On Provable Security for Complex Systems

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

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

genehmigte

Dissertation

von

Dirk Achenbach

aus Villingen-Schwenningen

Tag der mündlichen Prüfung: 29. Januar 2016

Erster Gutachter: Prof. Dr. Jörn Müller-Quade

Zweiter Gutachter: Prof. Dr. Ralf Reussner

Abstract

Computers are ubiquitous. They are critical for a multitude of operations in business, government, and society. Simultaneously they become ever more complex. Consequently, while they gain in importance, it is increasingly difficult to guarantee their security. Because security is a non-functional property, it can neither be tested for, nor can a system be made secure in an ad-hoc fashion. Security can only be included as an inherent property of a system *by design*.

We investigate the contribution of cryptographic proofs of security to a systematic security engineering process.

To this end we study how to model and prove security for concrete applications in three practical domains: computer networks, data outsourcing, and electronic voting.

In the first domain, computer networks, we investigate the security of a network component irrespective of the surrounding network. As a concrete application, we study how to combine several candidate firewalls to yield one provably-secure firewall, even if one candidate is compromised. Our modeling of a firewall network allows us to define security independent of a firewall's functionality. We show that a concatenation of firewalls is not secure, while a majority decision is. Second, we develop a framework for the privacy of outsourced data. Specifically, we consider privacy in the presence of queries. We identify three privacy objectives—data privacy, query privacy, and result privacy. They apply to a wide array of outsourcing applications. Our analysis yields generic relations among these objectives. We introduce a searchable encryption scheme that fits in the framework. The third domain is electronic voting. We focus on re-voting as a strategy to achieve coercion resistance. Insights from a cryptographic model of coercion resistance yield requirements for possible realisations. We provide a proof-of-concept scheme that achieves coercion resistance through revoting.

We conclude that cryptographic proofs of security can benefit a security engineering process in formulating requirements, influencing design, and identifying constraints for the implementation.

Zusammenfassung

Computer sind heute allgegenwärtig. Für viele Aufgaben in Wirtschaft, Politik und Gesellschaft sind sie unerlässlich. Gleichzeitig werden sie immer komplexer: In einem modernen Kraftfahrzeug arbeiten Programme im Umfang von einhundert Millionen Zeilen Quelltext [167]. Dabei steuert nicht ein einziger Computer alle Fahrzeugfunktionen. Im Fahrzeug arbeitet ein komplexes *System* aus Computern. Während Systeme der Informationstechnologie (IT) an Bedeutung und Komplexität zunehmen, wird es zunehmend schwieriger, ihre Sicherheit zu gewährleisten. Da IT-Sicherheit aber eine nicht-funktionale Eigenschaft ist, ist es schwer zu testen, ob ein System sicher ist: Darf es nach dem erfolgreichen Durchlaufen aller Testfälle als sicher gelten; decken die Testfälle alle möglichen Sicherheitslücken ab? Sicherheit kann nur als inhärente Systemeigenschaft bereits *beim Entwurf* eines Systems verankert werden. Ein systematisches Vorgehen, das ein sicheres IT-System zum Ziel hat, ist *Security Engineering*.

Der Nachweis von Sicherheit ist der Grundsatz der modernen Kryptographie. Katz und Lindell nennen drei Prinzipien [131]: die Formulierung exakter Definitionen, der Verlass auf präzise Annahmen und streng geführte Sicherheitsbeweise. Sicherheit ist eine wohldefinierte Eigenschaft eines Systems. Ein stringenter Beweis weist sie auf der Grundlage von Annahmen nach. Dazu müssen diese präzise formuliert sein und können somit objektiv untersucht werden. Die exakte Definition von Sicherheit wiederum erlaubt die begründete Auswahl von Sicherheitsmaßnahmen. Ein Beweis deckt den Zusammenhang zwischen Annahmen, einem Systementwurf und einer Sicherheitsgarantie auf.

In dieser Dissertation wird der Frage nachgegangen, welchen Beitrag die Methode der beweisbaren Sicherheit zu einem systematischen Entwurfsprozess für sichere IT-Systeme leisten kann. Gegenstand ist dabei das Wasserfallmodell: Die Entwicklung sicherer Systeme in diesem Modell verläuft linear in den Stufen Anforderungen, Entwurf, Implementierung, Überprüfung und Veröffentlichung.

Die Modellierung von Sicherheit und ihr Nachweis wird an konkreten Anwendungen in drei praktischen Feldern untersucht: Computernetzwerke, das Auslagern von Daten und elektronische Wahlen. Die Arbeit besteht aus sechs Kapiteln. Kapitel 1 motiviert das Vorhaben und macht einen Vorschlag für einen Entwicklungsprozess, der durch die kryptographische Methode unterstützt wird. Formale Grundlagen, Konventionen und Voraussetzungen werden in Kapitel 2 behandelt.

Kapitel 3 widmet sich dem ersten Anwendungsfeld, Computernetzwerken. Hier wird die Sicherheit einer Netzwerkkomponente unabhängig vom umgebenden Netzwerk untersucht. Dabei werden mehrere Firewalls so kombiniert, dass insgesamt eine beweisbar sichere Firewall entsteht – sogar dann, wenn eine der Firewalls bösartig ist [3]. Durch die

Verwendung des *Universal Composability (UC)*-Rahmenwerks [41] kann die Sicherheit einer Kombination von Firewalls unabhängig von der Funktionalität einer Firewall definiert werden. Dabei dient ein vertrauenswürdiger Hardwarebaustein, der Netzwerkpakete vergleicht, als Sicherheitsannahme. Es wird gezeigt, dass eine Serienschaltung zweier Firewalls keine sichere Lösung liefert, ein Konsensentscheid jedoch schon. Er birgt aber die Gefahr, dass der Angreifer durch die Unterdrückung von Paketen die Verfügbarkeit des Netzwerks angreift. Ein *2-aus-3*-Mehrheitsentscheid hat diese Schwäche scheinbar nicht. Da das reine UC-Modell aber Zeit nicht abbildet, ist der Nachweis einer solchen Verfügbarkeitseigenschaft nicht ohne Weiteres möglich.

Die Verfügbarkeit der Architektur, die einen *2-aus-3*-Mehrheitsentscheid realisiert, kann nachgewiesen werden [4], indem eine Erweiterung von Katz et al. [132] einbezogen wird. Um die kryptographische Modellierung von Netzwerken zu vereinfachen, werden weiter zwei Formalismen und ein Fünf-Schritt-Vorgehen bei der Definition von Netzwerkprotokollen vorgeschlagen.

In Kapitel 4 wird ein Rahmenwerk für die Geheimhaltung ausgelagerter Daten entwickelt. Insbesondere wird dabei die Datensicherheit betrachtet, wenn die ausgelagerten Daten abgefragt werden. Dazu wird eine ausgelagerte Datenbankanwendung in zwei Phasen modelliert. In der Vorbereitungsphase wird die auszulagernde Datenbank vorbereitet und an einen *Server* übertragen. In der Anfragephase stellt ein *Client* Anfragen an den Server [1]. Drei Schutzziele werden identifiziert: die Geheimhaltung der Daten selbst, die Geheimhaltung der Anfragen und die Geheimhaltung der Anfrageergebnisse. Die Definitionen dieser Ziele sind dabei von der konkreten Anwendung unabhängig und decken ausgelagerte relationale Datenbanken ebenso ab wie durchsuchbare Verschlüsselungen. Eine Analyse liefert grundlegende Zusammenhänge zwischen diesen Schutzzielen: Die Geheimhaltung der Daten ist unabhängig von der Geheimhaltung der Anfragen; beide zusammen sind jedoch äquivalent zur Geheimhaltung der Anfrageergebnisse. Derart gefundene Zusammenhänge sind unabhängig von einer konkreten Realisierung gültig und geben bereits in der Entwurfsphase Hinweise darauf, welche Schutzziele unabhängig, widersprüchlich oder redundant sind. Weiter wird eine durchsuchbare Verschlüsselung vorgestellt, deren Sicherheit im Rahmenwerk abgebildet werden kann [5]. Sie basiert auf gerichteten, azyklischen Wort-Graphen (*Directed Acyclic Word Graphs, DAWGs*). Ein DAWG ist ein endlicher Automat, der alle Teil-Zeichenketten einer Zeichenkette akzeptiert.

Kapitel 5 thematisiert elektronische Wahlen. Bei Wahlen, die im Internet stattfinden, verfügt der Wähler über keine Wahlkabine, in der er Schutz vor Erpressungsversuchen findet. (Smartphones mit Kamerafunktion relativieren selbst den Schutz einer Wahlkabine [26, 195].) Es bedarf daher besonderer Maßnahmen, den Wähler vor Erpressung zu schützen. Eine solche Maßnahme ist sogenanntes *Revoting*. Hier kann der Wähler seine Stimme jederzeit durch die Abgabe einer neuen überschreiben. Damit Revoting aber als Maßnahme dienen kann, einem Erpressungsversuch zu entgehen, muss der Wähler sein erneutes Abstimmen plausibel abstreiten können. Trotzdem muss sichergestellt sein, dass nur seine zuletzt abgegebene Stimme gezählt wird. Ein kryptographisches Modell

für Nichterpressbarkeit durch Revoting wird, basierend auf einem Modell für eine andere Maßnahme [122], entwickelt [2]. Weiterhin wird ein Verfahren vorgestellt, das die grundsätzliche Machbarkeit von Revoting als Strategie belegt.

Aus der Arbeit mit dem kryptographischen Modell wird eine Anforderung an die Autorisierung des Wählers abgeleitet, die unabhängig vom konkreten Verfahren ist: Sie muss *unveräußerlich* sein. Das heißt, es muss dem Wähler unmöglich sein sie abzugeben oder eine Kopie zu erstellen.

Kapitel 6 schließt die Arbeit ab und diskutiert offene Fragen.

Aus den vorangegangenen Untersuchungen ergibt sich, dass kryptographische Sicherheitsbeweise einen *Security-Engineering*-Prozess auf dreierlei Arten unterstützen können: Sie helfen Anforderungen zu formulieren, systematisieren den Entwurf und zeigen Bedingungen an eine Implementierung auf. Kryptographische Sicherheitsdefinitionen sind exakt formulierte Anforderungen an ein System und seine Komponenten. Beweise liefern Konstrukte für seinen Entwurf und geben Einschränkungen an die Implementierung.

Zwei Implementierungen des Paketvergleichers für Firewalls und die Implementierung der durchsuchbaren Verschlüsselung belegen die grundsätzliche Umsetzbarkeit der kryptographischen Protokolle. Die Integration der kryptographischen Methode in einen gesamten *System-Engineering*-Prozess wird jedoch nicht demonstriert. Ein solches Vorhaben bleibt ein wichtiger offener Schritt. Die Arbeit beschränkt sich weiterhin nur auf einen Prozess nach dem Wasserfallmodell. Die Frage, ob und wie sich die kryptographische Methode in einen agilen Entwicklungsprozess einbringen lässt, bleibt offen.

Acknowledgements

I am fortunate to have had the support of outstanding people in accomplishing this work.

First and foremost I express my gratitude to Jörn Müller-Quade for supporting and supervising my dissertation. I also express my thanks to the co-referee of my thesis, Ralf Reussner, and the other professors who took interest in my work and gave valuable feedback: Bernhard Beckert, Jürgen Beyerer, Klemens Böhm, Hannes Hartenstein, Gregor Snelting, and Martina Zitterbart. Also, I thank Jeroen van de Graaf for his counsel.

I had the honour to work with wonderful co-authors: Rolf Haynberg, Matthias Huber, Carmen Kempka, Bernhard Löwe, and Jochen Rill.

Thanks go to my colleagues for inspiring discussions and many happy memories. I am particularly grateful to Brandon Broadnax and Christian Henrich for their support.

My sincerest thanks go to Willi Geiselmann, Holger Hellmuth, and Carmen Manietta, for keeping the institute running behind the scenes and instilling it with an aura of calmness.

I had the pleasure of working with many amazing students, some of whom later became valued co-authors. In particular I want to thank Jeremias Mechler for his untiring assistance. Also, I point out Simon Karolus, whose Bachelor thesis taught me a lot about the practical aspects of my work, and thank Roland Bless, Christian Haas, and Hans Wippel for co-advising it.

I can never express my gratitude to friends and family enough. They got through some difficult times with me. Mama, Papa, Emanuele, thank you.

Contents

Acronyms

CPA Chosen-Plaintext Attack

DAWG Directed Acyclic Word Graph

DDH Decisional Diffie-Hellman

DPDK Data Plane Development Kit

EPET Encrypted Plaintext Equivalence Test

EUF-CMA Existential Unforgeability under Chosen-Message Attacks

HTTP Hypertext Transfer Protocol

IND-CPA Indistinguishability under Chosen-Plaintext Attack

IP Internet Protocol

IT Information Technology

ITM Interactive Turing Machine

ITSEC Information Technology Security Evaluation Criteria

KIT Karlsruhe Institute of Technology

LAN Local-Area Network

MPC Multi-Party Computation

NIZK Non-Interactive Zero-Knowledge

NIZKPoK Non-Interactive Zero-Knowledge Proof of Knowledge

ORAM Oblivious Random Access Machine

PET Plaintext Equivalence Test

PIN Personal Identification Number

PIR Private Information Retrieval

PPT Probabilistic Polynomial Time

PUF Physically Uncloneable Function

RAM Random Access Machine

RAM Random Access Memory

SDL Security Development Lifecycle

SEDAWG Searchable Encryption using Directed Acyclic Word Graphs

SSE Symmetric Searchable Encryption

STV Single Transferable Vote

TCP Transmission Control Protocol

TM Turing Machine

TTL Time To Live

UC Universal Composability

XPM Exact Pattern Matching

XPM-SSE Exact Pattern Matching for Symmetric Searchable Encryption

ZK Zero-Knowledge

1 Introduction

> The alternative [to rigorous proofs of security] is to design systems in an ad hoc manner, and to simply hope for the best. As we have seen, this approach often ends in disaster, or at least, an embarrassing mess.
>
> *(Victor Shoup, Why Chosen Ciphertext Security Matters [184])*

It is hard to imagine a life without computers. They are involved even in the most mundane activities. A computer was used to sell the author's breakfast. This text was typed on a keyboard that was designed on a computer, manufactured using computer-aided machinery and was delivered by a truck that was navigated with the help of a computer. There is a substantial chance that the author's life itself will at some point depend on a computer.

But not only are computers ubiquitous, they also have increased in complexity over time: Apollo 11's Guidance Computer (AGC) took roughly 145 000 lines of code to implement, a modern car runs a hundred million [119, 167].

It seems inappropriate to speak of a "computer" operating a car—inside the computer are more computers with distinct purposes and clearly-defined interfaces. It is more adequate to speak of a complex *system*.

As Information Technology (IT) is increasingly involved in ensuring and improving our wealth, well-being, and comfort, it is also a target for those wishing us harm. Systems need not only be robust against mischance, but also against malice [14]. Security is not a functional property of a system. For this reason one can hardly test for security: Is a system secure when it has passed all test cases; do the test cases cover all possible vulnerabilities? On the contrary, security is an inherent property of a system. In the same way complex buildings cannot be built in an *ad-hoc* fashion, but need planning by architects and engineers, reliable IT systems are designed in a systematic process. The facet of designing systems that is concerned with the system's security is *security engineering*.

A secure system is not trivially obtained by combining secure subsystems. As an example, take the Chip-and-PIN system that is used for electronic transactions at points of sale. It makes use of a cryptographic chip embedded in a bank card, issued by the bank to the customer. The protocol offers two alternative methods for authorisation: either the customer enters a Personal Identification Number (PIN) or the transaction is authorised by the customer's signature. Both procedures are acceptable methods.

Because of the way both are integrated, the combined system has a weakness. In 2010, Murdoch et al. [164] presented a man-in-the-middle attack that confirms any PIN as valid to the point-of-sale terminal, while the card never verifies the PIN and is led to believe the transaction is authorised by signature. The attack renders both authorisation methods useless. A year later, this weakness was exploited by fraudsters in Belgium, using credit cards stolen in France [82]. (This is also a prime example of a contrived attack that is actually implemented later.)

In this work, we address the question of how the cryptographic method of provable security can improve a security engineering process.

1.1 Security Engineering

A *security engineering* process is a methodical effort to develop a secure system. There are several approaches to such a methodology.

Figure 1.1: The baseline software development process that forms the basis of Microsoft's Security Development Lifecycle [151]. We highlighted the phases that can gain from a proactive proof of security.

Many standards and recommendations propose a development process closely related to the waterfall model in software engineering. Development in such a procedure model is organised in consecutive phases. The Information Technology Security Evaluation Criteria (ITSEC) [115] require four phases—Requirements, Architectural Design, Detailed Design, and Implementation—while the newer Common Criteria for Information Technology Security Evaluation (CC) [60] further elaborate the process model. In its "Maßnahmenkatalog M 2.378 System-Entwicklung", the Bundesamt für Sicherheit in der Informationstechnik (BSI) also requires that ITSEC's four development phases are gone through. The Security Development Lifecycle (SDL) [151] developed by Microsoft also follows a waterfall model consisting of the phases Requirements, Design, Implementation, Verification, and Release (see Figure 1.1). (The full SDL also has the phases Release To Manufacturing and Response, but these are outside the scope of this work. While a system's operation and maintenance are important aspects of its lifecycle, we focus on system development.)

Anderson [14] discusses the waterfall model too, but also introduces a spiral model and "evolutionary development", arguing that a top-down approach isn't always feasible in practice, thus giving need to an iterative design process. He makes a particular case for the evolutionary approach, where one starts with a small working system and continuously improves upon it.

So-called *agile* methodologies have been a fashion in software engineering in recent years. The Agile Manifesto [23, 104] gives four values of central importance:

1. individuals and interactions over processes and tools,

2. working software over comprehensive documentation,

3. customer collaboration over contract negotiation,

4. responding to change over following a plan.

We do not dispute the positive impact of agile methodologies on software engineering. We argue that the above-mentioned values induce inherent dynamics to a system engineering process that do not accommodate well to cryptographic rigour. For example, it is not immediately clear how "working software" translates to security properties. One can test easily if a piece of software "works", but not if it's secure. Indeed, security is assessed through comprehensive documentation.

We illustrate our point further for the Scrum method [179]. Development in Scrum is organised in consecutive "sprints". During each sprint, the team selects a set of requirements from a list and implements the code to satisfy them. Because sprint cycles are short, the process adapts well to changing requirements. Cryptographic proof, however, does not. If the model is changed, theorems must painfully be re-proven.

As the waterfall model features a linear progression of phases, it is thus particularly suited for our purposes. We continue this exposition with the waterfall model and leave the application of our insights to agile methods for future work. Yet we concede that in practice, one might be forced to "step back" at some point and resume the process from an earlier phase.

In summary, a systematic security engineering processes involves five stages:

1. Requirements Phase: Security goals are formulated and security challenges are anticipated.

2. Design Phase: The system's architecture is designed, threats to the design are determined and the system's attack surface is estimated.

3. Implementation Phase: The system is implemented, tested and integrated.

4. Verification: Code is reviewed, the implementation is tested.

5. Release: The system is released.

While this methodology is a systematic approach to a secure system design, it suffers from an inherent weakness. Threat modeling and attack surface estimation are processes that only cover threats that the system's designer can conceive of. They are thus limited to the designer's experience and imagination. This limitation leaves part of the security engineering practice an artistic endeavour rather than a scientific process. There is a lack of a systematic method to uncover unforeseen security challenges.

1.2 Provable Security

Modern cryptography has one central paradigm: The rigorous proof of security. Katz and Lindell formulate three basic principles of modern cryptography [131]: the formulation

of exact definitions, the reliance on precise assumptions, and rigorous proofs of security. In more detail, the central paradigm of modern cryptography is to prove the security of one's constructions, beginning at a precise assumption and reaching an exactly-defined claim of security by a rigorous argument. By precisely naming assumptions one opens up the prerequisites of a secure construction to objective analysis. Further, only by having defined what security exactly is, one can adequately select security measures—in the same way a pillar serves a concrete structural purpose in a highway bridge, components in IT systems must serve a clearly-defined purpose. A proof, then, unveils the relationship between assumptions, the construction, and security definitions.

In this thesis we address the question how the application of these principles can benefit a systematic security engineering process.

What is Security? Contrary to popular intuition, security is not the absence of vulnerabilities. Such an assertion were only meaningful if one gave a precise definition of what constitutes a vulnerability. On the contrary, (provable) security is the guarantee that no adversary can violate a precisely-defined property of a system, regardless of his strategy—that is, a security property is a predicate of a system design.

Such a security property is expressed in a model. A model is an abstraction of a system and its surroundings. It determines how each aspect of reality is to be expressed formally, and what the assumed capabilities of honest participants and the adversary are. Such an assumption might for example be "factoring natural numbers is hard" or "the adversary can eavesdrop on all communications, but not alter them" (i.e. is passive). Assumptions make the system designer's beliefs about reality explicit, thus opening them to scientific debate. Picking up an example from before, the designers of the Chip-and-PIN system *implicitly* assumed no adversary can control the communication between the card and the terminal. By violating this assumption, Murdoch et al. came up with an attack on the system. A cryptographic proof of security for the Chip-and-PIN system would have made this implicit assumption *explicit*, thus exposing it to scrutiny.

A cryptographic proof establishes a relationship between one or more assumptions and a security property. As for a definition of what is a proof, we quote Errett Bishop [137]: "A proof is any completely convincing argument." A proof of security, then, is a completely convincing argument that a security property holds under given assumptions.

Because a model is an abstraction, it only represents certain aspects of reality. This is an obvious advantage, as abstraction reduces complexity. As cryptographic models also abstract from concrete realisations, one can derive generic relations, possibilities and even impossibility results independently of an implementation. As a downside, if the model is too generic or not adequate, results obtained in the cryptographic abstraction do not apply to the concrete system.

1.3 Contributions of This Work

In this thesis we investigate how to model and prove security for complex systems in three practical domains.

1.3.1 Computer Networks

In Chapter 3 we investigate how to cryptographically model computer networks. This field of investigation is an example of how one can improve the security of a network component without compromising the network at large. We focus on a concrete application: network firewalls. Concretely, we address the research question of how to combine several candidate firewalls to yield a secure firewall, even if one candidate is compromised. We show that a concatenation of candidates does not yield a secure solution, while a quorum decision does. We assume a trusted hardware device that compares packets. We exploit a property of the Universal Composability framework that enables us to model the security of a firewall network without specifying a firewall functionality—our definition is valid for *any* firewall functionality. Our proof of security for a quorum of three firewalls requires us to assume a consensus of non-compromised firewalls. In a second step we extend the model to also cover availability, making use of a result of Katz et al. [132]. We propose a "wrapper" functionality $\mathcal{F}_{\mathsf{wrap}}$ and a 5-phase paradigm to ease the handling of formalisms. We also discuss two implementations of the trusted hardware device.

We learned an important lesson while trying to prove Theorem 2 in Chapter 3.5. Our original theorem asserted the opposite of the theorem's statement. By failing to prove the theorem and analysing our failure, we found an attack on the network that now is part of the correct theorem's proof. This experience shows that blind trust in one's security intuition is foolish. By forcing the cryptographer to formulate a straightforward argument the chance is increased that he becomes aware of his errors of reasoning.

1.3.2 Data Outsourcing

In Chapter 4 we develop a framework for the privacy of outsourced data. A particular focus of our efforts is to cover practical protocols as well as theoretical constructions, thus making their security comparable. The area of investigation in this chapter is an application of provable security for a single application. In addition to static security, we identify three different privacy goals when the data is queried—data privacy, query privacy, and result privacy. We prove fundamental relations among these concepts. Concretely, data privacy and query privacy are independent concepts, while result privacy is equivalent to both data privacy and query privacy (at the same time). The security of our own construction SEDAWG, a searchable encryption, fits in our privacy framework. We also discuss an implementation of our scheme.

Our results in this chapter are an example of a strength of cryptographic proofs of security. We showed in an abstract model that data privacy and query privacy are independent concepts. Thus, a protocol that hides the contents of interactive queries does not necessarily hide the contents of the underlying database—regardless of the concrete realisation.

1.3.3 Electronic Voting

In Chapter 5 we research modeling and proving coercion resistance in electronic voting—a complex system. Concretely, we investigate recasting one's vote—revoting—as a strategy

to evade coercion. The idea is that a voter cannot be coerced if she can always supersede her vote by a more recent one. To this end, the vote-tallying procedure must ensure that a voter can always plausibly deny having re-cast her vote. We adapt the model for coercion resistance of Juels, Catalano, and Jacobsson [122] to account for revoting. What is more, we give a construction that achieves deniable revoting and prove its coercion resistance.

We derived two requirements for deniable revoting. First, voter authentication must have the property of being inalienable. Otherwise no revoting scheme can offer coercion resistance. This requirement was derived from a model, independently of a concrete realisation, but places a direct constraint on a system design and its implementation. Second, any attempt at coercion must be given up before polls close, so that the voter actually has the chance to revote. This prerequisite is obvious, but explicitly reflected in the model. (We again caution the reader not to trust obviousness.)

1.3.4 Proofs of Security in Security Engineering

We propose amending security engineering processes in the Requirements, Design, and Implementation phase by cryptographic methods.

1. Requirements: To later be able to prove the security of one's design, one must give a precise definition of security. The definition serves as a basis for discussion of the entire engineering team: Does the security definition reflect our intuition of what the system is supposed to achieve?

   The choice of methodology in this phase will directly influence how proofs of security are to be carried out. Does one model security using the Universal Composability (UC) [41] framework (Chapter 3), a game-based model (Chapter 4), or is a combination of a game-based model and the real-vs-ideal-world paradigm (Chapter 5) the best choice?

2. Design: During the design phase, the cryptographer defines the adversarial model and comes up with a proof of security for the system's architecture. Constraints for the implementation arise.

3. Implementation: The implementation constraints that have been found in the design phase must be considered carefully by the programmers while they implement the system.

By precisely defining security before conceiving of a means to achieve it one achieves conceptual clarity. To give a concrete example, instead of requiring data to be encrypted (a concrete method), one should first define the confidentiality of data as a requirement. Then, one can select the correct means to ensure that confidentiality—depending on the definition this could be an encryption, an anonymisation technique or some other method. A proof of security then establishes that the means suits the requirement. The security engineering process already stipulates this approach, but the cryptographic method necessitates it.

The assurance of a cryptographic proof of security might lead one to believe that security testing is now unnecessary—security has already been proven. The contrary is the case. Security testing is an important means to validate the security model. The power of cryptographic proofs of security lies less in an absolute guarantee of security than in a formal analysis on how to obtain a secure construction, and under which conditions such a thing is impossible. A proof can never establish whether its premises are sound in practice.

In the same way the waterfall model is not followed slavishly in practice, a proof of security is not a one-shot process. While devising a proof, the cryptographer might need to adapt the model or to further elaborate the conclusion. We believe that, as it is carried out by humans, the proving process fits organically in a practical system engineering process.

Making sure that the system's implementation adheres to its specification is outside the scope of this work.

A Word of Caution Modeling and proving security is far from trivial. Cryptographic definitions are often complex and frameworks are intricate. We must point out that supplementing an engineering process with cryptographic proofs of security requires the participation of a trained cryptographer—or, at the very least, training the security engineer in cryptographic methodology.

We also gained valuable insights from our implementation efforts. As we explain in Chapter 3.9.2 a naïve optimisation of the trusted packet comparator's implementation would have caused the realisation to deviate from the model. More specifically, it would have given an adversary the ability to compute packet collisions, whereas this is impossible in the model—we simply do not model collisions. Such a deviation is easy to miss for a cryptographic layman. (We speak of a collision when two network packets are treated as identical although they aren't.)

The insight from this finding is that the cryptographer's work is not finished when he has modeled and proven security. He must also be present during the design and implementation of the system and carefully verify the implementation's adherence to the cryptographic model. We believe this is a point of concern worth stressing.

1.4 Structure of This Work

We define notation and preliminaries in Chapter 2. We also give an introduction to the cryptographic proof methods used in this thesis, as well as to the primitives we use.

In Chapter 3 we analyse how to model the security of computer networks. The concrete application we study is a secure combination of firewalls. After discussing related work, we first give a model for firewalls. Then, we analyse the serial concatenation of firewalls, as well as their parallel composition. To model availability, we extend our model and propose a 5-phase paradigm to model networks in the UC framework. We prove an availability guarantee for a quorum of three firewalls.

We develop a framework for the privacy of outsourced data in Chapter 4. Here, we model security with security games. We again first discuss related work. Then, we propose security notions for outsourced data. In extension to privacy in the absence of queries we also define privacy in the presence of queries—data privacy, query privacy, and result privacy. We prove fundamental relations between these notions and give generalisations. We show how to apply our notions to private information retrieval, database outsourcing, and searchable encryption. Our own construction for a searchable encryption also fits in the framework. We define Exact Pattern Matching for Symmetric Searchable Encryption and then present our construction. We also discuss its implementation.

The third application we study is electronic voting in Chapter 5. We extend a pre-existing definition of coercion resistance to cover the revoting strategy. The definition is game-based, but also makes use of the real-vs-ideal paradigm. We explain why revoting must be deniable to be of any use as a strategy to evade coercion. Then, after discussing related work, we show how to achieve deniable revoting. We define correctness, verifiability, and coercion resistance for a voting scheme with revoting. We present our own construction and prove that it is coercion-resistant.

Chapter 6 concludes the work.

2 Preliminaries

In this chapter, we introduce notation, conventions, and fundamental concepts we use throughout the thesis. We further give a brief introduction to the relevant cryptographic proof methodologies. Lastly, we discuss cryptographic building blocks we make use of in our constructions.

Comparison, Assignment, and Random Choice We use the *equal sign* $=$ for comparisons as well as assignments. If not clear from the context, we use the *definition symbol* $:=$ to explicitly indicate an assignment. To draw an item uniformly at random from a set, we use the symbol $\leftarrow$. Similarly, $\leftarrow$ is also used when the output of a randomised algorithm is assigned to a variable. The *bit-wise exclusive-or* is denoted by $\oplus$.

SI Prefixes k, M, G stand for 10^3, 10^6, 10^9, while Ki, Mi, and Gi stand for 2^{10}, 2^{20}, and 2^{30}, respectively.

Lists, Languages, and Strings A *list* is an ordered sequence of values. $L[i]$ is the $(i+1)$th element of list L. $|L|$ denotes the number of its elements. The append symbol $\Leftarrow$ appends an element to a list.

An *alphabet* is a finite and non-empty set. A *string* $S \in \Sigma^*$ is a finite sequence of symbols over an alphabet Σ. Its length is $|S|$. The *empty string* ε has a length of 0. $S[i]$ is the $(i+1)$th character of S for $i \in \mathbb{N}_0$ with $i < |S|$. The *concatenation* operator $\cdot$ concatenates two strings. The *slicing* operation $S[i..j]$ returns the *substring* $S[i] \cdot \ldots \cdot S[j]$ for $0 \leq i \leq j < n$. If $i = j$, $S[i..j] = S[i]$. The set of all substrings of S is denoted by $substrings_S = \{S[i..j] \mid 0 \leq i \leq j < n\} \cup \{\varepsilon\}$. Any substring $S[0..i], i \in \{0, \ldots, |S|-1\}$ of a string is called a *prefix*, $S[i..|S|-1]$ a *suffix* of S.

A *language* is a set of strings.

If x is not a set, string, or list, $|x|$ denotes the length of its binary representation.

Probabilities, Distributions, and Ensembles A *random variable* is a function from a sample space to a set (often $\mathbb{R}$ or $\{0,1\}^*$). A *distribution* assigns a probability to an outcome of a random experiment. The (discrete) *uniform distribution* $\mathcal{U}_n$ assigns the equal probability $\frac{1}{n}$ to all possible events. We denote the probability that the random variable X takes on the value x by $\Pr[X = x]$. Similarly, to denote the probability that some randomised procedure $\mathcal{A}$ outputs x, we write $\Pr[x \leftarrow \mathcal{A}]$ or $\Pr[\mathcal{A} \rightarrow x]$.

Definition (Probability Ensemble [90])**.** Let I be a countable index set. An *ensemble indexed by* I is a sequence of random variables indexed by I. Namely, any $X = \{X_i\}_{i \in I}$, where X_i is a random variable, is an ensemble indexed by I.

Polynomials For the purposes of this thesis, a *polynomial* $P[X]$ of degree n is a finite sum of multiples of powers $\leq n$ of a variable X: $P[X] = \sum_{i=0}^{n} a_i x^i$.

Graphs and Automatons A *directed graph* $G = (V, E)$ is a pair of a set of *vertices* V and a relation E on them, called *edges*: $E \subseteq V^2$, whereas $E \subseteq \{\{u, v\} \mid u, v \in V\}$ in an *undirected graph*. *Adjacency lists* are a representation of directed graphs that is particularly suited for the storage in computer memory: a list of its outgoing edges is stored alongside each vertex.

A *Deterministic Finite Automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times E \to Q$ is a unique transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accept states [186].

Groups A *group* is a pair $(\mathbb{G}, *)$, where $\mathbb{G}$ is a set and the *group operation* $* : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a map with the following properties:

- $\forall a, b, c \in \mathbb{G} : (a * b) * c = a * (b * c)$,
- $\exists e \in \mathbb{G} \forall a \in \mathbb{G} : a * e = e * a = a$, and
- $\forall a \in \mathbb{G} \exists a^{-1} \in \mathbb{G} : a * a^{-1} = a^{-1} * a = e$.

The group operation is often written as the multiplication $\cdot$ (also using the "power" notation a^n). When the group operation is clear from the context, we identify the group by its underlying set $\mathbb{G}$.

A *finite group* is a group where its order $\mathrm{Ord}(\mathbb{G}) := |\mathbb{G}|$ is finite. A group is *cyclic* if there is a $g \in \mathbb{G}$ such that $\mathbb{G} = \{g^n \mid n \in \mathbb{N}\}$. We write $\langle g \rangle = \mathbb{G}$ and call g a *generator* of $\mathbb{G}$. (Angle brackets $\langle\rangle$ also denote the output of an interactive protocol, see Section 2.4.4.)

In this work, we implicitly assume that the group operation is always efficiently computable. Also, we require the existence of an efficient group generator algorithm, i.e. an efficient procedure that outputs description of a group and all relevant parameters.

Asymptotic Notation To express the asymptotic behaviour of terms, we use the "Big-O" (or Landau) notation. We reproduce Goos's [97] definition and its implication:

$$
\begin{aligned}
O(f(n)) &:= \{g(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ s.t. } 0 \leq g(n) \leq c \cdot f(n) \text{ and } n \geq n_0\},\\
o(f(n)) &:= \{g(n) \mid \forall c > 0\, \exists n_0 \in \mathbb{N} \text{ s.t. } 0 \leq g(n) < c \cdot f(n) \text{ and } n \geq n_0\},\\
\Omega(f(n)) &:= \{g(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ s.t. } 0 \leq c \cdot f(n) \leq g(n) \text{ and } n \geq n_0\},\\
\omega((f(n)) &:= \{g(n) \mid \forall c > 0\, \exists n_0 \in \mathbb{N} \text{ s.t. } 0 \leq c \cdot f(n) \leq g(n) \text{ and } n \geq n_0\},\\
\Theta(f(n)) &:= \{g(n) \mid g(n) \in O(f(n)) \text{ and } g(n) \in \Omega(f(n))\}.
\end{aligned}
$$

Instead of $f(n) \in O(g(n))$ the notation $f(n) = O(g(n))$ is common, though technically it is not correct.

2.1 Negligibility

The concept of negligibility is important in cryptography when one imposes computational limits on the adversary. Consider for example an encryption scheme. If one requires it impossible for any adversary to extract the plaintext from its encryption, there is no secure encryption scheme (with the exception of schemes with perfect privacy [131, 182]): An adversary can always guess a decryption key and try it. It is possible he guessed the key correctly and thus obtains the plaintext. For practical applications, guessing keys is not a reasonable strategy to attack an encryption scheme with sufficiently many keys. We need a formalism to handle negligible probabilities.

We call a function f *negligible* if for every polynomial p there exists an N such that for all integers $k > N$ it holds that $f(k) < \frac{1}{p(k)}$. An equivalent definition is that a function μ is negligible if $\forall c \in \mathbb{N} \exists N \in \mathbb{N}$ such that $\forall k \geq N : \mu(k) \leq k^{-c}$.

The sum of two negligible functions is negligible and the product of any positive polynomial and a negligible function is negligible [131]. Intuitively, if an adversary's success probability is negligible in a value called the security parameter, we call a scheme secure.

2.2 Computational Complexity

Formally, a *Turing Machine (TM)* is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q is the finite set of states, Σ is the input alphabet, Γ is the tape alphabet, δ is the transition function, $q_0 \in Q$ is the start state, $q_{\text{accept}} \in Q$ is the accept state, and q_{reject} is the reject state [186]. We say a machine *accepts* if it reaches the accept state, and that it *rejects* if it reaches the reject state. We call a language L *decidable* if there is a Turing Machine that accepts all $w \in L$ and that rejects all $w' \notin L$. For a thorough introduction, we refer to Sipser [186].

To give a capability in a "black-box" manner to a TM, one can use an *oracle*. An oracle is a black-box entity with an interface that can be called just like a library function in a programming language. We call a TM that has access to one or more oracles an *Oracle TM*. In shorthand notation, to provide an oracle $\mathcal{O}$ to a machine $\mathcal{A}$, we sometimes write $\mathcal{A}^{\mathcal{O}}$. Similarly, to provide a function f with the first parameter "open" and the second one fixed to a variable K, we write $\mathcal{A}^{f(\cdot,K)}$.

In our constructions, we generally limit ourselves to *deterministic* TMs, i.e. machines with a unique transition function. The *time complexity* (or *running time*) of a deterministic TM is the maximum number of steps that the TM uses on a given input (or a given class of inputs of a given length). The running time of a nondeterministic TM is the maximum number of steps that it uses on any branch of its computation [186]. We use the terms *Turing Machine*, *machine*, *algorithm*, *program*, and *procedure* interchangeably. A *protocol* is a finite set of interacting TMs.

If not stated otherwise, the computation model throughout this work is the *Probabilistic Polynomial Time (PPT)* model—machines have access to a (private and uniformly distributed) source of randomness and their running time is bound by a fixed polynomial

in the size of their input. We call thus restricted machines efficient. The class of PPT machines is closed under composition, i.e. a PPT machine running a PPT subroutine is still PPT.

In security definitions, the value relative to which running time is measured is often called the *security parameter*. All cryptographic procedures must run efficient relative to the security parameter while breaking the security of a scheme should require super-polynomial effort in the security parameter. A popular example to illustrate this concept is encryption: Generating a key of length k, encrypting using such a key, and decryption should all be efficiently computable. On the other hand, breaking the encryption should be infeasible using $p(k)$ computation steps, for any polynomial p. Conversely, the probability that any efficient adversary breaks the encryption should be negligible. To explicitly give an algorithm a running time of k steps, we use the standard notation 1^k to indicate an extra input of a sequence of 1s of length k.

Definition ($\mathcal{P}$ [186])**.** $\mathcal{P}$ is the class of languages that are decidable in polynomial time on a deterministic single-tape TM.

Definition ($\mathcal{NP}$ [186])**.** $\mathcal{NP}$ is the class of languages that are decidable in polynomial time on a nondeterministic single-tape TM.

2.3 Cryptographic Proofs of Security

In this section we review concepts and techniques for cryptographic proofs of security that are relevant for this thesis. We point out that there are more, but covering those is outside the scope of this work. We refer to literature for a comprehensive overview [90, 91, 131]. For the purposes of this thesis, we restrict our security considerations to computational security. The following exposition reflects this. We point out that there are also the concepts of statistical and perfect security. They are more strict, and harder to achieve. In a nutshell, they restrict the adversarial model less. For practical purposes, computational security with an efficiency restriction on the adversary is widely considered adequate. (An area of research where this restriction is *not* considered adequate is "Post-Quantum Cryptography".)

2.3.1 Game-Based Security

The idea behind game-based security definitions is to play a "game" with the adversary. (These games are often called security experiments.) If the adversary is successful, i.e. wins the game, the result of the game is 1, otherwise it is 0. One defines the advantage of an adversary as his success probability minus the probability of "trivially" winning the game. The security property holds, finally, if no adversary has a non-negligible advantage.

More technically, Dodis et al. [70] define a security game as an interaction between a prover and a challenger:

> A game is defined by two PPT Oracle-TMs, a prover A and a challenger C, which have a common communication tape over which they can exchange

messages. The challenger finally output a decision bit. We say that A wins the game if this bit is 1 and denote this event by $\langle A, C\rangle \to 1$.

This notion captures the essential requirement that the "game" itself be PPT. This is important so that the thus defined security notion lends itself to a reduction argument (see below).

As an example, the Existential Unforgeability under Chosen-Message Attacks (EUF-CMA) security notion is defined as a game in which the adversary tries to come up with a valid signature of a message of his choosing (Section 2.4.2).

Indistinguishability

In modern cryptography, one often makes use of the concept of indistinguishability. This concept is very handy when security is about keeping information private. The idea is that the adversary may not even extract one single bit of the secret in question, regardless of what the secret is. If he were to extract one bit from any secret, he could use that bit to distinguish two secrets of his choosing.

Definition (Polynomial-Time Indistinguishability [90])**.** Two ensembles, $X := \{X_n\}_{n\in\mathbb{N}}$ and $Y := \{Y_n\}_{n\in\mathbb{N}}$ are *indistinguishable in polynomial time* if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large n's,

$$\Pr\left[D(X_n, 1^n) = 1\right] - \Pr\left[D(Y_n, 1^n) = 1\right] < \frac{1}{p(n)}$$

We often say *computational indistinguishability* instead of indistinguishability in polynomial time.

For example, in the Private Information Retrieval (PIR) scenario (see also Section 4.5.1), a server stores a database $d \in \{0,1\}^n$. The client wants to retrieve one bit $d[i], i \in \{0,\ldots,n-1\}$ from the database without the server learning any information about i. The formal PIR privacy requirement is that for any database d and any database indices i, j the output of any distinguisher is independent of whether $d[i]$ or $d[j]$ was retrieved.

In game-based notions, the concept is expressed differently. There, the adversary chooses two candidate secrets. The game then secretly flips a coin $b \leftarrow \{0,1\}$ and performs the procedure in question (e.g. an encryption) on secret b. The challenge of the adversary is to correctly guess b based on his observations. A typical example for this concept is the Indistinguishability under Chosen-Plaintext Attack (IND-CPA) notion for encryptions (Section 2.4.1). There, the adversary submits two plaintexts m_0, m_1 of his choosing and receives an encryption of m_b. His challenge is to correctly guess b.

Indistinguishability is a transitive relation, i.e. if A is indistinguishable from B and B is indistinguishable from C, then A is also indistinguishable from C.

Reductions

Proofs of perfect (i.e. unconditional) security are only possible for encryptions that use a key as long as the message itself (or longer), the One-Time-Pad [131, 182]. For all other

constructions, proofs of security are relative to assumptions that are believed to hold. One example is the Decisional Diffie-Hellman (DDH) assumption, which is that, given random elements g^a, g^b, g^c of a cyclic group $\mathbb{G} = \langle g \rangle$, it is hard to tell whether $g^c = g^{ab}$.

To prove the security of a construction relative to an assumption, one usually uses a reduction argument. One devises an efficient adversary against the assumption, using an adversary against the security of the construction in question. (One may only use the adversary in a black-box manner, without relying on any knowledge about its inner workings, lest the reduction work only for certain adversaries, not all.) To this end, the reduction must perfectly emulate the security game the adversary expects to run in. The typical challenge when contriving a reduction is to smartly embed the challenge from the assumption's security game into the emulated security game. This way, one can then extract the correct guess bit b from the adversary. See Figure 2.1 for an illustration.

The security of the scheme is proven by contraposition then: If there is a successful adversary against the security of the construction, there is also one against the assumption, which is a contradiction. Hence, there is no successful adversary against the security of the construction if the assumption holds.

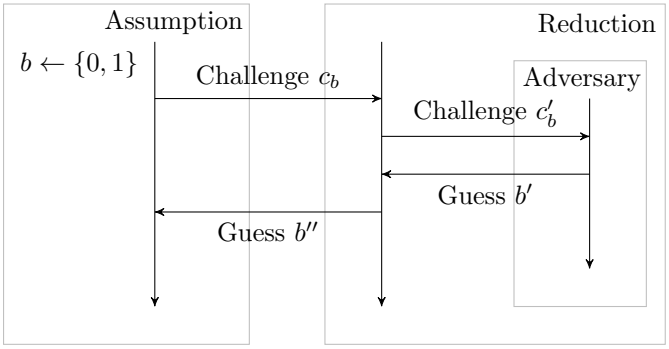

Figure 2.1: A reduction argument. An efficient adversary against the assumption is devised, making black-box use of an adversary against the security of the construction in question. The challenge for the reduction is embedded into the challenge for the adversary, thus making it possible to leverage the adversary's advantage.

Game-Hopping and Transformations One often-used proof technique is the game transformation or game-hopping technique. The basic idea is not to prove security in one single step, but in several steps. In each step, the security game is modified. For each step one shows that, for the adversary, the modified game is indistinguishable from the former game. By the transitivity of indistinguishability, the first game is indistinguishable from the last in the series of transformations. The last game in a series of transformations then has a special property, for example the adversaries input is independent of the challenge bit. The adversary then has no advantage over a blind guess. A particular application of this technique is to show security relative to several assumptions: in each step, a different assumption is used to modify the game. To be successful in the resulting

security game, the adversary must invalidate at least one of the assumptions used in the transformations.

2.3.2 Real-vs-Ideal Paradigm

In some applications, security cannot be defined in an absolute sense, as even an optimal realisation is bound to have an inherent weakness. In the case of elections (Chapter 5) for example, the mere release of the election result leaks information about the voters' choices. This "information leak" is unavoidable. To factor out the advantage an adversary has from inspecting the election result, one can use the real-vs-ideal paradigm when defining security: in addition to the "real" game, one defines the "ideal" game where the adversary only learns the election result. His total advantage is then measured as the difference between his advantage in the former versus his advantage in the latter game.

Also, one models Multi-Party Computation (Section 2.4.5) as a group of parties jointly computing an ideal functionality. The paradigm also finds an application in the Universal Composability (UC) [41] framework.

2.3.3 Universally Composable Security

In this chapter we give a brief review of the Universal Composability (UC) [41] framework. It is a tool for proving the security of multi-party protocols by comparing their execution with an idealised version of the protocol. Because of its composition theorem, Universal Composability (UC) [41]-secure protocols can be designed in a modular fashion. The framework allows us to model a system of firewalls as a protocol execution and underspecify the concrete functionality of the participating firewalls and only state what an ideal execution would look like.

This subsection has already been partially published as a part of a conference paper [3].

In the UC framework, participants in a protocol are modeled as Interactive Turing Machines (ITMs). Since there are different definitions of ITMs in literature, we briefly summarise the definition given by Canetti [41, 42].

Definition (Interactive Turing Machine)**.** An *Interactive Turing Machine (ITM)* is a multi-tape TM with the following tapes. A tape is *externally writeable* (EW), if every other machine can write (append) to it.

- an *identity tape* (EW)
- a *security parameter tape* (EW)
- an *input tape* (EW)
- a *communication tape* (EW)
- an *output tape*
- a *working tape*

- a *subprocess tape*

We call an ITM *probabilistic*, if it, in addition, has a *random tape*, which contains a random bitstring of a specific distribution.

A protocol π in the UC framework is executed in the context of two additional ITMs: the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. The adversary represents the party that attacks the protocol, the environment represents the perception of the execution from an outside point of view.

There are some general restrictions concerning the communication among the participating parties: the adversary and the environment are allowed to communicate freely. In addition, the adversary is allowed to write to the communication tapes of every participant and the environment can write to the input tapes and receive outputs of the parties of the protocol. This captures the notion that the environment represents the external input to the protocol but will not interfere with the protocol itself.

In Chapter 3 we realise protocol execution in the $\mathcal{F}$-hybrid model of computation: Parties do not communicate directly, but use a functionality $\mathcal{F}$ as a proxy, which is modeled as another ITM. $\mathcal{F}$ also communicates with the adversary. The exact behaviour of $\mathcal{F}$ is specified in advance and must reflect the exact nature of the communication link. For example, $\mathcal{F}$ might be set up such that $\mathcal{A}$ can intercept and send arbitrary network messages. We will setup $\mathcal{F}$ so that it reflects our network architecture: The adversary cannot intercept all communication or inject messages into the network at will. It is only possible to send messages on established links as specified in the architecture diagram. However, the adversary can send a special message to other parties: the *corruption message*. If a party receives a corruption message, it stops executing its own program and instead gives complete control of its functions to the adversary. This includes disclosing its internal state.

A protocol is executed in turns. If an ITM is activated, it can perform computation and write to a tape of any other ITM based on the aforementioned restrictions. Then its turn ends. If an ITM receives input on one of its tapes, it is the next to be activated. The first ITM to be activated is the environment $\mathcal{Z}$.

The output of the protocol is the output of $\mathcal{Z}$ and we assume, without loss of generality, that it consists of one bit. The distribution of all outputs of $\mathcal{Z}$ is a probability ensemble based on the two parameters z (the input) and k (the security parameter).

Definition (Ensemble of a protocol execution)**.** We denote the random variable which describes the execution of a protocol π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, input z, security parameter k as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$. The probability ensemble $\{\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in N,z\in\{0,1\}^*}$ is denoted as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

The security of a protocol execution in the UC framework is based on a comparison with an execution of an idealised version of the protocol: the *ideal protocol*. The ideal protocol contains the *ideal functionality* $\mathcal{F}_{\mathsf{ideal}}$ which completely realises the properties of the analysed protocol. In the ideal protocol, all parties only act as dummies which directly give their input to the ideal functionality and receive back their output without

performing any computation themselves. The ideal functionality may communicate with the adversary in order to model the allowed influence $\mathcal{A}$ has. Since the only instance which performs computations is $\mathcal{F}_{\mathsf{ideal}}$, which is ideal by definition, the whole protocol execution is ideal and thus secure.

In Chapter 3 we model the ideal functionality as the same firewall network, but without the compromised firewall. For example, when we combine two firewalls, of which one may be compromised, the ideal model we compare our protocol to is just one uncompromised firewall.

Definition (Ideal protocol)**.** Let $\mathcal{F}_{\mathsf{ideal}}$ be an ideal functionality. Then, the ideal protocol which realises $\mathcal{F}_{\mathsf{ideal}}$ is denoted as $\mathsf{IDEAL}_{\mathcal{F}}$.

Informally, a protocol π is UC secure if, for every adversary $\mathcal{A}$ and every environment $\mathcal{Z}$, $\mathcal{Z}$ can not distinguish if it is interacting with π or with the ideal protocol implementing π. To capture that notion formally, we define indistinguishability.

Definition (Indistinguishablity of probability ensembles)**.** Two binary ensembles X and Y are *indistinguishable* ($X \approx Y$), if $\forall c, d \in N\ \exists k_0 \in N$, so that for all $k > k_0$ and all $a \in \cup_{\kappa \leq k^d}\{0,1\}^\kappa$ holds:

$$|\Pr(X(k,a) = 1) - \Pr(Y(k,a) = 1)| < k^{-c}$$

Based on that notion, we now formalise the indistinguishability of two protocols in the UC framework. In place of the adversary we put a "simulated adversary" $\mathcal{S}$, called the simulator. The simulator's job is to simulate the presence of $\mathcal{A}$ to the environment, so that it may not distinguish the real protocol execution from the idealised version. The security notion requires that there is a successful simulator for every adversary.

Definition (UC emulates)**.** Let π and ϕ be two protocols. Then π UC emulates the protocol ϕ, if $\forall \mathcal{A}\ \exists \mathcal{S}$, so that $\forall \mathcal{Z}$ holds:

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$$

We can now formally state when a protocol *realises* a functionality.

Definition (UC realises)**.** A protocol π (securely) *UC realises* an ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, if π UC emulates the corresponding ideal protocol $\mathsf{IDEAL}_{\mathcal{F}}$.

If a protocol π realises a given ideal functionality, then we say π is *UC secure*.

The UC framework is a powerful instrument for analysing the security of protocols because it provides a composition theorem. Informally speaking, the composition theorem states that if π securely realises an ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, one can use π instead of $\mathcal{F}_{\mathsf{ideal}}$ in other protocols without compromising security.

In the following definition, $\pi/\mathcal{F}$ is shorthand for replacing all instances of $\mathcal{F}$ with instances of π.

Theorem (Composition Theorem, realising functionalities [42])**.** *Let $\mathcal{F}$, $\mathcal{G}$ be ideal functionalities such that $\mathcal{F}$ is PPT. Let ρ be a subroutine respecting protocol that UC-realises $\mathcal{G}$, and let π be a subroutine respecting protocol that securely realises $\mathcal{F}$. Then the composed protocol $\rho^{\pi/\mathcal{F}}$ securely realises $\mathcal{G}$.*

Synchronous UC

The UC framework is asynchronous—one machine runs at a time. This way, one execution of a protocol is unique. On the other hand, the passing of time cannot easily be modeled in the framework. In Section 3.7 we study an extension proposed by Katz et al. [132] that allows the expression of synchrony without modifying the framework.

2.4 Cryptographic Primitives

We make use of several cryptographic building blocks in the constructions in this work. In this section we define them and their security, where appropriate.

2.4.1 Encryption

Encryption is a method to ensure the confidentiality of data. A method for encrypting data is sound only when the original data can easily be recovered by an authorised user. More formally, an encryption scheme is a tuple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where Gen is a key generation algorithm, Enc an encryption procedure, and Dec a decryption algorithm. The exact definition of an encryption scheme depends on the kind of encryption to be defined.

Private-Key Encryption

In a private-key encryption scheme, the same key is used for encryption and decryption. We paraphrase Katz and Lindell's definition [131, Definition 3.7] here.

Definition (Private-Key Encryption Scheme)**.** A *private-key encryption scheme* is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. The *key-generation algorithm* Gen takes as input the security parameter 1^k and outputs a key K,
2. the *encryption algorithm* Enc takes as input a plaintext message $m \in \{0,1\}^*$ and a key K, and outputs a ciphertext c, and
3. the *decryption algorithm* Dec takes as input a ciphertext c and a key K, and outputs a message m.

It is required that for every k, every key K output by $\mathsf{Gen}(1^k)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{Dec}(\mathsf{Enc}(m, K), K) = m$.

Security Game (The (private-key) CPA indistinguishability experiment $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(k)$ [131])**.**

1. A key K is generated by running $\mathsf{Gen}(1^k)$.
2. The adversary $\mathcal{A}$ is given input 1^k and oracle access to $\mathsf{Enc}(\cdot, K)$, and outputs a pair of messages m_0, m_1 of the same length.

3. A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}(m_b, K)$ is computed and given to $\mathcal{A}$.

4. The adversary $\mathcal{A}$ continues to have oracle access to $\mathsf{Enc}(\cdot, K)$, and outputs a bit b'.

5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition (IND-CPA security for private-key encryption [131])**.** A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *indistinguishable encryptions under a chosen-plaintext attack* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(k) = 1\right] \leq \frac{1}{2} + negl(k),$$

where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment.

Block-Ciphers and Operation Modes One important class of private-key encryption schemes are block-ciphers. A block-cipher encrypts a block of a fixed length. Together with an operation mode, it realises a private-key encryption. One popular example is the Advanced Encryption Standard (AES) [80].

The idea behind block-ciphers is to implement a keyed permutation on a fixed-size block of bits, e.g. 128. (We call a mapping $f : A \to A$ a permutation on A if it is a bijection.)

Definition (Strong pseudorandom permutation [131])**.** Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, keyed permutation. We say that F is a *strong pseudorandom permutation* if for all PPT distinguishers D, there exists a negligible function *negl* such that:

$$\left|\Pr\left[D^{F_K(\cdot), F_K^{-1}(\cdot)}(1^k) = 1\right] - \Pr\left[D^{f(\cdot), f^{-1}(\cdot)}(1^k) = 1\right]\right| \leq negl(n),$$

where $k \leftarrow \{0,\}^n$ is chosen uniformly at random and f is chosen uniformly at random from the set of permutations on n-bit strings.

We require that F and F^{-1} are efficiently computable.

While a block-cipher only encrypts blocks of a fixed length, together with an operation mode documents of arbitrary length can be encrypted. The simplest construction is the Electronic Code Book (ECB) mode. Here, the document is split in n-bit blocks and each block is encrypted differently. That is, a message m is split into blocks $m_1 \cdot m_2 \cdot \ldots m_l = m$ and the ciphertext is computed as $c = c_1 \cdot c_2 \ldots c_l$ with $c_i = F(m_i)$. The ECB mode is not IND-CPA secure.

The Cipher Block Chaining (CBC) mode works by including the encryption of the previous block in the encryption. Concretely, $c_i = F(m_i \oplus c_{i-1})$. (For c_1 a random initialisation vector IV is chosen.) If F is a pseudorandom permutation then the Cipher Block Chaining mode realises a CPA-secure encryption [131].

Public-Key Encryption

In a public-key encryption scheme, a public key is used to encrypt, while a private (or secret) key is required to decrypt. As we did in the case of private-key encryption, we also paraphrase Katz and Lindell's definition [131, Definition 10.1] here.

Definition (Public-Key Encryption Scheme)**.** A *public-key encryption scheme* is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. The *key-generation algorithm* Gen takes as input the security parameter 1^k and outputs a pair of keys (pk, sk); a public key and a secret key.

2. The *encryption algorithm* Enc takes as input a plaintext message m (from some underlying plaintext space) and a public key pk, and outputs a ciphertext c, and

3. the *decryption algorithm* Dec takes as input a ciphertext c and a private key sk, and outputs a message m or a symbol $\perp$, indicating failure.

It is required that for every k, every key pair (pk, sk) output by $\mathsf{Gen}(1^k)$, and every appropriate m, the probability that $\mathsf{Dec}(\mathsf{Enc}(m, pk), sk) \neq m$ is negligible.

Security Game. The (public-key) CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(k)$ [131].

1. $\mathsf{Gen}(1^k)$ is run to obtain keys (pk, sk).

2. Adversary $\mathcal{A}$ is given pk as well as oracle access to $\mathsf{Enc}(\cdot, pk)$. The adversary outputs a pair of messages m_0, m_1 of the same length.

3. A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}(m_b, pk)$ is computed and given to $\mathcal{A}$.

4. $\mathcal{A}$ continues to have access to $\mathsf{Enc}(\cdot, pk)$, and outputs a bit b'.

5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition (IND-CPA for public-key encryption [131])**.** A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *indistinguishable encryptions under a chosen-plaintext attack* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that:

$$\Pr\left[\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

The Elgamal Encryption Scheme

Definition (The Decisional Diffie-Hellman problem [131])**.** Fix a group generator algorithm $\mathcal{G}$. We say that *the Decisional Diffie-Hellman (DDH) problem is hard relative to* $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function *negl* such that

$$|\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(k),$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^k)$ outputs $(\mathbb{G}, q, g)$, and then random $x, y, z \in \mathbb{Z}_q$ are chosen.

Definition (The Elgamal encryption scheme [76, 131])**.** Let $\mathcal{G}$ be a PPT algorithm that, on input 1^k, outputs a description of a cyclic group $\mathbb{G}$, its order q, and a generator g. Define a public-key encryption scheme as follows:

- Gen: on input 1^k run $\mathcal{G}(1^k)$ to obtain $(\mathbb{G}, q, g)$. Then choose a random $x \leftarrow \mathbb{Z}_q$ and compute $h := g^x$. The public key is $(\mathbb{G}, q, g, h)$ and the private key is $(\mathbb{G}, q, g, x)$.
- Enc: on input a public key $pk = (\mathbb{G}, q, g, h)$ and a message $m \in \mathbb{G}$, choose a random $y \leftarrow \mathbb{Z}_q$ and output the ciphertext

$$(g^y, h^y \cdot m).$$

- Dec: on input a private key $sk = (\mathbb{G}, q, g, x)$ and a ciphertext (c_1, c_2), output

$$m := c_2 \cdot c_1^{-x}.$$

Theorem (Elgamal is IND-CPA secure [131])**.** *If the DDH problem is hard relative to* $\mathcal{G}$*, then the Elgamal encryption scheme has indistinguishable encryptions under a Chosen-Plaintext Attack.*

Homomorphic Encryption We say an encryption is homomorphic with respect to $\cdot$, if for any two messages m_1, m_2 and any encryption key pk it holds that

$$\mathsf{Enc}(m_1, pk) \cdot \mathsf{Enc}(m_2, pk) = \mathsf{Enc}(m_1 \cdot m_2, pk).$$

If we make use of this property we write the encryption operation as $\mathsf{Enc}_{\mathsf{hom}}$ to make this obvious.

The encryption from the Elgamal scheme (Definition 2.4.1) has this property if one defines $\cdot$ as the piecewise multiplication. Observe that

$$\begin{aligned}\mathsf{Enc}_{\mathsf{hom}}(m_1, (\mathbb{G}, q, g, h)) \cdot \mathsf{Enc}_{\mathsf{hom}}(m_2, (\mathbb{G}, q, g, h)) &= (g^{y_1}, h^{y_1} \cdot m_1) \cdot (g^{y_2}, h^{y_2} \cdot m_2) \\ &= (g^{y_1} \cdot g^{y_2}, h^{y_1} \cdot m_1 \cdot h^{y_2} \cdot m_2) \\ &= (g^{y_1+y_2}, h^{y_1+y_2} \cdot m_1 \cdot m_2) \\ &= \mathsf{Enc}_{\mathsf{hom}}(m_1 \cdot m_2, (\mathbb{G}, q, g, h)).\end{aligned}$$

2.4.2 Digital Signatures

A digital signature is the digital equivalent of a handwritten signature, but also serves to protect the integrity of data—whereas a contract on paper can inconspicuously be modified after it has been signed, a digital signature is valid only if the signed data is not modified afterwards.

Definition (Signature Scheme [131])**.** A *signature scheme* is a tuple of three PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ satisfying the following:

1. The *key-generation algorithm* Gen takes as input a security parameter 1^k and outputs a pair of keys (pk, sk).

2. The *signing algorithm* Sign takes as input a message $m \in \{0,1\}^*$ and a private key sk. It outputs a signature $\sigma \leftarrow \mathsf{Sign}(m, sk)$.

3. The deterministic *verification algorithm* Verify takes as input a message m, a signature σ, and a public key pk. It outputs a bit $b := \mathsf{Verify}(m, \sigma, pk)$, with $b = 1$ meaning *valid*, and $b = 0$ meaning *invalid*.

It is required that for every k, every (pk, sk) output by $\mathsf{Gen}(1^k)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{Verify}(m, \mathsf{Sign}(m, sk), pk) = 1$. If $\mathsf{Verify}(m, \sigma, pk) = 1$ we say σ is a valid signature on message m with respect to public key pk.

Existential Unforgeability under Chosen-Message Attacks

Security Game. The signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(k)$ [131].

1. $\mathsf{Gen}(1^k)$ is run to obtain keys (pk, sk).

2. Adversary $\mathcal{A}$ is given pk and oracle access to $\mathsf{Sign}(\cdot, sk)$. The adversary outputs (m, σ). Let $\mathcal{Q}$ denote the set of messages whose signatures were requested by $\mathcal{A}$ during its execution.

3. The output of the experiment is defined to be 1 if and only if $\mathsf{Verify}(m, \sigma, pk)$ and $m \notin \mathcal{Q}$.

Definition (Existential Unforgeability under Chosen-Message Attacks (EUF-CMA) [131])**.** A signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is *existentially unforgeable under an adaptive chosen-message attack* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that

$$\Pr\left[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(k) = 1\right] \neq negl(k).$$

2.4.3 Cryptographic Hashes

The idea behind cryptographic hashing is to compute for a string x a value that is strictly shorter than x. The security intuition is then that no efficient adversary shall be able to find an x' such that the hash values match. Although constructions that are used in practice, such as the SHA-1 hash [79], are not formally proven secure, the security notions cited below give a good intuition as to what a cryptographic hash algorithm is supposed to achieve.

Definition (Hash function [131])**.** A *hash function* is a pair of PPT algorithms (Gen, H) fulfilling the following:

- Gen is a probabilistic algorithm which takes as input a security parameter 1^k and outputs a key s. We assume that 1^n is is implicit in s.
- There exists a polynomial l such that H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{l(k)}$ (where k is the value of the security parameter implicit in s.)

If H^s is defined only for inputs $x \in \{0,1\}^{l'(k)}$ and $l'(n) > l(n)$, then we say that (Gen, H) is a *fixed-length* hash function for inputs of length $l'(n)$.

Security Game. The collision-finding experiment $\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(k)$ [131].

1. A key s is generated by running $\mathsf{Gen}(1^k)$.
2. The adversary $\mathcal{A}$ is given s and outputs x, x'. (If Π is a fixed-length hash function for inputs of length $l'(k)$ then we require $x, x' \in \{0,1\}^{l'(k)}$.)
3. The output of the experiment is defined to be 1 is and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such a case we say that $\mathcal{A}$ has found a collision.

Definition (Collision resistance [131])**.** A hash function $\Pi = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{Hash\text{-}coll}_{A,\Pi}(k) = 1\right] \leq negl(k).$$

2.4.4 Zero-Knowledge

When one wants to prove the possession of a capability or the knowledge of a secret, one can demonstrate or reveal it. But if, on the other hand, one wants to give a credible proof without revealing any information besides one's capability or knowledge, one can employ Zero-Knowledge (ZK) proofs. Zero-Knowledge proofs make use of the simulator concept (see also the UC framework, Section 2.3.3): If any knowledge gained by the verifier can also be gained without interacting with the prover at all, the proof is Zero-Knowledge.

Definition (Interactive Proof System [90])**.** A pair of interactive machines (P, V) is called an *interactive proof system for a language* L if machine V is PPT and the following two conditions hold:

- Completeness: For every $x \in L, \Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$
- Soundness: For every $x \notin L$ and every interactive machine B, $\Pr[\langle B, V\rangle(x) = 1] \leq \frac{1}{3}$

Definition (Computational Zero-Knowledge [90])**.** Let (P, V) be an interactive proof system for some language L. We say that (P, V) is *computational zero-knowledge* if for every PPT interactive machine V^* there exists a PPT algorithm M^* such that the following ensembles are computationally indistinguishable:

- $\{\langle P, V^*\rangle(x)\}_{x\in L}$ (i.e., the output of the interactive machine V^* after it interacts with the interactive machine P on common input x)
- $\{M^*(x)\}_{x\in L}$ (i.e., the output of machine M^* on input x)

Machine M^* is called a *simulator* for the interaction of V^* with P.

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) := \{s \mid (x, s) \in R\}$ and $L_R := \{x \mid \exists s \text{ s.t. } (x, s) \in R\}$. If $(x, s) \in R$, then we call s a *solution* for x. We say that R is *polynomially bounded* if there exists a polynomial p such that $|s| \leq p(|x|)$ for all $(x, s) \in R$. We say that R is an $\mathcal{NP}$-relation if R is polynomially bounded and, in addition there exists a polynomial-time algorithm for deciding membership in R.

Definition (System for Proofs of Knowledge [90])**.** Let R be a binary relation and $\kappa : \mathbb{N} \to [0, 1]$. We say that an interactive function V is a *knowledge verifier for the relation* R with *knowledge error* κ if the following two conditions hold:

- Non-triviality: There exists an interactive machine P such that for every $(x, y) \in R$ all possible interactions of V with P on common input x and auxiliary input y are accepting.
- Validity (with error κ): There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K such that for every interactive function P, every $x \in L_R$, and every $y, r \in \{0,1\}^*$, machine K satisfies the following condition:

  Denote by $p(x, y, r)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P_{x,y,r}$. If $p(x, y, r) > \kappa(|x|)$, then, on input x and with access to oracle $P_{x,y,r}$, machine K outputs a solution $s \in R(x)$ within an expected number of steps bounded by

  $$\frac{q(|x|)}{p(x, y, r) - \kappa(|x|)}$$

  The oracle machine K is called a *universal knowledge extractor*.

When $\kappa(\cdot)$ is identically zero, we simply say that V is a *knowledge verifier for the relation* R. An interactive pair (P, V) such that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a *system for proofs of knowledge for the relation* R.

Definition (Non-Interactive Proof System [90])**.** A pair of probabilistic machines is called a *non-interactive proof system for a language* L if V is polynomial time and the following two conditions hold:

- Completeness: For every $x \in L$,

$$\Pr\left[V(x, R, P(x, R)) = 1\right] \geq \frac{2}{3}$$

  where R is a random variable uniformly distributed in $\{0,1\}^{\mathrm{poly}(|x|)}$.

- Soundness: For every $x \notin L$ and every algorithm B,

$$\Pr\left[V(x, R, B(x, R)) = 1\right] \leq \frac{1}{3}$$

  where R is a random variable uniformly distributed in $\{0,1\}^{\mathrm{poly}(|x|)}$.

The uniformly chosen string R is called the *common reference string.*

Definition (Non-Interactive Zero-Knowledge [90])**.** A non-interactive proof system (P, V) for a language L is *zero-knowledge* if there exists a polynomial p and a PPT algorithm M such that the ensembles $\{x, U_{p(|x|)}, P(x, U_{p(|x|)}))\}_{x \in L}$ and $\{M(x)\}_{x \in L}$ are computationally indistinguishable, where U_m is random variable uniformly distributed over $\{0,1\}^m$.

Theorem ([90])**.** *Assuming the existence of one-way permutations, each language in* $\mathcal{NP}$ *has a zero-knowledge non-interactive proof system. Furthermore, assuming the existence of families of trapdoor permutations, each language in* $\mathcal{NP}$ *has a zero-knowledge non-interactive proof system in which the prover can be implemented by a PPT machine that gets an* $\mathcal{NP}$*-witness as auxiliary input.*

2.4.5 Multi-Party Computation

In some cryptographic applications one faces the problem that no single party can be trusted to compute a certain function. The solution is to compute the function as a Multi-Party Computation (MPC). In our construction in Chapter 5 for example, we trust no single server to hold the decryption key for ballots. Instead, a multitude of servers jointly hold it—each server has a share of the key.

The security of an MPC is modeled after the real-vs-ideal paradigm. A multi-party protocol emulates a single party that computes the function in question. The UC framework (Section 2.3.3) draws inspiration from this idea.

We cite a well-known general possibility result here. Prior to that, we sketch Shamir's secret sharing scheme [181]. It is a so-called (k, n) threshold scheme, where a secret number D is divided into n pieces so that k or more pieces suffice to reconstruct D. The scheme works as follows: Pick a random polynomial $q(x) = D + a_1 x + \cdots + a_{k-1} x^{k-1}$ of degree $k - 1$. Then, evaluate $D_i = q(i)$ for $i \in \{1, \ldots, n\}$. These are the shares. By interpolation, using at least k shares, one can again unequivocally find q and compute $D = q(0)$. Using such a secret sharing scheme, one can split input to several parties who then jointly compute the desired function.

Theorem ([91])**.** *Assuming the existence of enhanced trapdoor permutations, general secure multi-party computation is possible in the following three models:*

1. *Passive adversary, for any number of dishonest parties.*
2. *Active adversary that may control only a strict minority of the parties.*
3. *Active adversary, controlling any number of bad parties, provided that suspension of execution is not considered a violation of security.*

In all these cases, the adversary is computationally bounded and non-adaptive. On the other hand, the adversary may tap the communication lines between honest parties (i.e., we do not assume the existence of private channels). The results for active adversaries assume a broadcast channel.

3 Computer Networks

> If control of international trade were to become an important instrument of government policy, then any international communications network involved with industrial or commercial computer-private systems will need the best protection that can be provided.
>
> *(Willis H. Ware,* Security and Privacy in Computer Systems, 1967*)*

Information Technology (IT) systems are at the heart of most automated systems today. Not only cars and airplanes rely on networked computer systems, but also factories, water supply plants, and nuclear facilities. At the same time, IT systems have never faced more serious threats—national and foreign secret services, criminal organisations, and even corporate hacker groups threaten the integrity and availability of services society relies on.

The aim of research in provable security is to construct IT systems that are invulnerable to attack. Because one faces an intelligent adversary when constructing IT systems it is not sufficient to test for known vulnerabilities before shipping a security solution. One needs a method for defending against previously-unknown attacks. To achieve this goal, formal security models are used. However, since proofs in these models are naturally very technical, they only exist for individual system components like secure channels [85] or simple protocols such as fair coin tosses [28]. However, since modern IT networks are very complex, they are usually not proven secure in a formal security model but only empirically tested for known security weaknesses. This constitutes a significant gap between systems which are proven to be secure and those which are actually in use.

This gap can be somewhat narrowed by security notions that offer *composability*: When the cryptographic framework ensures that the composition of secure components yields a secure system, security analysis can be broken down into analysing components with manageable complexity. Specifically, when one analyses computer network components, the secure composability property allows to focus on one sub-network without specifying the rest of the network. (One essentially proves the security of the component *for any* surrounding network.)

The protection of computer networks against attackers from the Internet is crucial for companies to protect their intellectual property. Network firewalls are used to shield networks against threats from the Internet. The task of a firewall is to inspect the network

packets that pass through it and then to decide whether to let them pass. Firewalls use a set of predefined rules to facilitate this decision. These rules may specify static filters, but may also be functions of the history of network traffic.

The protection from attacks from outside the network is a well understood problem and has become the business model of many companies. Firewalls are considered a secure *black box* which protects the network from attacks. However, since many firewall appliances are purchased from third-party vendors, people have no control and insight into their actual functionality. Companies and individuals alike rely on the trustworthiness of firewall appliances. Most firewalls are made of general-purpose hardware with just the same capabilities as any modern computer. In addition, firewalls are often equipped with update mechanisms which make their functionality completely replaceable. It seems naïve to see a firewall as a secure black box. Indeed, as the documents that were leaked by Edward Snowden in 2013 reveal, the National Security Agency has the capability to install backdoors in a number of commercial firewalls: JETPLOW, HALLUXWATER, FEEDTROUGH, GOURMETTROUGH, and SOUFFLETROUGH [118].

Under the assumption that firewalls are not actively malicious, i.e. do not attack the network on their own, one might still surmise that their rules are incomplete and do not filter all possible attacks. To alleviate this problem, one might concatenate firewalls from several suppliers just like sieves are layered in water filters. The serial concatenation of firewalls does not yield a secure solution if one considers the possibility of actively malicious firewalls: If one is unsure as to which firewall is compromised one also does not know which firewall the network needs protection from.

To realise a secure concatenation of firewalls, we suggest a trusted device that compares the input to a firewall network to its output. Such a trusted hardware would be much simpler than a complete firewall. We envision such a device to be realised as a microchip. Surprisingly, this approach still is insecure. A parallel composition of firewalls using a trusted comparator yields a secure firewall network, however.

In this chapter, we present a novel methodology to analyse architectures of multiple firewalls based on the Universal Composability (UC) [41] framework. The UC framework allows us to define security in a natural way while also providing us with a composition theorem. We rigorously analyse different approaches. Our analysis reveals subtle attacks on presumably secure architectures. We give ideal functionalities for these architectures which make these weaknesses explicit.

The basic UC framework does not feature a notion of time and thus cannot model availability. In the course of our efforts we extend our considerations to providing availability guarantees as well. We provide a "recipe" for modeling and analysing network components like firewalls. It is our belief that this work demonstrates the actual suitability of formal security models to analyse the security of computer networks.

This chapter is based on a Diploma thesis [9] and two conference papers [3, 4].

3.1 Related Work

To our knowledge, we are first to explicitly model computer networks in the UC framework.

Network Firewalls The purpose, aim and function of network firewalls is widely understood and agreed upon [25, 116, 178]. The informational Request for Comments 2979 [84] defines characteristics of firewalls. Since there is no globally agreed-on standard for what constitutes good and bad network packets however, there is also no complete specification of the function of a firewall.

The security of firewalls or systems of firewalls has mainly been studied under two aspects. One concern is verifying the correctness and soundness of rule sets. Gouda et al. [98] develop a formal model for verification of distributed rule sets based on trees. They are able to check whether the firewall system accepts or denies a specific class of packets. Ingols et al. [117] check for conflicting or otherwise problematic rules with the aid of Binary Decision Diagrams. Maldonado-Lopez et al. [155] develop a framework for modeling firewall policies to detect and prevent conflicts in firewall policies for software-defined networks.

Another concern is firewall testing. Vigna [196] proposes a firewall testing methodology based on a formal network model instead of checklist testing. He proposes modeling computer networks as hypergraphs. Our modeling of firewall networks is inspired by his work. Kamara et al. [124] provide an analysis of various security holes found in firewall solutions. They present vulnerability matrices which categorise firewall operation, security hole effect, and causes of security holes. One security hole they consider in their analysis is of particular interest: The Hypertext Transfer Protocol (HTTP) proxy of a commercial firewall appliance suffers from a buffer overflow weakness that makes it possible for an attacker to execute arbitrary code on the host running the software [180].

We are not aware of any works that consider the firewall as being malicious.

Formal Analysis of Computer Networks While network security is considered a practical field, formal methods have also been applied to computer networks. Research generally concentrates on modeling attacks and vulnerabilities [117] and on generating verification policies [112, 147]. While such approaches help in configuring specific network components and in mitigating threats, they do not have the advantages of cryptographic security models.

Universal Composability The UC framework [41, 42] was proposed by Canetti in 2001. It follows the "real-world-ideal-world" paradigm and is inherently asynchronous. Alternative approaches to the problem of composable security include the reactive simulatability framework [19, 173], the GNUC framework [110]—which aims at being a drop-in replacement for UC—or the Abstract Cryptography paradigm [157, 158]. Among these, the UC framework has received the most attention.

Various variants of UC have been proposed [43, 46, 47, 194].

Universally Composable Practical Protocols The UC framework is the quasi state-of-the-art framework for proving the security of cryptographic *building block* protocols like Commitments [45] or Oblivious Transfer [172]. Because it has a composition theorem, it is argued that more complex protocols can then be composed of these components.

However, the UC framework has also been used to prove the security of more complex schemes, such as Transport Layer Security [85], OAuth [51], and disk encryption [65]. A particularly well-known example of a UC-secure multi-party computation in a practical application is the Danish sugar beet auction [31]. Our contribution falls in line with this work. We investigate composing large computer networks.

Synchronous Universal Composition The UC framework is asynchronous—exactly one machine is running at any given moment. This convention greatly simplifies analysis but at the same time makes modeling the progression of time difficult. More concretely, one cannot simultaneously prove input completeness (the inputs of all honest parties are considered) and guaranteed termination (the protocol does not "hang"). There have been various approaches to modeling synchrony in the UC framework [108, 123, 168]. They, however, either lack in expressiveness or modify the foundations of the framework. The recently proposed approach by Katz, Maurer, Tackmann, and Zikas [132] generalises previous results. It hinges on two ideal hybrid functionalities $\mathcal{F}_{\mathsf{BD}}$ and $\mathcal{F}_{\mathsf{clock}}$ that do not change the framework. Backes et al. [18] propose TUC, an extension to the GNUC framework.

Secure Hardware In their seminal work, Goldreich and Ostrovsky [92] explore the challenge of software protection. They base their main result on Oblivious Random Access Machines (ORAMs). A Random Access Machine (RAM) is oblivious if the sequence of its memory accesses is independent of its input.

Hofheinz, Müller-Quade, and Unruh [109] use trusted signature cards as "catalysts" for Zero-Knowledge and bit commitments.

Katz [130] uses tamper-proof hardware to realise universally composable multi-party computation. He assumes tamper-proof tokens that can be programmed with an arbitrary program. Such a programmed token is then handed to another party in the protocol, which may then interact with the token. This idea has gained much interest [67, 71–74, 99, 163].

Goldwasser et al. [93] introduce the computational paradigm of one-time programs, i.e. programs that can only be run once, on one input. Of course, such programs cannot exist purely in software, as software can be copied indefinitely. Goldwasser et al. introduce "one-time-memory devices" to create a compiler for one-time programs.

A more recent idea is to use Physically Uncloneable Functions (PUFs) as a setup assumption. A PUF is a physical source of randomness that is assumed hard to clone, i.e. its output is unique and hard to predict [16, 67, 154, 170].

Robust Combiners The idea of mistrusting the implementation of a secure functionality has been studied in the scope of *robust combiners*. A (k, n)-robust combiner combines n candidate implementations of the secure functionality $\mathcal{P}$ in a way that the overall security still holds if at least k implementations are secure [106]. More formally: A (k, n)-robust combiner for a cryptographic primitive $\mathcal{P}$ is a mechanism that takes n candidate schemes for $\mathcal{P}$ and combines them so that if at least k candidate schemes

indeed implement $\mathcal{P}$, the combiner also implements $\mathcal{P}$. For example, the concatenation of encryptions $\mathsf{Enc}_c(x) = \mathsf{Enc}_1(\mathsf{Enc}_2(x))$ is a robust combiner for Indistinguishability under Chosen-Plaintext Attack (IND-CPA) secure encryption [106]. Further work includes robust combiners for Oblivious Transfer [103, 160], for Private Information Retrieval [159], and for collision-resistant hash functions [34].

The notion of a robust combiner is not suited well for our purposes. The very definition of robust combiners requires a specific and fixed functionality $\mathcal{P}$. However, in the case of firewalls, it is unclear what this functionality precisely is. Informally speaking, the functionality of a network firewall is "filtering all malicious packets". It is not possible to formalise this functionality in a protocol or a program, since, in general, it is not possible to decide whether an arbitrary packet is malicious.

Byzantine Fault Tolerance Our constructions are reminiscent of problems in byzantine fault tolerance. However, we use a very different communication structure. In the original Byzantine Generals Problem [148], every party can communicate with every other party. This leads to specific bounds concerning the number of trusted parties needed to achieve fault tolerance. Even when signing messages is possible, in order to allow for m corrupted parties, one still needs at least $(2m+1)$ trusted parties and $(m+1)$ rounds of communication. In our case, we do not allow the parties to communicate freely, but only according to the specific structure of the network—we do not allow firewalls to exchange messages with each other. Thus, the results which byzantine fault tolerance research provides are not applicable to our scenario. What is more, our research focus is on the application of cryptographic frameworks to practical networks, for which byzantine fault tolerance is only a special case.

3.2 A Model for Firewalls

We assume a packet-switched Local-Area Network (LAN) in which there are only uncompromised hosts. They are connected through a single uplink to the Internet, in which are potentially compromised hosts. To facilitate an easier discussion, we call machines in the LAN being "inside" and machines on the Internet being "outside". The "inside" is only connected to the "outside" through a firewall (network), whose job is to protect machines "inside" from machines "outside". For ease of exposition, we model communication in networks in one direction only.

We assume that each firewall can have any number of *network interfaces* for input and output. The output of a firewall then depends on the packet $p \in P$ it gets as input (where P is the set of all possible packets), its internal state $s \in S$ and the network interface $i \in I$ the packet arrived on.

After processing this information, the firewall then outputs a packet p' on a specific network interface i' and updates its internal state (e.g. outputs a new internal state s'). The functionality of a firewall is defined formally in Definition 1.

Definition 1 (The functionality of an ideal firewall F_j)**.**

$$F_{\mathrm{fw}_j} : P \times I \times S \to (P \cup \perp) \times (I \cup \perp) \times S$$

$$F_{\mathrm{fw}_j}(p,i,s) = \begin{cases} (p',i',s') & \text{if output is generated,} \\ (\bot,\bot,s') & \text{else.} \end{cases}$$

The exact purpose and function of a network firewall is ambiguous. Because we do not wish to impose any artificial restrictions on the functionality of a firewall in our model, we do not give a specific realisation of its function. We only describe the domain the firewall operates on.

We stress that our definition of a firewall functionality is universal. Because it is stateful—it receives its previous state as input, may use it for its computation and outputs an updated state—a firewall may base its output on an arbitrarily long history of incoming and outgoing packets. It may, for example, reconstruct a Transmission Control Protocol (TCP) session. Further, we do not restrict how its output depends on its input. A firewall might for example receive a packet, store it, transform it, and output it much later. Because the functionality processes whole packets including their payload, our definition covers the whole network protocol stack (e.g. Ethernet, IP, TCP, HTTP, HTML).

We assume an outside adversary is able to communicate with compromised firewalls inside the network through covert channels. (Through a Global System for Mobile Communications link, for example.)

3.3 Composing Firewalls

In this section, we discuss different architectural solutions to the problem of maliciously acting firewalls and analyse their security in the UC framework. To simplify the exposition, we only discuss unidirectional networks. The easiest approach for extending the model to bidirectional communication would be using a independent instance of $\mathcal{F}_{\mathsf{ideal}}$ for each direction and deducing the security of the composed system by using the composition theorem. However, this approach would require the protocols for each direction to be independent of each other and not have a joint state. Actual firewall solutions base their decisions on all observed packets (not only those in one direction), however. Thus, the security of the bidirectional extensions of the architectures we discuss has to be proven manually.

We only discuss the security of a single atomic building block for complex firewall architectures. The Composition Theorem of the UC framework provides us with a strong guarantee for networks composed of several building blocks.

Adaptive vs. Non-Adaptive Corruption The classic UC framework provides the adversary with the ability to corrupt parties adaptively using corruption messages. We will, however, be using a non-adaptive corruption model, where the adversary has to decide which party it wants to corrupt prior to the protocol execution. This simplifies the analysis and the proof of security significantly. In general, these two models of corruption are not equivalent [44]. As the constructions we discuss are symmetric, the two models of corruption are equivalent in our case. We will also restrict the adversary to corrupt

one party only. The adversary will gain full control over this party and will be able to send and receive messages in its name.

3.3.1 Trusted Hardware

The UC framework gives strong security guarantees. It is difficult to obtain secure protocols in the plain UC model, however. This problem can be alleviated by using a set-up assumption like a Common Reference String [45], a Public-Key Infrastructure [22], or trusted hardware [109, 130]. Secure hardware is often modelled as tokens that offer some black box functionality. They cannot be made to deviate from the specified functionality and parties may not learn the value of internally stored values if they are not allowed to do so.

We envision similar network devices for our task. They have two very simple functionalities depending on the direction of the packet flow. In one direction their job is to compare packets that come in from different sources and decide whether to let them pass. In the other direction their job is to split incoming packets and distribute them to several firewalls. Because these "packet comparators" offer only limited functionality, they could be manufactured easily, maybe even by the network owner himself. Also, it would be very hard to hide any "backdoors" or "undocumented functionality" in the device. Thirdly, because of its simple functionality, the device need not be able to download updates or even be freely programmable. We envision such a device to be realised as an Application-Specific Integrated Circuit. In our security analysis, we assume that the specialised hardware we use cannot be compromised, i.e. is *trusted hardware*.

We formalise the functionality for comparison in Figure 3.2 and the functionality for splitting in Figure 3.1. This explicit distinction is solely for the purpose of simplifying the model in the case of uni-directional communication. In our model, we model the trusted hardware as having unlimited storage capacity. In practice, memory size is of course limited. Hence, received packets cannot be stored indefinitely. It is a challenge to determine an optimal strategy for purging old packets (see also Section 3.9.2).

Practically realising the required functionality at wire speed is not trivial. It is not impractical, however: The trusted device need only compare packets as fast as the slowest of the firewalls can analyse them. Any network firewall realises a function that is comparable to that of our trusted device.

A firewall might change several Internet Protocol (IP) header fields during inspection of a packet. Take for example the Time To Live (TTL) field, which usually is decremented by 1 on each network device it passes. Hence, two packets that differ on the value of the TTL field need not necessarily be considered unequal by the trusted packet comparator. On the other hand, two packets may only be considered equal when their relevant fields match. Examples for relevant fields are the source address and the target address. We express this notion of "packet equivalence" with a relation $\equiv$ that we assume to be defined appropriately.

Further, a firewall may change the order of the packets it delivers. Some packets might need to be inspected more closely (Deep Packet Inspection), while others would just be waved through—take for example packages from a Voice over IP connection. Therefore,

An idealised description of trusted hardware split

Upon receiving packet p on the interface connected to the outside:

- Output p to every interface connected to a firewall.

Figure 3.1: The splitting functionality the trusted hardware must perform. We assume that there is a fixed interface for connecting with the outside network and a number of other interfaces to which the firewalls are connected.

it is not sufficient for the trusted hardware to compare packets one-by-one in the order they arrive. For the same reason, probabilistic packet comparison is not a feasible option to improve the performance of the trusted comparator.

3.4 Serial Concatenation of Two Firewalls

In the Introduction, we briefly discussed that the serial concatenation of firewalls (in the naïve way) is not secure even when using secure hardware. We formalise this claim in this section.

Figure 3.3 shows a graphical representation of the network architecture of the serial concatenation. fw_1, fw_2, split and hw will be the parties in the corresponding UC protocol.

Packets from outside the network always arrive at split first. Parties cannot communicate directly. Instead, we provide them with an ideal functionality for communication. This functionality ensures that parties can only communicate in a way that is fixed by the structure of the network. This is justified, since in an "real" network, components can also only communicate along the network cables. This can easily be realised in UC since ideal functionalities can write to the subprocess tape of all parties. Parties then can ignore all input on their communication tapes. We omit the session IDs from all descriptions of functionalities and protocols. Different instances behave independently. We use the notion of "public delayed output", introduced by Canetti [41]. This means that a message is given to the adversary prior to delivery. The adversary then decides when (or whether) it is delivered.

The main idea for the ideal functionality is that any firewall architecture, regardless of the amount of different firewalls or their specific rule set, should behave as if the corrupted firewall was not there (see Figure 3.4). This is well-defined in the UC framework, since we can remove that firewall from the architecture in the ideal model. We give the adversary the ability to block the delivery for each packet individually to explicitly exclude availability from the model.

The idea is that if fw_2 is corrupted, it could output a malicious packet just at the same time this packet arrives at split (sent by the environment). This would force hw to

An idealised description of trusted hardware hw

Keep a local cache realised as an unordered list.
Upon receiving packet p on interface i:

- If there is another input interface $j \neq i$, and a corresponding entry (j, q) with $p \equiv q$ in the cache:
  - Remove (j, q) from the cache,
  - output p.
- Otherwise, store (i, p) in the cache.

Figure 3.2: The function the trusted hardware must perform. Because network packets may arrive at different times or in a different order, they must be cached in order to be available for comparison. In our model the list has infinite capacity, but in practice there must be a strategy to purge old packets. The relation $\equiv$ for packets must consider all packet fields that are relevant.

output the packet, even though it was blocked by fw_1.

Definition 2 (The protocol of the serial firewall architecture π_{serial})**.**

1. split: Upon receiving (input, p): Call $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, \text{out}_{\text{fw}}, \text{out}_{\text{hw}}, p)$.
2. fw_k: Upon receiving (in, p): Compute $F_{fw_k}(p, \text{in}, s) = (p', i', s')$. If $p' \neq \perp$ and $i' \neq \perp$, call $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, i', p')$. Save the new internal state s'.
3. hw: Check whether there are two entries (p, in) and $(q, \text{in}_{\text{cmp}})$ in the locale storage (with $p \equiv q$). If so, write p to the output tape and delete the entries.

We now show that the serial concatenation of firewalls is not secure, even with a trusted comparator. To prove the statement, it suffices to show that there exists an attack which can not be simulated. We describe such an attack. The general idea is that if fw_2 is corrupted, it could output a malicious packet just at the same time this packet arrives at split (sent by the environment). This would force hw to output the packet, even though it was blocked by fw_1.

Theorem 1. π_{serial} *does* not *UC realise* $\mathcal{F}_{\mathsf{ideal}}$ *in the* $\mathcal{F}_{\mathsf{serial}}$*-hybrid model.*

Proof. Let fw_2 be corrupted and fw_1 be honest. Let p be a packet that is blocked by fw_1. The environment inputs p to split. This will cause $(p, \text{in}_{\text{cmp}})$ to be send to hw from split. In its next activation the adversary uses fw_2 to call $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, \text{out}, p)$ and advises the ideal functionality to deliver (p, in) to hw. hw will now have two identical packets on

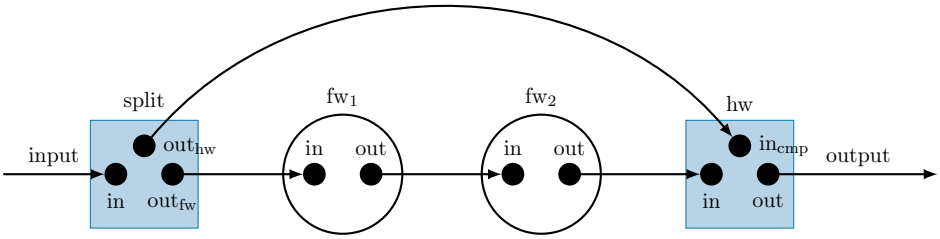

Figure 3.3: The serial concatenation of firewalls using secure hardware to compare packets. hw compares whether "what goes in, comes out". split forwards the packet to two components. The connecting arrows represent network cables in a "real" network.

The ideal functionality of two firewalls $\mathcal{F}_{\mathsf{ideal}}$

- Upon receiving (input, p):
  - Ask the adversary if p should be delivered. If yes, let fw_k be the non-corrupted party; compute $F_{\mathrm{fw}_k}(p, \mathrm{in}, s) = (p', i', s')$. Write p' to the output tape of hw, if $p' \neq \perp$ and $i' \neq \perp$. Else, do nothing. Save the new internal state s'.

Figure 3.4: The ideal functionality of two firewalls.

different interfaces (one from split and one from fw_2) in its storage and output p, even though p has been blocked by fw_1.

There is no simulator which can simulate this attack, since fw_1 will block the packet in the ideal model and the output of fw_2 will not be considered. □

3.5 Parallel Composition of Two Firewalls

The serial composition of two firewalls is not secure with regard to our envisioned ideal functionality. Better results can be achieved using parallel composition. The idea is that the trusted hardware only accepts a packet if both firewalls accept it. Figure 3.6 shows this composition. We will now discuss the security of this architecture.

The protocol of the parallel architecture is defined in Definition 3.

Definition 3 (The protocol of the parallel architecture π_{parallel})**.**

1. split: Upon receiving (input, p): Call $\mathcal{F}_{\mathsf{parallel}}(\mathsf{send}, \mathrm{out}_1, \mathrm{out}_2, p)$.
2. fw_k: Upon receiving (in, p): Compute

The ideal network function $\mathcal{F}_{\mathsf{serial}}$

Initialise an empty stack for both hw and fw_1.

- Upon receiving $(\mathsf{send}, \mathrm{out}_{\mathrm{fw}}, \mathrm{out}_{\mathrm{hw}}, p)$ from split:
  - Push p on the stack for hw and fw_1.
  - Give the adversary p and let him choose one of the stacks. Pop the most recent packet from that stack. If the adversary chose hw's stack, send $(p, \mathrm{in}_{\mathrm{cmp}})$ to hw, otherwise send (p, in) to fw_1.
- Upon receiving $(\mathsf{send}, \mathrm{out}, p)$ from fw_1: Provide a public delayed output of (p, in) to fw_2.
- Upon receiving $(\mathsf{send}, \mathrm{out}, p)$ from fw_2: Provide a public delayed output of (p, in) to hw.

Figure 3.5: The ideal network function representing the serial concatenation of firewalls with special hardware.

$F_{\mathrm{fw}_k}(p, \mathrm{in}, s) = (p', i', s')$. If $p' \neq \perp$ and $i' \neq \perp$, call $\mathcal{F}_{\mathsf{parallel}}(\mathsf{send}, p', i')$. Save the new internal state s'.

3. hw: Upon receiving (in_i, p), check if there is an entry (in_j, q) with $i \neq j$ and $p \equiv q$ in the internal storage. If so, write p to the output tape and remove both entries. Else, do nothing.

The functionality describing the network structure is depicted in Figure 3.7.

We will compare the protocol from Definition 3 with an ideal functionality. The ideal functionality is the same as in the serial case, since the natural approach of defining ideal functionalities only uses the uncorrupted firewall, which again leads to the functionality in Figure 3.4. However, as in the serial case, the parallel architecture does not realise this functionality.

Theorem 2. *π_{parallel} does not UC realise $\mathcal{F}_{\mathsf{ideal}}$ in the $\mathcal{F}_{\mathsf{parallel}}$-hybrid model.*

We prove this by describing an attack which can not be simulated.

Proof. Let, w.l.o.g., fw_1 be honest and fw_2 be corrupted. Also, let p_1 and p_2 be packets that are accepted by fw_1. The environment sends packets p_1 and p_2 to the architecture which the adversary delivers to fw_1. Both packets are accepted by fw_1 and forwarded to hw. Then, the adversary sends packets p_2 and p_1 from fw_2. Since both packets have been accepted and were sent to hw previously (but in reverse order), hw will send out

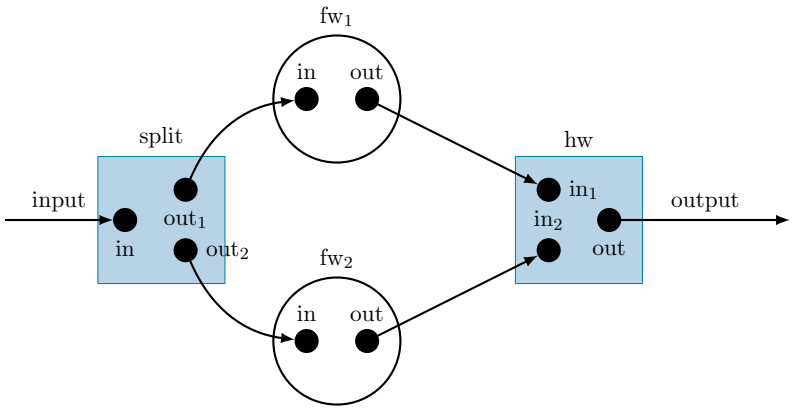

Figure 3.6: The parallel composition of two firewalls with trusted hardware. *hw* only accepts packets that are output by both firewalls.

p_2 and p_1—in this order. Thus, the adversary was able to reverse the order of packets. Since the adversary is not allowed to influence the order of packets in the ideal model, there exists no simulator which can simulate this attack. □

This attack does not seem troublesome in practice, however. The Internet Protocol explicitly does not give any guarantees about the ordering of packets, since the correct order is encoded in the package payload. The payload, however, can not be altered by the adversary. Thus, we modify our ideal functionality and explicitly grant the attacker the ability to reorder the outgoing packet stream.

This new ideal functionality is described in Figure 3.8. The attacker now has the ability to change the order of packets which have been accepted by the uncorrupted firewall. We will now prove that the parallel architecture realises this ideal functionality.

Theorem 3. π_{parallel} *UC realises* $\mathcal{F}_{\mathsf{ideal}_2}$ *in the* $\mathcal{F}_{\mathsf{parallel}}$*-hybrid model.*

Proof. To prove the statement, we will give the description of a simulator and show that this simulator can simulate every adversary, so that no environment can distinguish between the real and ideal model. Let w.l.o.g. fw_1 be corrupted and fw_2 be honest. Let $\mathcal{S}$ be a simulator with the following functionality:

- Upon activation, or when given a packet p, simulate the real model and observe its output. If the output of the real model is a packet p', compute the position of p' in the internal memory structure of the ideal functionality and advise the functionality to deliver the packet at that index. (The case that p' is not found in the internal memory structure of the ideal functionality need not be covered, as is proven below.)

Note that the simulator receives exactly the same input as the adversary in the real model—it can perfectly simulate the communication between the adversary and the

The ideal network function $\mathcal{F}_{\mathsf{parallel}}$
Initialise an empty stack for both fw_1 and fw_2.

- Upon receiving $(\mathsf{send}, \mathrm{out}_1, \mathrm{out}_2, p)$ from split: Push p on the stack for both fw_1 and fw_2.
- Upon receiving $(\mathsf{deliver}, \mathrm{fw}_k)$ from the adversary: Pop the most recent packet p from fw_k's stack. Provide public delayed output of (p, in) to fw_k.
- Upon receiving $(\mathsf{send}, \mathrm{out}, p)$ from fw_k: Provide a public delayed output of (p, in_k) to hw.

Figure 3.7: The ideal network function representing the parallel concatenation of firewalls with trusted hardware.

The ideal functionality of two firewalls with packet reordering $\mathcal{F}_{\mathsf{ideal}_2}$

- Upon receiving (input, p): Let w.l.o.g. fw_1 be the non-corrupted party; compute $F_{\mathrm{fw}_1}(p, \mathrm{in}, s) = (p', i', s')$. If $p' \neq \perp$ and $i' \neq \perp$, save p' in an indexed memory structure m at the next free index. Save new internal state s'. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j)$ from the adversary: If $m[j]$ contains a valid packet, write $(\mathrm{out}, m[j])$ to the output tape of hw and clear $m[j]$; else do nothing.

Figure 3.8: The ideal functionality of two firewalls with packet reordering.

environment. Thus, the environment can only distinguish the models based on their output streams. We argue that the output of the real and ideal model are identical. Assume towards a contradiction that they are not.

Let $\{fw_2(S)\}$ denote the set of all packets fw_2 outputs when given the input stream S. There are two possibilities which would cause a difference in output streams:

- The adversary in the real model suppresses a packet which is not suppressed in the ideal model. This is impossible however, since the simulator only advises the ideal functionality to deliver a packet if it observes it being output in its simulation of the real model.
- The real model outputs a packet which is not output in the ideal world.
  Assume that this were the case and let p be that packet. The following conditions have to hold: p has to be in $\{fw_2(S)\}$ and p has to be output by $\mathcal{A}$ (using fw_1).

This is true because the trusted hardware will ensure that a packet is only output when both firewalls accept it. For a packet *not* to be output in the *ideal model*, one of the following conditions have to hold:

- p is not in $\{fw_2(S)\}$. This way, p will not be in the internal memory structure of the ideal functionality. Thus, the simulator can not advise the delivery of that packet. This is a contradiction, since we assumed that p was output in the real model, which in turn implies that $p \in \{fw_2(S)\}$.
- $p \in \{fw_2(S)\}$ *and* the simulator did not advise the functionality to deliver p. This is also a contradiction, since we assumed that p was output in the real model. This would cause the simulator to advise the output of p by definition.

We now have shown that the assumption that the environment can observe a difference in packet output stream in the real and the ideal world leads to a contradiction in all cases. This, together with the ability of the simulator to fully simulate the adversary, proves the indistinguishability of the models. □

3.6 Parallel Composition of Three or More Firewalls

The parallel approach to compose firewalls described above does indeed improve security. The natural extension of the two-firewall architecture is a quorum of three firewalls. This seems more reliable in practice than the combination of only two firewalls.

In the following section, we assume that uncorrupted firewalls in this architecture will have the same behaviour. However, we allow them to disagree on the order of packets.

There is a non-trivial attack on this architecture. When both uncorrupted firewalls both output the same packet p, the adversary can use clever timing to output p from the corrupted firewall directly after the first uncorrupted firewall. The trusted hardware would then observe two packets p on different interfaces and output p. However, a third packet p would arrive from the second uncorrupted firewall. Then, the adversary could output p again. This would cause hw to output p again and thus duplicate the packet.

Interestingly, the natural extension of $\mathcal{F}_{\mathsf{ideal}_2}$ to the case of three firewalls already covers this attack. The functionality is depicted in Figure 3.9.

The other protocols and functionalities ($\pi_{\mathsf{parallel}_3}$ and $\mathcal{F}_{\mathsf{parallel}_3}$) are easily extended to the case of three firewalls by adding the third firewall as an additional party. In Section 3.7.1 we give a generalised definition $\mathcal{F}_{\mathsf{net}}$ that allows for an easy formulation of ideal network functionalities.

It can easily be seen that the attack described above can also be performed in $\mathcal{F}_{\mathsf{ideal}_3}$. When fw_1 and fw_2 both output the same packet, both will be saved in m. The adversary can now output both packets by giving the right indices.

Theorem 4. *$\pi_{\mathsf{parallel}_3}$ UC realises $\mathcal{F}_{\mathsf{ideal}_3}$ in the $\mathcal{F}_{\mathsf{parallel}_3}$-hybrid model.*

The proof is similar to the proof of Theorem 3.

Proof. We give the description of a simulator and show that this simulator can simulate every adversary, so that no environment can distinguish between the real and ideal model.

The ideal functionality of three firewalls $\mathcal{F}_{\mathsf{ideal}_3}$

- Upon receiving (input, p): Let w.l.o.g. fw_1 and fw_2 be the non-corrupted parties; compute $F_{\text{fw}_1}(p, \text{in}, s) = (p', i', s')$ and $F_{\text{fw}_2}(p, \text{in}, s) = (p'', i'', s'')$. If $p' \neq \perp$ and $i' \neq \perp$, save p' in an indexed memory structure m at the next free index. If $p'' \neq \perp$ and $i'' \neq \perp$, also save p'' in m at the next free index. Save the new internal states. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j)$ from the adversary: If $m[j]$ contains a valid packet, write $(\text{out}, m[j])$ to the output tape of hw and clear $m[j]$; else do nothing.

Figure 3.9: The ideal functionality of three firewalls.

Let w.l.o.g. fw_1 be corrupted, and fw_2 and fw_3 be honest. Let $\mathcal{S}$ be a simulator with the following functionality:

- Upon activation, or when given a packet p, simulate the real model and observe its output. If the output of the real model is a packet p', compute the position of p' in the internal memory structure of the ideal functionality and advise the functionality to deliver the packet at that index.

Note that the simulator receives exactly the same input as the adversary in the real model—it can perfectly simulate the communication between the adversary and the environment. Thus, the environment can only distinguish the models based on their output streams. We argue that the output of the real and ideal model are identical. Assume towards a contradiction that they are not.

Let $\{fw_i(S)\}$ denote the set of all packets fw_i outputs when given the input stream S. (Remember that fw_1 is controlled by the adversary.) There are two possibilities which would cause a difference in output streams:

- The adversary in the real model suppresses a packet which is not suppressed in the ideal model. This is impossible however, since the simulator only advises the ideal functionality to deliver a packet if it observes it being output in its simulation of the real model.
- The real model outputs a packet which is not output in the ideal world.
  Either $p \in \{fw_2(S)\}$ or $p \in \{fw_3(S)$ (or both) and p is output by $\mathcal{A}$ (using fw_1). This is true because the trusted hardware will ensure that a packet is only output when two firewalls accept it. For a packet *not* to be output in the *ideal model*, one of the following conditions have to hold:
  - p is not in $\{fw_2(S)\} \cup \{fw_3(S)\}$. This way, p will not be in the internal memory structure of the ideal functionality. Thus, the simulator can not advise the

delivery of that packet. This is a contradiction, since we assumed that p was output in the real model, which in turn implies that $p \in \{fw_2(S)\} \cup \{fw_3(S)\}$.

– $p \in \{fw_2(S)\} \cup \{fw_3(S)\}$ (and $p \notin \{fw_2(S) \cap \{fw_3(S)\})$ *and* the simulator did not advise the functionality to deliver p. This is also a contradiction, since we assumed that p was output in the real model. This would cause the simulator to advise the output of p by definition.

We now have shown that the assumption that the environment can observe a difference in packet output stream in the real and the ideal world leads to a contradiction in all cases. This, together with the ability of the simulator to fully simulate the adversary, proves the indistinguishability of the models. □

In practice, however, it is not acceptable to give an attacker the ability to duplicate packets. Even though higher protocols (such as TCP) must be able to deal with duplicated packets [114], a twofold increase in transmitted data can be problematic. Instead we will slightly alter the functionality of our trusted hardware to prevent the attack. The functionality is depicted in Figure 3.10. The general idea is that at the moment the hardware outputs a packet, exactly two firewalls must have output this packet before. Then, the hardware can mark this packet as missing from the third firewall. If it arrives eventually, this mark will be removed and no further action will be taken.

An idealised description of trusted hardware without packet duplication

hw_3

Keep a local cache for each incoming interface realised as an unordered list. Upon receiving packet p on interface i:

- Check if the cache of interface i contains an entry $-q$ with $p \equiv q$. If so, delete $-q$ and halt.
- Check if there exists an interface $j \neq i$ with an entry q with $p \equiv q$ in the cache of that interface:
  - Remove q from the cache,
  - output p,
  - add an entry $-p$ to the cache of all other interfaces k with $k \neq i$ and $k \neq j$.
- Otherwise, store p in the cache of interface i.

Figure 3.10: The updated functionality of the trusted hardware to prevent packet duplication. The hardware now marks a packet as "missing" if a firewall has not yet delivered it, but two others have.

The corresponding ideal functionality is depicted in Figure 3.11. The general idea is that the ideal functionality now continuously checks whether the amount of identical packets being given to hw matches the amount of identical packets which either one of the uncorrupted firewalls send. This way, the adversary will not be able to duplicate packets in the ideal model. As previously however, we will allow the reordering of packets.

The ideal functionality of three firewalls without packet duplication
$\mathcal{F}_{\mathsf{ideal}_4}$

Initialise three index based memory structures m_1, m_2 and m_3.

- Upon receiving (input, p): Let w.l.o.g. fw_1 and fw_2 be the non-corrupted parties; compute $F_{\mathrm{fw}_1}(p, \mathrm{in}, s) = (p', i', s')$ and $F_{\mathrm{fw}_2}(p, \mathrm{in}, s) = (p'', i'', s'')$. Save the new internal states. Save p' in m_1 and p'' in m_2. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j, k)$ $(k \in \{1, 2\})$: If $m_k[j]$ contains a valid packet p''':
  - Check how many times that packet (or an equivalent packet) is in m_3. Let that number be n.
  - Check, if either m_1 or m_2 (or both) contain that packet at least $n + 1$ times.
  - If so, write (out, p''') to the output tape of hw and to m_3.

Figure 3.11: The ideal functionality of three firewalls without packet duplication. For every packet, at least one of the firewalls must have send this packet at least as often, as it got passed to hw.

We now show that our firewall architecture with this updated trusted hardware realises an ideal functionality (given in Figure 3.11), which does not allow the adversary to duplicate packages.

Theorem 5. $\pi_{\mathsf{parallel}_4}$ *UC realises* $\mathcal{F}_{\mathsf{ideal}_4}$ *in the* $\mathcal{F}_{\mathsf{parallel}_3}$*-hybrid model.*

Proof. The proof is similar to the proof of Theorem 3. We argue that the simulator behaves identically to the adversary and that the output of the ideal network is identical to the output of the real network. Let $\mathcal{S}$ be a simulator with the following functionality:

- Upon activation, or when given a packet p, simulate the real model and observe its output. If the output of the real model is a packet p', compute (for the ideal functionality) the index of the memory structure in which p' is saved as well as its position within the memory. Advise the functionality to deliver the packet on that index. (The case that p' is not found in the internal memory structure of the ideal functionality need not be covered, as is proven below.)

The argument that $\mathcal{S}$ will never mistakenly suppress a packet in the ideal model is identical to Case 1 in the proof of Theorem 3. We need to argue Case 2: It is impossible

that $\mathcal{S}$ is unable to schedule a packet it observes in the output of its internal simulation of the real network. Let p be such a packet that, after the input stream S is processed, is written to the output tape of hw in the real model but not to the internal memory structure of $\mathcal{F}_{\mathsf{ideal}_4}$.

Let m_A, m_1 and m_2 be the lists the trusted hardware uses in the protocol for storing the packets output by the firewalls and marking the "negative" packets. Let m_{hw} be the list of all packets it has ever output. Let m'_1, m'_2, m'_{out} be the lists the ideal functionality uses for keeping track of the packets. Let $\|m\|_p$ denote the number of packets p the list m contains. We then define $|m|_p := \|m\|_p - \|m\|_{-p}$.

First, observe that $\mathcal{S}$ only schedules packets it observes in its simulation of the real model. Hence, by the description of hw: $|m_1|_p = |m'_1|_p - |m_{\text{hw}}|_p$ and $|m_2|_p = |m'_2|_p - |m_{\text{hw}}|_p$. Via the argument from Case 1 ($\forall p : |m'_{out}|_p \leq |m_{\text{hw}}|_p$) we have:

$$|m_1|_p \leq |m'_1|_p - |m'_{out}|_p \tag{3.1}$$
$$|m_2|_p \leq |m'_2|_p - |m'_{out}|_p \tag{3.2}$$

For p to be output in the real model, one of the following conditions has to hold:

$$|m_A|_p > 0 \text{ and } |m_1|_p > 0 \tag{3.3}$$
$$|m_A|_p > 0 \text{ and } |m_2|_p > 0 \tag{3.4}$$
$$|m_1|_p > 0 \text{ and } |m_2|_p > 0 \tag{3.5}$$

This is true because the trusted hardware will only forward packets which are in at least two of the packet lists. The functionality of *hw* can be restated in the following way: For every packet p which is output, insert a packet $-p$ into the lists of the three firewalls. If there are two packets p and $-p$ in the same list, both cancel each other out.

For p *not* to be written to the internal memory structure of $\mathcal{F}_{\mathsf{ideal}_4}$ in the ideal model, the following condition has to hold:

$$|m'_{out}|_p \geq |m'_1|_p \text{ and } |m'_{out}|_p \geq |m'_2|_p \tag{3.6}$$
$$\Leftrightarrow |m'_1|_p - |m'_{out}|_p \leq 0 \text{ and } |m'_2|_p - |m'_{out}|_p \leq 0 \tag{3.7}$$

This again describes the difference between the amount of packages p each individual firewall has output and the amount of packages p which got output in total after processing S.

Concluding the argument, conditions (3.1) to (3.5) give us $|m'_1|_p - |m'_{out}|_p > 0$ and $|m'_2|_p - |m'_{out}|_p > 0$, which contradict condition (3.7). □

3.7 Synchronous Universally Composable Computer Networks

The three-firewall approach intuitively has a better availability than the two-firewall approach. This intuition is supported by the observation that, in a two-firewall configuration, the adversary can mute network traffic by ceasing all communication, whereas in the three-firewall configuration he cannot. As the bare UC model is asynchronous and

thus does not model time we cannot substantiate the intuition by a formal analysis in the UC framework. What is more, our work thus far does not lend itself for analysing the security of different network components. For example, the network topology is "hard coded" in the protocol specification.

In this section we propose a methodology consisting of two generalised functionalities $\mathcal{F}_{\mathsf{wrap}}$ and $\mathcal{F}_{\mathsf{net}}$ for modeling network functions and a 5-step paradigm to model protocols. By abstracting various technical details of the framework, our methodology allows protocol designers to give a natural description of the network and its functionality and thus greatly simplifies its analysis. By incorporating a result of Katz et al. [132] our model also covers availability.

In this section we introduce our methodology for modeling computer networks with the UC framework. Using our methodology, we restate a well known result from Lamport et al. [148] and complete our previous results by modeling availability. The results presented in this section have been published in a conference paper [4].

Universally Composable Synchronous Computation

The UC framework is inherently asynchronous. Exactly one machine can run at any given moment. This simplification guarantees that the result of an execution is non-ambiguous. We perceive reality to be concurrent, however. Specifically, time passes and can be measured independently of the actions of any network component. To model a synchronised network with bounded latency we make use of the results of Katz, Maurer, Tackmann, and Zikas [132]. Specifically, we use their $\mathcal{F}_{\mathsf{clock}}$ functionality (Figure 3.12) as a synchronisation primitive. $\mathcal{F}_{\mathsf{clock}}$ allows the parties to wait for each other at synchronisation points. A party can signal when its round is complete. When all parties have completed their round, the round counter is reset.

Further, we use Katz et al.'s bounded-delay channels to model our network function $\mathcal{F}_{\mathsf{net}}$ (Figure 3.13). Each channel has an incoming queue. The idea is that the adversary may increase the channel delay up to a predefined limit. When a party polls the incoming queue for a channel, the counter is decreased. When it reaches zero, the party receives the next element from the channel queue.

$\mathcal{F}_{\mathsf{clock}}$ together with bounded-delay channels are sufficient to prove *guaranteed termination* for multi-party protocols [132], i.e. the protocol does not "hang" indefinitely. We express the availability of networks using this property.

3.7.1 The Basic Tools for Modeling a Computer Network

Ideally, modeling and analysing a network would require four steps: 1) Specify what the wanted functionality of the network is, 2) draw a graph of the network layout, 3) specify the protocol the machines in the network adhere to, and 4) prove that the protocol does achieve what the wanted functionality does.

We designed tools that capture various technical details of the UC framework and allow to use it in a way that is close to the intuitive approach. Specifically,

1. By defining $\mathcal{F}_{\mathsf{wrap}}$, we simplify the specification of an ideal network functionality.

The clock function $\mathcal{F}_{\mathsf{clock}}$

Initialise for each party p_i a bit $d_i := 0$.

- Upon receiving message (RoundOK) from party p_i set $d_i = 1$. If for all honest parties $d_i = 1$, then reset all d_i to 0. In any case, send (switch, p_i) to $\mathcal{A}$.
- Upon receiving message (RequestRound) from p_i, send d_i to p_i.

Figure 3.12: The ideal $\mathcal{F}_{\mathsf{clock}}$ functionality by Katz et al [132]. Parties can signal that they are done for the current round. When all honest parties have signalled RoundOK the round counter is reset. Further, parties can request the status of the round counter, learning whether the round has changed.

2. We provide an ideal network functionality $\mathcal{F}_{\mathsf{net}}^{G}$ that routes messages according to a given network topology induced by a network graph G.
3. We propose a 5-phase paradigm which allows for an easy and structured modeling of the behaviour of machines in the network.

The Ideal Network Functionality

We model the network as a directed graph $G = (V, E)$, while V is the set of machines in the network and $E \subseteq V^2$ is a relation on V. (We model the network as a directed graph to account for unidirectional links [127, 191].) To model bidirectional links, one requires that $(v, v') \in E$ iff $(v', v) \in E$. There is a delivery queue for each edge in the graph. Nodes can send messages for each outgoing edge and can poll incoming messages from each incoming edge. To send a packet, a party *src* can call the network functionality $\mathcal{F}_{\mathsf{net}}^{G}$ with a (finite) set of packets with corresponding recipients $\{(dest_1, msg_1), (dest_2, msg_2), \dots\}$. Each packet in the set will then be appended to the queue associated with the edge between nodes *src* and $dest_i$, if it exists. Further, modeling Katz et al.'s bounded delay channels [132], we associate two counters with each edge in the graph—one for the total delay and one for the accumulated delay of the channel. The adversary can increase the delay of a channel up to a fixed maximum amount. When a machine polls a queue the delay counter is decreased. When the delay has reached 0, a message is taken from the queue and handed to the machine. This allows for explicit modeling of different network latencies across different communication channels and allows the adversary to take advantage of that. This functionality makes it easy to define the communication channels for a network since one provides a graph of the network and the corresponding channel topology for the UC framework is generated automatically. We point out that we implicitly use Katz et al.'s "multisend" functionality where parties send multiple packets in one atomic call to the network. Because we do not consider adaptive corruption, the adversary cannot preempt parties during a send operation.

The ideal parameterised network function $\mathcal{F}_{\mathsf{net}}^{G,\delta}$

Interpret $G = (V, E)$ with $E \subseteq V^2$ as a directed graph. For each edge $e \in E$, initialise a queue Q_e and two variables d_e and d'_e which represent the current and the accumulated delay for the queue.

- Upon receiving a message (send, M) with $M = \{(dest_1, msg_1), (dest_2, msg_2), \dots\}$ from party src, for each tuple $(dest, msg) \in M$ do:
  - Check if $src, dest \in V$ and $(src, dest) \in E$. If so, continue. Else, ignore this tuple and start processing the next message.
  - Append msg to queue $Q_{(src,dest)}$. Hand msg to the adversary.
- Upon receiving message (delay, e, T) from $\mathcal{A}$: Let (d_e, d'_e) be the delay variables for the queue of edge e. If $d'_e + T \leq \delta$, set $d_e = d_e + T$ and $d'_e = d'_e + T$ and return $(\mathsf{delay\text{-}set})$ to the adversary. Else halt.
- Upon receiving message (fetch, Q) from party P and if $Q \subseteq V$:
  - Initialise a set of responses $r := \emptyset$ and for every party $P' \in Q \subseteq V$ with $(P', P) \in E$:
    - Let $(d_{(P',P)}, d'_{(P',P)})$ be the delay variables for edge (P', P).
    - Set $d_{(P',P)} = d_{(P',P)} - 1$. If $d_{(P',P)} = 0$, remove the first message msg from $Q_{(P',P)}$, set $d'_{(P',P)} = 0$, and set $r = r \cup (msg, (P', P))$.
  - If $r \neq \emptyset$, send r to P. Else halt.

Figure 3.13: The generalised ideal network function. It is parameterised with a graph that features protocol participants as nodes and expresses links as edges. We model the network as a directed graph to accommodate for specialised connection types as for example optical fibres or data diodes [127]. We also implemented Katz et al.'s bounded delay-channel [132] to model links with a delay.

The 5-Phase Paradigm

We propose a 5-phase paradigm for modeling network protocols. We require each honest party to follow this paradigm. An honest party will need exactly five explicit activations by the environment machine to finish its round. During its first activation ("input phase"), the party will accept input by the environment. Upon the second activation ("fetch phase"), it will issue a fetch request to the network to get its input which it will process and possibly send to other parties in one single call during the third activation ("send phase"). The fourth activation ("output phase") is the only activation in which a party will produce output to the environment. The fifth activation is used to signal "RoundOK" to $\mathcal{F}_{\mathsf{clock}}$: all work is done for this round. See Figure 3.14 for a summary.

Upon further activations the party will wait for the next round to begin. We stress that an honest party will poll the network exactly once per round while a compromised party might poll the network more often. We assume that every party will initialise and

The 5-phase paradigm for modeling protocols

1. "Input": Accept input by the environment $\mathcal{Z}$.
2. "Fetch": Request input from the network $\mathcal{F}_{\mathsf{net}}$.
3. "Send": Send output to the network $\mathcal{F}_{\mathsf{net}}$.
4. "Output": Produce output to the environment $\mathcal{Z}$.
5. "RoundOK": Signal "RoundOK" to $\mathcal{F}_{\mathsf{clock}}$.

Figure 3.14: We suggest a 5-phase paradigm to model computer networks in the UC framework.

update a round counter and further maintain state for the number of activations per round and whether (RoundOK) has already been signaled. This requires sending $\mathcal{F}_{\mathsf{clock}}$ a (RequestRound) request on activation and accordingly updating state information, but imposes no limitations for the party.

The Wrapper Functionality

To simplify the definition of ideal functionalities, we introduce an ideal "wrapper" functionality $\mathcal{F}_{\mathsf{wrap}}$ (see Figure 3.15). It "wraps around" the ideal functionality and moderates its communication with the dummy parties in the ideal world. Its main task is to count activations of dummy parties. Since honest parties adhere to the 5-phase paradigm, it will only notify the ideal functionality if the environment gives input to a party (during the first activation), if a party could create output in the real model (during its fourth activation), and when a round is complete. It also ensures that the adversary is activated at least as often as in the real model.

Specifying Ideal Functionalities

The tools introduced above allow for a natural description of ideal functionalities. $\mathcal{F}_{\mathsf{wrap}}$ will send a notification for important events (e.g. inputs, outputs and round changes) and the ideal functionality reacts to them appropriately. Specifically, the ideal functionality will not be required to count activations itself or activate the adversary sufficiently often. Since the network functionality provides a bound for the maximum delay a channel can have, it is also easily possible to model availability. The ideal functionality only has to maintain a counter corresponding to the delay δ of the channel for each packet and reduce this counter by one every time a round is complete. When the counter reaches zero, the packet can be output immediately when output is requested by $\mathcal{F}_{\mathsf{wrap}}$. Since all honest parties will poll for new packets once per round the adversary can delay a packet

The wrapping function for ideal functionalities $\mathcal{F}_{\mathsf{wrap}}$

Maintain an activation counter c_p for each of the honest dummy parties. Relay all communication from $\mathcal{F}_{\mathsf{ideal}}$ directly to the environment. Upon activation by the environment, i.e. upon receiving input m through a dummy party p:

- If $c_p < 5$ increase the activation counter of the party.
- If $c_p = 1$ send message (input, m, p) to $\mathcal{F}_{\mathsf{ideal}}$.
- If $c_p = 2$ or $c_p = 3$, send message $(\mathsf{activated}, \mathsf{p})$ to the adversary.
- If $c_p = 4$ send message (output, p) to $\mathcal{F}_{\mathsf{ideal}}$.
- If $\forall p' : c_{p'} = 5$ reset all activation counters and send $(\mathsf{RoundComplete})$ to $\mathcal{F}_{\mathsf{ideal}}$.

Figure 3.15: The ideal "wrapper" functionality $\mathcal{F}_{\mathsf{wrap}}$. Its purpose is to capture redundant formalisms by counting activations of protocol parties and notifying the ideal functionality of important events.

delivery for a maximum of δ rounds per channel.

Note that we only specify the behaviour for input by honest parties. We implicitly assume that messages from the adversary to corrupted parties or vice versa are delivered immediately.

3.7.2 Example: Byzantine Generals

As an example, we will use the presented methodology to model a popular example from the literature: the Byzantine Generals problem. We will then restate a popular result concerning this problem by giving a proof in our framework.

The Byzantine Generals Problem

The Byzantine Generals problem was first introduced by Lamport, Shostak, and Pease [148]. The motivation is as follows: suppose that a commanding general wants to give orders (for the sake of simplicity he will only use "attack" or "retreat") to his lieutenants but he does not know which of them are trustworthy. Also, the lieutenants do not know whether the general himself is trustworthy. Now suppose that each of the participants can communicate with each other participant via "oral" messages. The Byzantine Generals problem is to find an algorithm that, given a number of parties n (one of them is the general), ensures that:

1. All loyal lieutenants obey the same order, and

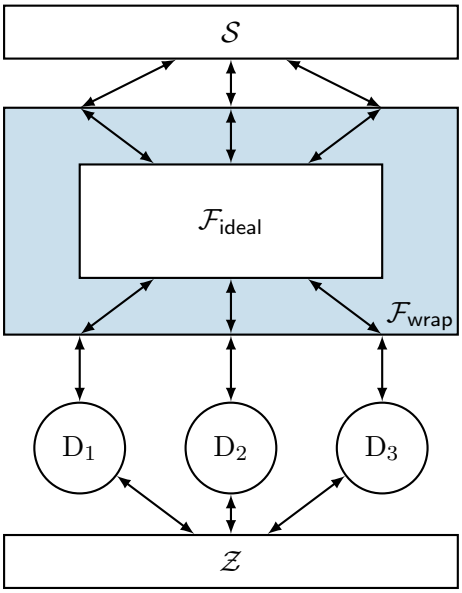

Figure 3.16: The ideal "wrapper" functionality $\mathcal{F}_{\text{wrap}}$ acts as a relay between the dummy parties and the ideal functionality. It counts activations of parties and notifies the ideal functionality of important events like round changes, thus simplifying the formulation of ideal functionalities.

2. If the general is loyal, then every loyal lieutenant obeys the order he sends.

Note that a disloyal (corrupted) lieutenant can arbitrarily lie about messages he received and try to deceive other lieutenants. He can also refuse to send any messages. However, it is assumed that loyal parties will notice when messages are missing. Lamport et al. [148] show that there can not be a generic solution to the problem for three parties, but there is a solution for four parties. We will now model the Byzantine Generals problem with four parties according to our methodology and give a formal security proof for a specific solution to the problem.

Modeling the Byzantine Generals Problem

The network in this example is fully connected. Every party can transmit messages to every other party. There is a maximum latency of 2δ until a packet is output by one of the parties: a possible delay of δ from the general to the lieutenants and another possible delay of δ for a packet from one lieutenant to reach the others.

The Byzantine Generals problem statement implies that a party notices if it will not receive any message from another party anymore so that it will not wait indefinitely. In reality this is usually realised by timeouts—we will use the same mechanism here.

Protocol π_{byz} implements a solution to the generals problem. The ideal network functionality allows for a maximum delay of δ for each message and messages have to be

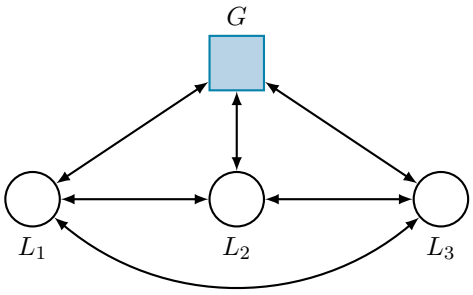

Figure 3.17: The network graph $\mathsf{byz} = (V, E)$ for the Byzantine Generals problem with $V = \{G, L_1, L_2, L_3\}$ and $E = V^2$. It is fully connected—each party can communicate with every other party.

sent from the general first and from the lieutenants afterwards. Thus a party will assume a timeout after 2δ rounds.

Definition 4 (A solution to the Byzantine Generals problem with four parties π_{byz})**.**

- Party G: Maintain a local round counter r.
  1. "Input": Upon first activation this round and input m by $\mathcal{Z}$, save m and ignore further inputs.
  3. "Send": Upon third activation, call $\mathcal{F}_{\mathsf{net}}^{\mathsf{byz}}(\mathsf{send}, (\mathrm{L}_1, m), (\mathrm{L}_2, m), (\mathrm{L}_3, m))$ if m was saved.
  5. "RoundOK": Upon fifth activation, send $(\mathsf{RoundOK})$ to $\mathcal{F}_{\mathsf{clock}}$.
- Party L_n: Maintain a local round counter r.
  2. "Fetch": Upon second activation,
     - call $\mathcal{F}_{\mathsf{net}}^{\mathsf{byz}}(\mathsf{fetch}, \{\mathrm{G}, \mathrm{L}_k, \mathrm{L}_j\})$ for $k \neq j \neq n$. If the call was successful, save the messages for the corresponding parties.
  3. "Send": Upon third activation,
     - if there is a message m by party G which has not been broadcast yet, broadcast it: call $\mathcal{F}_{\mathsf{net}}^{\mathsf{byz}}(\mathsf{send}, (\mathrm{L}_k, m), (\mathrm{L}_j, m))$ with $k, j \neq n$.
  4. "Output": Upon fourth activation,
     - if $r < 2\delta$ and there are two identical messages m from two different parties (other than G), output m. If there are three different messages from the different parties, output the message from party 1;
     - if $r = 2\delta$ output $\mathsf{retreat}$.
  5. "RoundOK": Upon fifth activation, send $(\mathsf{RoundOK})$ to $\mathcal{F}_{\mathsf{clock}}$.

Figure 3.18 shows the corresponding ideal functionality. This functionality fulfills the requirements for a solution to the Generals problem given earlier.

We will now show that this protocol realises the ideal functionality.

The ideal functionality of the Byzantine Generals problem with four parties $\mathcal{F}^{\delta}_{\mathsf{byz\text{-}ideal}}$.

Upon initialisation store a delay value $d := (2\delta)$ and initialise three variables $m_{L_1} := \bot, m_{L_2} := \bot, m_{L_3} := \bot$.

- Upon receiving message $(\mathsf{input}, m, \mathrm{G})$ from $\mathcal{F}_{\mathsf{wrap}}$ and if G is honest: store $m_{L_p} := m$ for $p \in \{1, 2, 3\}$ and send $(\mathsf{input}, m, \mathrm{G})$ to the adversary.
- Upon receiving message $(\mathsf{set}, m_1, m_2, m_3)$ from the adversary and if G is corrupted: if $m_{L_1} = \bot, m_{L_2} = \bot, m_{L_3} = \bot$, and there are two identical messages m_i, m_j with $i \neq j$, set $m_{L_1}, m_{L_2}, m_{L_3} := m_i$, else set $m_{L_1}, m_{L_2}, m_{L_3} := m_j$ where j is the smallest index for which $m_j \neq \bot$.
- Upon receiving message $(\mathsf{output}, p_1, p_2, p_3)$ from the adversary: mark messages m_{p_1},m_{p_2},m_{p_3} as ready for output.
- Upon receiving message (output, p) from $\mathcal{F}_{\mathsf{wrap}}$:
  - If $d = 0$: output $\mathsf{retreat}$ to p.
  - if $d \neq 0$ and if m_p is marked as ready for output, output m_p to p.
- Upon receiving message $(\mathsf{RoundComplete})$ from $\mathcal{F}_{\mathsf{wrap}}$, decrease d by 1 and send $(\mathsf{RoundComplete})$ to the adversary.

Figure 3.18: The ideal functionality of the three generals problem. If the general is honest, all honest parties will obey his order. If he is corrupted, all parties will obey the same order. As in the real protocol the adversary can not delay the output for more than 2δ rounds.

Theorem 6. *π_{byz} realises $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$ in the $\mathcal{F}_{\mathsf{net}}^{\mathsf{byz},\delta}$-hybrid model.*

Proof. We prove the theorem by giving a step-wise transformation from the real model to the ideal model. We argue that the individual transformation steps are indistinguishable for the environment, and thus, by the transitivity of indistinguishability, the real model is indistinguishable from the ideal model. Start with the real protocol.

Regroup all parties into a new machine $\mathcal{S}$. The adversary simulator $\mathcal{S}$ will simulate the real network in all transformation steps. Introduce dummy parties D_{G}, D_{L_1}, D_{L_2}, D_{L_3} for all protocol parties and relay messages from and to $\mathcal{Z}$ appropriately. Introduce a new machine $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$. Route all communication from the dummies to $\mathcal{S}$ and vice versa through $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$. The regrouping of parties is indistinguishable for the environment. In the upcoming transformation steps, we will gradually expand $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$'s functionality:

1. Initialise variables m_{L_1}, m_{L_2}, and m_{L_3}. When receiving a message m from dummy

party G, set $m_{L_1} := m$, $m_{L_2} := m$ and $m_{L_3} := m$. Also initialise and save a round counter $d := 2\delta$. This modification is indistinguishable, since it only stores information and does not alter the communication.

2. If G is corrupted, accept a message $(\mathsf{set}, m_1, m_2, m_3)$ from $\mathcal{S}$. Check if there are $i \neq j$ such that $m_i = m_j$. If so, set $m_{L_1}, m_{L_2}, m_{L_3}$ to m_i. Else set $m_{L_1} = m_1, m_{L_2} = m_2, m_{L_3} = m_3$. This modification again only stores information.

3. When $\mathcal{S}$ attempts to pass output m from an uncorrupted party p in the simulation back to the dummy party, only allow it to pass through $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$ if either

   a) m has been stored as m_p in $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$, or

   b) the message is $\mathsf{retreat}$.

   We have to argue the indistinguishability of this modification. A real protocol party will only output a message other than $\mathsf{retreat}$ when it has received two identical messages. This will only happen if

   a) G is honest—then, m will have been provided by $\mathcal{Z}$ through dummy party G and thus saved for every party in the ideal functionality, or

   b) G is corrupted and sent two identical messages. In this case, $\mathcal{S}$ will have used the set-message to provide these messages and they will also have been saved for every party.

4. Introduce $\mathcal{F}_{\mathsf{wrap}}$ as a wrapper around $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$. For each notification that a round is complete from $\mathcal{F}_{\mathsf{wrap}}$ decrease the delay value d and notify $\mathcal{S}$ that the round is complete. $\mathcal{F}_{\mathsf{wrap}}$ will not notify $\mathcal{S}$ about activations in phase 4 ("output"), but $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$ instead. The simulator is thus not able to accurately simulate the exact order of outputs. However, the simulator is still able to determine the set of messages to output for each party in each round: he still is notified about the input to the protocol, when a party sends a message, and when a round is complete. We alter the strategy of $\mathcal{S}$ to make the modification indistinguishable: in each round, observe which parties will output a message and notify the ideal functionality that these parties are ready for output. Now, when $\mathcal{Z}$ activates a party and expects output, the ideal functionality will output possible messages for that specific party. This allows for all messages other than $\mathsf{retreat}$ to be output correctly. So, if $d = 0$ after the fourth activation of a party, $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$ just outputs $\mathsf{retreat}$, mimicking the behaviour in the real model. $\mathcal{F}_{\mathsf{byz\text{-}ideal}}$ and $\mathcal{S}$ now behave as specified in the ideal model, perfectly emulating the real model.

This concludes the proof. □

3.8 Firewalls Revisited

In this section, we improve upon our previous results. We already showed that a quorum of three firewalls realises a secure firewall under the condition that at most one firewall is

corrupted. Our previous analysis lacks an availability guarantee though. We prove this guarantee for the construction in the now-improved model. First, we briefly restate the construction.

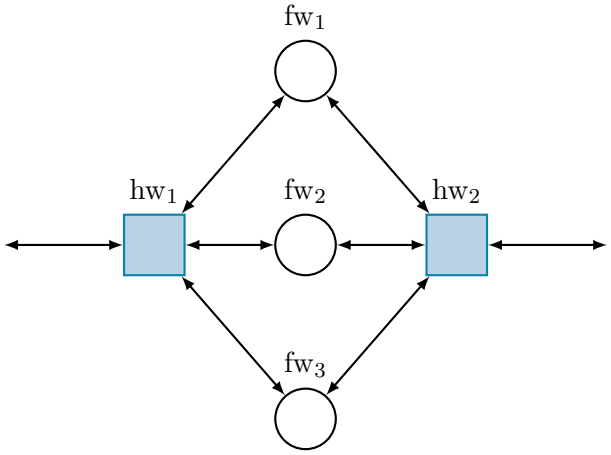

Figure 3.19: The three-firewall network. The graph directly serves as the network model for $\mathcal{F}_{\mathsf{net}}^{G}$: $G = (V, E)$ with $V = \{hw_1, hw_2, fw_1, fw_2, fw_3\}$ and $E' = \{(hw_1, fw_1), (hw_1, fw_2), (hw_1, fw_3), (hw_2, fw_1), (hw_2, fw_2), (hw_2, fw_3)\}, E = E' \cup \{(v, u) \mid (u, v) \in E'\}$.

Definition 5 (The functionality of an ideal firewall $\mathrm{F}_{\mathrm{fw}_j}$)**.**

$$\mathrm{F}_{\mathrm{fw}_j} : P \times V \times S \to (P \cup \bot) \times (V \cup \bot) \times S$$

$$F_{\mathrm{fw}_j}(p, v, s) = \begin{cases} (p', v', s') & \text{if output is generated,} \\ (\bot, \bot, s') & \text{else.} \end{cases}$$

Definition 5 provides a modified definition of the firewall function from Section 3.6, adapted to work with our graph based network model (Figure 3.13), using the network graph presented in Figure 3.19. The function accepts a packet p, a node from the network graph v and a state s and outputs another packet, another node (the receiver of that packet) and a new state.

Protocol π_{fw} (Definition 6) realises the three firewall solution as expressed using our tools. Figure 3.20 shows the corresponding ideal functionality.

Definition 6 (Protocol π_{fw})**.**

- party $\mathrm{hw_k}$:
  1. "Input": Upon the first activation by message (input, m) from $\mathcal{Z}$, save m.
  2. "Fetch": Upon the second activation by message (output) from $\mathcal{Z}$,

- call $\mathcal{F}_{\mathsf{fw\text{-}net}}(\mathsf{fetch}, \{\mathrm{fw}_1, \mathrm{fw}_2, \mathrm{fw}_3\})$, save the message m corresponding to fw_i as (m, i);
- if there are two entries (m, i) and $(-m, i)$ on the tape, delete both.

3. "Send": Upon the third activation by message (output) from $\mathcal{Z}$, call $\mathcal{F}_{\mathsf{fw\text{-}net}}(\mathsf{send}, (\mathrm{fw}_1, m), (\mathrm{fw}_2, m), (\mathrm{fw}_3, m))$ if m was saved previously. Delete m.
4. "Output": Upon the fourth activation by message (output) from $\mathcal{Z}$, if there are two saved entries (m, i) and (m', i') with $m \equiv m'$ and $i \neq i'$: delete both messages and output m. If $i, i' \neq 1$, save $(-m, 1)$, else if $i, i' \neq 2$, save $(-m, 2)$, else if $i, i' \neq 3$, save $(-m, 3)$.
5. "RoundOK": Upon the fifth activation by message (output) from $\mathcal{Z}$, send ($\mathsf{RoundOK}$) to $\mathcal{F}_{\mathsf{clock}}$.

- party fw_k:

2. "Fetch": Upon the second activation by message (output) from $\mathcal{Z}$,
   - call $\mathcal{F}_{\mathsf{fw\text{-}net}}(\mathsf{fetch}, \mathrm{hw}_1, \mathrm{hw}_2)$ and save the message m corresponding to hw_i as (m, i);
   - for all saved messages (m, i): compute $F_{\mathrm{fw}_k}(m, i, s) = (m', i', s')$ and replace that (m, i) with (m', i').
4. "Output": Upon the fourth activation by message (output) from $\mathcal{Z}$, if there are two messages (m, i) and (m', i'), call $\mathcal{F}_{\mathsf{fw\text{-}net}}(\mathsf{send}, (\mathrm{hw}_i, m), (\mathrm{hw}_{i'}, m'))$.
5. "RoundOK": Upon the fifth activation, send ($\mathsf{RoundOK}$) to $\mathcal{F}_{\mathsf{clock}}$.

Theorem 7. *π_{parallel} realises $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$ in the $\mathcal{F}_{\mathsf{net}}^{\mathsf{fw},\delta}$-hybrid model.*

Proof. We prove the lemma via a series of transformations, starting from the real model. In each step we will modify the ideal functionality and argue that the modification is indistinguishable. We will w.l.o.g. assume that fw_3 is corrupted. Encapsulate the network in a new machine $\mathcal{S}$, introduce dummies for all fw_i and hw_i, and construct a new machine $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$ which connects the dummy machines with their counterparts in the (now simulated) real network. Modify $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$ step-wise:

1. Introduce variables to keep state for the firewalls. When receiving (input, m) through hw_k, evaluate the firewall functionalities $\mathrm{F}_{\mathrm{fw}_1}$ and $\mathrm{F}_{\mathrm{fw}_2}$, update the respective firewall states and save the output packets p_1 and p_2 in a list Q_k as $(\mathsf{in}, 1, p_1, 2\delta)$ and $(\mathsf{in}, 2, p_2, 2\delta)$. This modification stores additional information but does not alter the communication and is thus indistinguishable.

2. When being advised to output a message p for a party hw_k by the simulator, only do so if there is an entry (in, i, p, d) in Q_k and delete that entry. Every message scheduled by the simulator in this manner was output by one of the firewalls in its simulation. Consequently, this message is also stored in Q_k. The real protocol party fw_k will internally delete all messages it outputs. Thus, this modification is indistinguishable.

The ideal functionality of the firewall architecture $\mathcal{F}_{\mathsf{fw\text{-}ideal}}^{\delta}$

Maintain a list of scheduled packets for each direction: $\mathsf{Q}_1, \mathsf{Q}_2$. Let w.l.o.g. fw_3 be the corrupted party. In each case, if there are multiple entries to choose from, pick the first.

- Upon receiving $(\mathsf{input}, m, \mathrm{hw}_k)$ from $\mathcal{F}_{\mathsf{wrap}}$: Compute the firewall functions and update the internal states. Let the outputs of F_{fw_1} and F_{fw_2} be p' and p''. Store $(\mathsf{in}, 1, p', 2\delta)$ and $(\mathsf{in}, 2, p'', 2\delta)$ in Q_k if there is no entry $(\mathsf{missing}, 1, p', 0)$ or $(\mathsf{missing}, 2, p'', 0)$ respectively. Send $(\mathsf{input}, m, \mathrm{hw}_k)$ to the adversary.
- Upon receiving $(\mathsf{output}, \mathrm{hw}_k)$ from $\mathcal{F}_{\mathsf{wrap}}$:
  - If there are two entries $(\mathsf{in}, 1, p', 0)$ and $(\mathsf{in}, 2, p', 0)$ in Q_k, erase the corresponding entries from the queue and output p' to hw_k.
  - Else: if there is an entry $(\mathsf{deliver}, i, p, d)$ in Q_k remove it. Check if there is another entry (in, i', p, d') in Q_k with $i \neq i'$. If so, remove that entry too, if not, add an entry $(\mathsf{missing}, |i-3|, p, 0)$ to Q_k.
- Upon receiving $(\mathsf{RoundComplete})$ from $\mathcal{F}_{\mathsf{wrap}}$: Replace each entry (in, i, p, d) (or $\mathsf{deliver}, i, p, d$) with $d > 0$ in Q with $(\mathsf{in}, i, p, d-1)$ (or $(\mathsf{deliver}, i, p, d)$ and send $(\mathsf{RoundComplete})$ to the adversary.
- Upon receiving $(\mathsf{output}, \mathsf{p}, \mathrm{hw}_\mathsf{k})$ from the adversary: if there is an entry (in, i, p, d) in Q_k, replace it by $(\mathsf{deliver}, i, p, d)$.

Figure 3.20: The parallel firewall network expressed using the newly-introduced tools. Parties hw are responsible for distributing incoming and merging outgoing packets. They will output a packet to the environment not more than once per round.

3. When a packet p is output based on any entry $(\ldots, i, p, d)$ in Q_k, check if there is another entry $(\ldots, j, p, d)$ with $i \neq j$. If so, delete that entry as well. If not, add an entry $(\mathsf{missing}, |i-3|, p, d)$ to Q_k. Further, when receiving (input, m) through hw_k and evaluating the firewall functionalities, before saving the resulting packets p_1 and p_2, check if there is an entry $(\mathsf{missing}, 1, p_1, 2\delta)$ or $(\mathsf{missing}, 2, p_2, 2\delta)$ in Q_k. If there is, remove that entry and do not save the resulting packet. This modification is indistinguishable as $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$ now implements the exact behaviour of hw_1 and hw_2.

4. Add $\mathcal{F}_{\mathsf{wrap}}$ as a wrapper around $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$. When receiving $(\mathsf{RoundComplete})$ from $\mathcal{F}_{\mathsf{wrap}}$, decrease the delay value d of each entry in Q_1 and Q_2 by 1. Send $(\mathsf{RoundComplete})$ to the simulator. When being advised to output a packet p for party hw_k by the

simulator, instead of outputting the packet immediately, replace the corresponding entry in Q_k by $(\mathsf{deliver}, i, p, d)$. When being asked to provide output for party hw_j by $\mathcal{F}_{\mathsf{wrap}}$, check if there is an entry in Q_j with $d = 0$. If so, output that packet. If not, check if there is an entry marked for delivery. If so, output the corresponding packet. Always perform the output according to the mechanism described in Step 3.

The simulator's simulation of the real network is not perfect after transformation step 4. Concretely, $\mathcal{S}$ is not notified of the fourth activation ("output") of honest protocol parties. However, as we argued in the proof of Theorem 6, the output *decision* is made during prior activations. Hence, by $\mathcal{S}$ announcing output early to $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$, $\mathcal{S}$ and $\mathcal{F}_{\mathsf{fw\text{-}ideal}}$ perfectly emulate the real protocol. ($\mathcal{F}_{\mathsf{wrap}}$ delivers output after the fourth activation only.) □

3.9 Implementing the Setup Assumption

Setup assumptions help construct protocols in the UC framework. For many assumptions it is not immediately clear why they are reasonable though. In our constructions, we assumed a trusted packet comparator that realises a quorum decision for network packets. To demonstrate the practicality of this assumption, we first implemented a proof-of-concept demonstrator. A particularly efficient realisation has later been developed in the context of a Bachelor thesis [7].

3.9.1 Proof-of-Concept Demonstrator

To demonstrate the feasibility of a trusted packet comparator Jochen Rill implemented a proof-of-concept realisation of the trusted hardware device specified in Figure 3.2. The implementation uses the Linux netfilter "queue" Application Programming Interface NFQUEUE[1] to pass IP packets from the kernel to a Perl script (Figure 3.21). The Perl script then implements the actual comparator logic in a total of 89 lines, including empty lines and comments. This setup was deployed on a Ubuntu 13.10 operating system. It can handle a throughput of about $10\,\mathrm{Mbit\,s^{-1}}$.

The prototype was presented at the CeBIT, the world's largest computer expo in Hanover, Germany, in March 2014. It was presented alongside a computer animation that explains the concept on the joint booth of the Karlsruhe Institute of Technology (KIT) and the FZI Forschungszentrum Informatik (see Figure 3.22). The concept animation was also presented on the booth of the TU9, the alliance of leading Institutes in Germany, on the Hannover Messe 2015.

3.9.2 Efficient Realisation in Intel DPDK

In a cooperation with the Institute of Telematics at the KIT we investigated an efficient implementation of the packet comparator in the course of a Bachelor thesis [7].

[1] `http://netfilter.org/projects/libnetfilter_queue/`

```
sub cb()
{
  if((time()-$lasttime)>$timeout){
     # Timeout reached, delete all packets
     %map = ();
     $lasttime = time();
  }
  my ($dummy,$payload) = @_;
  if ($payload) {
    my $ip_obj = NetPacket::IP->decode($payload->get_data());

    # IP fields src_ip, dest_ip, and data are
    # considered for comparison.
    my $hash = 0;
    $hash = sha1_hex($ip_obj->{src_ip}.$ip_obj->{dest_ip}.$ip_obj->{data});

    if(defined $map{$hash}){
      # Packet with hash $hash arrived
      # previously on interface $map{$hash}[0]
      if(!($map{$hash} eq $payload->get_indev())){
          # Same packet arrived on a different interface,
          # allow it and discard the original packet
          $payload->set_verdict($nfqueue::NF_ACCEPT);
          delete $map{$hash};
      }
    } else {
      # New packet arrived on interface $payload->get_indev()
      # with hash $hash
      $payload->set_verdict($nfqueue::NF_DROP);
      @{$map{$hash}} = $payload->get_indev();
    }
  }
}
```

Figure 3.21: Source code of the packet comparator for a proof-of-concept realisation, authored by Jochen Rill, annotated for better readability. The original source file has a total of 89 lines.

Intel's Data Plane Development Kit (DPDK)[2] was used to implement the packet comparator from Figure 3.2. Later, the updated functionality from Figure 3.10 was implemented. The DPDK is a framework aimed at an efficient processing of network packets by minimising overhead.

The design goal was to implement a packet comparator that performs at a wire speed of 10 Gbit s^{-1}. The assigned machine for performance evaluation runs an Intel® Core™ i7 Central Processing Unit at a clock speed of 3.4 GHz. As real network firewalls have different latencies when evaluating network traffic, early estimations suggested that a maximum of 2400 IP packets needed to be cached for the quorum decision to have merit—when the packet cache runs full, packets have to be cleared in order to make room for new incoming packets. The amount of memory reserved for temporarily caching

[2] http://dpdk.org/

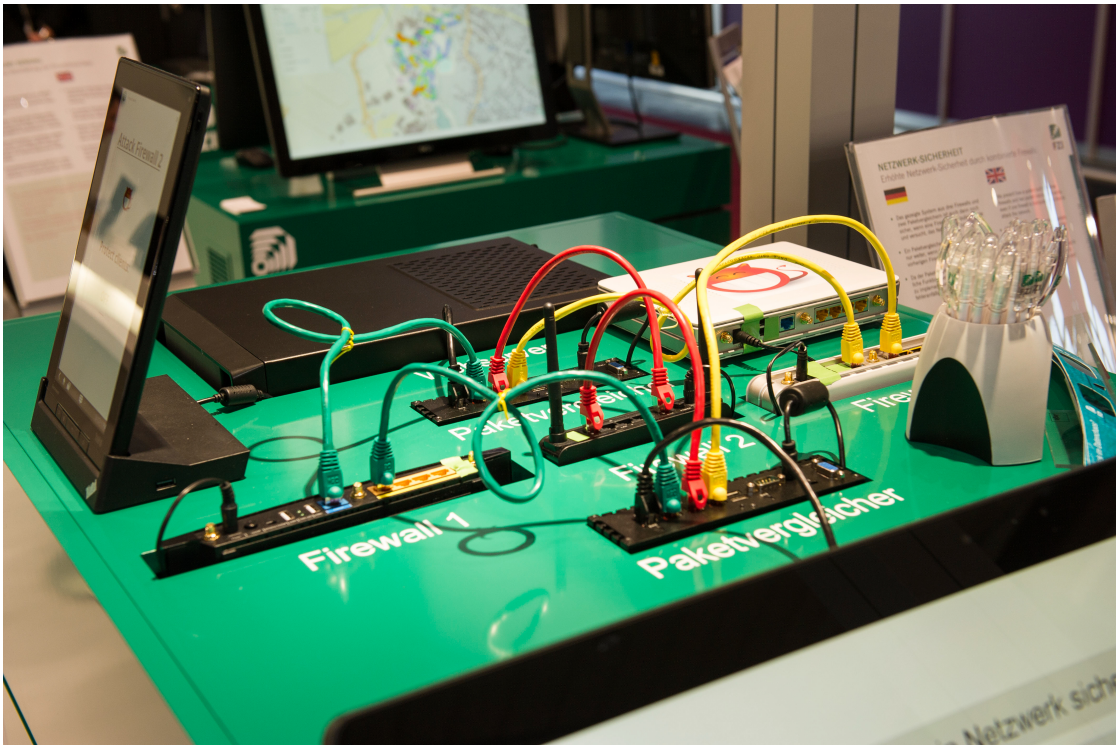

Figure 3.22: The proof-of-concept packet comparator, as it was presented on the joint booth of the KIT and the FZI Forschungszentrum Informatik on the CeBIT 2014. Image © Markus Breig/KIT.

network packets was 2 GiB.

The architecture was designed as follows: Three ring buffers store incoming packets. They are then stored, compared, and a quorum decision is made. Packets that pass the decision process are output to an output ring buffer. IP packets are stored in a hash table. The values stored are: a hash value of the payload (see below), a pointer to the packet in memory, the network interfaces the packet arrived on, their number, and a timestamp.

A bit-by-bit comparison of packets can have a large impact on the performance of the comparator. As a remedy, a hash value is computed which then facilitates a quicker comparison. However, an evaluation of the performance of various hash operations revealed that cryptographic hash functions impose much overhead: While a bit-by-bit comparison and a CRC32 [136] checksum have a performance "cost" of less than 0.7 cycles per Byte, the SHA-1 [79] hash needs more than 6 cycles per Byte. The Skein hash [81] weighs in at more than 1.5 cycles per Byte. However, the CRC32 checksum is not a cryptographic hash operation and thus lends itself to a collision attack by an adversary (see Section 2.4.3). While it is not immediately clear how such a vulnerability influences security, it is a clear deviation from the model. In our model, we assume an equivalency relation $\equiv$ for packets. The model is not designed to allow for collisions.

Consequently, using the CRC32 checksum for packet comparisons invalidates the security guarantees provided by a proof in the model. We point out that this is an instance where an implementation decision can subtly invalidate a formal security analysis.

Consequently, the packet comparison was implemented as follows. In a first step, the CRC32 checksums of the packets are compared. Then, if the checksums match, a bit-wise comparison is performed. This way, the performance of the comparison was optimised without sacrificing security. Indeed, as performance evaluations showed, the packet comparison has the largest influence on performance.

Concluding, the implementation can only cope with roughly 90% of traffic at a wire speed of $10\,\mathrm{Gbit\,s^{-1}}$. It is well suited for wire speeds of $1\,\mathrm{Gbit\,s^{-1}}$, however.

4 Data Outsourcing

> The million dollar payroll check or the threatening letter for an already settled bill can be laughed off for a while as the growing pains of adolescence. It will be less funny if the computer is allowed to become an intrusion on our privacy or a pollutant of our lives.
>
> *(Giles E. Hemmings, Foreword of James Martin's* Security, Accuracy, and Privacy in Computer Systems*, 1973)*

The Information Technology (IT) industry has seen numerous trends in the last two decades, yet a majority of them seem to revolve around a common paradigm shift: Data is stored less and less locally, but is outsourced to a data processing centre and accessed over the Internet. To retrieve parts of the outsourced data, clients submit queries to the servers which then execute them and return the result. Applications that fit into this paradigm are not restricted to relational databases, but to any information that can be structured meaningfully—be it searchable documents, relational databases, or image archives. Data outsourcing promises significant advantages, particularly for organisations that do not specialise in data management. But entrusting private data to a service provider also introduces a security risk. While an IT service provider can be trusted to provide the contractually agreed-on services faithfully, one can easily imagine that a curious employee might try to learn confidential information from the outsourced data. Cryptographic methods promise to secure confidentiality in such a scenario.

Many cryptographic schemes have a custom security notion. While a tailored security notion helps accurately express a scheme's security properties, it makes comparing security properties difficult. This work introduces a framework that allows for modelling the privacy guarantees of data outsourcing schemes on a common level of abstraction.

We identify three conceptually different privacy goals: keeping the outsourced data itself private, keeping the queries to the data private, and keeping the result of the query private. Our results are applicable to constructions from seemingly disparate fields of cryptographic research, e.g. private information retrieval, searchable encryption, and secure database outsourcing.

We show that data privacy and query privacy are independent concepts, while result privacy is consequential to them. What is more, we found that many existing privacy notions, e.g. for searchable encryption, mix these concepts.

In general, strong privacy can be bought with communication and computation complexity or a cost-inefficient distribution among many servers. Such schemes find no use in practice, however. On the other hand, existing formal security notions are often not applicable to schemes used in practice. Thus, we want to design security notions applicable to both practical (i.e. $O(\log(n))$ complexity, single server, single client) and highly elaborate (i.e. PIR, ORAM) outsourcing schemes.

Therefore our framework allows to independently specify bounds for the level of privacy for queries, the outsourced data, and the query result. We showcase the applicability of our formal notions by expressing existing notions in our framework.

This chapter is based on a Diploma thesis [6], a term paper ("Studienarbeit") [8], and two conference papers [1, 5]. Section 4.6 is based on an unpublished extended version of a conference paper [5].

4.1 Related Work

We distinguish between data privacy, query privacy, and result privacy, the latter implying the former two. Therefore, we divide the related work concerning security notions for data outsourcing schemes into three categories: notions which only consider the privacy of the outsourced data, notions which only consider the privacy of the queries, and those which intertwine both.

Security Notions for Data Privacy Many data outsourcing schemes only consider the privacy of the outsourced data in both static and adaptive settings. There are game-based notions [13, 78, 89, 113], simulation-based notions [48, 49, 125], and notions that use the Universal Composability framework [41, 138, 139, 190]. A well-known example for an adaptive security notion is IND-CKA established by Goh [89]. The intuition is that an adversary should not be able to distinguish two sets of data of his choosing based on the generated index even if he can issue and observe queries. However, in Goh's notion, the queries the adversary can choose are strongly restricted: he is not allowed to query for words that are exclusive two one of the two sets he chooses as challenge.

An example for a notion which only considers static security is Huber et al.'s IND-ICP [113]. Here, the idea is that an adversary should not be able to distinguish the encryptions of two databases. However, the databases the adversary is challenged on are restricted to being independent permutations of one another.

Security Notions for Query Privacy Hiding queries on outsourced data on a single server has been studied in the context of Single-Server Private Information Retrieval (PIR) [57, 69, 86]. The PIR notion requires that an adversary who observes access patterns cannot distinguish any two queries. PIR does not guarantee that the data itself is kept private [57]. There are PIR schemes in literature with a sublinear *communication complexity* [38, 87, 140]. However, all PIR schemes inherently have a *computational complexity* for the server which is linear in the size of the data [185]. Kantarcıoğlu and Clifton also propose a security notion that addresses hiding queries to an outsourced

database [128]. They show that a scheme that meets their security definition must scan the entire database non-negligibly often to execute queries. Asonov and Freytag get around this limitation by using a secure coprocessor [17].

The privacy of queries on data has also been investigated in the context of Oblivious Random Access Machines (ORAMs) first introduced by Goldreich and Ostrovsky [92] and further explored and improved upon by others [66, 174, 183]. Similar to PIR, an "oblivious" Random Access Machine (RAM) is one that cannot distinguish access patterns—the data itself is not required to be private. As is the case with PIR, all ORAM constructions can not be considered efficient in our sense. They either have polylogarithmic computation cost while requiring the client to store a constant amount of data [183] or have logarithmic computation cost, but require the client to store at least a sublinear amount of data dependent on the size of the RAM [96]. Therefore, the security notion for ORAM is not suitable for our cause.

Security Notions for Data Privacy as well as Query Privacy There are security notions in the literature which consider both data privacy as well as query privacy. Chase et al. [52–54] introduce the simulation-based notion of "chosen query attacks" which models both the privacy of queries and that of the data. However, in their notion, the concepts of privacy for data and privacy for queries are intertwined. In our original publication of the work presented in Section 4.6 [5] we try to separate both properties: we introduce the notion of "data privacy" and complement it with "pattern privacy", which is similar to PIR. Blass et al. [27] propose a scheme for word search that works uses the MapReduce framework. They distinguish Storage Privacy and Query Privacy. The simulation-based security definition put forward by Chase and Shen [52–54] also considers the privacy of both data and queries.

Modeling Information Leakage A reoccurring pattern in security notions for practical schemes is the use of a *leakage function* which describes the information the scheme leaks to the adversary during execution. A certain amount of leakage seems necessary in order for schemes to be efficient. Cash et al. investigate the construction of efficient and practical schemes that also have a formal security analysis [48, 49]. Their analyses follow a simulation-based approach. The constructions leak information about the plaintext and the query which they explicitly model by a *leakage function* $\mathcal{L}$. This is similar to Chase et al. [52–54], whose notion allows to describe the information that leaks through the encryption itself ($\mathcal{L}_1$) and the information about the ciphertext *and* the queries combined that is leaked by evaluating queries ($\mathcal{L}_2$). Stefanov et al. [190] employ the same technique in the Universal Composability Framework. In game-based notions such leakage is modelled by restricting the challenges the adversary can choose. Thus, in our framework we define "leakage relations" that model information leakage.

Searchable Encryption In the literature, there are two general approaches to searchable encrypted data: Symmetric Searchable Encryption (SSE) [52, 63, 89, 95, 125, 126, 138] and Public Key Encryption with Keyword Search (PEKS) [10, 20, 35, 50, 111, 171]. In

almost all cases these approaches are keyword-based. Keyword-based schemes generally allow arbitrary keywords to be stored in an index and not only keywords contained in the actual document. Furthermore, they allow indices to be created for arbitrary documents, not just strings. On the other hand, keyword-based schemes don't support substring search, exact pattern matching or return the number of occurrences of a substring. Also, keywords must be defined when constructing the index, whereas our approach only requires the pattern at search time. Xiaodong Song et al. [188] present a keywordless approach based on stream ciphers and supports pattern matching. However, the time required for performing a search scales linearly with the size of the document. Oleshchuck's construction [169] uses stream ciphers to achieve privacy-preserving pattern matching. He presents an "insecure solution", as well as "secure pattern matching". He provides no formal analysis. In their recent work, Chase and Shen [54] give a construction for a "queryable encryption" that uses suffix trees to achieve a pattern-matching search on encrypted data. They also achieve security against a malicious adversary. Their security definition is simulation-based and accounts for leakage.

There is a rich body of literature on exact pattern matching algorithms, going back to the late seventies. The general method for improving performance is precomputation. Approaches can be separated by where the precomputation occurs: There are algorithms that perform precomputation on the search pattern [21, 36, 129, 135] as well as algorithms that perform precomputation on the string [29, 30, 62, 156, 193]. Our scheme can be assigned to the latter. The aforementioned algorithms have been engineered for performance and efficiency alone and were not conceived in an adversarial scenario, however. Our scenario involves an honest-but-curious adversary and we therefore seek to hide as much information as possible from him.

Privacy Notions Data privacy has also been studied in the context of privacy notions for anonymised databases. A well-known example is the k-anonymity notion and related variants [150, 153, 192]. The idea behind k-anonymity is that any tuple in the anonymised database can be attributed to at least k individuals in the original database. These notions only define a property of an anonymised database but do not consider the process of querying the database. Newer approaches such as Differential Privacy [75] do model the anonymisation process, but not the the *querying* process—the adversary has access to the entire anonymised release. Privacy notions are thus static notions in our sense. In our scenario the adversary has access to the entire (encrypted) database, too, but tries to gain knowledge from a client querying a subset of the data.

4.2 A Model for Outsourced Data

Our basic object of interest is a *data set*—be it a database, an e-mail archive or a collection of images. One can execute *queries* on this data. The *result* of the execution of a query on a data set can be any function of the data.

We focus on queries that only return a function of the data they are executed on. We note, however, that updating data is also an area worthy of investigation.

In an outsourcing scenario, a *client* transfers the data to a *server* for storage. Before the data is transferred, the client *encrypts* it using a private key. Instead of executing queries locally, the client transforms them into an interactive *protocol* that it runs with the server. The client's input into the protocol is its private key, the server's input is the encrypted data. See Figure 4.1 for an illustration of our outsourcing model.

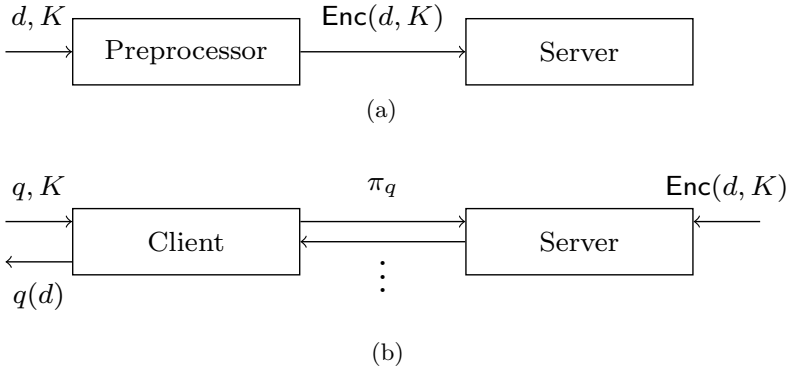

Figure 4.1: We model the interaction of the client with the server in two phases. In the initialisation phase (a) a preprocessing agent receives a data set d and an encryption key K, prepares the encrypted data $\mathsf{Enc}(d, K)$ and uploads it to the server. In the query phase (b) the client issues queries q using encryption key K by running protocol π_q with the server. The server's input is the encryption $\mathsf{Enc}(d, K)$. After the interaction the client outputs the query result $q(d)$.

We assume the server may be under adversarial influence. We restrict the adversary to honest-but-curious behaviour however—he may not deviate from the protocol as to interfere with its correct execution, but he may try to learn private information from the interaction.

4.2.1 Conventions

We say a protocol is *efficient* if its number of communication rounds does not exceed a fixed polynomial (in the security parameter). We denote the set of all efficient two-party protocols with Π. Our definitions can be extended to allow for the interaction of multiple servers with multiple clients. For the sake of clarity, we focus on the single-server-single-client case and leave a further generalisation of the definitions for future work.

We define our algorithms and protocols to operate on a *domain*. In this work, a domain is the set of all possible values in the given context—for example, in a database context, a domain would be the set of all databases. Our algorithms and protocols operate on three domains: a domain of the data to be encrypted (i.e. plaintexts) Δ, a domain of ciphertexts Γ, and a domain of results P.

4.2.2 Efficiency

We are particularly interested in capturing the security properties of *efficient* schemes within our framework. In practice, the size of outsourced data sets can easily measure terabytes. Since data is uploaded to the server only once while queries get executed repeatedly, it is not practical to use encryption schemes which process the whole data set on each query—or even a fraction of it. Therefore, we consider schemes efficient which have strictly logarithmic *communication and computation complexities* per query—for the client as well as for the server. This is in contrast to many schemes in the literature which are considered efficient even though they have a polynomial overhead.

4.2.3 Privacy

There are three privacy objectives: Keeping the data private, keeping the queries private, and keeping the results private.

Keeping the Data Private Bob runs an e-mail service. Alice uses the service and is concerned Bob might snoop through her private e-mail. *Data Privacy* guarantees that Bob does not learn anything about the content of Alice's e-mail.

Keeping the Queries Private In this scenario, Bob runs a patent search service. Clients can submit construction plans and Bob's service looks for any patents the construction is infringing. Alice is worried that Bob might learn her ideas and register them as patents himself. If the query protocol has *query privacy*, Bob cannot learn the content of Alice's requests.

Keeping the Results Private Bob also owns a database of DNA markers that are a sign of genetic diseases. He offers his customers the possibility to check blood samples against the database. Alice runs a clinic and is interested in Bob's services. She has no interest in disclosing to Bob whether her patients suffer from any genetic diseases. If the method of accessing Bob's database has *result privacy*, the results to Alice's requests are hidden from Bob. As we will show, result privacy implies database privacy as well as query privacy and vice versa.

4.3 Security Notions for Data Outsourcing Schemes

In this section we define precise terminology for securely outsourcing data and establish fundamental relations.

We now define the elementary concepts used in the rest of the chapter.

Definition 7. A *data set* $d \in \Delta$ is an element of a domain Δ. By $|d|$ we denote the length of its (unique) binary representation.

For example, in the scenario of an outsourced e-mail archive, Δ is the set of all mailboxes and a concrete data set (mailbox) $d \in \Delta$ is a set of e-mail messages. To

outsource data, one requires an algorithm that makes the data ready to be uploaded to a server. We call this process "encrypting" the data, as we will require later that no adversary can learn the original data from the outsourced data.

Definition 8. An *outsourcing scheme* for data sets is a tuple $(\mathsf{Gen}, \mathsf{Enc})$ such that for $c \in \mathbb{N}$,

$$\mathsf{Gen} : 1^k \to \{0,1\}^c$$
$$\mathsf{Enc} : \Delta \times \{0,1\}^c \to \Gamma$$

We call an outsourcing scheme for a data set *retrievable* if there is a function $\mathsf{Dec} : \Gamma \times \{0,1\}^c \to \Delta$ such that $\forall K \in \{0,1\}^c, d \in \Delta : \mathsf{Dec}(\mathsf{Enc}(d,K),K) = d$.

We do not require that encrypted data sets be decryptable. The main purpose of outsourcing data in this work is to remotely execute queries on it.

Definition 9. A *query* q is a PPT algorithm that, on input of a data set $d \in \Delta$ returns a *result set* $q(d) \in P$ and an updated data set $d_q \in \Delta$.

We point out that, to simplify notation, we do not model parameters for queries explicitly. Our model supports parameterised queries nevertheless, as for each pair of a query q with parameters p, there is an equivalent query $q(p)$ that has the parameters "hard coded". Without loss of generality we assume that queries are functions of the data, i.e. $\forall q \exists d_1, d_2 \in \Delta : q(d_1) \neq q(d_2)$, and the query result is the same if evaluated twice.

Our idea of the "correctness" of a protocol is relative to a given query q.

Definition 10. A two-party protocol $\pi_q \in \Pi$ between a Server $\mathfrak{S}$ and Client $\mathfrak{C}$ *executes a query* q for a outsourced data set $\mathsf{Enc}(d,K)$ if

- The Client, on input of a key K, outputs the result set $q(d) = \pi_q^{\mathfrak{C}}(K, \mathsf{Enc}(d,K))$.
- The Server, on input the outsourced data set $\mathsf{Enc}(d,K)$, outputs an updated outsourced data set $\mathsf{Enc}(d_q,K) = \pi_q^{\mathfrak{S}}(\mathsf{Enc}(d,K))$.

Note that although "update" queries are outside the scope of this work, Definition 10 also models protocols that update the outsourced data set.

Definition 11. A *queryable outsourcing scheme* for data sets is a tuple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ such that

- $(\mathsf{Gen}, \mathsf{Enc})$ is an outsourcing scheme for data sets, and
- $\mathsf{Q} \subseteq \Pi$ is a non-empty set of efficient two-party protocols that execute a query for outsourced data sets.

We stress that the client has no direct access to the data when interacting with the server in order to execute a query.

So that we can argue about privacy in the presence of queries to outsourced data, we require a notion of what a protocol party "sees" during the execution of a protocol.

Definition 12. A *view* of a protocol party is the totality of its inputs, received messages, sent messages and outputs. To denote the view of protocol party $\mathfrak{P} \in \{\mathfrak{C}, \mathfrak{S}\}$ in protocol π with inputs c and K, we write

$$\mathsf{view}^{\pi}_{\mathfrak{P}}(c, K).$$

In particular, the encrypted data is part of the server's view.

4.3.1 Static Security

The static notion of privacy for outsourced data captures the intuition that no adversary may deduce any information about the data from its ciphertext alone. We model it closely after the Indistinguishability under Chosen-Plaintext Attack (IND-CPA) notion.

Security Game 1 ($\mathsf{IND\text{-}CDA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$ and a random bit $b \leftarrow \{0, 1\}$.
2. The adversary $\mathcal{A}$ is given input 1^k and oracle access to $\mathsf{Enc}(\cdot, K)$.
3. $\mathcal{A}$ outputs two data sets d_0 and d_1 of equal length to the experiment.
4. $\mathcal{A}$ is given $\mathsf{Enc}(m_b, K)$.
5. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 13 (Static Security)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc})$ has *indistinguishable encryptions under chosen-data attacks* $\mathsf{IND\text{-}CDA}$ or *static security*, if for all Probabilistic Polynomial Time (PPT) adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{IND\text{-}CDA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

4.3.2 Privacy in the Presence of Queries

When outsourced data sets are queried three conceptually different privacy goals can be distinguished: keeping the data private, keeping the queries private, and keeping the results private. We model these privacy goals as security games. The adversary is supplied an oracle for views on the interaction between client and server and tries to discern the challenge bit b.

In all three security experiments, in addition to a challenge oracle, the adversary is supplied with an "open" view oracle. The oracle provides views for arbitrary queries executed on an encryption of arbitrary data sets *using the challenge key*. It implies that the scheme must have a probabilistic property in the sense that two identical queries on two different encryptions of the same plaintext will not access the same parts of the ciphertext. This can either be done by randomising the structure of the encrypted ciphertext (as we do in our construction in Section 4.6) or by randomising the protocol which executes the query.

Security Game 2 ($\mathsf{D\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The oracle takes a query q and a data set d as input and returns $\mathsf{view}^{\pi_q}_{\mathfrak{S}}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ outputs two data sets d_0 and d_1 of equal length to the experiment.
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge oracle: $\mathcal{A}$ is given access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$. That is, the oracle takes any query q such that $\pi_q \in \mathcal{Q}$ as input, internally runs the protocol π_q on $\mathsf{Enc}(d_b, K)$, and outputs $\mathsf{view}^{\pi_q}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$ to the adversary.
6. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Security Game 3 ($\mathsf{Q\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The oracle takes a query q and a data set d as input and returns $\mathsf{view}^{\pi_q}_{\mathfrak{S}}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ outputs two queries q_0 and q_1 to the experiment. q_0 and q_1 must yield protocols π_{q_0} and π_{q_1} with the same number of protocol messages.
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge oracle: $\mathcal{A}$ is given access to an oracle for $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$. That is, the oracle takes any data set $d \in \Delta$ as input, internally runs the protocol π_{q_b} on $\mathsf{Enc}(d, K)$, and outputs $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(d, K))$ to the adversary.
6. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 14 (Data Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has *Data Privacy*, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{D\text{-}IND}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

The privacy notion of Query Privacy captures the goal of hiding the queries themselves from the server. The notion is equivalent to Private Information Retrieval (see Section 4.5.1 for a discussion and proof).

Definition 15 (Query Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has *Query Privacy*, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{Q\text{-}IND}^{\mathcal{A},R_q}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

The third privacy goal, Result Privacy, captures the idea that the adversary must not learn the result of any query executed on any data. To formulate this idea formally, we allow the adversary to output two data-set-query pairs (d_0, q_0) and (d_1, q_1), as a result is always determined by a query and a data set on which it is evaluated. We then challenge the adversary on the view of query q_b executed on data set d_b.

Security Game 4 ($\mathsf{R\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The oracle takes a query q and a data set d as input and returns $\mathsf{view}^{\pi_q}_{\mathfrak{S}}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment. ($|d_0| = |d_1|$ and q_0 and q_1 must yield protocols π_{q_0} and π_{q_1} with the same number of protocol messages.)
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge: The experiment runs the protocol π_{q_b} on $\mathsf{Enc}(d_b, K)$ and outputs $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$ to the adversary.
6. $\mathcal{A}$ receives oracle access to $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$ and $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$.
7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 16 (Result Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has *Result Privacy*, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{R\text{-}IND}^{\mathcal{A},R_d,R_q}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

4.3.3 Fundamental Relations Among the Basic Security Notions

We establish fundamental relations among the three concepts of Data Privacy, Query Privacy, and Result Privacy.

Theorem 8 ($\mathsf{D\text{-}IND} \not\Rightarrow \mathsf{Q\text{-}IND}$)**.** *If a data outsourcing scheme that has Data Privacy exists, there is a data outsourcing scheme that has Data Privacy but no Query Privacy.*

Proof. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ be a data outsourcing scheme that has Data Privacy. We modify it in a way that we violate Query Privacy, but keep Data Privacy intact. To this end, we amend the protocols that execute queries to have the client transmit the executed query in the clear after the actual protocol is complete. We have to show that the modification violates Query Privacy, but does not violate Data Privacy.

With the modification, the adversary in experiment Q-IND can easily extract the executed query from any view and thus determine the challenge bit with certainty. Thus the modification violates Query Privacy. To see that this modification does not violate Data Privacy, first note that the modified scheme retains its Data Privacy up until the point of the modification. We argue that the transmission of the query in the clear does not break Data Privacy. Consider experiment D-IND. Because the experiment draws the key K after the scheme is fixed, the scheme is independent of the actual key used to encrypt the data set d. Further, because the query is supplied by the adversary in the experiment and the adversary has learned neither d_b nor K up to this point, the query is also independent of d_b and K. This concludes the argument. □

Theorem 9 (Q-IND $\not\Longrightarrow$ D-IND)**.** *If there is a retrievable data outsourcing scheme that has Static Security, there is a data outsourcing scheme that has Query Privacy and Static Security, but no Data Privacy.*

Proof. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ be a retrievable data outsourcing scheme that has Static Security. We construct a modified scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q}')$ that suits our purposes. By adopting Gen and Enc, we retain static security. We design Q' such that it has Query Privacy, but trivially loses Data Privacy. Q' is constructed iteratively, starting with an empty set. For each protocol $\pi_q \in \mathsf{Q}$ that realises a query q, we define a protocol π'_q to Q' as follows:

(Recall that the client's input is the encryption key K and a query q; the server's input is an encrypted data set $\mathsf{Enc}(d, K)$.)

1. Client: Transfer K to the Server.
2. Server: Decrypt $\mathsf{Enc}(d, K)$ and send $d = \mathsf{Dec}(\mathsf{Enc}(d, K), K)$ back to the Client.
3. Client: Execute query q locally on d and output $q(d)$.

Protocol π' transmits the data set d in the open, violating Data Privacy. Because the client executes q locally and never transmits any information that depends on q, π' does have Query Privacy. □

The following theorems show that Result Privacy is equivalent to both Data Privacy and Query Privacy (at the same time).

Theorem 10 (R-IND $\Longrightarrow$ D-IND)**.** *There is no data outsourcing scheme that has Result Privacy but no Data Privacy.*

Proof. Assume a data outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ for which there is an efficient adversary $\mathcal{A}$ against experiment D-IND. We give an efficient reduction for $\mathcal{A}$ that breaks the Result Privacy (experiment R-IND) of the scheme, contradicting the assumption.

The reduction is straightforward. It has to provide a challenge oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$. Such an oracle is provided by experiment R-IND and only has to be "passed through". □

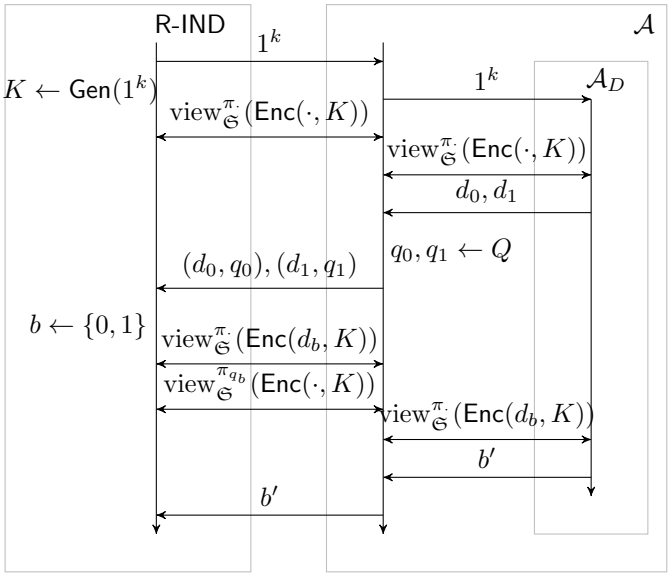

Figure 4.2: Sketch for the proof of Theorem 10.

Theorem 11 (R-IND $\implies$ Q-IND)**.** *There is no data outsourcing scheme that has Result Privacy but no Query Privacy.*

The proof of Theorem 11 is analogous to the proof of Theorem 10.

Proof. Assume a data outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ for which there is an efficient adversary $\mathcal{A}$ against experiment Q-IND. We give an efficient reduction for $\mathcal{A}$ that breaks the Result Privacy (experiment R-IND) of the scheme, contradicting the assumption.

Provide a challenge oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$. Such an oracle is provided by experiment—pass it through. □

Theorem 12 (D-IND $\wedge$ Q-IND $\implies$ R-IND)**.** *Data Privacy and Query Privacy together imply Result Privacy, i.e. there is no data outsourcing scheme that has Data Privacy and Query Privacy but no Result Privacy.*

We prove the statement using a game-hopping technique. Assume any adversary against R-IND. We replace both view oracles for d_b and q_b, respectively, with an oracle for fixed challenges d_0 and q_0. We argue the indistinguishability of these steps with Data Privacy and Query Privacy. Finally, in the now-transformed experiment, the adversary has no advantage since his input is independent of b. Concluding, given a scheme with Data Privacy and Query Privacy, no adversary against Result Privacy has a non-negligible advantage.

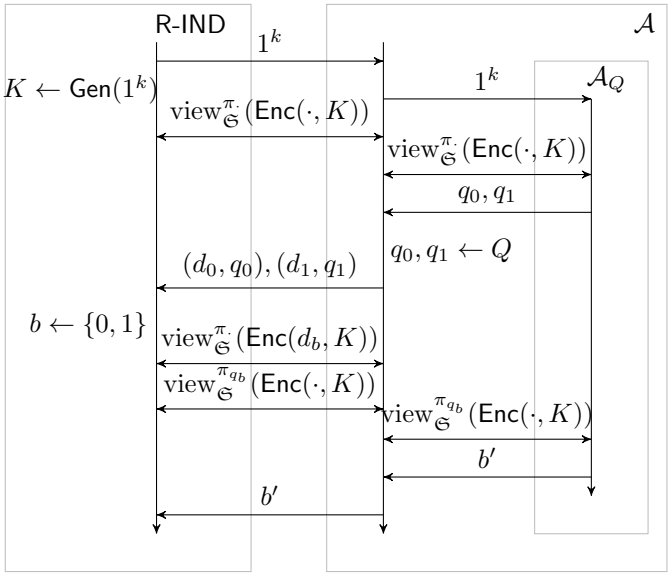

Figure 4.3: Sketch of the proof for Theorem 11.

Proof. We define two game transformations, R-IND′ and R-IND″, starting from the Result Privacy experiment R-IND. In the unmodified experiment R-IND, the adversary is supplied with two view oracles $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ and $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$. In R-IND′ we replace the $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ oracle by an oracle for $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$. Further, the challenge is generated not for d_b, but for d_0.

Security Game 5 ($\mathsf{R\text{-}IND}'^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The oracle takes a query q and a data set d as input and returns $\mathsf{view}_{\mathfrak{S}}^{\pi_q}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment. ($|d_0| = |d_1|$ and q_0 and q_1 must yield protocols π_{q_0} and π_{q_1} with the same number of protocol messages.)
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge: The experiment runs the protocol π_{q_b} on $\mathsf{Enc}(d_0, K)$ and outputs $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_0, K))$ to the adversary.
6. $\mathcal{A}$ receives oracle access to $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$ and $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$.
7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

In R-IND″ we further replace $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$ by $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(\cdot, K))$. The challenge is now independent of the challenge bit b.

Security Game 6 ($\mathsf{R\text{-}IND}''^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The oracle takes a query q and a data set d as input and returns $\mathsf{view}_{\mathfrak{S}}^{\pi_q}(\mathsf{Enc}(d, K))$.
3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment. ($|d_0| = |d_1|$ and q_0 and q_1 must yield protocols π_{q_0} and π_{q_1} with the same number of protocol messages.)
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge: The experiment runs the protocol π_{q_0} on $\mathsf{Enc}(d_0, K)$ and outputs $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(d_0, K))$ to the adversary.
6. $\mathcal{A}$ receives oracle access to $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$ and $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(\cdot, K))$.
7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

In R-IND″ the adversary receives no input that is dependent on the challenge bit b. He thus has no advantage over guessing b. We have to argue that R-IND″ is indistinguishable from R-IND for the adversary. To this end, we prove the indistinguishability of R-IND from R-IND′ in Lemma 1 and the indistinguishability of R-IND′ from R-IND″ in Lemma 2. □

Lemma 1. *An adversary who can distinguish between running in experiment* R-IND *and experiment* R-IND′ *yields a successful adversary against database privacy.*

We model the distinction as an experiment D-Oracle-IND in which the adversary must decide whether he is running in R-IND or in R-IND′. It is modeled closely after R-IND. In experiment R-IND′ the challenge database d_b is replaced with the fixed database d_0. Thus, in D-Oracle-IND the adversary is challenged on deciding whether he has access to an oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ or whether he is accessing the fixed oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$. (The query view oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$ is provided with no change.) We give a reduction $\mathcal{R}$ which transforms this adversary into an adversary on database privacy. To clearly separate the different challenge bits, we name the challenge bit in the distinction experiment c.

Security Game 7 ($\mathsf{D\text{-}Oracle\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$.
3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment (under the restriction that $|d_0| = |d_1|$ and that π_{q_0} and π_{q_1} exchange the same number of messages).

4. The experiment draws two random bits $b \leftarrow \{0,1\}$ and $c \leftarrow \{0,1\}$.
5. Challenge: If $c = 0$, output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_b, K))$ to the adversary. Else, output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_0, K))$.
6. If $c = 0$, $\mathcal{A}$ receives oracle access to $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$, else to $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$. In any case, $\mathcal{A}$ receives access to $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$.
7. $\mathcal{A}$ outputs a guess c' for c.

Proof. Assume an adversary $\mathcal{A}$ with a non-negligible advantage in experiment D-Oracle-IND. We construct a reduction that has a non-negligible advantage in experiment D-IND. The experiment D-IND provides us with two oracles $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ and $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$. We use these oracles to simulate the two oracles and the challenge in D-Oracle-IND:

- To generate the challenge, fix $q_b := q_1$ and output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(d_b, K))$ using the $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$ oracle provided by D-IND.
- $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_b, K))$. This oracle is provided by D-IND and is relayed.
- $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$. Fix $q_b := q_1$ and simulate the oracle using the $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ oracle provided by D-IND.

We can distinguish two cases. If $b = 1$ in D-IND, the challenge to $\mathcal{A}$ as well as the simulated oracles are consistent and the correct challenge bit in the reduction is $c = 0$. On the other hand, if $b = 0$, the views are inconsistent and $c = 1$. Thus, we invert $\mathcal{A}$'s guess as the reduction's own guess $b' = 1 - c'$ to inherit $\mathcal{A}$'s success probability. □

Lemma 2. *An adversary who can distinguish between* R-IND′ *and* R-IND″ *is also a successful adversary on Query Privacy.*

The proof of Lemma 2 is similar to that of Lemma 1.

We model the distinction as an experiment Q-Oracle-IND in which the adversary must decide whether he is running in R-IND′ or in R-IND″. In experiment R-IND″ the challenge query q_b is replaced with the fixed query q_0. Thus, in Q-Oracle-IND the adversary is challenged on deciding whether he has access to an oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$ or whether he is accessing the fixed oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(\cdot, K))$. (The oracle $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$ is provided with no change.) We give a reduction $\mathcal{R}'$ which transforms this adversary into an adversary on query privacy. To clearly separate the different challenge bits, we name the challenge bit in the distinction experiment c.

Security Game 8 ($\mathsf{Q\text{-}Oracle\text{-}IND}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$.
3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment (under the restriction that $|d_0| = |d_1|$ and that π_{q_0} and π_{q_1} exchange the same number of messages).

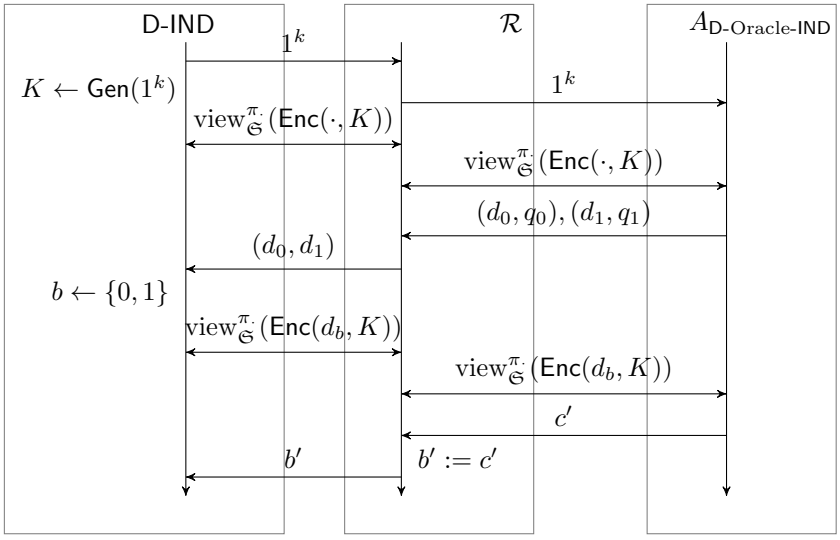

Figure 4.4: Sketch of the proof of Lemma 1. An efficient adversary who can decide whether he is running in experiment R-IND yields an efficient adversary against D-IND.

4. The experiment draws two random bits $b \leftarrow \{0,1\}$ and $c \leftarrow \{0,1\}$.
5. Challenge: If $c = 0$, output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_0, K))$ to the adversary. Else, output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(d_0, K))$.
6. If $c = 0$, $\mathcal{A}$ receives oracle access to $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$, else to $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_0}}(\mathsf{Enc}(\cdot, K))$. In any case, $\mathcal{A}$ receives access to $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$.
7. $\mathcal{A}$ outputs a guess c' for c.

Proof. Assume an adversary $\mathcal{A}$ with a non-negligible advantage in experiment Q-Oracle-IND. We construct a reduction that has a non-negligible advantage in experiment Q-IND. We need to simulate the challenge and two oracles:

- To generate the challenge, fix d_0 and output $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_0, K))$ using the $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$ oracle provided by Q-IND.
- $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(\cdot, K))$. This oracle is provided by Q-IND.
- $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(d_0, K))$. This oracle is easily simulated by using the $\mathsf{view}_{\mathfrak{S}}^{\pi_\cdot}(\mathsf{Enc}(\cdot, K))$ oracle provided by Q-IND, fixing d_0.

We can distinguish two cases. If $b = 0$ in Q-IND, the challenge to $\mathcal{A}$ as well as the simulated oracles are consistent and $\mathcal{A}$ has no advantage over guessing c. (The simulation is still perfect, however.) On the other hand, if $b = 1$, the views are inconsistent and $c = 0$. Thus, we invert $\mathcal{A}$'s guess as the reduction's own guess $b' = 1 - c'$ to inherit

$\mathcal{A}$'s success probability in the second case. Overall, the reduction inherits half of $\mathcal{A}$'s advantage. □

Corollary 1 (R-IND $\iff$ D-IND $\wedge$ Q-IND)**.** *Result Privacy is equivalent to both Data Privacy and Query Privacy (at the same time).*

4.4 Generalised Security Notions for Data Outsourcing Schemes

In this section, we generalise the security notions introduced in Section 4.3 to make them applicable to practical schemes. Protocols that—for the sake of efficiency—base decisions on the content of the queried data are bound to leak information about it [48, 49, 53] or are only secure for a limited number of queries (see for example our construction SEDAWG in Section 4.6). (We discuss information leakage in Section 4.1.)

Therefore, we first introduce bounds for the number of oracle calls. A special case is a bound of 1 that renders the notions non-adaptive.

Second, we define "leakage relations" R_d and R_q. Challenges the adversary can choose are subject to equivalence under these relations. This way, one can explicitly rule out specific distinction advantages. To model the leakage of the length of a data set for example, one would define $R_d \subset \Delta^2$ as the set of all data set pairs with equal length.

Third, we explicitly model the issuing of queries independently of handing out the results. This allows us to capture security notions where the adversary can alter the state of the database, but can not see the immediate result (e.g. he can only observe the result of the last issued query).

Goh [89] introduces restricting parameters into his security notion as well. They allow for a bound on the running time, the advantage, and the number of oracle calls. Our work in this section can be seen as a generalisation of this concept.

In Section 4.5 we showcase case studies that are direct applications of our generalised notions. We only give the security definitions here and defer discussion to the following section.

Security Game 9 ($\mathsf{IND\text{-}CDA}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$ and a random bit $b \leftarrow \{0,1\}$.
2. The adversary $\mathcal{A}$ is given input 1^k and oracle access to $\mathsf{Enc}(\cdot, K)$.
3. $\mathcal{A}$ outputs two data sets d_0 and d_1 to the experiment. The choice of d_0 and d_1 is restricted to data set pairs that are equivalent with regard to equivalence relation $R_d \subseteq \Delta^2$, i.e. $(d_0, d_1) \in R_d$.
4. $\mathcal{A}$ is given $\mathsf{Enc}(m_b, K)$.
5. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 17 (Static Security)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc})$ has *indistinguishable encryptions under chosen-data-attacks (*$\mathsf{IND\text{-}CDA}$*)* or *static security* with respect to R_d, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{IND\text{-}CDA}^{\mathcal{A},R_d}_{(\mathsf{Gen},\mathsf{Enc})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

Security Game 10 ($\mathsf{D\text{-}IND}^{\mathcal{A},R_d,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. $\mathcal{A}$ is only allowed to query the oracle for a total number of n_1 times.
3. $\mathcal{A}$ outputs two data sets d_0 and d_1 to the experiment. The choice of d_0 and d_1 is restricted to pairs of data sets that are equivalent with regard to equivalence relation $R_d \subseteq \Delta^2$, i.e. $(d_0, d_1) \in R_d$.
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge oracle: $\mathcal{A}$ is given access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$, and continues to have access to it. $\mathcal{A}$ may call the challenge oracle for a total number of n_2 times.
6. Run oracle: $\mathcal{A}$ is given access to an oracle $\mathrm{run}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$. The run oracle executes queries just as the view oracle does, but has no output. $\mathcal{A}$ is allowed to call the run oracle for a total number of n_3 times.
7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Security Game 11 ($\mathsf{Q\text{-}IND}^{\mathcal{A},R_q,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. $\mathcal{A}$ is only allowed to query the oracle for a total number of n_1 times.
3. $\mathcal{A}$ outputs two queries q_0 and q_1 to the experiment. The choice of q_0 and q_1 is restricted to query pairs that are equivalent with regard to equivalence relation $R_q \subseteq \Pi^2$, i.e. $(q_0, q_1) \in R_q$.
4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.
5. Challenge oracle: $\mathcal{A}$ is given access to an oracle for $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$. $\mathcal{A}$ may call the challenge oracle for a total number of n_2 times.
6. Run oracle: $\mathcal{A}$ is given access to an oracle $\mathrm{run}^{\pi_b}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. The run oracle executes queries just as the view oracle does, but has no output. $\mathcal{A}$ is allowed to call the run oracle for a total number of n_3 times.

7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 18 (n_1, n_2, n_3-Data Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has n_1, n_2, n_3-*Data Privacy* with respect to R_d, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{D\text{-}IND}^{\mathcal{A},R_d,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

Definition 19 (n_1, n_2, n_3-Query Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has n_1, n_2, n_3-*Query Privacy* with respect to R_q, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{Q\text{-}IND}^{\mathcal{A},R_q,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

Result Privacy is generalised in the same way.

Security Game 12 ($\mathsf{R\text{-}IND}^{\mathcal{A},R_d,R_q,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.

2. $\mathcal{A}$ receives access to an oracle for $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it. $\mathcal{A}$ is only allowed to query $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$ for a total number of n_1 times.

3. $\mathcal{A}$ outputs two data-set-query pairs (d_0, q_0) and (d_1, q_1) to the experiment. The choice of d_0, d_1, q_0, and q_1 is restricted to $(d_0, d_1) \in R_d$ and $(q_0, q_1) \in R_q$.

4. The experiment draws a random bit $b \leftarrow \{0, 1\}$.

5. Challenge: The experiment runs the protocol π_{q_b} on $\mathsf{Enc}(d_b, K)$ and outputs $\mathsf{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(d_b, K))$ to the adversary.

6. $\mathcal{A}$ receives oracle access to $\mathrm{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(m_b, K))$ and $\mathrm{view}^{\pi_{q_b}}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$. He is only allowed to call the oracles a total number of n_2, respectively n_3, times.

7. $\mathcal{A}$ outputs a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 20 (n_1, n_2, n_3-Result Privacy)**.** An outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ has n_1, n_2, n_3-*Result Privacy* with respect to R_d and R_q, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{R\text{-}IND}^{\mathcal{A},R_d,R_q,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

4.5 Case Studies

In this section we review security notions from the literature and examine how they fit into our framework. To that end, we translate these notions to our formalisms.

Some of the notions we discuss here intertwine the concepts of Data Privacy and Query Privacy and cannot be instantiated directly. We show implications between these and our security notions. We stress that we see a strong need to separate the concepts of Data Privacy and Query Privacy in order to have a clear understanding of the security properties of the underlying scheme.

4.5.1 Private Information Retrieval

We give a definition of the original (single-server) Computational Private Information Retrieval (cPIR) [140] notion using our conventions.

Definition 21 (Private Information Retrieval)**.** A queryable outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \{\pi\})$ exhibits *Computational Single-Server Private Information Retrieval (PIR)* when two conditions hold for any $n \in \mathbb{N}$, security parameter $k \in \mathbb{N}$, and any data set d over $\Sigma = \{0,1\}^n$:

1. Correctness: $\forall i \in \{0, \ldots, n-1\} : \pi_i^{\mathfrak{C}}(d) = d[i]$.
2. Privacy: $\forall c \in \mathbb{N}, i, j \in \{0, \ldots, n-1\}, \forall \mathcal{A} \exists N \in \mathbb{N}$ such that $\forall k > N$

$$\left|\Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d,K))) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d,K))) = 1\right]\right| < \frac{1}{\max(k,n)^c},$$

where $K \leftarrow \mathsf{Gen}(1^k)$.

Theorem 13. *Private Information Retrieval is equivalent to Query Privacy.*

In our proof we implicitly assume a queryable database outsourcing scheme that has, for each bit in the database, a query that retrieves it, i.e. we exclude schemes that store unretrievable information in the database. This is not a restriction, as one can easily construct a "non-redundant" scheme from a one that stores unretrievable information.

Proof. For this proof, let the domain of all data sets be $\Delta = \{0,1\}^*$. Fix a security parameter $k \in \mathbb{N}$. Without loss of generality assume any queryable outsourcing scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Q})$ with a protocol π_i that outputs the $i+1$th bit of the database for all $i \in \{0, \ldots, n-1\}$, where n is the length of the data set $d \in \Delta$. We prove the theorem in two steps.

PIR $\implies$ Q-IND. Assume any efficient adversary $\mathcal{A}$ who is successful in Q-IND with a non-negligible advantage. We show that there are $i, j \in \{0, \ldots, n-1\}$, and an efficient algorithm $\mathcal{A}'$ such that they violate the Privacy condition of Definition 21.

Construct $\mathcal{A}'$ as follows: Simulate experiment Q-IND to obtain π_i, π_j from $\mathcal{A}$. i and j are the required indices. Now relay the input $\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d,K))$ (for $b \in \{i,j\}$) to $\mathcal{A}$. Output $\mathcal{A}$'s guess b'.

Q-IND $\implies$ **PIR.** Assume any $i, j \in \{0, \ldots, n-1\}$ and any efficient algorithm $\mathcal{A}'$ such that $\mathcal{A}'$ violates the Privacy condition of Definition 21 at indices i and j. We construct an efficient adversary $\mathcal{A}$ that has a non-negligible advantage in Q-IND: Output π_i and π_j as the challenge queries. Output $b' = \mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d, K)))$ (for $b \in \{i, j\}$). (For any adversary with a success probability $< \frac{1}{2}$ there is an adversary with a success probability $> \frac{1}{2}$, easily obtained by flipping its guess.) □

4.5.2 Indistinguishability under Independent Column Permutations

Huber et al. [113] present a provably-secure database outsourcing scheme which is as efficient as the underlying database. In their notion the encryptions of two databases must be indistinguishable if they can be transformed into each other by permuting attribute values within columns. Since our generalised notions allow for defining a restriction on the plaintexts, this database-specific security notion also fits into our framework.

Definition 22 (Independent Column Permutation [113])**.** Let Φ be the set of database functions $\mathfrak{p} : \Delta \to \Delta$ such that each $\mathfrak{p} \in \Phi$ permutes the entries within each column of a database. We call $\mathfrak{p}$ an *independent column permutation*.

Security Game 13 ($\mathsf{IND\text{-}ICP}^{\mathcal{A}}_{\mathsf{Gen},\mathsf{Enc},\Phi}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$.
2. $\mathcal{A}$ outputs one plaintext m and an independent column permutation $\mathfrak{p} \in \Phi$ to the experiment.
3. The experiment chooses $m_0 := m$ and $m_1 := \mathfrak{p}(m)$ draws $b \leftarrow \{0, 1\}$ uniformly at random.
4. $\mathcal{A}$ is given $\mathsf{Enc}(m_b, K)$.
5. $\mathcal{A}$ submits a guess b' for b.

Theorem 14. IND-ICP *is equivalent to static security.*

Proof. IND-ICP is a direct instantiation of $\mathsf{IND\text{-}CDA}^{\mathcal{A},R_{\mathrm{ICP}}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$, where $R_{\mathrm{ICP}} \subset \Delta^2$ is the set of all pairs of databases that are independent column permutations of each other: We set $\Delta = DB$ and $R_{\mathrm{ICP}} = \Delta_{/\Phi(\Delta)}$. Then, each adversary that has a non-negligible advantage in $\mathsf{IND\text{-}ICP}^{\mathcal{A}}_{\mathsf{Gen},\mathsf{Enc},\Phi}(k)$ can efficiently be reduced to an adversary that has non negligible advantage in $\mathsf{IND\text{-}CDA}^{\mathcal{A},R_{\mathrm{ICP}}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$ and vice versa. The reduction from IND-ICP to $\mathsf{IND\text{-}CDA}^{\mathcal{A},R_{\mathrm{ICP}}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$ sets $m_0 := m$ and $m_1 := \mathfrak{p}(m)$, while the reduction from $\mathsf{IND\text{-}CDA}^{\mathcal{A},R_{\mathrm{ICP}}}_{(\mathsf{Gen},\mathsf{Enc})}(k)$ to IND-ICP determines $\mathfrak{p}$ with $\mathfrak{p}(m_0) = m_1$. □

4.5.3 Semantic Security Against Adaptive Chosen Keyword Attacks

Goh [89] presents a security notion for index-based searchable encryption schemes. We give a translation into our formalisms. Trapdoors are translated to a view oracle.

Security Game 14 ($\mathsf{IND\text{-}CKA}^{\mathcal{A}}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$ and a random bit $b \leftarrow \{0,1\}$.
2. The adversary $\mathcal{A}$ is given input 1^k and access to an oracle $\mathrm{view}^{\pi,\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot,K))$.
3. $\mathcal{A}$ outputs two plaintexts m_0 and m_1 of the same length to the experiment. The adversary must not have queried the oracle for words that are only in one of the two plaintexts.
4. $\mathcal{A}$ is given $\mathsf{Enc}(m_b,K)$ and access to an oracle $\mathrm{view}^{\pi,\cdot}_{\mathfrak{S}}(\mathsf{Enc}(m_b,K))$.
5. $\mathcal{A}$ submits a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

IND-CKA is a weaker form of Data Privacy. Were the adversary not restricted in choosing queries (Line 3. in Security Game 14), the notions were equivalent. As in the case of Curtmola et al.'s notion (Section 4.5.4), we prove the relation of Goh's notion to our model without considering this restriction. We point out that one could easily further generalise our security notions to include this restriction by additionally forcing queries to adhere to a relation to the data set. However, we decided against this, as the applications of such restrictions seem limited.

Theorem 15. IND-CKA *implies static security.*

Proof. Suppose an adversary who has a non-negligible advantage in IND-CDA (Static Security, Section 4.3.1.) This adversary can distinguish two encrypted databases even without access to queries. We give a reduction from the IND-CKA experiment:

- Use the $\mathsf{view}^{\pi,\cdot}_{\mathfrak{S}}(\mathsf{Enc}(,\mathrm{K}))$ oracle to provide the $\mathsf{Enc}(\cdot,K)$ oracle. (The encrypted database is part of the server's input.)
- Forward the challenge data sets from the adversary to the experiment.
- Hand over the challenge $\mathsf{Enc}(m_b,K)$ from the experiment to the adversary, never query the view oracle.
- Submit the adversary's guess as the reduction's own guess.

□

Theorem 16. *Database privacy implies* IND-CKA.

Proof. Assume an adversary $\mathcal{A}$ who who has a non-negligible advantage in experiment IND-CKA. We give an efficient reduction that breaks database privacy. The reduction forwards the two challenge data sets from $\mathcal{A}$ to experiment D-IND. All queries $\mathcal{A}$ executes using via the view oracle (before and after submitting the two challenge databases) can easily be simulated by using the oracles from the D-IND experiment (this is because the set of valid queries in D-IND is a superset of the valid queries in IND-CKA). The reduction forwards the adversary's guess b' to the experiment. □

4.5.4 Adaptive Security for Symmetric Searchable Encryption

Curtmola et al.'s notion *adaptive indistinguishability security for SSE* [64] is also a security notion for symmetric searchable encryption based on indices.

Security Game 15 ($\mathsf{Ind}^*_{\mathrm{SSE},\mathcal{A},(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$ [64])**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$ and a random bit $b \leftarrow \{0,1\}$.
2. The adversary $\mathcal{A}$ is given input 1^k and outputs two plaintexts m_0 and m_1 of the same length to the experiment.
3. $\mathcal{A}$ is given $\mathsf{Enc}(m_b, K)$.
4. $\mathcal{A}$ can output polynomially many pairs of queries (q_0, q_1) and is given $\mathrm{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_b, K))$.
5. $\mathcal{A}$ submits a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Note that in Curtmola et al.'s notion, Query Privacy and Data Privacy are intertwined. Thus, it can not be directly instantiated from our framework. We instead show how his notion relates to our notions. $\mathsf{Ind}^*_{\mathrm{SSE}}$ also requires the adversary to only choose plaintexts and corresponding search queries which have the same views for both databases. This is a very strict restriction which we do not take into consideration here. We instead focus on the more general notion.

Theorem 17 ($\mathsf{Q\text{-}IND} \wedge \mathsf{D\text{-}IND} \implies \mathsf{Ind}^*_{\mathrm{SSE}}$)**.** *If a queryable outsourcing scheme has Query Privacy, Data Privacy implies* Ind^*_{SSE}*.*

We prove the statement using a game-hopping technique. The argument is very similar to the proof of Theorem 12. We only sketch the technique.

Proof. Modify $\mathsf{Ind}^*_{\mathrm{SSE}}$ so that the challenge oracle always returns $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(d, K))$, independently of b. An adversary who can distinguish between the two games also breaks query privacy.

Database privacy implies the modified $\mathsf{Ind}^*_{\mathrm{SSE}}$ experiment. The reduction is straightforward. If the adversary against $\mathsf{Ind}^*_{\mathrm{SSE}}$ requests a view $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_b, K))$ for input (q_0, q_1), the reduction forwards q_1 to the oracle of Database Privacy and returns $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_1}}(\mathsf{Enc}(d_b, K))$. □

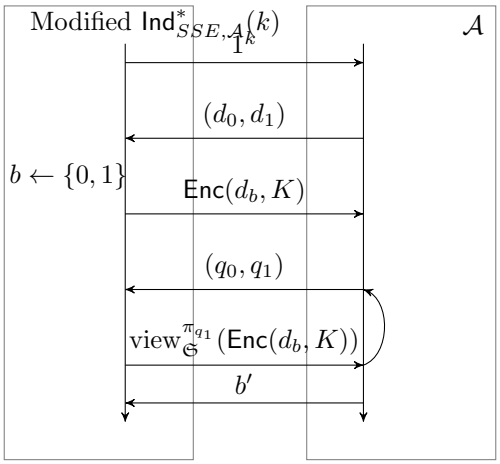

Figure 4.5: Sketch for the proof of Theorem 17.

Theorem 18 ($\mathsf{Ind}^*_{\mathrm{SSE}} \implies$ D-IND)**.** Ind^*_{SSE} *implies database privacy.*

Proof. We give a reduction from database privacy to $\mathsf{Ind}^*_{\mathrm{SSE}}$. It is straightforward. When the adversary against database privacy requests $\mathsf{view}_{\mathfrak{S}}^{\pi_q}(\mathsf{Enc}(d_b, K))$, the reduction fixes $q_0 := q_1 := q$ and returns $\mathsf{view}_{\mathfrak{S}}^{\pi_{q_b}}(\mathsf{Enc}(d_b, K))$. □

4.6 Searchable Encryption using Directed Acyclic Word Graphs

As another use case for the privacy framework we developed in this chapter, in this section we feature the security analysis of a searchable encryption scheme. Particularly, we propose a new class of searchable encryption schemes and give a concrete construction. Let us first motivate an application for the use case.

Consider an outsourced genome database. A biological research institution wishes to use a cloud storage service to store its genome database. Researchers desire to search the uploaded genomes for certain DNA sequences without having to download the whole genome first. Constructions that rely on search indices are not suited for such a purpose—one would need to store each genome sub-sequence in the index. (For a genome of length n, the index were to store $O(n+n-1+\ldots+1) = O(n^2)$ keywords.) A pattern-matching procedure is more efficient in such a scenario. To guarantee the privacy of the outsourced genomes, they should be kept secret from the cloud provider, e.g. through encryption.

We address the problem of secure exact pattern matching. A user encrypts a long string he later wishes to query for the occurrence of certain patterns. The patterns are assumed to be significantly shorter than the string. This encrypted string is then uploaded to a server. To perform a search, the user interacts with the server in a certain way. The server should never learn neither the string itself, nor the patterns searched for.

We introduce a primitive for a symmetric searchable encryption for exact pattern matching. To our knowledge, this is the first such primitive. Further, we offer a

construction that realises this primitive. The construction's query phase is particularly efficient, as our implementation shows.

In Section 4.6.1, we define a new primitive *Exact Pattern Matching for Symmetric Searchable Encryption (XPM-SSE)*. In Sections 4.6.2 and 4.6.3, we present our construction SEDAWG and discuss its properties. We describe our implementation in Section 4.6.4. Lastly, we analyse the security of XPM-SSE schemes in general and SEDAWG in particular in Section 4.6.5.

This section is based on a Diploma thesis [6], a term paper ("Studienarbeit") [8], and an unpublished extended version of a conference paper [5].

4.6.1 Exact Pattern Matching for Symmetric Searchable Encryption

We define a searchable encryption scheme which allows to perform a fulltext search within encrypted data. To this end, we define exact pattern matching first.

For the remainder of the subsection, if not mentioned otherwise, S will be an arbitrary string of length n over the alphabet Σ, that is $S \in \Sigma^*$.

Definition 23 (Exact Pattern Matching (XPM))**.** A PPT algorithm $\mathcal{A}$ is a technique for Exact Pattern Matching (XPM) over the alphabet Σ if it returns upon input $S \in \Sigma^*$ and $m \in \Sigma^*$ the exact number of occurrences of m in S.

Based on the exact pattern matching algorithm, we define Exact Pattern Matching for Symmetric Searchable Encryption.

Definition 24 (Exact Pattern Matching for Symmetric Searchable Encryption (XP-M-SSE))**.** Let $S \in \Sigma^n$ be an arbitrary but finite string over the alphabet Σ. A tuple $((\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}), \pi = (\mathfrak{S}, \mathfrak{C}))$ is a Symmetric Searchable Encryption scheme for Exact Pattern Matching (XPM-SSE scheme), if

- $\mathsf{Gen} : 1^k \to \{0,1\}^k$ is a PPT algorithm which, given a security parameter k, generates a key $K \leftarrow \mathsf{Gen}(1^k)$ of length k.
- $\mathsf{Enc} : \Sigma^* \times \{0,1\}^k \to \{0,1\}^*$ is a PPT algorithm which generates a ciphertext $C \leftarrow \mathsf{Enc}(S, K)$.
- $\mathsf{Dec} : \{0,1\}^* \times \{0,1\}^k \to \Sigma^*$ is a polynomially-bounded algorithm which outputs the plaintext $S = \mathsf{Dec}(\mathsf{Enc}(S, K), K)$ given the cipher text $\mathsf{Enc}(S, K)$ and the key K used for encryption.
- $\pi = (\mathfrak{S}, \mathfrak{C})$ is a protocol for a pair of machines (server and client) which perform an algorithm for Exact Pattern Matching. The server $\mathfrak{S}$ is provided with an encryption of the string $\mathsf{Enc}(S, K)$, the client $\mathfrak{C}$ is supplied with the search pattern w and the encryption key K, and outputs the exact number of occurrences of w in S. With $\mathsf{view}_{\mathfrak{S}}^{\pi_w}(\mathsf{Enc}(S, K))$ we denote the totality of the server $\mathfrak{S}$'s inputs, received messages, sent messages and outputs.

- The client's output after a successful protocol run is *correct* and *complete* with overwhelming probability, i.e. the output does not lack any correct results and only contains correct results.
- It has *Document Privacy* (Definition 30).

In the context of the chapter, the domain of a XPM-SSE scheme is the set of all strings over a given alphabet $\Delta = \Sigma^*$, while the ciphertext space is the set of all binary strings $\Gamma = \{0,1\}^*$.

The above-mentioned notion of Document Privacy captures the goal of hiding the string itself (the "document") from the adversary. Note that a static security definition (Definition 13, Section 4.3.1) does not suffice in such an outsourcing scenario we consider in this section. Suppose a construction that is statically secure, but which performs the following protocol when querying the string: To execute a query, the client uploads the decryption key to the server and lets the server decrypt the ciphertext and return the result. Such a protocol should not be considered secure. Hence, any meaningful notion must also account for the protocol messages that are exchanged in the evaluation of a query. We discuss the Data Privacy of XPM-SSE schemes in Section 4.6.5.

4.6.2 SEDAWG: A XPM-SSE Construction

Our searchable encryption scheme is based on the idea to store the encrypted data on the server and to perform the search on the client. The encryption algorithm uses Directed Acyclic Word Graphs (DAWGs) to prepare the encrypted data for searching. The actual search is done by graph traversal: Starting from the root node, one follows the edges of the DAWG whose labels match the search pattern. When an edge leads into a data block that is not present on the client, it is downloaded. See Figure 4.6 for an illustration.

Directed Acyclic Word Graphs A DAWG is a data structure derived from a string. It is similar to the Suffix Tree data structure and reduces the time complexity for several string operations, such as pattern matching, by making use of pre-computation. DAWGs were first introduced by Blumer et al. [29]. Our definition follows theirs.

Definition 25 (Endpoint Set)**.** Let x be a substring of a string S. Then

$$E_S(x) := \{j \mid \exists\, i : x = S[i..j]\}$$

is called the *Endpoint Set* for x with respect to S, i.e. the endpoints of all occurrences of x in S.

The substrings that specify the same Endpoint Set are important for the data structure. They induce an equivalence class.

Definition 26 (Equivalence Relation $\equiv_S^E$, Equivalence Class $[x]_S^E$)**.** Two substrings x and y of a string S are equivalent with respect to $\equiv_S^E$, if they specify the same Endpoint Set:

$$x \equiv_S^E y \Leftrightarrow E_S(x) = E_S(y)$$

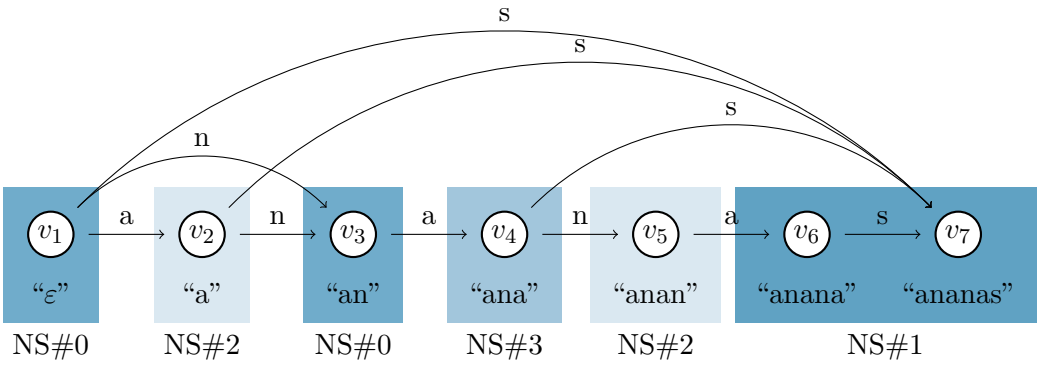

Figure 4.6: The DAWG for the input string S=`ananas`. It consists of seven nodes (v_1 to v_7) which are mapped to four node sets (NS#0 to NS#3). Each node is identified by its representative below. The transition edges are drawn as directed arrows which are labeled with their respective edge labels. Every outgoing path from v_1 is a subword of S.

The equivalence class of x with respect to $\equiv_S^E$ is denoted as $[x]_S^E$. The representative of an equivalence class is defined as the longest substring of that class and is denoted as $\overleftarrow{x}$.

Definition 27 ($DAWG(S)$)**.** The *Directed Acyclic Word Graph* of a prefix-free string S, $DAWG(S)$, is the directed graph (V, E) with

$$V = \{[x]_S^E \mid x \in \mathit{substrings}_S\},$$
$$E = \{([x]_S^E, [xa]_S^E) \mid a \in \Sigma, x, xa \in \mathit{substrings}_S, \overleftarrow{x} \neq \overleftarrow{xa}\},$$

where $\mathit{substrings}_S$ is the set of all substrings of S.

To give an intuition, the DAWG of a string has its suffixes as its nodes and joins them by symbols $\in \Sigma$. It can be interpreted as the Deterministic Finite Automaton that accepts all substrings of a string.

The equivalence class of the empty string $[\varepsilon]_S^E$ serves a special purpose. The corresponding node will be called the *root node* in the rest of the section. It does not have any incoming edges.

With every edge we associate a label.

Definition 28 (Edge Label $\mathit{edgeLabel}(e)$)**.** Let $([x]_S^E, [xa]_S^E) = e$ be an edge of the $DAWG(S)$ data structure. Then $\mathit{edgeLabel}(e) = a$ is called the *edge label* of e.

The edge labels are defined in such a way that every path in the graph $DAWG(S)$ corresponds to a subword of S. The path that corresponds to S itself plays an important role in the decryption algorithm of our scheme. We will refer to the edges of this path as *natural edges*.

Definition 29 (Natural Edge)**.** Let $([x]_S^E, [xa]_S^E) = e$ be an edge of the $DAWG(S)$ data structure. If there exists a string w in $[xa]_S^E$ such that w is a prefix of S, e is called a *natural edge* for this graph.

Data Representation

The representation of the DAWG plays a crucial part in our scheme as it strongly influences the access pattern on data elements when executing a query and thus the security and performance properties of our encryption scheme. In our XPM-SSE construction, we store the graph in adjacency lists. For a node $v \in V$ of $(V, E) = DAWG(S)$ the adjacency list contains one entry for each node w linked to v by an outgoing edge, i.e. it contains one entry for every edge $(v, w) \in E$. The adjacency list entry associated with edge (v, w) is referred to as $v[w]$. Every adjacency list entry contains the following fields:

- edgeLabel
- isNaturalEdge
- targetNodePosition
- targetNodeSetId

Further, we store with every node v a property $v.\mathsf{numOccurrs}$. (This property is only required if one desires to retrieve the number of occurrences of a given pattern.)

The size of adjacency lists in memory depends on the implementation. In our implementation (Section 4.6.4) it is

$$size(v) = constsize + |edges(v)| \cdot entrysize$$

where *constsize* is the size of the node's representation and *entrysize* is the size of one adjacency list entry.

Detailed Description of the Algorithms

In this section we describe the Searchable Encryption Scheme $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen}, \pi)$ by detailing each algorithm. In our algorithms we make black-box use of an IND-CPA secure secret-key encryption scheme which we will refer to as $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ (Section 2.4.1).

Encryption Algorithm Enc The four key steps of the encryption algorithm $\mathsf{Enc}_K(S)$ are:

1. Construct $DAWG(S)$ from the input string S. The DAWG is constructed from the input string S:

   $$(V, E) = DAWG(S)$$

   In our implementation we use the Blumer et al.'s construction algorithm [29]. During the graph generation, the edge labels and natural edge flags are set for each adjacency list entry $v[w]$:

   $$v[w].\mathsf{edgeLabel} = edgeLabel(v, w)$$

and

$$v[w].\mathsf{isNaturalEdge} = \begin{cases} \texttt{true} & \text{if } (v,w) \text{ is a natural edge} \\ \texttt{false} & \text{otherwise} \end{cases}$$

We set

$$v.\mathsf{numOccurrs} := |E_S(x)|$$

if node $v \in [x]_S^E$. This is the number of different occurrences of x in S because every element in $E_S(x)$ is one endpoint of such an occurrence.

2. Divide the nodes of the graph into disjoint node sets. The set of nodes V is split into pairwise-disjoint sets which we call *node sets*. The set of all node sets is denoted as N. Every node $v \in V$ is assigned to exactly one node set. Each node set is uniquely identified by an ID from $0, \ldots, |N|$ called the *node set ID* which we write as $id(ns)$ for any node set $ns \in N$.

   For every node set ns we specify a property $size(ns)$ which is calculated from the size of the nodes in the node set:

   $$size(ns) = \sum_{v \in ns} size(v)$$

   We require that the nodes are assigned to the node sets in such a way that there is no node set with size greater than a certain threshold $s^\star$. To ensure that every node can be assigned, $s^\star$ has a lower bound which is the maximum size of an adjacency list:

   $$s^\star > constsize + |\Sigma| \cdot entrysize$$

   The number of node sets $|N|$ is calculated as $2((2|S|-1) \cdot constsize + (3|S|-4) \cdot entrysize)/s^\star$. This number ensures that every node can be assigned to a node set while fulfilling the size constraint mentioned above, because $2|S|-1$ is an upper bound for the nodes, and $3|S|-4$ is an upper bound for the edges of $DAWG(S)$ as Blumer et al. show [29].

   For every node, we specify a property $position(v)$. It is used for referencing the node. It depends on the nodes previously added to the node set and is calculated during the assignment of a node to a node set.

   When assigning the nodes to node sets, the root node is assigned to the node set with ID 0 in a first step. Then for each node v of the remaining nodes, the following steps are performed:

   a) The target node set ns is drawn uniformly at random from a set of possible target node sets $\{ns \in N \mid s^\star \geq size(ns \cup v)\}$

   b) Set $position(v) := size(ns)$

   c) Set $nodeset(v) := id(ns)$ and $ns = ns \cup v$

3. Augment the adjacency list entries with references to the node set containing the target node. In this step, the values of the adjacency list fields `targetNodeSetId` and `targetNodePosition` are updated. For every edge $(v,w) \in E$ we set

- $v[w].\mathsf{targetNodePosition} := \mathit{position}(w)$, and
- $v[w].\mathsf{targetNodeSetId} := \mathit{nodeset}(w)$.

4. Create files from the node sets, pad and encrypt each file independently under the same key K and output the encrypted data. In this step, nodes that are assigned to the same node set are encrypted together into one file. For encryption, the probabilistic encryption algorithm E is used.The following steps are performed for every node set $ns \in N$:
   a) Write the data representation of each node $v \in ns$ to the file file_{ns} in order of their position $\mathit{position}(v)$.
   b) Pad file_{ns} to the size of $s^\star$.
   c) Encrypt $\mathsf{E}(\mathit{file}_{ns}, K)$.

   Lastly, all encrypted files are returned, paired with the node set ID of the encrypted node set. That is, the set

$$\{(\mathsf{E}(\mathit{file}_{ns}), \mathit{id}(ns), K)\}_{ns \in N}$$

   is output.

The pseudocode for this algorithm is shown in Algorithm 1.

Input: String $S \in \Sigma^*$, Key K
Output: A set B of files encrypted under K
Data: Set of vertices V, set of edges E, set of node sets N, $s^\star$

```
1  (V, E) := DAWG(S)                                          // Step 1
2  position(∘) := 0
3  nodeset(∘) := 0
4  getNodeSetById(0) := {∘}                  // Add root node to node set 0
5  foreach v ∈ V do                                           // Step 2
6  |  ns ← {ns ∈ N | s⋆ ≥ size(ns ∪ v)}
7  |  position(v) := size(ns)
8  |  nodeset(v) := id(ns)
9  |  ns := ns ∪ v
10 end
11 foreach ns ∈ N do                                          // Step 3
12 |  foreach v ∈ ns do
13 |  |  foreach (v, w) ∈ edges(v) do
14 |  |  |  v[w].targetNodePosition := position(w)
15 |  |  |  v[w].targetNodeSetId := nodeset(w)
16 |  |  end
17 |  end
18 end
19 foreach ns ∈ N do                                          // Step 4
20 |  b := ε
21 |  foreach v ∈ ns do
22 |  |  foreach (v, w) ∈ edges(v) do
23 |  |  |  b ⇐ v[w].isNaturalEdge ·
24 |  |  |  v[w].edgeLabel ·
25 |  |  |  v[w].targetNodePosition ·
26 |  |  |  v[w].targetNodeSetId
27 |  |  end
28 |  |  b ⇐ #
29 |  end
30 |  b := pad (b, s⋆)
31 |  b := E_K(b)
32 |  B := B ∪ {b}
33 end
34 return B
```

Algorithm 1: Algorithm $\mathsf{Enc}(S, K)$.

Decryption Algorithm Dec

The decryption algorithm reconstructs the input string S from the ciphertext $\mathsf{Enc}(S, K)$. We assume that all encrypted files from $\mathsf{Enc}(S, K)$ are available to the client. On a high level, the algorithm Dec consists of two steps:

1. Reconstruct the set of nodes by decrypting all files of $\mathsf{Enc}(S, K)$. For every file $\mathit{file} \in \mathsf{Enc}(S, K)$, compute $d \leftarrow \mathsf{D}(\mathit{file}, K)$ to retrieve the adjacency lists of the underlying graph.
2. Beginning with the adjacency list of the root node (stored at position 0 in file 0), follow the natural edge $v.\mathsf{naturalEdge}$ of the current node v until you reach the sink which has no outgoing edge. During the traversal, concatenate *edgeLabel*(e) of every followed edge e in the order of traversal.

The pseudocode for this algorithm is given in Algorithm 2.

```
   Input: Set of encrypted files B, Key K
   Output: Plaintext S
   Data: Set of Nodes V, Set of Edges E, Set of Node Sets N
 1 foreach b ∈ B do                                              // Step 1
 2 |   N := N ∪ D_K(b)
 3 end
 4 nsid := 0
 5 npos := 0
 6 while v := D_K(getFileFromServer(nsid))[npos] do          // For node ...
 7 |   foreach AdjacencyListEntry e ∈ v do      // ...find the matching edge
 8 |   |   if e.isNaturalEdge then
 9 |   |   |   nsid := e.targetNodeSetId
10 |   |   |   npos := e.targetNodePosition
11 |   |   |   S.append(e.edgeLabel)
12 |   |   |   break
13 |   |   end
14 |   end
15 end
16 return S
```

Algorithm 2: Algorithm $\mathsf{Dec}(B, K)$.

Key Generation Algorithm Gen

As the key of our scheme is only applied to E and D, we can use the key generation algorithm of the underlying encryption scheme G to generate the key, that is

$$\mathsf{Gen} = \mathsf{G}.$$

Pattern Matching Protocol π

The protocol π of the XPM-SSE scheme $((\mathsf{Enc}, \mathsf{Dec}, \mathsf{Gen}), \pi)$ describes how the client $\mathfrak{C}$ performs a search operation on the encrypted data $\mathsf{Enc}(S, K)$ which is stored on the server $\mathfrak{S}$. Let v_0 be the root node. To execute a search for w in S on the encrypted data, one follows the path $v_0, \ldots, v_{|m|}$ with $\mathit{edgeLabel}(v_0, v_1) \cdot \ldots \cdot \mathit{edgeLabel}(v_{|w|-1}, v_{|w|}) = w$. If it exists, return $v_{|w|}.\mathsf{numOccurrs}$ of the node at the end of the path and 0 if such a path does not exist. In the latter case, continue to retrieve random nodes.

Pseudocode for π is provided in Algorithm 3. See Figure 4.7 for a depiction of the interaction between client and server.

Input: Graph DAWG(S), String w, Key K
Output: The number of occurrences of w in S
Data: Node v, Boolean *matched*

```
1  Node v := getNode(0, 0, K)
2  Integer maxnodes := ⌊s*/(constsize + |Σ| · entrysize)⌋
3  Integer filenum := 2((2|S| − 1) · constsize + (3|S| − 4) · entrysize)/s*
4  Array[filenum] distribution := [0, ..., 0]
5  for i ∈ {0, ..., |w| − 1} do                    // Scan pattern from left to right
6      foreach AdjacencyListEntry e ∈ v do          // For all outgoing edges of v
7          matched := false                                          // reset variable
8          if w[i] = e.edgeLabel then                  // and find the matching edge
9              v := getNode(edge.targetNodeSetId, edge.targetNodePosition, K)
10             distribution[edge.targetNodeSetId]++
11             matched := true
12             break
13         end
14     end
15     if not matched then                              // w is not a substring of S
16         for j ∈ {i, ..., |w| − 1} do                                 // fix the story
17             repeat fid ← {0, ..., filenum} until distribution[fid] < maxnodes
18             getFile(fid)                  // load file distribution[fid]++
19         end
20         return 0                                      // No occurrences of w in S
21     end
22 end
23 return v.numOccurrs                                                // w occurs in S
```

Algorithm 3: Exact Pattern Matching Algorithm π_w.

4.6.3 Properties

In this subsection, we discuss our scheme's correctness and performance.

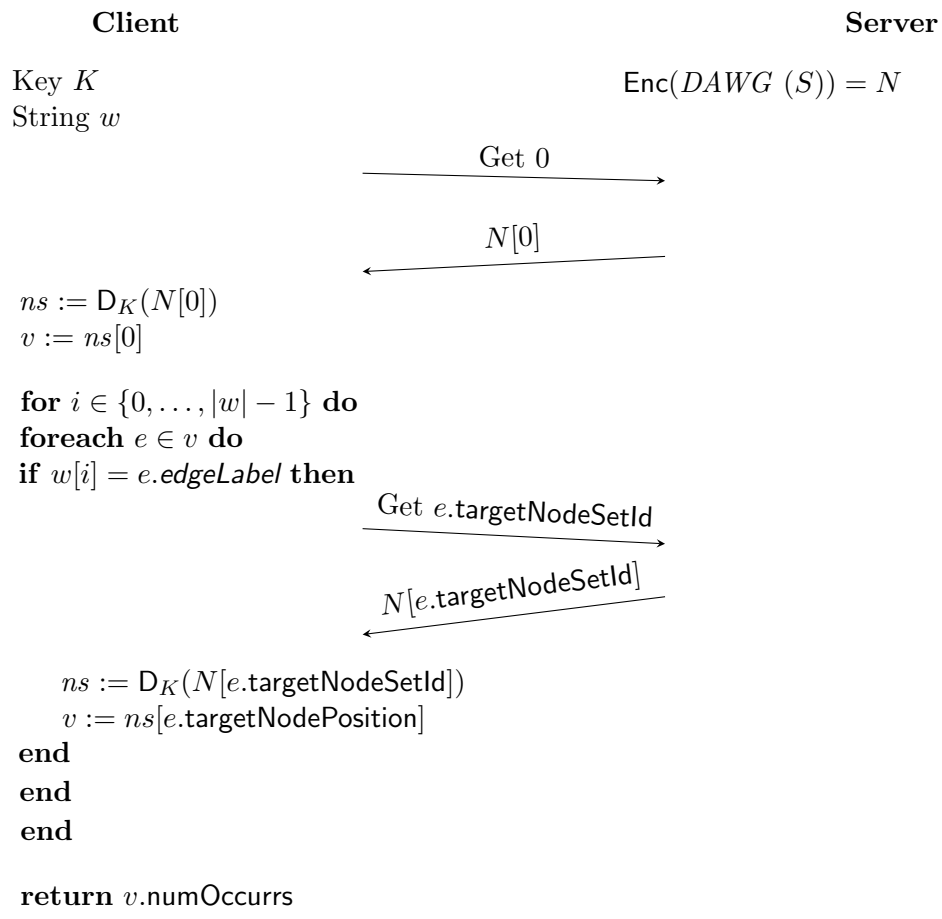

Figure 4.7: General communication pattern for searching S for pattern w. The case of $w \notin substrings_S$ is not covered in this example.

Correctness and Completeness

We now show the correctness and completeness of π.

Theorem 19 (Completeness and Correctness of π)**.** *Let the server $\mathfrak{S}$ be provided with the ciphertext $\mathsf{Enc}(S, K)$ and let the client $\mathfrak{C}$ be supplied with the search pattern w and the encryption key K. Then $\mathfrak{C}$ outputs the exact number of occurrences $|\{i \mid w = S[i..(i + |w| - 1)]\}|$ of w in S after the execution of π_w.*

Proof. As the algorithm π_w always outputs one value, it is trivially complete. To show the correctness of our algorithm, we use a property of the DAWG that follows directly from its definition:

$$w \in substrings_S \Leftrightarrow [w]_S^E \in DAWG(S)$$

We split the proof into two cases of output:

- Case 1 ($\pi_w = 0$): In iteration i the algorithm encountered a node v_i such that there is no outgoing edge e with $\mathit{edgeLabel}(e) = w[i+1]$. Thus, there is no edge $(w[i]_S^E, w[i+1]_S^E)$ in E. From the definition of the edges it follows that $w[i+1]_S^E \notin V$. With the above property of the DAWG, we have that w is not a subword of S and proved that the output is correct.
- Case 2 ($\pi_w \neq 0$): The algorithm found a path $p = v_0, \ldots, v_{|w|}$ with $\mathit{edgeLabel}(v_0, v_1) \cdot \ldots \cdot \mathit{edgeLabel}(v_{|w|-1}, v_{|w|}) = w$. From the definition of the DAWG it follows that $w \in [w]_S^E = v_{|w|}$ and with the above property, w is therefore a subword of S. The output $k = v_{|w|}.\mathsf{numOccurrs} = \pi_w$ is the number of occurrences of w in S by definition, thus the output is correct in this case.

As any output of the algorithm falls into one of these two cases, we showed the correctness of the algorithm. □

Performance and Space Complexity

For the performance analysis in this subsection, we assume a memory access requires one time step. Further, we assume that all integers that depend polynomially on the input length fit into one memory cell. This assumption facilitates the analysis, since pointers, characters and array indices can be modeled with constant memory.

Theorem 20 (Ciphertext Size)**.** *The space complexity of the ciphertext size* $\mathsf{Enc}(S, K)$ *is* $\in \Theta(|S|)$.

Proof. To see how the ciphertext size $|\mathsf{Enc}(S, K)|$ is related to the plaintext size $|S|$, we look at how the data is processed by the encryption algorithm Enc. In the first step, $DAWG(S)$ is generated from S. As Blumer et al. show [29], this generates a graph $DAWG(S)$ for which the number of nodes is bounded by $2|S| - 1$ and the number of edges by $3|S| - 4$.

Steps 2 and 3 update fields in the adjacency list entries and do not add to the ciphertext size. In step 4, every node set is then encrypted to a file which is then padded to size $s^\star > \mathit{constsize} + |\Sigma| \cdot \mathit{entrysize}$. The number of node sets is $|N| = 2((2|S| - 1) \cdot \mathit{constsize} + (3|S| - 4) \cdot \mathit{entrysize})/s^\star$, such that the total size of all files is $2((2|S| - 1) \cdot \mathit{constsize} \cdot s^\star + (3|S| - 4) \cdot \mathit{entrysize}) = \Theta(|S|)$. □

Absolute values of these figures for our implementation can be found in Section 4.6.4.

Theorem 21 (Execution Time of Enc)**.** *The time complexity, i.e. the execution time of* $\mathsf{Enc}(S, K)$*, is in* $O(|S|)$.

Proof. The first step, generating $DAWG(S)$, is bounded by $O(|S|)$ [29]. The second step iterates through all nodes V whose number is bounded by $2|S| - 1$. In each iteration, the set of all possible target node sets has to be calculated. This can be done in $O(|N|) \in O(|S|)$. Step 3 updates values for all edges E, whose number is bounded by $3|S| - 4$. Also, Step 4 iterates through all edges and writes the adjacency lists into files. Lastly, the set of all files is returned. Its size is $|N|$, as there is one file for each node set. In total, the time complexity of the algorithm is bounded by $O(|S|)$. □

Theorem 22 (Communication Complexity of π)**.** *The number of messages exchanged between server* $\mathfrak{S}$ *and client* $\mathfrak{C}$ *while executing* π_w *is* $2|w| = \Theta(|w|)$.

Proof. In Algorithm 3, only the methods `getNode` and `getFile` in lines 1, 9 and 18 require communication with the server. Line 1 is only executed once. The other two lines are included in a `for` loop which is executed at most $|w|$ times. Furthermore, line 9 is also nested within a `foreach` loop, but is at most executed once in this loop because the loop is escaped after the execution of this line. Line 18 is executed only if $matched = \mathsf{false}$ and in this case line 11 has not been executed. So the predecessing line 9 has also not been executed. Thus, when reaching line 18 in iteration i, line 9 has been executed i times, which means that i files have been downloaded. Then, line 18 downloads $|w| - i$ more files before the execution is stopped in line 19. Thus, the algorithm in Algorithm 3 always loads $|w|$ files, sending one request per file and receiving one reply. □

We always download $|w|$ node sets, whether $w \in S$ or not. The aim of this action is to conceal the result from an eavesdropper. If one optimises for performance, it is possible to transfer less files in many cases. If $w \notin S$, this fact becomes clear before $|w|$ files have been transmitted. One can then abort the procedure. Also, the implementation of a local cache for downloaded files may improve performance significantly. In our experiments, file access times dominated local computation time by far.

4.6.4 Implementation and Benchmarks

We have implemented the proposed scheme in Java and conducted experiments on different storage backends[1]. In addition to the local harddisk, we used Amazon's Simple Storage Service (Amazon S3)[2] and the web hosting service of hosting provider 1&1[3]. S3 is a cloud storage solution that is accessed via Representational State Transfer (REST) [83]. 1&1's webspace offers a WebDAV [146] interface and the ability to upload compressed archives and remotely unpack them.

We used the complete genome of the E. Coli bacterium from the Canterbury Corpus [24] in our experiments. Our implementation can handle arbitrary American Standard Code for Information Interchange encoded strings. In this specific case, it would have been possible to specifically tailor the implementation to a smaller alphabet, namely $\Sigma = \{a, t, g, c\}$, and hence reduce memory requirements. However, doing so would have hampered the generality of our results.

For our measurements we used a dual-core laptop computer clocked at 1.3 GHz with 4 GB of Random Access Memory (RAM), the test data was precomputed on a desktop computer with an Intel® Core™ 2 Duo Central Processing Unit clocked at 2.66 GHz and 8 GB of RAM. For Internet access we used a consumer-grade DSL connection.

In this section we provide and discuss results for encryption times, memory requirements, search and download times and size of the encrypted data in comparison to the

[1] At the time of this writing, the source code of our implementation is available online at `https://github.com/jochenrill/sse/`.

[2] `http://aws.amazon.com/s3/`

[3] `http://www.1und1.info/`

original text. All measurements have been taken using Java's internal memory and time measurement methods. In our implementation we use the Advanced Encryption Standard encryption scheme with a key length of 128 bits in the Cipher Block Chaining mode.

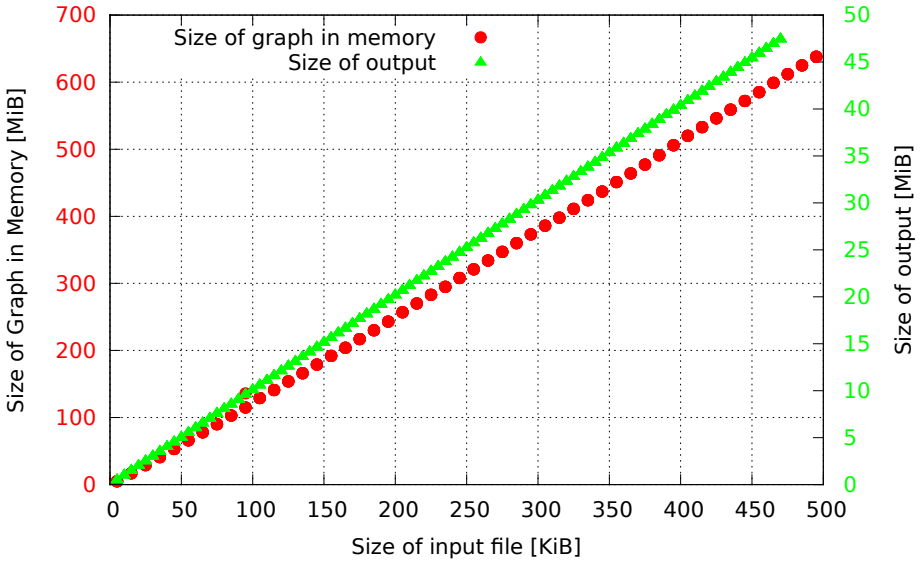

Figure 4.8: Red circles and left y-axis: The memory needed to store the DAWG, depending on input file size. The size grows linearly. Overhead is a factor of roughly 1300. Green triangles and right y-axis: The size of all encrypted data, depending on input file size. Overhead is a factor of roughly 100.

Figure 4.8 shows the memory needed to store the DAWG for a given text size with red circles (left y-axis). The measurement has been taken after processing the entire text and invoking the garbage collector to remove unnecessary objects from the heap. The size grows linearly with the input size.

Figure 4.8 also shows the size of the encrypted data, which consists of the sum of the sizes of the created files, depending on the input size (green triangles, right y-axis). The size of the encrypted data grows linearly with the size of the input text. However, there is an overhead factor of approximately 100. This is due to the fact that the representation of every edge in a node set uses 4 Bytes for each the node set reference and the starting position in a node set and 1 Byte for the edge label. In addition to that, every node stores its number of occurrences, which also is represented as a 4-byte value. As discussed in Section 4.6.2, the data has to be padded to its maximum size. To achieve that, it is assumed that the DAWG is maximal for a given text.

The execution time of a Java program depends on the size of available heap space. It was set to 3 GB in all our benchmarks. Since all files can be generated directly from the graph, the total execution time of the encryption very much depends on the device used to store these files and its write latency and speed. Table 4.1 shows the execution time for generating the graph in comparison to the time it takes to store the files, depending

Input [KiB]	n/o files	Graph [s]	Harddisk [s]	Amazon [s]	1&1 [s]
5	25	0.201	1.853	16.682	8.113
25	111	0.54	7.634	62.773	39.641
45	197	0.941	10.581	117.432	48.295
65	283	1.227	14.07	168.433	70.328
85	369	1.295	19.267	208.856	89.172
105	455	1.673	21.728	247.127	108.876
125	541	1.927	23.538	306.083	129.077
145	627	2.408	27.423	325.488	149.521
165	713	2.15	31.408	388.904	170.332

Table 4.1: The time needed to upload the files in comparison to creating the graph in memory, depending on the input file size. The block size is 28 KiB. The overall execution time is dominated by the time it takes to output the files. Generating the data structure takes very little time in comparison.

on input file size. The measurements concerning Amazon have been taken using the jets3t library[4] for the Amazon S3 Cloud Service. The measurements for the 1&1 web space include packing all files into an archive, uploading it using the sardine library[5] for WebDAV access and remotely unpacking it again. The table shows that the overall execution time is dominated by the time it takes to output the files, whereas generating the data structure takes very little time in comparison. Given the possibility to pack all files into an archive and unpack it on the backend, the performance is drastically better than uploading each file individually, since a full run of the authentication protocol for the backend is needed for each file.

Table 4.2 shows the time it takes to search for a word of a specific length using different storage backends. The search is performed using a randomly selected set of words and substrings from the Canterbury Corpus sample. As discussed in Section 4.6.2, the length of the pattern w equals the number of files which are transferred. Again, the overall execution time is dominated by the time it takes to access a file on the storage device, which can be seen when comparing the results to the time it takes to search a word when using the local harddisk as the backend. To compare our scheme, we also implemented the trivial approach of uploading the encrypted text and downloading it again to perform a search on it using the Boyer-Moore algorithm [37], provided by the stringsearch library[6] for each backend. The performance of our scheme can compete with that of the trivial solution for short queries. With long query strings, server response times hamper the performance of our scheme. The possibility to establish persistent connections with the backend would drastically improve the performance of our scheme.

[4] http://www.jets3t.org/
[5] http://code.google.com/p/sardine/
[6] http://johannburkard.de/software/stringsearch/

$\|w\|$	Amazon [s]	Amazon trivial [s]	1&1 [s]	1&1 trivial [s]	Harddisk [s]
2	4.29	5.46	2.02	4.50	0.32
3	4.70	5.40	2.13	4.50	0.39
4	5.27	5.38	2.29	4.49	0.44
8	6.61	5.36	2.72	4.43	0.47
16	9.47	5.37	3.62	4.44	0.64
32	16.68	5.46	5.47	4.28	0.77
64	29.28	9.12	9.14	4.38	1.20

Table 4.2: The average search time using different backends in comparison to a trivial search. Values show the average of 10 samples. The overall execution time is dominated by the time it takes to access a file on the storage device.

4.6.5 Security

We now analyse the security of our construction.

Data Privacy

In Section 4.6.2, we proposed a XPM-SSE scheme. We now analyse its security properties, namely that of Data Privacy. We define $\mathsf{PrivK}^{\mathsf{cppa}}$, a flavor of D-IND (Definition 14). It allows for the adversary to submit multiple search queries, but to receive only one transcript of the client-server communication of his choice.

Also, we grant the adversary access to any transcripts for plaintexts he supplies.

We define the security game $\mathsf{PrivK}^{\mathsf{cppa}}$ as a direct instantiation of $\mathsf{D\text{-}IND}^{\mathcal{A},R_d,n_1,n_2,n_3}_{(\mathsf{Gen},\mathsf{Enc},\mathsf{Q})}(k)$ with parameters $R_d = \{d_0, d_1\}$, $|d_0| = |d_1|$, $n_1 = poly(k)$, $n_2 = 1$, and $n_3 = poly(k)$ (where *poly* is any fixed polynomial).

Security Game 16 ($\mathsf{PrivK}^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$)**.**

1. The experiment chooses a key $K \leftarrow \mathsf{Gen}(1^k)$ and a random bit $b \leftarrow \{0,1\}$.
2. The adversary receives oracle access to $\mathsf{view}^{\pi_\cdot}_{\mathfrak{S}}(\mathsf{Enc}(\cdot, K))$, and continues to have access to it.
3. $\mathcal{A}$ outputs two plaintexts m_0 and m_1 of the same length to the experiment.
4. $\mathcal{A}$ outputs a number of queries $x_0, \ldots, x_q$ and an integer $i \in \{0, \ldots, q\}$ to the experiment.
5. The queries $x_0, \ldots, x_q$ are evaluated in that order.
6. $\mathcal{A}$ is given the view on the challenge ciphertext to query i: $\mathsf{view}^{\pi_{x_i}}_{\mathfrak{S}}(\mathsf{Enc}(m_b, K))$.
7. $\mathcal{A}$ submits a guess b' for b.

The result of the experiment is 1 if $b' = b$ and 0 else.

Definition 30 (Document Privacy)**.** A scheme $((\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}), (\mathfrak{S}, \mathfrak{C}))$ has Document Privacy if for all PPT adversaries $\mathcal{A}$ there exists a negligible function *negl* such that

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k) = 1\right] \leq \frac{1}{2} + negl(k).$$

We now present our result that our construction SEDAWG from Section 4.6.2 has Document Privacy.

Theorem 23 (SEDAWG has Document Privacy)**.** *Our construction SEDAWG has Document Privacy if* $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ *is a IND-CPA secure scheme.*

Proof. Suppose $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ is a IND-CPA secure encryption scheme and $\mathcal{A}$ is a PPT adversary who can win game $\mathsf{PrivK}^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ (Security Game 16) with non-negligible probability. In $\mathsf{PrivK}^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, $\mathcal{A}$ receives one encryption $\mathsf{Enc}(m_b, K)$ and one server's view to a protocol run $\mathsf{view}^{\pi_{x_i}}_{\mathfrak{S}}(\mathsf{Enc}(m_b, K))$. He then outputs a guess b' for b.

The server's view of a protocol run consists of the IDs of the requested files and the files themselves. Consider a modification of the game, $\mathsf{PrivK}'^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$: Instead of $\mathsf{Enc}(m_b, K)$ and $\mathsf{view}^{\pi_{x_i}}_{\mathfrak{S}}(\mathsf{Enc}(m_b, K))$, the adversary is sent the encryption of a zero string with the same length as m_0 and m_1: $\mathsf{Enc}(0^{|m_0|}, K)$. Also, $\mathsf{view}^{\pi_{x_i}}_{\mathfrak{S}}(\mathsf{Enc}(m_b, K))$ is altered in such a way that the transmitted files are taken from $\mathsf{Enc}(0^{|m_0|}, K)$ instead of $\mathsf{Enc}(m_b, K)$ ("so the story fits"). Call $\mathcal{A}$'s output from the modified game b''. Because $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ is IND-CPA secure, $\mathcal{A}$ cannot distinguish $\mathsf{Enc}(m_b, K)$ from $\mathsf{Enc}(0^{|m_0|}, K)$. Hence, b'' is statistically close to b'.

In a second modification $\mathsf{PrivK}''^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, we replace the IDs from the server's view with random IDs (chosen uniformly at random from the set of available file IDs, without replacement), except for the first request—the file ID that is first requested is always 0. Call $\mathcal{A}$'s output from this modified game b'''. Because the original file IDs have been chosen in the same manner as the IDs in our modified game (see the description of Enc in Section 4.6.2, Step 2, and the pseudocode in Algorithm 1) and the adversary is only supplied with one view, her output b''' is again statistically close to b''.

Following our argument, if the adversary's output in $\mathsf{PrivK}^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ is correlated to b, its output in $\mathsf{PrivK}''^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ is also correlated to b. But in $\mathsf{PrivK}''^{\mathsf{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, $\mathcal{A}$ receives no input that correlates with b. This is a contradiction. □

Query Privacy

In Section 4.6.2 we presented SEDAWG, a XPM-SSE scheme that has Document Privacy, i.e. keeps the outsourced string private from the server, even in the presence of a (single) query. It seems natural to ask whether the construction also has some form of Query Privacy (Definition 15) and hides the search pattern. In this subsection we investigate that question.

Theorem 13 (Section 4.5.1) states that Query Privacy is equivalent to (single-server) PIR. Private Information Retrieval schemes, on the other hand, have a lower bound for the server's computational complexity which is linear in the document size [185]. Assuming the search pattern is substantially shorter than the document itself, SEDAWG imposes a far lower computation complexity on the server: it is linear in the length of the search pattern. It follows that SEDAWG cannot have Query Privacy. Indeed, one can easily construct a PIR scheme given an XPM-SSE scheme that has Query Privacy.

Theorem 24. *Any efficient XPM-SSE scheme that has Query Privacy yields an efficient PIR scheme.*

Proof. To prove the theorem, we give a construction using a XPM-SSE scheme and show that it achieves PIR. We show how to construct a string from a binary database with n entries. We then explain how to perform a database query and how to interpret the result.

i	0	1	2	3	4	5
$D[i]$	1	0	1	0	1	0

$\Rightarrow$ `!000$ !010$ !100$`

Figure 4.9: Example database with the corresponding string to be used for a database retrieval with a XPM-SSE scheme.

The alphabet for the XPM-SSE construction is $\Sigma = \{0, 1, !, \$\}$. Construct S as follows: Start with an empty string S. Iterate i through $\{0, \ldots, n-1\}$, and, if $D[i] = 1$, append the binary encoding of $!i\$$ (with leading zeroes to hide the size of i), to S. Output S. (See Figure 4.9 for an example.) To retrieve bit i from S (stored on the server), run the search protocol for the binary encoding of $!i\$$. If the search is a success, the retrieved bit is 1 and 0 otherwise.

The query results of the XPM-SSE scheme are, per definition, complete and correct. By construction, the binary encoding of i is $\in S$ iff $D[i] = 1$. It follows that the above construction delivers correct results.

Assume a successful adversary $\mathcal{A}$ against the PIR construction above. We show that a successful adversary $\mathcal{A}_Q$ against $\mathsf{Q\text{-}IND}$ exists.

As per Definition 21, there are $n, N \in \mathbb{N}, d \in \{0,1\}^n, i, j \in \{0, \ldots, n-1\}$ such that for all $k > N$

$$\left|\Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d, K))) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d, K))) = 1\right]\right| \geq \frac{1}{\max(k, n)^c}.$$

We discuss the case that

$$\Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d, K))) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d, K))) = 1\right] \geq \frac{1}{\max(k, n)^c}.$$

The other case is symmetric. (For each adversary $\mathcal{A}_Q$ there is an "inverted" adversary $\overline{\mathcal{A}_Q}$ that outputs the exact opposite.)

$\mathcal{A}_Q$ outputs π_i and π_j as the challenge, requests $\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d,K))$ from the challenge oracle and returns $\mathcal{A}$'s output $\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_b}(\mathsf{Enc}(d,K)))$ as his guess b'. Observe that $\mathcal{A}$ is run on $\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d,K))$ if $b=1$, and on $\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d,K))$ else $(\star)$.

Let $\Pr[b' \leftarrow \mathcal{A}]$ denote the probability that $\mathcal{A}$ outputs guess b' in $\mathsf{Q\text{-}IND}$. Now consider

$$
\begin{aligned}
&\Pr\left[\mathsf{Q\text{-}IND}^{\mathcal{A}_Q}(k)=1\right] = \Pr[b=0 \wedge 0 \leftarrow \mathcal{A}_Q] + \Pr[b=1 \wedge 1 \leftarrow \mathcal{A}_Q] \\
&= \frac{1}{2}\Pr[0 \leftarrow \mathcal{A}_Q \mid b=0] + \frac{1}{2}\Pr[1 \leftarrow \mathcal{A}_Q \mid b=1] \\
&= \frac{1}{2}\left(1 - \Pr[1 \leftarrow \mathcal{A}_Q \mid b=0]\right) + \frac{1}{2}\Pr[1 \leftarrow \mathcal{A}_Q \mid b=1] \\
&= \frac{1}{2} + \frac{1}{2}\left(\Pr[1 \leftarrow \mathcal{A}_Q \mid b=1] - \Pr[1 \leftarrow \mathcal{A}_Q \mid b=0]\right) \\
&\overset{(\star)}{=} \frac{1}{2} + \frac{1}{2}\left|\Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_j}(\mathsf{Enc}(d,K)))=1\right] - \Pr\left[\mathcal{A}(\mathsf{view}_{\mathfrak{S}}^{\pi_i}(\mathsf{Enc}(d,K)))=1\right]\right| \\
&\geq \frac{1}{2} + \frac{1}{2 \cdot \max(k,n)^c}.
\end{aligned}
$$

Thus, $\left|\Pr\left[\mathsf{Q\text{-}IND}^{\mathcal{A}_Q}(k)=1\right] - \frac{1}{2}\right|$ is not negligible in k. □

5 Electronic Voting

> It may be the warriors who get the glory. But it's the engineers who build societies.
>
> *(B'Elanna Torres,* Star Trek: Voyager, Flesh and Blood*)*

Free elections are the backbone of a democracy. In traditional elections, voters fill out their ballot in a voting booth to ensure the privacy of the vote. Because traditional presence elections are tedious, there is a significant interest in carrying out elections over the Internet. It seems impossible to give the same privacy guarantee that a voting booth does in an online setting. Further, advances in consumer electronics even call the privacy of the voting booth into question. To sell their vote, voters can easily record their choice with their smartphone [26, 195].

To mitigate these problems, several solutions have been developed to make elections resistant against such attacks. *Fake credentials* [121, 122] and *revoting* [195, 197] are two of these. While fake credentials successfully disable the adversary from obtaining a voter's credential by coercion, they are contradictory to a meaningful response whether the ballot has been cast successfully. Revoting, on the other hand, allows for a meaningful response.

The idea behind revoting is that, if a voter is allowed to supersede her vote arbitrarily many times, she effectively cannot be coerced. We stress, however, that this proposed revoting must be perfectly deniable—no party, including all servers, must be able to tell whether a coerced voter has evaded coercion by revoting.

As a side-effect, deniable revoting is applicable to elections which allow the voter to change her mind after casting a legit vote. An example for this is delegated voting [100, 161], in which the voter can delegate her vote to a so-called proxy, who then votes in her stead. She can later overwrite this delegation with a new delegation or by casting a vote by herself.

We investigate deniable revoting as a strategy for evading coercion. We build upon the well-known work of Juels, Catalano, and Jakobsson [121, 122]. Specifically, we adapt their definitions of correctness and coercion resistance to include revoting and its deniability. While their construction enables voters to re-cast their vote, their security model does not account for revoting as a strategy to evade coercion. Indeed, their construction allows the adversary to observe whether a ballot is superseded. We present a tallying protocol which allows voters to deniably re-cast their votes, thus making revoting a viable strategy to evade coercion. The macrostructure of our protocol is the same as that of Juels et al., hence it is compatible with fake voter credentials. To facilitate a simpler exposition, we do not investigate the possibility of offering both evasion strategies simultaneously.

We present a method for counting votes which ensures the deniability of revotes, and prove the correctness of our protocol as well as the deniability of revoting. This work seeks to serve as a proof of concept and does not claim to provide an efficient solution.

Existing protocols that allow revoting either do not achieve both deniability and public verifiability at the same time, or require the voter to remember some information about previous votes [145]. To the best of our knowledge, we are the first to propose a publicly verifiable method for enabling deniable revoting without requiring the voter to save state between votes.

There is one caveat to the revoting approach. If a coercer observes the voter's behaviour until after the polls close, the voter cannot re-cast her vote and thus revoting does not work as a strategy for evading coercion. This caveat applies to the strategy in general and thus cannot be remedied by any protocol. For this reason, our adversarial model does not allow the adversary to observe the voter's behaviour indefinitely—he must give up his control before the polls close. On the other hand, a coerced voter does not employ any evasion strategy during adversarial observation—she submits to the adversary's instructions completely.

After discussing related work in Section 5.1, we sketch our construction in Section 5.2 and discuss authentication in 5.3. We introduce the adversarial model and underlying assumptions in Section 5.4. In Section 5.5, we adapt the security notions of Juels et al. [122] to model deniable revoting. We present our voting scheme and prove its security in Section 5.6.

This chapter has been published in a journal article [2].

Juels, Catalano, and Jakobsson's Approach Let us briefly recap the approach of Juels, Catalano, and Jakobsson. To cast a vote, voters post their ballot on a public bulletin board, together with proofs of knowledge of their credential and the validity of their choice. After the voting phase, the tallying authority computes the tally in five steps:

1. Check proofs: The tallying authority checks all proofs associated with the ballots and discards ballots with invalid proofs.

2. Remove duplicate ballots: At most one ballot per credential is kept. To identify duplicate ballots, *Plaintext Equivalence Tests (PETs)* are employed on the encrypted voter credentials. For a pair of ballots with identical voter credentials, a predetermined policy decides which ballot is kept.

3. Shuffle: The remaining ballots are shuffled.

4. Check credentials: The tallying authority discards ballots with invalid voter credentials.

5. Tally: All remaining valid ballots are decrypted and counted.

This solution uses fake credentials as a strategy to evade coercion. Revoting is supported in the sense that double votes are handled. However, voters cannot plausibly deny having revoted—this is not the aim of the construction. Imagine an adversary forcing a voter to

cast a certain ballot in his presence. Since the ballots are only shuffled (Step (3)) after duplicates have been removed (Step (2)), the adversary can easily monitor if his observed ballot is deleted. This way, the adversary can deduce that the coerced voter has later cast a ballot with the same credential. This does not impose a problem in Juels et al.'s approach—the observed ballot was cast with a fake credential, and does not need to be superseded to evade coercion. To employ revoting as a strategy for evading coercion however, the sorting process must ensure its deniability. A first step is to shuffle the list of ballots before removing duplicates. This conceals the chronological order of ballots, however. For two ballots cast with the same credential, it cannot be determined anymore which one was cast more recently. We devised a method of an "encrypted labeling" which allows voters to privately re-cast their ballot, while preserving enough information about the chronological order of ballots.

5.1 Related Work

One can think of coercion resistance as an extension of ballot privacy. The basic idea is that if no adversary can learn the voter's choice, he cannot blackmail her into picking a particular choice. Further, when the voter receives no documentation of her vote—i.e. when the scheme is *receipt-free*—she cannot even willingly collaborate with the adversary by selling proof of her choice. The notion of coercion resistance even further takes into account that the adversary might try to influence the casting of the vote, which is a reasonable assumption for Internet voting schemes. Delaune et al. [68] show that coercion resistance implies receipt freeness, which in turn implies vote privacy. More recent work by Küsters et al. [143] reveals that the relationship between privacy and coercion resistance is more subtle.

Coercion Resistance Various definitions of coercion resistance and verifiability of e-voting schemes exist in the literature [40, 68, 141, 144, 162, 194]. Several of them are also applicable to our voting scheme. However, because our work aims to extend the work of Juels et al. [122], we stay close to their model for better comparability.

By introducing the caveat coercitor [101], Grewal et al. relaxed the requirement of coercion resistance to coercion evidence: a voter can be coerced, but the coercion is detectable. Their work addresses the problem of silent coercion, where the voter loses her credential to the adversary without noticing. With their approach, changing one's mind and overwriting a legit ballot is not possible (it would be recognised as a coercion attempt). Our work, in contrast, does not investigate the problem of silent coercion.

A number of voting schemes have been proposed that provably offer coercion resistance. For example, Bohli et al.'s Bingo Voting [33, 105] is coercion-resistant, except for forced abstentions, as Küsters et al. [144] show. They also prove the coercion resistance of Scantegrity II [55, 142], with the same exception. Kempka [133] proposes a fuzzy tally representation that maintains coercion resistance even when one accounts for forced abstentions. We note that fuzzy tally representations are compatible with a wide array of constructions, including that of Juels et al. and our own. Delaune et al. [68] show that

the scheme due to Lee et al. [149] is coercion-resistant. Juels et al.'s construction [122] is also coercion-resistant. (On a side note, it is not immediately apparent how the different notions of coercion resistance relate.)

Revoting The importance, advantages and challenges of revoting were discussed by Volkamer and Grimm [197], as well as by Post [195], though no solution for deniable revoting was given. Several Internet voting schemes allow for revoting, not necessarily only as a defense against coercion. Neither of the voting schemes known to us uses a revoting strategy which is both deniable and publicly verifiable, while not requiring the voter to save state between votes. For example, in Kutyłowski and Zagórski's scheme [145], in order to revoke her vote, the voter must reproduce the random challenge from the casting protocol. They further do not provide a formal proof of coercion resistance for their scheme. In a similar vein, Spycher et al. [189] propose a hybrid voting system where the voter can overrule her online vote in person. To this end, she must produce a re-encryption of the ballot formerly posted.

The election of the president of the Université Catholique de Louvain in 2008 was conducted using an adapted version of Helios [12]. Because of the election's low risk of coercion attempts, revoting was supported mostly for convenience. Newer ballots of a voter substituted her older ballots on the bulletin board, so revoting was not deniable. The voting scheme designed for the 2011 local government elections of Norway [88] supports non-deniable revoting. Verification is done by auditors, who can see the number of votes of the same voter. The Riigikogu Election Act requires the possibility to revote for the Estonian election [175], but to the best of our knowledge, no deniable solution is offered [152].

As described in more detail above, the work of Juels et al. [121] supports revoting in principle, but since their work concentrates on fake voting credentials as a strategy to evade coercion, no deniable revoting strategy is proposed. Fake voting credentials are an important alternative approach for achieving coercion resistance. After the work of Juels et al. [121] and its implementation Civitas [165], two password-based voting schemes, Selections [58, 59] and Cobra [77], were published. They use panic passwords as fake credentials, which can easily be created by a human. Selections is proven secure in an adapted version of the model introduced by Juels et al. [122]. Verifiable revoting is possible in both Selections and Cobra, but the number of votes cast with the same credential, or the fact that a certain ballot has been superseded, is not hidden. Efficiency improvements of Juels et al.'s voting scheme [122] were proposed by Araújo et al. [15], using another form of fake credential, and by Smith [187], who introduced a faster removal procedure of duplicate ballots. Revoting is supported, but not deniable. The fake credential approach avoids the problem of coercions occuring shortly before the end of the voting phase. However, the voter receives no reliable confirmation whether her vote was cast properly. Particularly, systems that rely on human memory suffer from a high probability of error. Panic passwords for example pose the risk that the voter uses a fake credential without noticing.

5.2 Achieving Deniable Revoting

In this section we sketch the challenges of deniable revoting, and how to overcome them, in more detail. For revoting to be suitable as a strategy for evading coercion, the voter must be able to *deniably* do so. Intuitively, we say a voter can deniably re-cast her vote if no adversary can, by his observations, tell whether she did. At the same time, we still require the tally to be computed correctly. In particular, at any time during the voting period, for each participating voter who has already cast a vote, there is exactly one *valid* ballot (i.e. one ballot which counts as long as it is not superseded), so a re-cast ballot must invalidate this one ballot with matching voter credentials, and become the new valid ballot corresponding to this credential. These requirements impose several privacy challenges on the tallying process.

More concretely, the deniability requirement of the revoting process implies that the adversary must not be able to figure out how often a certain voter has re-cast her ballot. Further, he must not even infer which of the voters did revote at all. Also, if he can tell whether any particular ballot is part of the final tally, revoting is not deniable. Nevertheless, to maintain the universal verifiability of the tally, the correctness of each tallying step must be transparent. To give an intuition for the subtle challenges of authenticating re-castable ballots, we describe two exemplary attacks.

For the first example recap the attack described in the former section. There, the adversary is forcing the voter to cast a ballot in his presence. He can then observe the ballot appearing on the bulletin board right after. If this particular ballot is deleted or marked as invalid in the tallying process, he can deduce that his coercion attempt was not successful. As mentioned above, we employ encrypted labelling and re-encryption mixes after the casting phase to solve this problem.

While Step (2) in Juels et al.'s construction (see Section 5) does hide the identity of voters with duplicate ballots, it leaks information about the number of removed duplicates per credential. Consider the following attack, which is similar to the "1009 attack" described by Warren Smith [187]: The adversary forces the voter to vote, in his presence, a number of times no other voter is likely to vote, e.g. exactly 1009 times. During the tallying phase he then verifies that the tallying authority indeed identifies 1009 duplicate ballots for some credential. This becomes fatal in the absence of fake credentials[1]: the adversary can be sure that the coerced voter did not revote after casting her vote in the adversary's presence. Consequently, a tally which supports deniable revoting must even hide the number of revotes *any* voter has cast. We achieve this by anonymously checking ballots two-by-two for matching voter credentials, to avoid grouping ballots with the same credential.

As we detailed above, the ballots on the bulletin board must be shuffled before discarding superseded ballots. Because the chronological order of ballots is not retained in the shuffling process, we "memorise" the information whether a ballot was superseded with an encrypted label: Before shuffling, for each ballot we calculate an encrypted value

[1] As Smith describes, this attack is fatal for fake credentials, too: if the adversary can infer that a heap with 1009 ballots is discarded because of an invalid credential, the adversary knows the credential presented to him was a fake one.

o_i which is computed in a verifiable way by comparing voter credentials $^{\mathrm{pk}_i}/_{\mathrm{pk}_j}$ between the current ballot b_i and each more recent ballot b_j. The ballots themselves are not shuffled until o_i is computed for all ballots. Only then we shuffle the encrypted choice together with the o_i tag.

$$\begin{array}{rccccccc} 12 = & 81 & \cdot & 53 & \cdot & 87 & \cdot & 48 \mod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 1 = & 1 & \cdot & 1 & \cdot & 1 & \cdot & 1 \mod N \end{array}$$

(a) Ballot is not superseded. No factor equals 1.

$$\begin{array}{rccccccc} 25 = & 81 & \cdot & 53 & \cdot & 1 & \cdot & 48 \mod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 139 = & 1 & \cdot & 1 & \cdot & 139 & \cdot & 1 \mod N \end{array}$$

(b) Ballot is superseded once. One factor equals 1.

Figure 5.1: One cannot tell from a product if one of the factors was = 1. To exploit the homomorphic property of the encryption, we swap the encryption of an arbitrary number with an encryption of a 1 and vice versa. Now one can tell from the product whether one of the factors was $\neq 1$.

We briefly sketch our construction. To make the duplicate test private, we mark each ballot with the encrypted information o_i whether there is another ballot that supersedes it. We compute o_i by comparing each ballot with each subsequent ballot. To do so in a publicly verifiable and private manner, we run an Encrypted Plaintext Equivalence Test (EPET) on the credentials. An EPET receives two ciphertexts as input, and outputs an encryption of a 1 if the contained plaintexts are equal, and the encryption of an arbitrary number otherwise. We compare ballot i with each subsequent ballot by comparing its voter credential $\mathsf{Enc}(pk_i)$ to the credentials $\mathsf{Enc}(pk_{i+1})$, $\mathsf{Enc}(pk_{i+2})$, etc. using an EPET.

A ballot is superseded if and only if its voter credential is equal to that of *at least one* newer ballot. We seek to exploit the homomorphic property of the encryption to aggregate the comparison results without revealing the result of any single comparison. A single EPET result is the encryption of a 1 iff the voter credentials of the ballots match. Thus, our goal is to determine for a number of EPET results whether at least one of them encrypts a 1. We cannot exploit the homomorphic property of the encryption directly, since a factor of 1 does not change the product (see Figure 5.1). As a remedy, we "convert" the values: We swap an encryption of a 1 with an encryption of an arbitrary number and vice versa. Now the product o_i tells us whether at least one factor is not equal to 1—if so, $o_i \neq \mathsf{Enc}_{\mathsf{hom}}(1)$ and the ballot is superseded, otherwise $o_i = \mathsf{Enc}_{\mathsf{hom}}(1)$ and the ballot is counted. We stress that the content of o_i is hidden by the encryption. Before checking o_i to potentially remove a ballot, the ballot's connection to its appearance on the bulletin board at the time of casting is broken by shuffling and re-encryption.

Our protocol results in a list of ballots containing the newest, valid, *encrypted* choice of each voter, and security is proven up to this point. From there, ballots can be tallied in an arbitrary way, using standard techniques like decryption mixes or homomorphic tallying, depending on the form of the choice. Security in combination with our protocol is then given by the security of the tallying procedure. As a consequence, our protocol supports arbitrary election formats, including Single Transferable Vote (STV) [107] or write-in

candidates. However, our security definitions do not take into account information which is leaked by the tally result itself, which is possibly published in the form of decrypted votes. Information leaks like names of write-in candidates or a preference order can impede a voter's privacy. This problem is shared by any voting scheme which publishes the tally result or decrypted ballots, and needs to be accounted for separately by the tallying procedure.

5.3 Authentication for Coercion-Resistant Voting Schemes

All voting schemes use some method to authenticate voters. Even in elections where everybody is eligible to vote, authentication is necessary to make sure that of each voter only one vote is counted. Voter authentication can—in principle—follow any of three paradigms: authentication using something you know, something you have, or something you are. There are two properties of the authentication mechanism which are of grave importance for any voting scheme, regardless of the paradigm used: First, the adversary must not be able to impersonate the voter (e.g. by learning her secret key). Second, the adversary must not be able to prevent the voter from authenticating (e.g. by stealing her authentication token). We call an authentication *inalienable* when both properties hold. They are necessary conditions for the incoercibility of the voter. After a ballot is cast however, authentication and thus its inalienability lose their relevance. In an election which offers revoting, ballots are only ultimately cast when the polls close. Hence, any voting scheme that offers revoting requires the inalienability of the authentication until the polls have closed.

Revoting Versus Fake Credentials Fake credentials and revoting are complementing strategies. While handing out fake credentials protects the confidentiality of the voter's true credentials, re-casting ballots "undoes" an adversary's coercion attempts. If the adversary cannot learn whether the voter has re-cast her ballot, revoting is a valid strategy for evading coercion.

A voting scheme can only support fake credentials if it does not provide any feedback on whether the voter authenticated successfully. Consequently, the voter does not learn whether her ballot was cast correctly. As Juels et al.'s construction uses fake credentials as a countermeasure against coercion, it has this property. Since our construction is an extension of theirs, their strategy for producing fake keys also works for our scheme. On the other hand, to give feedback about the success of authentication, one can publish the list of all registered voters and their public keys before the voting period—eliminating fake credentials as a strategy for evading coercion.

The fake credential strategy is an effective means to make the authentication inalienable, thus helping to achieve coercion resistance. For the strategy to work however, the adversary must not take total control over the voter's actions, even if only temporarily. On the other hand, the revoting strategy requires an inalienable authentication, but achieves a form of coercion resistance that is robust even against temporary-but-perfect corruption.

5.4 Adversarial Model and Assumptions

Our protocol is divided into five phases: A setup phase, a registration phase, a publication phase, the voting phase, and finally a tallying phase. We describe the phases in detail in Section 5.6.3. The tallying authority gets a key pair $(PK_{\mathcal{T}}, SK_{\mathcal{T}})$ during the setup phase. The secret key $SK_{\mathcal{T}}$ is shared among the servers which make up the tallying authority. In the registration phase a registrar $\mathcal{R}$ creates a public key secret key pair (pk_i, sk_i) for each voter.

Let b_i denote a ballot which is published on a bulletin board $\mathcal{BB}$, where $i \in \{1, \ldots, n\}$ and let ts_i denote the point in time when b_i has been published. The ballot represents the voter's choice $\beta_i \in \mathcal{C} = \{c_1, \ldots, c_{n_C}\}$. We describe the ballot in detail in Section 5.6.3. Overall n_V voters participate in the election. Let n_A denote the number of voters which are completely controlled by the adversary $\mathcal{A}$. Considering the adversary tries to coerce exactly one voter, there are $n_U = n_V - n_A - 1$ voters which add noise to the tally $\mathbf{X}$. The noise (choices of these voters) is defined by the distribution D_{n_U,n_C} which we describe in Section 5.5.

In the experiments $\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}$, $\mathsf{Exp}^{\mathsf{ver}}_{ES,\mathcal{A}}$, $\mathsf{Exp}^{\mathsf{revoting\text{-}c\text{-}resist}}_{ES,\mathcal{A},H}$, $\mathsf{Exp}^{\mathsf{revoting\text{-}c\text{-}resist\text{-}ideal}}_{ES,\mathcal{A},H}$ we use security parameters k_1 for the registration, k_2 for the voting function, and k_3 for the tallying process.

5.4.1 Adversarial Model

The adversary may corrupt a minority of the servers forming the tallying authority. He may also corrupt a number of voters and act in their place. Then, the adversary selects one (uncorrupted) voter he tries to coerce. In contrast to the model of Juels et al., we allow the coercion to be perfect—the coerced voter does exactly as told by the adversary. The adversary does not learn the voter's secret key, however. (See Section 5.4.2 for details on this assumption.)

Further, we require all adversarial control of the coerced voter, including direct observation, to end before the polls close. More concretely, we enable the coerced voter to re-cast her vote in secret. Intuitively, if the adversary cannot infer whether the coerced voter has re-cast her vote after his coercion attempt, we say the scheme offers *deniable revoting*. Clearly, revoting offers no protection against adversaries who observe or control the voter until after polls have closed.

5.4.2 Assumptions

We rely on various assumptions for our construction.

- Voter List. We assume an accepted voter list, i.e. there is a general agreement on who is allowed to vote.
- Bulletin Board. As is common for electronic voting schemes, we assume a publicly accessible append-only bulletin board. Anyone can read and append to the bulletin board, but no party can remove data from it.

- Authentic Timestamps. To ensure the correct order of votes, we rely on correct timestamps. Without loss of generality, we further assume that timestamps are unique.
- Anonymous Channel for Voters. To cast a vote, voters post their ballot to the bulletin board during the casting phase. For the votes to remain anonymous, we assume an anonymous channel the voters have access to.
- Inalienable Secret Credentials. In our construction, with each voter we associate a secret that she uses to authenticate her vote. The secret is used as a signature key for ensuring the integrity of the ballot. Also, the corresponding public key is contained on the ballot in encrypted form. In a practical realisation of our scheme, one would have to take special precautions to guarantee that the adversary can neither learn the secret credential nor deny the voter access to it. Realising this assumption is outside the scope of this work. However, we point out that the assumption can be argued for: The election secret could be stored on a tamper-proof device that also serves a different function (e.g. a federal identity card), such that the voter cannot reasonably be without it. Voters would have reservations against handing out such a device (see also Section 5.3).

5.5 Security Notions

To model voter behaviour when one is allowed to re-cast one's vote, we define D_{n_U,n_C} to be a distribution over all vectors over all (bounded) series of all candidates, including abstentions and invalid votes. Let ϕ denote the null ballot (abstention) and λ an invalid ballot. Then D_{n_U,n_C} is a distribution over vectors $((\beta_1)_j, (\beta_2)_j, \ldots, (\beta_{n_U})_j), \beta_i \in (C \cup \{\phi, \lambda\})$. For technical reasons, the length of the vote series is bound by a constant, lest the length of the voter's choices exceeds the runtime of all machines involved. In practice, one can imagine this bound to be the number of nanoseconds between the start of the voting period and its end, for example.

Further, we define the number of uncertain votes, i.e. votes cast by voters not under adversarial control or coercion, as $n_U := n_V - n_A - 1$. For any experiment $\mathsf{Exp}^x_{\mathcal{A}}$, where $x \in \{\mathsf{corr}, \mathsf{ver}, \mathsf{revoting\text{-}c\text{-}resist}, \mathsf{revoting\text{-}c\text{-}resist\text{-}ideal}\}$, we define the adversary's success probability as $\mathsf{Succ}^x_{\mathcal{A}}(k_1, k_2, k_3, n_V, n_C) := \Pr[\mathsf{Exp}^x_{\mathcal{A}}(k_1, k_2, k_3, n_V, n_C) = 1]$. All algorithms are implicitly assumed to be Probabilistic Polynomial Time (PPT).

Let, for the remainder of the chapter, `register`, `vote`, `tally` be shorthand for voter registration, the voting procedure, and the tallying procedure, respectively.

5.5.1 Correctness

Our notion of correctness follows that of Juels et al. We model voters not as posting one ballot, but a series of ballots. The adversary may corrupt a number of voters and vote in their place. We call a tally correct when, regardless of the behaviour of corrupted parties, the last ballot, and only the last ballot, of each voter is counted. See Security Game 17 for Experiment $\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}$.

Intuitively, Security Game 17 works as follows:

1. The voters are registered.
2. $\mathcal{A}$ corrupts voters.
3. $\mathcal{A}$ chooses votes for uncontrolled voters.
4. The honest voters cast their ballots.
5. $\mathcal{BB}$ is tallied honestly, leading to the grand total $\mathbf{X}$ and a proof P.
6. The adversary posts additional ballots on $\mathcal{BB}$.
7. A second grand total $\mathbf{X}'$ (and corresponding proof P) is computed.

The adversary wins if $\mathbf{X} \neq \mathbf{X}'$.

Security Game 17 ($\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$)**.**

1. $\{(sk_i, pk_i) := \mathtt{register}(SK_{\mathcal{R}},\ i,\ k_1)\}_{i=1}^{n_V}$
2. $V := \mathcal{A}(\{pk_i\}_{i_1}^{n_V}$, "choose controlled voter set")
3. $\{\beta_{i,j}\}_{i \notin V} := \mathcal{A}$("choose votes from uncontrolled voters")
4. $\mathcal{BB} \Leftarrow \{\mathtt{vote}(sk_i, PK_{\mathcal{T}}, \mathcal{BB}, n_C, \beta_{i,0}, k_2)\}_{i \notin V}$
5. $(\mathbf{X}, P) := \mathtt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$
6. $\mathcal{BB} \Leftarrow \mathcal{A}$("cast ballots", $\mathcal{BB}$)
7. $(\mathbf{X}', P') := \mathtt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$

The result of the experiment is 1 if $\mathtt{verify}(PK_{\mathcal{T}}, \mathcal{BB}, n_C, \mathbf{X}', P') = 1$ and ($\{\beta_{i,0} \not\subset \langle \mathbf{X}' \rangle$ **or** $|\langle \mathbf{X}' \rangle| - |\langle \mathbf{X} \rangle| > |V|$). It is 0 else.

A voting protocol is *correct* if $\mathsf{Succ}^{\mathsf{corr}}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$ is negligible in k_1, k_2, k_3 for any adversary $\mathcal{A}$.

5.5.2 Verifiability

We adopt Juels et al.'s notion of verifiability. See Security Game 18 for Experiment $\mathsf{Exp}^{\mathsf{ver}}_{ES,\mathcal{A}}$. A voting protocol is *verifiable* if $\mathsf{Succ}^{\mathsf{ver}}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$ is negligible in k_1, k_2, k_3 for any adversary $\mathcal{A}$. The intuition behind Security Game 18 is:

1. The voters are registered.
2. The adversary $\mathcal{A}$ concocts the full election. $\mathbf{X}$ is a manipulated grand total, P a corresponding proof.
3. The votes on $\mathcal{BB}$ are tallied. $\mathbf{X}'$ is the grand total of an honest tally and P' the corresponding valid proof.

The adversary wins the game if his faked grand total $\mathbf{X}$ differs from the honest grand total $\mathbf{X}'$ and proof P is valid.

Security Game 18 ($\mathsf{Exp}^{\mathsf{ver}}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$)**.**

1. $\{(sk_i, pk_i) \quad := \quad \texttt{register}(SK_{\mathcal{R}},\ i,\ k_1)\}_{i=1}^{n_V}$
2. $(\mathcal{BB}, \mathbf{X}, P) \quad := \quad \mathcal{A}(SK_{\mathcal{T}}, \{(sk_i, pk_i)\}_{i=1}^{n_V}, \text{“forge election”})$
3. $(\mathbf{X}', P') \quad := \quad \texttt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$

The result of the experiment is 1 if $\mathbf{X} \neq \mathbf{X}'$ and $\texttt{verify}(PK_{\mathcal{T}}, \mathcal{BB}, n_C, \mathbf{X}, P) = 1$. It is 0 else.

5.5.3 Deniability of Revoting

Similarly to how Juels, Catalano, and Jakobsson model coercion resistance, we model the deniability of revoting as two experiments following the “real-world-ideal-world” paradigm: Even in a “flawless” voting scheme some information leaks—the total number of cast ballots, the tally, and possibly even who participated in the vote. When measuring the advantage of the adversary, we seek to factor out “unavoidable” leaks of information. To this end we define an *ideal* experiment ($\mathsf{Exp}^{\mathsf{revoting\text{-}c\text{-}resist\text{-}ideal}}_{ES,\mathcal{A},H}$) that captures all such leaks. The *real* experiment ($\mathsf{Exp}^{\mathsf{revoting\text{-}c\text{-}resist}}_{ES,\mathcal{A},H}$) captures the concrete voting protocol. We then examine the difference of the probability of success of an adversary in the ideal world versus an adversary in the real protocol.

In the “real world” experiment, an election is carried out as specified by the protocol. The adversary can corrupt and thus completely control a set of voters. We model this by handing the secret keys of corrupted voters to the adversary. Uncorrupted voters cast their ballots according to the distribution D_{n_U,n_C}. Further, the adversary may select one (uncorrupted) voter as his coercive target. As in the original definition by Juels et al., an extension to a simultaneous coercion of more than one voter is straightforward. The voter does not carry out an evasion strategy in our experiment, i.e. the adversary receives the coerced voter’s secret key and may cast votes in the voter’s stead. (Note that we actually assume the adversary *cannot* learn the coerced voter’s secret key in reality (see Section 5.4.2)). We model time-limited, perfect coercion by giving the adversary access to the secret key, and not accepting further output later.) Then, a bit b is flipped. If $b = 0$, the coerced voter evades coercion by revoting. If $b = 1$, the voter submits to the coercion, i.e. does nothing. After all votes are cast, a tally and corresponding proofs are computed and handed to the adversary. The adversary then guesses b. See Security Game 19 for Experiment $\mathsf{Exp}^{\mathsf{revoting\text{-}c\text{-}resist}}_{ES,\mathcal{A},H}$.

Security Game 19 ($\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1, k_2, k_3, n_V, n_A, n_C)$)**.**

1. V := $\mathcal{A}$(voter names, "control voters")
2. $\{(sk_i, pk_i)$:= $\texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$
3. (j, β) := $\mathcal{A}(\{sk_i\}_{i\in V}$, "set target voter and vote")
4. **if** $|V| \neq n_A$ **or** $j \notin \{1, 2, \ldots, n_v\} - V$ **or** $\beta \notin \{1, 2, \ldots, n_C\} \cup \emptyset$ **then** output 0;
5. $\mathcal{BB}$ $\Leftarrow$ $\texttt{vote}(\{sk_i\}_{i\notin V},\ PK_{\mathcal{T}},\ n_C,\ D_{n_U,n_C},\ k_2)$
6. $\mathcal{BB}$ $\Leftarrow$ $\mathcal{A}(sk_j,\ \mathcal{BB},$ "cast ballots")
7. $b \leftarrow \{0, 1\}$;
8. **if** $b = 0$ **then**
   $\mathcal{BB}$ $\Leftarrow$ $\texttt{vote}(\{sk_j\},\ PK_{\mathcal{T}},\ n_C,\ D_{n_U,n_C}, k_2)$
9. $(\mathbf{X}, P)$:= $\texttt{tally}(SK_{\mathcal{T}},\ \mathcal{BB},\ n_C,\ \{pk_i\}_{i_1}^{n_V}, k_3)$
10. b' := $\mathcal{A}(\mathbf{X},\ P,\ \mathcal{BB},$ "guess b")

The result of the experiment is 1 if $b' = b$ and 0 else.

There is a possibility for the adversary to guess b by the tally alone, depending on the distribution of votes D_{n_U,n_D} and his knowledge thereof. For example, consider an election in a county where all of the voters vote for the "red" party and no one votes for the "blue" party. The adversary could try to coerce a single voter to vote for the "blue" party. If the tally then shows one vote for the "blue" party, his coercion was successful. Otherwise, the target voter obviously re-cast her vote to "red". The possibility of the adversary to detect success of his coercion attempt by the tally is inherent to the principle of publishing the tally one-to-one, and not a property of revoting or any concrete strategy. In fact, this way, a coercer could also infer if a voter has given him a fake credential. In a real-world election we expect the adversary's advantage using such a strategy to be negligible. To characterize the adversarial advantage using "unavoidable" strategies, we define an "ideal" experiment. We then measure the difference of the adversary's advantage in the real and ideal models to express the adversarial advantage against the concrete scheme.

In the ideal experiment, we make use of a function `ideal-tally`, which represents an ideal tallying process. Its working depends on the challenge bit b. If $b = 0$ (voter evades coercion) it counts the coercive target voter's vote. Otherwise (voter submits to coercion) it counts the adversary's choice for the coercive target voter's vote. Further, the ideal adversary is not supplied with the contents of the bulletin board, but only its length l, as well as the election result $\mathbf{X}$. If the coercive target voter evades coercion, the reported length of the bulletin board is increased by the length of one ballot. (Note that the overall number of revotes can always be inferred if the number of cast ballots and the number of counted ballots is visible.) See Security Game 20 for Experiment $\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$.

Security Game 20 ($\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}(k_1, k_2, k_3, n_V, n_A, n_C)$)**.**

1. $V := \mathcal{A}'(\text{voter names, "control voters"})$
2. $\{(sk_i, pk_i) := \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$
3. $(j, \beta) := \mathcal{A}'(\text{"set target voter and vote"})$
4. **if** $|V| \neq n_A$ **or** $j \notin \{1, 2, \ldots, n_v\} - V$ **or** $\beta \notin \{1, 2, \ldots, n_C\} \cup \emptyset$ **then** output 0
5. $b \leftarrow \{0, 1\}$
6. $\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_i\}_{i \notin V}, PK_{\mathcal{T}}, n_C, D_{n_U, n_C}, k_2)$
7. $\mathcal{BB} \Leftarrow \mathcal{A}'(sk_j, |\mathcal{BB}|, \text{"cast ballots"})$
8. $l \Leftarrow |\mathcal{BB}|$
9. **if** $b = 0$ **then** $l \Leftarrow |l| + 1$
10. $(\mathbf{X}, P) := \texttt{ideal-tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3, b)$
11. $b' := \mathcal{A}'(\mathbf{X}, l, \text{"guess b"})$

The result of the experiment is 1 if $b' = b$ and 0 else.

We define the *advantage* of the adversary $\mathcal{A}$ as

$$\begin{aligned}\mathbf{Adv}^{revoting}_{\mathcal{A}}(k_1, k_2, k_3, n_V, n_A, n_C) := \; & \mathsf{Succ}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1, k_2, k_3, n_V, n_A, n_C) \\ & - \mathsf{Succ}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(k_1, k_2, k_3, n_V, n_A, n_C).\end{aligned}$$

A voting scheme features *deniable revoting* if $\mathbf{Adv}^{\text{revoting}}_{\mathcal{A}}$ is negligible in k_1, k_2, k_3 for any adversary.

5.6 A Coercion-Resistant Voting Protocol with Deniable Revoting

In this section, we first introduce several building blocks used in our protocol. We then describe our construction and prove its security according to the definitions introduced in the previous section.

5.6.1 Black-Box Ideal Functionalities

We use several primitives as blackboxes and assume the existence of an efficient realisation. Though we suggest efficient instantiations where possible, we focus on our general approach for achieving deniable revoting. We see this work as a proof of concept.

Verifiable Secret Shuffle

A verifiable shuffle computes a function $\mathsf{Shuffle}(C) \mapsto (C', P)$, which receives a list $C := [c_1, \ldots, c_n]$ of randomisable ciphertexts as input. Its output consists of a list $C' := [c'_{\pi(1)}, \ldots, c'_{\pi(n)}]$, where π is a random permutation and $c'_{\pi(i)}$ is a re-encryption of c_i for $i = 1, \ldots, n$, and a proof P of correctness of the shuffle.

We assume our shuffles are *secret* and *verifiable*: secrecy implies it is infeasible to link an input ciphertext c_i to its corresponding output ciphertext $c'_{\pi(i)}$ better than by guessing, and verifiability requires a proof P that C is indeed a permutation (with re-encryption) of C'.

We further assume our shuffles are *distributable* among several servers, and speak of a *mix-net* if a shuffle is distributed (see also Section 2.4.5). Various verifiable secret shuffles and mix-nets have been proposed [11, 94, 102, 134, 166, 176].

EUF-CMA Secure Digital Signatures

In our voting scheme, voters use unforgeable signatures to prove their eligibility (see Existential Unforgeability under Chosen-Message Attacks (EUF-CMA), Section 2.4.2). We require our signatures to allow for an efficient zero-knowledge proof of signature knowledge.

Non-Interactive Zero-Knowledge Proofs (of Knowledge)

We make use of Non-Interactive Zero-Knowledge (NIZK) proofs and Non-Interactive Zero-Knowledge Proofs of Knowledge (NIZKPoKs) in our construction (Section 2.4.4). NIZK proofs and NIZKPoK exist for arbitrary $\mathcal{NP}$ statements. Correct decryption can be proven by using the Chaum-Pedersen protocol [56] as a proof of logarithm equivalence. Logarithm knowledge can be proven with Schnorr's protocol [177]. A proof of signature knowledge was introduced by Camenisch et al. [39].

5.6.2 Building Blocks

Modified Elgamal Encryption (m-Elgamal)

M-Elgamal is a modification of the Elgamal encryption scheme (Section 2.4.1), used by Juels et al. [122]. Given a multiplicative cyclic group G of prime order p in which the decisional Diffie-Hellman problem is hard, and generators $g_1, g_2 \in G$, for key generation choose random $x_1, x_2 \in \mathbb{Z}_p$, and output secret key (x_1, x_2) and public key $y := g_1^{x_1} g_2^{x_2}$. To encrypt a message m with public key y, choose $r \in \mathbb{Z}_p$ at random, and output the ciphertext $c = \mathsf{Enc}_{\mathsf{hom}}(m, y) := (g_1^r, g_2^r, y^r m)$. To decrypt a ciphertext $c = (A, B, C)$ with secret key (x_1, x_2), compute $m = \mathsf{Dec}_{\mathsf{hom}}(c, (x_1, x_2)) := A^{-x_1} B^{-x_2} C$. To simplify notation, we write $\mathsf{Enc}_{\mathsf{hom}}(m)$ or $\mathsf{Dec}_{\mathsf{hom}}(c)$ if the keys are clear from the context. Decryption can be distributed using Cramer et al.'s construction [61]. Ciphertexts created with m-Elgamal are multiplicatively homomorphic and randomisable by multiplying with an encryption of 1:

$$\begin{aligned}\mathsf{Enc}_{\mathsf{hom}}(m_1) \cdot \mathsf{Enc}_{\mathsf{hom}}(m_2) &= (g_1^r, g_2^r, y^r m_1) \cdot (g_1^s, g_2^s, y^s m_2) \\ &= (g_1^{r+s}, g_2^{r+s}, y^{r+s} m_1 \cdot m_2) \\ &= \mathsf{Enc}_{\mathsf{hom}}(m_1 \cdot m_2).\end{aligned}$$

A proof of encryption of a certain plaintext can be achieved by publishing the encryption randomness followed by randomisation. Correct decryption can be proven with the Chaum-Pedersen-protocol [56].

Plaintext Equality Test

Plaintext Equivalence Tests were introduced by Juels et al. [120, 122]. They decide whether two ciphertexts contain the same plaintext. We denote this by a function $\mathsf{Pet}(c_1, c_2, t) \mapsto (b, \Pi)$ which receives as input two ciphertexts c_1 and c_2 and a decryption trapdoor t. Its output is a bit b with $b = 1$ if c_1 and c_2 contain the same plaintext, and $b = 0$ otherwise, as well as a proof Π of correctness. In our protocol, we perform computations on encrypted Plaintext Equivalence Test (PET) results. We define *Encrypted Plaintext Equivalence Tests (EPETs)* denoted by a function $\mathsf{Epet}(c_1, c_2) \mapsto (c, \Pi_c)$ which outputs an encryption of 1 if c_1 and c_2 contain the same plaintext, and an encryption of a random number, if the plaintexts are different, as well as a proof Π_c of correctness. EPETs can be computed without a trapdoor. We require PETs to be distributable.

(Encrypted) Plaintext Equality Test for Homomorphic Encryption Juels et al. [120] introduce a plaintext equality test for Elgamal encryption. We generalise their result to a PET for multiplicatively homomorphic encryption which can also be used as an EPET. Instantiating it with Elgamal or m-Elgamal allows for a distributed execution of the PET.

Let $\mathsf{Enc}_{\mathsf{hom}}$ denote a multiplicatively homomorphic encryption function, i.e. $\mathsf{Enc}_{\mathsf{hom}}(m_1) \cdot \mathsf{Enc}_{\mathsf{hom}}(m_2) = \mathsf{Enc}_{\mathsf{hom}}(m_1 \cdot m_2)$, and $\mathsf{Dec}_{\mathsf{hom}}$ its decryption function. We perform a PET on two ciphertexts $c_1 = \mathsf{Enc}_{\mathsf{hom}}(m_1)$ and $c_2 = \mathsf{Enc}_{\mathsf{hom}}(m_2)$ for plaintexts m_1 and m_2 as follows:

Algorithm $\mathsf{Epet}(c_1, c_2)$: On input of two ciphertexts c_1 and c_2, choose r at random and compute $c_{diff} := (c_1/c_2)^r$ with a NIZKPoK Π_c of an r such that $c_{diff} = (c_1/c_2)^r$, for example by using the Chaum-Pedersen-protocol [56]. Exploiting the homomorphic property of the encryption, we have $c_{diff} = (c_1/c_2)^r = \mathsf{Enc}_{\mathsf{hom}}((m_1/m_2)^r) = 1^r = 1$ if $m_1 = m_2$ (or $r \equiv 0 \mod \mathrm{Ord}(G)$, see below). By outputting c_{diff} and the proof Π_c, this scheme can be used as an EPET.

Algorithm $\mathsf{Pet}(c_1, c_2, t)$: To make a PET out of the EPET, first compute $(c_{diff}, \Pi_c) := \mathsf{Epet}(c_1, c_2)$, then decrypt c_{diff} using decryption key t, with a proof Π_d of correct decryption. Output 1 if $\mathsf{Dec}_{\mathsf{hom}}(c_{diff}) = 1$, and 0 otherwise, as well as the proof $\Pi := (\Pi_c, \Pi_d)$. If $m_1 \neq m_2$, c_{diff} is a random number because of the mask r, and can be revealed in order to prove correctness of the result. Also, if it is not clear from the form of c_{diff} that $r \neq 0 \mod \mathrm{Ord}(G)$, r can be opened if $\mathsf{Dec}_{\mathsf{hom}}(c_{diff}) = 1$.

5.6.3 Our Construction

We divide elections into five phases: setup, registration, setup publication, voting, and tallying.

1. Setup. The tellers create a joint public encryption key $PK_{\mathcal{T}}$ and a shared secret decryption key sk_T for threshold-decrypting the homomorphic encryption m-Elgamal introduced in Section 5.6.2. Let $\mathsf{Enc}_{\mathsf{hom}}$ denote its encryption function.

2. Registration. Each voter V_j obtains their credential for voting, i.e. a key pair (pk_j, sk_j) for an EUF-CMA secure signature scheme (Section 2.4.2). We assume an efficient NIZKPoK for this signature scheme. The public verification key obtained here also acts as identification of the voter. For each voter, an entry $(ID, \mathsf{Enc}_{\mathsf{hom}}(pk_j))$ is stored in a List L_0, where ID encodes the voter's name. The encryption $\mathsf{Enc}_{\mathsf{hom}}(pk_j)$ is computed using the teller's public key $PK_{\mathcal{T}}$.

3. Setup publication. A candidate slate $\mathcal{C} = (c_1, \ldots, c_l)$ is published on the public bulletin board. It contains the electable candidates $c_1, \ldots, c_l$. The list L_0 is published and serves as a list of eligible voters and their encrypted public verification keys. The public key $PK_{\mathcal{T}}$ of the tellers is published as well.

4. Voting. Voters create ballots and submit them to the bulletin board. A ballot b_i is a three-tuple as explained below. Encrypted parts are encryptions under the public key of the tellers:

$$b_i = (\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}}pk_i, \mathrm{ts}_i),\ \mathit{NIZKPoK}$$

   Here, $\mathsf{Enc}_{\mathsf{hom}}(\beta_i)$ is an encryption of the choice $\beta_i \in \mathcal{C}$. We assume this for the simplicity of our description. To support arbitrary choices like STVs, we can alternatively let the choice be a function of the candidate slate and the voter's intention. Furthermore, $\mathsf{Enc}_{\mathsf{hom}}(pk_i)$ is an encryption of the voter's public key pk_i, and ts_i is a timestamp created right before the ballot is cast. The timestamp is not encrypted. In addition to the ballot itself, the voter submits a Non-Interactive Zero-Knowledge Proof of Knowledge *NIZKPoK* to prove that the ballot has been cast with her consent. The NIZKPoK proves knowledge of a signature of $(\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}}pk_i, \mathrm{ts}_i)$ w.r.t. the public key encrypted in the ballot. (Directly attaching the signature itself would reveal how many votes a single voter has cast.) Stated more formally, the voter proves knowledge of a signature σ_i with

$$\mathsf{Verify}(\sigma_i, (\mathsf{Enc}_{\mathsf{hom}}(\beta_i), \mathsf{Enc}_{\mathsf{hom}}(pk_i), \mathrm{ts}_i), pk_i') = 1 \ \wedge \ pk_i' = pk_i.$$

5. Tallying. Before the ballots can be tallied, all superseded ballots have to be sorted out. This procedure is described in detail in the next section. After all ballots with invalid proofs of signature knowledge, all superseded ballots, and all ballots with invalid credentials have been omitted, the remaining ballots can be tallied with standard techniques, for example by a decryption mix.

5.6.4 Sorting Out Superseded Ballots

In this section, we describe how to mark all superseded ballots. A ballot is superseded when a more recent ballot of the same voter—that is, a ballot with a newer timestamp

and matching voter credential—is posted to the bulletin board. Our method protects the privacy of the voter and is also publicly verifiable.

In four steps, the tallying authority computes a shuffled list of all valid ballots without their timestamps. They will be re-encrypted and in a random order. The resulting list will only contain the last votes cast by eligible voters. Without loss of generality we assume all submitted choices are valid.

1. **Encrypted Tests Whether Two Ballots Are From The Same Voter** The procedure starts off with a list of all "well-formed" ballots on the bulletin board, i.e. all ballots with a valid NIZKPoK as described above. After it has been checked, the NIZKPoK is omitted in further steps. The list is sorted according to the timestamp in ascending order. For each ballot b_i, the tallying authority tests if the encrypted public key pk_i of b_i matches the public key pk_j of any newer ballot b_j: for each ballot b_j $(i < j < n)$, the tallying authority performs distributed EPETs to obtain $(d_{i,j}, \Pi_{i,j}) := \mathsf{Epet}(\mathsf{Enc}_{\mathsf{hom}}(pk_i), \mathsf{Enc}_{\mathsf{hom}}(pk_j))$. The encrypted differences $d_{i,j}$ and the proofs $\Pi_{i,j}$ of correctness of the result of the EPET are published. ($d_{i,j} = \mathsf{Enc}_{\mathsf{hom}}(1)$ iff $pk_i = pk_j$.)

2. **Marking Ballots As Superseded** The tallying authority performs a verifiable conversion on all computed differences. If $d_{i,j}$ is an encryption of 1, replace it by an encryption of a random number. Else, replace it with an encryption of 1. Differences are detached from their corresponding ballots before conversion, and later sorted back. The details of this step are described in Section 5.6.4. For each ballot, the tallying authority aggregates all converted $d'_{i,j}$ by multiplying them, exploiting the homomorphic property of the encryption: $o_i := \prod_j d'_{i,j}$

3. **Omit Superseded Ballots** The tallying authority jointly compares o_i with $\mathsf{Enc}_{\mathsf{hom}}(1)$ for each ballot b_i. It omits all ballots b_i with $\mathsf{Epet}(o_i, \mathsf{Enc}_{\mathsf{hom}}(1)) \neq 1$ from future computations. (Those are the ballots that have been superseded by a more recent one of the same voter.) The last ballot is never superseded.

4. **Omit Ballots With Invalid Credentials** Before tallying, the tallying authority checks the voter credentials by verifying that each $\mathsf{Enc}_{\mathsf{hom}}(pk_i)$ has a corresponding entry in the published list of the encrypted public keys of eligible voters. Similarly to the technique of Juels et al. [122], the tallying authority shuffles the encrypted public keys of the list L_0, and performs a PET with the encrypted public key of each ballot.

We now describe the four steps in detail.

Encrypted Tests Whether Two Ballots Are From The Same Voter

In the first step of the process, for each pair of ballots (b_i, b_j) with $j > i$, the tallying authority performs EPETs (see Section 5.6.2) to the encrypted public keys of the voter. While, technically, this step can be performed during the tally phase, we propose to directly perform it during the casting phase:

When a ballot $b_i = ((\mathsf{Enc}_{\mathsf{hom}}(\beta_i), \mathsf{Enc}_{\mathsf{hom}}(pk_i), \mathrm{ts}_i)$ is cast, the tallying authority checks the corresponding NIZKPoK and discards the ballot if the proof is invalid, i.e. the ballot is marked as invalid and not considered in further computations, but remains on the bulletin board for public verification. Otherwise, the tallying authority jointly runs an EPET on the encrypted public key of b_i and those of all already cast ballots b_j, to obtain values $(d_{i,j}, \Pi_{i,j}) = \mathsf{Epet}(\mathsf{Enc}_{\mathsf{hom}}(pk_i), \mathsf{Enc}_{\mathsf{hom}}(pk_j))$. All encrypted differences $d_{i,j}$ are stored alongside b_j as indicated in Figure 5.2, defining a list L with entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$.

After the casting phase, if n well-formed ballots have been cast, b_i has $n - i$ associated differences.

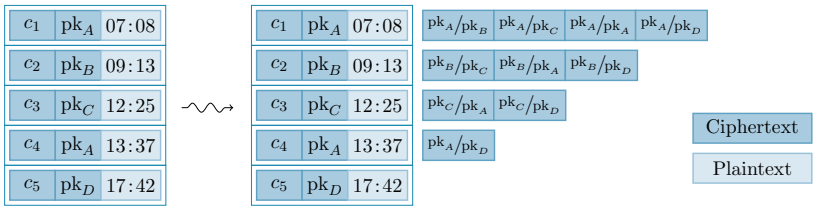

Figure 5.2: Encrypted Plaintext Equality Tests (EPETs) are performed for each pair of ballots (b_i, b_j) with $j > i$ by the tallying authority: If $pk_i = pk_j$ the result is an encrypted 1, otherwise it is an encrypted random value. We denote an encrypted value by a box with a dark background and an unencrypted value by a box with a lighter background. In this example, the third fraction in the first row $(\mathrm{pk}_A/\mathrm{pk}_A)^r$ divides identical credentials, hence ballot 4 supersedes ballot 1.

Marking Ballots as Superseded

Before computing the supersede mark o_i for each ballot, the differences $d_{i,j}$ computed during the voting phase are converted as indicated by the mapping

$$\mathsf{Enc}_{\mathsf{hom}}(x) \mapsto \begin{cases} \mathsf{Enc}_{\mathsf{hom}}(r) & \text{if } x = 1 \\ \mathsf{Enc}_{\mathsf{hom}}(1) & \text{if } x \neq 1 \end{cases}$$

The value r can be any fixed value other than 1 (2, for example) or a number drawn at random.

To compute the mapping, the tallying authority creates a shuffled and randomised list of all encrypted differences. To this end, each entry $d_{i,j}$ is associated an encrypted tag $\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i)$ (see Figure 5.3).

The list of all tuples $(\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i), d_{i,j})$ is then re-randomised and reordered, using a verifiable secret shuffle. After shuffling, the differences are converted (see Figure 5.4). To convert $d_{i,j}$ to $d'_{i,j}$ we perform a multi-party computation that involves a "coordinator" and the voting authority. In this step, the authority's task is to decrypt each $d_{i,j}$ it receives from the coordinator and return either an encryption of a 1 or of another (random

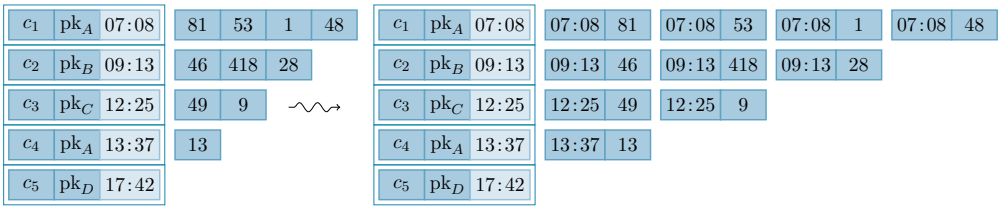

Figure 5.3: The fractions from the previous step are complemented by the encrypted timestamp from the ballot, forming tuples $(\mathsf{Enc}_{\mathsf{hom}}(ts_i), d_{i,j})$. For the sake of clarity, we evaluated the fractions from Figure 5.2 to exemplary values. The superseded first ballot thus has a "1" associated with it.

or fixed) number, according to the map above. The task of the coordinator is to send fake differences or real ones to the authority. He has to make sure that the authority does not learn which conversion is real. Therefore, each element is randomised before and after each conversion by the coordinator. The randomised version of the output of each "real" conversion is fed to the authority to be converted a second time and get a fake difference. The second conversion is necessary to hide how many of the $d_{i,j}$ contain a 1. All (real and fake) differences are sent to the authority in random order. The coordinator and the authority must not collude.

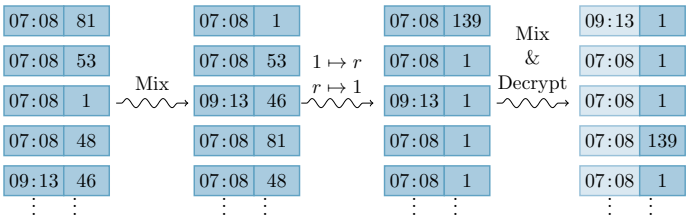

Figure 5.4: Converting $d_{i,j}$ to $d'_{i,j}$: if $\mathsf{Dec}_{\mathsf{hom}}(d_{i,j}) = 1$ replace $d_{i,j}$ by $\mathsf{Enc}_{\mathsf{hom}}$ (r) $(r \neq 1)$ and by $\mathsf{Enc}_{\mathsf{hom}}$ (1) else. The special "superseded" value 1 has been converted to an arbitrary number, while all other values are mapped to 1.

Irrespective of how the entries $d_{i,j}$ are converted to $d'_{i,j}$, their correctness can easily be proven: either $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d_{i,j}) = 1$ or $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d'_{i,j}) = 1$, never both. To prove the correctness of the inversion without revealing the values, we use a verifiable shuffle on $(d_{i,j},\ d'_{i,j})$ to obtain (a, b). We then check whether $\mathsf{Pet}(a \cdot b,\ a) = 1$ or $\mathsf{Pet}(a \cdot b,\ b) = 1$ (exclusively). See Figure 5.5 for a compact description of the procedure. The procedure starts with the list of all ballots and a corresponding list of differences next to each ballot and attaches an encrypted mark to each ballot. The encrypted mark tells whether the ballot has been superseded. We make use of subroutines like a secret verifiable shuffle (see Section 5.6.1). Since each of those subroutines is verifiable, it produces a proof of correctness beside the actual result. Each proof contributes to the over-all proof of correctness. In abuse of notation and for better readability, we do not explicitly mention them.

After a second shuffling step, the tags are decrypted, and the encrypted, converted

Input A list L with (ballot,differences)-entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$.
Output A converted list L' of ballots and differences, and a list P of proofs of correctness.
Algorithm (Each operation which can not be recomputed by everyone outputs a proof of correctness. For the sake of readability, we do not explicitly state the output of the proofs, but implicitly assume them to be appended to the list P.)

1. Initialize list L' with $L[i] := (b_i, [])$ (the second entry will be filled later).
   Initialize a look up table TS with an entry (i, ts_i) for each ballot $b_i = (\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}}pk_i, \mathrm{ts}_i)$
2. For each ballot b_i prepare a tag $\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i)$.
3. $\forall 1 \leq i \leq n,\ \min(i+1, n) \leq j \leq n$: Save $(\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i), d_{i,j}, [], [])$ to a list D. The empty entries will be filled later
4. Shuffle D: $D' := \mathsf{Shuffle}(D)$.
5. For each $d_{i,j}$ in D'
   convert $d_{i,j}$ to $d'_{i,j} := \begin{cases} \mathsf{Enc}_{\mathsf{hom}}(r) & \text{if } d_{i,j} = \mathsf{Enc}_{\mathsf{hom}}(1) \\ \mathsf{Enc}_{\mathsf{hom}}(1) & \text{else} \end{cases}$ where $r \neq 1$
   and store $d'_{i,j}$ in D' next to $d_{i,j}$: $(\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, [])$.
6. create the proof Π of correct conversion and store it in D': $(\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, \Pi)$. The correctness of the conversion can be verified based on D'.
7. Shuffle D': $D'' := \mathsf{Shuffle}(D')$
8. For each $(\mathsf{Enc}_{\mathsf{hom}}(\mathrm{ts}_i), d'_{i,j}) \in D''$:
   - Decrypt tag $\mathsf{Enc}_{\mathsf{hom}}$ (ts_i) to ts_i with proof of correct decryption
   - look up entry (i, ts_i) in TS
   - append $d'_{i,j}$ to the second entry of $L'[i]$.
9. For each ballot b_i: multiply each element $d'_{i,j}$ in the second entry of L'[i] and replace them by the result
10. Output (L', P).

Figure 5.5: Conversion $\mathsf{Convert}(L)$, marking each ballot whether it has been superseded.

differences are sorted back to their ballot of origin (see step one in Figure 5.6). All steps can be verified by the public. The tallying authority then computes the homomorphic product over all the associated marks of each ballot (see step two in Figure 5.6): for each i compute

$$o_i := \prod_{j=\min(i+1,n)}^{n} d'_{i,j}.$$

Observe that if $\prod d'_{i,j}$ contains only encryptions of 1, it is itself an encryption of a 1, whereas if it has a factor $\neq 1$, it is itself an encryption of a number $\neq 1$ with overwhelming probability (see Figure 5.1).

Omit Superseded Ballots

Before any further processing, particularly before checking if $o_i = Enc(1)$, all ballots are weeded and shuffled. Only the encrypted choice of the voter, his encrypted public key, and the mark o_i are kept. Therefore, the tallying authority forms a list L_W with entries $L_W[i] := (b'_i, o_i)$, where $b'_i := (\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}}pk_i)$ if $b_i = (\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}}pk_i, \mathrm{ts}_i)$.

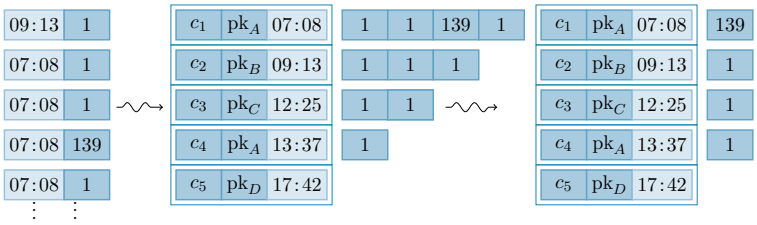

Figure 5.6: The encrypted, converted differences are sorted back to their ballot of origin. Afterwards o_i is computed as the homomorphic product over all the associated marks of ballot b_i. Note that we arranged the $d'_{i,j}$ values in the same order as their preimages. In a practical realisation of our scheme this would only coincidentally be the case. Ballots now have an encrypted "tag" that tells whether they are superseded by a more recent ballot.

The tallying authority computes and publishes $(L'_W, \Pi) := \mathsf{Shuffle}(L_W)$ and then jointly compares o_i with $\mathsf{Enc}_{\mathsf{hom}}(1)$ in L'_W using a PET, and publishes all proofs. Only ballots b_i with $o_i = \mathsf{Enc}_{\mathsf{hom}}(1)$ and the last ballot are kept, the others are marked as discarded.

Omit Ballots With Invalid Credentials

Finally, the validity of the public keys is checked. Note that, at this point, only one ballot per credential remains. Equally to the method of Juels et al., the tallying authority shuffles the encrypted public keys of L_0 and performs a joint PET with the encrypted public key of each ballot. Ballots with valid public keys are retained. The encrypted choices of the retained ballots can then be mixed and opened for tallying.

5.6.5 Proving the Security of our Scheme

In this section, we prove the correctness, verifiability, and coercion resistance of our protocol. We will show coercion resistance by proving that revoting in our protocol is deniable.

Correctness

Theorem 25. *The scheme proposed in Section 5.6 is correct (Section 5.5.1).*

Proof. Recall that a voting scheme is correct if the success probability of any adversary in Experiment $\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$ (Figure 17) is negligible in $\max(k_1, k_2, k_3)$. We show that our construction is correct by an induction over the number of ballots n on the bulletin board in Step 8 of experiment $\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}$. The number $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ is comprised of $n_{\mathcal{V}}$, the number of ballots posted by honest voters on the bulletin board ($\mathcal{BB}$) in Step 4, and $n_{\mathcal{A}}$, the number of ballots posted on $\mathcal{BB}$ by the adversary in Step 6. Per assumption, the bulletin board is append-only.

Fix $n = 1$ as the base case, i.e. either $n_{\mathcal{A}} = 0$ or $n_{\mathcal{V}} = 0$. If $n_{\mathcal{A}} = 0$, then $\mathbf{X}' = \mathbf{X}$ and $P' = P$, and the experiment outputs 0. On the other hand, if $n_{\mathcal{V}} = 0$, then we have

to show that the adversary can only cast a vote for a corrupted voter. (The adversary cannot overwrite an uncorrupted voter's vote, because there are none.) Ballots containing an invalid voter public key, i.e. a public key $pk \notin L_0$ are discarded in the tallying phase (see Section 5.6.4, Step 5). The probability that the adversary generates a public key pair (sk, pk) such that sk generates a valid signature for the ballot and $pk \in L_0$ is negligible in the security parameter of the signature scheme. Thus, the experiment outputs 0 except for a negligible probability. (We stress that "malformed" ballots without a correct proof of knowledge of a signature on the ballot are discarded before the tallying phase of our protocol.)

Now assume there are $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ ballots on the bulletin board in Step 8 of $\mathsf{Exp}^{\mathsf{corr}}_{ES,\mathcal{A}}$ and the probability that the experiments outputs 1 is negligible in the security parameters for any adversary. To transfer to the $n+1$-case, we distinguish two cases to post an additional ballot on $\mathcal{BB}$.

- The additional ballot is cast in Step 4, i.e. by an uncontrolled voter. We have to show that it supersedes the most recent ballot by the same voter. This is achieved by a pairwise comparison of the public keys on the ballots (see Section 5.6.4).
- The additional ballot is cast in Step 6, i.e. by a controlled voter. We have to show that it does not supersede a ballot by a different voter and itself is not already superseded. By the same argument as above, only ballots that contain identical public keys are superseded during the tallying phase. (For the adversary to post a ballot containing the public key of an uncontrolled voter, he would have to forge the signature of the ballot and thus break the EUF-CMA security of the signature scheme.) Because the newest ballot has the newest timestamp per assumption, it is not superseded in this step.

This concludes the argument. □

Verifiability

We argue the verifiability of our voting scheme informally. A process composed of several subroutines is verifiable if each step is verifiable by itself. Our process is composed of a small set of verifiable black-box functionalities and other verifiable building blocks.

Each of these subroutines produces proofs of correctness along with its result. If either of these proofs is invalid, $\mathsf{Exp}^{\mathsf{ver}}_{ES,\mathcal{A}}$ outputs 0. All these proofs are published next to the result of the subroutine on the bulletin board. The whole bulletin board with the input $\{b_i\}_{i=1}^{n}$, the final output $\mathbf{X}$, all subroutine proofs, and all interim results is the global proof P of correctness of the tally. We point out that each interim result is published except for the secret keys and some randomness which is necessary to protect the privacy of the voter. The input of each subroutine is also marked down on the bulletin board. Sometimes the output from one routine has to be reshaped to match the input format from another one. This reshaping is always deterministic.

Several operations like the reshaping or the computation of the product over all marks $d_{i,j}$ of each ballot b_i can be recomputed by anyone. Therefore no explicit proof is necessary

for these steps. All operations which are not recomputable by the public are accompanied by proofs of correctness.

In conclusion, our tallying process is verifiable because intermediate values are public, and all subroutines are publicly verifiable.

Deniability of Re-Cast Votes

Theorem 26. *The scheme proposed in Section 5.6 offers deniable revoting (Section 5.5.3).*

To prove the deniability of revoting, we give a reduction of the privacy of our scheme to the Decisional Diffie-Hellman (DDH) problem (Section 2.4.1). More concretely, given any (successful) adversary $\mathcal{A}$ against $\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}$, we construct a reduction $\mathcal{S}$ which is, with non-negligible probability, able to decide for a given quadruple (g, g^a, g^b, g^c) whether $c = ab$ in the cyclic group $G = \langle g \rangle$.

As in the definition, we consider only one coerced voter. An extension to the case of multiple coerced voters is straightforward—we just have accordingly many parallel instances of the DDH problem. Our proof is similar to that of Juels et al. [121]. We give a reduction from any adversary that uses the published ballots to break the incoercibility of the scheme to an adversary against the DDH assumption. To this end, we simulate the execution of experiment $\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}$.

Juels et al. call their reduction a "simulator", because it simulates the execution of the security game to the adversary. This choice of terms is not wrong per-se, however it lends itself to confusion. We use the word "reduction" here to avoid confusion with the simulator concept in the Universal Composability (UC) [41] framework we use in Chapter 3. The proof methodology we use here is more akin to the security-game-reduction methodology used in Chapter 4 (Section 2.3.1). In our original publication [2] we use the term "simulator" as well.

Proof. The DDH experiment $\mathbf{Exp}^{\mathsf{ddh}}_G$ secretly draws a bit d, if $d = 1$ sets $g^c = g^{ab}$, and to a random element $g^c \in G$ otherwise. Given a DDH challenge (g, g^a, g^b, g^c), $\mathcal{S}$ deducts two public keys. In the setup publication phase, $\mathcal{S}$ sends the first one to the adversary, while it uses the second one during the rest of the protocol. $\mathcal{S}$ deducts the following two m-Elgamal public keys:

$$(g_1 := g,\ g_2 := g^a,\ y_g := g_1^{x_1} g_2^{x_2} = g^{x_1} g^{a x_2}) \text{ and}$$

$$(h_1 := g^b,\ h_2 := g^c,\ y_h := h_1^{x_1} h_2^{x_2} = g^{b x_1} g^{c x_2}).$$

Recall that, to encrypt m using the Modified Elgamal Encryption (Section 5.6.2) using public key (g_1, g_2, y), one chooses a random r and outputs $(g_1^r, g_2^r, y^r m)$. When $c = ab$ ($d = 1$ in the surrounding experiment), for any m there are r, r' such that $(g_1^{r'}, g_2^{r'}, y_g^{r'} m) = (h_1^r, h_2^r, y_h^r m)$. ($r' = br$, concretely.) Therefore, ciphertexts created using the above public keys have the same distribution.

When, on the other hand, $c \neq ab$ ($d = 0$), m is perfectly hidden in the encryption. A message m, encrypted with the second public key and decrypted with the first (given to $\mathcal{A}$), yields:

$$(h_1^r = g^{br} = g_1^{br}, h_2^r = g^{cr} = g_2^{(c/a)r},$$

$$y_h^r m = g^{rbx_1} g^{rcx_2} m = g^{rbx_1} g^{abrx_2} g^{(c-ab)rx_2} m = y_g^{br} g^{(c-ab)rx_2} m)$$

In $\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}$ the choices of the voters as well as their public keys are perfectly hidden in the m-Elgamal Encryption when $d = 0$. In this case the adversary's capabilities are reduced to those in experiment $\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$. To justify this assertion, we must show how the adversary's input can be faked in the $d = 0$ case with only the information from $\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ available. It is then obvious that the adversary cannot gain any additional advantage from the simulated tallying process.

In the ideal experiment $\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ the adversary learns the total number of cast ballots and the number of valid ballots. The individual m-Elgamal encryptions the reduction outputs to the adversary hide the plaintext perfectly when $d = 0$. In this case, they are not distinguishable from any random group element and thus can be faked by random output. We must also be able to fake the quantity and structure of the data the adversary learns during the simulation. These can be determined from the difference of cast and valid ballots, and vice versa.

We state the simulation of $\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}$ and argue that its perfect. The reduction $\mathcal{S}$ receives a $W \in D_{n_U,n_C}$ and a challenge (g, g^a, g^b, g^c) with $c = ab$ if the random bit $d = 1$ or $c \neq ab$ if $d = 0$.

Setup The reduction $\mathcal{S}$ chooses the secret key $SK_{\mathcal{T}} := (x_1, x_2)$, uniformly and at random. $\mathcal{S}$ also calculates the public key $PK_{\mathcal{T}} := (g^{x_1}, g^{x_2}, h = (g_1^{x_1} g_2^{x_2}) \mod p$ and sets $C = \{c_i\}_{i=1}^{n_C}$.

Registration $\mathcal{S}$ simulates the registrar $\mathcal{R}$: $\mathcal{S}$ generates the voter credentials: $\{(pk_i, sk_i)\}_{i=1}^{n_V}$

Setup publication $\mathcal{S}$ publishes the public key of the tally $PK_{\mathcal{T}}$, the set of candidates C, and $L_0 = \{$"voter's name i", $\mathsf{Enc}_{\mathsf{hom}}(pk_i)\}_{i=1}^{n_V}$.

Adversarial corruption The adversary $\mathcal{A}$ selects n_A voters to corrupt. Let V denote the group of corrupted voters. He also selects a voter j for coercion. He determines the choices β for the corrupted and the coerced voters. The simulation is terminated if the selection is invalid.

Coin flip The bit b is chosen uniformly at random: $b \leftarrow \{0,\ 1\}$

Honest voter simulation For each honest voter, the reduction creates the ballots with the proofs:

$$A_0 := \{(\mathsf{Enc}_{\mathsf{hom}}\beta_i, \mathsf{Enc}_{\mathsf{hom}} pk_i, \mathrm{ts}_i),\ NIZKPoK\}$$
$$= \{b_i := ((h_1^{r_i}, h_2^{r_i}, h_1^{r_i x_1} h_2^{r_i x_2} c_j), (h_1^{k_i}, h_2^{k_i}, h_1^{k_i x_1} h_2^{k_i x_2} pk_i), ts), NIZKPoK\}_{i=1}^{n}$$

Thereby all r_i and k_i are chosen at random in Z_q. The choices are determined in W. For creating the proofs the reduction $\mathcal{S}$ uses the voter's secret keys.

Adversarial ballot posting The adversary calculates the ballots for each voter $v \in V$ and the voter j in the same way. We call the set of these ballots B_0. $\mathcal{A}$ posts B_0.

Tallying simulation $\mathcal{S}$ simulates a honest tallying authority, whereby it uses the secret key $SK_{\mathcal{T}}$ from the setup step. Since the correctness of every step from each authority can be verified, any modification from an adversary can be ignored.

Proof checking The proof of each ballot in A_0 and B_0 is checked. Let E_1 be all ballots with valid proofs.

Creating $d_{i,j}$ The reduction $\mathcal{S}$ creates the $d_{i,j}$ by performing the necessary EPETs. $\mathcal{S}$ also creates the proofs honestly.

Converting $d_{i,j}$ $\mathcal{S}$ performs the protocol honestly and provides the proofs. During this process, it uses the secret key $SK_{\mathcal{T}}$ to decrypt the EPETs.

Creating o_i $\mathcal{S}$ accumulates the $d_{i,j}$ as scheduled.

Creating final ballot list For each ballot in E_1 create a cut list as described in the protocol. $\mathcal{S}$ creates a shuffle and the according proofs. Denote the result as E_2. $\mathcal{S}$ creates list of tuples, using the secret key: $E_3 := \{(\mathsf{Enc}_{\mathsf{hom}}(\beta_i), \mathsf{Enc}_{\mathsf{hom}}(pk_i))\} : \mathsf{Pet}(o_i, \mathsf{Enc}_{\mathsf{hom}}(1)) = 1$

Checking Voter Credentials $\mathcal{S}$ shuffles the second components of L_0 into a new list L_1: Let L_0' be the list of all encrypted public keys in L_0. $L_1 := \mathsf{Shuffle}(L_0')$. For each ballot in E_3, $\mathcal{S}$ performs PETs with the second part of each entry. $\mathcal{S}$ uses the secret key for decryption. Let E_4 denote the list of all encrypted choices from the ballots with a according item in L_1.

Choice decryption The decryption of the valid choices (E_4) is done by $\mathcal{S}$ with the secret key.

Adversarial guess The adversary outputs a guess bit b'. $\mathcal{S}$ itself returns $d' := (b' \stackrel{?}{=} b)$ as his guess bit, i.e. whether the adversary correctly guessed b.

Since the reduction executes the protocol as specified, for $d = 1$ the simulation is indistinguishable to $\mathcal{A}$ from a real protocol. Let $\mathcal{V}$ denote the view of the adversary. Thus, we have

$$\Pr[S = 1 \mid d = 1] = \Pr\left[\mathsf{Exp}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}(\mathcal{V}) = 1\right] = \mathsf{Succ}^{\text{revoting-c-resist}}_{ES,\mathcal{A},H}(\mathcal{V}).$$

On the other hand, as we argued above, the adversary in the simulation does not have any advantage to the adversary in $\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V})$ if $d = 0$. Thus,

$$\Pr[S = 1 \mid d = 0] = \Pr\left[\mathsf{Exp}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V}) = 1\right] = \mathsf{Succ}^{\text{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V}).$$

Concluding the argument, we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{S}}^{\mathsf{ddh}} &= \Pr\left[S = 1 \mid d = 1\right] - \Pr\left[S = 1 \mid d = 0\right] \\ &= \mathsf{Succ}_{ES,\mathcal{A},H}^{\text{revoting-c-resist}}(\mathcal{V}) - \mathsf{Succ}_{ES,\mathcal{A},H}^{\text{revoting-c-resist-ideal}}(\mathcal{V}) \\ &= \mathbf{Adv}_{\mathcal{A}}^{revoting}.\end{aligned}$$

□

6 Conclusion

We investigated the question of how cryptographic proofs of security can improve a systematic security engineering process. To this end, we modeled and analysed security for concrete applications in three practical domains: computer networks, data outsourcing, and electronic voting. We conclude that cryptographic proofs of security can benefit a security engineering process in formulating requirements, influencing design, and identifying constraints for the implementation. Cryptography's tenet of *provable* security adds scientific rigour to areas of security engineering that otherwise were to rely on experience, intuition, and imagination.

6.1 Directions for Further Research

In Chapter 3 we studied combinations of candidate firewalls to yield one provably-secure firewall, even if one candidate is compromised. Firewall combination strategies other than those we discussed need further exploration. In extension to a generalised k-out-of-n quorum approach, one might consider complex firewall networks in the fashion of multiple parallel and serial stages. Also, how to deduce security guarantees for bidirectional communication (via a "Bidirection Theorem" similar to the Composition Theorem in the Universal Composability (UC) [41] framework) from a unidirectional modeling is an important open problem. The question whether the trust assumption on the packet comparator is necessary or can be remedied by a cryptographic protocol is also promising. Future work should also improve our methodology to ease the handling of adaptive corruptions.

Another important direction for future research is to simplify the representation of protocols and guarantees further. Enclosing technicalities in the framework instead of the protocol description facilitates a simpler exposition of core ideas while still yielding a formally sound security analysis. An intuitive access to formal security notions increases their acceptance in practice and thus helps improve the security of systems in use.

During the implementation of the trusted packet comparator a practical concern arose: Our proof of security is based on the assumption that uncorrupted firewalls agree on decisions. While it is a strength of the proof that the assumption is explicit, it can be contested whether such an assumption is practical. Further research into this question promises to provide insight into its practical soundness.

Our work in Chapter 4 provides directions for several interesting new research topics. In particular, the relations between concrete instantiations of our generalised security notions (e.g., if a weaker instantiation of Data Privacy is still strictly separable from a weaker instantiation of Query Privacy) remain an open question, as well as giving a complete hierarchy of instantiations of one security notion (e.g., if Data Privacy with

two queries is weaker than Data privacy with three queries). In particular, a precise characterisation of schemes which fit into each hierarchy class are of interest.

Further, our security notions are game-based. Investigating simulation-based techniques for our purposes might lead to further insights. Since game-based and simulation-based formulations of the same security notion are not necessarily equivalent (as it is in the case with Selective Opening Security [32]), analysing the relations between our framework and a simulation-based variant could further deepen the understanding security notions for outsourcing schemes.

As to our searchable encryption scheme SEDAWG, further research can be directed at extending the scheme to allow modifications or extensions of the encrypted text without the need for a complete re-encryption.

In Chapter 5, we investigated revoting as a strategy to evade coercion. Compared to other mechanisms like, for example, fake credentials, revoting is a very simple and efficient strategy for evading coercion. It is worth pointing out, however, that the underlying assumptions are debatable. While resistance to adversarial coercion is very strong *during* the coercion attempt, we must make certain assumptions to prove the deniability of *any* revoting mechanism. Not only does the coercer have to give up his coercion attempt at some point, but he may also not learn or destroy the voter's secret credentials. These assumptions are not unrealistic per-se, but should be investigated more closely in future work.

6.2 Open Questions

In this work, we developed valuable insights as how to amend a security engineering process by cryptographic modeling and analysis. The practical effort to actually demonstrate such an endeavour is left open. The implementations we discussed are a first step in this direction. They however also serve a different purpose. Only by concretely implementing provably-secure schemes one demonstrates that the underlying model applies to actual systems and is thus fit to provide practical insights. Along the same line, a security model's relevancy to practice cannot be proven directly. It can be disproven by breaking a provably-secure implementation, however. Thus, by failing to break one's implementation, one demonstrates the validity of one's model. We hope efforts in this direction will be made in the future.

Another area worth investigating is integrating the cryptographic method into an agile process. In Chapter 1 we argued that the waterfall model accommodates cryptographic proofs of security particularly well. On the other hand, agile methods adapt better to changing or unclear requirements. An agile process that benefits from the cryptographic method might bring the best of both worlds, or be impossible to implement.

Author's Publications

[1] Dirk Achenbach, Matthias Huber, Jörn Müller-Quade, and Jochen Rill. "Closing the Gap: A Universal Privacy Framework for Outsourced Data". In: *Cryptography and Information Security in the Balkans*. Ed. by Enes Pasalic and Lars R. Knudsen. Vol. 9540. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 134–151. ISBN: 978-3-319-29171-0. DOI: 10.1007/978-3-319-29172-7_9.

[2] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. "Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting". In: *USENIX Journal of Election Technology and Systems (JETS) and Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, EVT/WOTE*. Vol. 3/2. 2015.

[3] Dirk Achenbach, Jörn Müller-Quade, and Jochen Rill. "Universally Composable Firewall Architectures Using Trusted Hardware". In: *Cryptography and Information Security in the Balkans*. Ed. by Berna Ors and Bart Preneel. Vol. 9024. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 57–74. ISBN: 978-3-319-21355-2. DOI: 10.1007/978-3-319-21356-9_5.

[4] Dirk Achenbach, Jörn Müller-Quade, and Jochen Rill. "Synchronous Universally Composable Computer Networks". In: *Cryptography and Information Security in the Balkans*. Ed. by Enes Pasalic and Lars R. Knudsen. Vol. 9540. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 95–111. ISBN: 978-3-319-29171-0. DOI: 10.1007/978-3-319-29172-7_7.

[5] Rolf Haynberg, Jochen Rill, Dirk Achenbach, and Jörn Müller-Quade. "Symmetric Searchable Encryption for Exact Pattern Matching using Directed Acyclic Word Graphs". In: *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*. Ed. by Pierangela Samarati. SciTePress, 2013, pp. 403–410. ISBN: 978-989-8565-73-0.

Students' Theses

[6] Rolf Haynberg. "Symmetric Searchable Encryption für exakte Mustersuche". Diplomarbeit. Karlsruher Institut für Technologie (KIT), 2010.

[7] Simon Karolus. "DPDK-basierte Implementierung eines Paket-Vergleichers im 3-Firewall-Modell". Bachelorarbeit. Karlsruher Institut für Technologie (KIT), 2015.

[8] Jochen Rill. "Implementierung und Analyse eines Verfahrens für durchsuchbare Verschlüsselung zur exakten Mustersuche". Studienarbeit. Karlsruher Institut für Technologie (KIT), 2011.

[9] Jochen Rill. "Sicherheitsarchitekturen für Firewalls in IP-Netzwerken". Diplomarbeit. Karlsruher Institut für Technologie (KIT), 2013.

References

[10] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions". In: *Journal of Cryptology* 21 (3 2008), pp. 350–391. ISSN: 0933-2790.

[11] Masayuki Abe and Fumitaka Hoshino. "Remarks on Mix-Network Based on Permutation Networks". In: *Public Key Cryptography*. Ed. by Kwangjo Kim. Vol. 1992. Lecture Notes in Computer Science. Springer, 2001, pp. 317–324. ISBN: 3-540-41658-7.

[12] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. "Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios". In: *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*. EVT/WOTE'09. Montreal, Canada: USENIX Association, 2009, pp. 10–10.

[13] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. "Provably-secure schemes for basic query support in outsourced databases". In: *Data and Applications Security XXI*. Springer, 2007, pp. 14–30.

[14] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2nd ed. Wiley Publishing, 2008. ISBN: 9780470068526.

[15] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. "A Practical and Secure Coercion-Resistant Scheme for Internet Voting". English. In: *Towards Trustworthy Elections*. Ed. by David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida. Vol. 6000. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 330–342. ISBN: 978-3-642-12979-7. DOI: 10.1007/978-3-642-12980-3_20.

[16] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. "A formalization of the security features of physical functions". In: *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE. 2011, pp. 397–412.

[17] Dmitri Asonov and Johann-Christoph Freytag. "Almost optimal private information retrieval". In: *Privacy enhancing technologies*. Springer. 2003, pp. 209–223.

[18] Michael Backes, Praveen Manoharan, and Esmaeil Mohammadi. "TUC: Time-sensitive and Modular Analysis of Anonymous Communication". In: *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*. IEEE. 2014, pp. 383–397.

[19] Michael Backes, Birgit Pfitzmann, and Michael Waidner. "A General Composition Theorem for Secure Reactive Systems". In: *Theory of Cryptography*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 336–354. ISBN: 978-3-540-21000-9. DOI: 10.1007/978-3-540-24638-1_19.

[20] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. "Public key encryption with keyword search revisited". In: *Computational Science and Its Applications–ICCSA 2008*. Springer, 2008, pp. 1249–1259.

[21] Ricardo Baeza-Yates and Gaston H. Gonnet. "A New Approach to Text Searching". In: *Commun. ACM* 35.10 (1992), pp. 74–82. ISSN: 0001-0782.

[22] B. Barak, R. Canetti, J.B. Nielsen, and R. Pass. "Universally composable protocols with relaxed set-up assumptions". In: *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*. Oct. 2004, pp. 186–195. DOI: 10.1109/FOCS.2004.71.

[23] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/ (visited on 11/28/2015).

[24] Tim Bell, Matt Powell, Joffre Horlor, and Ross Arnold. *The Canterbury Corpus*. 2001. URL: http://corpus.canterbury.ac.nz/ (visited on 12/02/2015).

[25] S.M. Bellovin and W.R. Cheswick. "Network firewalls". In: *Communications Magazine, IEEE* 32.9 (1994), pp. 50–57. ISSN: 0163-6804. DOI: 10.1109/35.312843.

[26] Josh Benaloh. "Rethinking Voter Coercion: The Realities Imposed by Technology". In: *Presented as part of the USENIX Journal of Election and Technology and Systems (JETS)* (2013), pp. 82–87.

[27] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. "PRISM–privacy-preserving search in MapReduce". In: *Privacy Enhancing Technologies*. Springer. 2012, pp. 180–200.

[28] Manuel Blum. "Coin flipping by telephone a protocol for solving impossible problems". In: *ACM SIGACT News* 15.1 (1983), pp. 23–27.

[29] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas. "The smallest automaton recognizing the subwords of a text". In: *Theoretical Computer Science* 40 (1985). Eleventh International Colloquium on Automata, Languages and Programming, pp. 31–55. ISSN: 0304-3975. DOI: DOI:10.1016/0304-3975(85)90157-4.

[30] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. "Complete Inverted Files for Efficient Text Retrieval and Analysis". In: *J. ACM* 34.3 (1987), pp. 578–595. ISSN: 0004-5411. DOI: http://doi.acm.org/10.1145/28869.28873.

[31] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. "Secure multiparty computation goes live". In: *Financial Cryptography and Data Security*. Springer, 2009, pp. 325–343.

[32] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. *On definitions of selective opening security*. Cryptology ePrint Archive, Report 2011/678. http://eprint.iacr.org/2011/678. 2011.

[33] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. "Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator". In: *VOTE-ID 2007*. Ed. by A. Alkassar and M. Volkamer. Vol. 4896. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 111–124.

[34] Dan Boneh and Xavier Boyen. "On the impossibility of efficiently combining collision resistant hash functions". In: *Advances in Cryptology-CRYPTO 2006*. Springer, 2006, pp. 570–583.

[35] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. "Public Key Encryption with Keyword Search". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 506–522.

[36] Robert S. Boyer and J. Strother Moore. "A Fast String Searching Algorithm". In: *Commun. ACM* 20.10 (1977), pp. 762–772. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/359842.359859.

[37] Robert S Boyer and J Strother Moore. "A fast string searching algorithm". In: *Communications of the ACM* 20.10 (1977), pp. 762–772.

[38] Christian Cachin, Silvio Micali, and Markus Stadler. "Computationally Private Information Retrieval with Polylogarithmic Communication". English. In: *Advances in Cryptology – EUROCRYPT '99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1999, pp. 402–414. ISBN: 978-3-540-65889-4. DOI: 10.1007/3-540-48910-X_28.

[39] Jan Camenisch and Markus Stadler. "Efficient group signature schemes for large groups". English. In: *Advances in Cryptology – CRYPTO '97*. Ed. by Jr. Kaliski BurtonS. Vol. 1294. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, pp. 410–424. ISBN: 978-3-540-63384-6. DOI: 10.1007/BFb0052252.

[40] R. Canetti and R. Gennaro. "Incoercible multiparty computation". In: *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*. Oct. 1996, pp. 504–513. DOI: 10.1109/SFCS.1996.548509.

[41] Ran Canetti. "Universally composable security: A new paradigm for cryptographic protocols". In: *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE. 2001, pp. 136–145.

[42] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. `http://eprint.iacr.org/`. 2013.

[43] Ran Canetti, Asaf Cohen, and Yehuda Lindell. "A Simpler Variant of Universally Composable Security for Standard Multiparty Computation". In: (2014).

[44] Ran Canetti, Ivan Damgard, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. "Adaptive versus non-adaptive security of multi-party protocols". In: *Journal of Cryptology* 17.3 (2004), pp. 153–207.

[45] Ran Canetti and Marc Fischlin. "Universally Composable Commitments". English. In: *Advances in Cryptology — CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 19–40. ISBN: 978-3-540-42456-7. DOI: `10.1007/3-540-44647-8_2`.

[46] Ran Canetti and Tal Rabin. "Universal composition with joint state". In: *Advances in Cryptology-Crypto 2003*. Springer, 2003, pp. 265–281.

[47] Ran Canetti and Margarita Vald. "Universally composable security with local adversaries". In: *Security and Cryptography for Networks*. Springer, 2012, pp. 281–301.

[48] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, M Rosu, and Michael Steiner. "Dynamic searchable encryption in very-large databases: Data structures and implementation". In: *Network and Distributed System Security Symposium, NDSS*. Vol. 14. 2014.

[49] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. "Highly-scalable searchable symmetric encryption with support for boolean queries". In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.

[50] Yan-Cheng Chang and Michael Mitzenmacher. "Privacy Preserving Keyword Searches on Remote Encrypted Data". In: *Applied Cryptography and Network Security*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 442–455.

[51] Suresh Chari, Charanjit S Jutla, and Arnab Roy. "Universally Composable Security Analysis of OAuth v2.0". In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 526.

[52] Melissa Chase and Seny Kamara. "Structured Encryption and Controlled Disclosure". English. In: *Advances in Cryptology - ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 577–594. ISBN: 978-3-642-17372-1. DOI: `10.1007/978-3-642-17373-8_33`.

[53] Melissa Chase and Emily Shen. *Pattern Matching Encryption*. Cryptology ePrint Archive, Report 2014/638. `http://eprint.iacr.org/2014/638`. 2014.

[54] Melissa Chase and Emily Shen. "Substring-Searchable Symmetric Encryption". In: *Proceedings on Privacy Enhancing Technologies* 2015.2 (2015), pp. 263–281.

[55] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. *Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation*. `http://www.usenix.org/event/evt08/tech/full_papers/chaum/chaum.pdf`. 2008.

[56] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. London, UK, UK: Springer-Verlag, 1993, pp. 89–105. ISBN: 3-540-57340-2.

[57] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private Information Retrieval". In: *J. ACM* 45.6 (Nov. 1998), pp. 965–981. ISSN: 0004-5411. DOI: `10.1145/293347.293350`.

[58] Jeremy Clark and Urs Hengartner. *Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance*. Cryptology ePrint Archive, Report 2011/166. `http://eprint.iacr.org/`. 2011.

[59] Jeremy Clark and Urs Hengartner. "Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance". In: *Financial Cryptography*. Ed. by George Danezis. Vol. 7035. Lecture Notes in Computer Science. Springer, 2011, pp. 47–61. ISBN: 978-3-642-27575-3.

[60] *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1, Revision 4*. `http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R4.pdf`. Sept. 2012.

[61] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A Secure and Optimally Efficient Multi-Authority Election Scheme". In: Springer-Verlag, 1997, pp. 103–118.

[62] Maxime Crochemore and Renaud Vérin. "On Compact Directed Acyclic Word Graphs". In: *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*. London, UK: Springer-Verlag, 1997, pp. 192–211. ISBN: 3-540-63246-8.

[63] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions". In: *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*. Alexandria, Virginia, USA: ACM, 2006, pp. 79–88. ISBN: 1-59593-518-5. DOI: `http://doi.acm.org/10.1145/1180405.1180417`.

[64] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Full version available at `https://eprint.iacr.org/2006/210`. Alexandria, Virginia, USA: ACM, 2006, pp. 79–88. ISBN: 1-59593-518-5. DOI: `10.1145/1180405.1180417`.

[65] Ivan Damgård and Kasper Dupont. *Universally Composable Disk Encryption Schemes*. Cryptology ePrint Archive, Report 2005/333. `http://eprint.iacr.org/2005/333`. 2005.

[66] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. "Perfectly secure oblivious RAM without random oracles". In: *Theory of Cryptography*. Springer, 2011, pp. 144–163.

[67] Ivan Damgård and Alessandra Scafuro. "Unconditionally secure and universally composable commitments from physical assumptions". In: *Advances in Cryptology-ASIACRYPT 2013*. Springer, 2013, pp. 100–119.

[68] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. "Verifying Privacy-type Properties of Electronic Voting Protocols". In: *Journal of Computer Security* 17.4 (July 2009), pp. 435–487. DOI: `10.3233/JCS-2009-0340`.

[69] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. "Single Database Private Information Retrieval Implies Oblivious Transfer". In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 122–138.

[70] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. "On the Generic Insecurity of the Full Domain Hash". English. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 449–466. ISBN: 978-3-540-28114-6. DOI: `10.1007/11535218_27`.

[71] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. "Unconditional and Composable Security Using a Single Stateful Tamper-Proof Hardware Token." In: *TCC*. Vol. 6597. Springer. 2011, pp. 164–181.

[72] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. "General Statistically Secure Computation with Bounded-Resettable Hardware Tokens". In: *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*. 2015, pp. 319–344. DOI: `10.1007/978-3-662-46494-6_14`.

[73] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. "Implementing Resettable UC-Functionalities with Untrusted Tamper-Proof Hardware-Tokens". In: *TCC*. 2013, pp. 642–661. DOI: `10.1007/978-3-642-36594-2_36`.

[74] Rafael Dowsley, Jörn Müller-Quade, and Tobias Nilges. "Weakening the Isolation Assumption of Tamper-Proof Hardware Tokens". In: *Information Theoretic Security - 8th International Conference, ICITS 2015, Lugano, Switzerland, May 2-5, 2015. Proceedings*. 2015, pp. 197–213. DOI: 10.1007/978-3-319-17470-9_12.

[75] Cynthia Dwork. "Differential Privacy". English. In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Vol. 4052. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35907-4. DOI: 10.1007/11787006_1.

[76] Taher Elgamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology Proceedings of CRYPTO 84*. Ed. by David Chaum George Robert Blakley. Berlin, Heidelberg: Springer-Verlag, 1985, pp. 10–18. ISBN: 978-3-540-15658-1.

[77] Aleksander Essex, Jeremy Clark, and Urs Hengartner. "Cobra: Toward Concurrent Ballot Authorization for Internet Voting". In: *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*. EVT/WOTE'12. Bellevue, WA: USENIX Association, 2012, pp. 3–3.

[78] Sergei Evdokimov, Matthias Fischmann, and Oliver Gunther. "Provable security for outsourcing database operations". In: *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE. 2006, pp. 117–117.

[79] *Federal Information Processing Standards Publication (FIPS 180-4). Secure Hash Standard (SHS)*. Aug. 2015.

[80] *Federal Information Processing Standards Publication (FIPS 197). Advanced Encryption Standard (AES)*. 2001.

[81] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. "The Skein hash function family". In: *Submission to NIST (round 3)* 7.7.5 (2010), p. 3.

[82] Houda Ferradi, Rémi Géraud, David Naccache, and Assia Tria. "When organized crime applies academic results: a forensic analysis of an in-card listening device". In: *Journal of Cryptographic Engineering* (2015), pp. 1–11.

[83] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. University of California, Irvine, 2000.

[84] Ned Freed. "Behavior of and requirements for Internet firewalls". In: *RFC 2979* (2000).

[85] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. "Universally composable security analysis of TLS". In: *Provable Security*. Springer, 2008, pp. 313–327.

[86] William Gasarch. "A Survey on Private Information Retrieval". In: *Bulletin of the EATCS* 82 (2004), pp. 72–107.

[87] Craig Gentry and Zulfikar Ramzan. "Single-Database Private Information Retrieval with Constant Communication Rate". English. In: *Automata, Languages and Programming*. Ed. by Luís Caires, GiuseppeF. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Vol. 3580. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 803–815. ISBN: 978-3-540-27580-0. DOI: 10.1007/11523468_65.

[88] Kristian Gjøsteen. "Analysis of an internet voting protocol." In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 380.

[89] Eu-Jin Goh. "Secure Indexes". In: (2003). http://eprint.iacr.org/2003/216/.

[90] Oded Goldreich. *Foundations of Cryptography*. 1. publ. Vol. 1: Basic Tools. Cambridge: Cambridge University Press, 2001. ISBN: 0-521-79172-3.

[91] Oded Goldreich. *Foundations of Cryptography*. 1. publ. Vol. 2: Basic Applications. Cambridge: Cambridge University Press, 2004. ISBN: 0-521-83084-2 ; 0-521-79172-3.

[92] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *J. ACM* 43.3 (May 1996), pp. 431–473. ISSN: 0004-5411. DOI: 10.1145/233551.233553.

[93] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "One-Time Programs". In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 39–56. ISBN: 978-3-540-85173-8. DOI: 10.1007/978-3-540-85174-5_3.

[94] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. "Universal Re-encryption for Mixnets". In: *Topics in Cryptology – CT-RSA 2004*. Ed. by Tatsuaki Okamoto. Vol. 2964. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 163–178. ISBN: 978-3-540-20996-6. DOI: 10.1007/978-3-540-24660-2_14.

[95] Philippe Golle, Jessica Staddon, and Brent Waters. "Secure Conjunctive Keyword Search over Encrypted Data". In: (2004).

[96] Michael T Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. "Oblivious RAM simulation with efficient worst-case access overhead". In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM. 2011, pp. 95–100.

[97] Gerhard Goos. *Vorlesungen über Informatik: Band 1: Grundlagen und funktionales Programmieren*. 3. Auflage. Springer-Verlag, 2000.

[98] Mohamed G Gouda, Alex X Liu, and Mansoor Jafry. "Verification of distributed firewalls". In: *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*. IEEE. 2008, pp. 1–5.

[99] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. "Founding cryptography on tamper-proof hardware tokens". In: *Theory of Cryptography*. Springer, 2010, pp. 308–326.

[100] James Green-Armytage. *Direct Voting and Proxy Voting*. Available at `http://inside.bard.edu/~armytage/proxy.pdf`. 2014.

[101] Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. "Caveat Coercitor: Coercion-Evidence in Electronic Voting". In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 367–381. ISBN: 978-0-7695-4977-4. DOI: 10.1109/SP.2013.32.

[102] Jens Groth. "A Verifiable Secret Shuffe of Homomorphic Encryptions". In: *Public Key Cryptography - PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. 10.1007/3-540-36288-6_11. Springer Berlin / Heidelberg, 2002, pp. 145–160.

[103] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. "On Robust Combiners for Oblivious Transfer and Other Primitives". In: *Advances in Cryptology EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 96–113. ISBN: 978-3-540-25910-7. DOI: 10.1007/11426639_6.

[104] Orit Hazzan and Yael Dubinsky. *Agile software engineering*. Undergraduate topics in computer science. London: Springer, 2008. ISBN: 978-1-84800-198-5.

[105] Christian Henrich. *Improving and Anaysing Bingo Voting*. Dissertation, Karlsruhe Institute of Technology. 2012.

[106] Amir Herzberg. "Folklore, practice and theory of robust combiners". In: *Journal of Computer Security* 17.2 (2009), pp. 159–189.

[107] I.D. Hill, B.A. Wichmann, and D.R. Woodall. "Algorithm 123: Single Transferable Vote by Meek's Method". In: *Computer Journal* 30.3 (1987), pp. 277–281.

[108] Dennis Hofheinz and Jörn Müller-Quade. "A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer". In: *FCS'04* (2004), p. 117.

[109] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. "Universally composable zero-knowledge arguments and commitments from signature cards". In: *5th Central European Conference on Cryptology*. 2005.

[110] Dennis Hofheinz and Victor Shoup. "GNUC: A New Universal Composability Framework". English. In: *Journal of Cryptology* 28.3 (2015), pp. 423–508. ISSN: 0933-2790. DOI: 10.1007/s00145-013-9160-y.

[111] Dennis Hofheinz and Enav Weinreb. *Searchable Encryption with Decryption in the Standard Model*. Cryptology ePrint Archive, Report 2008/423. 2008.

[112] Hejiao Huang and H. Kirchner. "Formal Specification and Verification of Modular Security Policy Based on Colored Petri Nets". In: *Dependable and Secure Computing, IEEE Transactions on* 8.6 (Nov. 2011), pp. 852–865. ISSN: 1545-5971. DOI: 10.1109/TDSC.2010.43.

[113] Matthias Huber, Matthias Gabel, Marco Schulze, and Alexander Bieber. "Cumulus4j: A Provably Secure Database Abstraction Layer". In: *CD-ARES Workshops*. 2013, pp. 180–193.

[114] Information Science Institute University of Southern California. *Transmission Control Protocol*. RFC 793. RFC Editor, Sept. 1981, pp. 1–85.

[115] *Information Technology Security Evaluation Criteria (ITSEC)*. https://www.bsi.bund.de/DE/Themen/ZertifizierungundAnerkennung/ZertifizierungnachCCundITSEC/ITSicherheitskriterien/ITSEC/itsec_node.html. London, 1991.

[116] Kenneth Ingham and Stephanie Forrest. "A history and survey of network firewalls". In: *University of New Mexico, Tech. Rep* (2002).

[117] Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. "Modeling modern network attacks and countermeasures using attack graphs". In: *Computer Security Applications Conference, 2009. ACSAC'09. Annual*. IEEE. 2009, pp. 117–126.

[118] Spiegel Online International. *Interactive Graphic: The NSA's Spy Catalog*. Dec. 2013. URL: http://www.spiegel.de/international/world/a-941262.html (visited on 11/28/2015).

[119] ITworld. *Curiosity about lines of code*. Aug. 2012. URL: http://www.itworld.com/article/2725085/big-data/curiosity-about-lines-of-code.html (visited on 11/11/2015).

[120] Markus Jakobsson and Ari Juels. "Mix and Match: Secure Function Evaluation via Ciphertexts". English. In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 162–177. ISBN: 978-3-540-41404-9. DOI: 10.1007/3-540-44448-3_13.

[121] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. Alexandria, VA, USA: ACM, 2005, pp. 61–70.

[122] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-Resistant Electronic Elections". English. In: *Towards Trustworthy Elections*. Ed. by David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida. Vol. 6000. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 37–63. ISBN: 978-3-642-12979-7. DOI: 10.1007/978-3-642-12980-3_2.

[123] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. "Concurrent general composition of secure protocols in the timing model". In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. ACM. 2005, pp. 644–653.

[124] Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen. "Analysis of vulnerabilities in Internet firewalls". In: *Computers & Security* 22.3 (2003), pp. 214–232. ISSN: 0167-4048. DOI: http://dx.doi.org/10.1016/S0167-4048(03)00310-9.

[125] Seny Kamara and Charalampos Papamanthou. "Parallel and dynamic searchable symmetric encryption". In: *Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.

[126] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. "Dynamic searchable symmetric encryption". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 965–976.

[127] Myong H Kang, Ira S Moskowitz, and Stanley Chincheck. "The pump: A decade of covert fun". In: *Computer Security Applications Conference, 21st Annual*. IEEE. 2005, 7–pp.

[128] Murat Kantarcıoglu and Chris Clifton. "Security issues in querying encrypted data". In: *Data and Applications Security XIX*. Springer, 2005, pp. 325–337.

[129] Richard M. Karp and Michael O. Rabin. "Efficient Randomized Pattern-Matching Algorithms". In: *IBM Journal of Research and Development* 31.2 (Mar. 1987), pp. 249–260. ISSN: 0018-8646. DOI: 10.1147/rd.312.0249.

[130] Jonathan Katz. "Universally Composable Multi-party Computation Using Tamper-Proof Hardware". In: *Advances in Cryptology – EUROCRYPT 2007*. Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 115–128. ISBN: 978-3-540-72539-8. DOI: 10.1007/978-3-540-72540-4_7.

[131] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. ISBN: 1584885513.

[132] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. "Universally Composable Synchronous Computation". In: *Theory of Cryptography*. Ed. by Amit Sahai. Vol. 7785. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 477–498. ISBN: 978-3-642-36593-5. DOI: 10.1007/978-3-642-36594-2_27.

[133] Carmen Kempka. "Coercion-resistant electronic elections with write-in candidates". In: *Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections*. EVT/WOTE'12. Bellevue, WA: USENIX Association, 2012.

[134] Shahram Khazaei, Tal Moran, and Douglas Wikström. "A Mix-Net from Any CCA2 Secure Cryptosystem". English. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 607–625. ISBN: 978-3-642-34960-7. DOI: 10.1007/978-3-642-34961-4_37.

[135] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. "Fast Pattern Matching in Strings". In: *SIAM Journal on Computing* 6.2 (1977), pp. 323–350.

[136] Philip Koopman. "32-bit cyclic redundancy codes for internet applications". In: *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE. 2002, pp. 459–468.

[137] Steven G Krantz. *The history and concept of mathematical proof*. Accessed via `http://www.math.wustl.edu/~sk/eolss.pdf` (28 September 2007). 2007.

[138] Kaoru Kurosawa and Yasuhiro Ohtaki. "UC-Secure Searchable Symmetric Encryption". English. In: *Financial Cryptography and Data Security*. Ed. by AngelosD. Keromytis. Vol. 7397. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 285–298. ISBN: 978-3-642-32945-6. DOI: `10.1007/978-3-642-32946-3_21`.

[139] Kaoru Kurosawa and Yasuhiro Ohtaki. *How to Construct UC-Secure Searchable Symmetric Encryption Scheme*. Cryptology ePrint Archive, Report 2015/251. `http://eprint.iacr.org/2015/251`. 2015.

[140] E. Kushilevitz and R. Ostrovsky. "Replication is not needed: Single Database, Computationally-Private Information Retrieval". In: *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1997, p. 364. ISBN: 0-8186-8197-7.

[141] Ralf Kusters and Tomasz Truderung. "An epistemic approach to coercion-resistance for electronic voting protocols". In: *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE. 2009, pp. 251–266.

[142] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Proving coercion-resistance of scantegrity II". In: *Information and Communications Security*. Springer, 2010, pp. 281–295.

[143] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Verifiability, privacy, and coercion-resistance: New insights from a case study". In: *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE. 2011, pp. 538–553.

[144] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "A Game-Based Definition of Coercion-Resistance and its Applications". In: *Journal of Computer Security (special issue of selected CSF 2010 papers)* 20.6/2012 (2012), pp. 709–764.

[145] Mirosław Kutyłowski and Filip Zagórski. "Verifiable Internet Voting Solving Secure Platform Problem". English. In: *Advances in Information and Computer Security*. Ed. by Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg. Vol. 4752. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 199–213. ISBN: 978-3-540-75650-7. DOI: `10.1007/978-3-540-75651-4_14`.

[146] Ed. L. Dusseault. *RFC4918 – HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. Request For Comment. `https://tools.ietf.org/html/rfc4918`. Network Working Group, June 2007.

[147] Romain Laborde, Bassem Nasser, Frédéric Grasset, François Barrère, and Abdelmalek Benzekri. "Network security management: A formal evaluation tool based on RBAC policies". In: *Network Control and Engineering for QoS, Security and Mobility, III*. Springer, 2005, pp. 69–80.

[148] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176.

[149] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. "Providing receipt-freeness in mixnet-based voting protocols". In: *Information Security and Cryptology-ICISC 2003*. Springer, 2004, pp. 245–258.

[150] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-closeness: Privacy beyond k-anonymity and l-diversity". In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE. 2007, pp. 106–115.

[151] S. Lipner. "The trustworthy computing security development lifecycle". In: *Computer Security Applications Conference, 2004. 20th Annual*. Dec. 2004, pp. 2–13. DOI: 10.1109/CSAC.2004.41.

[152] Epp Maaten. "Towards Remote E-Voting: Estonian case." In: *Electronic Voting in Europe*. Ed. by Alexander Prosser and Robert Krimmer. Vol. 47. LNI. GI, 2004, pp. 83–100. ISBN: 3-88579-376-8.

[153] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. "l-diversity: Privacy beyond k-anonymity". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), p. 3.

[154] Roel Maes and Ingrid Verbauwhede. "Physically unclonable functions: A study on the state of the art and future research directions". In: *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 3–37.

[155] Ferney A. Maldonado-Lopez, Eusebi Calle, and Yezid Donoso. "Detection and prevention of firewall-rule conflicts on software-defined networking". In: *Reliable Networks Design and Modeling (RNDM), 2015 7th International Workshop on*. Oct. 2015, pp. 259–265. DOI: 10.1109/RNDM.2015.7325238.

[156] Udi Manber and Gene Myers. "Suffix Arrays: A New Method for On-Line String Searches". In: *SODA '90: Proceedings of the first annual ACM-SIAM symposium on discrete algorithms*. San Francisco, California, United States: Society for Industrial and Applied Mathematics, 1990, pp. 319–327. ISBN: 0-89871-251-3.

[157] Ueli Maurer. "Constructive cryptography – A new paradigm for security definitions and proofs". In: *Theory of Security and Applications (TOSCA 2011)*. Ed. by S. Moedersheim and C. Palamidessi. Vol. 6993. Lecture Notes in Computer Science. Springer-Verlag, Apr. 2011, pp. 33–56.

[158] Ueli Maurer and Renato Renner. "Abstract Cryptography". In: *The Second Symposium in Innovations in Computer Science, ICS 2011*. Ed. by Bernard Chazelle. Tsinghua University Press, Jan. 2011, pp. 1–21.

[159] Remo Meier and Bartosz Przydatek. "On Robust Combiners for Private Information Retrieval and Other Primitives". In: *Advances in Cryptology - CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 555–569. ISBN: 978-3-540-37432-9. DOI: 10.1007/11818175_33.

[160] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. "Robuster Combiners for Oblivious Transfer". In: *Theory of Cryptography*. Ed. by SalilP. Vadhan. Vol. 4392. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 404–418. ISBN: 978-3-540-70935-0. DOI: 10.1007/978-3-540-70936-7_22.

[161] James C. III Miller. "A program for direct and proxy voting in the legislative process". English. In: *Public Choice* 7.1 (1969), pp. 107–113. ISSN: 0048-5829. DOI: 10.1007/BF01718736.

[162] Tal Moran and Moni Naor. "Receipt-Free Universally-Verifiable Voting With Everlasting Privacy". In: *Advances in Cryptology – CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Springer, Aug. 2006, pp. 373–392.

[163] Tal Moran and Gil Segev. "David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware". In: *Advances in Cryptology–EUROCRYPT 2008*. Springer, 2008, pp. 527–544.

[164] Steven J Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. "Chip and PIN is Broken". In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 433–446.

[165] Andrew C. Myers, Michael Clarkson, and Stephen Chong. "Civitas: Toward a Secure Voting System". In: *IEEE Symposium on Security and Privacy*. IEEE. May 2008, pp. 354–368.

[166] C. Andrew Neff. *A verifiable secret shuffle and its application to e-voting*. CCS '01 Proceedings of the 8th ACM conference on Computer and Communications Security. 2001.

[167] Doug Newcomb. *WIRED: The Next Big OS War Is in Your Dashboard*. Mar. 2012. URL: http://www.wired.com/2012/12/automotive-os-war/ (visited on 11/11/2015).

[168] Jesper Buus Nielsen. "On protocol security in the cryptographic model". PhD thesis. BRICS, Computer Science Department, University of Aarhus, 2003.

[169] V. Oleshchuk. "Privacy Preserving Pattern Matching on Remote Encrypted Data". In: *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on*. Sept. 2007, pp. 609–613. DOI: 10.1109/IDAACS.2007.4488493.

[170] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. "Universally composable secure computation with (malicious) physically uncloneable functions". In: *Advances in Cryptology–EUROCRYPT 2013*. Springer, 2013, pp. 702–718.

[171] Dong Jin Park, Kihyun Kim, and Pil Joong Lee. "Public Key Encryption with Conjunctive Field Keyword Search". In: *Information Security Applications*. Ed. by Chae Hoon Lim and Moti Yung. Vol. 3325. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 73–86.

[172] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. "A Framework for Efficient and Composable Oblivious Transfer". English. In: *Advances in Cryptology—CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 554–571. ISBN: 978-3-540-85173-8. DOI: 10.1007/978-3-540-85174-5_31.

[173] Birgit Pfitzmann and Michael Waidner. "A model for asynchronous reactive systems and its application to secure message transmission". In: *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE. 2001, pp. 184–200.

[174] Benny Pinkas and Tzachy Reinman. "Oblivious RAM revisited". In: *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 502–519.

[175] Riigikogu. *Riigikogu Election Act*. Riigi Teataja. https://www.riigiteataja.ee/en/eli/ee/501092014005/consolide/current. 2002.

[176] Kazue Sako and Joe Kilian. "Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth". In: *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*. EUROCRYPT'95. Saint-Malo, France: Springer-Verlag, 1995, pp. 393–403. ISBN: 3-540-59409-4.

[177] C.P. Schnorr. "Efficient signature generation by smart cards". English. In: *Journal of Cryptology* 4.3 (1991), pp. 161–174. ISSN: 0933-2790. DOI: 10.1007/BF00196725.

[178] Christoph L Schuba and Eugene H Spafford. "A reference model for firewall technology". In: *Computer Security Applications Conference, 1997. Proceedings., 13th Annual*. IEEE. 1997, pp. 133–145.

[179] Ken Schwaber. *Agile project management with Scrum*. Microsoft professional. Redmond, Wash.: Microsoft Press, 2004. ISBN: 0-7356-1993-X; 978-0-7356-1993-7.

[180] SecurityFocus. *Symantec Norton Personal Firewall/Internet Security 2001 Buffer Overflow Vulnerability*. July 2002. URL: http://www.securityfocus.com/bid/5237 (visited on 18/11/2015).

[181] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[182] Claude E Shannon. "Communication theory of secrecy systems*". In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

[183] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. "Oblivious RAM with O ((logN) 3) worst-case cost". In: *Advances in Cryptology–ASIACRYPT 2011*. Springer, 2011, pp. 197–214.

[184] Victor Shoup. *Why Chosen Ciphertext Security Matters*. Tech. rep. Rüschlikon: IBM Research Division, Zurich Research Laboratory, 1998.

[185] Radu Sion and Bogdan Carbunar. "On the computational practicality of private information retrieval". In: *In Proceedings of the Network and Distributed Systems Security Symposium*. 2007.

[186] Michael Sipser. *Introduction to the Theory of Computation*. 3rd edition. Cengage Learning, 2013.

[187] Warren D. Smith. *New cryptographic election protocol with best-known theoretical properties*. Frontiers in Electronic Elections (FEE 2005). 2005.

[188] D. Xiaodong Song, D. Wagner, and A. Perrig. "Practical Techniques for Searches on Encrypted Data". In: *IEEE Symposium on Security and Privacy* (2000). `http://citeseer.nj.nec.com/song00practical.html`, pp. 44–55.

[189] Oliver Spycher, Rolf Haenni, and Eric Dubuis. "Coercion-resistant hybrid voting systems". In: *Electronic Voting*. 2010, pp. 269–282.

[190] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. *Practical Dynamic Searchable Encryption with Small Leakage*. Cryptology ePrint Archive, Report 2013/832. `http://eprint.iacr.org/2013/832`. 2013.

[191] Malcolm W Stevens and M Pope. *An Implementation of an Optical Data Diode*. Citeseer, 1999.

[192] Latanya Sweeney. "k-anonymity: A model for protecting privacy". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[193] Esko Ukkonen. "On-Line Construction of Suffix Trees". In: *Algorithmica* 14.3 (1995), pp. 249–260.

[194] Dominique Unruh and Jörn Müller-Quade. "Universally Composable Incoercibility". English. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 411–428. ISBN: 978-3-642-14622-0. DOI: `10.1007/978-3-642-14623-7_22`.

[195] Gerald V. Post. "Using re-voting to reduce the threat of coercion in elections". In: *Electronic Government, an International Journal*. Volume 7, Number 2/2010. Inderscience Publishers, 2010, pp. 168–182.

[196] Giovanni Vigna. *A formal model for firewall testing*. 2002.

[197] Melanie Volkamer and Rüdiger Grimm. "Multiple Casts in Online Voting: Analyzing Chances". In: *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria*. Ed. by Robert Krimmer. Vol. 86. LNI. GI, 2006, pp. 97–106. ISBN: 978-3-88579-180-5.