiding game sends a commitment $\mathsf{com}'_0 = \textsc{Com}'.\mathsf{Com}(b)$ to the QPT adversary $\mathcal{B}$, who computes a key-pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{Fhe}.\mathsf{KeyGen}(1^\kappa)$, creates ciphertexts honestly, sends $\vec{0}$ to the commitment oracle, obtains $\mathsf{com}'_{\mathsf{sk}}$ and sends $(\mathsf{com}'_0, \mathsf{com}'_{\mathsf{sk}}, ct_1, ct_2)$ to $\mathcal{B}'$.

Both ct_1 and ct_2 were honestly created and C_{Hid} either sends a commitment on 0 or on 1, the former corresponding to $\textsc{Game}_3(\kappa)$ and the latter to $\textsc{Game}_4(\kappa)$. Hence, $\mathcal{B}$ can forward the output of $\mathcal{B}'$ to C_{Hid}, to inherit the success probability $1/2 + \alpha$. With $\textsc{Com}'$ being computationally hiding, it follows that $\alpha \in \mathsf{negl}(\kappa)$. □

$\textsc{Game}_5(\kappa)$: This game is as $\textsc{Game}_4(\kappa)$, but instead of using a zero-commitment for $(\mathsf{com}_{\mathsf{sk}}, \mathsf{unv}_{\mathsf{sk}})'$ this game creates a valid commitment $(\mathsf{com}'_{\mathsf{sk}}, \mathsf{unv}'_{\mathsf{sk}}) \leftarrow \textsc{Com}'.\mathsf{Com}(\mathsf{sk})$. This game corresponds to an honest commitment $\textsc{Com}^*.\mathsf{Com}(1)$.

Lemma 19.2.4 (Indistinguishability of $\textsc{Game}_4(\kappa)$ and $\textsc{Game}_5(\kappa)$). *If $\textsc{Com}'$ is quantum-computationally hiding, then any QPT distinguisher $\mathcal{B}$ trying to distinguish $\textsc{Game}_4(\kappa)$ from $\textsc{Game}_5(\kappa)$ has only negligible advantage $\alpha \in \mathsf{negl}(\kappa)$.*

Proof. This proof is equivalent to that of Lemma 19.2.1. □

We thus have shown that $\textsc{Com}^*.\mathsf{Com}(0)$ from $\textsc{Game}_1(\kappa)$ is not efficiently distinguishable from $\textsc{Com}^*.\mathsf{Com}(1)$ from $\textsc{Game}_4(\kappa)$ for any QPT adversary $\mathcal{B}$. Thus, Lemmas 19.2.1 to 19.2.4 lead to the following corollary:

Corollary 19.2.5 (Computationally hiding). *If $\textsc{Com}'$ is a* quantum-computationally hiding *commitment scheme and $\textsc{Fhe}$ is a q-IND-CPA-secure FHE scheme, $\textsc{Com}^*$ from Figs. 19.1 and 19.2 is* quantum-computationally hiding.

19.2.2. Receiver Security

We now investigate the binding property of the commitment scheme:

Lemma 19.2.6 (Unconditionally binding). *If $\textsc{Com}'$ is an* unconditionally binding *commitment scheme, $\textsc{Com}^*$ from Figs. 19.1 and 19.2 is* unconditionally binding.

```
Algorithm $\mathcal{B}(|\phi\rangle, otp, \mathsf{com}_{\vec{b}} := \{(\mathsf{com}'_{\vec{b}[i]}, \mathsf{com}'_{\mathsf{sk}_i}, ct_{1,i}, ct_{2,i})\})$
for $i \in [\mathcal{N}]$ do
    $(|\phi\rangle_i, ct_i) \leftarrow \textsc{QFhe}.\mathrm{Eval}(|\psi\rangle[i], ct_{2,i}, \mathrm{H}^{(ct_{1,i})})$
done
$c' \leftarrow \mathsf{measure}_{+}(|\phi\rangle)$
return $(c', \mathsf{com}_{\vec{b}}, \{ct_i\}_{i=1}^{\mathcal{N}}, otp)$
```

Figure 19.4.: QPT part of the Obfuscated Measurement Attack to break the everlasting hiding property.

Proof. The reduction from the unconditional binding property of $\textsc{Com}^*$ to the unconditional binding property of $\textsc{Com}'$ is straightforward. Assume that there is an adversary $\mathcal{A}^*$ on the binding property of $\textsc{Com}^*$. We construct an adversary $\mathcal{A}'$ on the binding property of $\textsc{Com}'$ as follows:

When $\mathcal{A}^*$ sends a commitment of the form $(\mathsf{com}'_b, \mathsf{com}'_{\mathsf{sk}}, ct_1, ct_2)$ to $\mathcal{A}'$, $\mathcal{A}'$ forwards com'_b to the challenger C_{Bind}. $\mathcal{A}'$ then receives a bit b that com'_b has to be unveiled to, which it directly forwards to $\mathcal{A}^*$. When $\mathcal{A}^*$ sends the unveil information $(b, \mathsf{unv}'_b, \mathsf{sk}, \mathsf{unv}'_{\mathsf{sk}})$, $\mathcal{A}'$ forwards (b, unv'_b) to the challenger C_{Bind}.

If $\mathcal{A}^*$ wins with probability $1/2 + \alpha$, then $\mathcal{A}'$ also wins with probability $1/2 + \alpha$, thus implying that $\alpha \in \mathsf{negl}(\kappa)$. □

19.3. An Obfuscated Measurement Attack

19.3.1. Description

We now present an Obfuscated Measurement Attack from Figs. 19.4 and 19.5 on the everlasting hiding property of the quantum commitment protocol Π_{QCom}. The attack employs the instantiation $\textsc{Com}'$ from Figs. 19.1 and 19.2 for the classical component of Π_{QCom}.

The attack consists of two algorithms: the QPT adversary $\mathcal{B}$, and the unbounded (classical) adversary $\mathcal{A}$: After creating the generator matrix $\vec{G}$ according to Fig. 16.3 and sending the result to the sender, $\mathcal{B}$ receives a message $(|\phi\rangle, otp, \mathsf{com}_{\vec{b}})$. Recall that due to our instantiation of $\textsc{Com}'$ as a bit commitment, $\mathsf{com}_{\vec{b}}$ is a list of $\mathcal{N}$ quadruplets $(\mathsf{com}'_{\vec{b}_i}, \mathsf{com}_{\mathsf{sk}_i}, ct_{1,i} := \textsc{Fhe}.\mathrm{Enc}(\mathsf{sk}_i, [\vec{b}_i = \times]), ct_{2,i} := \textsc{Fhe}.\mathrm{Enc}(\mathsf{sk}_i, (0,0)))$.

$ct_{1,i}$ puts the adversary in possession of an encrypted control bit, which is 1 iff the corresponding quantum state was encoded in diagonal bases. With $ct_{2,i}$, the adversary has

```
Algorithm $\mathcal{A}(c', \mathsf{com}_{\vec{b}} := \{(\mathsf{com}'_{\vec{b}[i]}, \mathsf{com}'_{\mathsf{sk}_i}, ct_{1,i}, ct_{2,i})\}, \{ct_i\}_{i=1}^{\mathcal{N}}, otp)$
for $i \in [\mathcal{N}]$ do
    Extract $\mathsf{sk}_i$ from $\mathsf{com}_{\mathsf{sk}}$
    $(\mathsf{x}[i], \mathsf{z}[i]) \leftarrow \textsc{Fhe}.\mathsf{Dec}(\mathsf{sk}_i, ct_i)$
done
$c := c' \oplus \mathsf{x}$
$b := \langle c, otp \rangle$
return $b$
```

Figure 19.5.: Unbounded part of the Obfuscated Measurement Attack to break the everlasting hiding property.

an encryption of the Quantum One Time Pad $(0, 0)$, corresponding to a quantum state hidden with an empty QOTP—that is, one where x = and $\mathsf{z} = 0$ and hence where neither X nor Z was applied to. This suffices to let the adversary homomorphically adjust all quantum states from Wiesner encoding to QOTP encodings in the computational basis $+$: $\mathcal{B}$ homomorphically applies a Controlled Hadamard gate with encrypted control bit $ct_{1,i}$. We show later that this is possible using the scheme of Brakerski [32] due to a result from Shi [122].

$\mathcal{B}$ can then measure the outcome and return all the resulting classical information to $\mathcal{A}$. The unbounded adversary then extracts the secret key sk from the commitment $\mathsf{com}'_{\mathsf{sk}}$ contained in the commitment created by $\textsc{Com}^*$ and uses it to break the encryptions of the QOTP. With the code word c, $\mathcal{A}$ can reconstruct b and thus break the hiding property.

Lemma 19.3.1. *The adversary $\mathcal{A}$ can successfully reconstruct the sender's initial input with overwhelming probability.*

Proof. An honestly created quantum state $|\phi\rangle$ implies that each of the $\mathcal{N}$ quantum states $|\phi\rangle[i]$ is in one of four possible states $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$. Homomorphically applying a Controlled Hadamard (CH) gate to this state using the (encrypted) control bit for $[\vec{b}_i = \times]$ has the following effects: It leaves $|0\rangle$ and $|1\rangle$ untouched, but applies the Hadamard gate to $|+\rangle$ and $|-\rangle$, transforming them into $|0\rangle$ and $|1\rangle$, respectively.

The quantum state at this point is a QOTP encrypted version of either $|0\rangle$ or $|1\rangle$. The keys of a QOTP define the *Pauli* gates applied to the quantum state. With only two bits and one (binary) quantum state, there are only eight combinations of the binary QOTP keys (x, z) and quantum states $|\phi\rangle$ for which it holds that $\mathsf{Z}^{(\mathsf{z})}\mathsf{X}^{(\mathsf{x})}|\phi\rangle = |0\rangle$ or $\mathsf{Z}^{(\mathsf{z})}\mathsf{X}^{(\mathsf{x})}|\phi\rangle = |1\rangle$. The set of possible quantum states $|\phi\rangle$ are shown in Table 19.1, proving that those too are either $|0\rangle$ or $|1\rangle$.

Upon measurement in the computational basis, no information is destroyed; a perfect measurement apparatus yields a deterministic output. From the correctness of the QFHE scheme, it follows that the FHE ciphers of the new OTPs are correct as well.

$\lvert\psi\rangle = \lvert 0\rangle$				$\lvert\psi\rangle = \lvert 1\rangle$			
$x = 0$	$x = 1$		$x = 0$		$x = 1$		$x = 0$
$z = 0$		$z = 1$		$z = 0$		$z = 1$	
$\lvert\phi\rangle = \lvert 0\rangle$	$\lvert\phi\rangle = \lvert 1\rangle$	$\lvert\phi\rangle = \lvert 1\rangle$	$\lvert\phi\rangle = \lvert 0\rangle$	$\lvert\phi\rangle = \lvert 1\rangle$	$\lvert\phi\rangle = \lvert 0\rangle$	$\lvert\phi\rangle = \lvert 0\rangle$	$\lvert\phi\rangle = \lvert 1\rangle$

Table 19.1.: Possible quantum states $\lvert\phi\rangle = Z^{(z)}X^{(x)}\lvert\psi\rangle$ a QOTP encryption of $\lvert\psi\rangle = \lvert 0\rangle$ or $\lvert\psi\rangle = \lvert 1\rangle$ can be in. For the sake of simplicity, we assume $-\lvert 1\rangle = \lvert 1\rangle$.

$\textsc{Com}'$ is unconditionally binding, hence both $\mathsf{com}'_{\vec{b}}$ and sk can be extracted by the unbounded adversary, who then learns the actual basis vector $\vec{b}$ and the secret key sk used by the sender.

With the extracted key sk, $\mathcal{A}$ can extract the QOTP keys from the encryption and classically undo the x part to obtain the correct code word c. Now, $\mathcal{A}$ can extract the bit b as $\langle c, otp\rangle$, which equals the sender's original input with overwhelming probability. □

Lemma 19.3.2. *The attack part of $\mathcal{B}$ from Fig. 19.4 is a QPT algorithm.*

Proof. Most operations in Fig. 19.4 can trivially be performed by any QPT adversary; the only operation for which it is not obvious is the homomorphic evaluation of $H^{(cct)}$ with an *encrypted* control bit cct.

We assume that ($\textsc{QFhe}$, $\textsc{Fhe}$) are instantiated such that there are QPT algorithms capable of homomorphically evaluating a *Hadamard* gate H and a *Toffoli* gate $X^{(cb_1,cb_2)}$ over *classically* encrypted inputs (cb_1, cb_2) on a QOTP-encrypted quantum state $\lvert\phi\rangle$, which holds *e.g.* for the scheme from Brakerski [32].

Following Shi [122, Theorem 3.2], Toffoli and Hadamard gates are *complete*: they can be used to compute *any* QPT circuit. Thus, there exists a polynomial-sized quantum circuit that only contains Toffoli and Hadamard gates realizing the *Controlled Hadamard* gate $H^{(cct)}$. □

Finally, we arrive at the following corollary:

Corollary 19.3.3 (Obfuscated Measurement Attack). *In order to prevent the OMA on Π_{QCom}, the classical commitment component of Π_{QCom} requires more security properties than unconditionally binding and computational hiding.*

20. Conclusion

We formalize quantum decoherence as a new cryptographic assumption that reflects the major challenge of keeping quantum states coherent for a long period of time. We construct a Quantum Commitment on a classical bit and prove that our assumption enables the scheme to have everlasting hiding and unconditional binding properties in the QROM. In the context of everlasting security, our adversary is modeled as a QPT algorithm which is active during the protocol execution and becomes classically unbounded after polynomially many time steps. Our protocol extends the work of Brassard et al. [36], but circumvents attacks similar to Mayers [108] by adding a classical commitment to the bases used for encoding the quantum state. A consequence of this setup is the possibility to achieve everlasting security without the need to interrupt the parties during the execution of the protocol.

Moreover, we show that our findings might not be transferable when leaving the QROM and using a classical commitment based on standard assumptions or primitives (for example Quantum Fully Homomorphic Encryption) instead. Indeed then the Quantum Commitment scheme may become vulnerable to our innovative Obfuscated Measurement Attack. This raises the question which properties were inherited by the classical commitment from the QROM that were not apparent in the homomorphic commitment protocol $\textsc{Com}^*$ from Figs. 19.1 and 19.2, and which additional assumptions and requirements we need to reach everlasting security without deploying a QROM.

Finally, we note that a limitation needs to be considered. Our construction is susceptible to an adversary that can *store* a given quantum state over a long period of time, *i.e.* until he can perform a super-polynomial number of computational steps, which is believed to be easier than *storing and manipulating* over a long time. To identify and formalize a quantum cryptographic assumption that enforces the latter is left as a subject to future work.

Part III.

Anonymous Whistleblowing

21. Introduction

Secure Multi-Party Computation (MPC) allows a set of N parties to jointly evaluate a function f on their private inputs. The protocol is secure, if the computation does not reveal anything about these inputs beyond what can be inferred from the output. There are many scenarios where this is advantageous. For example, the danish sugar beet auction [59] enabled private trading of sugar beets and settling on a price that would maximize the revenue while selling as many sugar beets as possible.

The general scenario there is that the parties mutually agree to engage in a protocol execution that yields the result of the to-be-computed function f on the private inputs x_i. However, in certain scenarios, the mere fact that a party is interested to participating in a certain protocol execution or even is planning to do so reveals a lot of information about that party. Consider for instance the case where your company was hacked but you don't have enough forensic data to trace the attackers. If several companies fell victim to the same hacker, a joint effort might yield a sufficient amount of information to successfully trace the hacker. However, the very fact that you are participating in such a protocol reveals the fact that your company has been hacked. As an other example, consider the classical motivation for MPC where Bob wants to ask Alice out for a date but does not want her to know he has feelings for her unless she shares these feelings. Approaching Alice and suggesting to compute a secure AND to find out whether they will go on a date or not already reveals the intention from Bob to ask Alice out.

In these situations it would be desirable to hide even the *computation* and not just the inputs. A first step in this direction was taken by von Ahn, Hopper, and Langford [133] and Chandran et al. [48] and was later extended by Jarecki [93] and Couteau [56] in the form of Covert Multi-Party Computation (CMPC). In this setting a set of parties can run protocols in a way that (1) no outsider is able to tell that a protocol execution has taken place at all among the participants, and (2) no participant can tell if a protocol execution even took place if the outcome is unfavorable or if not all parties were actively participating. This method solves both problems mentioned above. However, this setting has two major shortcomings. First, after the protocol terminates, all participants learn about each other's participation. This can be undesirable in certain situations. Consider the following scenario: You are happily employed by some government agency. However, one day, you learn that your employer violates human rights. You strongly disagree with this breach of trust and law but you are bound by law to keep internal information secret. Consequently, you are faced with a dilemma: Either you ignore the human rights violation, or you face dishonorable discharge or even jail. In fact, whistleblowers often take an immense personal risk, and face sentences ranging from exile [12] to incarceration [115] or worse.

The desired solution for any whistleblower is to leak this information without being identified by anyone in the process. The importance of this question is well recognized in cryptography and security. It has been the subject of several influential works (*e.g.* DC-nets [49], Riposte [55] or Blinder [3]). Concrete solutions include the use of secure messaging apps [54, 25], mix-nets [50], onion routing systems such as the Tor network [62], or solutions built on top of DC-nets and secure computation techniques [55, 3] (see also [68, 113]). Yet, all current approaches to anonymous whistleblowing rely on trusted parties (or non-colluding partially trusted servers), which either receive privately the communication, or implement a distributed protocol to emulate an anonymous network. Therefore, however ingenious and scalable some of these solutions are, whistleblowers must ultimately trust that they will interact with parties or servers which will (at least for some of them) remain honest and refuse to collude throughout the transmission.

Covert computation does not have such trust assumption but does not help in this scenario either since as soon as the protocol finishes, the receiver of the message learns the origin of the message. The second shortcoming of CMPC is with respect to non-participants. By design, the protocol *only* yields a result if all parties participate (and the result is *favorable*), but it fails to provide a meaningful result if even one of the parties is not participating. As a result, a bystander can accidentally break the entire computation and a malicious party can even abort any such ongoing protocol without repercussions.

This part of this thesis is dedicated to investigate whether these restrictions are inherent or if we can construct a primitive we call “undetectable computation”, where parties participate without revealing their participation even towards other participants and where a fixed set of non-participants is tolerated by the protocol. More precisely, we study the following problem: K individuals are interacting. Among them, N players are willing to jointly compute a public function on their private inputs, while the remaining $(K - N)$ are not interested in taking part to the protocol.

So the key research question for this part of the thesis comes down to the following:

Is it possible for N participating players to evaluate a given function f on their private inputs, such that no one (not even the players themselves) can determine who else took part to the protocol execution?

In this part of this thesis we will reduce the above problem of evaluating f on private inputs $(x_1, \ldots, x_N)$ while hiding participation to the players ability to exchange messages without revealing their identity. We will therefore focus on the following question:

Is it possible for a sending party to transmit a message to a receiving party in the presence of unaware non-participating parties such that the identity of the sender remains hidden among the group of non-sending parties even to the receiver?

We call this notion Anonymous Transfer (AT), which we consider to be an interesting topic in itself. Following strong impossibility results on the feasibility of an AT with asymptotic security and overwhelming correctness *and* anonymity guarantees we focus on circumventing the impossibility result by considering fine-grained security instead of asymptotic ones and by relaxing the anonymity guarantee to hold only for an arbitrary polynomial.

The primitive is without a doubt quite powerful as it solves—among other things—a general whistleblowing-problem without relying on anything other than authenticated channels. Consider the communication channel to be a publicly accessible board such as Twitter. Say via Twitter a reporter obtained classified information on human rights violations of government agencies, where the AT protocol itself is public knowledge and known to be executed among all the twitter followers from the journalist—actual protocol messages are embedded into the distribution of respective tweets. Assume further that a court order forces the journalist to cooperate with the state, forcing him to open any information available on the protocol. Our primitive requires that any follower of the journalist—which might even be the judge—is equally likely to be the sender of the message, even when all of the receivers messages and his random tape are known.

As any follower of the journalist is equally likely to be the sender of the message—even the judge if he follows the journalist as well—it follows that the real sender cannot be determined with sufficient certainty, and that thus no penalty can be imposed on the whistleblower.

We thus model our setting as more simplistic semi-honest case where parties only can be corrupted after the execution finished.

21.1. Contribution

Our contribution in this part of the thesis consists of:

Anonymous transfer. We formally define Anonymous Transfer (AT) as an interactive protocol between a sender, a receiver and an unknowing non-participant. We assume all parties to interact in a synchronous model over a public broadcast channel. That is, in each round each individual broadcasts a message which only depends on messages from previous rounds. The non-participant is not aware that a protocol takes place. We follow [133, 48] and model non-participating parties as parties that only broadcast uniform randomness in each round, since any ordinary communication pattern can be embedded into the uniform distribution using standard techniques [133, 86, 132].

We formally define the security properties of an AT. The sender aims to transmit a message to the receiver in a way that does not leak the identity of the sender. We say that an AT protocol is ε-correct if the probability that the receiver successfully receives the message is at least ε. Further, we say that an AT protocol is δ-anonymous if no adversary is able to determine the identity of the sender (given the transcript and the receiver's random tape) has advantage more than $(1 - \delta)/2$ over guessing. These are the core properties which shape an AT protocol.

In order to use the AT as building block to construct undetectable MPC we require any AT protocol to additionally satisfy secrecy. On a high level, an AT protocol satisfies secrecy if it is infeasible to recover the transmitted message without access to the receivers random tape. Combined with the correctness property this means that only the party acting as receiver during the protocol execution is able to reconstruct the transferred message, which is contrary to our notion of *Silent Receiver Anonymous Transfer*; yet we stress that both make sense individually. While SRAT is used for our analysis on the feasibility of AT (which in itself ignores secrecy and only focusses on anonymity and correctness), secrecy is only required when using AT as a building block for a larger protocol.

Impossibility of AT. The two main properties of AT, namely correctness and anonymity, contradict each other. Correctness requires the receiver to recover the correct message after the protocol execution, while anonymity ensures that the receiver is unable to tell which party was the sender. As such, a protocol that is perfectly correct is subject to a generic *replacement* attack. From the perfect correctness of a protocol it follows that an honest transcript *always* yields the correct bit[1], yet when replacing *all* messages with randomness the protocol yields a bit from a given pre-defined distribution (say it yields a uniformly distributed bit).

The impossibility is easy to see for any non-interactive protocol. By replacing the message of only *one* party with randomness while taking the other parties message from the transcript, the resulting transcript is either completely random (and hence yields a bit according to the aforementioned distribution) if the senders message was replaced, or still a valid transcript for transferring the desired bit (as the random message was replaced by a different random message, and since the sender has to choose the message *before* knowing the non-participants message, correctness guarantees that the message is transferred regardless of the non-participants message) if the non-participants message was changed.

The same idea can be extended to any c-round protocol. Here, we focus on the last message of an execution. Either the protocol is such that the message is already sufficiently fixed before the final round, or the final-round message of the sender changes the distribution significantly. The former implies the existence of a $(c-1)$-round protocol where the parties execute the first $(c-1)$ rounds of the protocol, and to reconstruct the receiver inserts random messages for both parties during the final round. This protocol would be equally correct but only requires $(c-1)$ rounds. The latter enables an attack similar to the non-interactive case, where the message of one party for the final round is replaced by a random message.

[1] Except for the negligible chance that the non-participants messages, which were sampled uniformly random, happen to be a precise transcript for transferring the respective other bit; but we ignore this possibility here as it is highly unlikely.

While the latter case enables the attack directly, the former can be continued via induction; either the latter case happens eventually, or we end up with a non-interactive protocol with the same (or just slightly less) correctness guarantees as the interactive one, to which the impossibility described above applies.

We elaborate in Chapter 24 but stress that it was taken verbatim from Agrikola, Couteau, and Maier [5, Section 4] and is not a contribution of this thesis.

Feasibility of fine-grained AT. To circumvent the impossibility results described above, we give up asymptotic security and resort to the fine-grained setting. That is, we only require anonymity against adversaries which require polynomially more resources than an honest protocol execution. We propose a protocol which—assuming ideal obfuscation—allows to reduce the problem of de-anonymizing the sender to a distribution testing problem.

More precisely, we show that determining the real sender in a c-round protocol given only a transcript of the AT protocol is *as hard* as differentiating between a *Bernoulli* oracle that returns 1 with probability p and one that returns 1 with $p + 1/2c$. For this problem, strong lower bounds on the number of required samples and thus the adversarial runtime are known.

'Philosophical implications': between obfustopia and impossibilitopia. There is a small remaining gap between our negative and positive results: the possibility of building Anonymous Transfer secure against arbitrary polynomial-time adversaries, but with non-negligible (*e.g.* inverse polynomial) anonymity error remains open. Closing this gap would have an intriguing philosophical consequence: stretching the terminology of Impagliazzo [89] on the "worlds" of cryptography, it would establish the existence of a cryptographic primitive that plausibly exists in obfustopia (the world where indistinguishability obfuscation is possible) in the fine-grained setting, yet does not exist ("reside in impossibilitopia") with standard hardness gaps.

Interestingly, there are several known examples where fine-grained constructions of a "higher world" primitive reside in a lower world; for example, (exponentially secure) one-way functions (a Minicrypt assumption) imply fine-grained public-key encryption (a Cryptomania assumption) [109, 26]. Our work seems to provide a new example of this behavior, at the highest possible level of the hierarchy, showing that impossible primitives might end up existing if we weaken their security to the fine-grained setting.

Open questions. Our work leaves open two exciting questions:

(1) *Can our impossibility result for logarithmic-round AT protocols be extended to a polynomial number of rounds?*

(2) *Is it possible to instantiate AT in the fine-grained setting from "Obfustopia" standard assumptions?*

Assuming that both our open questions can be answered affirmatively, this would separate the realm of asymptotic security from the realm of fine-grained security. Item (1) rules out the fact that AT can be instantiated even when using assumptions from obfustopia, it would then be in a world we call "Impossibilitopia". Yet Item (2) would place AT in the fine-grained version of obfustopia, providing a clear separation between these two worlds.

Furthermore, we only studied the feasibility of Anonymous Transfer in presence of *passive* adversaries, where all non-participants are not even required to be *aware* of the execution of the protocol and are only required to generate traffic. As we show, this weaker assumption does not suffice against arbitrary polynomial-time adversaries, but possibly suffices against bounded polynomial-time adversaries (where the bound is sub-quadratic). As a natural next step, one could extend the question and ask: What if some of the non-participating parties were in fact planted by a malicious adversary, and now act *maliciously* during the protocol? It seems plausible, that our general strategy can be extended to deal with malicious non-participants. However, we expect the analysis to require different techniques than the ones we used. We leave a thorough analysis and formal proof in the malicious setting to future work.

Organization. We start by introducing necessary preliminaries in Chapter 22 and formal definitions for Anonymous Transfer and strong Anonymous Transfer in Chapter 23. We then introduce the impossibility result in Chapter 24. We then proceed to Chapter 25 where we show how to circumvent the impossibility result by considering fine-grained security. In Chapter 26 we extend this result from single-bit AT to n-bit AT at the cost of roughly twice as many rounds. We define our novel primitive Undetectable Oblivious Transfer in Chapter 27 which allows two parties to hide their OT execution in a group of K individuals and instantiate it using strong AT in Chapter 28. Finally, in Chapter 29 we provide definitions for Undetectable Multi-Party Computation, which allows a number of N parties to perform an MPC protocol while hiding their identities among K individuals and instantiate UMPC using Undetectable Oblivious Transfer for $N = 3$.

21.2. Related Work

We already elaborated on the differences between Undetectable Multi-Party Computation and the existing notions of Covert Multi-Party Computation [133, 48, 93, 56] in our introduction, where we stressed that Undetectable Multi-Party Computation also works in the setting where K parties are present, of which $N < K$ parties try to perform a computation, instead of only focusing on the setting of $N = K$ where *all* parties present participate in the protocol execution.

Our positive and negative results on Anonymous Transfer relate to the literature on anonymous broadcast and secure whistleblowing. In general, a whistleblower willing to reveal something anonymously has two alternative choices: (1) either the whistleblower has access to an anonymous communication channel, for example by putting their message (say, encrypted with the receiver public key) on some public website that somehow cannot be traced to them. However, access to an anonymous channel is typically a "physical" assumption, and one which is *very* hard to guarantee. This issue is developed in great detail in the literature: see for example the discussion in Spectrum [113] about how metadata have been used by federal judges to trace and prosecute people who leaked data through secure messaging apps, or the discussion in Riposte [55] and Express [68] on how traffic analysis can be used to trace whistleblowers on the Tor network or the SecureDrop service. Hence, most of the literature focuses on scenario (2): the individuals interact over a communication network, and we do not assume that this network guarantees anonymity in itself. In this case, what we want is to *emulate* this anonymity, by developing a strategy to help the whistleblower transmit a message anonymously to the receiver.

The literature on this subject is incredibly vast, but this emulated anonymity is *always* achieved using the same template in all solutions we are aware of (including Spectrum, Blinder [3], Riposte, Express, Talek, P3, Pung, Riffle, Atom, XRD, Vuvuzela, Alpenhorn, Stadium (or any other Mixnet-based solution), Karaoke, Dissent, Verdict, and many more): when the whistleblower wants to anonymously transmit a message, either to everyone (anonymous broadcast) or to a target receiver, other users generate "honest" traffic in which communications can be hidden. To do so, the users interact with a set of *non-colluding* servers (sometimes two servers, sometimes more, some with honest majority, some without). This is never even discussed or remarked: it is taken as an obvious fact that this is *the* structure of an anonymous broadcast (or messaging) protocol. And indeed, the need to generate honest traffic feels clear – if the whistleblower is the sole sender, observing traffic directly leaks their identity. That the use of non-colluding servers was never challenged or even discussed probably means that it also *feels* clear – but this assumption is precisely what we challenge in our work: we do assume that some users generate honest traffic, but we ask whether the assumption of non-colluding participating servers is avoidable. Of course, any scientific treatment of a broad question ("are non-colluding helpful participants required for anonymous broadcast?") is bound to move from the broad question to a formal model, in which (feasibility or impossibility) results can be achieved. Nevertheless, we believe that our impossibility result demonstrates that the use of non-colluding servers in all previous works was indeed unavoidable, at least insofar as their aim was to achieve anonymity against arbitrary polynomial-time adversaries.

22. Preliminaries

22.1. Steganography

Informally, steganography describes the art of hiding information in such a way, that no outsider is able to even notice that any information is hidden at all. The concept has been around for a long time, yet it was first brought up as a field of scientific research by Simmons [124] and put into a more complexity-theoretic and cryptographic context by Hopper, Langford, and von Ahn [86]. Simmons [124] described the problem analogously to prison communication: Two prisoners can talk to each other and want to come up with an escape plan, but a warden hears everything they say. The problem of steganography can roughly be described by the following question: How can the two prisoners discuss their escape plan without the warden noticing that the two prisoners are up to something?

While it is true that in most scenarios, the two prisoners could simply encrypt their messages and discuss their escape plan through a secure channel, this method is insufficient for the scenario where a warden is present: If the warden notices that the two prisoners are up to something, they end up in solitary confinement. Thus the problem is much harder than the one considered in *classical* encryption, as the *actual communication itself* needs to be undetectable for the warden. In contrast, it suffices for classical encryption schemes if any other person can *detect* messages as long as they are unable to *decrypt* them. We consider the differentiation to be similar to the one between Secure Multi-Party Computation and Covert Multi-Party Computation, where the former only hides *information* whereas the latter hides the *entire execution*.

A steganographic channel is modeled as indistinguishable from the *uniform* channel over m bits [86, 132, 133], where each party broadcasts *uniformly random* bitstrings of length m. This is neither a restriction nor an additional trust assumption: steganographic methods use actual sources of randomness like the least significant bit in timestamps or the least significant bits of images (where minor differences can occur due to the noise induced by the camera) that follow two basic rules: (1) when naturally occurring, they are uniformly random, and (2) it is efficiently possible by a sender to change this source such that it has a target value, without acting in a noticeably different way. A random source is suitable for steganography if it fulfills both these requirements: Unaware parties *unknowingly* broadcast random bits automatically, while sending parties can efficiently embed any message.

As a practical example that not only motivates how steganography can be used in practice, but also illustrates the fact that the natural messages are indistinguishable from a sampling of the uniform channel, think of a public forum where all registered parties can post images of cats. Each post has its timestamp attached (in UNIX-time, *i.e.*, seconds since the first of January in 1970), which is set by the sender during the upload and not by the server. As such, this can be exploited as a steganographic channel: In this (arguably inefficient) steganographic channel the timestamps of each post encodes a single bit, which can be extracted directly by taking the least significant bit of the timestamp.

It is easy to see that under normal circumstances, people who just use the forum to upload cat images whenever they feel like it do not need to be aware of the fact that the steganographic channel exists, yet they still broadcast random bits simply by doing what they naturally do, since the least significant bit of the timestamp provides a sufficiently uniform source of randomness. Furthermore, the delay of up to one second ends up unnoticed, so it is possible to covertly embed a message by manipulating the least significant bit of the timestamp.

While it might not be the case that all parties send the *same* number of messages—which might be required if we execute a protocol over several rounds by embedding (random looking) messages into the steganographic channel as was done by von Ahn, Hopper, and Langford [133])—this is not actually a restriction. If party P_i has finished a given round by publishing m bits while P_j still only published $m' < m$ bits, P_i continues as normal and publishes images at arbitrary timestamps until all other parties also broadcast m bits. The protocol then only considers the first m messages for each party in each round and ignores all other messages of the resp. parties until the next round starts.

22.2. Distribution Testing

In this section, we introduce preliminaries for probability testing. We start by describing the *Total Variational Distance* between two distributions.

Definition 22.2.1 (Total Variational Distance)**.** *Let* p *and* q *be two probability distributions. The* total variational distance *between* p *and* q *is defined as:*

$$\mathrm{d}_{\mathrm{TV}}(\mathrm{p}, \mathrm{q}) \coloneqq \frac{1}{2}\sum_i |\mathrm{p}_i - \mathrm{q}_i| = \frac{1}{2}\|\mathrm{p} - \mathrm{q}\|_1 \tag{22.1}$$

An important property of the total variational distance is that it acts *sublinear* when taking many samples. When taking t samples from a Bernoulli distribution the corresponding distribution can be described by taking a single sample from a t-bit *Binomial* distribution. The sub-additivity then bounds the total variational distance of the corresponding binomial distribution:

Lemma 22.2.2 (Total variational distance of a t-fold probability distribution)**.** *Let* p *and* q *be two Bernoulli distributions with total variational distance* $\mathrm{d}_{\mathrm{TV}}(\mathrm{p}, \mathrm{q})$*. Then it holds for* *the binomial distributions* $\mathrm{p}^{\otimes t}$ *and* $\mathrm{q}^{\otimes t}$ *that result from sampling* t *times from the respective distributions:*

$$d_{\mathrm{TV}}(\mathrm{p}^{\otimes t}, \mathrm{q}^{\otimes t}) \leq t \cdot d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q}) \tag{22.2}$$

Thus we can bound the distinguishing advantage of *any* distinguisher who has taken t samples form the same oracle using the total variational distance of the respective distributions directly.

A similar rule also holds for two *different* distributions, where the distinguisher has to distinguish whether two samples originate from $\mathrm{p} \otimes \mathrm{r}$ or from $\mathrm{q} \otimes \mathrm{s}$ for known values of p, q, r and s. In this case the rule states that:

Lemma 22.2.3 (Sub-Additivity of the Total Variational Distance for Product Distributions)**.** *Let* p *and* q *be a probability distribution over* $\{0,1\}^m$ *with total variational distance* $d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q})$*. Let* r *and* s *be two Bernoulli distributions with total variational distance* $d_{\mathrm{TV}}(\mathrm{r}, \mathrm{s})$*. Then it holds for the distribution derived from sampling from each distribution once and concatenating the outputs (which yields a sample from* $\{0,1\}^{m+1}$ *originating either from* $\mathrm{p} \circ \mathrm{r}$ *or* $\mathrm{q} \circ \mathrm{s}$*) that*

$$d_{\mathrm{TV}}(\mathrm{p} \circ \mathrm{r}, \mathrm{q} \circ \mathrm{s}) \leq d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q}) + d_{\mathrm{TV}}(\mathrm{r}, \mathrm{s})$$

The following lemma limits the distinguishing advantage of any distinguisher that tries to distinguish two distributions p and q based on a single sample.

Lemma 22.2.4 (Distinguishing distributions based on the Total Variational Distance)**.** *Let* p *and* q *be two distributions with total variational distance* $d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q})$*. If* $d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q}) < \frac{1}{3}$*, then no algorithm can exist that distinguishes* p *and* q *with probability* $\geq \frac{2}{3}$ *based on a single sample.*

We stress here that this is not our proof but instead adapted from `http://cs.brown.edu/courses/csci1951-w/lec/lec%2011%20notes.pdf`. Since we did not find a citeable source our adapted proof follows, which basically follows the source:

Proof. Let $\mathcal{D}$ be the distinguisher between p and q. Due to symmetry it holds for the correctness that the optimal algorithm has:

$$\begin{aligned}
&\Pr\left[\ \mathrm{out}_{\mathcal{D}} = \mathrm{p} \mid \mathrm{p}\ \right] - \Pr\left[\ \mathrm{out}_{\mathcal{D}} = \mathrm{p} \mid \mathrm{q}\ \right] \\
=&\Pr\left[\ \mathrm{out}_{\mathcal{D}} = \mathrm{p} \mid \mathrm{q}\ \right] - \Pr\left[\ \mathrm{out}_{\mathcal{D}} = \mathrm{q} \mid \mathrm{q}\ \right] \\
=&d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q})
\end{aligned} \tag{22.3}$$

With a single sample the adversary can only output a single bit indicating whether it thinks the sample is from p or from q. This cannot be done with better than $d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q})$. To get an intuition as to why this is the case, the total variational distance can also be written as:

$$d_{\mathrm{TV}}(\mathrm{p}, \mathrm{q}) := \sup_{X \subseteq [m]} \mathrm{p}(X) - \mathrm{q}(X) \tag{22.4}$$

that is, it corresponds to the largest difference in the probability of occurrence of some output. Thus, no distinguisher can—with a single sample—guess correctly with larger advantage than $\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})$. Guessing is always an option, which would result in the distinguisher being correct with probability $1/2$. Thus, there can be no distinguisher which fulfills *both* these inequalities:

$$\begin{aligned}\Pr\left[\ \mathtt{out}_{\mathcal{D}} = \mathrm{p} \mid \mathrm{p}\ \right] &> \frac{1}{2} + \frac{1}{2} \cdot \mathrm{d_{TV}}(\mathrm{p},\mathrm{q}) \\ \Pr\left[\ \mathtt{out}_{\mathcal{D}} = \mathrm{q} \mid \mathrm{q}\ \right] &> \frac{1}{2} + \frac{1}{2} \cdot \mathrm{d_{TV}}(\mathrm{p},\mathrm{q})\end{aligned} \tag{22.5}$$

as this would imply an overall correctness larger than $\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})$.

By using the requirement that $\mathrm{d_{TV}}(\mathrm{p},\mathrm{q}) < 1/3$ in Eq. (22.5) we get the bound of $2/3$ on the correctness of $\mathcal{D}$. □

Using Lemmas 22.2.2 and 22.2.4 we can provide lower bounds on the sampling complexity of distinguishing two distributions p and q with advantage $\alpha/2$.

Corollary 22.2.5 (Distinguishing two Bernoulli-Distributions with t samples)**.** *Any distinguisher $\mathcal{D}$ that distinguishes between p and q with probability $\geq \frac{1}{2} + \frac{\alpha}{2}$ requires $t \in \Omega\left(\frac{\alpha}{\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})}\right)$ samples.*

Proof. The total variational distance limits the advantage as $\alpha = \mathrm{d_{TV}}(\mathrm{p}^{\otimes t},\mathrm{q}^{\otimes t})$.

We know from the subadditivity of $\mathrm{d_{TV}}$ that

$$\mathrm{d_{TV}}(\mathrm{p}^{\otimes t},\mathrm{q}^{\otimes t}) \leq t \cdot \mathrm{d_{TV}}(\mathrm{p},\mathrm{q}) \tag{22.6}$$

and as such,

$$\Pr[\mathcal{D} \text{ correct}] \leq 1/2 + 1/2 \cdot t \cdot \mathrm{d_{TV}}(\mathrm{p},\mathrm{q}) \tag{22.7}$$

We have that $t \in \Omega\left(\frac{\alpha}{\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})}\right) \implies t \geq \frac{\alpha \cdot c}{\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})}$. We claim that this even holds for $c = \frac{1}{4}$. This implies for Lemma 22.2.2 that

$$\begin{aligned}\mathrm{d_{TV}}(\mathrm{p}^{\otimes t},\mathrm{q}^{\otimes t}) &\leq \frac{c \cdot \alpha}{\mathrm{d_{TV}}(\mathrm{p},\mathrm{q})}\mathrm{d_{TV}}(\mathrm{p},\mathrm{q}) \\ &= \alpha c = \frac{\alpha}{4}\end{aligned} \tag{22.8}$$

Thus, the total variation distance is less than $\alpha/4$ for $\alpha \leq 1$, which in term is less than $1/3$.

Remember that $\mathrm{d_{TV}}(\mathrm{p}^{\otimes t},\mathrm{q}^{\otimes t})$ corresponds to distinguishing the t-fold (binomial) distribution with a single sample. Thus Lemma 22.2.4 states that with total variation distance less than $1/3$ no distinguisher $\mathcal{D}$ can successfully distinguish with probability $\geq 2/3$. □

22.3. Covert Oblivious Transfer

Covert Oblivious Transfer is an extension of classical Oblivious Transfer first defined and used by von Ahn, Hopper, and Langford [133]. It extends classical OT by two important properties to ensure that the computation cannot be distinguished from an innocent-looking conversation: (1) the sender cannot distinguish between the case that the receiver is following the COT protocol or if the receiver is sending uniformly random messages, and (2) if the transferred messages are uniformly random, the receiver cannot distinguish between the case that the sender is following the COT protocol and the case where the sender is sending uniformly random messages.

The latter has to hold even after the protocol execution is finished, as the authors use Covert Oblivious Transfer during the execution of their Covert Multi-Party Computation protocol and actual participation should not be revealed before the computation finishes.

Definition 22.3.1 (Covert Oblivious Transfer, [133, Appendix A]). *A 2-party Covert Oblivious Transfer (COT) between a sender* S *and a receiver* R *is an Oblivious Transfer but additionally fulfills the following two requirements:*

(1) S *cannot distinguish between the case that* R *is following the COT protocol and the case that* R *is drawing uniformly random messages from* U_m.

(2) If $\Sigma_0, \Sigma_1 \overset{\$}{\leftarrow} \mathrm{U}_m$, R *cannot distinguish between the case that* S *is following the COT protocol and the case that* S *is drawing uniformly random messages from* U_m.

22.4. Indistinguishability from Random under Chosen Ciphertext Attacks

For our instantiations it is crucial that protocol messages can be transferred covertly, that is, without bystanders noticing. To that end we require an encryption that yields ciphertexts which look like uniformly random strings. A formal property ensuring this was defined by Rogaway [119] as Indistinguishability from Random.

While the paper also defines IND$-CPA, we focus on chosen *ciphertext* attacks only and provide definitions for symmetric and asymmetric schemes. For symmetric encryption schemes the definition looks as follows:

Definition 22.4.1 (IND$-CCA for Symmetric Encryption). *A Symmetric Encryption Scheme* $\mathcal{SKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *fulfills Indistinguishability from Random under Chosen Ciphertext Attacks (IND$-CCA) if the following probability is negligible in the security parameter* κ *for all PPT valid adversaries* $\mathcal{A}$ *that do not decrypt a challenge:*

$$\begin{aligned} &\left| \Pr\left[\mathcal{A}^{\mathrm{Enc}(\mathsf{sk},\cdot),\mathrm{Dec}(\mathsf{sk},\cdot)} = 1 \;\middle|\; \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\kappa) \right] \right. \\ &\left. - \Pr\left[\mathcal{A}^{\mathrm{PRF}(\cdot),\mathrm{Dec}(\mathsf{sk},\cdot)} = 1 \;\middle|\; \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\kappa) \right] \right| \end{aligned} \tag{22.9}$$

PRF is a random function. It is hence the adversaries task to distinguish whether it is interacting with a random oracle or with an actual encryption oracle.

The definition for asymmetric schemes is equivalent except that the adversary is additionally given a public key. Yet the challenge still comes from either an encryption oracle or a random oracle.

Definition 22.4.2 (IND$-CCA for Asymmetric Encryption)**.** *An Asymmetric Encryption Scheme* $\textsc{Pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *fulfills Indistinguishability from Random under Chosen Ciphertext Attacks (IND$-CCA) if the following advantage is negligible in the security parameter* κ *for all PPT valid adversaries* $\mathcal{A}$ *that do not decrypt a challenge:*

$$\begin{aligned} &|\Pr\left[\ \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{pk}) = 1 \;\middle|\; (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)\ \right] \\ -&\Pr\left[\ \mathcal{A}^{\mathsf{PRF}(\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{pk}) = 1 \;\middle|\; (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)\ \right]| \end{aligned} \tag{22.10}$$

22.5. Strong Existential Unforgeability under Chosen Message Attacks

For our construction we rely on a signature scheme. Their security is defined as follows [75]:

Definition 22.5.1 (Strong Existential Unforgeability under Chosen Message Attacks for Signatures)**.** *A signature scheme* $\textsc{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vfy})$ *fulfills Strong Existential Unforgeability under Chosen Message Attacks (sEUF-CMA) if for every PPT-adversary* $\mathcal{A}$ *with access to a signing oracle* $\mathsf{Sign}(\mathsf{k},\cdot)$ *it holds that the following probability is negligible in the security parameter* κ*:*

$$\Pr\left[\ \textsc{Sig}.\mathsf{Vfy}(\mathsf{vk},\sigma^*,X^*) = 1 \;\middle|\; \begin{array}{l} (\mathsf{vk},\mathsf{k}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa) \\ (\sigma^*,X^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{k},\cdot)}(\mathsf{vk},1^\kappa) \end{array} \right] \tag{22.11}$$

and for all queries X *to* $\mathsf{Sign}(\mathsf{k},\cdot)$ *which were returned by a signature* σ *it holds that* $X \neq X^*$ *and* $\sigma \neq \sigma^*$*.*

22.6. Ideal Obfuscation

We use an abstract concept of idealized obfuscation which replaces a given program P directly with an oracle that evaluates the program on the given input.

Definition 22.6.1 (Ideal Obfuscation)**.** *An ideal obfuscator describes the existence of two oracles* $(\mathsf{Obfuscate}, \mathsf{Evaluate})$*, where*

Obfuscate *takes as input a program* P *and randomly samples a handle* $h \in \{0,1\}^\kappa$*, saves* (P, h) *in a list and returns* h*;*

Evaluate *takes as input a handle* h *and an input* msg *and searches for* h *in the list. If* h *is in the list, evaluate the corresponding program* P *on the given input* msg *and return its output, otherwise output* $\perp$*.*

23. Anonymous Transfer

For an K-party protocol we consider the following situation: some individual P_ϕ for $\phi \in [K-1]$ is willing to transfer a message Σ to a receiver $\mathsf{R} = \mathsf{P}_K$, while hiding his identity ϕ among the $(K-1)$ potential senders. We call Anonymous Transfer (AT) an interactive protocol that achieves this goal.

23.1. Network Model and Non-Participating Parties

The goal of an anonymous transfer protocol is to hide the transfer of a message among innocent conversations by individuals, which are not taking part in the protocol. By a well-established folklore result in steganography (*cf.* Section 22.1), this task can be reduced to the simpler task of hiding the transferred message among *uniformly random beacons*, broadcast by the other individuals: the uniform channel, where all protocol messages look uniformly random, can be compiled into any other ordinary communication pattern [133, 86, 132]. Therefore, as in previous works (see von Ahn, Hopper, and Langford [133] and Chandran et al. [48]), we consider a set of K parties who interact with each other via broadcast channels and focus, without loss of generality, on protocols for the uniform channel. Consequently, we will model the non-participating parties as "dummy parties" that only broadcast uniformly random messages of a fixed length at each round.

23.2. The Model

Let $\phi \in \{1, \ldots, K-1\}$ denote the index of the real sender among the $(K-1)$ potential senders and let $\Sigma \in \{0,1\}^n$ be the message that P_ϕ wants to transfer to the receiver. We consider an interactive protocol in the Common Reference String (CRS) model between K individuals $(\mathsf{P}_1, \ldots, \mathsf{P}_{K-1}, \mathsf{R})$, where R and P_ϕ participate to the protocol, and P_i for $i \in [K-1] \setminus \phi$ are non-participating but present individuals that only broadcast random strings. The receiver R gets the CRS as input and the sender P_ϕ gets the CRS and the message Σ as input. For any player P, let T_P denote the random tape from which P draws his random coins. All the individuals $(\mathsf{P}_1, \ldots, \mathsf{P}_K)$ interact through authenticated broadcast channels in the synchronous model: the protocol proceeds in rounds, and each player broadcasts a message at each round. We denote by $\langle \mathsf{R}, \mathsf{P}_1, \ldots, \mathsf{P}_{K-1} \rangle(crs, \phi, \Sigma)$ the distribution of the possible transcripts of the protocol in this setting (*i.e.*, the sequence of all messages broadcasted by the players during an execution of the protocol), where the probabilities are taken over the random coins T_{P} of the players $\mathsf{P} \in \{\mathsf{R}, \mathsf{P}_1, \ldots, \mathsf{P}_{K-1}\}$ and the random choice of the CRS *crs*.

Definition 23.2.1 (($\varepsilon, \delta, c, n$)-Anonymous Transfer). *A K-party $(\varepsilon, \delta, c, n)$-Anonymous Transfer (AT) for $\varepsilon, \delta \in \mathbb{R}_{[0,1]}$ and $K, c, n \in \mathbb{N}$ is a tuple containing two PPT algorithms (*Setup*,* Reconstruct*) and a PPT protocol (*Transfer*). The number of rounds in the* Transfer *protocol is given as c and the bitlength n defines the length of the transferred message Σ. The algorithms are defined as follows:*

Setup(1^κ) *takes as input the security parameter 1^κ in unary encoding and outputs a Common Reference String crs.*

Transfer(crs, ϕ, Σ) *defines a c-round protocol[1] that takes as input the Common Reference String crs, an index $\phi \in [K-1]$ specifying the sender, and the message $\Sigma \in \{0,1\}^n$ from the sender and outputs a transcript τ. The non-sender send independent uniformly distributed noise in each round. All protocol messages sent by the receiver, the sender and the non-participating parties at each round are bitstrings of length $m = m(\kappa)$, where m is implicitly specified by the* Transfer *protocol.*

Reconstruct$(crs, \tau, T_{\mathsf{R}})$ *is a local algorithm executed by the receiver that takes as input the CRS crs, the protocol transcript τ and the receiver's random tape T_{R} and outputs a message Σ'.*

The algorithms additionally satisfy the ε-correctness and the δ-anonymity properties defined in Definitions 23.2.2 and 23.2.3.

Definition 23.2.2 (ε-Correctness). *An Anonymous Transfer protocol Π_{AT^n} between players $(\mathsf{P}_1, \ldots, \mathsf{P}_{K-1}, \mathsf{R})$ is ε-*correct *if for any $\phi \in [K-1]$, any $\Sigma \in \{0,1\}^n$, and any $crs \leftarrow$* Setup(1^κ),

$$\Pr\left[\ \Sigma = \Sigma' \ \middle|\ \begin{array}{l} \tau \overset{\$}{\leftarrow} \mathsf{Transfer}_{\langle \mathsf{R}, \mathsf{P}_1, \ldots, \mathsf{P}_{K-1} \rangle}(crs, \phi, \Sigma) \\ \Sigma' \leftarrow \mathsf{Reconstruct}(crs, \tau, T_{\mathsf{R}}) \end{array} \right] \geq \varepsilon \tag{23.1}$$

Note that ε can take on any value between 0 and 1. The naive algorithm that lets the receiver sample a uniformly random n-bit string has $\varepsilon = 1/2^n$.

Definition 23.2.3 (δ-Anonymity). *An Anonymous Transfer protocol Π_{AT^n} between players $(\mathsf{P}_1, \ldots, \mathsf{P}_{K-1}, \mathsf{R})$ is δ-*anonymous *if for any PPT algorithm $\mathsf{A}_{guess} = (\mathsf{A}_{guess_0}, \mathsf{A}_{guess_1})$, it holds that*

$$\left| \Pr_{\phi \overset{\$}{\leftarrow} [K-1]} \left[\mathit{Exp}^{anon}_{\Pi_{AT^n}, \mathcal{A}, \phi}(\kappa) = \phi \right] - \frac{1}{K-1} \right| \leq (1-\delta) \cdot \frac{K-2}{K-1} \tag{23.2}$$

where $\mathit{Exp}^{anon}_{\Pi_{AT^n}, \mathcal{A}, \phi}(\kappa)$ is defined in Fig. 23.1.

[1] A c-round protocol corresponds to a synchronous model, where each message is broadcast and the messages in each round only depend on messages from previous rounds.

```
**Experiment** $\text{Exp}^{\text{anon}}_{\Pi_{\text{AT}^n},\mathcal{A},\phi}(\kappa)$
$crs \xleftarrow{\$} \text{Setup}(1^\kappa)$
$T_{\text{R}} \leftarrow \{0,1\}^{\text{poly}(\kappa)}$
$(\Sigma, st) \leftarrow \text{A}_{guess_0}(crs, T_{\text{R}})$
$\tau \xleftarrow{\$} \text{Transfer}_{\langle \text{R},\text{P}_1,\ldots,\text{P}_{K-1}\rangle}(crs, \phi, \Sigma; T_{\text{R}}, \cdot, \cdot)$
**return** $\text{A}_{guess_1}(\tau, T_{\text{R}}, st)$
```

Figure 23.1.: Definition of the anonymity experiment $\text{Exp}^{\text{anon}}_{\Pi_{\text{AT}^n},\mathcal{A},\phi}(\kappa)$.

The value δ can take any value between 0 and 1. The higher δ the stronger the provided anonymity guarantees. If a protocol is $\delta = 1$-anonymous, the advantage over guessing at random equals 0, and if a protocol is $\delta = 0$-anonymous, the advantage over guessing at random equals 1.

Note that we require anonymity to hold, in particular, against the receiver. Therefore, the adversary in the anonymity game may know the receiver's random tape T_{R} from the beginning.

The guessing algorithm is split between A_{guess_0} who is given the CRS and the random tape T_{R} the receiver is going to use during the protocol, and outputs the target message Σ that should be transferred and a state st. In the second phase, the algorithm A_{guess_1} which is given the inputs τ and the state.

Unless stated otherwise, we consider the case $K = 3$, *i.e.*, one non-participant.

23.3. Fine-grained Anonymous Transfer

Fine-grained cryptographic primitives are only secure against adversaries with an a-priori bounded runtime which is greater than the runtime of the honest algorithms, [109, 60]. We use the notion of [60]. In the following, FC_1 and FC_2 are function classes.

Definition 23.3.1 (FC_1-fine-grained $(\varepsilon, \delta, c, n)$-Anonymous Transfer against FC_2). *The tuple* (Setup, Transfer, Reconstruct) *(as defined in Definition 23.2.1) is a* FC_1*-fine-grained* $(\varepsilon, \delta, c, n)$*-Anonymous Transfer for* $\varepsilon, \delta \in \mathbb{R}_{[0,1]}$ *and* $c, n \in \mathbb{N}$ *against* FC_2 *if the following two conditions hold:*

Efficiency. *The algorithms* (Setup, Transfer, Reconstruct) *are in* FC_1.

Security. *Anonymity (Definition 23.2.3) is only required to hold against adversaries in* FC_2.

The definition of correctness remains as in Definition 23.2.2.

Example 23.3.2 (Merkle-Puzzles). *Merkle-Puzzles [109] are a fine-grained protocol to exchange a shared key from symmetric encryptions where successful encryptions can be efficiently distinguished from false ones. The sender S creates n_{mer} many ciphertexts, each under a different (relatively short) key, containing a unique identifier and a symmetric key. The receiver R then randomly picks one of the ciphertexts and runs a brute-force attack (which we assume to cost m_{mer} many steps) to recover the key and to send the identifier back to the sender.*

Here $\mathsf{FC}_1 := O(n_{mer} + m_{mer})$ as the sender has to create n_{mer} puzzles and the receiver must use m_{mer} steps to break one of them, and $\mathsf{FC}_2 := o(n_{mer} \cdot m_{mer})$ as an adversary has to break at worst all the n_{mer} ciphertexts to recover the key.

23.4. Trivial Anonymous Transfers

In this section we introduce two example ATs. Those are taken verbatim from Agrikola, Couteau, and Maier [5, Section 3.4]. For simplicity, we focus on 3-party anonymous transfer in the following discussions, with two players $\mathsf{P}_0, \mathsf{P}_1$ and a receiver R.

Remark 23.4.1 (Perfect correctness.). *A perfectly correct (i.e. $\varepsilon = 1$) protocol is impossible. Given a player P_ϕ with input Σ, there is always a probability that the non-participating player $\mathsf{P}_{1-\phi}$ behaves exactly as a participating player with input $\Sigma' \neq \Sigma$, in which case R is not guaranteed to output the correct message Σ.*

Therefore, the best one can hope for is a correctness statistically close to 1. In the following, we demonstrate ATs with trivial parameters.

Example 23.4.2 (Trivial single-bit AT). *Consider the following trivial single-round AT to transfer a single bit b: P_ϕ broadcasts the input bit b (and $\mathsf{P}_{1-\phi}$ broadcasts a random bit). Upon receiving (b_0, b_1) from P_0 and P_1, if $b_0 = b_1$, R outputs b_0; otherwise, R outputs a uniformly random bit. As $\mathsf{P}_{1-\phi}$ broadcasts a random bit, it holds that $\Pr[b_0 = b_1] = 1/2$. In this case R always obtains the correct output $b = b_\phi$. Otherwise, if $b_0 \neq b_1$, R obtains the correct output with probability $1/2$ due to the involved guessing. Thus, R obtains the correct output with overall probability $3/4$. The protocol is $1/2$-anonymous since the adversary knows the message to be transmitted and can hence determine the sender whenever the transmitted bits are distinct and guess with probability $1/2$ otherwise. Hence, the above protocol is a $(3/4, 1/2, 1, 1)$-AT.*

Example 23.4.3 (Trivial n-bit AT). *One can also construct a trivial n-bit AT. To transmit a message $\Sigma \in \{0,1\}^n$: P_ϕ simply sends Σ repeated κ times. Clearly, (not only) R finds out both Σ and ϕ with overwhelming probability. Hence, the above protocol is a $(1 - \mathsf{negl}(\kappa), \mathsf{negl}(\kappa), \kappa \cdot n, n)$-AT.*

In this work, we study whether ATs with non-trivial parameters can exist.

23.5. Reductions Among AT Protocols

This section does not count as contribution to this thesis but was taken verbatim from Agrikola, Couteau, and Maier [5, Section 3.5] due to its relevance. In this section, we show that several simplified variants of anonymous transfer are equivalent to the original definition.

23.5.1. AT implies silent-receiver AT.

We say that an anonymous transfer has *silent receiver* if the receiver never sends messages during the Transfer protocol, and Reconstruct is a deterministic function based on the CRS and the transcript τ. Any AT directly implies a silent-receiver AT with the same parameters for correctness and anonymity, but at the cost of secrecy: Any (non-)participant is able to reconstruct the message given only the transcript of broadcasted messages, not just the receiving party of the protocol. This might be undesirable for practical applications. Let Π_{AT^n} be a $(\varepsilon, \delta, c, n)$-Anonymous Transfer. Define the Silent Receiver Anonymous Transfer Π_{SRAT} as follows:

$\Pi_{SRAT}.\mathrm{Setup}(1^\kappa)$ runs $crs \leftarrow \Pi_{\mathrm{AT}^n}.\mathrm{Setup}(1^\kappa)$ and samples a uniform random tape T_{R} for R. It outputs (crs, T_{R}).

$\Pi_{SRAT}.\mathrm{Transfer}(crs, \phi, \Sigma)$ proceeds exactly as $\Pi_{\mathrm{AT}^n}.\mathrm{Transfer}(crs, \phi, \Sigma)$, except that the receiver R does not broadcast any message. At each round $\chi = 1$ to $\chi = c$, the sender P_ϕ locally appends the χ-th receiver message msg_χ in $\Pi_{\mathrm{AT}^n}.\mathrm{Transfer}(crs, \phi, \Sigma; T_{\mathrm{R}}, \cdot, \cdot)$ to the current transcript $\tau[\chi]$ (note that msg_χ can be computed deterministically from $\tau[\chi]$ and T_{R}), and compute its next message as in $\Pi_{\mathrm{AT}^n}.\mathrm{Transfer}$ using the transcript $\tau[\chi] \| msg_\chi$.

$\Pi_{SRAT}.\mathrm{Reconstruct}(crs, \tau, T_{\mathrm{R}})$ is defined exactly as $\Pi_{\mathrm{AT}^n}.\mathrm{Reconstruct}(crs, \tau, T_{\mathrm{R}})$, except that it first expands the transcript τ by recomputing (deterministically) the messages of R in $\Pi_{\mathrm{AT}^n}.\mathrm{Transfer}(crs, \phi, \Sigma; T_{\mathrm{R}}, \cdot, \cdot)$ and appending them to τ at each round.

The notion of silent receiver AT captures the notion of an anonymous transfer whose aim is to *publicly reveal* a message (*i.e.*, whistleblowing) rather than sending it to a single receiver.

Lemma 23.5.1. *Π_{SRAT} is an $(\varepsilon, \delta, c, n)$-Anonymous Transfer.*

Proof (sketch). Correctness and number of rounds follow directly from the description of Π_{SRAT}, which simply mimics Π_{AT^n}, except that the random tape of the receiver is made public, and its messages are computed on the fly locally by the sender and during the reconstruction. Anonymity follows also immediately by observing that T_{R} is given to the adversary in the anonymity game, hence making it public cannot harm anonymity. $\square$

Since the converse direction is straightforward, AT and silent receiver AT are therefore equivalent.

23.6. Strong Anonymous Transfer

For several practical applications we require the additional property that the real message can only be extracted by the real receiver. This gives rise to a stronger definition of AT:

Definition 23.6.1 (Strong $(\varepsilon, \delta, \varsigma, c, n)$-Anonymous Transfer)**.** *A strong $(\varepsilon, \delta, \varsigma, c, n)$-Anonymous Transfer is defined analogously to a $(\varepsilon, \delta, c, n)$-Anonymous Transfer from Definition 23.2.1 but additionally satisfies ς-secrecy from Definition 23.6.2.*

The additional property we require formally states that without the receivers random tape, the message cannot be successfully reconstructed from the transcript. We focus on the 3-party case, since case captures the primitive we will need to construct undetectable oblivious transfer. However, we stress that the definition can easily be generalized to any number of parties.

Definition 23.6.2 (ς-Secrecy)**.** *An Anonymous Transfer protocol Π_{AT^n} between players $(\mathsf{P}_0, \mathsf{P}_1, \mathsf{R})$ is ς-secret if for any PPT reconstruction algorithm A_{guess}, any $\Sigma \in \{0,1\}^n$, and any $crs \leftarrow \mathsf{Setup}(1^\kappa)$, it holds that:*

$$\begin{aligned} &\left| \Pr\left[\Sigma = \Sigma' \;\middle|\; \begin{array}{l} \tau \leftarrow \mathsf{Transfer}_{\langle \mathsf{R}, \mathsf{P}_0, \mathsf{P}_1 \rangle}(crs, \phi, \Sigma) \\ \Sigma' \leftarrow \mathsf{A}_{guess}(crs, \tau) \end{array} \right] - \frac{1}{2^n} \right| \\ \leq &(1-\varsigma) \cdot \frac{2^n - 1}{2^n} \end{aligned} \tag{23.3}$$

The secrecy is a value between 0 and 1, where $\varsigma = 0$ implies that A_{guess} can reconstruct Σ with absolute certainty and $\varsigma = 1$ means that Σ can at best be guessed. Note that secrecy requires T_{R} to remain secret. Hence, the transformation to silent receiver AT from Section 23.5 does not preserve secrecy.

24. Impossibility of Anonymous Transfer

In this section, we prove that no Anonymous Transfer protocol, with an arbitrary polynomial number of rounds, can simultaneously enjoy overwhelming correctness ($\varepsilon = 1 - \mathsf{negl}(\kappa)$) and overwhelming anonymity ($\delta = 1 - \mathsf{negl}(\kappa)$), even for transmitting single bit messages.

Theorem 24.0.1 (Impossibility of AT). *Let $\mu \colon \mathbb{N} \mapsto \mathbb{R}$ be any negligible function and p be any polynomial. There is no $(1 - \mu(\kappa), 1 - \mu(\kappa), p(\kappa), 1)$-Anonymous Transfer, for any number of parties.*

Theorem 24.0.1 will follow as a corollary from a more general result bounding the relation between ε and δ in any c-round protocol. In Sections 24.1 and 24.2 we will focus on $K = 3$, that is, the case with one dummy player. This is without loss of generality as we will show at Section 24.3 by proving that this impossibility also holds for any K-party Anonymous Transfer with $K > 3$.

We remark that this section contains parts that do not count as a contribution of this thesis but that originate from the paper this part of the thesis is based on. Namely, we do not claim Sections 24.1 and 24.2 (the impossibility result for $K = 3$) as our contribution but rather copy it from Agrikola, Couteau, and Maier [5, Section 4] and use it to motivate our later construction in Section 24.3 (the extension from $K = 3$ to $K > 3$) and Chapters 25 and 26 (the fine-grained constructions).

24.1. The Attacker

From now on, we focus on building a generic attack against 3-party *silent-receiver* Anonymous Transfer for κ-bit messages. The theorem will follow from the reductions from 1-bit Anonymous Transfer to multi-bit silent-receiver Anonymous Transfer described in Section 23.5.

Let Π_{AT^κ} be a silent-receiver $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer. Let $m = m(\kappa)$ be the bitlength of the message from the non-participating party. Let Rand denote the following procedure: on input a transcript τ of Π_{AT^κ}, $\mathsf{Rand}(\tau)$ truncates τ to $c - 1$ rounds of the AT protocol, and replaces the messages of the last round by two uniformly random length-m bitstrings[1]. It

[1] Since the protocol is silent-receiver, there is no message from the receiver; furthermore, assuming that the sender message is m-bit is without loss of generality, since otherwise the protocol is trivially not anonymous

```
Algorithm $\mathsf{A}_{guess_0}^{\Xi}(crs)$
$(\Sigma_1, \cdots, \Sigma_\Xi) \overset{\$}{\leftarrow} \mathsf{QD}_{\mathcal{R}}$
$\Sigma^\Xi \overset{\$}{\leftarrow} \{0,1\}^\kappa \setminus \{\Sigma_1, \cdots, \Sigma_\Xi\}$    / Exists as $\Xi \ll 2^\kappa$
return $(\Sigma^\Xi, st = (crs, \Sigma^\Xi))$
```

Figure 24.1.: Definition of the first part of the guessing algorithm, $\mathsf{A}_{guess_0}^{\Xi}$, against the δ-anonymity of the silent-receiver κ-bit AT protocol Π_{AT^κ}, parameterized by a polynomial $\Xi = \Xi(\kappa)$.

outputs the new rerandomized transcript τ'. For every $\Sigma \in \{0,1\}^\kappa$ and $\phi \in \{0,1\}$, we let $D_{\phi,\Sigma}, D'_{\phi,\Sigma}, \mathsf{QD}_{\mathcal{R}}$ denote the following distribution:

$$\begin{aligned}
D_{\phi,\Sigma} &= \left\{ \Sigma' \;:\; \begin{array}{l} crs \leftarrow \mathsf{Setup}(1^\kappa), \\ \tau \leftarrow \mathsf{Transfer}(\phi, \Sigma), \\ \Sigma' \leftarrow \mathsf{Reconstruct}(crs, \tau) \end{array} \right\} \\
D'_{\phi,\Sigma} &= \left\{ \Sigma' \;:\; \begin{array}{l} crs \leftarrow \mathsf{Setup}(1^\kappa), \\ \tau' \leftarrow \mathsf{Rand}(\mathsf{Transfer}(\phi, \Sigma)), \\ \Sigma' \leftarrow \mathsf{Reconstruct}(crs, \Sigma') \end{array} \right\} \\
\mathsf{QD}_{\mathcal{R}} &= \left\{ \Sigma' \;:\; \begin{array}{l} crs \leftarrow \mathsf{Setup}(1^\kappa), \\ \tau \overset{\$}{\leftarrow} (\{0,1\}^m \times \{0,1\}^m)^c, \\ \Sigma' \leftarrow \mathsf{Reconstruct}(crs, \tau) \end{array} \right\}
\end{aligned} \tag{24.1}$$

Fix an arbitrary polynomial Ξ. We define an attacker $\mathcal{A}^\Xi = (\mathsf{A}_{guess_0}^{\Xi}, \mathsf{A}_{guess_1}^{\Xi})$ against the anonymity of Π_{AT^κ}, parameterized by the polynomial Ξ, on Figs. 24.1 and 24.2. In the following, we will not use $\mathcal{A}^\Xi$ directly to attack the full c-round protocol: rather, we will use $\mathcal{A}^\Xi$ as a distinguisher between the c-round protocol Π_{AT^κ}, and the $(c-1)$-round protocol obtained by running Π_{AT^κ} for $(c-1)$ rounds, and replacing the messages of the last round by uniformly random m-bit strings. From there, the proof of impossibility will proceed by induction; we refer the reader to the introduction for a high-level intuition of our proof.

Base case: advantage of $\mathcal{A}^\Xi$ when $c = 1$. We start the induction by bounding the advantage of $\mathcal{A}^\Xi$ in the anonymity game when Π_{AT^κ} is non-interactive (*i.e.*, Transfer consists of a single message from each of $\mathsf{P}_0, \mathsf{P}_1$ to the receiver). Before proceeding, we make two key observations:

(1) When $c = 1$, $D'_{\phi,\Sigma} = \mathsf{QD}_{\mathcal{R}}$ for any (ϕ, Σ). In particular, this means that $D'_{\phi,\Sigma}$ is independent of (ϕ, Σ).

```
Algorithm $\mathsf{A}_{guess_1}^{\Xi}(\Sigma, st, \tau)$
$(crs, \Sigma^{\Xi}) := st$
$(\tau[c-1], (msg_0, msg_1)) := \tau$
$(msg_0', msg_1') \xleftarrow{\$} \{0,1\}^m \times \{0,1\}^m$
$\tau_0 := (\tau[c-1], (msg_0, msg_1'))$
$\tau_1 := (\tau[c-1], (msg_0', msg_1))$
for $\phi^* \in \{0,1\}$ do
    $\Sigma'_{\phi^*} \leftarrow \mathsf{Reconstruct}(crs, \tau_{\phi^*})$
done
if $\Sigma'_0 = \Sigma^{\Xi}$ then
    return $0$
elseif $\Sigma'_1 = \Sigma^{\Xi}$ then
    return $1$
else
    $\phi' \xleftarrow{\$} \{0,1\}$
    return $\phi'$
fi
```

Figure 24.2.: Definition of the second part of the guessing algorithm, $\mathsf{A}_{guess_1}^{\Xi}$, against the δ-anonymity of the silent-receiver κ-bit AT protocol Π_{AT^κ}, parameterized by a polynomial $\Xi = \Xi(\kappa)$.

(2) When $c = 1$ and $\phi = 0$, the distribution of the values (Σ_0', Σ_1') constructed by $\mathsf{A}_{guess_1}^{\Xi}$ given as input a random transcript $\tau \leftarrow \mathsf{Transfer}(0, \Sigma^{\Xi})$ is exactly the distribution $D_{0,\Sigma^{\Xi}} \times \mathrm{QD}_{\mathcal{R}}$. This is because msg_0 is a random message from the sender with input $\phi = 0$ and value Σ^{Ξ}, and (msg_1, msg_0', msg_1') are three uniformly random elements of $\{0,1\}^m$, hence (msg_0, msg_1') is exactly a random transcript of Π_{AT^κ} with (ϕ, Σ^{Ξ}), while (msg_0', msg_1) is just a pair of random messages. Similarly, if $\phi = 1$, the distribution of the values (Σ_0', Σ_1') constructed by $\mathsf{A}_{guess_1}^{\Xi}$ given as input a random transcript $\tau \leftarrow \mathsf{Transfer}(1, \Sigma^{\Xi})$ is exactly the distribution $\mathrm{QD}_{\mathcal{R}} \times D_{1,\Sigma^{\Xi}}$.

Both observations follow directly from the definitions of $D_{\phi,\Sigma}$, $D'_{\phi,\Sigma}$, $\mathrm{QD}_{\mathcal{R}}$ and of $\mathsf{A}_{guess_1}^{\Xi}$. Building on the above observations, we show that for an appropriate choice of Ξ, the advantage of $\mathcal{A}^{\Xi}$ in the anonymity game can be made arbitrarily close to $(\varepsilon - 1)/2$:

Claim 24.1.1. *For any polynomial Ξ^*, there is a polynomial Ξ such that*

$$\left| \Pr_{\phi \xleftarrow{\$} \{0,1\}} \left[Exp^{anon}_{\Pi_{AT^\kappa}, \mathcal{A}^{\Xi}, \phi}(\kappa) = \phi \right] - 1/2 \right| \geq \frac{\varepsilon}{2} - \frac{1}{\Xi^*}, \tag{24.2}$$

which implies that any silent-receiver $(\varepsilon, \delta, 1, \kappa)$-Anonymous Transfer must satisfy $\delta \leq 1 - \varepsilon + 2/\Xi^*$ for any polynomial Ξ^*; equivalently, $\delta \leq 1 - \varepsilon + \mathsf{negl}(\kappa)$. In particular, this means that if the AT has overwhelming correctness ($\varepsilon = 1 - \mathsf{negl}(\kappa)$), then δ must be negligible.

Proof. Let $\Xi \leftarrow \kappa \cdot \Xi^*$, and let $\mathsf{Bad} \subset \{0,1\}^\kappa$ denote the set

$$\mathsf{Bad} = \left\{ \Sigma \in \{0,1\}^\kappa \ : \ \Pr_{\Sigma' \xleftarrow{\$} D'}[\Sigma' = \Sigma] > \tfrac{2}{\Xi^*} \right\}.$$

We first show that

$$\Pr_{\Sigma^\Xi \xleftarrow{\$} \mathsf{A}_{guess_0}^{\Xi}(crs)}\left[\Sigma^\Xi \in \mathsf{Bad}\right] \leq e^{-\kappa}.$$

Fix any $\Sigma \in \mathsf{Bad}$. Then

$$\begin{aligned}
\Pr_{\Sigma^\Xi \xleftarrow{\$} \mathsf{A}_{guess_0}^{\Xi}(crs)}[\Sigma = \Sigma^\Xi] &\leq \Pr_{\Sigma_1, \cdots, \Sigma_\Xi \xleftarrow{\$} D'}[\Sigma \notin \{\Sigma_1, \cdots, \Sigma_\Xi\}] \\
&= \Pr_{\Sigma_1, \cdots, \Sigma_\Xi \xleftarrow{\$} D'}\left[\bigwedge_{i=1}^{\Xi} (\Sigma_i \neq \Sigma)\right] \\
&< \left(1 - \frac{2}{\Xi^*}\right)^{\Xi} \leq e^{-2\Xi/\Xi^*} = e^{-2\kappa}.
\end{aligned}$$

Therefore, by a union bound over all $\Sigma \in \mathsf{Bad}$, $\Pr_{\Sigma^\Xi \xleftarrow{\$} \mathsf{A}_{guess_0}^{\Xi}(crs)}[\Sigma^\Xi \in \mathsf{Bad}] \leq |\mathsf{Bad}| \cdot e^{-2\kappa} \leq 2^\kappa \cdot e^{-2\kappa} \leq e^{-\kappa}$. Now, denoting ϕ the identity of the sender, we bound the success probability of $\mathcal{A}^\Xi = (\mathsf{A}_{guess_0}^{\Xi}, \mathsf{A}_{guess_1}^{\Xi})$ in guessing ϕ.

Consider first the case $\phi = 0$. Then by observation (2), the values (Σ'_0, Σ'_1) form a random sample from $D_{0,\Sigma^\Xi} \times D'$ (in particular, they are independent samples from these two distributions). Now, by definition of $\mathsf{A}_{guess_1}^{\Xi}$, we have

$$\begin{aligned}
\Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa}, \mathcal{A}^\Xi, \phi}(\kappa) = 0 \ \middle|\ \phi = 0\ \right] &= \Pr[(\Sigma'_0 = \Sigma^\Xi) \vee ((\Sigma'_0 \neq \Sigma^\Xi) \wedge \phi' = 0)] \\
&= \Pr[\Sigma'_0 = \Sigma^\Xi] + \frac{1}{2} \cdot \Pr[\Sigma'_0 \neq \Sigma^\Xi] \\
&= \frac{1}{2} \cdot \left(1 + \Pr[\Sigma'_0 = \Sigma^\Xi]\right) \geq \frac{1+\varepsilon}{2},
\end{aligned}$$

where the second equality is because the events are disjoint, and the last inequality is by the ε-correctness of the AT. Now, consider the case $\phi = 1$; in this case, the values (Σ'_0, Σ'_1)

computed by $A_{guess_1^{\Xi}}(\tau, st)$ are distributed exactly as a random sample from $QD_{\mathcal{R}} \times D_{1,\Sigma^{\Xi}}$. Then, if we condition on Σ^{Ξ} being outside of the set Bad, we have

$$\begin{aligned}
&\Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{t},\phi}(\kappa) = 1 \;\middle|\; \phi = 0, \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&= \Pr\left[\, ((\Sigma_0' \neq \Sigma^{\Xi}) \wedge (\Sigma_1' = \Sigma^{\Xi})) \vee ((\Sigma_0' \neq \Sigma^{\Xi}) \wedge (\Sigma_1' \neq \Sigma^{\Xi}) \wedge \phi' = 1) \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&= \Pr\left[\, (\Sigma_0' \neq \Sigma^{\Xi}) \wedge (\Sigma_1' = \Sigma^{\Xi}) \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&\qquad\qquad + \frac{1}{2} \cdot \Pr\left[\, (\Sigma_0' \neq \Sigma^{\Xi}) \wedge (\Sigma_1' \neq \Sigma^{\Xi}) \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&= \Pr\left[\, \Sigma_0' \neq \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \cdot \Pr\left[\, \Sigma_1' = \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&\qquad\qquad + \frac{1}{2} \cdot \Pr\left[\, \Sigma_0' \neq \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \cdot \Pr\left[\, \Sigma_1' \neq \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \\
&= \frac{1}{2} \cdot \Pr\left[\, \Sigma_0' \neq \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \cdot \left(1 + \Pr\left[\, \Sigma_1' = \Sigma^{\Xi} \;\middle|\; \Sigma^{\Xi} \notin \mathrm{Bad} \,\right]\right) \\
&\geq \left(1 - \frac{2}{\Xi^*}\right) \cdot \frac{1+\varepsilon}{2},
\end{aligned}$$

where the third equality follows from the fact that Σ_0' and Σ_1' are independent samples, and the last inequality follows from ε-correctness and the definition of the set Bad. Now, since

$$\begin{aligned}
&2 \cdot \Pr[\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = \phi] \\
&= \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 0 \;\middle|\; \phi = 0 \,\right] \\
&\quad + \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 1 \;\middle|\; \phi = 1 \,\right] \\
&\geq \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 0 \;\middle|\; \phi = 0 \,\right] \\
&\quad + \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 1 \;\middle|\; \phi = 1, \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \cdot \Pr[\Sigma^{\Xi} \notin \mathrm{Bad}] \\
&\geq \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 0 \;\middle|\; \phi = 0 \,\right] \\
&\quad + \Pr\left[\, \mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 1 \;\middle|\; \phi = 1, \Sigma^{\Xi} \notin \mathrm{Bad} \,\right] \cdot (1 - e^{-\kappa}),
\end{aligned}$$

we get

$$\begin{aligned}
\Pr[\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = \phi] &\geq \frac{1}{2} \cdot \left(\frac{1+\varepsilon}{2} + \left(1 - \frac{2}{\Xi^*}\right) \cdot \frac{1+\varepsilon}{2} \cdot (1 - e^{-\kappa})\right) \\
&\geq \frac{1+\varepsilon}{2} \cdot \left(1 - \frac{1}{\Xi^*}\right) \cdot (1 - e^{-\kappa}) \\
&\geq \frac{1+\varepsilon}{2} - \frac{1}{\Xi^*},
\end{aligned}$$

where the last inequality uses the fact that $(1+\varepsilon) \cdot (1 - e^{-\kappa})/2\Xi^* \leq 1/\Xi^*$. This concludes the proof. □

Induction case. Fix a polynomial number of rounds $c(\kappa)$ and assume that any silent-receiver $(\varepsilon, \delta, c-1, \kappa)$-Anonymous Transfer must satisfy

$$\frac{1-\delta}{2} \geq \frac{1}{c-1} \cdot \left(\frac{\varepsilon}{2} - \frac{1}{\Xi^*}\right) \text{ for any polynomial } \Xi^*.$$

Equivalently, this means that $(1-\delta)/2 \geq \varepsilon/2(c-1) - \mathsf{negl}(\kappa)$. Let Π_{AT^κ} be any silent-receiver $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer.

Claim 24.1.2. *For any polynomial* Ξ^*,

$$\frac{1-\delta}{2} \geq \frac{1}{c} \cdot \left(\frac{\varepsilon}{2} - \frac{1}{\Xi^*}\right). \tag{24.3}$$

Proof. We have

$$\begin{aligned}\Pr[\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = \phi] =& \frac{1}{2} \cdot \Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = 0 \ \middle|\ \phi = 0\ \right] \\ &+ \frac{1}{2} \cdot \Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = 1 \ \middle|\ \phi = 1\ \right].\end{aligned}$$

We bound separately the two probabilities on the right hand side:

$$\begin{aligned}&\Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = 0 \ \middle|\ \phi = 0\ \right] \\ &= \Pr[(\Sigma'_0 = \Sigma^\Xi) \vee ((\Sigma'_0 \neq \Sigma^\Xi) \wedge \phi' = 0)] \\ &= \Pr[\Sigma'_0 = \Sigma^\Xi] + \frac{1}{2} \cdot \Pr[\Sigma'_0 \neq \Sigma^\Xi] \\ &= \frac{1}{2} \cdot \left(1 + \Pr[\Sigma'_0 = \Sigma^\Xi]\right) \geq \frac{1+\varepsilon}{2},\end{aligned}$$

where the second equality is because the events are disjoint, and the last inequality is by the ε-correctness of the AT. Note that nothing really changes when $\phi = 0$ compared to the base case, because the independence of Σ'_0 and Σ'_1 needs only to be invoked in the case $\phi = 1$ (this is because the attacker "favors Σ'_0" in case of tie). Now, consider the case $\phi = 1$. In this case, the values (Σ'_0, Σ'_1) computed by $\mathsf{A}_{guess_1^\Xi}(\tau, st)$ are distributed as *correlated samples* from D'_{1,Σ^Ξ} and D_{1,Σ^Ξ}. We have

$$\begin{aligned}&\Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = 1 \ \middle|\ \phi = 1\ \right] \\ &= \Pr[((\Sigma'_0 \neq \Sigma^\Xi) \wedge (\Sigma'_1 = \Sigma^\Xi)) \vee ((\Sigma'_0 \neq \Sigma^\Xi) \wedge (\Sigma'_1 \neq \Sigma^\Xi) \wedge \phi' = 1)] \\ &= \Pr[(\Sigma'_0 \neq \Sigma^\Xi) \wedge (\Sigma'_1 = \Sigma^\Xi)] + \frac{1}{2} \cdot \Pr[(\Sigma'_0 \neq \Sigma^\Xi) \wedge (\Sigma'_1 \neq \Sigma^\Xi)] \\ &= \frac{1}{2} \cdot \left(\Pr[(\Sigma'_0 \neq \Sigma^\Xi) \wedge (\Sigma'_1 = \Sigma^\Xi)] + \Pr[\Sigma'_0 \neq \Sigma^\Xi]\right),\end{aligned}$$

where the second equality follows from the fact that the events $\Sigma'_1 = \Sigma^{\Xi}$ and $\Sigma'_1 \neq \Sigma^{\Xi}$ are disjoint. Now, observe that

$$\begin{aligned}\Pr[\Sigma'_1 = \Sigma^{\Xi}] = {} & \Pr\left[\ \Sigma'_1 = \Sigma^{\Xi} \ \middle|\ \Sigma'_0 = \Sigma^{\Xi}\ \right] \cdot \Pr[\Sigma'_0 = \Sigma^{\Xi}] + \Pr[\Sigma'_1 = \Sigma^{\Xi} \wedge \Sigma'_0 \neq \Sigma^{\Xi}] \\ & \leq \Pr[\Sigma'_0 = \Sigma^{\Xi}] + \Pr[\Sigma'_1 = \Sigma^{\Xi} \wedge \Sigma'_0 \neq \Sigma^{\Xi}].\end{aligned}$$

By construction, $\Pr[\Sigma'_1 = \Sigma^{\Xi}]$ is exactly the probability that Reconstruct outputs Σ^{Ξ} given a random transcript for $\phi = 1$ and transmitted value Σ^{Ξ}. By ε-correctness of the protocol, we therefore have $\Pr[\Sigma'_1 = \Sigma^{\Xi}] \geq \varepsilon$. Plugging this into the previous inequality, we get

$$\Pr[\Sigma'_1 = \Sigma^{\Xi} \wedge \Sigma'_0 \neq \Sigma^{\Xi}] \geq \varepsilon - \Pr[\Sigma'_0 = \Sigma^{\Xi}],$$

which implies that

$$\begin{aligned}&\Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{AT}^{\kappa}},\mathcal{A}^{\Xi},\phi}(\kappa) = 1 \ \middle|\ \phi = 1\ \right] \geq \frac{1}{2} \cdot \left(\varepsilon - \Pr[\Sigma'_0 = \Sigma^{\Xi}] + \Pr[\Sigma'_0 \neq \Sigma^{\Xi}]\right) \\ &= \frac{\varepsilon + 1}{2} - \Pr[\Sigma'_0 = \Sigma^{\Xi}].\end{aligned}$$

It remains to bound $\Pr[\Sigma'_0 = \Sigma^{\Xi}]$. To do so, we define a *round-reduced* Anonymous Transfer protocol (Setup′, Transfer′, Reconstruct′), built on top of $\Pi_{\mathsf{AT}^{\kappa}}$:

Setup′ = Setup

Transfer′(crs, ϕ, Σ): proceeds as Transfer, except that the parties interrupt the protocol Transfer(crs, ϕ, Σ) after $c - 1$ rounds, obtaining a transcript τ. In the $(c - 1)$'s round, P_0 and P_1 both additionally broadcast $2m$ random bits, denoted (r_0, r_1). Define the final transcript of Transfer′ to be (τ, r_0, r_1).

Reconstruct′$(crs, (\tau, r_0, r_1))$: on input a transcript (τ, r_0, r_1), set $r \longleftarrow r_0 \oplus r_1$ and parse r as a concatenation of two m-bit messages (msg_0, msg_1). Let τ' be a c-round transcript where the transcript of the first $c - 1$ rounds is τ, and the last round transcript is (msg_0, msg_1). Output Reconstruct(crs, τ').

A transcript for the round-reduced protocol is exactly a transcript for $\Pi_{\mathsf{AT}^{\kappa}}$ where the last two messages have been replaced by uniform random strings[2]. Now, observe that when $\phi = 1$, replacing only the message of ϕ_1 by random in the last round still leads to the exact same distribution over transcript. In other terms, $\Pr[\Sigma'_0 = \Sigma^{\Xi}]$ is exactly the probability that Reconstruct′ outputs Σ^{Ξ} given a transcript for the round-reduced protocol. Of course, the round-reduced protocol is δ-anonymous[3], and has $c - 1$ rounds. Hence, by the induction hypothesis, we can bound the correctness of the round-reduced protocol:

$$\begin{aligned}\frac{1-\delta}{2} &\geq \frac{1}{c-1} \cdot \left(\frac{\Pr[\Sigma'_0 = \Sigma^{\Xi}]}{2} - \frac{1}{\Xi^*}\right) \\ \Longrightarrow\ \Pr[\Sigma'_0 = \Sigma^{\Xi}] &\leq (c-1) \cdot (1-\delta) + 2/\Xi^*.\end{aligned}$$

[2] We define these strings to be the XOR of two strings sent in the previous rounds by P_0 and P_1 to guarantee that Reconstruct remains deterministic; this is just a syntactic manipulation which is not crucial, but simplifies the rest of the exposition.

[3] $\Pi_{\mathsf{AT}^{\kappa}}$ is δ-anonymous; truncating the transcript can trivially not decrease anonymity.

Therefore, we get

$$\Pr\left[\ \mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = 1 \ \middle|\ \phi = 1\ \right] \geq \frac{\varepsilon+1}{2} - (c-1)\cdot(1-\delta) - \frac{2}{\Xi^*}.$$

Putting everything together, we obtain

$$\begin{aligned}\Pr[\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = \phi] &\geq \frac{1}{2}\cdot\left(\frac{\varepsilon+1}{2} + \frac{\varepsilon+1}{2} - (c-1)\cdot(1-\delta) - \frac{2}{\Xi^*}\right)\\ &= \frac{\varepsilon+1}{2} - \frac{(c-1)\cdot(1-\delta)}{2} - \frac{1}{\Xi^*},\end{aligned}$$

hence

$$\left|\Pr[\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = \phi] - \frac{1}{2}\right| \geq \frac{\varepsilon}{2} - \frac{(c-1)\cdot(1-\delta)}{2} - \frac{1}{\Xi^*}.$$

Now, by definition of δ-anonymity, we also have

$$\begin{aligned}\frac{1-\delta}{2} &\geq \left|\Pr[\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^\kappa},\mathcal{A}^\Xi,\phi}(\kappa) = \phi] - \frac{1}{2}\right|\\ &\geq \frac{\varepsilon}{2} - \frac{(c-1)\cdot(1-\delta)}{2} - \frac{1}{\Xi^*}\\ \implies c\cdot\frac{1-\delta}{2} &\geq \frac{\varepsilon}{2} - \frac{1}{\Xi^*},\end{aligned}$$

which concludes the proof of the claim. □

24.2. Putting the Pieces Together

With the above analysis, we showed that for any silent-receiver $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer, it must necessarily hold that $(1-\delta)/2 \geq \varepsilon/2c - \mathsf{negl}(\kappa)$. Since any $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer implies a silent-receiver $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer (with the exact same parameters, see Section 23.5), we obtain:

Corollary 24.2.1. *Any $(\varepsilon, \delta, c, \kappa)$-Anonymous Transfer must satisfy*

$$\frac{1-\delta}{2} \geq \frac{\varepsilon}{2c} - \mathsf{negl}(\kappa).$$

In particular, this implies that there exists no κ-bit AT with overwhelming correctness and anonymity, for any polynomial number of rounds.

Furthermore, as shown in Section 23.5, any *single-bit* c-round AT with correctness $\varepsilon = 1 - \mathsf{negl}(\kappa)$ and anonymity $\delta = 1 - \mathsf{negl}(\kappa)$ implies a κ-bit AT with correctness $\varepsilon' = \varepsilon^\kappa = (1-\mathsf{negl}(\kappa))^\kappa = 1-\mathsf{negl}(\kappa)$, and anonymity $\delta' = (\delta-1)\cdot\kappa-\delta+2 = 1-\mathsf{negl}(\kappa)$. Combining this reduction with Corollary 24.2.1 concludes the proof of Theorem 24.0.1.

Protocol Π'_{AT}

A c-round Anonymous Transfer Protocol Π'_{AT} for $(K-1)$ Parties.

On input (ϕ, b), P_ϕ follows the role of P_ϕ on input (ϕ, b) according to Π_{AT}.

On input $\perp$, $\mathsf{P}_{\phi'}$ for $\phi' \in [K-2] \setminus \phi$ follows the role of $\mathsf{P}_{\phi'}$ on input $\perp$ according to Π_{AT}.

On input $\perp$, R follows the role of P_{K-1} on input $\perp$ according to Π_{AT}.

For each round χ from 1 to c :

- P_ϕ computes the next message according to P_ϕ in Π_{AT} with previous messages $((msg_1^{(1)}, \dots, msg_{K-2}^{(1)}, msg_{\mathsf{R}}^{(1)}), \dots, (msg_1^{(\chi-1)}, \dots, msg_{K-2}^{(\chi-1)}, msg_{\mathsf{R}}^{(\chi-1)}))$.
- $\mathsf{P}_{\phi'}$ for $\phi' \in [K-2] \setminus \phi$ broadcasts $msg_{\phi'}^{(\chi)} \xleftarrow{\$} \{0,1\}^m$.
- R broadcasts $msg_{\mathsf{R}}^{(\chi)} \xleftarrow{\$} \{0,1\}^m$.

R outputs $\Pi_{AT}.\mathsf{Reconstruct}((msg_1^{(1)}, \dots, msg_{K-2}^{(1)}, msg_{\mathsf{R}}^{(1)}), \dots, (msg_1^{(c)}, \dots, msg_{K-2}^{(c)}, msg_{\mathsf{R}}^{(c)}))$.

Figure 24.3.: The protocol Π'_{AT} that constructs $(K-1)$-party AT from a given K-party SRAT for K parties.

24.3. Impossibility of Anonymous Transfer for $K > 3$

In this section we will expand the impossibility proof we showed previously from $K = 3$ to an arbitrary $K > 3$. For that purpose, we prove the following lemma:

Lemma 24.3.1. *Let Π_{AT} be an (ε, δ) Silent Receiver Anonymous Transfer protocol for K parties. Then, protocol Π'_{AT} defined in Fig. 24.3 that makes black-box access to Π_{AT} is an (ε, δ') AT protocol for $(K-1)$ parties, where $\delta' > 1 - (1-\delta) \cdot \frac{K-2}{K-3}$.*

Combining this with the insight from Lemma 23.5.1 that (ε, δ)-AT imply (ε, δ)-SRAT, this implies that an (ε, δ)-AT for K parties implies an (ε, δ')-SRAT for $(K-1)$ parties.

We then proceed to show that if δ is overwhelming, then δ' is overwhelming. The correctness ε does not change by our transformation. Hence, we can iteratively apply this step for *any* number of parties K. Therefore, if there is a K-party $(\mathsf{owhl}(\kappa), \mathsf{owhl}(\kappa))$-AT, then there is also a three-party $(\mathsf{owhl}(\kappa), \mathsf{owhl}(\kappa))$-AT. Since we have proven the latter to be impossible in Corollary 24.2.1, a K-party AT with overwhelming correctness *and* anonymity cannot exist for any polynomial number of parties K.

To prove our claim we construct a $(K-1)$-party AT Π'_{AT} as shown in Fig. 24.3. The protocol is based on an (ε, δ)-SRAT Π_{AT}. All parties in Π'_{AT} follow their respective roles in Π_{AT}, which means that the sender is constructing *special* messages as advised by Π_{AT} whereas the dummy friends only broadcast random bits. To compensate for the missing party (the K-th party that does not exist in the $K-1$ party protocol) the messages of the K-th party are sent by the receiver, who takes on the role of one dummy friend. This is possible because the K-party protocol is a *Silent Receiver Anonymous Transfer* where the receiver sends no messages, whereas in the $(K-1)$-party protocol the receiver is allowed to report messages in each round. Then parties continue with the *extended* transcript and proceed to the next round.

24.3.1. Security Analysis of Π'_{AT}

Correctness. Correctness of Π'_{AT} follows directly from correctness of Π_{AT}.

Lemma 24.3.2 (Correctness). *Let Π_{AT} be a K-party (ε, δ)-SRAT. Then Π'_{AT} has correctness ε.*

Proof. This trivially follows from the fact that the extended transcript is distributed identically to an execution of Π_{AT}: The messages of the first $(K-1)$ parties are distributed correctly by design, as they follow the original protocol, and the messages of the K-th party are also distributed in the same way, but accounted to a different party. Thus, the reconstruction algorithm is queried with the same inputs. □

Anonymity. Anonymity requires some work, since one party (the non-silent receiver) in Π'_{AT} is guaranteed to not be the sender.

Lemma 24.3.3 (Anonymity). *Let Π_{AT} be a K-party (ε, δ)-SRAT. Then Π'_{AT} has anonymity $\delta' = 1 - (1-\delta) \cdot \frac{K-2}{K-3}$.*

Proof. We want to find a bound δ' for the δ'-anonymity of Π'_{AT}, that is, a value δ' for which the following holds for any PPT algorithm A'_{guess}:[4]

$$\Pr[A'_{guess} \text{ wins}] \leq \frac{1}{(K-2)} + (1-\delta') \cdot \frac{K-3}{K-2} \tag{24.4}$$

Now let A'_{guess} be an adversary on the anonymity of the $(K-1)$-party protocol Π'_{AT}. We will now demonstrate how A'_{guess} can be used by a guessing algorithm A_{guess} to break the anonymity of Π_{AT}:

[4] Ignoring the absolute value here is without loss of generality as we want to bound the *best* possible guessing algorithm, which would have $\Pr[A_{guess} \text{ wins}] - 1/(K-2) > 0$.

Upon receiving the challenge transcript τ_C for the K-party AT, A_{guess} picks a uniformly random party from the group of potential senders (*i.e.*, among everyone except the receiver) and *erases* that party from the transcript. A_{guess} reports this party's messages as coming from the receiver in Π'_{AT}. Observe that the resulting transcript is perfectly distributed as an honest transcript of Π'_{AT} (given that a non-participant was erased). Now, A_{guess} feeds this simulated transcript to A'_{guess}, and outputs whatever A'_{guess} outputs.

If A_{guess} selects a non-participant—which is the case with probability $(K-2)/(K-1)$ as the party is uniformly sampled among all parties except for the receiver and there is only one actual sender—then the modified transcript is a valid transcript of Π'_{AT} and the probability that A'_{guess} returns the correct sender is given by probability $\Pr[A'_{guess} \text{ wins}]$ that A'_{guess} wins the anonymity-game. Otherwise, if A_{guess} selected the actual sender, then all possible return values for A'_{guess} are dummy friends and the probability that A_{guess} is correct equals 0.

Hence, the overall probability that A_{guess} wins the anonymity game is given by the following term:

$$\begin{aligned} \Pr[A_{guess} \text{ wins}] &= \frac{K-2}{K-1} \cdot \Pr[A'_{guess} \text{ wins}] \\ \iff \Pr[A'_{guess} \text{ wins}] &= \frac{K-1}{K-2} \cdot \Pr[A_{guess} \text{ wins}] \end{aligned} \tag{24.5}$$

Due to the δ-anonymity of Π_{AT} it holds that

$$\Pr[A_{guess} \text{ wins}] \leq \frac{1}{K-1} + (1-\delta) \cdot \frac{K-2}{K-1} \tag{24.6}$$

By merging Eqs. (24.5) and (24.6) we get that

$$\begin{aligned} \Pr[A'_{guess} \text{ wins}] &= \frac{K-1}{K-2} \cdot \Pr[A_{guess} \text{ wins}] \\ &\leq \frac{K-1}{K-2}\left(\frac{1}{K-1} + (1-\delta) \cdot \frac{K-2}{K-1}\right) \\ \implies \Pr[A'_{guess} \text{ wins}] &\leq \frac{1}{K-2} + (1-\delta) \end{aligned} \tag{24.7}$$

However, to apply δ'-anonymity we need Eq. (24.7) to be in the form from Eq. (24.4), which means that we have to set:

$$\Pr[A'_{guess} \text{ wins}] \leq \frac{1}{K-2} + (1-\delta) \overset{!}{=} \frac{1}{K-2} + (1-\delta')\frac{K-3}{K-2} \tag{24.8}$$

Solving Eq. (24.8) for δ' yields the following result:

$$\delta' = 1 - (1-\delta) \cdot \frac{K-2}{K-3} \tag{24.9}$$

For this value of δ', the guessing algorithm A'_{guess} on Π'_{AT} cannot be used to violate the δ-anonymity of Π_{AT}.

In contrast, if there is some guessing algorithm A'_{guess} which violates the δ'-anonymity for the above value of δ', then it holds that:

$$\Pr[\mathsf{A}'_{guess} \text{ wins}] > \frac{1}{K-2} + (1-\delta') \cdot \frac{K-3}{K-2} \tag{24.10}$$

Then the probability that A_{guess} can successfully determine the sender in the K-party AT is given as follows:

$$\begin{aligned} \Pr[\mathsf{A}_{guess} \text{ wins}] &= \frac{K-2}{K-1} \cdot \Pr[\mathsf{A}'_{guess} \text{ wins}] \\ &> \frac{K-2}{K-1} \cdot \left(\frac{1}{K-2} + (1-\delta') \cdot \frac{K-3}{K-2}\right) \\ \Pr[\mathsf{A}_{guess} \text{ wins}] &> \frac{1}{K-1} + (1-\delta') \cdot \frac{K-3}{K-1} \end{aligned} \tag{24.11}$$

By inserting the computed value of δ' from Eq. (24.9) into Eq. (24.11) and get:

$$\Pr[\mathsf{A}_{guess} \text{ wins}] > \frac{1}{K-1} + \left(1 - \left(1 - (1-\delta) \cdot \frac{K-2}{K-3}\right)\right) \cdot \frac{K-3}{K-1} \tag{24.12}$$

So, in total A_{guess} would be successful with probability

$$\begin{aligned} \Pr[\mathsf{A}_{guess} \text{ wins}] &> \frac{1}{K-1} + \left(1 - 1 + (1-\delta) \cdot \frac{K-2}{K-3}\right) \cdot \frac{K-3}{K-1} \\ \Longrightarrow \Pr[\mathsf{A}_{guess} \text{ wins}] &> \frac{1}{K-1} + (1-\delta) \cdot \frac{K-2}{K-1} \end{aligned} \tag{24.13}$$

which would contradict the assumed δ-anonymity of Π_{AT}. □

From Lemmas 24.3.2 and 24.3.3 the claim from Lemma 24.3.1 follows, which yields the following corollary:

Corollary 24.3.4. *Let Π_{AT} be an (ε, δ) Anonymous Transfer protocol for K parties where $\varepsilon, \delta \in \mathsf{owhl}(\kappa)$. Then there is a protocol three-party (ε, δ')-AT protocol Π'_{AT} where $\varepsilon, \delta \in \mathsf{owhl}(\kappa)$.*

Proof. Let the number K of parties be given. From Π_{AT} we can construct a Silent Receiver Anonymous Transfer with the same values for ε and δ, so the constructed Silent Receiver Anonymous Transfer still has overwhelming correctness and anonymity.

By then applying Lemma 24.3.1 it follows that there is a protocol Π'_{AT} for $(K-1)$ parties which still has the same (hence overwhelming) correctness and anonymity $\delta' > 1 - (1 - \delta) \cdot \frac{K-2}{K-3}$.

As we assume that $\delta \in \mathsf{owhl}(\kappa)$ it holds that $(1-\delta) \in \mathsf{negl}(\kappa)$, and since $\frac{K-2}{K-3} \in \mathsf{poly}(\kappa)$ it holds that $(1-\delta) \cdot \frac{K-2}{K-3} \in \mathsf{negl}(\kappa)$.

So it total it holds that $\delta' > 1 - \text{negl}(\kappa) \in \text{owhl}(\kappa)$.

By then alternating between an application of Lemma 23.5.1 to construct a Silent Receiver Anonymous Transfer from the normal AT and Lemma 24.3.1 to reduce the number of parties by one we can proceed until $K = 3$, where still it holds that both correctness and anonymity are overwhelming. □

This insight, however, contradicts Corollary 24.2.1, which states that no three-party AT with overwhelming correctness and anonymity can exist, which leads to our final corollary:

Corollary 24.3.5. *Let κ be the security parameter. Let $K \in \text{poly}(\kappa)$ be the number of individuals present in an execution of Anonymous Transfer. Then for any K-party AT protocol Π_{AT} it holds that Π_{AT} is not a $(\text{owhl}(\kappa), \text{owhl}(\kappa))$-AT.*

Proof. We prove this corollary by contradiction. Assuming that Π_{AT} is an $(\text{owhl}(\kappa), \text{owhl}(\kappa))$-AT means that due to Corollary 24.3.4, there is an $(\text{owhl}(\kappa), \text{owhl}(\kappa))$-AT for three parties. However, such a protocol cannot exist due to Corollary 24.2.1. □

24.4. Extensions and Limitations

The adversary in our impossibility result makes a black-box use of an arbitrary 3-party silent receiver multi-bit Anonymous Transfer; the reduction to K-party single-bit Anonymous Transfer is black-box as well. In particular, this means that our impossibility result relativizes: it remains true relative to any oracle, where access to the oracle is granted to all participants and all algorithms (including the adversary).

In the next section, we will provide a heuristic construction of *fine-grained* Anonymous Transfer. The aim of this construction is to complement our impossibility result, and to draw an interesting and surprising picture: Anonymous Transfer appears to be impossible to realize with the standard superpolynomial cryptographic hardness gaps, but becomes feasible if one settles for a small polynomial hardness gap. Our fine-grained construction is described and formally proven secure using an ideal obfuscation scheme; instantiating the scheme with candidate indistinguishability obfuscation schemes gives a plausible heuristic construction (the same way that instantiating the random oracle model with standard hash functions gives plausible heuristic constructions of various cryptographic primitives, when the construction is not pathological). Because our impossibility result relativizes, in contrast, standard Anonymous Transfer remains provably impossible relative to an ideal obfuscation oracle (while fine-grained Anonymous Transfer, as we will see, provably exist relative to such an oracle).

Impossibility of fine-grained multi-bit AT with overwhelming correctness and anonymity. In the multi-bit setting, where the sender wants to transmit $\omega(\log \kappa)$ bits to the receiver, our result further demonstrates that there exists no *fine-grained* Anonymous Transfer with overwhelming correctness and anonymity $(1 - \mathrm{negl}(\kappa))$, even with an *arbitrary small polynomial gap* between the runtime of the honest parties and that of the adversary. Indeed, let $r = O(c \cdot m)$ be a lower bound on the runtime of the honest parties (r is the total number of bits sent by the sender, hence it is a clear lower bound on its running time), and consider an adversary $\mathcal{A}^{\Xi}$ with $\Xi = \kappa \cdot c^g$, where $g > 0$ is an arbitrarily small constant. Then by construction, the runtime of $\mathcal{A}^{\Xi}$ is $O(\kappa \cdot r \cdot c^g) \leq O(\kappa \cdot r^{1+g})$ (as it is dominated by the cost of sampling Ξ random transcripts for $\mathsf{A}_{guess_0^{\Xi}}$). Then this adversary satisfies

$$\frac{1-\delta}{2} \geq \left| \Pr\left[\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{\mathrm{AT}^\kappa}, \mathcal{A}^{\Xi}, \phi}(\kappa) = \phi \right] - \frac{1}{2} \right| \geq \frac{1}{c} \cdot \left(\frac{\varepsilon}{2} - \frac{1}{c^g} \right), \tag{24.14}$$

which implies that δ and ε cannot be simultaneously equal to $1 - \mathrm{negl}(\kappa)$ (since $1/(2c) - 1/c^{1+g}$ cannot be a negligible function for any polynomial c and any constant $g > 0$).

24.4.1. Limitations of the impossibility result.

Even putting aside the heuristic security guarantee of our fine-grained construction (or its security in an idealized model), a gap remains between our impossibility result and our construction: our impossibility result does not rule out the possibility of having, say, a $(1 - \mathrm{negl}(\kappa), 1 - 1/c, c, \kappa)$-Anonymous Transfer – that is, an Anonymous Transfer with overwhelming correctness, and vanishing anonymity error $1/c$ in c rounds, with standard (superpolynomial) security. In contrast, our heuristic construction only achieves overwhelming correctness and anonymity arbitrarily close to $1/c$ against *fine-grained* adversaries. It is an interesting open question to close this gap. We conjecture that the true answer is negative:

Conjecture 24.4.1. *There exists no $(1 - \mathrm{negl}(\kappa), 1 - 1/c, c, \kappa)$-Anonymous Transfer.*

What follows assumes that the reader is familiar with standard philosophical considerations on the worlds of Impagliazzo. Proving the above conjecture would have a very interesting (theoretical) consequence: it would demonstrate the existence of a natural cryptographic primitive that plausibly exists within the realm of fine-grained cryptography, yet is impossible with standard hardness gap. It is known that fine-grained constructions sometimes allow building "high-end" cryptographic primitives in "low-end" cryptographic realms. For example, Merkle puzzles, which can be instantiated under exponentially strong one-way functions [26], provide a fine-grained key exchange; borrowing Impagliazzo's terminology [89], this places "fine-grained Cryptomania" inside (a strong form of) Minicrypt. Proving the conjecture would induce a comparable result, but at the highest level of the hierarchy: it would, in a sense, place fine-grained Impossibilitopia (a world of cryptographic primitives so powerful that they simply cannot exist) inside Obfustopia.

25. Fine-Grained AT from Ideal Obfuscation

25.1. The Protocol

In this section we aim to circumvent the impossibility result from Chapter 24. Recall that we have proven that no *asymptotically secure* Anonymous Transfer protocol with *overwhelming* anonymity and correctness can exist. Thus we relax two factors and provide a realization of Anonymous Transfer in the *fine-grained* security setting according to Definition 23.3.1 that only provides *polynomial* anonymity. More precisely, we construct a c-round protocol which achieves anonymity δ, where the honest parties have runtime in $\mathsf{FC}_1 := O(c)$ against adversaries in $\mathsf{FC}_2 := o(c^2(1-\delta))$, where $c = c(\kappa)$ is a polynomial in κ. While this sounds easy at first, we show that even in this seemingly trivial setting the task is highly non-trivial.

For the sake of simplicity we introduce the protocol with $K = 3$, implying a single dummy friend. However, expanding this protocol to an arbitrary $K \in \mathcal{N}$ is straightforward as the behavior of all dummy friends is the same by definition and instead of two messages, each round now contains $K - 1$ messages.

We exploit the limited runtime of the adversary and provide a protocol in Fig. 25.1 with c rounds. In each new round (or with each valid sender message) the probability that the correct bit is eventually returned increases, *i.e.*, each valid round increases the receiver's confidence in the received message. Each round lets the sender compute a signature σ_ϕ using a sEUF-CMA secure signature scheme[1] for the transcript of the previous round. The transferred bit b and the signature σ_ϕ are then sent. The verification key for the signature scheme is transmitted by the sender in the first round. In order to make the messages look random the message is not sent directly. Instead, the sender encrypts the message using an IND\$-CCA secure encryption scheme[2], [119]. As not every bitstring of length m is a valid ciphertext, we use a special function Dec^* instead of the normal function Dec, which is defined as follows: If Dec on input ct returns $\perp$ then Dec^* returns $\mathsf{PRF}(ct)$, otherwise Dec^* returns $\mathsf{Dec}(ct)$. Hence, every possible input allows an interpretation as a cleartext. We use those for both the asymmetric and symmetric schemes.

In order to make the output unusable for any other party, the receiver draws a One Time Pad as first message which eventually masks the final output, and a verification key of a signature scheme. The latter is used to ensure that the receiver *approves* with the

[1] See Section 22.5 for a definition of sEUF-CMA.
[2] See Section 22.4 for a definition of IND\$-CCA.

Protocol Π_{AT}

Protocol Π_{AT} for realizing Anonymous Transfer in the fine-grained setting. It is running with a set of 3 parties $(\mathsf{P}_0, \mathsf{P}_1, \mathsf{R})$ where one of P_0 and P_1 acts as sender and R is the receiver.

It is parameterized by an IND\$-CCA-secure *public-key* encryption scheme $\textsc{Pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, a *multi-challenge* IND\$-CPA-secure *symmetric* encryption scheme $\textsc{Ske} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an EUF-CMA-secure signature scheme $\textsc{Sig}\colon \{0,1\}^* \times \{0,1\}^\kappa \mapsto \{0,1\}^m$, and an obfuscated program P_{FG-AT} from Fig. 25.2.

On activation, R draws $otp \overset{\$}{\leftarrow} \{0,1\}$ and computes $(\mathsf{k}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$. Then R sets $msg_\mathsf{R}^{(0)} \leftarrow \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (otp, \mathsf{vk}_\mathsf{R}))$ and broadcasts $msg_\mathsf{R}^{(0)}$.

On input (ϕ, b), P_ϕ computes a signature key pair $(\mathsf{vk}_\phi, \mathsf{k}_\phi) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$ and a symmetric key $\mathsf{sk}_\phi \leftarrow \textsc{Ske}.\mathsf{KeyGen}(1^\kappa)$.

Then, P_ϕ computes a signature $\sigma_\phi \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\phi, msg_\mathsf{R}^{(0)})$ and broadcasts $msg_\phi^{(0)} \leftarrow \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (\mathsf{sk}_\phi, \mathsf{vk}_\phi)) \| \textsc{Ske}.\mathsf{Enc}(\mathsf{sk}_\phi, b)$.

Upon activation , $\mathsf{P}_{1-\phi}$ sets uniformly random $msg_{\bar{\phi}}^{(0)}$.

For each round χ **from** 1 **to** c :

P_ϕ computes $\sigma_\phi \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\phi, (msg_0^{(\chi-1)}, msg_1^{(\chi-1)}))$ and sets $msg_\phi^{(\chi)} \leftarrow \textsc{Ske}.\mathsf{Enc}(\mathsf{sk}_\phi, (b, \sigma_\phi))$.

$\mathsf{P}_{1-\phi}$ broadcasts $msg_{1-\phi}^{(\chi)} \overset{\$}{\leftarrow} \{0,1\}^m$.

R computes $\sigma_\mathsf{R} \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\mathsf{R}, (msg_\mathsf{R}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(c)}, msg_1^{(c)})))$, compute $b' := \mathsf{P}_{FG-AT}(msg_\mathsf{R}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(c)}, msg_1^{(c)}), \sigma_\mathsf{R})$ and output $otp \oplus b'$.

Figure 25.1.: The protocol Π_{AT} for fine-grained Anonymous Transfer.

transcript; after the two potential senders provided all messages, the receiver *signs* the entire transcript and only if this signature verifies the entire previous transcript, the circuit continues. The first message of the receiver is broadcast, while the signature is only used locally. This ensures that only the receiver can obtain a usable output, as other parties are unable to compute the correct signature on the transcript.

The receiver obtains its output by inserting the final transcript alongside the signature into an obfuscated circuit which is supplied in a common reference string. The circuit is obfuscated using ideal obfuscation[3]. It hides a PRF key and a secret decryption key sk_P for

[3] See Section 22.6 for a definition of ideal obfuscation.

```
Program $\mathrm{P}_{FG-AT}[\mathrm{pk}_\mathrm{P}, c]\left(msg_\mathrm{R}^{(0)}, \left\{msg_0^{(\chi)}, msg_1^{(\chi)}\right\}_{\chi=0}^{c}\right)$

$(otp, \mathrm{vk}_\mathrm{R}) := \mathrm{PKE.Dec}^*(\mathrm{sk}_\mathrm{P}, msg_\mathrm{R}^{(0)})$,
$(\mathrm{sk}_0, \mathrm{vk}_0) := \mathrm{PKE.Dec}^*(\mathrm{sk}_\mathrm{P}, msg_0^{(0)}[1\colon m])$,
$(b_0, \sigma_0) := \mathrm{SKE.Dec}^*(\mathrm{sk}_0, msg_0^{(0)}[m+1\colon 2m])$,
$(\mathrm{sk}_1, \mathrm{vk}_1) := \mathrm{PKE.Dec}^*(\mathrm{sk}_\mathrm{P}, msg_1^{(0)}[1\colon m])$,
$(b_1, \sigma_1) := \mathrm{SKE.Dec}^*(\mathrm{sk}_1, msg_1^{(0)}[m+1\colon 2m])$,
if $\neg\mathrm{SIG.Vfy}(\mathrm{vk}_\mathrm{R}, (msg_\mathrm{R}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(c)}, msg_1^{(c)})))$ then :
   return $\mathrm{CointossS}_{(0.5)}^{(\tau)}(0, 1)$
$\chi_0 := [\![\mathrm{SIG.Vfy}(\sigma_0, \mathrm{vk}_0, msg_\mathrm{R}^{(0)})]\!] \cdot (c+1)$,
$\chi_1 := [\![\mathrm{SIG.Vfy}(\sigma_1, \mathrm{vk}_1, msg_\mathrm{R}^{(1)})]\!] \cdot (c+1)$,
foreach $\chi \in \{1, \ldots, c\}$ do:
   foreach $\phi \in \{\phi' | \phi' \in \{0,1\}, \chi_\phi = (c+1)\}$ do:
      $X_\phi := \mathrm{SKE.Dec}^*(\mathrm{sk}_\phi, msg_\phi^{(\chi)})$,    $b'_\phi := X_\phi[0]$,    $\sigma_\phi := X_\phi[1 : |X_\phi|]$
      if $\neg\mathrm{SIG.Vfy}(\sigma_\phi, \mathrm{vk}_\phi, \tau[\chi-1]) \vee b_\phi \neq b'_\phi$ then :
         $\chi_\phi := \chi$    / Remember first bad round.
$\phi' := \mathrm{argmax}_\phi(\chi_\phi)$
return $otp \oplus \mathrm{CointossS}_{(1/2\cdot(1+\chi_{\phi'}/c))}^{(\tau)}(b_{\phi'}, (1-b_{\phi'}))$
```

Figure 25.2.: Obfuscated program P_{FG-AT} for a single-bit Anonymous Transfer in the fine-grained setting with c rounds.

the IND\$-CCA secure PKE. The corresponding encryption key pk_P is also part of the CRS and, hence, known to all parties. This encryption scheme is used by the sender and the receiver to hide their respective *first* message. This uniquely determines the symmetric key used to decrypt the remaining messages of each potential sender. The message also contains a verification key used to sign the previous messages in future rounds, the bit that the sender wants to transfer, and the initial signature on the receivers message. The remaining rounds of the sender are encrypted using a *symmetric* scheme, namely the IND\$-CCA secure SKE scheme, using the key transferred to the circuit in the first round.

The circuit is shown in Fig. 25.2. It starts by extracting the verification keys and symmetric encryption keys (one per potential sender) alongside the bits that the respective party wants to transfer and the initial signatures on the first receiver messages from the respective initial messages of both parties, and the receivers OTP and verification key from the receiver message. Then the circuit starts by verifying the signature of the receiver on the entire transcript, and if that does not match, returns a uniformly random bit[4]. Otherwise, if the receiver's signature is valid, the circuit searches for the first *faulty* round of each potential sender. That is, the first round of each potential sender where the decrypted message does not have the expected format—meaning that either signature on the previous round fails to verify or the encoded bit differs from the bit extracted from the initial message. The party who sent the most consecutive valid rounds is selected as the sending party. The circuit outputs the bit transmitted by that party with probability depending on the ratio between valid sender messages and the total number of rounds, which ranges between $1/2$ (*i.e.*, a uniformly random bit) if no round was valid for any party and 1 (*i.e.*, deterministically returning the correct bit) if all rounds were correct. However, as stated before, the circuit does not output that bit directly, but instead *masks* it using the OTP extracted from the receiver's first message. This ensures secrecy, as other parties only get a masked output which information-theoretically hides the actual bit.

25.2. Security Analysis

In the following we will analyze the security of this protocol. In particular, we will analyze its correctness in Section 25.2.1, the anonymity in Section 25.2.2, and the secrecy in Section 25.2.3.

25.2.1. Correctness

In this subsection we will prove the following theorem:

Theorem 25.2.1 (Correctness)**.** *If the protocol from Fig. 25.1 is instantiated with an Ideally Obfuscated version of the circuit from Fig. 25.2 the protocol is ε-correct with $\varepsilon = (1 - \mathrm{negl}(\kappa))$.*

Proof. At a high level, at the end of an honest protocol execution, the maximum round in which a valid signature has been provided equals the number of rounds c. With overwhelming probability, the sending parties' input is the only one that contains c many valid rounds. Hence, the correctly masked bit is returned. Since the mask is input by the receiver and later applied to the output, the receiver obtains the correctly masked bit.

More formally, in a correct execution, there is one sender P_ϕ who encodes the bit b_ϕ in the first round. Every round that follows has to have a valid signature σ on both messages of the previous round. With honest parties doing that, the only way that correctness is not given is if the circuit confuses $\mathsf{P}_{\bar{\phi}}$ to be P_ϕ. This only happens if the inputs sent by $\mathsf{P}_{\bar{\phi}}$ are interpreted as valid inputs, causing the **argmax** to return the bit of $\mathsf{P}_{\bar{\phi}}$.

[4] This is denoted in the figure by the $\mathrm{CointossS}^{(\tau)}_{(p)}(b, \overline{b})$ function, which returns b, *i.e.* the first argument, with probability p, and $\overline{b}$, *i.e.* the second argument, with the complementary probability $(1 - p)$, where the randomness for p is extracted from the argument provided by τ.

Since we assume the sending party to act honestly, the value of χ_ϕ is c which is as high as this variable can go. Thus, if the **argmax** ever returns $(\bar{\phi})$, then $\chi_{\bar{\phi}} = c$ as well. This can only happen if $\mathsf{P}_{\bar{\phi}}$ has sent for c rounds a valid encryption of $b_{\bar{\phi}} := (1 - b_\phi)$ (if we assume a non-correct output, the bit has to differ, naturally) and a valid signature on $\tau[\chi - 1]$ in each of the c rounds.

By requirement, Pke has sparse ciphertexts. With $\mathsf{P}_{\bar{\phi}}$ sending uniformly random ciphertexts, the probability that this is a valid ciphertext is negligible in κ.

Assuming that $\mathsf{P}_{\bar{\phi}}$ had a valid ciphertext by chance, the probability that it encodes the same bit that was decrypted in the first round is given by $1/2$; yet the probability that the signature verifies the previous round is negligible.

Now we have χ rounds. In each round we have a probability of $\mathrm{negl}(\kappa) \cdot 1/2 \cdot \mathrm{negl}(\kappa)$ that the message is valid. So for $\mathsf{P}_{\bar{\phi}}$ to accidentally send valid messages of $\bar{b_\phi}$, the probability is given by $(\mathrm{negl}(\kappa) \cdot 1/2 \cdot \mathrm{negl}(\kappa))^c$. Additionally, in round 0 there has to be a valid encryption of $\bar{b_\phi}$, which also happens with negligible probability as once more we have the sparseness of Pke. And in the negligible case that the message counts as cipher, there is only a probability of 1/2 that the message encodes the correct bit.

Thus, in an honest protocol execution, the output of the circuit is $b'_\phi = otp \oplus b_\phi$ with overwhelming probability. Thus by outputting $otp \oplus b'_\phi = otp \oplus otp \oplus b_\phi = b_\phi$ the receiver will output the correct bit with overwhelming probability.

Thus according to the definition of ε-correctness from Eq. (23.1) we get $\varepsilon \in (1 - \mathrm{negl}(\kappa))$. This concludes our proof. □

25.2.2. Anonymity.

In this section we aim to analyze the anonymity of our protocol. In particular, we try to prove the following theorem:

Theorem 25.2.2 (Anonymity)**.** *Let Pke be an IND\$-CCA secure asymmetric encryption scheme, let Ske be a tightly secure multi-challenge IND\$-CCA secure symmetric encryption scheme, let Sig be an sEUF-CMA secure signature scheme, let $\mathbb{O}$ be an ideal obfuscator, let PRF be a secure PRF, and let κ be the security parameter. Then the c-round protocol Π_{AT} satisfies δ-anonymity for all adversaries in* $\mathsf{FC}_2 := \mathrm{o}(c^2(1 - \delta))$.

On a high level, the proof is structured into two parts. In the first part, we modify the anonymity game $\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{AT},\mathcal{A},\phi}(\kappa)$ and the obfuscated circuit oracle P_{FG-AT} to remove as much computationally hidden information about ϕ as possible. More precisely, we exploit the non-malleability of Pke and the sEUF-CMA security of Sign to unnoticeably alter the oracle to determine the number of valid rounds by counting how many rounds of the input transcript are identical to the challenge transcript provided by $\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{AT},\mathcal{A},\phi}(\kappa)$. The first round which is not entirely identical to the challenge transcript (*i.e.* either the sender message or the non-sender message differ) increases the valid rounds count only if the input sender message is identical to the challenge sender message or if the input sender message decrypts to the same content as the challenge sender message. The following round will be counted as invalid since the signature verification will fail. After this step, the decryption keys of SKE and PKE are not necessary for chosen-ciphertext simulation anymore. Then, we first replace the sender messages which are encrypted using SKE and then the first round sender message which is encrypted using PKE with uniform randomness exploiting the IND\$-CCA security of both encryption schemes.

The only information about the bit ϕ that is left in the present game is due to the oracle which counts valid sender messages by comparing the input sender message with the challenge sender message. Clearly, the final modification of the game must be the removal of this dependency on ϕ. However, this removal will noticeably alter the output distribution of the oracle. Hence, an adversary with arbitrary polynomial runtime will be able to distinguish this hop with constant probability [47]. However, if we can limit the runtime of the adversary to be sub-quadratic in the runtime of the honest protocol execution, we are able to apply results from distribution testing to achieve a good bound for this distinguishing advantage. We will elaborate on this final game hop in more detail below and will refer to the second last game as $\text{GAME}_7(\kappa)$ and to the last game (*i.e.* the game, where no information about ϕ remains) as $\text{GAME}_8(\kappa)$.

We proceed over a series of games.

$\text{GAME}_1(\kappa)$: This is the original game (after replacing the obfuscated circuit with oracle access to the circuit, the PRF with an actual random oracle and the adversary with the simulator), where the sending party P_ϕ is chosen uniformly at random.

$\text{GAME}_2(\kappa)$: This game follows $\text{GAME}_1(\kappa)$, but during the simulation of the oracle P_{FG-AT} from Fig. 25.1 the simulation enforces correctness of the challenge transcript τ_C: if the input transcript τ matches the challenge transcript τ_C, it returns $otp_C \oplus b_C$. Due to Theorem 25.2.1, no PPT adversary can distinguish between $\text{GAME}_1(\kappa)$ and $\text{GAME}_2(\kappa)$.

Lemma 25.2.3 (Indistinguishability of $\text{GAME}_1(\kappa)$ and $\text{GAME}_2(\kappa)$)**.** *Let PKE be an IND\$-CCA secure asymmetric encryption scheme, let SKE be a tightly secure multi-challenge IND\$-CCA secure symmetric encryption scheme, let SIG be an sEUF-CMA secure signature scheme, let $\mathbb{O}$ be an ideal obfuscator, and let* PRF *be a secure PRF. Then, for all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_1(\kappa)$ and $\text{GAME}_2(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{GAME_1(\kappa)} = 1] - \Pr[\mathtt{out}_{GAME_2(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability automatically follows from the fact that both simulated circuits behave identical with overwhelming probability due to Theorem 25.2.1. □

$\text{GAME}_3(\kappa)$: This game follows $\text{GAME}_2(\kappa)$ but during simulation of the circuit the adversary aborts if any of the first-round messages from P_0 or P_1 differ from the messages reported in the challenge transcript *and* the decryptions still yield the same verification key or symmetric key. This game hop is justified by the non-malleability of PKE.

Lemma 25.2.4 (Indistinguishability of $\text{GAME}_2(\kappa)$ and $\text{GAME}_3(\kappa)$)**.** *Let PKE be an IND\$-CCA secure asymmetric encryption scheme, let SKE be a tightly secure multi-challenge IND\$-CCA secure symmetric encryption scheme, let SIG be an sEUF-CMA secure signature scheme, let $\mathbb{O}$ be an ideal obfuscator, and let* PRF *be a secure PRF. Then, for all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_2(\kappa)$ and $\text{GAME}_3(\kappa)$ is bounded by:*

$$|\Pr[\texttt{out}_{\text{GAME}_2(\kappa)} = 1] - \Pr[\texttt{out}_{\text{GAME}_3(\kappa)} = 1]| \in \text{negl}(\kappa)$$

Proof. In order to successfully cause a situation where $\text{GAME}_2(\kappa)$ would yield output whereas $\text{GAME}_3(\kappa)$ aborts a distinguisher $\mathcal{D}$ would have to essentially *rerandomize* the first ciphertext.

To that end we use several hybrids to capture all the cases in which rerandomization might occur. We start with the case where the same verification key vk is encoded and then handle the case that the ciphertext has the same symmetric key sk.

H_0 is the game from $\text{GAME}_2(\kappa)$.

H_1 is as H_0 but the circuit aborts only if the first-round message of the sending party P_{ϕ_C} in the input transcript encrypts the same verification key as that from the challenge transcript and the decryption of the new first-round message is from the random oracle—that is, the decryption of that message using the first-round message of the input transcript using the actual decryption algorithm yields the error symbol $\perp$.

So the goal of the adversary is to distinguish, which is only possible if a different pre-image for the random oracle is found which evaluates to something that shares the same verification key as the random oracle when evaluated with the first sending parties ciphertext. However, as we already use a genuine random oracle in this game, this is information-theoretically impossible and hence the adversary cannot distinguish.

Thus it holds that

$$Adv_{\mathcal{D}} = |\Pr[\texttt{out}_{\mathcal{D},H_1} = 1] - \Pr[\texttt{out}_{\mathcal{D},H_0=1}]| \in \text{negl}(\kappa) \tag{25.1}$$

Now let E be the event that $\mathcal{D}$ can find a collision for a fixed value. We can incorporate this event into Eq. (25.1) as follows:

$$\begin{aligned} Adv_{\mathcal{D}} =&|\Pr[\texttt{out}_{\mathcal{D},H_1} = 1] - \Pr[\texttt{out}_{\mathcal{D},H_0} = 1]| \\ =&|\Pr[E] \cdot (\Pr[\texttt{out}_{\mathcal{D},H_1} = 1|E] \\ &\quad - \Pr[\texttt{out}_{\mathcal{D},H_0} = 1|E]) \\ &+ (1 - \Pr[E]) \cdot (\Pr[\texttt{out}_{\mathcal{D},H_1} = 1|\neg E] \\ &\quad - \Pr[\texttt{out}_{\mathcal{D},H_0} = 1|\neg E])| \end{aligned} \tag{25.2}$$

It trivially holds that $\neg E \implies \Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_1} = 1] = \Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_0}]$ as there the two games act exactly the same. Hence it holds for the second line of Eq. (25.2) that $(1-\Pr[E])\cdot(\Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_1} = 1|\neg E] - \Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_0} = 1|\neg E]) = 0$. So it holds that:

$$\begin{aligned} Adv_{\mathcal{D}} = |\Pr[E]\cdot(&\Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_1} = 1|E] \\ &-\Pr[\mathtt{out}_{\mathcal{D},\mathrm{H}_0} = 1|E])| \end{aligned} \tag{25.3}$$

Which holds information-theoretically due to the properties of the random oracle. Thus it holds that $\Pr[E] \in \mathrm{negl}(\kappa)$ and hence $Adv_{\mathcal{D}}$ is negligible.

This concludes our proof.

H_2 is as H_1 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is a **valid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is an **invalid** ciphertext, and (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same verification key for the challenge- and input-transcript.

With the message in the challenge transcript being a valid ciphertext we can reduce a distinguisher $\mathcal{D}$ between H_2 and H_1 to an adversary breaking the pre-image resistance of the random oracle. Let E be the event that $\mathcal{D}$ can find a different pre-image for which the second half, which is interpreted as the verification key, is the same as for the challenge message. In this case this pre-image has to be a value that is mapped by the random oracle to the value that was reported by the adversary as first dummy-friend message.

First note that the distinguisher can only ever win this game if the challenge-transcript contains an invalid ciphertext for the dummy friend in the first round. Denote by C the event that while sampling a random value for the first message of the dummy friend the message is not a valid ciphertext. We stress that $\neg C$ implies that the two games cannot be distinguished as the change induced by the gamehop will never be noted; simply because the additional check will never succeed.

We can thus conclude for now that with probability $(1-\Pr[C])$ the gamehop is perfectly indistinguishable and focus on the case where the adversary reported a dummy-friend message that is not a valid cipher. Note that in this case we can use the same argument as in the last game hop, as any distinguisher would fail if $\neg E$ where E is the event that $\mathcal{D}$ breaks the collision resistance, and $\Pr[E]$ is negligible due to the collision resistance of the random oracle.

H_3 is as H_2 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is an **invalid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is an **invalid** ciphertext, (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and

(4) the Dec* algorithm outputs the same verification key for the challenge- and input-transcript of the dummy friend. This then means that both decryptions are internally replaced by the PRF. Thus for distinguishing (which only works by causing an abort in H_3 where H_2 would have continued with the execution as the remaining part of the two games do not differ) the distinguisher would have to find a different message—which is a valid pre-image for the random oracle—that points to an output that shares the same second half as the value reported in the challenge transcript, which violates the collision resistance of the random oracle. Hence the two games cannot be distinguished.

H_4 is as H_3 but the circuit aborts only if the first-round message of the sending party P_{ϕ_C} in the input transcript encrypts the same verification key as that from the challenge transcript and the decryption of the new first-round message yields a valid message—that is, the message is a valid cipher and the random oracle is not used for this message.

We want to show that for all PPT distinguishes $\mathcal{D}$ the following statement is true:

$$|\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3} = 1]| \in \mathrm{negl}(\kappa)$$

We first define an intermediate event $E_{\mathcal{D}}$ as the event that $\mathcal{D}$ can *rerandomize* parts of a ciphertext. This leads to the following observation:

$$\begin{aligned} Adv_{\mathcal{D}} :=& |\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3}] = 1| \\ =& E \wedge |\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3}] = 1| \vee \\ & \neg E \wedge |\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3} = 1]| \\ =& E \wedge |\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3} = 1]| + \\ & \neg E \wedge |\Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] - \Pr[\mathtt{out}_{\mathcal{D},H_3} = 1]| \end{aligned} \tag{25.4}$$

Now note that $\neg E$ implies an equivalent behavior in both games, hence:

$$\neg E \wedge \Pr[\mathtt{out}_{\mathcal{D},H_4} = 1] = \neg E \wedge \Pr[\mathtt{out}_{\mathcal{D},H_3} = 1]$$

Note next that E is negligible due to the IND-CCA-property of the encryption system PKE; if $E \notin \mathrm{negl}$ the following attack on the IND-CCA security would be possible: (1) The adversary creates two random messages $msg_0 = (\cdot, \mathsf{vk}_0)$ and $msg_1 = (\cdot, \mathsf{vk}_1)$, where the first half is not important but the second half differs, and sends them as challenges. (2) The challenger returns a ciphertext y. (3) The adversary rerandomizes y to y' such that the second half remains the same and sends y' to the decryption oracle. (4) If the second half of the result is vk_0 then $\mathcal{A}$ returns $\beta = 0$ and if it is vk_1 it returns $\beta = 1$. Otherwise it returns a uniformly random bit. If E can occur with non-negligible probability then $\mathcal{A}$ would violate the IND-CCA security (which is implied by the IND\$-CCA security of PKE), thus we stress that $\Pr[E]$ is at most negligible.

Our claim follows.

H_5 is as H_4 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is a **valid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is a **valid** ciphertext, (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same verification key for the challenge- and input-transcript.

In this case we would have a distinguisher $\mathcal{D}$ that can effectively *partially re-randomize* transcripts. This distinguisher could be used in order to break the IND-CCA property of Pke as was the case in H_4.

H_6 is as H_5 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is an **invalid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is a **valid** ciphertext, and (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same verification key for the challenge- and input-transcript.

Essentially, to distinguish the distinguisher $\mathcal{D}$ has to create a ciphertext that encodes the same verification key that is in the image domain of the random oracle. Since this is truly random *and* the distinguisher cannot query the oracle it follows that this is possible with negligible probability only which proves our claim.

H_7 This game-hop is purely cosmetic and hence does not change the distribution.

H_8 is as H_7 but the circuit aborts only if the first-round message of the sending party P_{ϕ_C} in the input transcript encrypts the same symmetric key as that from the challenge transcript and the decryption of the new first-round message is from the random oracle—that is, the decryption of that message using the first-round message of the input transcript using the actual decryption algorithm yields the error symbol $\perp$.

This is similar to the game hop from H_1 and hence indistinguishable.

H_9 is as H_8 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is a **valid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is an **invalid** ciphertext, and (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same symmetric key for the challenge- and input-transcript.

This is the same situation as the game hop between H_2 and H_1 and thus indistinguishable.

H_{10} is as H_9 only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is an **invalid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is an **invalid** ciphertext, (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same symmetric key for the challenge- and input-transcript of the dummy friend. This then means that both decryptions are internally replaced by the PRF. Showing indistinguishability here is similar to that of H_3 and H_2.

H_{11} is as H_{10} but the circuit aborts only if the first-round message of the sending party P_{ϕ_C} in the input transcript encrypts the same verification key as that from the challenge transcript and the decryption of the new first-round message yields a valid message—that is, the message is a valid cipher and the random oracle is not used for this message.

Indistinguishability follows from the indistinguishability between H_3 and H_4 as the proof is almost identical.

H_{12} is as H_{11} only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is a **valid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is a **valid** ciphertext, (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same symmetric key for the challenge- and input-transcript.

Again the proof is analogous to the indistinguishability of H_{11} and H_{12} and we do not write it up specifically.

H_{13} is as H_{12} only that the circuit additionally aborts if (1) the first message of the dummy friend in the *challenge transcript* is an **invalid** ciphertext, and (2) the first message of the dummy friend in the *input transcript* is a **valid** ciphertext, and (3) the first message of the dummy friend in the *challenge transcript* differs from the first message of the dummy friend in the *input transcript*, and (4) the Dec^* algorithm outputs the same symmetric key for the challenge- and input-transcript.

The non-existence of a successful PPT distinguisher $\mathcal{D}$ which is better than the naive distinguisher that randomly guesses a bit follows from the non-existence of a distinguisher for the hybrid game hop from H_5 to H_6.

H_{14} This game-hop is purely cosmetic and hence does not change the distribution. We only include it for consistency with $\textsc{Game}_3(\kappa)$.

□

$\textsc{Game}_4(\kappa)$: This game follows $\textsc{Game}_3(\kappa)$ but simulates the circuit slightly different: If the first receiver message of the input transcript τ is the same as that of the challenge transcript τ_C, instead of decrypting it the circuit directly sets $otp = otp_C$ and $\mathsf{vk}_\mathsf{R} = \mathsf{vk}_{C\mathsf{R}}$ as the values used in the creation of the challenge transcript.

Lemma 25.2.5 (Indistinguishability of $\textsc{Game}_3(\kappa)$ and $\textsc{Game}_4(\kappa)$)**.** *Let* $\textsc{Pke}$ *be an IND\$-CCA secure asymmetric encryption scheme, let* $\textsc{Ske}$ *be a tightly secure multi-challenge IND\$-CCA secure symmetric encryption scheme, let* $\textsc{Sig}$ *be an sEUF-CMA secure signature scheme, let* $\mathbb{O}$ *be an ideal obfuscator, and let* PRF *be a secure PRF. Then, for all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* $\textsc{Game}_3(\kappa)$ *and* $\textsc{Game}_4(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Indistinguishability easily follows from the perfect correctness of the encryption scheme $\textsc{Pke}$. If there was any message $msg_\mathsf{R}^{(0)}$ created by the adversary as $\textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (otp, \mathsf{vk}_\mathsf{R}))$ which does *not* decrypt with sk_P to $(otp, \mathsf{vk}_\mathsf{R})$ this would mean that $\textsc{Pke}.\mathsf{Dec}(\mathsf{sk}_\mathsf{P}, \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (otp, \mathsf{vk}_\mathsf{R}))) \neq (otp, \mathsf{vk}_\mathsf{R})$. As this cannot happen by requirement indistinguishability follows. □

$\textsc{Game}_5(\kappa)$: This game follows $\textsc{Game}_4(\kappa)$ but simulates the oracle differently if the first-round messages of both parties match the first-round messages in the challenge transcript τ_C. In this case, the program compares the input transcript τ with the challenge transcript τ_C until it finds the first round χ^* in which the input differs from the challenge transcript. It then checks round $\chi^* + 1$, and if it contains the same message from the sending party, it adds one to χ.

Finally, the circuit flips a biased coin, which returns the correct bit b_C with probability $p := 1/2 + \chi^*/2c$ and the complementary bit $(1 - b_C)$ otherwise.

Lemma 25.2.6 (Indistinguishability of $\textsc{Game}_4(\kappa)$ and $\textsc{Game}_5(\kappa)$)**.** *Let* $\textsc{Pke}$ *be an IND\$-CCA secure asymmetric encryption scheme, let* $\textsc{Ske}$ *be a tightly secure multi-challenge IND\$-CCA secure symmetric encryption scheme, let* $\textsc{Sig}$ *be an sEUF-CMA secure signature scheme, let* $\mathbb{O}$ *be an ideal obfuscator, and let* PRF *be a secure PRF. Then, for all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* $\textsc{Game}_4(\kappa)$ *and* $\textsc{Game}_5(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_5(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. A distinguisher $\mathcal{D}$ between $\textsc{Game}_4(\kappa)$ and $\textsc{Game}_5(\kappa)$ can be reduced to an adversary $\mathcal{A}$ breaking the sEUF-CMA security of the used signature scheme $\textsc{Sig}$.

Creating the transcript. The creation of the transcript works similar in both games, hence the transcript can be created by letting the adversary play all three parties roles according to $\textsc{Game}_2(\kappa)$ for a randomly chosen sender P_{ϕ_C} and a randomly transmitted bit b_C. The only change is with respect to signatures. While the verification key from the challengers signature oracle is known to the adversary and hence can be embedded into the senders first message all signatures of the form

$\sigma_\phi \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\phi, (msg_0^{(\chi-1)}, msg_1^{(\chi-1)}))$ created in line 11 of the protocol are replaced by queries $(msg_0^{(\chi-1)}, msg_1^{(\chi-1)})$ to the signing oracle.

The complete transcript τ_C is then sent to the distinguisher $\mathcal{D}$.

Simulating the oracle. When $\mathcal{D}$ sends some transcript τ for evaluation to the oracle the adversary $\mathcal{A}$ behaves according to Section 25.2.2: If the challenge transcript corresponds to the input transcript then it returns the correct bit, and if any of the first-round messages differ it simulates the circuit entirely:

If both parties input different messages than in the original transcript then we know due to the check in line 3 of the circuit code that both encrypt a different message and hence a use a new verification key (and also a symmetric key) for the remaining rounds. By knowing the secret key $\mathsf{sk_P}$ the adversary can thus extract both and simulate accordingly.

The same reasoning is true if only the sending parties message is replaced as then still both parties messages are independent from the challenge.

Special care thus only has to be taken into the simulation for transcripts where the first sending party message matches.

If the dummy friends message differs then the circuit also executes the same circuit, which the adversary can simulate as it has all the required information; the only thing that depends on the challenge oracle are the signatures and being in possession of the verification key the simulator can efficiently verify a given signature.

If the dummy friends message also matches then the circuit acts by counting rounds until the messages differ for the first time. Let $\mathcal{D}$ be a distinguisher that uses this behavior to distinguish. In $\textsc{Game}_5(\kappa)$ the round χ^* is fixed due to lines $7 - 11$ as the first round where any of the messages differs from the challenge transcript, where potentially 1 is added in case the sending parties message for the next round is different. Thus for a given transcript τ we can fix p for $\textsc{Game}_5(\kappa)$; in order to distinguish this value p' must differ in $\textsc{Game}_3(\kappa)$. (i) $p' < p$, meaning that the circuit outputs the correct bit with a lower probability when the message is handled by the original code of the circuit. This, however, would contradict the correctness of the scheme. (ii) $p' > p$, meaning that the probability of output b_C is *larger* when the transcript is handled by the actual code than when it is handled by our modification. By requirement we know that up until round χ^* all messages are takes directly from the challenge transcript, and that $\chi^* < c$ (as otherwise the case would have been handled by the code in line 1). We can now differentiate between the two possible replacements n round $(\chi^* + 1)$: (i.1) If the *sending* parties message has been replaced in this round then χ^* stays the same and $p := 1/2 + \chi^*/2c$. In order to get a larger p' the adversary would have to create a *replacement* message which encrypts the same bit and a different signature on the same round-$(\chi^* - 1)$-messages. Knowing the secret key used for the transcript the reduction adversary can extract this signature and send it as valid signature on $(msg_0^{(\chi^*-1)}, msg_1^{(\chi^*-1)})$ to the challenger. Since the content of the message differs but the bit has to be the same (as otherwise the message would be rejected in $\text{GAME}_3(\kappa)$) it follows that the distinguisher created a new signature that was never returned from the challenge-oracle. (ii.2) If the *dummy friend* parties message has been replaced in this round then χ^* is increased by 1 and in order to get $p' > p$ the distinguisher has to create a message for round (χ^*+2) that contains a valid signature for this new message. Since by requirement the dummy friends message has been replaced in round $(\chi^* + 1)$ this too corresponds to a new message and hence is a valid forgery for the challenger.

As $\mathcal{A}$ can only have a negligible chance to create a forgery due to the sEUF-CMA security of the signature scheme Sig it follows that $\alpha \in \text{negl}(\kappa)$. □

$\text{GAME}_6(\kappa)$: This game is the same as $\text{GAME}_5(\kappa)$, but in creating the challenge transcript τ_C, this game only reports randomness for the first-round message msg_ϕ^0 that specifies the symmetric key sk_ϕ and the verification key vk_ϕ to be used for the remaining communication with the circuit alongside the receiver message $msg_{\text{R}}^{(0)}$ that specifies the One Time Pad and the verification key. Note that all keys are still created as they are needed for the remaining rounds.

Lemma 25.2.7 (Indistinguishability of $\text{GAME}_5(\kappa)$ and $\text{GAME}_6(\kappa)$)**.** *Let* Pke *be an* IND\$-CCA *secure asymmetric encryption scheme, let* Ske *be a tightly secure multi-challenge* IND\$-CCA *secure symmetric encryption scheme, let* Sig *be an* sEUF-CMA *secure signature scheme, let $\mathbb{O}$ be an ideal obfuscator, and let* PRF *be a secure* PRF*. Then, for all* PPT *guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_5(\kappa)$ and $\text{GAME}_6(\kappa)$ is bounded by:*

$$|\Pr[\texttt{out}_{\text{GAME}_5(\kappa)} = 1] - \Pr[\texttt{out}_{\text{GAME}_6(\kappa)} = 1]| \in \text{negl}(\kappa)$$

Proof. Let $\mathcal{D}$ be an efficient distinguisher that can predict if it is in $\text{GAME}_5(\kappa)$ or in $\text{GAME}_6(\kappa)$ with probability $1/2 + \alpha$. Out of $\mathcal{D}$ we can construct an adversary $\mathcal{A}$ on the IND\$-CCA property of Pke.

The reduction algorithm $\mathcal{A}$ starts by creating the key-pair for the signature scheme and the key for the symmetric scheme honestly and follows $\text{GAME}_5(\kappa)$ the creation of the transcript, only that instead of encrypting the first message using the public-key scheme Pke directly, $\mathcal{A}$ forwards the messages $(otp, \text{vk}_{\text{R}})$ and $(\text{sk}_\phi, \text{vk}_\phi)$ as challenge to the challenger $C_{IND\$-CCA}$ of the IND\$-CCA game.

For further evaluation of the oracle the reduction adversary uses the *decryption* oracle from the challenger $C_{IND\$-CCA}$ for any first-round message that is not from the reported transcript τ and continues simulation using the extracted secret key of that respective party; the same first-round message is covered automatically since $\text{GAME}_5(\kappa)$ and the same receiver message in $\text{GAME}_4(\kappa)$.

The reported transcript contains exactly the same views that are required by the two games: if the output is a proper encryption then the view is equivalent to $\text{GAME}_5(\kappa)$, and if the output of the oracle is uncorrelated randomness, the view corresponds to $\textsc{Game}_6(\kappa)$.

Thus, if $\mathcal{D}$ can differentiate the two games with probability $1/2 + \alpha$, then $\mathcal{A}$ can differentiate the oracles with the same probability. The IND$-CCA requirement for $\textsc{Pke}$ thus implies $\alpha \in \mathsf{negl}(\kappa)$. □

$\textsc{Game}_7(\kappa)$: This game is the same as $\textsc{Game}_6(\kappa)$ but in creating the challenge transcript τ_C, this game also reports randomness instead of transcripts for all messages msg_ϕ^χ for $\chi \in [c]$ that shift the bit towards b_C. That means that instead of using the IND$-CPA secure symmetric scheme $\textsc{Ske}$ with the symmetric key sk_ϕ the challenge transcript now only contains randomly sampled messages.

We also do not let the adversary create the keys for $\textsc{Sig}$ and $\textsc{Ske}$ as they are no longer needed for creating the transcript.

Lemma 25.2.8 (Indistinguishability of $\textsc{Game}_6(\kappa)$ and $\textsc{Game}_7(\kappa)$)**.** *Let $\textsc{Pke}$ be an IND$-CCA secure asymmetric encryption scheme, let $\textsc{Ske}$ be a tightly secure multi-challenge IND$-CCA secure symmetric encryption scheme, let $\textsc{Sig}$ be an sEUF-CMA secure signature scheme, let $\mathbb{O}$ be an ideal obfuscator, and let* PRF *be a secure PRF. Then, for all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\textsc{Game}_6(\kappa)$ and $\textsc{Game}_7(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_6(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_7(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce a distinguisher $\mathcal{D}$ that distinguishes $\textsc{Game}_6(\kappa)$ from $\textsc{Game}_7(\kappa)$ to an adversary $\mathcal{A}$ on the IND$-CPA property of the symmetric encryption scheme $\textsc{Ske}$. Again, we adapt the LR-view of Rogaway [119] to account for the multiple challenges required, in that the game is played via oracle access to an oracle which either outputs valid encryptions of the input, or which outputs uncorrelated randomness.

We now describe how the adversary $\mathcal{A}$ behaves in order to embed the challenge of the IND$-CCA challenger $C_{IND\$-CCA}$ into a challenge for the distinguisher $\mathcal{D}$.

The adversary starts by following the protocol according to $\textsc{Game}_6(\kappa)$ and creating the messages, with the one exception that any call to $\textsc{Ske}.\mathsf{Enc}(\mathsf{sk}_\phi, b\|\sigma_\phi)$ is replaced by an *oracle* call to the challenge oracle with input $(b\|\sigma_\phi)$. This causes the transcript to either only have truly random messages (in which case the view corresponds to $\textsc{Game}_7(\kappa)$) or actual encryptions under the challengers secret key sk (which results in a valid transcript for $\textsc{Game}_6(\kappa)$).

Thus the adversary inherits the advantage α as long as $\mathcal{A}$ can simulate the decryption oracle accordingly. Fortunately this is the case here. Further, as we assume a lazy evaluation of the or[5] the decryption oracle is only called on the first round message

[5] This means that first the simulator checks for equal messages and if that is already true, the second condition is ignored, and hence the sending parties message is only forwarded to the decryption oracle if it differs from the challenge.

```
Oracle $O_i^\beta$
if $\beta = 0$ then
    $p_i := \frac{i+c-1}{2c}$
else
    $p_i := \frac{i+c}{2c}$
return $\mathtt{Ber}(p_i)$

$C(c)$
$\beta \xleftarrow{\$} \{0,1\}$
return $\mathcal{A}^{O_1^\beta,\dots,O_c^\beta}(1^\kappa)$

$\mathcal{A}(1^\kappa)$
for $j = 1 \dots t$ do
    $i_j \leftarrow Computations$
    $msg_j \xleftarrow{\$} O_{i_j}$
$\beta' \leftarrow Computations((i_j, msg_j)_{j=1}^t)$
return $\beta'$
```

Figure 25.3.: Game to distinguish whether Bernoulli oracles follow a given distribution p or $q = p - 1/2c$.

which differs from the challenge transcript and, hence, the decryption oracle is never called on the challenge ciphertext. The leaked public key pk_P is controlled by the reduction adversary and hence any new transcript that contains a different first-round message than the challenge transcript can be simulated by first decrypting the secret key (which is different from the one used by the challenger with overwhelming probability) and then decrypting each round individually. □

$\textsc{Game}_8(\kappa)$: This game follows $\textsc{Game}_7(\kappa)$, but instead of choosing a random sender at the beginning of the game and considering this message to be the right one, the oracle ignores the additional check and only looks for the first round where *both* messages are identical to the challenge.

Note that $\textsc{Game}_8(\kappa)$ is entirely independent of the real sender, hence given a challenge transcript, it is trivially impossible to obtain a non-negligible advantage to determine the sending party.

For the sake of reducing complexity of the problem of proving indistinguishability between $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$ we describe an intermediate game in Fig. 25.3 that is provably as hard to solve as distinguishing the two games.

The key idea is the following: The challenger C_{Ber} creates c oracles where the probability to return 1 is equally distributed between 1/2 and 1 in c steps.

On $\beta = 0$ the oracles are distributed equally between $[1/2, 1)$. On $\beta = 1$ the oracles are distributed equally between $(1/2, 1]$. That is, on $\beta = 0$ the oracle χ returns 1 with probability $(c + \chi - 1)/(2c)$ and on $\beta = 1$ it returns 1 with probability $(c + \chi)/(2c)$.

We now stress that this game is *as hard* as the problem of distinguishing the two games from $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$:

Lemma 25.2.9. *Let $\mathcal{D}$ be a distinguisher between $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$ with advantage α over guessing. Let t be the number of queries that $\mathcal{D}$ sends to the obfuscated circuit. There is a reduction adversary $\mathcal{A}$ that uses $\mathcal{D}$ which has advantage α over guessing in winning Fig. 25.3.*

Proof. Creating the Transcript. Upon activation the adversary samples a bit $b \in \{0, 1\}$ that is to be transferred in the transcript and a bit $\phi \in \{0, 1\}$ that defines the sending party. Using c as the number of rounds and m as the length of each message $\mathcal{A}$ creates the transcript by reporting two uniformly sampled messages of length m for each round inside the loop, and for the preparational round two messages of length $2m$ each for the dummy friend and the sender and one message of length m for the receivers messages. The resulting transcript τ_C is reported to the distinguisher $\mathcal{D}$.

Simulating the Circuit. When receiving a transcript τ, $\mathcal{A}$ checks if $\tau[0] \neq \tau_C[0]$. If this is the case $\mathcal{A}$ executes the program honestly as defined in $\textsc{Game}_7(\kappa)$ and returns the output bit. Note that the normal execution of a *different* transcript is entirely independent of the secret β chosen by C_{Ber} and effectively of the whole change induced in the transition from $\textsc{Game}_7(\kappa)$ to $\textsc{Game}_8(\kappa)$. Hence $\mathcal{A}$ has all the information to simulate *new* transcripts.

Otherwise, if $\tau[0] = \tau_C[0]$, $\mathcal{A}$ embeds the challenge oracle by finding $\chi^* = \max_\chi \tau[0 \ldots \chi] = \tau_C[0 \ldots \chi]$ by comparing the input to the previously reported challenge transcript. Then $\mathcal{A}$ checks the messages $\tau[\chi^* + 1]$. Let ${msg^C}_0$ and ${msg^C}_1$ be the message from the challenge transcript at round $\chi^* + 1$ of parties P_0 and P_1, respectively, and let msg_0 and msg_1 be the respective messages from the transcript that was input by $\mathcal{D}$.

$\mathcal{A}$ only considers the message reported for P_ϕ; if $msg_\phi = {msg^C}_\phi$, that is, if at round $\chi^* + 1$, the input transcript contains the same message of the *sending* party, $\mathcal{A}$ sends χ^* to the oracle O_{χ^*} provided by C_{Ber} and obtains a binary output b^*. Since the output of C_{Ber} is going towards 1, whereas the bit b_C was uniformly distributed, $\mathcal{A}$ has to *adjust* the output if $b_C = 0$, such that 0 is output with higher probability. Hence, $\mathcal{A}$ sends $b^* \oplus \bar{b}_C$ back as output to $\mathcal{D}$; this essentially flips the output bit iff $b_C = 0$, in which case the output of the oracle lies between 0 and $1/2$.

Otherwise, if $msg_\phi \neq {msg^C}_\phi$, $\mathcal{A}$ locally samples a Bernoulli-distributed bit according to p_{χ^*} and sends the result back to $\mathcal{D}$.

Translating the Result. After (at most) t queries the distinguisher terminates and sends its guess. If $\mathcal{D}$ guesses it was playing $\textsc{Game}_8(\kappa)$ then $\mathcal{A}$ reports $\beta' = 0$ to C_{Ber}, thus guessing that the χ-th oracle returns 1 with probability $(\chi + c - 1)/2c$. If $\mathcal{D}$ guesses it was playing $\textsc{Game}_7(\kappa)$ then $\mathcal{A}$ sends the output $\beta' = 1$ to C_{Ber}, indicating that the probabilities were given as $(\chi + c)/2$.

For a tight reduction it remains to show that the view of $\mathcal{D}$ is statistically close to that from $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$, depending on the choice of C_{Ber}:

First, note that the way that C_{Ber} handles the queries in Fig. 25.3 mimics exactly the situation from the game change: either a query for χ returns the 1 with probability $\frac{\chi+c}{2c}$ or with probability $\frac{\chi-1+c}{2c}$. The former implies that the sending parties message is still considered as in $\textsc{Game}_7(\kappa)$, the latter representing $\textsc{Game}_8(\kappa)$ where the oracle ignores a valid message in round $\chi + 1$ and only considers the round χ^* until *both* messages match the challenge transcript.

The challenge oracle is only queried when the *right* sending message for the following round is used. This is also the only instance where $\textsc{Game}_7(\kappa)$ and $\textsc{Game}_8(\kappa)$ differ from each other.

Finally, note that at most t sample were sent to the challenge oracle. Thus, the view is simulated perfectly and the view of $\mathcal{D}$ corresponds to $\textsc{Game}_8(\kappa)$ iff the additional factor of $-1/2c$ is ignored and to $\textsc{Game}_7(\kappa)$ otherwise.

So if $\mathcal{D}$ has non-negligible advantage α over guessing then $\mathcal{A}$ inherits this advantage for winning the game from Fig. 25.3, albeit if $\mathcal{D}$ sends t queries to $\mathcal{A}$, less than t queries are forwarded to C_{Ber}. □

Proving indistinguishability has thus been reduced to showing that no fine-grained adversary can win the game from Fig. 25.3 with non-negligible advantage. The interface of an adversary in this game is given as a set of $2c$ *oracles*. Each oracle follows a *Bernoulli* distribution that returns the correct bit b_C with probability p. For each round $\chi < c$ any distinguisher $\mathcal{D}$ is given access to two oracles. Each oracle can be queried by copying the first χ messages of *both* parties, but then using (exactly) one new message for round $(\chi + 1)$—which replaces either the sending parties message or that of the dummy friend. Any upper bound on winning the game from Fig. 25.3 translates to the underlying problem of distinguishing the final two games.

Analyzing the game from Fig. 25.3 comes down to probability theory. Recall from Corollary 22.2.5 that in order to distinguish two Bernoulli distributions p and q with advantage $\alpha/2$ we require $\Omega(\alpha/\mathrm{d_{TV}}(p,q))$ many samples. Applying this corollary to Fig. 25.3 implies that we have c instances where the χ-th instance is to distinguish $p = \frac{\chi+c}{2c}$ from $q = \frac{\chi+c-1}{2c}$. This implies the following L_1-norm between p and q in round χ:

$$\begin{aligned}\mathrm{d_{TV}}(p,q) &= \frac{1}{2}(|\Pr[p=1]-\Pr[q=1]| + |\Pr[p=0]-\Pr[q=0]|)\\ &= \frac{1}{2}\left(\left|\frac{c+\chi}{2c} - \frac{c+\chi-1}{2c}\right| + \left|\frac{c-\chi}{2c} - \frac{c-\chi+1}{2c}\right|\right)\\ &= \frac{1}{2}\left(\frac{1}{2c}+\frac{1}{2c}\right) = \frac{1}{2c}\end{aligned} \tag{25.5}$$

Note here that the total variational distance in round χ is *independent* from the round χ and the same for all c oracles. In combination with Lemma 22.2.3 this means that *any* distribution p and q resulting from sampling t times from *arbitrary* oracles results in a total variational distance $\leq t\frac{1}{2c}$. [6]

[6] This is in contrast to the Hellinger-distance $\mathrm{d_H}$ which yields tighter bounds but where the amount of information from a single query really depends on the oracle O_χ which is queried. This makes it harder to provide meaningful bounds for adversaries querying different oracles with their t samples.

We now merge this insight with the result of Eq. (25.5) and the bound of Corollary 22.2.5. This leads a lower bound of:

$$t \in \Omega\left(\frac{\alpha}{\mathrm{d_{TV}}(p,q)}\right) = \Omega(\alpha c) \tag{25.6}$$

We thus have:

Corollary 25.2.10. *Let $\mathcal{D}$ be a distinguisher in Fig. 25.3 that uses t samples and has runtime in $\mathsf{FC}_2 := o(c^2/\alpha)$. Let the cost of acquiring a single sample be $O(c)$. Then the distinguisher $\mathcal{D}$ is correct with probability at most $1/2 + \alpha/2$.*

Proof. The bound from Eq. (25.6) covers any adversary trying to win Fig. 25.3 regardless of how the t samples are distributed between the c oracles. This follows from the subadditional property of the total variational distance shown in Lemma 22.2.3 and the computation in Eq. (25.5) showing that the total variational distance is the same between all oracles; thus the bound from Lemma 22.2.2 still is valid and the total variational distance between any pair of t-fold distributions is at most $t \cdot \frac{1}{2c}$.

Thus Lemma 22.2.4 maintains its validity. Hence the lower bound of Eq. (25.6) matches our setting. The bound is linear in c with the linear cost of querying a single sample (as the adversary has to evaluate the entire circuit for each sample, which requires $O(c)$ runtime) this limits the distinguisher in such a way that only strictly less samples can be drawn than required according to Eq. (25.6). □

Putting everything together, we have shown that for all PPT distinguishers $\mathscr{D}$, $|\Pr[\mathtt{out}_{0,\mathscr{D}} = 1] - \Pr[\mathtt{out}_{8,\mathscr{D}} = 1]|$ is negligible in κ. In particular, $|\Pr[\mathtt{out}_{0,\mathscr{D}} = 1] - \Pr[\mathtt{out}_{8,\mathscr{D}} = 1]|$ is negligible for distinguishers $\mathscr{D}$ in FC_2. Additionally, the employed reductions are in $\mathsf{FC}_1 = O(c)$. Furthermore, for all adversaries $\mathcal{A}$, $\left|\Pr\left[\mathtt{out}_{8,\mathcal{A}} = 1 \;\middle|\; \phi = 0\right] - \Pr\left[\mathtt{out}_{8,\mathcal{A}} = 1 \;\middle|\; \phi = 1\right]\right| \leq \alpha$, where the runtime of the game also is in FC_1. Hence, we may conclude that for all adversaries $\mathcal{A}$ in FC_2, $|\Pr_{\phi \xleftarrow{\$} \{0,1\}}[\mathrm{Exp}^{\mathsf{anon}}_{\Pi_{AT},\mathcal{A},\phi}(\kappa) = \phi] - 1/2| \leq \alpha/2$.

On the need for stronger obfuscation. It was shown by Canetti et al. [44] that indistinguishability obfuscation (or more precisely, its probabilistic variant) can only guarantee indistinguishability if the distance between the output distributions of two circuits is statistically close to zero. This is not the case in our final game hop. Therefore, we crucially require a stronger form of obfuscation such as virtual black-box obfuscation or ideal obfuscation. Using ideal obfuscation, we obtain a heuristic candidate proven secure in an idealized model and consider our result as a first step towards instantiating anonymous transfer.

25.2.3. Secrecy

Theorem 25.2.11 (Secrecy). *Let* Pke *be an IND\$-CCA secure asymmetric encryption scheme, let* Ske *be an IND\$-CCA secure symmetric encryption scheme, let* Sig *be a sEUF-CMA secure signature scheme, let* $\mathbb{O}$ *be an ideal obfuscator, and let* PRF *be a secure PRF. Then,* Π_{AT} *satisfies* ς*-secrecy with* $\varsigma \in \mathrm{owhl}(\kappa)$*.*

Essentially the proof shows that no PPT distinguisher that gets the transcript but not the receivers transcript can exist that can distinguish between the case where we always transfer a 0 and where we always transfer the 1 better than by guessing. This follows from the fact that the One Time Pad chosen uniformly at random by the receiver masks the output bit. As this masking bit is never revealed and only sent in an encrypted form with the key of the circuit our claim follows.

Game$_1(\kappa)$: This is the original game (after replacing the obfuscated circuit with oracle access to the circuit, the PRF with an actual random oracle and the adversary with the simulator), where the sending party P_ϕ is chosen uniformly at random but the sending party always sends $b = 0$.

Game$_2(\kappa)$: This game follows Game$_1(\kappa)$, but with the following changes:

- During the simulation of the oracle P_{FG-AT} from Fig. 25.1 the simulation enforces correctness of the challenge transcript τ_C: if the input transcript τ matches the challenge transcript τ_C, it returns $otp_C \oplus b_C$.
- During simulation of the circuit the adversary aborts if any of the first-round messages from P_0 or P_1 differ from the messages reported in the challenge transcript *and* the decryptions still match.
- During simulation of the circuit, if the first receiver message of the input transcript τ is the same as that of the challenge transcript τ_C, instead of decrypting it the circuit directly sets $otp = otp_C$ and $\mathsf{vk}_\mathsf{R} = \mathsf{vk}_{C\mathsf{R}}$ as the values used in the creation of the challenge transcript.
- During simulation of the oracle, if the first-round messages of both parties match the first-round messages in the challenge transcript τ_C. In this case, the program compares the input transcript τ with the challenge transcript τ_C until it finds the first round χ^* in which the input differs from the challenge transcript. It then checks round $\chi^* + 1$, and if it contains the same message from the sending party, it adds one to χ.

  Finally, the circuit flips a biased coin, which returns the correct bit b_C with probability $p := 1/2 + \chi^*/2c$ and the complementary bit $(1 - b_C)$ otherwise.

Lemma 25.2.12. *Let* Pke *be an IND\$-CCA secure public-key encryption scheme. Let* Ske *be an IND\$-CCA secure secret-key encryption scheme. Let* Sig *be an sEUF-CMA* *secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for* $\textsc{Game}_1(\kappa)$ *and* $\textsc{Game}_2(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_1(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_2(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Follows from Lemmas 25.2.3 to 25.2.8. □

$\textsc{Game}_3(\kappa)$: This game follows $\textsc{Game}_2(\kappa)$ but the oracle is simulated slightly different: If the first receiver message is the same as the one reported in the challenge transcript and the input transcript is *not* the challenge transcript, then the circuit reports a uniformly random bit.

Note that this does not work in the anonymity proof as there we assume that the adversary is given access to the receivers random tape, and hence can create their own new signature on the modified transcript.

Lemma 25.2.13. *Let* Pke *be an IND\$-CCA secure public-key encryption scheme. Let* Ske *be an IND\$-CCA secure secret-key encryption scheme. Let* Sig *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for* $\textsc{Game}_3(\kappa)$ *and* $\textsc{Game}_4(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce a distinguisher $\mathcal{D}$ between these two games to an adversary $\mathcal{A}$ on the EUF-CMA security of the signature scheme Sig.

Creating the transcript. The creation of the transcript is straightforward and works by sampling random messages for each party.

Simulating the oracle. The used verification key vk_R is set to be the verification key of the challenger. Thus the final signature σ_R on the entire transcript, which is required for the circuit to not *abort* (by outputting a random bit), needs to be forged. Hence for each input transcript the adversary first checks if the receiver message is equivalent to that from the challenge transcript. If it isn't then the transcript cannot be used for distinguishing anyways. If it is, the adversary checks if the remaining transcript is the same as well. If it is, the transcript cannot be used for distinguishing and simulation just continues as the path taken is equivalent in both games. If it isn't then the adversary checks if the signature σ_R verifies under the used key. If it doesn't then both games act exactly the same and output a random bit. Hence to distinguish the signature has to verify. Then, however, the signature is a valid forgery.

Translating the result. Assuming that no valid signature was queried then—as mentioned above—the distinguishing advantage must be negligible. For a non-negligible advantage the distinguisher has to query at least one signature. This can be used as forgery to break the EUF-CMA security of Sig.

As stated above, the advantage of the distinguisher is directly related to the probability of successfully distinguishing. Hence $\mathcal{A}$ will have a valid forgery with non-negligible advantage.

This would contradict the EUF-CMA security of the signature scheme and thus completes our proof. □

Game$_4(\kappa)$: This game is as Game$_3(\kappa)$ but instead of fixing $b := 0$, we now act as if the sending party sends $b := 1$.

Lemma 25.2.14. *Let Pke be an IND\$-CCA secure public-key encryption scheme. Let Ske be an IND\$-CCA secure secret-key encryption scheme. Let Sig be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for Game$_3(\kappa)$ and Game$_4(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Security automatically follows from the statistical security of the One Time Pad encryption: if there was a distinguisher $\mathcal{D}$ with non-negligible advantage α in distinguishing these two games then there would be an adversary $\mathcal{A}$ which can decrypt One Time Pad encrypted bits with non-negligible advantage.

Simulating the transcript. As the transcript is the same in both games, we construct the transcript by uniformly sampling each message.

Simulating the oracle. If $\mathcal{A}$ is given a ciphertext ct (which by definition is defined as $otp \oplus b$ for some random otp and b) we simulate the oracle as described in both games (since they are equivalent) and whenever the code says **return** $\text{CointossS}^{(\tau)}_{(p)}(otp_C \oplus b, otp_C \oplus \overline{b})$, we replace $otp_C \oplus b$ with msg and $otp_C \oplus \overline{b}$ with $\overline{msg}$.

Translating the result. If the distinguisher assumes Game$_3(\kappa)$ then $\mathcal{A}$ assumes $b = 0$ (in which case the transferred bit is $b_C := msg$), and if the distinguisher assumes Game$_4(\kappa)$ then $\mathcal{A}$ guesses $b = 1$ (and the transferred bit is $b_C := \overline{msg}$).

Note that if $\mathcal{D}$ is correct with advantage α over guessing, hence the guess of $\mathcal{A}$ is correct with advantage α as well. □

Game$_5(\kappa)$: This game follows Game$_4(\kappa)$ but undoes all the changes from the first-to-second gamehop and that from Game$_3(\kappa)$.

Lemma 25.2.15. *Let* $\textsc{Pke}$ *be an IND\$-CCA secure public-key encryption scheme. Let* $\textsc{Ske}$ *be an IND\$-CCA secure secret-key encryption scheme. Let* $\textsc{Sig}$ *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* $\textsc{Game}_4(\kappa)$ *and* $\textsc{Game}_5(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_5(\kappa)} = 1]| \in \mathrm{negl}(\kappa)$$

Proof. Follows from Lemmas 25.2.3 to 25.2.8 and 25.2.13. □

Final Result. Let $c = c(\kappa)$ be a polynomial in κ. Let $\mathrm{FC}_1 := \mathcal{O}(c)$ and let $\mathrm{FC}_2 := \mathrm{o}(c^2(1-\delta))$ for some $\delta \in \mathbb{R}_{[0,1]}$. Putting Theorems 25.2.1, 25.2.2 and 25.2.11 together, we have:

Corollary 25.2.16. *The protocol* Π_{AT} *is a strong* FC_1*-fine-grained* $(1-\mathrm{negl}(\kappa), \delta, 1-\mathrm{negl}(\kappa), c, 1)$*-AT against* FC_2*.*

So in total, we get that:

Corollary 25.2.17. *The protocol* Π'_{AT^n} *defined in Chapter 26 is a strong* FC_1*-fine-grained* $(1-\mathrm{negl}(\kappa), \delta, 1-\mathrm{negl}(\kappa))$*-AT against* FC_2*, where* $\mathrm{FC}_1 := \mathcal{O}(c)$ *and* $\mathrm{FC}_2 := \mathrm{o}(c^2(1-\delta))$*.*

26. A fine-grained Anonymous Transfer for n-bit messages

26.1. The Protocol

In this section we perform some slight modifications of the protocol introduced in Chapter 25 to see how it can handle the transfer of n-bit messages directly.

The protocol is given in Fig. 26.1, the corresponding obfuscated program in Fig. 26.2. The protocol in Fig. 26.1 is essentially the same as the one in Fig. 25.1 for single-bit AT, where the bit $b \in \{0, 1\}$ was replaced by a message $\Sigma \in \{0, 1\}^n$ and the circuit is now the one from Fig. 26.2 instead of the one from Fig. 25.2. The modified circuit from Fig. 26.2 is basically the one from Fig. 25.2 but with a message Σ instead of a bit b. Additionally, there is one major change: Instead of returning the actual bit b with probability proportional to the amount of correct rounds of the sender P_ϕ and *the complementary bit* with the remaining probability the new circuit from Fig. 26.2 separates between returning either the message Σ_ϕ or a dedicated *error state* $\perp$. The actual message Σ_ϕ is implicitly defined by the respective first-round messages. Accordingly, the probability does not start at 50% between 0 and 1 but starts at returning $\perp$ with probability 100% and each new round moves this towards returning Σ_ϕ with probability 100%.

26.2. Security Analysis

26.2.1. Correctness

We start by analyzing the *correctness* of the new protocol.

Lemma 26.2.1 (Correctness). *If the protocol from Fig. 26.1 is instantiated with an Ideally Obfuscated version of the circuit from Fig. 26.2 the protocol is ε-correct with $\varepsilon = (1 - \mathsf{negl}(\kappa))$.*

Proof. Again, in an honest execution the value of $\chi_\phi = c$ and hence the correct message is returned with probability 1, the argumentation here is similar to that of Theorem 25.2.1. We only have to assume that there is a dedicated failure-state $\perp$ differs from any message a party would ever send. □

Protocol Π_{AT^n}

Protocol Π_{AT^n} for realizing n-bit Anonymous Transfer in the fine-grained setting. It is running with a set of 3 parties $(\mathsf{P}_0, \mathsf{P}_1, \mathsf{R})$ where one of P_0 and P_1 acts as sender and R is the receiver.

It is parameterized by an IND\$-CCA-secure *public-key* encryption scheme $\textsc{Pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, a *multi-challenge* IND\$-CPA-secure *symmetric* encryption scheme $\textsc{Ske} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an EUF-CMA-secure signature scheme $\textsc{Sig}\colon \{0,1\}^* \times \{0,1\}^\kappa \mapsto \{0,1\}^m$, and an obfuscated program P_{FG-AT} from Fig. 26.2.

Upon activation , R draws a random $otp \xleftarrow{\$} \{0,1\}^n$ and computes $(\mathsf{k}_\mathsf{R}, \mathsf{vk}_\mathsf{R}) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$. Then R sets $msg_\mathsf{R}^{(0)} \leftarrow \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (otp, \mathsf{vk}_\mathsf{R}))$ and broadcasts $msg_\mathsf{R}^{(0)}$.

On input (ϕ, Σ), P_ϕ computes a signature key pair $(\mathsf{vk}_\phi, \mathsf{k}_\phi) \leftarrow \textsc{Sig}.\mathsf{KeyGen}(1^\kappa)$ and a symmetric key $\mathsf{sk}_\phi \leftarrow \textsc{Ske}.\mathsf{KeyGen}(1^\kappa)$.

Then, P_ϕ computes a signature $\sigma_\phi \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\phi, msg_\mathsf{R}^{(0)})$ and broadcasts $msg_\phi^{(0)} \leftarrow \textsc{Pke}.\mathsf{Enc}(\mathsf{pk}_\mathsf{P}, (\mathsf{sk}_\phi, \mathsf{vk}_\phi)) \| \textsc{Ske}.\mathsf{Enc}(\mathsf{sk}_\phi, \Sigma)$.

Upon activation , $\mathsf{P}_{1-\phi}$ sets uniformly random $msg_{\bar{\phi}}^{(0)}$.

For each round χ **from** 1 **to** c :

P_ϕ computes $\sigma_\phi \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\phi, (msg_0^{(\chi-1)}, msg_1^{(\chi-1)}))$ and sets $msg_\phi^{(\chi)} \leftarrow \textsc{Ske}.\mathsf{Enc}(\mathsf{sk}_\phi, (\Sigma, \sigma_\phi))$.

$\mathsf{P}_{1-\phi}$ broadcasts $msg_{1-\phi}^{(\chi)} \xleftarrow{\$} \{0,1\}^m$.

R computes $\sigma_\mathsf{R} \leftarrow \textsc{Sig}.\mathsf{Sign}(\mathsf{k}_\mathsf{R}, (msg_\mathsf{R}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(c)}, msg_1^{(c)})))$, computes $\Sigma' := \mathsf{P}_{FG-AT}(msg_\mathsf{R}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(c)}, msg_1^{(c)}), \sigma_\mathsf{R})$ and outputs $otp \oplus \Sigma'$.

Figure 26.1.: The protocol Π_{AT^n} for fine-grained n-bit Anonymous Transfer.

26.2.2. Anonymity

Once again, the anonymity takes a lot more effort to analyze. However, the proof is still relatively close to that of single-bit AT leading to the anonymity in Corollary 25.2.16.

$\textsc{Game}_1(\kappa)$: This is the original game (after replacing the obfuscated circuit with oracle access to the circuit, the PRF with an actual random oracle and the adversary with the simulator), where the sending party P_ϕ is chosen uniformly at random.

```
**Program** $\mathrm{P}_{FG-AT}[\mathrm{pk}_{\mathrm{P}}, c]\left(msg_{\mathrm{R}}^{(0)}, \left\{msg_0^{(\chi)}, msg_1^{(\chi)}\right\}_{\chi=0}^{c}\right)$
$(otp, \mathrm{vk}_{\mathrm{R}}) := \textsc{Pke}.\mathrm{Dec}^*\left(\mathrm{sk}_{\mathrm{P}}, msg_{\mathrm{R}}^{(0)}\right),$
$(\mathrm{sk}_0, \mathrm{vk}_0) := \textsc{Pke}.\mathrm{Dec}^*(\mathrm{sk}_{\mathrm{P}}, msg_0^{(0)}[1\colon m]),$
$(\Sigma_0, \sigma_0) := \textsc{Ske}.\mathrm{Dec}^*(\mathrm{sk}_0, msg_0^{(0)}[m+1\colon 2m]),$
$(\mathrm{sk}_1, \mathrm{vk}_1) := \textsc{Pke}.\mathrm{Dec}^*(\mathrm{sk}_{\mathrm{P}}, msg_1^{(0)}[1\colon m]),$
$(\Sigma_1, \sigma_1) := \textsc{Ske}.\mathrm{Dec}^*(\mathrm{sk}_1, msg_1^{(0)}[m+1\colon 2m]),$
**if** $\neg\textsc{Sig}.\mathrm{Vfy}(\mathrm{vk}_{\mathrm{R}}, (msg_{\mathrm{R}}^{(0)}, (msg_0^{(0)}, msg_1^{(0)}), \ldots, (msg_0^{(0)}, msg_1^{(c)})))$ **then** :
    **return** $\bot$
$\chi_0 := [\![\textsc{Sig}.\mathrm{Vfy}(\sigma_0, \mathrm{vk}_0, msg_{\mathrm{R}}^{(0)})]\!] \cdot (c+1),$
$\chi_1 := [\![\textsc{Sig}.\mathrm{Vfy}(\sigma_1, \mathrm{vk}_1, msg_{\mathrm{R}}^{(1)})]\!] \cdot (c+1),$
**foreach** $\chi \in \{1, \ldots, c\}$ **do**:
    **foreach** $\phi \in \{\phi' | \phi' \in \{0,1\}, \chi_\phi = (c+1)\}$ **do**:    / Check for each potential sender.
        $X_\phi := \textsc{Ske}.\mathrm{Dec}^*\left(\mathrm{sk}_\phi, msg_\phi^{(\chi)}\right),$    $\Sigma'_\phi := X_\phi[0\colon (m-1)],$
        $\sigma_\phi := X_\phi[m\colon |X_\phi|]$
        **if** $\neg\textsc{Sig}.\mathrm{Vfy}(\sigma_\phi, \mathrm{vk}_\phi, \tau[\chi-1]) \lor \Sigma_\phi \neq \Sigma'_\phi$ **then** :
            $\chi_\phi := \chi$    / Remember first bad round.
$\phi' := \mathbf{argmax}_\phi(\chi_\phi)$
**return** $otp \oplus \mathrm{CointossS}^{(\tau)}_{((\chi_{\phi'}/c))}(\Sigma_{\phi'}, \bot)$
```

Figure 26.2.: Obfuscated program P_{FG-AT} for an n-bit Anonymous Transfer in the fine-grained setting with c rounds.

GAME$_2(\kappa)$**:** This game follows $\textsc{Game}_1(\kappa)$, but during the simulation of the oracle P_{FG-AT} from Fig. 26.1 the simulation enforces correctness of the challenge transcript τ_C: if the input transcript τ matches the challenge transcript τ_C, it returns Σ_C.

Lemma 26.2.2 (Indistinguishability of $\textsc{Game}_1(\kappa)$ and $\textsc{Game}_2(\kappa)$)**.** *Let* $\textsc{Pke}$ *be an IND\$-CCA secure asymmetric encryption scheme. Let* $\textsc{Ske}$ *be an IND\$-CCA secure symmetric encryption scheme. Let* $\textsc{Sig}$ *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* $\textsc{Game}_1(\kappa)$ *and* $\textsc{Game}_2(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_1(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_2(\kappa)} = 1]| \in \mathrm{negl}(\kappa)$$

Proof. See Lemma 26.2.1. □

GAME$_3(\kappa)$**:** This game follows $\textsc{Game}_2(\kappa)$ but during simulation of the circuit the adversary aborts if any of the first-round messages differ from the messages reported in the challenge transcript *and* the decryptions still match.

Lemma 26.2.3 (Indistinguishability of $\text{GAME}_2(\kappa)$ and $\text{GAME}_3(\kappa)$)**.** *Let PKE be an IND$-CCA secure asymmetric encryption scheme. Let SKE be an IND$-CCA secure symmetric encryption scheme. Let SIG be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_2(\kappa)$ and $\text{GAME}_3(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\mathit{GAME}_2(\kappa)} = 1] - \Pr[\mathtt{out}_{\mathit{GAME}_3(\kappa)} = 1]| \in \mathrm{negl}(\kappa)$$

Proof. The claim is similar to that from Lemma 25.2.4 and thus follows from their proof. □

GAME$_4(\kappa)$: This game follows $\text{GAME}_3(\kappa)$ but simulates the circuit slightly different: If the first receiver message of the input transcript τ is the same as that of the challenge transcript τ_C, instead of decrypting it the circuit directly sets $otp = otp_C$ and $\mathsf{vk}_\mathsf{R} = \mathsf{vk}_{C\mathsf{R}}$ as the values used in the creation of the challenge transcript.

Lemma 26.2.4 (Indistinguishability of $\text{GAME}_3(\kappa)$ and $\text{GAME}_4(\kappa)$)**.** *Let PKE be an IND$-CCA secure asymmetric encryption scheme. Let SKE be an IND$-CCA secure symmetric encryption scheme. Let SIG be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_3(\kappa)$ and $\text{GAME}_4(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\mathit{GAME}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\mathit{GAME}_4(\kappa)} = 1]| \in \mathrm{negl}(\kappa)$$

Proof. This proof also follows from the correctness of the encryption scheme PKE just as Lemma 25.2.5. □

GAME$_5(\kappa)$: This game follows $\text{GAME}_4(\kappa)$ but simulates the oracle differently if the first-round messages of both parties match the first-round messages in the challenge transcript τ_C. In this case, the program compares the input transcript τ with the challenge transcript τ_C until it finds the first round χ^* in which the input differs from the challenge transcript. It then checks round $\chi^* + 1$, and if it contains the same message from the sending party, it adds one to χ.

Finally, the circuit flips a biased coin, which returns the correct message Σ_C with probability $p := 1/2 + \chi^*/2c$ and an error symbol $\perp$ otherwise.

Lemma 26.2.5 (Indistinguishability of $\text{GAME}_4(\kappa)$ and $\text{GAME}_5(\kappa)$)**.** *Let PKE be an IND$-CCA secure asymmetric encryption scheme. Let SKE be an IND$-CCA secure symmetric encryption scheme. Let SIG be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\text{GAME}_4(\kappa)$ and $\text{GAME}_5(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\mathit{GAME}_4(\kappa)} = 1] - \Pr[\mathtt{out}_{\mathit{GAME}_5(\kappa)} = 1]| \in \mathrm{negl}(\kappa)$$

Proof. Note that the unforgeability of a valid round-zero message required for detecting the new branch is similar to Lemma 25.2.6. Thus we can focus entirely on the second claim, namely that the behavior inside the branch reflects that of the actual circuit.

In fact, we have to show that at line 31 in $\textsc{Game}_2(\kappa)$ if the maximum round is χ_ϕ then the input transcript is equal to the challenge transcript until *at least* round $(\chi_\phi - 1)$.

Let $\mathcal{D}$ be a PPT distinguisher who can create input τ for which it holds that χ^* is such that any of the messages from round $(\chi^* - 2)$ differ from the challenge transcript. Then the only way that the verification in round $(\chi^* - 1)$ succeeds is if the sending parties message encodes a valid signature on the changed message from round $(\chi^* - 2)$, which would violate the EUF-CMA security of the used signature scheme $\textsc{Sig}$.

Thus the claim follows. □

$\textsc{Game}_6(\kappa)$: This game is the same as $\textsc{Game}_5(\kappa)$, but in creating the challenge transcript τ_C, this game only reports randomness for the first-round message $\widehat{msg}^0_\phi$ that specifies the symmetric key sk_ϕ to be used for the remaining communication with the circuit. Note that both the signature- and the symmetric-key are still created for the sending party P_ϕ as they are needed for the remaining rounds.

Lemma 26.2.6 (Indistinguishability of $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_6(\kappa)$)**.** *Let $\textsc{Pke}$ be an IND\$-CCA secure asymmetric encryption scheme. Let $\textsc{Ske}$ be an IND\$-CCA secure symmetric encryption scheme. Let $\textsc{Sig}$ be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\textsc{Game}_5(\kappa)$ and $\textsc{Game}_6(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_5(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_6(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. See Lemma 25.2.7. □

$\textsc{Game}_7(\kappa)$: This game is the same as $\textsc{Game}_6(\kappa)$ but in creating the challenge transcript τ_C, this game also reports randomness instead of transcripts for all messages msg^χ_ϕ for $\chi \in [c]$ that shift the message towards Σ_C. That means that instead of using the IND\$-CPA secure symmetric scheme $\textsc{Ske}$ with the symmetric key sk_ϕ the challenge transcript now only contains randomly sampled messages.

We also do not let the adversary create the keys for $\textsc{Sig}$ and $\textsc{Ske}$ as they are no longer needed for creating the transcript.

Lemma 26.2.7 (Indistinguishability of $\textsc{Game}_6(\kappa)$ and $\textsc{Game}_7(\kappa)$)**.** *Let $\textsc{Pke}$ be an IND\$-CCA secure asymmetric encryption scheme. Let $\textsc{Ske}$ be an IND\$-CCA secure symmetric encryption scheme. Let $\textsc{Sig}$ be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\textsc{Game}_6(\kappa)$ and $\textsc{Game}_7(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_6(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_7(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. See Lemma 25.2.8. □

```
$C_{Ber}(c)$
$\beta \xleftarrow{\$} \{0,1\}$
Create $c$ orales $\{O_i\}_{i=1}^{c}$
foreach Oracles $O_i$ do
    if $\beta = 0$ then
        $\Pr[\mathtt{out}_{O_i} = 1] := \frac{i-1}{c}$
    else
        $\Pr[\mathtt{out}_{O_i} = 1] := \frac{i}{c}$
    fi done
```

```
$\mathcal{A}(1^{\kappa})$
for $j = 1 \ldots t$ do
    $i_j \leftarrow Computations$
    $msg_j \xleftarrow{\$} O_{i_j}$
done
$\beta' \leftarrow Computations((i_j, msg_j)_{j=1}^{t})$
return $\beta'$
```

Figure 26.3.: Game to determine whether c Bernoulli-oracles follow a given distribution p_χ or $q_\chi = p - 1/c$.

GAME$_8(\kappa)$: This game follows GAME$_7(\kappa)$, but instead of choosing a random sender at the beginning of the game and considering this message to be the right one, the oracle ignores the additional check and only looks for the first round where *both* messages are identical to the challenge.

Note that this game is entirely independent of the real sender, hence given a challenge transcript it is trivially impossible to obtain a non-negligible advantage to determine the sending party.

Claim 26.2.8 (Indistinguishability of GAME$_7(\kappa)$ and GAME$_8(\kappa)$)**.** *Let $\mathcal{D}$ be a distinguisher with runtime $o(c^2/\alpha)$. Let the cost of acquiring a single sample be $\mathcal{O}(c)$. Then the distinguishing advantage is limited by*

$$|\Pr[\mathtt{out}_{GAME_7(\kappa)} = 1] - \Pr[\mathtt{out}_{GAME_8(\kappa)} = 1]| \leq \alpha \tag{26.1}$$

The actual proof is similar to the single-bit case, only that we do not differentiate a *Bernoulli* oracle that returns a single bit but one that returns either a n-bit message or an error state. However, we slightly adapt the game from Fig. 25.3 to our new scenario; basically instead of ranging from 0.5 to 1 the oracles now range from 0 to 1 in equally-sized steps:

Lemma 26.2.9. *Let $\mathcal{D}$ be a distinguisher distinguishing GAME$_7(\kappa)$ and GAME$_8(\kappa)$ with advantage α over guessing. Let t be the number of queries that $\mathcal{D}$ sends to the obfuscated circuit. There is a reduction adversary $\mathcal{A}$ that uses $\mathcal{D}$ which has advantage α over guessing in winning Fig. 26.3.*

Proof. The overall idea of the proof is similar to that used in Fig. 25.3 but has some minor changes in order to incorporate the different range of the oracles and the fact that instead of 0 or 1, the oracle returns $\perp$ or Σ.

Creating the transcript. The transcript is the same in both games and hence can be created canonically. Upon activation the adversary $\mathcal{A}$ samples the required information for the challenge transcript, that is, $\Sigma_C \overset{\$}{\leftarrow} \{0,1\}^n$ and $\phi \overset{\$}{\leftarrow} \{0,1\}$, and creates and stores a transcript τ_C by sampling uniformly random messages for both parties. This transcript is then reported to the distinguisher.

Simulating the circuit. When $\mathcal{D}$ sends some input to the simulated circuit $\mathcal{A}$ has to return some message Σ according to the respective distributions. There are several possible inputs for the circuit but the below list covers all the possibilities:

(At least) one of the first-round messages is different. Then the adversary simulates the circuit by following the actual protocol. This case is equivalent for both games.

First different message for both parties in round $(\chi + 1)$. Then the adversary flips a biased coin that lands on heads with probability $p := \chi/c$ and on heads $\mathcal{A}$ returns Σ and otherwise $\mathcal{A}$ returns $\perp$.

First different message only for P_ϕ in round $(\chi + 1)$. This case is equivalent to the one before and does not change in the game hop.

First different message only for $\mathsf{P}_{\overline{\phi}}$ in round $(\chi + 1)$. This is the interesting case as this class contains all the transcripts that are actually treated differently in the two games.

In this case the adversary queries the $(\chi + 1)$st oracle that returns 1 with probability χ/c if $\beta = 0$ and with probability $(\chi + 1)/c$ if $\beta = 1$. When receiving output 0 the adversary returns $\perp$ as oracle-output. On output 1 the adversary returns the challenge message Σ_C.

It is easy to see that this reflects the two games; if $\beta = 0$ then Σ is returned with probability χ/c as it is in $\textsc{Game}_7(\kappa)$, and if $\beta = 1$ then Σ is returned with probability $(\chi + 1)/c$ as in $\textsc{Game}_8(\kappa)$.

Using the response. After (at most) t queries the distinguisher terminates and sends its guess. If $\mathcal{D}$ guesses it was playing $\textsc{Game}_8(\kappa)$ then $\mathcal{A}$ reports $\beta' = 1$ to C_{Ber}. And if $\mathcal{D}$ guesses it was playing $\textsc{Game}_7(\kappa)$ then $\mathcal{A}$ sends the output $\beta' = 0$ to C_{Ber}.

As was described above the two distributions can be perfectly simulated, hence the probability that $\mathcal{D}$ guesses the correct game is the same that $\mathcal{A}$ has to correctly guess β. So if $\mathcal{D}$ has non-negligible advantage α over guessing then $\mathcal{A}$ inherits this advantage for winning the game from Fig. 26.3, albeit if $\mathcal{D}$ sends t queries to $\mathcal{A}$, less than t queries are forwarded to C_{Ber}. $\square$

We now have a new game in Fig. 26.3 which we used for a more intuitive proof, however we still need to analyze how hard winning Fig. 26.3 really is. Fortunately, the game is already quite close to Fig. 25.3 only with a larger range. We thus borrow ideas from the proof of the single-bit case and adapt them to our new scenario.

Note that Lemma 22.2.4 can also be applied, and that Corollary 22.2.5 also holds for Fig. 26.3.

We can thus skip towards adjusting Corollary 25.2.10, for which we first need to embed Corollary 22.2.5 into our problem setting: We have c instances where the χ-th instance corresponds to distinguishing $p := (\chi - 1)/c$ from $q := (\chi)/c$. This implies the following L_1-norm between p and q in round χ:

$$\begin{aligned}
\mathrm{d}_{\mathrm{TV}}(p,q) &= \frac{\|p-q\|_1}{2} \\
&= \frac{1}{2}\big(|\Pr[p=1]-\Pr[q=1]| \\
&\quad + |\Pr[p=0]-\Pr[q=0]|\big) \\
&= \frac{1}{2}\left(\left|\frac{\chi-1}{c}-\frac{\chi}{c}\right| + \left|\frac{\chi+1}{c}-\frac{\chi}{c}\right|\right) \\
&= \frac{1}{2}\left(\frac{2}{c}\right) \\
&= \frac{1}{c}
\end{aligned} \tag{26.2}$$

This implies again that the total variational distance in round χ is *independent* of the round χ and hence the same for all oracles. So when combining this observation with the subadditivity of the total variational distance (Lemma 22.2.3) we get that any distribution resulting from t samples have a total variational distance of at most $\frac{t}{c}$. Thus we get:

$$t \in \Omega(\alpha/\mathrm{d}_{\mathrm{TV}}(p,q)) = \Omega(\alpha \cdot c) \tag{26.3}$$

We thus end up with the following corollary.

Corollary 26.2.10. *Let $\mathcal{D}$ be a distinguisher playing the game from Fig. 26.3 using a fixed number t of samples and has runtime $\mathrm{o}(c^2/\alpha)$. Let the cost to acquire a single sample be $O(c)$. Then the distinguisher $\mathcal{D}$ is correct with probability at most $1/2+\alpha/2$.*

Proof. See Corollary 25.2.10. Eq. (26.3) yields a lower bound for the number of samples required by $\mathcal{D}$ to distinguish with advantage α. With each sample having a cost of $O(c)$ an advantage of α requires runtime in $O(c^2 \cdot \alpha)$ to get the samples alone. With the runtime of $\mathcal{D}$ being bounded as $\mathrm{o}(c^2/\alpha)$ it follows that $\mathcal{D}$ cannot have constant advantage. □

26.2.3. Secrecy

We conclude this section with a secrecy analysis of the protocol. Fortunately, this proof is quite close to that from the single-bit protocol from Theorem 25.2.11. The main difference is that we have to change an entire message instead of only changing the single bit.

However, the crucial part is still the statistical security of the One Time Pad, which holds also in the multi-bit case. To prove our claim we thus show that the transfer of a uniformly random message cannot be distinguished efficiently from the transfer of the all-zero bitstring without the receiver's random tape.

In total we want to show that:

$$\begin{aligned} &\{\tau \leftarrow \mathsf{Transfer}_{\langle \mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle}(crs, \phi, \Sigma)\} \\ \approx &\{\tau \leftarrow \mathsf{Transfer}_{\langle \mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle}(crs, \phi, \vec{0}^n)\} \end{aligned} \tag{26.4}$$

With that in mind we adjust the gamehops as follows:

$\textsc{Game}_1(\kappa)$: This is the original game (after replacing the obfuscated circuit with oracle access to the circuit, the PRF with an actual random oracle and the adversary with the simulator), where the sending party P_ϕ is chosen uniformly at random and the sender uses the input message Σ.

$\textsc{Game}_2(\kappa)$: This game follows $\textsc{Game}_1(\kappa)$, but with the following changes:

- During the simulation of the oracle P_{FG-AT} from Fig. 26.1 the simulation enforces correctness of the challenge transcript τ_C: if the input transcript τ matches the challenge transcript τ_C, it returns $otp_C \oplus \Sigma_C$.
- During simulation of the circuit the adversary aborts if any of the first-round messages from P_0 or P_1 differ from the messages reported in the challenge transcript *and* the decryptions still match.
- During simulation of the circuit, if the first receiver message of the input transcript τ is the same as that of the challenge transcript τ_C, instead of decrypting it the circuit directly sets $otp = otp_C$ and $\mathsf{vk}_\mathsf{R} = \mathsf{vk}_{C\mathsf{R}}$ as the values used in the creation of the challenge transcript.
- During simulation of the oracle, if the first-round messages of both parties match the first-round messages in the challenge transcript τ_C. In this case, the program compares the input transcript τ with the challenge transcript τ_C until it finds the first round χ^* in which the input differs from the challenge transcript. It then checks round $\chi^* + 1$, and if it contains the same message from the sending party, it adds one to χ.

  Finally, the circuit flips a biased coin, which returns the correct message $otp_C \oplus \Sigma_C$ with probability $p := \chi^*/c$ and an error message $\perp$ otherwise.

Lemma 26.2.11. *Let* PKE *be an* IND\$-CCA *secure public-key encryption scheme. Let* SKE *be an* IND\$-CCA *secure secret-key encryption scheme. Let* SIG *be an* sEUF-CMA *secure signature scheme. For all* PPT *guessing algorithms* A_{guess}*, the distinguishing advantage for* $\text{GAME}_1(\kappa)$ *and* $\text{GAME}_2(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\text{GAME}_1(\kappa)} = 1] - \Pr[\mathtt{out}_{\text{GAME}_2(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Follows from Lemmas 26.2.2 to 26.2.7. □

$\text{GAME}_3(\kappa)$: This game follows $\text{GAME}_2(\kappa)$ but the oracle is simulated slightly different: If the first receiver message is the same as the one reported in the challenge transcript and the input transcript is *not* the challenge transcript, then the circuit outputs an error symbol $\perp$.

Note that this does not work in the anonymity proof as there we assume that the adversary is given access to the receivers random tape, and hence can create their own new signature on the modified transcript.

Lemma 26.2.12. *Let* PKE *be an* IND\$-CCA *secure public-key encryption scheme. Let* SKE *be an* IND\$-CCA *secure secret-key encryption scheme. Let* SIG *be an* sEUF-CMA *secure signature scheme. For all* PPT *guessing algorithms* A_{guess}*, the distinguishing advantage for* $\text{GAME}_3(\kappa)$ *and* $\text{GAME}_4(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\text{GAME}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\text{GAME}_4(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce a distinguisher $\mathcal{D}$ between these two games to an adversary $\mathcal{A}$ on the EUF-CMA security of the signature scheme SIG.

Creating the transcript. The creation of the transcript is straightforward and works by sampling random messages for each party.

Simulating the oracle. The used verification key vk_R is set to be the verification key of the challenger. Thus the final signature σ_R on the entire transcript, which is required for the circuit to not *abort* (by outputting a random bit), needs to be forged. Hence for each input transcript the adversary first checks if the receiver message is equivalent to that from the challenge transcript. If it isn't then the transcript cannot be used for distinguishing anyways. If it is, the adversary checks if the remaining transcript is the same as well. If it is, the transcript cannot be used for distinguishing and simulation just continues as the path taken is equivalent in both games. If it isn't then the adversary checks if the signature σ_R verifies under the used key. If it doesn't then both games act exactly the same and output an error state $\perp$. Hence to distinguish the signature has to verify. Then, however, the signature is a valid forgery.

Translating the result. Assuming that valid signature was queried then—as mentioned above—the distinguishing advantage must be negligible. For a non-negligible advantage the distinguisher has to query at least one signature. This can be used as forgery to break the EUF-CMA security of $\textsc{Sig}$.

As stated above, the advantage of the distinguisher is directly related to the probability of successfully distinguishing. Hence $\mathcal{A}$ will have a valid forgery with non-negligible advantage.

This would contradict the EUF-CMA security of the signature scheme and thus completes our proof. □

$\textsc{Game}_4(\kappa)$: This game is as $\textsc{Game}_3(\kappa)$ but instead of using a uniformly random Σ for the transferred message during simulation of the circuit we now fix $\Sigma = \vec{0}^n$ as the all-zero bitstring of appropriate size.

Lemma 26.2.13. *Let $\textsc{Pke}$ be an IND\$-CCA secure public-key encryption scheme. Let $\textsc{Ske}$ be an IND\$-CCA secure secret-key encryption scheme. Let $\textsc{Sig}$ be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\textsc{Game}_3(\kappa)$ and $\textsc{Game}_4(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_3(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Again indistinguishability follows for any adversary who does not have the receivers random tape. The message is only ever used in its encrypted form via the uniformly random One Time Pad, and distinguishing two bitstrings after they have been masked is statistically impossible to do with non-negligible advantage. Thus indistinguishability follows. □

$\textsc{Game}_5(\kappa)$: This game follows $\textsc{Game}_4(\kappa)$ but undoes all the changes from the first-to-second gamehop and that from $\textsc{Game}_3(\kappa)$.

Lemma 26.2.14. *Let $\textsc{Pke}$ be an IND\$-CCA secure public-key encryption scheme. Let $\textsc{Ske}$ be an IND\$-CCA secure secret-key encryption scheme. Let $\textsc{Sig}$ be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for $\textsc{Game}_4(\kappa)$ and $\textsc{Game}_5(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_4(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_5(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. Follows from Lemmas 26.2.2 to 26.2.7 and 26.2.12. □

Thus *otp* is hidden to any guessing algorithm and so is Σ.

Corollary 26.2.15. *The Anonymous Transfer from Fig. 26.1 is ς-secret with $\varsigma \in \mathsf{owhl}(\kappa)$.*

In total, we thus have:

Corollary 26.2.16. *The protocol Π_{AT^n} from Fig. 26.1 is a strong FC_1-fine-grained $(1 - \mathsf{negl}(\kappa), \delta, 1 - \mathsf{negl}(\kappa))$-AT in the fine-grained setting against $\mathsf{FC}_2 := \mathrm{o}(c^2/\delta)$.*

27. Undetectable Oblivious Transfer

For a candidate application of Anonymous Transfer we propose Undetectable Oblivious Transfer (UOT): A K party protocol with $N = 2$ participants, one sender and one receiver, performing classical Oblivious Transfer without (1) the sender knowing which of the $K - 1$ other individuals is the receiver, (2) the receiver knowing which of the $K - 1$ other individuals is the sender, and (3) the dummy friends knowing that a computation is in progress at all.

Our construction necessarily requires the secrecy of AT which we defined in Section 23.6. We define in Section 27.1 the security properties of an Undetectable Oblivious Transfer scheme.

In Chapter 28 we will provide a candidate instantiation for $K = 3$ that can be canonically extended to any number $K > 3$.

27.1. Definitions for Undetectable Oblivious Transfer

Towards our goal of Undetectable Multi-Party Computation we consider a notion of Undetectable Oblivious Transfer. In Undetectable Oblivious Transfer we have a set of K parties. Two of these parties want to run an Oblivious Transfer protocol without the participants learning with whom they executed the OT, and without the $K-2$ other parties realizing that the protocol is executed.

Definition 27.1.1 (($\varepsilon, \delta, c, n$)-Undetectable Oblivious Transfer). *An $(\varepsilon, \delta, c, n)$-Undetectable Oblivious Transfer for $\varepsilon, \delta \in \mathbb{R}_{[0,1]}$ and $n, c \in \mathbb{N}$ is a tuple containing three PPT algorithms* (Setup, OT, Reconstruct). *The number of rounds in the* OT *protocol is given as c and the bitlength n defines the length of the transferred message Σ_b.*

The algorithms are defined as follows:

Setup(1^κ) *takes as input the security parameter 1^κ in unary encoding and outputs a Common Reference String crs.*

OT$(crs, \varpi, \Sigma_0, \Sigma_1, b)$ *is a c-round protocol that takes as input the Common Reference String crs, a permutation $\varpi \in \mathbb{S}_K$ to determine which of $(\mathsf{P}_1, \dots, \mathsf{P}_K)$ is the sender, which is the receiver and which are the dummy friends, two messages Σ_0 and $\Sigma_1 \in \{0,1\}^n$ from the sender and one bit $b \in \{0,1\}$ from the receiver and outputs a transcript τ. All non-sender send independent uniformly distributed random strings in each round.*

```
$\mathrm{Exp}^{\text{anon-ot}}_{\Pi_{UOT},\mathcal{A},\varpi}(\kappa)$
$crs \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$
$(\Sigma_0, \Sigma_1, \mathsf{P}, st) \leftarrow \mathsf{A}_{guess_0}(crs)$
$\tau \xleftarrow{\$} \mathsf{OT}_{\varpi(\mathsf{P}_1,\ldots,\mathsf{P}_K)}(crs, \phi, \Sigma)$
return $\mathsf{A}_{guess_1}(\tau, T_{\mathsf{P}}, st)$
```

Figure 27.1.: Definition of the game $\mathrm{Exp}^{\text{anon-ot}}_{\Pi_{UOT},\mathcal{A},\varpi}(\kappa)$.

$\mathsf{Reconstruct}(crs, \tau, T_{\mathsf{R}})$ *is a local algorithm executed by the receiver that takes as input the CRS crs, protocol transcript τ and the receiver's random tape T_{R} and outputs a message Σ_b.*

The algorithms additionally satisfy the ε-correctness, the δ-anonymity and the privacy from Definitions 27.1.2 to 27.1.4.

Definition 27.1.2 (ε-Correctness). *An Undetectable Oblivious Transfer protocol between players $(\mathsf{P}_1, \ldots, \mathsf{P}_K)$ is ε-Correct if for any $\varpi \in \mathbb{S}_K$, any $(\Sigma_0, \Sigma_1) \in \{0,1\}^{2n}$, any $b \in \{0,1\}$ and any $crs \leftarrow \mathsf{Setup}(1^\kappa)$, it holds that*

$$\Pr\left[\Sigma_b = \Sigma'_b \;\middle|\; \begin{array}{l} \tau \xleftarrow{\$} \mathsf{OT}_{\langle \varpi(\mathsf{P}_1,\ldots,\mathsf{P}_K)\rangle}(crs, (\Sigma_0, \Sigma_1), b) \\ \Sigma'_b \leftarrow \mathsf{Reconstruct}(crs, \tau, T_{\mathsf{R}}) \end{array} \right] \geq \varepsilon \tag{27.1}$$

This is quite similar to the definition of correctness for AT from Definition 23.2.2.

Definition 27.1.3 (δ-Anonymity). *An Undetectable Oblivious Transfer protocol between players $(\mathsf{P}_1, \ldots, \mathsf{P}_K)$ is δ-Anonymous if for any PPT guessing algorithm $\mathsf{A}_{guess} = (\mathsf{A}_{guess_0}, \mathsf{A}_{guess_1})$, it holds that*

$$\left| \Pr_{\varpi \xleftarrow{\$} \mathbb{S}_K}\left[Exp^{anon\text{-}ot}_{\Pi_{UOT},\mathsf{A}_{guess},\varpi}(\kappa) = \varpi \right] - 1/(K-1) \right| \leq (1-\delta) \cdot \frac{(K-2)}{(K-1)} \tag{27.2}$$

$\mathbb{S}_K$ is the symmetric group over K elements alongside all possible bijections into themselves and ϖ is a random permutation drawn from that set. This reflects the intuition that a *participant* is trying to recover the permutation *after* finishing the execution. The guessing algorithm A_{guess_0} selects which party to *corrupt* (in a semi-honest setting, hence post-execution) and which messages should be transferred. A_{guess_1} gets the *random tape* of the corrupted party and the *state* st of the adversary that selected which party to corrupt. The definition implies uncertainty of each party regarding the role of each other individual; the sender is unaware which of the two parties present acts as receiver, the receiver is unable to determine the sender better than by guessing, and the dummy friend does not know any of the other individuals roles either.

Definition 27.1.4 (Privacy). *An Undetectable Oblivious Transfer protocol between players $(\mathsf{P}_1, \ldots, \mathsf{P}_K)$ is private for the sender and the receiver if for any guessing algorithm A_{guess},* *any* $\varpi \in \mathbb{S}_K$, *any* $(\Sigma_0, \Sigma_1) \in \{0,1\}^{2n}$, *any* $b \in \{0,1\}$ *and any* $crs \leftarrow \mathsf{Setup}(1^\kappa)$, *the following two conditions hold against every PPT guessing algorithm* A_{guess}*:*

$$\Pr\left[\ \Sigma_{\overline{b}} = \Sigma'_{\overline{b}}\ \middle|\ \begin{array}{l} \tau \leftarrow \mathsf{OT}_{\varpi(\mathsf{P}_1,\ldots,\mathsf{P}_K)}(crs, (\Sigma_0, \Sigma_1), b) \\ \Sigma'_{\overline{b}} \leftarrow \mathsf{A}_{guess}(\tau, T_\mathsf{R}) \end{array}\right] \in \mathsf{negl}(\kappa) \tag{27.3}$$

$$\left|\Pr\left[\ b = b'\ \middle|\ \begin{array}{l} \tau \leftarrow \mathsf{OT}_{\varpi(\mathsf{P}_1,\ldots,\mathsf{P}_K)}(crs, (\Sigma_0, \Sigma_1), b) \\ b' \leftarrow \mathsf{A}_{guess}(\tau, T_\mathsf{S}) \end{array}\right] - 1/2\right| \in \mathsf{negl}(\kappa) \tag{27.4}$$

For the sender the requirement from Eq. (27.3) means that even with the random tape T_R of the receiver it is not efficiently possible for the guessing algorithm A_{guess} to predict the message $\Sigma_{\overline{b}}$ that was *not* transferred. For the receiver the requirement from Eq. (27.4) implies security of the choice despite having the random tape T_S of the sender.

28. Undetectable Oblivious Transfer Instantiation

In this section we present an instantiation of Undetectable Oblivious Transfer based on any two-round Covert Oblivious Transfer (COT) protocol with overwhelming correctness such as the one from von Ahn, Hopper, and Langford [133, Protocol 4]. The protocol is fairly canonical and basically just uses a different communication structure where messages are sent via Anonymous Transfer.

In Fig. 28.1 we present a protocol that takes any two-round Covert Oblivious Transfer protocol and a strong $(\varepsilon, \delta, \varsigma, c, n)$-Anonymous Transfer and constructs an (ε, δ)-Undetectable Oblivious Transfer. For the sake of simplicity we only introduce a generic three-party protocol (that has one dummy friend), but we stress that the same technique can easily be modified to incorporate more parties at the cost of using 2 more ATs with the respective dummy friend as the receiver.

28.1. Correctness

For analyzing the correctness we assume that the Covert Oblivious Transfer protocol reconstructs the bit correctly with overwhelming probability. Then it holds for the correctness of the Undetectable Oblivious Transfer:

Lemma 28.1.1 (Correctness of the Undetectable Oblivious Transfer protocol)**.** *Let Π_{COT} be a COT protocol that is correct with overwhelming probability. Let Π_{AT^n} be a strong $(\varepsilon, \delta, \varsigma, c, n)$-AT. Then the Undetectable Oblivious Transfer protocol from Fig. 28.1 is correct with $\varepsilon_{UOT} = \frac{\varepsilon^2+2\varepsilon-1}{2}$.*

Proof. When executed directly, *i.e.* with direct communication instead of using AT, Π_{COT} is correct as long as all messages are present, and it is wrong if a message was transferred incorrectly. As our only change lies in the communication channels it holds that the probability that the message has been transferred correctly corresponds to the probability that *both* protocol messages have been transferred correctly—the first-round message by the receiver to the sender and the second-round message by the sender to the receiver.

Protocol Π_{UOT}

Two-round protocol Π_{UOT}. It is running with a set of three parties $(\mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_2)$, where we denote by P_S the sender, by P_R the receiver, and by P_D the non-participating party. It is parameterized by a two-round Covert Oblivious Transfer protocol Π_{COT} and a $(\varepsilon, \delta, \varsigma)$-Anonymous Transfer protocol Π_{AT}.

On input (Σ_0, Σ_1), P_S stores both Σ_0 and Σ_1 and sets $msg_0 := \perp$.

On input $b \in \{0, 1\}$, P_R chooses msg_1 as the first message the receiver sends to the sender in Π_{COT} for bit b.

On input $\perp$, P_D sets $msg_\mathsf{D} := \perp$.

Each party P_i sends msg_i to both instances of Π_{AT} where party P_j for $j \neq i$ is the receiver.

On input msg_0^* from Π_{AT} where P_S is the receiver, P_S computes msg_0 according to Π_{COT} on input (Σ_0, Σ_1) and first message msg_0^*.

On input $\perp$ from Π_{AT} where P_R is the receiver, P_R sets $msg_1 := \perp$.

On input $\perp$ from Π_{AT} where P_D is the receiver, P_D sets $msg_2 := \perp$.

Each party P_i sends msg_i to both instances of Π_{AT} where party P_j for $j \neq i$ is the receiver.

On input $\perp$ from Π_{AT} where P_S is the receiver, P_S outputs $\perp$.

On input msg_0 from Π_{AT} where P_R is the receiver, P_R reconstructs Σ_b according to Π_{COT} on inputs (msg_0, msg_1, b) and outputs Σ_b.

On input $\perp$ from Π_{AT} where P_D is the receiver, P_D outputs $\perp$.

Figure 28.1.: The two-round protocol Π_{UOT} for Undetectable Oblivious Transfer with given protocols for Anonymous Transfer and Covert Oblivious Transfer.

With the AT being ε-correct it follows from Eq. (23.1) that

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ \tau \xleftarrow{\$} \langle \mathsf{R}, \mathsf{P}_0, \mathsf{P}_1 \rangle(\phi, b) \\ b' \leftarrow \mathsf{f}(\tau, T_\mathsf{R}) \end{array} \right] \geq (\varepsilon + 1)/2 \tag{28.1}$$

The probability that *both* messages are transferred correctly squares that value. As such it holds that:

$$\left(\Pr\left[\, b = b' \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ \tau \xleftarrow{\$} \langle \mathsf{R}, \mathsf{P}_0, \mathsf{P}_1 \rangle(\phi, b) \\ b' \leftarrow \mathrm{f}(\tau, T_{\mathsf{R}}) \end{array} \right]\right)^2 \geq ((\varepsilon + 1)/2)^2 = \frac{1}{4}(\varepsilon^2 + 2\varepsilon + 1) \tag{28.2}$$

So we have for the correctness of the OT:

$$\begin{aligned} &\Pr\left[\, msg_b = msg'_b \;\middle|\; \begin{array}{l} \varpi \xleftarrow{\$} \mathbb{S}_3, \\ (msg_0, msg_1) \xleftarrow{\$} \{0,1\}^2, \\ b \xleftarrow{\$} \{0,1\}, \\ msg'_b \leftarrow \textsc{Ot}(\varpi\{(msg_0, msg_1), b, \perp\}) \end{array} \right] \\ \geq &\frac{1}{4}(\varepsilon^2 + 2\varepsilon + 1) = \frac{\varepsilon^2}{4} + \frac{\varepsilon}{2} + \frac{1}{4} \end{aligned} \tag{28.3}$$

and hence, in the form of Eq. (27.1),

$$\begin{aligned} &\Pr\left[\, msg_b = msg'_b \;\middle|\; \begin{array}{l} \varpi \xleftarrow{\$} \mathbb{S}_3, \\ (msg_0, msg_1) \xleftarrow{\$} \{0,1\}^2, \\ b \xleftarrow{\$} \{0,1\}, \\ msg'_b \leftarrow \textsc{Ot}(\varpi\{(msg_0, msg_1), b, \perp\}) \end{array} \right] - \frac{1}{2} \\ \geq &\frac{\varepsilon^2}{4} + \frac{\varepsilon}{2} - \frac{1}{4} = \frac{1}{2} \cdot \left(\frac{\varepsilon^2 + 2\varepsilon - 1}{2}\right) \end{aligned} \tag{28.4}$$

Thus we get that according to Eq. (27.1) we have $\varepsilon_{UOT} := \frac{\varepsilon^2 + 2\varepsilon - 1}{2}$. □

28.2. Privacy

The analysis of (input-) privacy is pretty straightforward, simply because our transformation leaves no extra insecurities that would allow extraction of either the choice bit b or the message $\Sigma_{\bar{b}}$ *not* selected by the receiver. Yet in the following we provide a full proof of privacy.

28.2.1. Sender Privacy

We first define the privacy notion of classical OT protocols in a game-based notion:

Definition 28.2.1 (Sender-Privacy in Oblivious Transfer)**.** *Let* $\textsc{Ot}$ *be an oblivious transfer protocol.* $\textsc{Ot}$ *provides computational* sender-privacy *if for all PPT adversaries* $\mathcal{A}$ *it holds that the following probability is negligible in the security parameter* κ*:*

$$\Pr\left[\, (msg_0, msg_1) = (msg_0^*, msg_1^*) \;\middle|\; \begin{array}{l} (msg_0, msg_1) \xleftarrow{\$} \{0,1\}^{2n}, \\ (b, st) \leftarrow \mathcal{A}_0(1^\kappa), \\ \tau \leftarrow \textsc{Ot}((msg_0, msg_1), b), \\ (msg_0^*, msg_1^*) \leftarrow \mathcal{A}_1(\tau, st, T_{\mathsf{R}}) \end{array} \right] \tag{28.5}$$

The definition enforces that the adversary cannot extract *both* bits input by the sender. While the bit msg_b is easy to extract given the transcript, the state of $\mathcal{A}_0$ and the random tape T_{R}, we enforce that the other bit $msg_{\bar{b}}$ cannot be determined better than by guessing.

Definition 28.2.2 (Receiver-Privacy in Oblivious Transfer)**.** *Let* $\textsc{Ot}$ *be an Oblivious Transfer protocol.* $\textsc{Ot}$ *provides computational* receiver-privacy *if for all PPT adversaries* $\mathcal{A}$ *it holds that:*

$$\left| \Pr\left[b = b^* \;\middle|\; \begin{array}{l} (msg_0, msg_1, st) \leftarrow \mathcal{A}_0(1^\kappa), \\ b \xleftarrow{\$} \{0,1\}, \\ \tau \leftarrow \textsc{Ot}((msg_0, msg_1), b), \\ b^* \leftarrow \mathcal{A}_1(\tau, st, T_{\mathrm{S}}) \end{array} \right] - 1/2 \right| \in \mathrm{negl}(\kappa) \tag{28.6}$$

This definition is practically the same as that for sender-privacy in Definition 28.2.1 but lets the adversary play as the (semi-honest) *sender* who has to recover the choice bit by the *receiver.*

With that we can prove the following claim:

Lemma 28.2.3 (Privacy of the Undetectable Oblivious Transfer protocol)**.** *Let* Π_{COT} *be an OT protocol with overwhelming sender-privacy. Then* Π_{UOT} *from Fig. 28.1 is private for both parties.*

Proof. We show separately the sender and receiver privacy.

Sender-Privacy

Any COT protocol automatically fulfills the privacy requirement of an ordinary OT from Definition 28.2.1, as such we reduce sender privacy of Π_{UOT} to the sender privacy of Π_{COT}. To that end, let A_{guess} be a guessing algorithm that breaks the sender privacy of Π_{UOT} with non-negligible advantage α. From A_{guess} we construct an adversary $\mathcal{A}$ who breaks the condition from Eq. (28.5) as follows:

$\mathcal{A}$ asks A_{guess_0} for receiver input b and hands this to the challenger C_{COT} of the COT sender privacy game. From C_{COT}, $\mathcal{A}$ obtains a transcript τ of sent messages. $\mathcal{A}$ simulates inserting each sender message of τ into Π_{AT^n} with receivers $\mathrm{P_R}$ and $\mathrm{P_D}$ and each receiver message into Π_{AT} with receivers $\mathrm{P_S}$ and $\mathrm{P_D}$.

This results in a transcript of Π_{UOT} which $\mathcal{A}$ hands to A_{guess}. Eventually A_{guess} returns a tuple (msg_0^*, msg_1^*) which $\mathcal{A}$ hands directly to the challenger C_{COT}.

Note that if A_{guess} is correct, then so is $\mathcal{A}$, and if A_{guess} is incorrect then the guess of $\mathcal{A}$ is also wrong, as the simulation did not change the sent messages.

Thus if A_{guess} has non-negligible advantage then so has $\mathcal{A}$, which it does not have by requirement.

Receiver Privacy

P_0 sender	P_1 sender
Π_{AT} with receiver $\underline{\mathrm{P}_S:}$	Π_{AT} with receiver $\underline{\mathrm{P}_S:}$
$(msg_{COT^R}, \perp)$	$(msg_{COT^R}, \perp)$
Π_{AT} with receiver $\underline{\mathrm{P}_0:}$	Π_{AT} with receiver $\underline{\mathrm{P}_0:}$
$(\perp, \perp)$	$(msg_{COT^R}, \perp)$
Π_{AT} with receiver $\underline{\mathrm{P}_1:}$	Π_{AT} with receiver $\underline{\mathrm{P}_1:}$
$(msg_{COT^R}, \perp)$	$(\perp, \perp)$
Π_{AT} with receiver $\underline{\mathrm{P}_S:}$	Π_{AT} with receiver $\underline{\mathrm{P}_S:}$
$(\perp, \perp)$	$(\perp, \perp)$
Π_{AT} with receiver $\underline{\mathrm{P}_0:}$	Π_{AT} with receiver $\underline{\mathrm{P}_0:}$
$(msg_{COT^S}, \perp)$	$(msg_{COT^S}, \perp)$
Π_{AT} with receiver $\underline{\mathrm{P}_1:}$	Π_{AT} with receiver $\underline{\mathrm{P}_1:}$
$(msg_{COT^S}, \perp)$	$(msg_{COT^S}, \perp)$

Figure 28.2.: The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the sender.

We let A_{guess} be a guessing algorithm attacking the receiver privacy of Π_{UOT} and construct an adversary $\mathcal{A}$ attacking the receiver privacy of Π_{COT} with the same success probability. The adversary asks A_{guess} for the senders input $(msg_0, msg_1) \in \{0, 1\}^2$ and forwards that to the challenger C_{COT}. The obtained transcript τ is translated into a transcript for Π_{UOT} by simulating the AT with the inputs being the messages reported by the respective parties, where the input by the dummy friend is constantly $\perp$.

$\mathcal{A}$ then hands the transcript over to A_{guess} who guesses a bit b. The bit is then forwarded to C_{COT}.

Again, the transformation did not change any values, hence if A_{guess} is correct with probability $1/2 + \alpha$ then $\mathcal{A}$ is also correct with probability $1/2 + \alpha$. The latter, however, is not possible by requirement from Π_{COT}, which concludes our proof. □

28.3. Anonymity

As the adversary is given the choice to corrupt any party we consider each potential party individually and show the two distributions that need to be indistinguishable.

28.3.1. Corrupted Sender

The guessing algorithm has to distinguish the following two distributions:

$$\begin{aligned}&\{(\Sigma_0,\Sigma_1)\overset{\$}{\leftarrow}\{0,1\}^{2n}, b\overset{\$}{\leftarrow}\{0,1\}: \tau\overset{\$}{\leftarrow}\langle \mathsf{P}_\mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle(b,(\Sigma_0,\Sigma_1),\perp)\}\\ \approx&\{(\Sigma_0,\Sigma_1)\overset{\$}{\leftarrow}\{0,1\}^{2n}, b\overset{\$}{\leftarrow}\{0,1\}: \tau\overset{\$}{\leftarrow}\langle \mathsf{P}_\mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle(b,\perp,(\Sigma_0,\Sigma_1))\}\end{aligned} \tag{28.7}$$

In this section we prove anonymity against any adversary that corrupts the sender after the execution. This adversary then has to guess which of the remaining parties acted as a receiver and which was the dummy friend.

The two possible transcripts if he sender is fixed are depicted in Fig. 28.2.

The single game required for that—which is actually the same as in Fig. 28.2.

The formal description is as follows:

GAME$_1(\kappa)$: This is the original game where the sender is P_S and the receiver is P_R.

GAME$_2(\kappa)$: This game is as GAME$_1(\kappa)$ but in providing the transcript the simulator follows GAME$_1(\kappa)$ for the first round but inserts $msg_\mathsf{R}^{(0)}$ in the name of the *dummy friend* and $msg_\mathsf{D}^{(0)}$ in the name of the *receiver* into $\Pi_{AT}(\mathsf{S})$. Note that this corresponds to the final distribution from Fig. 28.2.

Lemma 28.3.1. *Let PKE be an IND\$-CCA secure public-key encryption scheme. Let SKE be an IND\$-CCA secure secret-key encryption scheme. Let SIG be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for GAME$_1(\kappa)$ and GAME$_2(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{\mathrm{GAME}_1(\kappa)} = 1] - \Pr[\mathtt{out}_{\mathrm{GAME}_2(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce to the δ-anonymity of Π_{AT}. If there was some distinguisher $\mathcal{D}$ who can distinguish GAME$_1(\kappa)$ from GAME$_2(\kappa)$ with advantage $1/2+\alpha$ then there is some adversary $\mathcal{A}$ (which implies a guessing algorithm A_{guess}) who can determine the sender with advantage α. The limit δ then provides an upper bound on α.

Creating the transcript. We denote by C_{AT} the challenger for the anonymity-game of Π_{AT}. $\mathcal{A}$ determines the message to-be-transferred as $\Pi_{COT}^{\mathsf{R}}(b)$, that is, the first message sent by a receiver in Π_{COT} who wants to receive msg_b. From that C_{AT} returns a transcript of the Π_{AT} instance, which $\mathcal{A}$ inserts into the transcript provided to the distinguisher $\mathcal{D}$ instead of honestly simulating the first Π_{AT} instance where the sender S of the AT is the receiver, the remainder is simulated as-is.

P_0 sender	P_1 sender
Π_{AT} with receiver P_0:	Π_{AT} with receiver P_0:
$(msg_{COT^R}, \perp)$	$(msg_{COT^R}, \perp)$
Π_{AT} with receiver P_1:	Π_{AT} with receiver P_1:
$(msg_{COT^R}, \perp)$	$(msg_{COT^R}, \perp)$
Π_{AT} with receiver P_R:	Π_{AT} with receiver P_R:
$(\perp, \perp)$	$(\perp, \perp)$
Π_{AT} with receiver P_0:	Π_{AT} with receiver P_0:
$(\perp, \perp)$	$(msg_{COT^S}, \perp)$
Π_{AT} with receiver P_1:	Π_{AT} with receiver P_1:
$(msg_{COT^S}, \perp)$	$(\perp, \perp)$
Π_{AT} with receiver P_R:	Π_{AT} with receiver P_R:
$(msg_{COT^S}, \perp)$	$(msg_{COT^S}, \perp)$

Figure 28.3.: The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the receiver.

Translating the result. Eventually $\mathcal{D}$ returns a guess which is either $\text{GAME}_1(\kappa)$ or $\text{GAME}_2(\kappa)$. In case $\mathcal{D}$ assumes to be in $\text{GAME}_1(\kappa)$ we assume the party we refer to as P_R is the sender, and if $\mathcal{D}$ assumes to be in $\text{GAME}_2(\kappa)$ we let $\mathcal{A}$ refer to P_D as the sender in the AT.

As can be seen the distributions match perfectly, as the only change is the sender and that one perfectly translates to our problem of de-anonymization.

Thus it follows that $\mathcal{D}$ is correct with probability

$$\begin{aligned} &\Pr[\mathcal{D} \text{ correct}] \leq 1/2 + \alpha \\ \Longrightarrow\ &|\Pr[\mathcal{D} \text{ correct}] - 1/2| = |\alpha| = \alpha \leq (1-\delta)/2 \end{aligned} \tag{28.8}$$

and thus if the adversary corrupts the sender then the protocol provides δ-Anonymity where δ is inherited from Π_{AT}. □

28.3.2. Corrupted Receiver

In this section we analyze the anonymity of the remaining parties provided that the adversary corrupts the receiver of the message. In this case the party P_R is fixed and the adversary has to distinguish between the two cases depicted in Fig. 28.3: Either P_0 is the sender and P_1 is the dummy friend or vice versa. The guessing algorithm has to distinguish the following two distributions:

$$\begin{aligned}&\{(\Sigma_0,\Sigma_1)\overset{\$}{\leftarrow}\{0,1\}^{2n}, b\overset{\$}{\leftarrow}\{0,1\}: \tau\overset{\$}{\leftarrow}\langle \mathsf{P}_\mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle(b,(\Sigma_0,\Sigma_1),\bot)\}\\ \approx&\{(\Sigma_0,\Sigma_1)\overset{\$}{\leftarrow}\{0,1\}^{2n}, b\overset{\$}{\leftarrow}\{0,1\}: \tau\overset{\$}{\leftarrow}\langle \mathsf{P}_\mathsf{R},\mathsf{P}_0,\mathsf{P}_1\rangle(b,\bot,(\Sigma_0,\Sigma_1))\}\end{aligned} \tag{28.9}$$

To prove the hardness of distinguishing these two cases we proceed as follows:

The formal description is as follows:

GAME$_1(\kappa)$: This is the original game where the sender is P_S and the receiver is P_R.

GAME$_2(\kappa)$: This game is as GAME$_1(\kappa)$ but in providing the transcript the simulator follows GAME$_1(\kappa)$ for the first round but inserts $msg_\mathsf{S}^{(1)}$ in the name of the *dummy friend* and $msg_\mathsf{D}^{(1)}$ in the name of the *sender* into $\Pi_{AT}(\mathsf{R})$. Note that this corresponds to the final distribution from Fig. 28.3.

Lemma 28.3.2. *Let* PKE *be an IND\$-CCA secure public-key encryption scheme. Let* SKE *be an IND\$-CCA secure secret-key encryption scheme. Let* SIG *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms A_{guess}, the distinguishing advantage for GAME$_1(\kappa)$ and GAME$_2(\kappa)$ is bounded by:*

$$|\Pr[\mathtt{out}_{GAME_1(\kappa)}=1]-\Pr[\mathtt{out}_{GAME_2(\kappa)}=1]|\in \mathsf{negl}(\kappa)$$

Proof. We reduce to the δ-anonymity of Π_{AT}. If there was some distinguisher $\mathcal{D}$ who can distinguish GAME$_1(\kappa)$ from GAME$_2(\kappa)$ with advantage $1/2+\alpha$ then there is some adversary $\mathcal{A}$ (which implies a guessing algorithm A_{guess}) who can determine the sender with advantage α. The limit δ then provides an upper bound on α.

Creating the transcript. We denote by C_{AT} the challenger for the anonymity-game of Π_{AT}. $\mathcal{A}$ determines the message to-be-transferred as $\Pi_{COT}^{\mathsf{S}}(msg_0, msg_1, \Sigma)$, that is, the response sent by a sender in Π_{COT} after having received the first message by the receiver. Note that Σ is the result of the first Π_{AT} instance where R inserts the message, and we take the output of that which only corresponds to the desired message $\Pi_{COT}^{\mathsf{R}}(b)$ with probability $1/2+\varepsilon/2$. However, we stress that this is not important to our proof, as the relevant part is only the insertion of that message as transferred message into the instance of Π_{AT} where P_R is the receiver and which is played with the challenger C_{AT}.

C_{AT} returns a transcript of the Π_{AT} instance, which $\mathcal{A}$ inserts into the transcript provided to the distinguisher $\mathcal{D}$ instead of honestly simulating the second Π_{AT} instance where the receiver R of the AT is also the receiver of the OT, the remainder is simulated as-is.

P_0 sender	P_1 sender
Π_{AT} with receiver $\underline{\mathsf{P}_\mathsf{D}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathsf{D}:}$
$(msg_{COT^\mathsf{R}}, \perp)$	$(msg_{COT^\mathsf{R}}, \perp)$
Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{0}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{0}:}$
$(msg_{COT^\mathsf{R}}, \perp)$	$(\perp, \perp)$
Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{1}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{1}:}$
$(\perp, \perp)$	$(msg_{COT^\mathsf{R}}, \perp)$
Π_{AT} with receiver $\underline{\mathsf{P}_\mathsf{D}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathsf{D}:}$
$(msg_{COT^\mathsf{S}}, \perp)$	$(msg_{COT^\mathsf{S}}, \perp)$
Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{0}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{0}:}$
$(\perp, \perp)$	$(msg_{COT^\mathsf{S}}, \perp)$
Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{1}:}$	Π_{AT} with receiver $\underline{\mathsf{P}_\mathbf{1}:}$
$(msg_{COT^\mathsf{S}}, \perp)$	$(\perp, \perp)$

Figure 28.4.: The two distributions we have to prove indistinguishable in case the adversary A_{guess_0} picks the dummy friend.

Translating the result. Eventually $\mathcal{D}$ returns a guess which is either $\textsc{Game}_1(\kappa)$ or $\textsc{Game}_2(\kappa)$. In case $\mathcal{D}$ assumes to be in $\textsc{Game}_1(\kappa)$ we assume the party we refer to as P_S is the sender, and if $\mathcal{D}$ assumes to be in $\textsc{Game}_2(\kappa)$ we let $\mathcal{A}$ refer to P_D as the sender in the AT.

As can be seen the distributions match perfectly, as the only change is the sender and that one perfectly translates to our problem of de-anonymization.

Thus it follows that $\mathcal{D}$ is correct with probability

$$\begin{aligned} &\Pr[\mathcal{D} \text{ correct}] \leq 1/2 + \alpha \\ \Longrightarrow\ &|\Pr[\mathcal{D} \text{ correct}] - 1/2| = |\alpha| = \alpha \leq (1-\delta)/2 \end{aligned} \tag{28.10}$$

and thus if the adversary corrupts the sender then the protocol provides δ-Anonymity where δ is inherited from Π_{AT}. □

28.3.3. Corrupted Dummy Friend

If the adversary corrupts the dummy friend after the execution of the Undetectable Oblivious Transfer protocol then it is guaranteed that the two remaining parties are the sender and the receiver.

$$\begin{aligned} &\{(\Sigma_0, \Sigma_1) \overset{\$}{\leftarrow} \{0,1\}^{2n}, b \overset{\$}{\leftarrow} \{0,1\} : \tau \overset{\$}{\leftarrow} \langle \mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_\mathsf{D} \rangle ((\Sigma_0, \Sigma_1), b, \perp)\} \\ \approx &\{(\Sigma_0, \Sigma_1) \overset{\$}{\leftarrow} \{0,1\}^{2n}, b \overset{\$}{\leftarrow} \{0,1\} : \tau \overset{\$}{\leftarrow} \langle \mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_\mathsf{D} \rangle (b, (\Sigma_0, \Sigma_1), \perp)\} \end{aligned} \tag{28.11}$$

However, distinguishing which is which is hard; to prove this claim we use the following games:

GAME$_1(\kappa)$: This game is the original game where the sender is $\mathsf{P_S}$ and the receiver is $\mathsf{P_R}$.

GAME$_2(\kappa)$: This game is as GAME$_1(\kappa)$ but in providing the transcript the simulator follows GAME$_1(\kappa)$ for the first round but inserts $msg_{\mathsf{R}}^{(0)}$ in the name of the *sender* and $msg_{\mathsf{S}}^{(0)}$ in the name of the *receiver* into $\Pi_{AT}(\mathsf{D})$.

Lemma 28.3.3. *Let* PKE *be an IND\$-CCA secure public-key encryption scheme. Let* SKE *be an IND\$-CCA secure secret-key encryption scheme. Let* SIG *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* GAME$_1(\kappa)$ *and* GAME$_2(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\mathrm{GAME}_1(\kappa)} = 1] - \Pr[\mathtt{out}_{\mathrm{GAME}_2(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce to the δ-anonymity of Π_{AT}. If there was some distinguisher $\mathcal{D}$ who can distinguish GAME$_1(\kappa)$ from GAME$_2(\kappa)$ with advantage $1/2 + \alpha$ then there is some adversary $\mathcal{A}$ (which implies a guessing algorithm A_{guess}) who can determine the sender with advantage α. The limit δ then provides an upper bound on α.

Creating the transcript. We denote by C_{AT} the challenger for the anonymity-game of Π_{AT}. $\mathcal{A}$ determines the message to-be-transferred as $\Pi_{COT}^{\mathsf{R}}(b)$, that is, the first message sent by a receiver in Π_{COT} who wants to receive msg_b. From that C_{AT} returns a transcript of the Π_{AT} instance, which $\mathcal{A}$ inserts into the transcript provided to the distinguisher $\mathcal{D}$ instead of honestly simulating the first Π_{AT} instance where the sender S of the AT is the dummy friend, the remainder is simulated as-is.

Translating the result. Eventually $\mathcal{D}$ returns a guess which is either GAME$_1(\kappa)$ or GAME$_2(\kappa)$. In case $\mathcal{D}$ assumes to be in GAME$_1(\kappa)$ we assume the party we refer to as $\mathsf{P_R}$ is the sender, and if $\mathcal{D}$ assumes to be in GAME$_2(\kappa)$ we let $\mathcal{A}$ refer to $\mathsf{P_S}$ as the sender in the AT.

As can be seen the distributions match perfectly, as the only change is the sender and that one perfectly translates to our problem of de-anonymization.

Thus it follows that $\mathcal{D}$ is correct with probability

$$\begin{aligned} &\Pr[\mathcal{D} \text{ correct}] \leq 1/2 + \alpha \\ \implies &|\Pr[\mathcal{D} \text{ correct}] - 1/2| = |\alpha| = \alpha \leq (1-\delta)/2 \end{aligned} \tag{28.12}$$

and thus if the adversary corrupts the sender then the protocol provides δ-Anonymity where δ is inherited from Π_{AT}. □

GAME$_3(\kappa)$: This game is as GAME$_2(\kappa)$ but in providing the transcript the simulator follows GAME$_2(\kappa)$ but inserts $msg_{\mathsf{S}}^{(1)}$ in the name of the *receiver* and $msg_{\mathsf{R}}^{(1)}$ in the name of the *sender* into $\Pi_{AT}(\mathsf{D})$.

Lemma 28.3.4. *Let* $\textsc{Pke}$ *be an IND\$-CCA secure public-key encryption scheme. Let* $\textsc{Ske}$ *be an IND\$-CCA secure secret-key encryption scheme. Let* $\textsc{Sig}$ *be an sEUF-CMA secure signature scheme. For all PPT guessing algorithms* A_{guess}*, the distinguishing advantage for* $\textsc{Game}_2(\kappa)$ *and* $\textsc{Game}_3(\kappa)$ *is bounded by:*

$$|\Pr[\mathtt{out}_{\textsc{Game}_2(\kappa)} = 1] - \Pr[\mathtt{out}_{\textsc{Game}_3(\kappa)} = 1]| \in \mathsf{negl}(\kappa)$$

Proof. We reduce to the δ-anonymity of Π_{AT}. If there was some distinguisher $\mathcal{D}$ who can distinguish $\textsc{Game}_2(\kappa)$ from $\textsc{Game}_3(\kappa)$ with advantage $1/2 + \alpha$ then there is some adversary $\mathcal{A}$ (which implies a guessing algorithm A_{guess}) who can determine the sender with advantage α. The limit δ then provides an upper bound on α.

Creating the transcript. We denote by C_{AT} the challenger for the anonymity-game of Π_{AT}. $\mathcal{A}$ determines the message to-be-transferred as $\Pi^{\mathsf{S}}_{COT}(msg_0, msg_1, \Sigma)$, that is, the response sent by a sender in Π_{COT} after having received the first message by the receiver. Note that Σ is the result of the first Π_{AT} instance where S inserts the message and sends it to R since $\textsc{Game}_2(\kappa)$ (where the roles for the first round has been changed), and we take the output of that which only corresponds to the desired message $\Pi^{\mathsf{R}}_{COT}(b)$ with probability $1/2 + \varepsilon/2$. However, we stress that this is not important to our proof, as the relevant part is only the insertion of that message as transferred message into the instance of Π_{AT} where P_{R} is the receiver and which is played with the challenger C_{AT}.

C_{AT} returns a transcript of the Π_{AT} instance, which $\mathcal{A}$ inserts into the transcript provided to the distinguisher $\mathcal{D}$ instead of honestly simulating the second Π_{AT} instance where the receiver R of the AT is also the receiver of the OT, the remainder is simulated as-is.

Translating the result. Eventually $\mathcal{D}$ returns a guess which is either $\textsc{Game}_2(\kappa)$ or $\textsc{Game}_3(\kappa)$. In case $\mathcal{D}$ assumes to be in $\textsc{Game}_2(\kappa)$ we assume the party we refer to as P_{S} is the sender, and if $\mathcal{D}$ assumes to be in $\textsc{Game}_3(\kappa)$ we let $\mathcal{A}$ refer to P_{R} as the sender in the AT.

As can be seen the distributions match perfectly, as the only change is the sender and that one perfectly translates to our problem of de-anonymization.

Thus it follows that $\mathcal{D}$ is correct with probability

$$\begin{aligned} &\Pr[\mathcal{D}\text{ correct}] \leq 1/2 + \alpha \\ \implies &|\Pr[\mathcal{D}\text{ correct}] - 1/2| = |\alpha| = \alpha \leq (1-\delta)/2 \end{aligned} \tag{28.13}$$

and thus if the adversary corrupts the sender then the protocol provides δ-Anonymity where δ is inherited from Π_{AT}. □

Given that this is the least efficient strategy it thus follows that the anonymity is bounded by δ for any possible A_{guess_0} corrupting any party.

29. Towards Undetectable Multi-Party Computation

In this section, we make a first step to define Undetectable Multi-Party Computation by providing an informal definition of the security requirement in Section 29.1 and a candidate instantiation for Undetectable Two-Party Computation (U2PC) in Section 29.2.

29.1. Defining Undetectable Multi-Party Computation

In this section, we informally define the security requirements of Undetectable Multi-Party Computation as the increment to the respective definition of Undetectable Oblivious Transfer from Section 27.1.

Again, we have K individuals, but now N of them are trying to compute a function f on N secret inputs.

ε-Correctness: This definition puts a lower bound on the probability that all players obtain the correct result of f evaluated on the N inputs.

δ-Anonymity: This definition puts an upper bound on the advantage of any of the K players to determine which of the $\binom{K-1}{N-1}$ subsets of remaining individuals has participated in the execution.

Privacy: This definition limits the amount of information that can be extracted on the other participants inputs beyond what can be inferred from the function output.

29.2. Towards constructing Undetectable Two-Party Computation from Undetectable Oblivious Transfer

We provide a candidate construction for the simpler case of $N = 2$ which we call Undetectable Two-Party Computation. That is, two parties out of K try to compute a bivariate function f on their respective secret inputs. To that end we use a covert two-party computation protocol like that from von Ahn, Hopper, and Langford [133] that uses COT for communication and replace all invocations of COT between the sender and the receiver by invocations of Undetectable Oblivious Transfer between all participants. That way, the message is transferred from the sender to the receiver without leaking information regarding the identity of the respective parties.

Assuming an $(\varepsilon, \delta, c, n)$-Undetectable Oblivious Transfer with $\varepsilon \in \mathrm{owhl}(\kappa)$, the protocol Π_{UMPC} for computing a bivariate function f using Undetectable Two-Party Computation we obtain by replacing all COTs by UOTs in [133, Protocol 3] is a $(\varepsilon, \delta, c \cdot |f|)$-Undetectable Two-Party Computation.

The protocol by von Ahn, Hopper, and Langford [133] essentially uses garbled circuits [139] and distributes the keys using COT instead of using classical OT. The protocol we propose replaces the COTs with UOTs where from the K parties only two are actually participating, one as sender and one as receiver.

Assuming that each UOT is correct with overwhelming probability the correctness of the entire scheme then follows from the correctness of garbled circuits.

Anonymity remains the same as in order to determine any of the participating parties, the only way is to de-anonymize any of the UOTs, the success probability of which is bounded by δ.

Privacy is inherited by the Garbled Circuits; since the Undetectable Oblivious Transfers are private as well our changes induce no additional leakage.

Finally, note that for a garbled circuit we require $|\mathsf{f}|$ many OTs and each requires c rounds, resulting in the new number of rounds.

30. Conclusion

While Secure Multi-Party Computation has been known as a tool to hide the inputs since the 80s [139, 78], it was only in 2005 when the first methods were propose that would also hide the *computation itself*. This was first proposed for the two-party setting [133] and later extended to the K-party case [48]. Except for a *fitness* function that has to be fulfilled and which decides whether the output is *favorable* to all parties or not, participants only learn the output if *all* parties followed the protocol. While this implicitly makes sense for a two-party protocol—if only one party follows the protocol then there is nothing to be learned from the interaction—it comes as a restriction if we consider $K > 2$, as a single non-participant can ruin the entire computation.

We investigated whether a stronger expansion to K is actually possible, where out of the K parties present we know a priori that N parties are performing the computation; we want the participants to learn the output while hiding their computation in the existing communication, such that no participant knows the identity of the other individuals who took part in the computation.

We defined what we mean by this in Chapter 29 and showed there that it can be instantiated using a new tool called Undetectable Oblivious Transfer, which lets K parties perform an OT where one party is the sender, one party is the receiver, and the remaining parties are uninvolved bystanders. The protocol transfers the correct message chosen by the receiver, but neither reveals the real identities of the parties, nor their respective inputs.

We instantiated an Undetectable Oblivious Transfer protocol based on Covert Oblivious Transfer and Anonymous Transfer in Chapter 28. While Covert Oblivious Transfer already existed before and has been instantiated by von Ahn, Hopper, and Langford [133], Anonymous Transfer is a new primitive that we defined in Chapter 23. It lets one of $K - 1$ potential senders send a bit (resp. a message) to the receiver, without revealing who the actual sender was.

We investigated feasibility of Anonymous Transfer. We first provided a negative result in Chapter 24 where we showed that in the asymptotic setting, Anonymous Transfer with overwhelming correctness *and* anonymity is not possible, meaning that AT is in *Impossibilitopia* in the extended worlds from Impagliazzo [89]. Then, in Chapters 25 and 26 we showed that in a weaker, fine-grained setting, AT with constant anonymity is indeed possible when assuming idealized obfuscation. If the anonymity of this could be raised to be overwhelming in the security parameter then AT would separate the extended worlds of Impagliazzo [89] at the highest level.

Acronyms

AT Anonymous Transfer.

BK Bookkeeping.

BKA Bookkeeping and Analytics.

cf. confer.

CH Controlled Hadamard.

CMPC Covert Multi-Party Computation.

COT Covert Oblivious Transfer.

CRS Common Reference String.

e.g. exempli gratia.

EUF-CMA Existential Unforgeability under Chosen Message Attacks.

FHE Fully Homomorphic Encryption.

FP Function Parameter.

H Hadamard.

i.e. id est.

IND-CCA Indistinguishability under Chosen Ciphertext Attacks.

IND-CPA Indistinguishability under Chosen Plaintext Attacks.

IND$-CCA Indistinguishability from Random under Chosen Ciphertext Attacks.

IND$-CPA Indistinguishability from Random under Chosen Message Attacks.

INDD Indistinguishability from Random.

ITM Interactive Turing Machine.

LR Left-Right.

MPC Secure Multi-Party Computation.

NIZK Non-Interactive Zero Knowledge.

NIZKPoK Non-Interactive Zero-Knowledge Proof of Knowledge.

NP Nondeterministic Polyomial time.

OA Outsourced Analytics.

OMA Obfuscated Measurement Attack.

ORR Onion-Routing with Replies.

OS Outsource.

OT Oblivious Transfer.

OTP One Time Pad.

PAF Privacy Amplification Function.

pid Party Identifier.

PKE Public Key Encryption.

PKI Public-Key Infrastructure.

POV Positive Operator-Valued.

PPT Probabilistic Polyonmial time.

PRF Pseudorandom Function.

q-IND-CPA Quantum Indistinguishability under Chosen Plaintext Attacks.

QCom Quantum Commitment.

QD Quantum Decay.

QFHE Quantum Fully Homomorphic Encryption.

QFHET Quantum Fully Homomorphic Encryption Tuple.

QOT Quantum Oblivious Transfer.

QOTP Quantum One Time Pad.

QPT Quantum Polynomial time.

QRO Quantum Random Oracle.

QROM Quantum Random Oracle Model.

Reg Registration.

ROM	Random Oracle Model.
RSS	Robust Secret Sharing.
s.t.	such that.
sEUF-CMA	Strong Existential Unforgeability under Chosen Message Attacks.
SFE	Secure Function Evaluation.
SFP	Sign Function Parameter.
sid	Session Identifier.
SKE	Symmetric Key Encryption.
SMT	Secure Message Transfer.
SRAT	Silent Receiver Anonymous Transfer.
ssid	Subsession Identifier.
T	Toffoli.
TSA	Trusted Signing Authority.
TTP	Trusted Third Party.
U2PC	Undetectable Two-Party Computation.
UC	Universal Composability.
UH	User History.
UI	Update Information.
UMPC	Undetectable Multi-Party Computation.
UOT	Undetectable Oblivious Transfer.
Upd	Update.
UReg	User Registration.
w.l.o.g.	without loss of generality.
ZK	Zero Knowledge.

Symbols

$\mathcal{A}$**:** The adversary, that is, a malicious party in any security framework.

A**:** A generic algorithm, that is, a set of instructions executed by a single party.

$\vec{a}$**:** Part of the result of a (direct two-party) computation between a user and the operator. It represents a vector of $\mathbb{Z}_o$ elements that should be added to the User History..

α**:** The advantage of an adversary, that is, a value quantifying how much better than randomly guessing the adversary really is.

$\vec{\alpha}$**:** Part of the result of a (direct two-party) computation between a user and the operator. It contains a permutation that should be applied to the User History.

$\mathcal{B}$**:** The adversary, that is, a malicious party in any security framework. In Part II we use this to denote the QPT adversary.

$\vec{b}$**:** The basis vector containing vectors from $\{+, \times\}$.

b**:** A bit.

B**:** The bases $(1/2 \cdot |0\rangle + 1/2 \cdot |+\rangle, 1/2 \cdot |1\rangle + 1/2 \cdot |-\rangle)$.

$\times$**:** The diagonal basis states $(|+\rangle, |-\rangle)$.

$+$**:** The rectangular basis states $(|0\rangle, |1\rangle)$.

b_C**:** The transferred bit in the challenge transcript.

$\texttt{Ber}$**:** A bernoulli-distribution which returns 1 with probability p.

β**:** The bit that is sampled by a challenger C in a security game which has to be guessed by the adversary.

C**:** The challenger in a game-based reduction.

C**:** The challenge a challenger C provides to the adversary in a game-based proof.

$\mathscr{C}$**:** A linear code.

C**:** The committer.

c**:** The number of rounds in an interactive protocol.

$\mathscr{c}$**:** A codeword from a code $\mathscr{C}$.

CAdd: The method of a (homomorphic) commitment scheme to homomorphically add two commitments.

cb: A (classical) control bit that is used to trigger a quantum gate in Part II of this thesis.

cct: The control bit ciphertext, that is, an encryption of a classical control bit that is used to trigger a quantum gate in Part II of this thesis.

χ: An intermediate round in an interactive protocol.

cod: The codomain, that is, the (finite) output space of a method or oracle.

$\mathrm{CointossS}$: A function realizing a (biased) coin toss. We write $\mathrm{CointossS}^{(r)}_{(p)}(x, y)$ to denote the coin toss that returns x with probability p and y with probability $(1 - p)$, where the randomness is extracted from the input r.

$\textsc{Com}$: A commitment scheme.

Com: The method of a commitment scheme to commit to a message.

$\mathtt{com}$: The message that commits a party to a bit (or a string of bits).

crs: The common reference string, that is, a global string accessible to all parties that was sampled from a known distribution.

CS: The ciphertext space.

ct: A cipher text, that is, an encrypted message.

$\mathscr{D}$: The codeword distance, that is, the minimum distance of two codewords $(c, c') \in \mathscr{C}$.

$\mathcal{D}$: The distinguisher, i.e. a QPT algorithm which distinguishes two distributions.

D: A distribution.

D: The dummy friend who is not participating in any protocol but is still present for some reason.

Dec: A method of an encryption scheme to decrypt a given ciphertext with a given key to obtain a message.

δ: The measurement of anonymity for an AT or UOT protocol.

$\mathrm{d_H}$: The Hellinger Distance, defined over two probability distributions p and q as $\mathrm{d_H}(p, q) := \frac{1}{\sqrt{2}}\sqrt{\sum_i\left(\sqrt{p_i} - \sqrt{q_i}\right)^2}$.

$\mathrm{d_{Ham}}$: The hamming distance between two bit strings.

$\mathrm{d_{TV}}$: The Total Variational Distance, defined over two probability dis tributions p and q as $\mathrm{d_{TV}}(p, q) := \frac{1}{2}\sum_i|p_i - q_i|$.

dom: The (preimage) domain of a method or oracle, that is, the (finite) input space.

e: A pairing equation as part of a pairing group.

ek: The evaluation key of a homomorphic encryption scheme, can be used to deliberately manipulate the clear text given only its ciphertext.

Enc: A method of an encryption scheme to encrypt a given message with a given key to obtain a ciphertext.

encode: The encode function which encodes a given classical string into a quantum state using Wiesners encoding in Part II of this thesis.

ε: The measurement of correctness for an AT or UOT protocol.

ϵ: The error rate, that is, the difference between the received codeword $\mathcal{c}$ and the measured codeword $\mathcal{c}'$ in percent.

ES: The evaluation key space for a homomorphic encryption scheme.

Eval: Evaluation method of homomorphic encryption schemes. Enables manipulation of a ciphertext such that the messages before and after the evaluation are in a (known) relation to each other but the messages themselves remain hidden throughout the process.

Evaluate: The method of an obfuscator $\mathbb{O}$ to evaluate a given program given only the handle.

Exp: An experiment as part of a game.

ExtractWit: The method of a Zero Knowledge scheme that a simulator uses to (partially) extract the witness from a zero knowledge proof given that SetupExt was used instead of Setup.

F: The fidelity of two density matrices ρ_1 and ρ_2, that is, $\mathrm{tr}\left((\sqrt{\rho_1}\rho_2\sqrt{\rho_1})^{\frac{1}{2}}\right)$.

$\mathbb{F}$: A finite field.

$\mathcal{F}$: An ideal functionality in a simulation-based framework.

f: The function that is to be evaluated in Secure Multi-Party Computation.

FC: A function class, that is, the set of functions that belong in a certain complexity class.

Fhe: A fully homomorphic encryption scheme.

fp: The function parameters, that is, the set of parameters that define the computation.

$\vec{\mathrm{G}}$: A generator matrix spanning a linear code.

$\mathbb{G}$: A (finite cyclic) group.

g: The generator of a group $\mathbb{G}$.

Game: The game in a game-hop based proof.

γ: The parameter defining the probability in a *Bernoulli* distribution to return 1.

Gen: The generate method to create the groups for a zero knowledge scheme.

H: The entropy of a random variable.

H: Quantum-gate for Hadamard-Transformation.

H: A helper that coordinates computations involving the data of several users and is capable of more hardware-consuming computations in Part I of this thesis.

H: The hilbert space.

$\mathbf{H}$: A hybrid game in a game-based proof system. Induces a slight change compared to its neighbors that can not be distinguished.

h: The handle an obfuscator returns that enables evaluating the program.

$\mathscr{K}$: The dimenson of the vector space spanned by codewords $\mathscr{c} \in \mathscr{C}$.

K: The number of individuals present during an MPC execution. Note that these parties do not necessarily participate or know that a computation is in progress.

k: The number of inputs on which the value table of the random oracle provides different outputs than the actual oracle would have provided.

k: The signing key of a signature scheme, can be used to sign a given message such that third parties can verify the authenticity.

κ: The security parameter.

$|\psi\rangle$: A quantum state.

$|\phi\rangle$: A quantum state that was encrypted using a Quantum One Time Pad.

KeyGen: A generic key generation method used for encryption and signature schemes.

ℓ: Running variable for the function parameters.

Λ: The language of a zero knowledge proof.

λ: The logbook, that is, the amount of information the user stores regarding its current state in Part I of this thesis.

lin: The linking number which is stored in a logbook in Part I of this thesis. If it is 0 then no outsourcing-triplet has been started by the owner of the logbook. Otherwise, it is some $\mathbb{Z}_o$-element which is shared with helper and operator and used for connecting subsequent executions of Outsource, Outsourced Analytics and Update that belong to the same user.

M: A mix-server for mix-nets such as TOR in Part I of this thesis.

msg**:** A generic (plain text) message.

m**:** The length of the protocol messages.

measure**:** The measure function which measures a quantum state using provided bases $\vec{b}$ in Part II of this thesis.

MS**:** The message space.

msg^C**:** A message reported in the challenge transcript.

$\mathcal{N}$**:** The length of a codeword $c \in \mathcal{C}$.

N**:** The number of parties (actively) participating in a Secure Multi-Party Computation.

n**:** Bit-length of input messages.

negl**:** The set of functions that are negligible with respect to a given argument (usually the security parameter κ), that is, the set of functions that asymptotically decrease faster than any polynomial in the given argument.

ν**:** A nonce (number used once).

$\mathbb{O}$**:** An ideal obfuscator that obfuscates a given program.

O**:** The operator of the system in Part I.

O**:** An oracle. That is, a machine than can be queried and that returns the output in $O(1)$.

o**:** The order of a group.

Obfuscate**:** The method of an obfuscator $\mathbb{O}$ to obfuscate a given program.

OT**:** Transfer protocol for an OT and UOT scheme.

OT: An Oblivious Transfer scheme.

otp**:** A one time pad, that is, binary string that information-theoretically hides a message.

otp_C**:** The used One-Time-Pad in the challenge transcript.

$\mathtt{out}$**:** Output of a party playing a cryptographic game.

owhl**:** The set of functions that are overwhelming with respect to a given argument (usually the security parameter κ), that is, the set of functions that asymptotically increase faster than any polynomial.

P**:** A party that is present during a protocol execution.

P**:** An obfuscated program that takes as input a transcript τ outputs a message Σ for the receiver.

$\mathtt{p}$**:** A probability distribution, *i.e.* a vector of probabilities where $\mathtt{p}[i] = \Pr[\mathtt{out}_\mathtt{p} = i]$. We sometimes write $\mathtt{p}_i$ instead of $\mathtt{p}[i]$ when convenient.

p**:** A given probability.

φ**:** A quantum channel.

ϕ**:** A value between 1 and K describing which player P_ϕ participates in the protocol.

ϕ_C**:** The challenge player in a game-based proof.

Π**:** A protocol, that is, a set of instructions on how parties in a system interact and which messages they send to each other.

ϖ**:** A random permutation on a given set.

π**:** The proof message of a (non-interactive) zero knowledge proof.

pid**:** The party identifier that is unique to each party.

pk**:** The public key of an encryption scheme, can be used to encrypt a message into a cipher text.

PKE: A public key / asymmetric encryption scheme.

poly**:** The set of polynomials with respect to a given argument, that is, poly(κ) denotes all functions which can be described as $a_c\kappa^c + a_{c-1}\kappa^{c-1} + \cdots + a_1\kappa + a_0$ for any constant c.

pp**:** The parameter for defining a (pairing) group.

PRF**:** A (quantum-resistant) Pseudorandom Function.

Proof**:** The method of a Zero Knowledge scheme a prover uses in order to create the proof of a statement.

Ψ**:** The number of hops, *i.e.*, intermediate mixservers, in the path for Onion-Routing with Replies.

q**:** An other probability distribution, see the entry for p.

QD**:** The quantum distance, that is, the distance of two initially equivalent quantum states after t oracle queries, each following a unitary transformation.

QFHE: A fully homomorphic encryption scheme over quantum inputs.

R**:** The receiving party.

r**:** A random value.

Rand**:** The rerandomization of the last round of an AT.

R_C**:** A compressed oracle due to Zhandry [142].

Reconstruct**:** Reconstruction algorithm for an AT.

Rel A statement-witness-relation of an NP-language..

R_H**:** Random Oracle.

ρ**:** A density matrix.

R_Q**:** Quantum Random Oracle, that is, a Random Oracle that can be queried in superposition.

R_S**:** The simulated (random) oracle a reduction algorithm provides for the distinguisher.

rs**:** The robust share of a value. A robust share for one party contains the additive share itself, a *commitment* on the other parties share and *unveil* information on the commitment the other party has on the own share.

S**:** The sending party.

$\mathcal{S}$**:** The simulator in a simulation-based framework. The task of the simulator is to interact with an ideal functionality and create a transcript of an interaction for honest parties in such a way, that this transcript can not be distinguished from one that honest parties create when interacting according to a protocol – the interaction of honest parties is *simulated* based on the interaction with the ideal functionality only.

$\mathbb{S}$**:** A symmetric group, that is, the group of elements alongside all bijections/permutations onto itself.

$\vec{s}$**:** Part of the result of a (direct two-party) computation between a user and the operator. It represents a vector of $\mathbb{Z}_o$ elements (or $\perp$) that the User History should be set to (except for the slots marked with $\perp$).

s**:** The slack, that is, the difference between two variables x and y: if $x < y$, we can write $x + s = y$, allowing for an easier analysis..

ser**:** The serial number to specify a revision of a logbook in Part I.

Setup**:** The setup method to create relevant parameters of a zero knowledge scheme.

SetupExt**:** The setup method to create relevant parameters of a zero knowledge scheme such that the witness can be (partially) extracted by a simulator.

SetupSim**:** The setup method of a Zero Knowledge scheme to create relevant parameters such that a simulator can forge a valid proof without knowing the witness.

sid**:** The session identifier.

SIG: A signature scheme.

Σ**:** The to-be-transferred message of a protocol.

ς**:** The measurement of secrecy for an AT, which is high if the non-receiver is unable to reconstruct the transferred bit better than by guessing.

σ**:** A signature that ensures authenticity of a message.

Σ_C**:** The transferred message in the challenge transcript.

Sign**:** A method of a signature scheme to create a signature of a given message for a given signing key.

SimZK**:** The method of a Zero Knowledge scheme a simulator uses to forge a zero-knowledge proof without the witness that will be accepted by the verifier if SetupSim was used instead of Setup.

sk**:** The secret key of an encryption scheme, can be used to decrypt a cipher text.

Ske: A symmetric encryption scheme.

*ssid***:** The subsession identifier.

*st***:** The state of a party containing its memory storage and random tape.

*stmt***:** The statement of a zero knowledge proof.

T**:** A third party in Part I that signs valid operator function parameters if they meet certain security requirements.

T**:** The (random) tape of a player.

t**:** The number of queries allowed on a given oracle.

τ**:** Transcript of a protocol.

τ_C**:** The challenge transcript of a game-based proof.

*td***:** A trapdoor with respect to a given CRS, that is, additional information from the creation of the string that can be used in a simulation.

θ**:** A qubit that is used as control qubit for a quantum state.

ϑ**:** The token (called *temp* by Kuhn et al. [103]) that links the message sent through the onion network with its reply.

Transfer**:** The transfer protocol for an AT.

U**:** The uniform distribution.

U**:** The user of the system in Part I who collects validated data.

UAdd**:** The method of a (homomorphic) commitment scheme to homomorphically add two unveil informations.

*UH***:** The User History, that is, the set of authenticated data.

*UI***:** The Update Information to update the User History.

Unv**:** The method of a commitment scheme to unveil a commitment to a message.

unv**:** The message that opens a given commitment message to its bit (or string of bits).

Verify: The method of a Zero Knowledge scheme a verifier uses to verify a given proof of a known statement.

Vfy: A method of a signature scheme to verify the integrity of a message given a signature and a verification key.

view: The view of a given party, that is, the set of inputs, outputs and messages visible to that party in a protocol execution or security game.

vk: The verification key of a signature scheme, suffices to check auth enticity of a signature over a known message.

vk_C: The verification key used in the challenge transcript.

w: A generic n-bit string we refer to as a *word*.

wit: The witness of a zero knowledge proof.

X: Parameterized by a value $\phi \in [K-1]$, this is the value obtained in the program P_{FG-AT} after decrypting the message msg_ϕ with the supposed secret key sk_ϕ.

X: Quantum-gate for Negation.

x: The input that is given to a function f during a Secure Multi-Party Computation.

x: The binary (classic) key of a Quantum One Time Pad triggering the negation gate in Part II of this thesis.

poly: An arbitrary polynomial used for the impossibility result in Part III of this thesis.

ξ: The noise rate of the quantum channel.

y: The output obtained from a function f during a Secure Multi-Party Computation.

$\mathcal{Z}$: The environment in a simulation-based model; models everything that happens outside of the current execution.

Z: Quantum-gate for Phase Flip.

Z: The number of inputs (*i.e.* UHs from different users) a given function f requires for the computation in Part I of this thesis.

z: The binary (classic) key of a Quantum One Time Pad triggering the phase shift gate in Part II of this thesis.

ζ: The current index of inputs (*i.e.* UHs from different users) a given function f requires for the computation in Part I of this thesis.

Zk: A zero knowlege scheme.

Bibliography

[1] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. “Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups”. In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2011, pp. 649–666. DOI: 10.1007/978-3-642-22792-9_37.

[2] Masayuki Abe, Markulf Kohlweiss, Miyako Ohkubo, and Mehdi Tibouchi. “Fully Structure-Preserving Signatures and Shrinking Commitments”. In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 35–65. DOI: 10.1007/978-3-662-46803-6_2.

[3] Ittai Abraham, Benny Pinkas, and Avishay Yanai. “Blinder - Scalable, Robust Anonymous Committed Broadcast”. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. Virtual Event, USA: ACM Press, Nov. 2020, pp. 1233–1252. DOI: 10.1145/3372297.3417261.

[4] Mark Adcock and Richard Cleve. “A Quantum Goldreich-Levin Theorem with Cryptographic Applications”. In: *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*. Ed. by Helmut Alt and Afonso Ferreira. Vol. 2285. Lecture Notes in Computer Science. Springer, 2002, pp. 323–334. DOI: 10.1007/3-540-45841-7_26.

[5] Thomas Agrikola, Geoffroy Couteau, and Sven Maier. “Anonymous Whistleblowing over Authenticated Channels”. In: *TCC 2022*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Lecture Notes in Computer Science. To appear. Springer, Heidelberg, Germany, Nov. 2022.

[6] Dorit Aharonov and Michael Ben-Or. “Fault-Tolerant Quantum Computation With Constant Error”. In: *29th Annual ACM Symposium on Theory of Computing*. El Paso, TX, USA: ACM Press, May 1997, pp. 176–188. DOI: 10.1145/258533.258579.

[7] Miklós Ajtai. “Generating Hard Instances of Lattice Problems (Extended Abstract)”. In: *28th Annual ACM Symposium on Theory of Computing*. Philadephia, PA, USA: ACM Press, May 1996, pp. 99–108. DOI: 10.1145/237814.237838.

[8] Gorjan Alagic, Yfke Dulek, Christian Schaffner, and Florian Speelman. “Quantum Fully Homomorphic Encryption with Verification”. In: *Advances in Cryptology – ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, Dec. 2017, pp. 438–467. DOI: 10.1007/978-3-319-70694-8_16.

[9] Romain Alléaume, Jan Bouda, Cyril Branciard, Thierry Debuisschert, Mehrdad Dianati, Nicolas Gisin, Mark Godfrey, Philippe Grangier, Thomas Länger, Anthony Leverrier, Norbert Lütkenhaus, Philippe Painchault, Momtchil Peev, Andreas Poppe, Thomas Pornin, John G. Rarity, Renato Renner, Gregoire Ribordy, Michel Riguidel, Louis Salvail, Andrew Shields, Harald Weinfurter, and Anton Zeilinger. *SECOQC White Paper on Quantum Key Distribution and Cryptography.* arXiv e-prints, Report quant-ph/0701168. https://arxiv.org/abs/quant-ph/0701168. 2007.

[10] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. "Quantum Attacks on Classical Proof Systems: The Hardness of Quantum Rewinding". In: *55th Annual Symposium on Foundations of Computer Science.* Philadelphia, PA, USA: IEEE Computer Society Press, Oct. 2014, pp. 474–483. DOI: 10.1109/FOCS.2014.57.

[11] Megumi Ando and Anna Lysyanskaya. "Cryptographic Shallots: A Formal Treatment of Repliable Onion Encryption". In: *TCC 2021: 19th Theory of Cryptography Conference, Part III.* Ed. by Kobbi Nissim and Brent Waters. Vol. 13044. Lecture Notes in Computer Science. Raleigh, NC, USA: Springer, Heidelberg, Germany, Nov. 2021, pp. 188–221. DOI: 10.1007/978-3-030-90456-2_7.

[12] Suzanna Andrews, Bryan Burrough, and Sarah Ellison. "The Snowden Saga: A shadowland of secrets and light". In: *Vanity Fair* 23 (2014).

[13] Diego F. Aranha, Conrado Porto Lopes Gouvêa, Tobias Markmann, Riad S. Wahby, and Kevin Liao. *RELIC is an Efficient LIbrary for Cryptography.* https://github.com/relic-toolkit/relic. 2020.

[14] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio Cesar López-Hernández. "Faster Explicit Formulas for Computing Pairings over Ordinary Curves". In: *Advances in Cryptology – EUROCRYPT 2011.* Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 48–68. DOI: 10.1007/978-3-642-20465-4_5.

[15] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Jimmy Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Michael Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew Harrigan, Michael Hartmann, Alan Ho, Markus Rudolf Hoffmann, Trent Huang, Travis Humble, Sergei Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, Dave Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod Ryan McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin Jeffery Sung, Matt Trevithick, Amit Vainsencher, Benjamin Villalonga, Ted White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John Martinis. "Quantum Supremacy using a Programmable Superconducting Processor". In: *Nature* 574 (7779 Oct. 2019), pp. 505–510. DOI: s41586-019-1666-5.

[16] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. “Privacy-Preserving Search of Similar Patients in Genomic Data”. In: *Proceedings on Privacy Enhancing Technologies* 2018.4 (2018), pp. 104–124. DOI: 10.1515/popets-2018-0034.

[17] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. “ObliviAd: Provably Secure and Practical Online Behavioral Advertising”. In: *2012 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2012, pp. 257–271. DOI: 10.1109/SP.2012.25.

[18] European Central Bank. *Average number of cash and card transactions per person per day in the Euro Area in 2016, by country*. Website. https://www.statista.com/statistics/893459/average-number-of-transactions-per-person-per-day-by-method/, last visited 2021-11-18. Nov. 2017.

[19] European Central Bank. *Fifth Report on card fraud, September 2018*. Website. https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport201809.en.html, last visited 2020-04-27. Sept. 2018.

[20] Paulo S. L. M. Barreto and Michael Naehrig. “Pairing-Friendly Elliptic Curves of Prime Order”. In: *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. Lecture Notes in Computer Science. Kingston, Ontario, Canada: Springer, Heidelberg, Germany, Aug. 2006, pp. 319–331. DOI: 10.1007/11693383_22.

[21] Donald Beaver, Silvio Micali, and Phillip Rogaway. “The Round Complexity of Secure Protocols (Extended Abstract)”. In: *22nd Annual ACM Symposium on Theory of Computing*. Baltimore, MD, USA: ACM Press, May 1990, pp. 503–513. DOI: 10.1145/100216.100287.

[22] Mihir Bellare and Phillip Rogaway. “Entity Authentication and Key Distribution”. In: *Advances in Cryptology – CRYPTO’93*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1994, pp. 232–249. DOI: 10.1007/3-540-48329-2_21.

[23] Michael Ben-Or, Ran Canetti, and Oded Goldreich. “Asynchronous secure computation”. In: *25th Annual ACM Symposium on Theory of Computing*. San Diego, CA, USA: ACM Press, May 1993, pp. 52–61. DOI: 10.1145/167088.167109.

[24] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. “Privacy Amplification by Public Discussion”. In: *SIAM Journal on Computing* 17.2 (Apr. 1988), pp. 210–229. DOI: 10.1137/0217014.

[25] Charles Berret. *Guide to SecureDrop*. 2016. DOI: 10.7916/D84178B2.

[26] Eli Biham, Yaron J. Goren, and Yuval Ishai. “Basing Weak Public-Key Cryptography on Strong One-Way Functions”. In: *TCC 2008: 5th Theory of Cryptography Conference*. Ed. by Ran Canetti. Vol. 4948. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Mar. 2008, pp. 55–72. DOI: 10.1007/978-3-540-78524-8_4.

[27] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. "Updatable Anonymous Credentials and Applications to Incentive Systems". In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. London, UK: ACM Press, Nov. 2019, pp. 1671–1685. DOI: 10.1145/3319535.3354223.

[28] Manuel Blum. "Coin Flipping by Telephone". In: *Advances in Cryptology – CRYPTO'81*. Ed. by Allen Gersho. Vol. ECE Report 82-04. Santa Barbara, CA, USA: U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981, pp. 11–15.

[29] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. "Secure Multiparty Computation Goes Live". In: *FC 2009: 13th International Conference on Financial Cryptography and Data Security*. Ed. by Roger Dingledine and Philippe Golle. Vol. 5628. Lecture Notes in Computer Science. Accra Beach, Barbados: Springer, Heidelberg, Germany, Feb. 2009, pp. 325–343.

[30] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Seoul, South Korea: Springer, Heidelberg, Germany, Dec. 2011, pp. 41–69. DOI: 10.1007/978-3-642-25385-0_3.

[31] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9.

[32] Zvika Brakerski. "Quantum FHE (Almost) As Secure As Classical". In: *Advances in Cryptology – CRYPTO 2018, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2018, pp. 67–95. DOI: 10.1007/978-3-319-96878-0_3.

[33] Zvika Brakerski and Vinod Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. Cryptology ePrint Archive, Report 2011/344. https://eprint.iacr.org/2011/344. 2011.

[34] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". In: *52nd Annual Symposium on Foundations of Computer Science*. Ed. by Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, Oct. 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12.

[35] Howard E. Brandt. "Quantum decoherence and qubit devices". In: *Noise and Information in Nanoelectronics, Sensors, and Standards*. Ed. by Laszlo B. Kish, Frederick Green, Giuseppe Iannaccone, and John R. Vig. SPIE, 2003, pp. 308–344. DOI: 10.1117/12.488482.

[36] Gilles Brassard, Claude Crépeau, Richard Jozsa, and Denis Langlois. "A Quantum Bit Commitment Scheme Provably Unbreakable by both Parties". In: *34th Annual Symposium on Foundations of Computer Science*. Palo Alto, CA, USA: IEEE Computer Society Press, Nov. 1993, pp. 362–371. DOI: 10.1109/SFCS.1993.366851.

[37] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Physical Review A* 71 (2 Feb. 2005), p. 022316. DOI: 10.1103/PhysRevA.71.022316.

[38] Anne Broadbent and Stacey Jeffery. "Quantum Homomorphic Encryption for Circuits of Low T-gate Complexity". In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2015, pp. 609–629. DOI: 10.1007/978-3-662-48000-7_30.

[39] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[40] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. https://eprint.iacr.org/2000/067. 2000.

[41] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd Annual Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[42] Ran Canetti. "Universally Composable Signature, Certification, and Authentication". In: *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 219. DOI: 10.1109/CSFW.2004.24.

[43] Ran Canetti and Marc Fischlin. "Universally Composable Commitments". In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 19–40. DOI: 10.1007/3-540-44647-8_2.

[44] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. "Obfuscation of Probabilistic Circuits and Applications". In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, Mar. 2015, pp. 468–497. DOI: 10.1007/978-3-662-46497-7_19.

[45] Rémi Canillas, Rania Talbi, Sara Bouchenak, Omar Hasan, Lionel Brunie, and Laurent Sarrat. "Exploratory Study of Privacy Preserving Fraud Detection". In: *19th International Middleware Conference 2018*. ACM, 2018. DOI: 10.1145/3284028.3284032. URL: https://doi.org/10.1145/3284028.3284032.

[46] Gizem S Cetin, Hao Chen, Kim Laine, Kristin Lauter, Peter Rindal, and Yuhou Xia. *Private Queries on Encrypted Genomic Data*. Cryptology ePrint Archive, Report 2017/207. https://eprint.iacr.org/2017/207. 2017.

[47] Siu-on Chan, Ilias Diakonikolas, Paul Valiant, and Gregory Valiant. "Optimal Algorithms for Testing Closeness of Discrete Distributions". In: *25th Annual ACM-SIAM Symposium on Discrete Algorithms*. Ed. by Chandra Chekuri. Portland, OR, USA: ACM-SIAM, Jan. 2014, pp. 1193–1203. DOI: 10.1137/1.9781611973402.88.

[48] Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. "Covert Multi-Party Computation". In: *48th Annual Symposium on Foundations of Computer Science*. Providence, RI, USA: IEEE Computer Society Press, Oct. 2007, pp. 238–248. DOI: 10.1109/FOCS.2007.21.

[49] David Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: *Journal of Cryptology* 1.1 (Jan. 1988), pp. 65–75. DOI: 10.1007/BF00206326.

[50] David Chaum. "Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms". In: *Secure Electronic Voting*. Ed. by Dimitris Gritzalis. Vol. 7. Advances in Information Security. Springer, Heidelberg, Germany, 2003, pp. 211–219. DOI: 10.1007/978-1-4615-0239-5_14. URL: https://doi.org/10.1007/978-1-4615-0239-5%5C_14.

[51] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. *Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning*. arXiv e-prints, Report 1712.05526. https://arxiv.org/abs/1712.05526. 2017.

[52] Benny Chor and Lior Moscovici. "Solvability in Asynchronous Environments (Extended Abstract)". In: *30th Annual Symposium on Foundations of Computer Science*. Research Triangle Park, NC, USA: IEEE Computer Society Press, Oct. 1989, pp. 422–427. DOI: 10.1109/SFCS.1989.63513.

[53] Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. "On the Compressed-Oracle Technique, and Post-Quantum Security of Proofs of Sequential Work". In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 2021, pp. 598–629. DOI: 10.1007/978-3-030-77886-6_21.

[54] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol". In: *Journal of Cryptology* 33.4 (Oct. 2020), pp. 1914–1983. DOI: 10.1007/s00145-020-09360-1.

[55] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: An Anonymous Messaging System Handling Millions of Users". In: *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 2015, pp. 321–338. DOI: 10.1109/SP.2015.27.

[56] Geoffroy Couteau. *Revisiting Covert Multiparty Computation*. Cryptology ePrint Archive, Report 2016/951. https://eprint.iacr.org/2016/951. 2016.

[57] Claude Crépeau. “Quantum Oblivious Transfer”. In: *Journal of Modern Optics* 41 (Dec. 1994), pp. 2445–2454. DOI: 10.1080/09500349414552291.

[58] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[59] Ivan Damgård and Tomas Toft. “Trading Sugar Beet Quotas - Secure Multiparty Computation in Practice”. In: *ERCIM News* 2008.73 (2008). URL: http://ercim-news.ercim.eu/trading-sugar-beet-quotas-secure-multiparty-computation-in-practice.

[60] Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. “Fine-Grained Cryptography”. In: *Advances in Cryptology – CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 533–562. DOI: 10.1007/978-3-662-53015-3_19.

[61] Daniel Demmler, Thomas Schneider, and Michael Zohner. “ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”. In: *ISOC Network and Distributed System Security Symposium – NDSS 2015*. San Diego, CA, USA: The Internet Society, Feb. 2015.

[62] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. “Tor: The Second-Generation Onion Router”. In: *USENIX Security 2004: 13th USENIX Security Symposium*. Ed. by Matt Blaze. San Diego, CA, USA: USENIX Association, Aug. 2004, pp. 303–320.

[63] Nico Döttling, Alexander Koch, Sven Maier, Jeremias Mechler, Anne Müller, Jörn Müller-Quade, and Marcel Tiepelt. *Towards Everlasting Bit Commitment from Quantum Decay*. Unpublished manuscript. 2022.

[64] Tamara M. Dugan and Xukai Zou. “A Survey of Secure Multiparty Computation Protocols for Privacy Preserving Genetic Tests”. In: *Proceedings of the First IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies, CHASE 2016, Washington, DC, USA, June 27-29, 2016*. IEEE Computer Society, 2016. DOI: 10.1109/CHASE.2016.71. URL: https://doi.org/10.1109/CHASE.2016.71.

[65] Paul Dumais, Dominic Mayers, and Louis Salvail. “Perfectly Concealing Quantum Bit Commitment from any Quantum One-Way Permutation”. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, May 2000, pp. 300–315. DOI: 10.1007/3-540-45539-6_21.

[66] DWave. *DWave Quantum Computer*. https://web.archive.org/web/20210702142326/https://www.dwavesys.com/sites/default/files/D-Wave%202000Q%20Tech%20Collateral_0117F.pdf. Archived; accessed 20 Oct 2021. 2015.

[67] Alex Escala and Jens Groth. “Fine-Tuning Groth-Sahai Proofs”. In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, Heidelberg, Germany, Mar. 2014, pp. 630–649. DOI: 10.1007/978-3-642-54631-0_36.

[68] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. “Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy”. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 1775–1792.

[69] Shimon Even, Oded Goldreich, and Abraham Lempel. “A Randomized Protocol for Signing Contracts”. In: *Advances in Cryptology – CRYPTO’82*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 205–210.

[70] Valerie Fetzer, Marcel Keller, Sven Maier, Markus Raiber, Andy Rupp, and Rebecca Schwerdt. “PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics”. In: *Proceedings on Privacy Enhancing Technologies* 2022.2 (Apr. 2022), pp. 447–516. DOI: 10.2478/popets-2022-0054.

[71] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86 (3 Sept. 2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324.

[72] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. *Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review*. arXiv e-prints, Report 2007.10760. https://arxiv.org/abs/2007.10760. 2020.

[73] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *41st Annual ACM Symposium on Theory of Computing*. Ed. by Michael Mitzenmacher. Bethesda, MD, USA: ACM Press, May 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[74] Craig Gentry, Amit Sahai, and Brent Waters. “Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based”. In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5.

[75] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Vol. 2. Cambridge, UK: Cambridge University Press, 2004. ISBN: ISBN 0-521-83084-2 (hardback).

[76] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Vol. 1. Cambridge, UK: Cambridge University Press, 2001, pp. xix + 372. ISBN: 0-521-79172-3 (hardback).

[77] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[78] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract)”. In: *27th Annual Symposium on Foundations of Computer Science*. Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 1986, pp. 174–187. DOI: 10.1109/SFCS.1986.47.

[79] Shafi Goldwasser and Silvio Micali. “Probabilistic Encryption”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299.

[80] Google AI Blog. *On the Path to Cryogenic Control of Quantum Processors*. https://ai.googleblog.com/2019/02/on-path-to-cryogenic-control-of-quantum.html. Online; accessed 21 Oct 2021. 2019.

[81] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. “Founding Cryptography on Tamper-Proof Hardware Tokens”. In: *TCC 2010: 7th Theory of Cryptography Conference*. Ed. by Daniele Micciancio. Vol. 5978. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Heidelberg, Germany, Feb. 2010, pp. 308–326. DOI: 10.1007/978-3-642-11799-2_19.

[82] Matthew Green, Watson Ladd, and Ian Miers. “A Protocol for Privately Reporting Ad Impressions at Scale”. In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. Vienna, Austria: ACM Press, Oct. 2016, pp. 1591–1601. DOI: 10.1145/2976749.2978407.

[83] Jens Groth and Amit Sahai. “Efficient Non-interactive Proof Systems for Bilinear Groups”. In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Istanbul, Turkey: Springer, Heidelberg, Germany, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24.

[84] Saikat Guha, Bin Cheng, and Paul Francis. “Privad: Practical Privacy in Online Advertising”. In: *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. Ed. by David G. Andersen and Sylvia Ratnasamy. USENIX Association, 2011. URL: https://www.usenix.org/conference/nsdi11/privad-practical-privacy-online-advertising.

[85] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. “BBA+: Improving the Security and Applicability of Privacy-Preserving Point Collection”. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. Dallas, TX, USA: ACM Press, Oct. 2017, pp. 1925–1942. DOI: 10.1145/3133956.3134071.

[86] Nicholas J. Hopper, John Langford, and Luis von Ahn. “Provably Secure Steganography”. In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 77–92. DOI: 10.1007/3-540-45708-9_6.

[87] Klaus Hornberger. "Introduction to Decoherence Theory". In: *Entanglement and Decoherence: Foundations and Modern Trends*. Ed. by Andreas Buchleitner, Carlos Viviescas, and Markus Tiersch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 221–276. ISBN: 978-3-540-88169-8. DOI: 10.1007/978-3-540-88169-8_5.

[88] Lane P. Hughston, Richard Jozsa, and William K. Wootters. "A Complete Classification of Quantum Ensembles Having a Given Density Matrix". In: *Physics Letters A* 183 (1993), pp. 14–18. ISSN: 0375-9601.

[89] Russell Impagliazzo. "A Personal View of Average-Case Complexity". In: *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*. IEEE Computer Society, 1995, pp. 134–147. DOI: 10.1109/SCT.1995.514853.

[90] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. "Founding Cryptography on Oblivious Transfer - Efficiently". In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2008, pp. 572–591. DOI: 10.1007/978-3-540-85174-5_32.

[91] Fayaz Itoo, Meenakshi, and Satwinder Singh. "Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection". In: *International Journal of Information Technology* (2020), pp. 1–9. DOI: 10.1007/s41870-020-00430-yj. URL: https://doi.org/10.1007/s41870-020-00430-y.

[92] Tibor Jager and Andy Rupp. "Black-Box Accumulation: Collecting Incentives in a Privacy-Preserving Way". In: *Proceedings on Privacy Enhancing Technologies* 2016.3 (July 2016), pp. 62–82.

[93] Stanislaw Jarecki. "Efficient Covert Two-Party Computation". In: *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10769. Lecture Notes in Computer Science. Rio de Janeiro, Brazil: Springer, Heidelberg, Germany, Mar. 2018, pp. 644–674. DOI: 10.1007/978-3-319-76578-5_22.

[94] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference". In: *USENIX Security 2018: 27th USENIX Security Symposium*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, Aug. 2018, pp. 1651–1669.

[95] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. Virtual Event, USA: ACM Press, Nov. 2020, pp. 1575–1590. DOI: 10.1145/3372297.3417872.

[96] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. Vienna, Austria: ACM Press, Oct. 2016, pp. 830–842. DOI: 10.1145/2976749.2978357.

[97] Adrian Kent. “Unconditionally Secure Bit Commitment”. In: *Physical Review Letters* 83 (Aug. 1999), pp. 1447–1450. DOI: 10.1103/PhysRevLett.83.1447.

[98] Navanshu Khare and Saad Yunus Sait. “Credit Card Fraud Detection Using Machine Learning Models and Collating Machine Learning Models”. In: *International Journal of Pure and Applied Mathematics* 118.20 (2018), pp. 825–838.

[99] Joe Kilian. “Founding Cryptography on Oblivious Transfer”. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 1988, pp. 20–31. DOI: 10.1145/62212.62215.

[100] Robert Koenig, Stephanie Wehner, and Jürg Wullschleger. “Unconditional Security From Noisy Quantum Storage”. In: *IEEE Transactions on Information Theory* 58.3 (2012), pp. 1962–1984. DOI: 10.1109/TIT.2011.2177772.

[101] Vladimir Kolesnikov, Ranjit Kumaresan, and Abdullatif Shikfa. “Efficient Verification of Input Consistency in Server-Assisted Secure Function Evaluation”. In: *CANS 12: 11th International Conference on Cryptology and Network Security*. Ed. by Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis. Vol. 7712. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, Dec. 2012, pp. 201–217. DOI: 10.1007/978-3-642-35404-5_16.

[102] Takeshi Koshiba and Takanori Odaira. “Statistically-Hiding Quantum Bit Commitment from Approximable-Preimage-Size Quantum One-Way Function”. In: *Theory of Quantum Computation, Communication, and Cryptography, 4th Workshop, TQC 2009, Waterloo, Canada, May 11-13, 2009, Revised Selected Papers*. Ed. by Andrew M. Childs and Michele Mosca. Vol. 5906. Lecture Notes in Computer Science. Springer, 2009, pp. 33–46. DOI: 10.1007/978-3-642-10698-9_4.

[103] Christiane Kuhn, Dennis Hofheinz, Andy Rupp, and Thorsten Strufe. *Onion Routing with Replies*. Cryptology ePrint Archive, Report 2021/1178. https://eprint.iacr.org/2021/1178. 2021.

[104] Christiane Kuhn, Dennis Hofheinz, Andy Rupp, and Thorsten Strufe. “Onion Routing with Replies”. In: *Advances in Cryptology – ASIACRYPT 2021, Part II*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13091. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Dec. 2021, pp. 573–604. DOI: 10.1007/978-3-030-92075-3_20.

[105] Hoi-Kwong Lo and Hoi-Fung Chau. “Is Quantum Bit Commitment Really Possible?” In: *Physical Review Letters* 78 (17 Apr. 1997), pp. 3410–3413. DOI: 10.1103/PhysRevLett.78.3410.

[106] Urmila Mahadev. “Classical Homomorphic Encryption for Quantum Circuits”. In: *59th Annual Symposium on Foundations of Computer Science*. Ed. by Mikkel Thorup. Paris, France: IEEE Computer Society Press, Oct. 2018, pp. 332–338. DOI: 10.1109/FOCS.2018.00039.

[107] Dominic Mayers. *The Trouble with Quantum Bit Commitment*. arXiv e-prints, Report quant-ph/9603015. https://arxiv.org/abs/quant-ph/9603015. 1996.

[108] Dominic Mayers. “Unconditionally Secure Quantum Bit Commitment is Impossible”. In: *Physical Review Letters* 78 (17 Apr. 1997), pp. 3414–3417. DOI: 10.1103/PhysRevLett.78.3414.

[109] Ralph C. Merkle. “Secure Communications Over Insecure Channels”. In: *Communications of the ACM* 21.4 (1978), pp. 294–299. DOI: 10.1145/359460.359473.

[110] MIT Technology Review. *IBM Raises the Bar with a 50-Qubit Quantum Computer.* https://www.technologyreview.com/2017/11/10/147728/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/. Online; accessed 21 Oct 2021. 2017.

[111] Payman Mohassel and Peter Rindal. “ABY^3: A Mixed Protocol Framework for Machine Learning”. In: *ACM CCS 2018: 25th Conference on Computer and Communications Security.* Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. Toronto, ON, Canada: ACM Press, Oct. 2018, pp. 35–52. DOI: 10.1145/3243734.3243760.

[112] Jörn Müller-Quade and Dominique Unruh. “Long-Term Security and Universal Composability”. In: *TCC 2007: 4th Theory of Cryptography Conference.* Ed. by Salil P. Vadhan. Vol. 4392. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Feb. 2007, pp. 41–60. DOI: 10.1007/978-3-540-70936-7_3.

[113] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. “Spectrum: High-bandwidth Anonymous Broadcast”. In: *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022.* Ed. by Amar Phanishayee and Vyas Sekar. USENIX Association, 2022. URL: https://www.usenix.org/conference/nsdi22/presentation/newman.

[114] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition).* Cambridge University Press, 2016. ISBN: 978-1-10-700217-3. URL: https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB.

[115] Dave Philipps. “Reality Winner, former NSA translator, gets more than 5 years in leak of Russian hacking report”. In: *New York Times* 23 (2018).

[116] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer.* Cryptology ePrint Archive, Report 2005/187. https://eprint.iacr.org/2005/187. 2005.

[117] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *37th Annual ACM Symposium on Theory of Computing.* Ed. by Harold N. Gabow and Ronald Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

[118] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. “On data banks and privacy homomorphisms”. In: *Foundations of secure computation.* Vol. 4. 11. 1978, pp. 169–180.

[119] Phillip Rogaway. “Nonce-Based Symmetric Encryption”. In: *Fast Software Encryption – FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. Lecture Notes in Computer Science. New Delhi, India: Springer, Heidelberg, Germany, Feb. 2004, pp. 348–359. DOI: 10.1007/978-3-540-25937-4_22.

[120] Kamyar Saeedi, Stephanie Simmons, Jeff Z. Salvail, Phillip Dluhy, Helge Riemann, Nikolai V. Abrosimov, Peter Becker, Hans-Joachim Pohl, John J. L. Morton, and Mike L. W. Thewalt. “Room-Temperature Quantum Bit Storage Exceeding 39 Minutes Using Ionized Donors in Silicon-28”. In: *Science* 342.6160 (2013), pp. 830–833. ISSN: 0036-8075. DOI: 10.1126/science.1239584.

[121] Andrew W. Senior, Sharath Pankanti, Arun Hampapur, Lisa M. Brown, Ying-li Tian, Ahmet Ekin, Jonathan H. Connell, Chiao-Fe Shu, and Max Lu. “Enabling Video Privacy through Computer Vision”. In: *IEEE Security & Privacy* 3.3 (2005). DOI: 10.1109/MSP.2005.65. URL: https://doi.org/10.1109/MSP.2005.65.

[122] Yaoyun Shi. *Both Toffoli and Controlled-NOT need little help to do universal quantum computation*. arXiv e-prints, Report quant-ph/0205115. https://arxiv.org/abs/quant-ph/0205115. 2002.

[123] Peter W. Shor. “Fault-Tolerant Quantum Computation”. In: *37th Annual Symposium on Foundations of Computer Science*. Burlington, Vermont: IEEE Computer Society Press, Oct. 1996, pp. 56–65. DOI: 10.1109/SFCS.1996.548464.

[124] Gustavus J. Simmons. “The Prisoners’ Problem and the Subliminal Channel”. In: *Advances in Cryptology – CRYPTO’83*. Ed. by David Chaum. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1983, pp. 51–67.

[125] Krysta M. Svore, Alfred V. Aho, Andrew W. Cross, Isaac L. Chuang, and Igor L. Markov. “A Layered Software Architecture for Quantum Computing Design Tools”. In: *Computer* 39.1 (2006), pp. 74–83. DOI: 10.1109/MC.2006.4.

[126] Swamit S. Tannu and Moinuddin K. Qureshi. “Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers”. In: *International Conference on Architectural Support for Programming Languages and Operating Systems*. Ed. by Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck. ACM, 2019, pp. 987–999. DOI: 10.1145/3297858.3304007.

[127] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. “Adnostic: Privacy Preserving Targeted Advertising”. In: *ISOC Network and Distributed System Security Symposium – NDSS 2010*. San Diego, CA, USA: The Internet Society, Feb. 2010.

[128] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. “Stealing Machine Learning Models via Prediction APIs”. In: *USENIX Security 2016: 25th USENIX Security Symposium*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 601–618.

[129] Dominique Unruh. “Everlasting Multi-party Computation”. In: *Advances in Cryptology – CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2013, pp. 380–397. DOI: 10.1007/978-3-642-40084-1_22.

[130] Dominique Unruh. "Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model". In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 755–784. DOI: 10.1007/978-3-662-46803-6_25.

[131] Leslie G. Valiant. "Universal Circuits (Preliminary Report)". In: *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*. Ed. by Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison. Hershey, Pennsylvania, USA: ACM, May 1976, pp. 196–203. DOI: 10.1145/800113.803649.

[132] Luis von Ahn and Nicholas J. Hopper. "Public-Key Steganography". In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Interlaken, Switzerland: Springer, Heidelberg, Germany, May 2004, pp. 323–341. DOI: 10.1007/978-3-540-24676-3_20.

[133] Luis von Ahn, Nicholas J. Hopper, and John Langford. "Covert two-party computation". In: *37th Annual ACM Symposium on Theory of Computing*. Ed. by Harold N. Gabow and Ronald Fagin. Baltimore, MA, USA: ACM Press, May 2005, pp. 513–522. DOI: 10.1145/1060590.1060668.

[134] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: *2019 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2019, pp. 707–723. DOI: 10.1109/SP.2019.00031.

[135] Ye Wang, Mark Um, Zhang Junhua, Shuoming An, Ming Lyu, Jing-ning Zhang, L.-M. Duan, Dahyun Yum, and Kihwan Kim. "Single-qubit quantum memory exceeding 10-minute coherence time". In: *Nature Photonics* 11.10 (Sept. 2017), pp. 646–650. ISSN: 1749-4893. DOI: 10.1038/s41566-017-0007-1.

[136] Stephen Wiesner. "Conjugate Coding". In: *SIGACT News* 15.1 (Jan. 1983), pp. 78–88. DOI: 10.1145/1008908.1008920.

[137] Douglas Wikström. "A Universally Composable Mix-Net". In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 317–335. DOI: 10.1007/978-3-540-24638-1_18.

[138] Thomas Winkler and Bernhard Rinner. "A systematic approach towards user-centric privacy and security for smart camera networks". In: *2010 Fourth ACM/IEEE International Conference on Distributed Smart Cameras, Atlanta, GA, USA - August 31 - September 4, 2010*. Ed. by Marilyn Wolf, Gian Luca Foresti, and Horst Bischof. ACM, 2010. DOI: 10.1145/1865987.1866009. URL: https://doi.org/10.1145/1865987.1866009.

[139] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *27th Annual Symposium on Foundations of Computer Science*. Toronto, Ontario, Canada: IEEE Computer Society Press, Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[140] Andrew Chi-Chih Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd Annual Symposium on Foundations of Computer Science*. Chicago, Illinois: IEEE Computer Society Press, Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[141] Hyunwoo Yu, Jaemin Lim, Kiyeon Kim, and Suk-Bok Lee. “Pinto: Enabling Video Privacy for Commodity IoT Cameras”. In: *ACM CCS 2018: 25th Conference on Computer and Communications Security*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. Toronto, ON, Canada: ACM Press, Oct. 2018, pp. 1089–1101. DOI: 10.1145/3243734.3243830.

[142] Mark Zhandry. “How to Record Quantum Queries, and Applications to Quantum Indifferentiability”. In: *Advances in Cryptology – CRYPTO 2019, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2019, pp. 239–268. DOI: 10.1007/978-3-030-26951-7_9.

[143] Mark Zhandry. “Secure Identity-Based Encryption in the Quantum Random Oracle Model”. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 758–775. DOI: 10.1007/978-3-642-32009-5_44.

[144] Shuaining Zhang, Yao Lu, Kuan Zhang, Wentao Chen, Ying Li, Jing-Ning Zhang, and Kihwan Kim. “Error-mitigated quantum gates exceeding physical fidelities in a trapped-ion system”. In: *Nature Communications* 11.1 (Jan. 2020). ISSN: 2041-1723. DOI: 10.1038/s41467-020-14376-z.

[145] Manjin Zhong, Morgan Hedges, Rose Ahlefeldt, John Bartholomew, Sarah Beavan, Sven Wittig, Jevon Longdell, and Matthew Sellars. “Optically addressable nuclear spins in a solid with a six-hour coherence time”. In: *Nature* 517 (Jan. 2015), pp. 177–180. DOI: 10.1038/nature14025.