Towards Solving Real-World Challenges for Privacy-Preserving Systems

Zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften

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

genehmigte

Dissertation

von

Valerie Maria Fetzer

aus Karlsruhe

Tag der mündlichen Prüfung:	18. Dezember 2024
1. Referent:	Prof. Dr. Jörn Müller-Quade
2. Referent:	Prof. Dr. Daniel Slamanig
Betreuer:	Dr. Andy Rupp

Abstract

In today's digital world, many applications handle privacy-sensitive data, and protecting users' privacy should be of paramount importance, especially given the ever-present risk of data breaches and misuse of personal information. Allowing users to interact with these systems anonymously is desirable because it reduces the risks associated with the disclosure of personal information. However, there are significant challenges in balancing the desired anonymity with the requirements of real-world systems: For example, users may exploit anonymity to cheat, so mechanisms are needed to ensure accountability even in anonymous systems. In light of this, a primary focus of this thesis is on balancing privacy and real-world requirements in privacy-sensitive systems. These real-world requirements vary by application, but common ones include dealing with malicious users, broken physical hardware, and legal requirements. We now outline our contributions to this topic.

We are surrounded by an ever-growing number of privacy-sensitive cyber-physical systems such as electronic toll collection systems, cashless payment systems for public transport, the smart grid, smart meters, and many more. Building secure cyber-physical systems that could be used in practice brings great challenges. They should be provably secure, provably protect the privacy of users, be able to handle malicious or uncooperative users and broken hardware, all while being efficient enough to be used in practice. The first part of this thesis deals with how to build cyber-physical systems such that they are provably secure under standard cryptographic assumptions, practically efficient, provide strong privacy guarantees for the user and deal with real-world issues like uncooperative users and broken hardware. As an example of a privacy-preserving cyber-physical system that will be increasingly important in our digital future, we focus on the electronic toll collection (ETC) scenario. In this thesis, we propose P4TC: a flexible security model and cryptographic protocol framework designed for privacy-preserving ETC. P4TC is the first privacy-preserving ETC system in the Dedicated Short Range Communication setting with a rigorous security model and proof in the Universal Composability (UC) framework while being efficient enough to be used in practice. It also addresses several real-world issues: To mitigate the consequences of a user account becoming inaccessible (for example, due to physical failure of the on-board unit or theft of the vehicle), user accounts can be temporarily suspended from the system and the user's outstanding debt can be recalculated. There are also several mechanisms to deal with malicious users, as well as a mechanism to mitigate financial loss due to compromised road-side units.

In privacy-preserving systems, it is sometimes necessary to revoke the anonymity of users, e.g., to combat revenue loss due to malicious users or to comply with legal requirements. While the simplest way to achieve this is through a system-wide backdoor, this approach theoretically grants the central authority holding the backdoor the ability to compromise the privacy of all users. To leverage the power of the central party a bit, some systems have moved from a system-wide backdoor to user-specific backdoors. While this is more privacy-friendly than a system-wide backdoor, it does not solve the problem of having to trust the central instance not to misuse these backdoors. To address the problem of central parties potentially misusing backdoors, the second part of this thesis deals with making the use of a user-specific backdoor auditable. That is, it should become visible, e.g., to a special auditor party, if a user-specific backdoor has been used. In this thesis, we therefore develop a generic building block for auditable surveillance. In order to prevent the misuse of backdoors, we employ short-term user-specific backdoors that are protected in multiple ways: Backdoor access capabilities are shared between trustworthy parties to avoid a single point of failure, and backdoor access is given only conditionally. In addition, there are audit trails and public statistics for every (granted) backdoor request, and surveillance remains silent, i.e., users do not know that they are being surveilled. More precisely, we present an ideal functionality in the UC framework that can be used to augment existing applications with auditable surveillance capabilities. Our realization of this functionality uses threshold cryptography to protect the short-term user-specific backdoors, and backdoor access is handled by an anonymous evolving committee. This committee can verify that the predefined conditions for backdoor access are fulfilled, e.g., that law enforcement is in possession of a valid surveillance warrant. Our auditable surveillance system is the first that can be used to enhance existing systems with conditional and auditable backdoor access under reasonable assumptions, while scaling well and having a proof in the UC framework.

Zusammenfassung

In der heutigen digitalen Welt verarbeiten viele Anwendungen datenschutzsensible Daten, und der Schutz der Privatsphäre der Nutzer sollte von höchster Bedeutung sein, insbesondere angesichts der allgegenwärtigen Gefahr von Datenpannen und des Missbrauchs personenbezogener Daten. Es ist wünschenswert, den Nutzern die Möglichkeit zu geben, anonym mit diesen Systemen zu interagieren, da dies die mit der Preisgabe persönlicher Informationen verbundenen Risiken verringert. Es ist jedoch eine große Herausforderung, die gewünschte Anonymität mit den Anforderungen realer Systeme in Einklang zu bringen: Beispielsweise könnten Nutzer ihre Anonymität ausnutzen, um zu betrügen, weshalb Mechanismen erforderlich sind, die die Rechenschaftspflicht auch in anonymen Systemen sicherstellen. Vor diesem Hintergrund liegt der Schwerpunkt dieser Dissertation darauf, eine Balance zwischen der Privatsphäre der Nutzer und den Anforderungen der realen Welt in datenschutzsensiblen Systemen zu finden. Diese Anforderungen variieren je nach Anwendung, umfassen jedoch oftmals den Umgang mit böswilligen Nutzern, defekter physischer Hardware und rechtlichen Anforderungen. Im Folgenden skizzieren wir unsere Beiträge zu diesem Thema.

Wir sind umgeben von einer ständig wachsenden Zahl datenschutzsensibler cyber-physischer Systeme, wie beispielsweise elektronischen Mautsystemen, bargeldlosen Bezahlsystemen für den öffentlichen Nahverkehr, dem intelligenten Stromnetz („Smart Grid"), intelligenten Stromzählern („Smart Meter") und vielen anderen. Die Entwicklung sicherer cyber-physischer Systeme, die in der Praxis eingesetzt werden könnten, stellt erhebliche Herausforderungen dar. Sie müssen beweisbar sicher sein, die Privatsphäre der Nutzer beweisbar schützen, mit böswilligen oder unkooperativen Nutzern und defekter Hardware umgehen können und gleichzeitig effizient genug sein, um in der Praxis eingesetzt werden zu können. Der erste Teil dieser Dissertation beschäftigt sich mit der Frage, wie cyber-physische Systeme so entwickelt werden können, dass sie unter gängigen kryptographischen Annahmen beweisbar sicher sind, praktisch effizient sind, starke Privatsphäre-Garantien für den Nutzer bieten und mit praxisrelevanten Problemen wie unkooperativen Nutzern und defekter Hardware umgehen können. Als Beispiel für ein cyber-physisches System, das die Privatsphäre der Nutzer schützt und das in unserer digitalen Zukunft zunehmend an Bedeutung gewinnen wird, konzentrieren wir uns auf das Szenario der elektronischen Mauterhebung. In dieser Dissertation stellen wir P4TC vor: ein flexibles Sicherheitsmodell und ein kryptographisches Protokoll-Framework, das für die privatsphärenschonende elektronische Mauterhebung entwickelt wurde. P4TC ist das erste privatsphärenschonende elektronische Mautsystem im Dedicated Short Range Communication Umfeld mit einem rigorosen Sicherheitsmodell und einem Beweis im Universal Composability (UC) Framework, das gleichzeitig effizient genug ist, um in der Praxis eingesetzt zu werden. Es adressiert auch mehrere praxisrelevante Probleme: Um die Folgen eines nicht mehr zugänglichen Nutzerkontos (zum Beispiel aufgrund eines physischen Ausfalls der On-Board-Unit oder eines Diebstahls des Fahrzeugs) abzumildern, können Nutzerkonten vorübergehend aus dem System ausgeschlossen und die ausstehenden Schulden des Nutzers neu berechnet werden. Darüber hinaus gibt es mehrere Mechanismen zum Umgang mit böswilligen Nutzern sowie einen Mechanismus zur Minimierung finanzieller Verluste durch kompromittierte Road-Side Units.

In privatsphäreschonenden Systemen ist es manchmal notwendig, die Anonymität der Nutzer aufzuheben, zum Beispiel um Umsatzeinbußen durch böswillige Nutzer zu vermeiden oder um gesetzlichen

Anforderungen zu genügen. Dies lässt sich am einfachsten durch eine systemweite Backdoor („Hintertür“) erreichen. Allerdings gibt dieser Ansatz der zentralen Instanz, die die Backdoor besitzt, theoretisch die Möglichkeit, die Privatsphäre aller Nutzer zu kompromittieren. Um die Macht der zentralen Instanz etwas einzuschränken, sind einige Systeme von einer systemweiten Backdoor zu nutzerspezifischen Backdoors übergegangen. Dies ist zwar besser für die Privatsphäre als eine systemweite Backdoor, löst aber nicht das Problem, dass man der zentralen Instanz vertrauen muss, dass sie diese Backdoors nicht missbraucht. Um das Problem des potentiellen Missbrauchs von Backdoors durch die zentrale Instanz zu lösen, befasst sich der zweite Teil dieser Dissertation damit, die Verwendung einer nutzerspezifischen Backdoor nachprüfbar zu machen. Das heißt, es sollte sichtbar gemacht werden, zum Beispiel für einen speziellen Auditor, wenn eine nutzerspezifische Hintertür verwendet wurde. In dieser Dissertation entwickeln wir daher einen generischen Baustein für nachprüfbare Überwachung. Um den Missbrauch von Backdoors zu verhindern, verwenden wir kurzlebige, nutzerspezifische Backdoors, die auf mehrere Arten geschützt sind: Die Zugriffsrechte auf die Backdoors werden zwischen vertrauenswürdigen Parteien geteilt, um einen Single Point of Failure zu vermeiden, und der Zugriff auf die Backdoors wird nur unter bestimmten Bedingungen gewährt. Darüber hinaus gibt es Prüfpfade und öffentliche Statistiken für jede (gewährte) Backdoor-Anfrage, und die Überwachung erfolgt im Stillen, d. h. die betroffenen Nutzer wissen nicht, dass sie überwacht werden. Genauer gesagt stellen wir eine ideale Funktionalität im UC-Framework vor, die verwendet werden kann, um bestehende Anwendungen mit nachprüfbaren Überwachungsfunktionen zu erweitern. Unsere Realisierung dieser Funktionalität verwendet Threshold-Kryptographie („Schwellwertkryptographie“), um die kurzlebigen nutzerspezifischen Backdoors zu schützen, und der Zugang zu den Backdoors wird von einem anonymen, häufig wechselnden Komitee verwaltet. Dieses Komitee kann überprüfen, ob die vordefinierten Bedingungen für den Backdoor-Zugang erfüllt sind, zum Beispiel ob die Strafverfolgungsbehörden im Besitz eines gültigen Überwachungsbeschlusses sind. Unser nachprüfbares Überwachungssystem ist das erste, das dazu verwendet werden kann, um bestehende Systeme mit konditionalem und nachprüfbarem Backdoor-Zugang unter realistischen kryptographischen Annahmen zu erweitern, während es gleichzeitig gut skalierbar ist und einen Beweis im UC-Framework hat.

Own Publications

[Bro+18] Brandon Broadnax, **Valerie Fetzer**, Jörn Müller-Quade, and Andy Rupp. "Non-malleability vs. CCA-Security: The Case of Commitments". In: *PKC 2018, Part II.* Vol. 10770. LNCS. 2018, pp. 312–337.
DOI: 10.1007/978-3-319-76581-5_11.

[Fau+25] Dennis Faut, **Valerie Fetzer**, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. "POBA: Privacy-Preserving Operator-Side Bookkeeping and Analytics". In: *IACR Communications in Cryptology* 2.2 (7, 2025).
DOI: 10.62056/av11zo-3y.

[Fet+20] **Valerie Fetzer**, Max Hoffmann, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. "P4TC – Provably-Secure yet Practical Privacy-Preserving Toll Collection". In: *PoPETs* 2020.3 (2020), pp. 62–152.
DOI: 10.2478/popets-2020-0046.

[Fet+22] **Valerie Fetzer**, Marcel Keller, Sven Maier, Markus Raiber, Andy Rupp, and Rebecca Schwerdt. "PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics". In: *PoPETs* 2022.2 (2022), pp. 447–516.
DOI: 10.2478/popets-2022-0054.

[Fet+23] **Valerie Fetzer**, Michael Klooß, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. "Universally Composable Auditable Surveillance". In: *ASIACRYPT 2023, Part II.* Vol. 14439. LNCS. 2023, pp. 453–487.
DOI: 10.1007/978-981-99-8724-5_14.

[FMN16] **Valerie Fetzer**, Jörn Müller-Quade, and Tobias Nilges. "A Formal Treatment of Privacy in Video Data". In: *ESORICS 2016, Part II.* Vol. 9879. LNCS. 2016, pp. 406–424.
DOI: 10.1007/978-3-319-45741-3_21.

[Jol+25] Amirhossein Adavoudi Jolfaei, Andy Rupp, Stefan Schiffner, and **Valerie Fetzer**. "Differential Privacy to the Rescue? On Obfuscating Tolls in Privacy-Preserving ETC Systems". In Submission. 2025.

[Sch+19] Rebecca Schwerdt, Matthias Nagel, **Valerie Fetzer**, Tobias Gräf, and Andy Rupp. "P6V2G: A Privacy-Preserving V2G Scheme for Two-Way Payments and Reputation". In: *Energy Informatics* 2.1 (27, 2019), p. 32.
DOI: 10.1186/s42162-019-0075-1.

Acknowledgments / Danksagung

Im Folgenden möchte ich mich bei all jenen bedanken, die mich während meiner Promotionszeit unterstützt, motiviert, inspiriert, begleitet und bereichert haben.

Zuallererst möchte ich mich ganz herzlich bei meinen Betreuern Jörn Müller-Quade und Andy Rupp bedanken. Andy danke ich dafür, dass er mir die Promotionsstelle angeboten hat, für seine stetige Unterstützung und den fachlichen Austausch sowie für die Einladungen nach Luxemburg. Jörn danke ich für seinen Enthusiasmus für Kryptographie, für die Freiheit, die ich bei meiner Arbeit genießen durfte, und für die angenehme Atmosphäre an seinem Lehrstuhl. Mein besonderer Dank gilt Daniel Slamanig, der sich die Zeit genommen hat, als Zweitgutachter meiner Arbeit zu fungieren. Ebenso möchte ich all meinen Co-Autoren für die konstruktive Zusammenarbeit und die intensiven Diskussionen danken.

Auch vor meiner Promotion konnte ich bereits etwas in Jörns Lehrstuhl hineinschnuppern: Als Tutor für die Vorlesung „TGI“ habe ich die Übungsleiter Dirk und Tobias kennengelernt, die mich dazu motiviert haben an Jörns Lehrstuhl meine Abschlussarbeiten zu schreiben. Ein besonderer Dank geht an Tobias für die Betreuung meiner Bachelorarbeit, für Kreta und für hilfreiche Gespräche. Ebenso möchte ich Brandon für die Betreuung meiner Masterarbeit danken. Vielen Dank auch an Dennis Hofheinz, dessen Vorlesung im Stammmodul „Sicherheit“ mein Interesse an der Kryptographie geweckt hat.

Über die Jahre hatte ich die Ehre, mit zahlreichen Kolleginnen und Kollegen zusammenzuarbeiten. Vielen Dank an alle ehemaligen und aktuellen Kolleginnen und Kollegen der Lehrstühle von Jörn und Dennis, die den gemeinsamen Flur bereichert haben. Insbesondere danken möchte ich (in pseudozufälliger Reihenfolge): Sven und Thomas für die Party-Überfälle im Dreierbüro und den großartigen Kalender; Michael, Sven und Thomas für eine tolle Kanufahrt; Michael für seine Hingabe an wirklich korrekte Definitionen und dafür, dass mein Schreibtisch nie der vollste im Büro war; Sven für den schönen Trip nach Japan (und Kathrin und Anna-Louise für das Ermöglichen dieses Trips); Thomas und Dani für die Organisation der Wanderausflüge und für Brasilien; Markus für seine MPC-Expertise und dafür, dass ich so oft länger im Büro geblieben bin als eigentlich geplant; Astrid dafür, dass sie mich immer motiviert hat, und für sehr viel Organisation am Lehrstuhl; Rebecca für meinen angestiegenen Teekonsum und für Lambda; Marcel und Robin für die Debatte über den besten Hin- und Rückweg zur Mensa; Robin für Diskussionen zum Thema Zeitreisen in Harry Potter; Felix für seinen nie endenden Vorrat an Krypto-Geschichten aus der „echten Welt“; Eva für die Brettspielabende; Clemens für seine schauspielerische Leistung; Lukas für die Bekanntmachung mit der kulinarischen Gaumenfreude des Mais-Erdnuss-„Salats“; Björn für gute Ratschläge; Alex für sein LaTeX-Wissen; Jeremias und Matthias N. für ihr UC-Wissen; Dennis H. für unsere Namenswortspiele; Akin für einen sehr witzigen RPG-Abend und dafür, dass es in seiner Gegenwart nie langweilig wird; Julia K. für die Dekoration aller Whiteboards mit Eichhörnchen; Lisa für Inspiration; Jessi für die Verteilung von Spitznamen; dem ganzen C1 Team für die produktive aber auch lustige Zeit. Auch möchte ich allen Kolleginnen und Kollegen danken, die meine Reisen zu Konferenzen und Workshops bereichert haben. Nicht zuletzt möchte ich Carmen, Willi und Holger dafür danken, dass am Lehrstuhl alles wie geschmiert läuft und für ihre vermutlich unendliche Geduld. Thanks to the Table Tennis Club for the (much needed) breaks from work and the Bouldering & Pub Crew for enjoyable evenings. Thanks to Geoffroy for your just incredible knowledge about cryptography. Thanks to Bogdan for ensuring that I can eat my lunch at a comfortable pace. Thanks to Gabriela for her incredible Excel skills that might have saved our sanity.

Auch möchte ich mich bei allen aus dem H^{2}T-Kosmos bedanken für den interdisziplinären Austausch zum Thema Promotion sowie vielfältige Unternehmungen. Insbesondere möchte ich mich bei Isabel und Florian bedanken für die gemeinsamen Ausflüge; bei der entspannten Wandergruppe für die entspannten Wanderungen; bei Julia B. für Nachmittage voller Tee und fürs ins Leben rufen der DSA-Runde; und bei der DSA-Truppe für unzählige Abende mit ein bisschen Abenteuer und vielen Abschweifungen.

Zu guter Letzt möchte ich mich bei meiner Familie und meinen Freunden bedanken: meinen Eltern, vor allem meiner Mutter, die immer an mich glaubt; meinem Stiefvater, der immer da ist, wenn man ihn braucht; Eve, die mich stets motiviert hat. Mein besonderer Dank gilt meiner besseren Hälfte Pascal für seine Unterstützung, seinen Ansporn und seine Geduld mit mir. Diese Arbeit wäre ohne dich nicht dieselbe.

Contents

1. Introduction

The term *cryptography* stems from the Ancient Greek words “kryptós” (hidden) and “gráphein” (to write) and can therefore be defined as “hidden writing” [Ps10]. This is fitting, since historically (symmetric) encryption[1] has been the most prominent form of cryptography. The use of encryption goes back thousands of years, with well-known examples being the skytale used in ancient Greece and the Caesar cipher used by Julius Caesar. The Oxford English dictionary defines cryptography either as “the art or practice of writing in code or cipher” or as “the science of encryption”. While the former accurately describes the historical use of cryptography, the latter does not really capture the fact that today cryptography permeates our everyday lives: One can send signed and encrypted emails, chat with people in end-to-end encrypted instant messengers, use passwords to access online accounts, make electronic payments, etc., all while (maybe unknowingly) using cryptography.

One exciting field that has emerged in recent years is the use of cryptography to secure *cyber-physical systems (CPS)*[2]. A prominent subclass of cyber-physical systems are cyber-physical systems that interact directly with users and contain a (post-)payment or point collection function. Examples include electronic toll collection, cashless payment systems for public transport (like London’s Oyster Card or the Japanese IC cards), charging/discharging of electric vehicles in smart grids, and smart meters. For all these scenarios one can imagine a system in which users accumulate “points” (which can be toll fees, fares, or kWh) over some time period and at some point turn in those points to settle the debt.

Since these CPSs involve the processing of (potentially sensitive) user data, there is a natural tension between the need to collect user data and the importance of protecting user privacy. In order to protect user privacy, it is natural to ask whether the collection of user data could simply be omitted. However, the collection of user data serves several important purposes: First and foremost, the data is needed to ensure the functionality of the system (e.g., to calculate the correct total charge for a user). However, operators also use the collected data to perform various types of analyses, the results of which can be used, for example, to improve their service or to analyze market competitiveness. Using the above applications as examples, an electronic toll collection operator can use the collected user data to calculate road usage and plan where to expand roads in the future; a public transportation provider can calculate network usage to better plan where and when to run more or fewer trains; an electric vehicle charging station provider can calculate where to build new charging stations; and a smart meter provider can better predict the load on the electricity grid.

In the European Union, the General Data Protection Regulation (GDPR) [Eur16] defines legal requirements for the processing of user data from EU citizens. Among other things, the GDPR requires that data processing must be *lawful* [Eur16, Articles 5.1(a) & 6.1], with examples for lawfulness including ensuring system functionality, legitimate interests or user consent. Clearly, there are many legitimate reasons for storing and processing sensitive user data. However, if this data is not adequately protected,

[1] The goal of encryption is to confidentially transmit a message from one person to another, i.e., no one except the designated receiver should be able to read the message in the clear.

[2] According to [Col+21], a cyber-physical system is a “computer system able to interact continuously with the physical system in which it operates” and “cyber-physical systems have been applied in various fields from smart grids to medical devices, smart factories, intelligent transport systems, smart cities, and smart buildings”.

there is a risk of privacy violations. There are examples that show that mishandling user data, such as user location data, can have serious consequences for user privacy [ABC08; ACL15; Ame15; Dat07; de +13; Ele24; GP09; Kru07; Tie10]. For instance, [Ele24] describes how some car companies sell the data they collect on their customers' driving behavior, for example to analytics companies. There are also many cases where cyber attackers breached company networks and stole large amounts of user-specific data [CSO23; IT 23]. The GDPR also states the principle of *data minimisation*: "Personal data shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed" [Eur16, Article 5.1(c)]. With cryptography, we can implement the principle of data minimization particularly well, since with suitable cryptographic primitives and protocols we need to store less data with the operator than in conventional systems. This approach also fits very well with the GDPR, as operators are responsible for the confidentiality of the stored data [Eur16, Article 5]. After all, data that is not stored cannot be misused.

In this work, we use cryptography to build privacy preserving systems, which in itself is not a new concept. Taking electronic toll collection as an example, several privacy-preserving solutions have been developed by the research community (cf. Section 3.1.1). However, solutions developed by theoretical cryptographers tend to overlook the requirements of real-world systems, such as the fact that physical hardware can break, get lost or stolen. On the other side, in practice, ETC systems are deployed all over the world, but the majority rely on user identification at each toll point. Consequently, our objective is to bring these two sides closer together. In particular, the main goal of this thesis can be stated as follows:

> ***Main Goal:***
> *This thesis aims towards bridging the gap between theoretical cryptography and practice by building privacy-preserving systems with a solid cryptographic foundation while taking into account several real-world requirements.*

When designing privacy-preserving systems for real-world problems, there are additional challenges to consider beyond those of building purely theoretical cryptographic protocols. The typical requirements for privacy-preserving cryptographic protocols are correctness, efficiency (which for theoretical cryptographers usually means polynomial runtime), and provable security and privacy guarantees for meaningful definitions of "security" and "privacy". Security is primarily about ensuring system functionality even when some parties (such as system operators and users) are controlled by an adversary. Privacy deals with how much information system operators can obtain about their users. The less operators learn about their users, the more privacy the system achieves. Ideally, one would want an anonymous[3] system. The additional real-world requirements differ for each specific application, but common challenges are:

(1) The system must be efficient enough to be used in practice.

(2) One needs to think about what happens after an protocol abort, since aborts can have consequences on the physical world.

(3) Users can be malicious and can for example give dishonest inputs.

(4) Physical hardware, such as user devices, can be broken, lost or stolen.

(5) For systems involving payments, billing must be correct (even for anonymous systems).

(6) The system needs to fulfill all legal requirements.

[3] As the word "anonymous" is sometimes used differently in different contexts, we want to briefly note that when we mention an "anonymous system" in this thesis, we mean a pseudonymous system with unlinkable transactions.

Challenges (3) to (6) all imply the need for some form of user *accountability* in case something goes wrong, e.g., an anonymous customer refuses to pay after using the service offered by the system. This contrasts with the desire for an anonymous system, which is why we use *conditional anonymity revocation* to achieve accountability in an anonymous system. When combining the requirements for privacy-preserving systems with real-world requirements, the main challenge is to find the right balance between anonymity and accountability.

In order to achieve our main goal, we address it in two stages: First, we investigate how to build privacy-preserving solutions for privacy-sensitive CPSs, taking into account real-world requirements. Second, we examine how conditional anonymity revocation can be used to achieve a balance between anonymity and accountability. In the following, we elaborate on both goals.

Let us first look again at privacy-sensitive CPSs, using the application examples discussed earlier: electronic toll collection systems, cashless payment systems for public transport, charging/discharging of electric vehicles in smart grids, and smart meters. The first goal of this thesis is to provide a *privacy-preserving* solution for these kind of systems. Users shall be able to anonymously collect points, but turning in points is non-anonymous and only reveals the total sum of points (and not the individual points gained during the anonymous transactions) to the system operator. This means that the operator will not be able to track the individual transactions of the user. Using the running examples from above, this translates into the operator not learning about individual toll stations visited, or individual subway rides, or individual charging processes, or individual smart meter readings.

In this thesis, we focus on the electronic toll collection scenario as an example for a privacy-sensitive CPS. Specifically, we develop a flexible security model and cryptographic protocol for privacy-preserving electronic toll collection that satisfies all of the aforementioned challenges: It is correct, provably secure under standard cryptographic assumptions, practically efficient, provides strong privacy guarantees for the users, and provides solutions to real-world problems such as uncooperative users and broken/lost/stolen user-side hardware.

While the development of a privacy-preserving solution for electronic toll collection already represents an important contribution to a more privacy-aware future, it also serves as an exemplary solution for the more general first goal of this thesis, which is the following:

> ***First Goal:***
> *This thesis aims to develop privacy-preserving solutions for privacy-sensitive CPSs and overcoming the challenges that arise from building systems that influence and get influenced by the physical world.*

When looking at our running examples for privacy-preserving cyber-physical point collection systems, then one finds that they all contain some kind of *payment component*. The idea is that users pay one bill at the end of a billing period for all transactions (individual toll charges/ride fares/electricity consumption) made during that billing period. The individual transactions are anonymous from the operator's point of view since they are stored on the user's side, but it is cryptographically ensured that users cannot tamper with their transaction record undetected. However, since the operator cannot link individual transactions to users, uncooperative users must be taken into account: How to deal with users who use the benefits of the system for a whole billing period but then do not participate in the billing process? The operator does not even know how much money the user has to pay. To deal with this, these systems need some kind of *anonymity revocation mechanism*. In the case that users refuse to take part in the billing process (or show otherwise fraudulent behavior), the operator needs to be able to map the previously anonymous transactions to the uncooperative/fraudulent user. For that purpose, some kind of *backdoors* are necessary for these systems. Historically, *system-wide backdoors* have been popular [Ate+00; CMS96; KP98], meaning that operators could theoretically use the same backdoor to deanonymize *all* transactions in the system. Since backdoors are typically stored at the operator's side, this harbors great risks for misuse: Without some kind of control mechanism, the operator could in theory deanonymize every transaction in the system, thus completely undermining user privacy. Since the research community also has recognized this privacy risk, newer systems have switched to *user-specific backdoors* that can only deanonymize the transactions of a single user. In our privacy-preserving electronic toll collection system, we go one step further and use *short-term user-specific backdoors*. There, a single backdoor can only deanonymize the transactions made by a single user during a specific billing period. But regardless of whether there is one backdoor for everything or many backdoors, there is always the problem that if the operator simply has access to the backdoor(s), nothing prevents him from using them. To prevent this, we employ the help of a trusted third party (TTP) in our privacy-preserving electronic toll collection system. Put simply, the user encrypts the backdoor for their own transactions under the TTP's key. If the operator wants to deanonymize the user's transactions, he asks the TTP to decrypt the backdoor. The TTP complies with the request on the condition that the reason for the deanonymization provided by the operator to the TTP is plausible.

Since TTPs are sometimes necessary, but a rather undesirable assumption in the cryptography community, we now ask the question of whether the backdoor problem can be solved in a different way, i.e., without a TTP. To do this, let us go back to the topic of privacy-preserving systems in general. As mentioned previously, one category of real-world requirements that must be met when bringing a privacy-preserving system into practice is *legal requirements*. In fact, many applications face legal requirements or regulations that prohibit unconditional anonymity guarantees, e.g., in electronic payments where monitoring is mandated to investigate suspected crimes.

In the European Union, the Anti-Money Laundering (AML) Directives regulate the financial system to prevent money laundering and terrorism financing. For example, financial institutions are required to implement Know-Your-Customer (KYC) processes, which include verifying the identity of their customers. Hence, the use of banking services cannot be anonymous. Since the 5th AML Directive [Eur18], cryptocurrency exchanges must also follow these regulations. Also in the European Union, the GDPR [Eur16, Article 23.1(d)] states that the user data privacy regulations can be restricted when law enforcement has a legitimate interest (e.g., for "the prevention, investigation, detection or prosecution of criminal offences"). Similarly, in the United States, 18 U.S. Code §2703 [US 86] allows law enforcement to obtain user data from companies to investigate online fraud, hate crimes, and other criminal activities.

Thus, even if unconditional anonymity is desired for many applications, it is not always fully feasible due to laws and regulations. Therefore, some kind of backdoor(s) to remove user anonymity in special cases is needed in these applications as well. In order to adequately prevent the misuse of these backdoors, we require that the anonymity revocation be *conditional*, where conditional means that law enforcement must have a legitimate reason. To further protect against abuse, we require that the use of a backdoor can be *audited* after the fact. Thus, to adequately protect user privacy while still allowing law enforcement access for legitimate reasons, we propose to use *auditable and conditional anonymity revocation*. The feasibility of this, without simply letting a TTP do all the work, is the second research goal of this thesis:

> ***Second Goal:***
> *This thesis aims to evaluate the feasibility of incorporating (unwanted but necessary) backdoors into otherwise anonymous systems in a way that ensures that the use of a backdoor is only possible under certain, predefined conditions and that the use of a backdoor can be audited afterwards.*

1.1. Contribution of the Thesis

We now elaborate on the contribution of this thesis to the two aforementioned goals.

On Building Privacy-Preserving Electronic Toll Collection Systems. Building privacy-preserving solutions for privacy-sensitive CPSs is challenging because one must simultaneously satisfy the requirements of "normal" privacy-preserving protocols—namely correctness, polynomial runtime, and provable security and privacy—as well as real-world requirements, such as "practical efficiency", handling aborts, faulty inputs, uncooperative users, and broken hardware. In order to develop techniques to overcome all these challenges of privacy-sensitive CPS, we take the example of privacy-preserving electronic toll collection (ETC) and develop a system specifically for this scenario.

In this thesis, we propose P4TC: a flexible security model and cryptographic protocol designed for privacy-preserving ETC in the dominant real-world setting, i.e., Dedicated Short Range Communication (DSRC) ETC. In DSRC ETC, vehicles are equipped with *on-board units (OBUs)* that perform all user-side computations. Tolls are charged each time a vehicle passes a *road-side unit (RSU)*, which are subsidiaries of the operator and calculate the toll charge together with the OBU based on several factors such as vehicle type, time of day, current congestion level, and so on. While DSCR-based ETC is the most common setting in currently deployed ETC systems (which often rely on user identification for payments), scientific ETC systems appear to favor the Global Navigation Satellite System (GNSS) setting[4] over the DSCR setting (see Section 3.1.1). To support an eventual real-world adoption, we decided to model P4TC in the more widely used DSRC setting. For the same reason, we also focused on efficiency, as P4TC should support open road tolling, where vehicles can pass the RSUs at full speed. Thus, a major challenge in designing P4TC was to combine provable security/privacy with practical efficiency and to address real-world issues such as broken OBUs, stolen RSUs, and uncooperative or malicious users. To the best of our knowledge, P4TC is the first privacy-preserving ETC system in the DSRC setting with a rigorous security model and proof and arguably the most comprehensive formal treatment of ETC security and privacy overall.

To build P4TC, we start with a payment system building block called black-box accumulators (BBA+) [Har+17]. BBA+ provides the core functionality of an unlinkable user wallet that maintains a balance. An operator can add and subtract values from the wallet balance and the use of an old wallet state is detected by a double-spending mechanism adopted from the e-cash literature. In addition to unlinkability of transactions, the system guarantees that a wallet can only be used by its legitimate owner and with its legitimate balance. However, BBA+ misses several features that are necessary for a practical ETC system. Therefore, P4TC also includes the following features:

(1) In open road tolling, there are no physical barriers to stop vehicles that fail to successfully communicate with the RSU they are passing. This can happen for a variety of reasons, ranging from defective OBU hardware to malicious users. In any case, this must be taken into account for a practical ETC system. For this purpose, P4TC assumes that there are *enforcement cameras* behind each RSU that take a photo of the vehicle (and thus the license plate) in case something goes wrong.[5]

[4] In GNSS-based ETC, each OBU is equipped with geolocation capabilities and collects location-time data or road segment prices to send to the ETC operator. Unpredictable spot checks are used to help ensure that users report this data honestly.

[5] Note that the cameras are only needed for open road tolling. If toll plazas are used, there may be physical barriers to prevent uncooperative vehicles from leaving.

(2) In the event that the enforcement cameras capture more than one vehicle in a single photo (or in the event that the camera was triggered by mistake), P4TC has a mechanism that allows users to provide proof that they have successfully communicated with the RSU in question and that they are not guilty of not paying tolls.

(3) In order for the operator to be able to deal with malicious users who are evading toll charges, or to deal with stolen OBUs, we introduce a blacklisting mechanism.

(4) To deal with the operator's lost revenue due to users refusing to pay their debts or due to stolen/broken OBUs, the operator can deanonymize all transactions made by a blacklisted wallet with the help of a TTP. Once all transactions made by a single user during a single billing period are deanonymized, the operator can calculate the total toll charge for that user and billing period.

(5) P4TC also adds user attributes to the wallet to allow for a more flexible pricing system. For example, the type of vehicle can affect the toll charge, or there could be discounts for elderly people.

(6) BBA+ assumes a single operator involved in all transactions. In P4TC, we introduce the RSUs as subsidiaries of the operator that handle communication on the road. Therefore, P4TC also includes mechanisms to mitigate the damage caused by corrupted RSUs.

To prove the security and privacy guarantees of the P4TC protocol, we model an idealized privacy-preserving ETC scheme as an ideal functionality in the *Universal Composability (UC) framework*. Our security proof then shows that the P4TC protocol is "as secure as" the ideal model, thus the protocol "UC-realizes" the ideal functionality. P4TC is one of the few systems that combines a complex, yet practical cryptosystem with a thorough UC-security analysis.

On Making Backdoor Usage Auditable. For many privacy-sensitive applications where unconditional anonymity would be ideal, anonymity revocation mechanisms are required (e.g., to deal with malicious users or to comply with laws). As a result, many systems have no effective privacy protection at all, or have backdoors, e.g., stored at the operator side of the system, that can be used to disclose any user's private information to law enforcement authorities. In our privacy-preserving ETC system P4TC, we have the problem that since we cannot proactively *prevent* fraudulent transactions (since offline transactions are required), we have to *detect* them after the fact (and subsequently punish the user). To solve this, P4TC has short-term user-specific backdoors (all accessible only by a trusted entity). The problem with such backdoors in general is that they also enable silent mass surveillance within the system *if* the entity managing the backdoors turns out to be untrustworthy after all. To prevent such misuse, some approaches have been suggested which limit possible abuse or ensure it can be detected. However, much of the previous work is limited to *proactive* surveillance, where the surveillance decision has to be made before the monitored transaction(s) take place, e.g., [BGK95; GGM16; KL95; KV03; PY00; YY98].

In order to allow more flexibility with respect to surveillance decisions, it would be more desirable to have *retrospective* surveillance, where the surveillance decision can also be made after the transaction(s) have taken place. For retrospective surveillance, there is currently only a small amount of related work (see Section 4.1.2). Of these, only the work by Green, Kaptchuk, and Van Laer [GKV21] satisfies most of our desired properties, but it is based on extractable witness encryption, a primitive for which it is currently unclear whether it can be instantiated.

In this thesis, we develop a building block for *auditable retrospective surveillance* that can be used to enhance existing applications with auditable surveillance capabilities. In order to prevent misuse of the system or even mass surveillance, backdoors are protected in multiple ways:

(1) Backdoors are *short-term* and *user-specific*.

(2) Backdoors are *shared* between trustworthy parties to avoid a single point of failure.

(3) Backdoor access is *conditional*.

(4) There are *audit trails* and *public statistics* for every (granted) backdoor request.

(5) Surveillance remains *silent*, i.e., users do not know that they are surveilled.

Concretely, we first present an abstract ideal functionality in the UC framework. This provides a basic target functionality to be realized, and serves as a separation between the low-level implementation of the auditability mechanism and the high-level decision to add auditability to a system. We then provide a protocol that UC-realizes this ideal functionality. The protocol uses several advanced cryptographic primitives, such as non-interactive zero-knowledge proofs of knowledge and threshold public key encryption, to achieve the same security and privacy guarantees as the ideal model.

To protect user privacy, the operator is only able to use a user's backdoor under certain conditions. More precisely, backdoor access in our system must be *lawful* to prevent arbitrary access. In our case, "lawful" corresponds to law enforcement agencies being in possession of a court-signed warrant for the user.

Decryption of (threshold-encrypted) user backdoors is handled by an anonymous, evolving committee. The committee can verify that the conditions for backdoor access are met, i.e., that law enforcement is in possession of a valid surveillance warrant (via a zero-knowledge proof). In addition, backdoor access leaves an audit trail on a ledger, allowing an auditor to retrospectively review surveillance decisions. The committee is cast in the YOSO (You-Only-Speak-Once) model [Gen+21]. In the YOSO model, protocols are executed by roles, with each role being allowed to send messages only once. Which party is playing a particular role is hidden until it sends its messages. This prevents targeted corruption of the parties that make up the current committee, which makes it easier to maintain an honest majority in the committee, and also allows to achieve security against mobile adaptive adversaries.

1.2. Other Results

This section contains a brief summary of other results that were obtained during this author's doctoral studies but are not included in this thesis. The works presented here all deal with different aspects of the topic of privacy-preserving protocols and thus complement the results of this thesis.

A complete list of publications by the author of this thesis can be found right before the table of contents. Note that the works [FMN16] and [Bro+18] from the complete list of publications are omitted in this section, as they evolved from this author's bachelor's and master's theses, respectively.

P6V2G: A Privacy-Preserving V2G Scheme for Two-Way Payments and Reputation. In [Sch+19] we build upon P4TC to realize P6V2G, a privacy-preserving payment and reward system for charging and discharging of electric vehicles in the smart grid scenario, which is another example for a privacy-sensitive CPS. P6V2G relies heavily on P4TC, but introduces some new features: It supports two-way payments (since electric vehicles can also be *dis*charged to supply the smart grid with energy), adds a reputation mechanism to rate the reliability of users, features multiple operators (corresponding to multiple electricity providers), and user accounts are designed to be portable. This means that the user's side is divided into two separate parties: user accounts, which contain the users' wallets, and electric vehicles, which are represented by their communication controllers. This allows for one person to own multiple vehicles with the same account, or for car-sharing concepts where multiple people use the same vehicle. As with P4TC, the security of P6V2G is proven in the UC framework.

(Schwerdt, Nagel, Fetzer, Gräf, Rupp – Energy Informatics 2019)

PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics. In [Fet+22] we propose PUBA, a generic building block that builds on top of BBA+ and supports storing more general information than just points in the user's wallet. In PUBA, users own so-called "logbooks", which store the user's accumulated transaction history and can also store multiple values (remember that in BBA+ only a single balance is stored). Again, users can participate in anonymous transactions, during which the logbook is updated accordingly, while the content itself remains hidden from the operator. In PUBA, the operator is also able to perform privacy-preserving analytics computations on the user logbooks, e.g., to perform market analyses. To keep these analytics privacy-preserving, Multi-Party Computation (MPC) is used to compute the analysis function over (multiple) user logbooks, while keeping the contents hidden from the operator and keeping the details of the analysis function hidden from the users. The security of PUBA is again proven in the UC framework.

(Fetzer, Keller, Maier, Raiber, Rupp, Schwerdt – PoPETs 2022)

POBA: Privacy-Preserving Operator-Side Bookkeeping and Analytics. In [Fau+25] we present POBA, which is also a generic building block for bookkeeping and analytics, but is designed for multiple operators. The main difference to PUBA is that POBA moves the data storage from the user-side to the operator-side. This firstly reduces the computation and communication effort for users, who are assumed to use smartphones as user devices. Also, in PUBA, each time an analytics function is to be computed that involves a user's logbook, that user must be available to participate in the computation. In POBA, users only need to interact with an operator when collecting new data, so operators can evaluate analytic functions whenever they agree to do so.

(Faut, Fetzer, Müller-Quade, Raiber, Rupp – IACR Communications in Cryptology 2025)

Differential Privacy to the Rescue? On Obfuscating Tolls in Privacy-Preserving ETC Systems. Privacy-preserving ETC systems are all based on the assumption that the operator, who learns only the sum of accumulated tolls for a user, can not infer anything about the individual toll station visits. Since this corresponds to the NP-complete subset sum problem, this is asymptotically true. However, other works have shown that this may not be the case for concrete, real-world scenarios, such as the ETC system currently used in Brisbane, Australia. In [Jol+25] we investigate whether differential privacy approaches can be used to blur the final sum in such a way that the operator can no longer guess the individual toll station visits with a high probability.

(Adavoudi Jolfaei, Fetzer, Rupp, Schiffner – In Submission)

1.3. Structure of the Thesis

In Chapter 1 we gave an introduction into the considered scenarios and stated the contribution of this thesis. Next, in Chapter 2, we present the cryptographic foundations needed for the remainder for this thesis.

The main part of this thesis starts in Chapter 3, where the privacy-preserving electronic toll collection system P4TC is presented. Note that the security proof is postponed to Appendix A. Afterwards we tackle how to build a universally composable auditable surveillance framework in Chapter 4. The security proofs here are again postponed to Appendix B.

2. Preliminaries

In this chapter we introduce some general notation and definitions, as well as the algebraic setting and building blocks we will make use of. In particular, the latter includes encryption (SKE, PKE, TPKE), commitments, digital signatures, NIZKPoKs, and pseudo-random functions.

The definitions in this chapter are taken (mostly verbatim) from [Fet+18; Fet+20] and [Fet+23a; Fet+23b], with additional details from [KL14].

General Notation. We make use of the following notation, similar to [KL14]:

- We use $\coloneqq$ to refer to deterministic assignment.
- If $\mathcal{X}$ is a set, then $x \leftarrow \mathcal{X}$ denotes that x is chosen uniformly from $\mathcal{X}$. Sometimes, we also write this as $x \overset{R}{\leftarrow} \mathcal{X}$ to emphasize the *random* sampling.
- For a (probabilistic) algorithm A, we write $y \leftarrow \mathsf{A}(x)$ for running algorithm A on input x (with uniformly chosen random coins) and getting y as result. Sometimes we want to make the random coins r explicit, in that case we write $y \coloneqq \mathsf{A}(x; r)$.
- We denote with $\lambda \in \mathbb{N}$ the security parameter. We assume it is implicitly given to all cryptographic algorithms in unary representation, i.e., 1^λ.
- If the running time of A is polynomial in λ, then A is called probabilistic polynomial-time (PPT).
- $\{0,1\}^n$ is the set of all bit-strings of length n and $\{0,1\}^*$ is the set of all finite bit-strings.
- 0^n denotes the string consisting of n zeroes and 1^n denotes the string consisting of n ones.
- $\Pr[X]$ denotes the probability of event X.

Negligible Function. We use the following definition of a negligible function from [Ps10].

Definition 2.1 (Negligible Function)**.** A function $\mathsf{negl}(n)$ is *negligible* if for every c, there exists some n_0 such that for all $n > n_0$

$$\mathsf{negl}(n) < \frac{1}{n^c}.$$

Intuitively, a negligible function is asymptotically smaller than the inverse of any fixed polynomial. Examples of negligible functions include 2^{-n} and $2^{-\log\log n}$. We say that a function $t(n)$ is *non-negligible* if there exists some constant c such that for infinitely many points $\{n_0, n_1, \ldots\}$ $t(n_i) > n_i^c$. This notion becomes important in proofs that work by contradiction.

Bilinear Group Setting. Our protocol instantiations in Chapter 3 are based on an asymmetric bilinear group setting $\mathsf{gp} \coloneqq (G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2)$.

Definition 2.2 (Prime-Order Bilinear Group Generator)**.** A *prime-order bilinear group generator* is a PPT algorithm SetupGrp that on input of a security parameter 1^λ outputs a tuple of the form

$$\mathsf{gp} \coloneqq (G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^\lambda)$$

where G_1, G_2, G_T are descriptions of cyclic groups of prime order q, $\log \mathsf{q} = \Theta(\lambda)$, g_1 is a generator of G_1, g_2 is a generator of G_2, and $e\colon\ G_1 \times G_2 \longrightarrow G_T$ is a map (aka pairing) which satisfies the following properties:

- *Efficiency:* e is efficiently computable.
- *Bilinearity:* $\forall a \in G_1, b \in G_2, x, y \in \mathbb{Z}_\mathsf{q}\colon e\left(a^x, b^y\right) = e\left(a, b\right)^{xy}$.
- *Non-Degeneracy:* $e\left(g_1, g_2\right)$ generates G_T.

Like [GPS08], we differentiate between three types of pairings:

Type 1: $G_1 = G_2$

Type 2: $G_1 \neq G_2$, but there is an efficiently computable homomorphism $\phi : G_2 \longrightarrow G_1$

Type 3: $G_1 \neq G_2$, and there are no efficiently computable homomorphisms between G_1 and G_2

Type 1 is also called a *symmetric* pairing and types 2 and 3 are also called *asymmetric* pairings. In the remainder of this thesis, we always consider type 3 pairings, unless specified otherwise.

2.1. Cryptographic Assumptions

Our construction in Chapter 3 relies on the co-CDH assumption for identification, and the security of the building blocks in asymmetric bilinear groups. For our special instantiation of the building blocks (cp. Section 3.4.4), security holds under the SXDH and co-DLIN assumption. The former implies the co-CDH assumption.

The SXDH assumption essentially asserts that the DDH assumption holds in both source groups G_1 and G_2 of the bilinear map and is formally defined as:

Definition 2.3 (DDH Assumption & SXDH Assumption)**.**

(1) We say that the *DDH assumption* holds with respect to SetupGrp over G_i if $\mathsf{Succs}^{\mathsf{DDH}}_{\mathsf{SetupGrp},i,\mathcal{A}}(\lambda)$ is defined by

$$\Pr\left[\ b = b' \ \middle|\ \begin{array}{c} \mathsf{gp} \coloneqq (G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ x, y, z \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathsf{q}; h_0 \coloneqq g_i^{xy}; h_1 \coloneqq g_i^z \\ b \overset{\mathrm{R}}{\leftarrow} \{0, 1\} \\ b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{gp}, g_i^x, g_i^y, h_b) \end{array} \right]$$

and the advantage

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathsf{SetupGrp},i,\mathcal{A}}(\lambda) \coloneqq \left| \mathsf{Succs}^{\mathsf{DDH}}_{\mathsf{SetupGrp},i,\mathcal{A}}(\lambda) - \frac{1}{2} \right|$$

is a negligible function in λ for all PPT algorithms $\mathcal{A}$.

(2) We say that the *SXDH assumption* holds with respect to SetupGrp if the above holds for both $i = 1$ and $i = 2$.

The co-CDH assumption is defined as follows:

Definition 2.4 (Co-CDH Assumption)**.** We say that the *co-CDH assumption* holds with respect to SetupGrp if the advantage $\mathsf{Adv}^{\mathsf{co\text{-}CDH}}_{\mathsf{SetupGrp},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[a = g_2^x \;\middle|\; \begin{array}{c} \mathsf{gp} \coloneqq (G_1, G_2, G_\mathrm{T}, e, \mathrm{q}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ x \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{q} \\ a \leftarrow \mathcal{A}(1^\lambda, \mathsf{gp}, g_1^x) \end{array} \right]$$

is a negligible function in λ for all PPT algorithms $\mathcal{A}$.

The co-DLIN assumption is defined as follows:

Definition 2.5 (Co-DLIN Assumption)**.** We say that the *co-DLIN assumption* holds with respect to SetupGrp if the advantage $\mathsf{Adv}^{\mathsf{co\text{-}DLIN}}_{\mathsf{SetupGrp},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{gp} \coloneqq (G_1, G_2, G_\mathrm{T}, e, \mathrm{q}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ \alpha, \beta, \gamma \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{q} \\ b \overset{\mathrm{R}}{\leftarrow} \{0,1\} \\ \check{h}_1 \coloneqq g_1^\alpha, \check{h}_2 \coloneqq g_1^\beta, \check{h}_3 \coloneqq g_1^{\alpha+\beta+b\gamma} \\ \hat{h}_1 \coloneqq g_2^\alpha, \hat{h}_2 \coloneqq g_2^\beta, \hat{h}_3 \coloneqq g_2^{\alpha+\beta+b\gamma} \\ b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{gp}, \check{h}_1, \check{h}_2, \check{h}_3, \hat{h}_1, \hat{h}_2, \hat{h}_3) \end{array} \right]$$

is a negligible function in λ for all PPT algorithms $\mathcal{A}$.

2.2. Encryption

Encryption, the oldest form of cryptography, is traditionally used when two parties want to communicate *confidentially*. We say that the sender *encrypts* the message and sends the resulting *ciphertext* over a potentially insecure channel to the receiver, who *decrypts* the ciphertext to obtain the message. Security informally means that the ciphertext should not give other parties any information about the message.

For this thesis, we cover multiple types of encryption:

- In *symmetric encryption*, the parties must exchange a secret key before communicating. This key is then used for both encryption and decryption.
- In *asymmetric encryption*, the receiver generates a key pair consisting of a public key and a secret key. The public key (which can be shared with the general public) is used for encryption, while the secret is used for decryption and should only be known by the receiver.
- *Threshold encryption* is a special form of asymmetric encryption in which the decryption capability is distributed among multiple parties and a minimum number of parties (the threshold) must cooperate to decrypt the ciphertext.

In the following we define all three types along with their standard security notions.

2.2.1. Symmetric Encryption

We use standard definitions for symmetric encryption schemes and corresponding security notions.

Definition 2.6 (Symmetric Encryption Scheme). A *symmetric encryption scheme* (often also called *secret-key encryption (SKE) scheme*) $\mathsf{SKE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three PPT algorithms:

- $\kappa \leftarrow \mathsf{Gen}(1^\lambda)$ takes as input 1^λ (i.e., the security parameter written in unary) and outputs a (random) key κ.
- $c \leftarrow \mathsf{Enc}(\kappa, m)$ takes a key κ and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext c.
- $m/\bot = \mathsf{Dec}(\kappa, c)$ takes a key κ and a ciphertext c and outputs a plaintext message m or an error. We denote a generic error by the symbol $\bot$. We assume that Dec is deterministic.

We say that SKE is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\ \forall \kappa \leftarrow \mathsf{Gen}(1^\lambda)\ \forall m \in \mathcal{M}\ \forall c \leftarrow \mathsf{Enc}(\kappa, m)\ :\ \mathsf{Dec}(\kappa, c) = m$$

We now give the usual security definitions for SKE schemes, namely IND-CPA-security and IND-CCA2-security.[1]

Definition 2.7 (IND-CPA-Security for SKE Schemes). A symmetric encryption scheme SKE is IND-CPA-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}sym}}_{\mathsf{SKE},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \kappa \leftarrow \mathsf{Gen}(1^\lambda) \\ (state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot)}(1^\lambda) \\ b \xleftarrow{R} \{0,1\} \\ c^* \leftarrow \mathsf{Enc}(\kappa, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot)}(state, c^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ, where $|m_0| = |m_1|$, and $\mathsf{Enc}(\kappa, \cdot)$ denotes an oracle that returns $\mathsf{Enc}(\kappa, m)$ for a m chosen by the adversary.

Definition 2.8 (IND-CCA2-Security for SKE Schemes). A symmetric encryption scheme SKE is IND-CCA2-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2\text{-}sym}}_{\mathsf{SKE},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \kappa \leftarrow \mathsf{Gen}(1^\lambda) \\ (state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot),\mathsf{Dec}(\kappa,\cdot)}(1^\lambda) \\ b \xleftarrow{R} \{0,1\} \\ c^* \leftarrow \mathsf{Enc}(\kappa, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot),\mathsf{Dec}'(\kappa,\cdot)}(state, c^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ, where $|m_0| = |m_1|$, $\mathsf{Enc}(\kappa, \cdot)$ and $\mathsf{Dec}(\kappa, \cdot)$ denote oracles that return $\mathsf{Enc}(\kappa, m)$ and $\mathsf{Dec}(\kappa, c)$ for a m or c chosen by the adversary, and $\mathsf{Dec}'(\kappa, \cdot)$ is the same as $\mathsf{Dec}(\kappa, \cdot)$, except that it returns $\bot$ on input c^*.

[1] Note that IND-CCA2-security is often just called IND-CCA-security.

We now define a multi-message version of IND-CCA2-security. It is a well-known fact that IND-CCA2-security in the multi-message setting is equivalent to standard IND-CCA2-security. This can be shown via a standard hybrid argument.

Definition 2.9 (Multi-Message IND-CCA2-Security for SKE Schemes). A symmetric encryption scheme SKE is *multi-message* IND-CCA2-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{mm\text{-}IND\text{-}CCA2\text{-}sym}}_{\mathsf{SKE},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \kappa \leftarrow \mathsf{Gen}(1^\lambda) \\ (state, j, \mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot),\mathsf{Dec}(\kappa,\cdot)}(1^\lambda) \\ b \overset{\text{R}}{\leftarrow} \{0,1\} \\ \mathbf{c}^* \leftarrow (\mathsf{Enc}(\kappa, m_{b,1}), \ldots, \mathsf{Enc}(\kappa, m_{b,j})) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\kappa,\cdot),\mathsf{Dec}'(\kappa,\cdot)}(state, \mathbf{c}^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ, where $\mathbf{m}_0$, $\mathbf{m}_1$ are two vectors of $j \in \mathbb{N}$ bitstrings each such that for all $1 \leq i \leq j$: $|m_{0,i}| = |m_{1,i}|$, $\mathsf{Enc}(\kappa, \cdot)$ and $\mathsf{Dec}(\kappa, \cdot)$ denote oracles that return $\mathsf{Enc}(\kappa, m)$ and $\mathsf{Dec}(\kappa, c)$ for a m or c chosen by the adversary, and $\mathsf{Dec}'(\kappa, \cdot)$ is the same as $\mathsf{Dec}(\kappa, \cdot)$, except that it returns $\bot$ on input of any c_i^* that is contained in $\mathbf{c}^*$.

Definition 2.10 (Key-Committing for SKE Schemes). Let SKE be a symmetric encryption scheme. We say SKE is *(secret-)key-committing*, if it is hard to find a tuple (κ_1, κ_2, c) such that $\kappa_1 \neq \kappa_2$ but $\mathsf{Dec}(\kappa_1, c) \neq \bot$ and $\mathsf{Dec}(\kappa_2, c) \neq \bot$.

2.2.2. Asymmetric Encryption

We use the standard definitions for asymmetric encryption schemes and corresponding security notions.

Definition 2.11 (Asymmetric Encryption Scheme). An *asymmetric encryption scheme* (often also called *public-key encryption (PKE) scheme*) $\mathsf{PKE} \coloneqq (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three PPT algorithms:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ takes as input 1^λ (i.e., the security parameter written in unary) and outputs a pair of public key pk and secret key sk.
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ takes as input a public key pk, a message m in the message space $\mathcal{M}$, and outputs a ciphertext c. Note that the message space may depend on pk.
- $m/\bot = \mathsf{Dec}(\mathsf{sk}, c)$ takes as input a secret key sk, a ciphertext c, and outputs a message $m \in \mathcal{M}$ or $\bot$. We assume that Dec is deterministic.

We say that PKE is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\; \forall (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)\; \forall m \in \mathcal{M}\; \forall c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \;:\; \mathsf{Dec}(\mathsf{sk}, c) = m$$

Definition 2.12 (IND-CPA-Security for PKE Schemes). An asymmetric encryption scheme PKE is IND-CPA-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}asym}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (state, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda) \\ b \overset{\text{R}}{\leftarrow} \{0,1\} \\ c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}(state, c^*) \end{array} \right] - \frac{1}{2} \right|$$

```
$\mathsf{Exp}^{\mathsf{RIND-SO}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$
1 : $b \leftarrow \{0,1\}$
2 : $(\mathsf{pk},\mathsf{sk}) \coloneqq (\mathsf{pk}_i,\mathsf{sk}_i) \leftarrow (\mathsf{Gen}(1^\lambda))_{i\in[n]}$
3 : $m \coloneqq (m_i)_{i\in[n]} \leftarrow \mathcal{D}$
4 : $c \coloneqq (c_i)_{i\in[n]} \leftarrow (\mathsf{Enc}(\mathsf{pk},m_i))_{i\in[n]}$
5 : $state \leftarrow \mathcal{A}^{C(\cdot)}(c)$
6 : $m' \leftarrow \mathsf{Resample}_{\mathcal{D}}(m_{\mathcal{I}})$
7 : $m^* = m$ if $b = 0$, else $m^* = m'$
8 : $b' \leftarrow \mathcal{A}(m^*, state)$
9 : return $b = b'$
```

```
Corruption Oracle $C(i)$
1 : if $i \notin [n]$ return $\perp$
2 : $\mathcal{I} \coloneqq \mathcal{I} \cup \{i\}$
3 : return $\mathsf{sk}_i$
```

Figure 2.1.: Experiment for receiver selective opening security

is negligible in λ, where $|m_0| = |m_1|$.

Definition 2.13 (IND-CCA2-Security for PKE Schemes)**.** An asymmetric encryption scheme PKE is IND-CCA2-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2\text{-}asym}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot)}(1^\lambda) \\ b \xleftarrow{\mathrm{R}} \{0,1\} \\ c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Dec}'(\mathsf{sk},\cdot)}(state, c^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ, where $|m_0| = |m_1|$, $\mathsf{Dec}(\mathsf{sk},\cdot)$ denotes an oracle that returns $\mathsf{Dec}(\mathsf{sk},c)$ for a c chosen by the adversary, and $\mathsf{Dec}'(\mathsf{sk},\cdot)$ is the same as $\mathsf{Dec}(\mathsf{sk},\cdot)$, except that it returns $\perp$ on input c^*.

Definition 2.14 (Key-Committing for PKE Schemes)**.** Let PKE be a perfectly correct asymmetric encryption scheme. We say PKE is *(public-)key-committing*, if it is hard to find a tuple $((\mathsf{pk}_1,\mathsf{sk}_1), (\mathsf{pk}_2,\mathsf{sk}_2), c)$ such that $(\mathsf{pk}_i,\mathsf{sk}_i) \in \mathrm{Supp}(\mathsf{Gen}(1^\lambda))$ for $i \in \{1,2\}$ and $\mathsf{pk}_1 \neq \mathsf{pk}_2$ but $\mathsf{Dec}(\mathsf{sk}_1,c) \neq \perp$ and $\mathsf{Dec}(\mathsf{sk}_2,c) \neq \perp$.

We recall *receiver selective opening (RIND-SO) security* from Hazay et al. [HPW15].

Definition 2.15 (RIND-SO-Security for PKE Schemes)**.** An asymmetric encryption scheme scheme PKE is called RIND-SO-*secure* if for all PPT adversaries $\mathcal{A}$ and all efficiently resamplable distributions $\mathcal{D}$ we have that the advantage

$$\mathsf{Adv}^{\mathsf{RIND-SO}}_{\mathsf{PKE},\mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathsf{Exp}^{\mathsf{RIND-SO}}_{\mathsf{PKE},\mathcal{A}}(\lambda)\right] - \frac{1}{2} \right|$$

is negligible in λ, where $\mathsf{Exp}^{\mathsf{RIND-SO}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ is defined in Fig. 2.1.

In Chapter 4 we will need two different asymmetric encryption schemes: (1) One that is correct and IND-CPA-secure and (2) one that is correct and RIND-SO-secure.

In Chapter 3 we will need a correct and IND-CCA2-secure asymmetric encryption scheme, but we will also need a so-called *structure-preserving* asymmetric encryption scheme. Structure-preserving asymmetric encryption schemes were introduced by [Cam+11] and are defined as follows.

Definition 2.16 (Structure-Preserving Asymmetric Encryption (from [Cam+11, Def. 1]))**.** An asymmetric encryption scheme is said to be *structure-preserving* if

(1) its public keys, messages, and ciphertexts consist entirely of elements of a bilinear group,

(2) its encryption and decryption algorithm perform only group and bilinear map operations, and

(3) it is IND-CCA2-secure.

Therefore, we now syntactically define *group-based* asymmetric encryption schemes, along with the corresponding IND-CCA2-security definition. The only difference to standard asymmetric encryption schemes is that group-based asymmetric encryption schemes have an additional SetupGrp algorithm that sets up a bilinear group and outputs group parameters gp, which are (implicitly) given to the other algorithms as input.

Definition 2.17 (Group-Based Asymmetric Encryption Scheme and IND-CCA2-Security)**.** A *group-based asymmetric encryption scheme* $\mathsf{E} := (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of four PPT algorithms:

- $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)$ takes as input a security parameter 1^λ and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp})$ outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of keys, where pk is the (public) encryption key and sk is the (secret) decryption key.
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ takes a key pk and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext c.
- $m/\bot = \mathsf{Dec}(\mathsf{sk}, c)$ takes a key sk and a ciphertext c and outputs a plaintext message m or $\bot$. We assume that Dec is deterministic.

We say that E is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\ \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)\ \forall (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp})\ \forall m \in \mathcal{M}\ \forall c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\ :\\ \mathsf{Dec}(\mathsf{sk}, c) = m$$

A group-based asymmetric encryption scheme E is IND-CCA2-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2\text{-}asym}}_{\mathsf{E},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (\mathit{state}, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{pk}) \\ b \xleftarrow{R} \{0,1\} \\ c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{Dec}'(\mathsf{sk},\cdot)}(\mathit{state}, c^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ, where $|m_0| = |m_1|$, $\mathsf{Dec}(\mathsf{sk}, \cdot)$ is an oracle that gets a ciphertext c from the adversary and returns $\mathsf{Dec}(\mathsf{sk}, c)$ and $\mathsf{Dec}'(\mathsf{sk}, \cdot)$ is the same, except that it returns $\bot$ on input c^*.

2.2.3. Threshold PKE

Definition 2.18 (Threshold PKE)**.** A (t,n)*-threshold PKE (TPKE)* scheme $\mathsf{TPKE} \coloneqq (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TDec}, \mathsf{ShareVer}, \mathsf{Combine}, \mathsf{Sk2Pk})$ consists of seven PPT algorithms:

- $(\mathsf{pk}, \mathsf{sk}, \{\mathsf{vk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\lambda)$ outputs a public key pk, secret key sk and sets $\{\mathsf{vk}_i\}_{i\in[n]}$ of verification keys and $\{\mathsf{sk}_i\}_{i\in[n]}$ of secret key shares.
- $ct \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ takes a public key pk, a message m in the message space $\mathcal{M}$, and outputs a ciphertext ct.
- $m \coloneqq \mathsf{Dec}(\mathsf{sk}, ct)$ takes a secret key sk, a ciphertext ct and outputs a message $m \in \mathcal{M}$.
- $ct_i \coloneqq \mathsf{TDec}(\mathsf{sk}_i, ct)$ takes a secret key share sk_i, a ciphertext ct, and outputs a decryption share ct_i.
- $1/0 = \mathsf{ShareVer}(ct, \mathsf{vk}_i, ct_i)$ takes a ciphertext ct, a verification key vk_i and a decryption share ct_i, and outputs either 1 or 0. If the output is 1, ct_i is called a *valid* decryption share.
- $m \coloneqq \mathsf{Combine}(ct, T)$ takes a ciphertext ct and a set of valid decryption shares T such that $|T| = t+1$, and outputs a message $m \in \mathcal{M}$.
- $\mathsf{pk} \coloneqq \mathsf{Sk2Pk}(\mathsf{sk})$ takes a secret key sk (or a secret key share sk_i) as input and outputs the corresponding public key pk (or the corresponding verification key vk_i)

We assume that Dec, TDec, $\mathsf{ShareVer}$, $\mathsf{Combine}$ and $\mathsf{Sk2Pk}$ are deterministic.

We require *correctness*, i.e., the following two conditions to hold for all $(\mathsf{pk}, \{\mathsf{vk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\lambda)$:

(1) $\forall m \in \mathcal{M}, ct \leftarrow \mathsf{Enc}(\mathsf{pk}, m), i \in [n]$ it holds that $\mathsf{ShareVer}(ct, \mathsf{vk}_i, \mathsf{TDec}(\mathsf{sk}_i, ct)) = 1$.

(2) $\forall m \in \mathcal{M}, ct \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ and any set $T \coloneqq (ct_1, \ldots, ct_{t+1})$ of valid decryption shares $ct_i \coloneqq \mathsf{TDec}(\mathsf{sk}_i, ct)$ for $t+1$ distinct secret key shares sk_i, it holds that $\mathsf{Combine}(ct, T) = m = \mathsf{Dec}(\mathsf{sk}, ct)$.

In particular, later in Chapter 4 we will in the ideal world make use of Dec as a shorthand for TDec plus $\mathsf{Combine}$.

A TPKE scheme is called *simulatable* if additionally an efficient simulation algorithm $\mathsf{SimTDec}(\mathsf{pk}, ct, m, \{ct_k\}_{k\in\mathcal{B}:|\mathcal{B}|\le t})$ exists that takes a public key, ciphertext, target message and up to t decryption shares of corrupted parties as input and outputs decryption shares $\{ct_i\}_{i\in[n]\setminus\mathcal{B}}$ for the honest parties that cause $\mathsf{Combine}$ to output the desired message m and for which $\mathsf{ShareVer}$ outputs 1.

A TPKE scheme is called *re-randomizable* if it also has a PPT algorithm $\mathsf{Rand}(ct)$ such that $\forall(\mathsf{pk}, \{\mathsf{vk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Gen}(1^\lambda)$:

- $\mathsf{Rand}(ct)$ takes a ciphertext ct, and outputs a new ciphertext ct'.
- $\forall m \in \mathcal{M} : \mathsf{Rand}(\mathsf{Enc}(\mathsf{pk}, m)) \approx \mathsf{Enc}(\mathsf{pk}, m)$

A TPKE scheme is called *binding* if it holds that $\forall$ PPT $\mathcal{A}$: $\Pr[(\mathsf{sk}, \mathsf{sk}', ct) \leftarrow \mathcal{A}(1^\lambda) : \mathsf{Sk2Pk}(\mathsf{sk}) = \mathsf{Sk2Pk}(\mathsf{sk}') \wedge \mathsf{Dec}(\mathsf{sk}, ct) \neq \mathsf{Dec}(\mathsf{sk}', ct)] \neq \mathsf{negl}(\lambda)$

IND-CPA-security for a TPKE scheme is defined as usual, except that the adversary can choose up to t key shares to receive.

In Chapter 4 we will need a (t, n)-TPKE scheme that is correct, IND-CPA-secure, simulatable re-randomizable and binding.

2.3. Commitments

A commitment scheme allows a user to commit to a message m and publish the result, called commitment *com*, in a way that m is hidden from others, but also the user cannot claim a different m afterwards when he opens *com*.

Definition 2.19 (Commitment Scheme)**.** A (non-interactive) *commitment scheme* $\mathsf{COM} := (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ consists of the following three PPT algorithms:

- $\mathsf{crs_{com}} \leftarrow \mathsf{Setup}(1^\lambda)$ takes 1^λ as input and outputs public parameters $\mathsf{crs_{com}}$.
- $(com, decom) \leftarrow \mathsf{Com}(\mathsf{crs_{com}}, m)$ takes as input parameters $\mathsf{crs_{com}}$ and a message $m \in \mathcal{M}$ and outputs a commitment *com* to m and some decommitment value *decom*.
- $1/0 = \mathsf{Open}(\mathsf{crs_{com}}, com, decom, m)$ takes as input parameters $\mathsf{crs_{com}}$, a commitment *com*, a decommitment value *decom* and a message m. It returns either 0 or 1. (Intuitively, returning 1 means that *com* is a valid commitment that can be opened to m with *decom*. Returning 0 means it is invalid.) We assume that Open is deterministic.

We say that COM is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\ \forall \mathsf{crs_{com}} \leftarrow \mathsf{Setup}(1^\lambda)\ \forall m \in \mathcal{M}\ \forall (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs_{com}}, m)\ :\\ \mathsf{Open}(\mathsf{crs_{com}}, com, decom, m) = 1$$

An *extractable* commitment scheme has the following additional algorithms:

- $(\mathsf{crs^{ext}_{com}}, \mathsf{td^{ext}_{com}}) \leftarrow \mathsf{ExtSetup}(1^\lambda)$
- $m \leftarrow \mathsf{Extract}(\mathsf{td^{ext}_{com}}, com)$

An *equivocable* commitment scheme has the following additional algorithms:

- $(\mathsf{crs^{eq}_{com}}, \mathsf{td^{eq}_{com}}) \leftarrow \mathsf{SimSetup}(1^\lambda)$
- $(\widetilde{com}, eq) \leftarrow \mathsf{SimCom}(\mathsf{td^{eq}_{com}})$
- $decom \leftarrow \mathsf{Equiv}(\mathsf{td^{eq}_{com}}, m, eq)$

Definition 2.20 (Hiding Commitment Scheme)**.** A commitment scheme COM is (computationally) *hiding* if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{crs_{com}} \leftarrow \mathsf{Setup}(1^\lambda) \\ (m_0, m_1, state) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs_{com}}) \\ b \overset{R}{\leftarrow} \{0,1\} \\ (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs_{com}}, m_b) \\ b' \leftarrow \mathcal{A}(com, state) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ.

The scheme is called *statistically hiding* if $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is negligible for an unbounded adversary $\mathcal{A}$. The scheme is called *perfectly hiding* if $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is zero for an unbounded adversary $\mathcal{A}$.

Definition 2.21 (Binding Commitment Scheme). A commitment scheme COM is (computationally) *binding* if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[\begin{array}{c} \mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom, m) = 1 \\ \wedge \\ \mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom', m') = 1 \end{array}\middle|\begin{array}{c} \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(1^\lambda) \\ (com, m, decom, m', decom') \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}_{\mathsf{com}}) \\ m \neq m' \end{array}\right]$$

is negligible in λ.

The scheme is called *statistically binding* if $\mathsf{Adv}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is negligible for an unbounded adversary $\mathcal{A}$. The scheme is called *perfectly binding* if $\mathsf{Adv}^{\mathsf{Binding}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ is zero for an unbounded adversary $\mathcal{A}$.

A commitment scheme that is perfectly hiding (resp. hiding) is also statistically hiding (resp. hiding), and a commitment scheme that is statistically hiding (resp. hiding) is also computationally hiding (resp. hiding). Note that a commitment scheme can not be statistically hiding and statistically binding at the same time (see, e.g., [Dam99]).

Definition 2.22 (Equivocal Commitment scheme). A commitment scheme COM is *equivocal* if there exist PPT algorithms SimSetup, SimCom and Equiv such that for all PPT adversaries $\mathcal{A}$ we have that the advantage $\mathsf{Adv}^{\mathsf{SimSetup}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\left|\begin{array}{r} \Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}}) \ \middle|\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(1^\lambda)\ \right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}) \ \middle|\ (\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\lambda)\ \right] \end{array}\right|$$

is negligible in λ and we have that the advantage $\mathsf{Adv}^{\mathsf{Equiv}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\left|\begin{array}{r} \Pr\left[\ 1 \leftarrow \mathcal{A}\left(\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}, m, com, decom\right) \ \middle|\ \begin{array}{c} (\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\lambda), \\ m \leftarrow \mathcal{M}, \\ (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, m) \end{array}\right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}\left(\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}, m, \widetilde{com}, decom\right) \ \middle|\ \begin{array}{c} (\mathsf{crs}^{\mathsf{eq}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\lambda), \\ (\widetilde{com}, eq) \leftarrow \mathsf{SimCom}(\mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}), \\ m \leftarrow \mathcal{M}, \\ decom \leftarrow \mathsf{Equiv}(\mathsf{td}^{\mathsf{eq}}_{\mathsf{com}}, m, eq) \end{array}\right] \end{array}\right|$$

is zero.

Definition 2.23 (Extractable Commitment Scheme). A commitment scheme is *extractable* if there exist PPT algorithms ExtSetup and Extract such that for all PPT adversaries $\mathcal{A}$ we have that the advantage $\mathsf{Adv}^{\mathsf{ExtSetup}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\left|\begin{array}{r} \Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}}) \ \middle|\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(1^\lambda)\ \right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{crs}^{\mathsf{ext}}_{\mathsf{com}}) \ \middle|\ (\mathsf{crs}^{\mathsf{ext}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{ext}}_{\mathsf{com}}) \leftarrow \mathsf{ExtSetup}(1^\lambda)\ \right] \end{array}\right|$$

is negligible in λ and we have that the advantage $\mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{COM},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[\ \mathsf{Extract}(\mathsf{td}^{\mathsf{ext}}_{\mathsf{com}}, com) \neq m \ \middle|\ \begin{array}{c} (\mathsf{crs}^{\mathsf{ext}}_{\mathsf{com}}, \mathsf{td}^{\mathsf{ext}}_{\mathsf{com}}) \leftarrow \mathsf{ExtSetup}(1^{\lambda}), \\ com \leftarrow \mathcal{A}(\mathsf{crs}^{\mathsf{ext}}_{\mathsf{com}}), \\ \exists!\, m \in \mathcal{M}, r : com \leftarrow \mathsf{Com}(\mathsf{crs}^{\mathsf{ext}}_{\mathsf{com}}, m; r) \end{array} \right]$$

is zero.

In Chapter 4 we will use a correct, (computationally) hiding, (statistically) binding, and (dual-mode) extractable and equivocable commitment scheme. In Chapter 3 we will use the (dual-mode) equivocal and extractable commitment scheme from Groth and Sahai [GS08], which is also correct, as well as hiding, and binding (depending on the mode one them of holds perfectly while the other one holds computationally).

For Chapter 3 we additionally need a (group-based) F_{gp}-binding commitment scheme, which we will now introduce, along with all needed security definitions. Informally, a commitment scheme is called an *F_{gp}-binding commitment scheme* for a bijective function F_{gp} on the message space, if one commits to a message m but opens the commitment using $F_{\mathsf{gp}}(m)$. We call the codomain of F_{gp} the *implicit message space.*

Definition 2.24 ((Group-Based) F_{gp}-Binding Commitment Scheme)**.** A *F_{gp}-binding commitment scheme* $\mathsf{C} \coloneqq (\mathsf{SetupGrp}, \mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ consists of four PPT algorithms:

- $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^{\lambda})$ takes as input a security parameter 1^{λ} and outputs public parameters gp. These parameters also define a message space $\mathcal{M}$, an implicit message space $\mathcal{M}'$ and a function $F_{\mathsf{gp}} : \mathcal{M} \longrightarrow \mathcal{M}'$ mapping a message to its implicit representation. We assume that gp is given as implicit input to all algorithms.
- $\mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp})$ takes gp as input and outputs public parameters $\mathsf{crs}_{\mathsf{com}}$.
- $(com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m)$ takes as input parameters $\mathsf{crs}_{\mathsf{com}}$ and a message $m \in \mathcal{M}$ and outputs a commitment *com* to m and some decommitment value *decom*.
- $1/0 = \mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom, M)$ takes as input parameters $\mathsf{crs}_{\mathsf{com}}$, a commitment *com*, a decommitment value *decom* and an implicit message $M \in \mathcal{M}'$. It returns either 1 or 0. (Intuitively, returning 1 means that *com* is a valid commitment that can be opened to M with *decom*. Returning 0 means it is invalid.) We assume that Open is deterministic.

We say that C is *correct* if the following holds:

$$\begin{gathered} \forall \lambda \in \mathbb{N}\ \forall \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^{\lambda})\ \forall \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp})\ \forall m \in \mathcal{M} \\ \forall (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m)\ :\ \mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom, F(m)) = 1 \end{gathered}$$

We say that C is a (computationally) *hiding*, *F_{gp}-binding*, *equivocal*, *extractable* commitment scheme if it has the following properties:

(1) *Hiding:* For all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{C},\mathcal{A}}(\lambda)$ defined by

$$\left|\ \Pr\left[\ b = b' \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^{\lambda}) \\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp}) \\ (m_0, m_1, state) \leftarrow \mathcal{A}(1^{\lambda}, \mathsf{crs}_{\mathsf{com}}) \\ b \overset{R}{\leftarrow} \{0,1\} \\ (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m_b) \\ b' \leftarrow \mathcal{A}(com, state) \end{array} \right] - \frac{1}{2}\ \right|$$

is negligible in λ.

(2) *F_{gp}-Binding:* For all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}_{\mathsf{C},\mathcal{A}}^{F_{\mathsf{gp}}\text{-}\mathsf{Binding}}(\lambda)$ defined by

$$\Pr\left[\begin{array}{c}\mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom, M) = 1\\ \wedge\\ \mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, decom', M') = 1\end{array}\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)\\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp})\\ (com, M, decom, M', decom') \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}_{\mathsf{com}})\\ M \neq M'\end{array}\right]$$

is negligible in λ.

(3) *Equivocal:* There exist PPT algorithms SimSetup, SimCom and Equiv such that for all PPT adversaries $\mathcal{A}$

(a) we have that the advantage $\mathsf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{SimSetup}}(\lambda)$ defined by

$$\left|\begin{array}{r}\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}})\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp})\end{array}\right]\\ -\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}})\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ (\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}) \leftarrow \mathsf{SimSetup}(\mathsf{gp})\end{array}\right]\end{array}\right|$$

is negligible in λ.

(b) we have that the advantage $\mathsf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{Equiv}}(\lambda)$ defined by

$$\left|\begin{array}{r}\Pr\left[1 \leftarrow \mathcal{A}\begin{pmatrix}\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}},\\ m, com, decom\end{pmatrix}\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ (\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}) \leftarrow \mathsf{SimSetup}(\mathsf{gp}),\\ m \leftarrow \mathcal{M},\\ (com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, m)\end{array}\right]\\ -\Pr\left[1 \leftarrow \mathcal{A}\begin{pmatrix}\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}},\\ m, \widetilde{com}, decom\end{pmatrix}\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ (\mathsf{crs}_{\mathsf{com}}^{\mathsf{eq}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}) \leftarrow \mathsf{SimSetup}(\mathsf{gp}),\\ (\widetilde{com}, eq) \leftarrow \mathsf{SimCom}(\mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}),\\ m \leftarrow \mathcal{M},\\ decom \leftarrow \mathsf{Equiv}(\mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}, m, eq)\end{array}\right]\end{array}\right|$$

is zero.

(4) *Extractable:* There exist PPT algorithms ExtSetup and Extract such that for all PPT adversaries $\mathcal{A}$

(a) we have that the advantage $\mathsf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{ExtSetup}}(\lambda)$ defined by

$$\left|\begin{array}{r}\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}})\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ \mathsf{crs}_{\mathsf{com}} \leftarrow \mathsf{Setup}(\mathsf{gp})\end{array}\right]\\ -\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}}^{\mathsf{ext}})\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ (\mathsf{crs}_{\mathsf{com}}^{\mathsf{ext}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{ext}}) \leftarrow \mathsf{ExtSetup}(\mathsf{gp})\end{array}\right]\end{array}\right|$$

is negligible in λ.

(b) we have that the advantage $\mathsf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{Ext}}(\lambda)$ defined by

$$\Pr\left[\mathsf{Extract}\left(\mathsf{td}_{\mathsf{com}}^{\mathsf{ext}}, com\right) \neq F_{\mathsf{gp}}(m)\middle|\begin{array}{c}\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda),\\ (\mathsf{crs}_{\mathsf{com}}^{\mathsf{ext}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{ext}}) \leftarrow \mathsf{ExtSetup}(\mathsf{gp}),\\ com \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{com}}^{\mathsf{ext}}),\\ \exists!\, m \in \mathcal{M}, r : com \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}^{\mathsf{ext}}, m; r)\end{array}\right]$$

is zero.

Furthermore, assume that the message space of C is an additive group. Then C is called *additively homomorphic*, if there exist additional PPT algorithms $com \leftarrow \mathsf{CAdd}(\mathsf{crs}_{\mathsf{com}}, com_1, com_2)$ and $decom \leftarrow \mathsf{DAdd}(\mathsf{crs}_{\mathsf{com}}, decom_1, decom_2)$ which on input of two commitments and corresponding decommitment values $(com_1, decom_1) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m_1)$ and $(com_2, decom_2) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m_2)$, output a commitment com and decommitment $decom$, respectively, such that $\mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, com, F_{\mathrm{gp}}(m_1 + m_2), decom) = 1$.

Finally, we call C *opening complete* if for all $M \in \mathcal{M}'$ and arbitrary values $com, decom$ with $\mathsf{Open}(\mathsf{crs}_{\mathsf{com}}, M, com, decom) = 1$ holds that there exists $m \in \mathcal{M}$ and randomness r such that $(com, decom) \leftarrow \mathsf{Com}(\mathsf{crs}_{\mathsf{com}}, m; r)$.

In Chapter 3 we will use a group-based commitment scheme that is correct, statistically hiding, additively homomorphic, equivocal, and F_{gp}-Binding for $F_{\mathrm{gp}}(m_1, \ldots, m_\alpha) \coloneqq (g_1^{m_1}, \ldots, g_1^{m_\alpha})$.

2.4. Digital Signatures

A digital signature scheme allows a signer to issue a signature σ on a message m using its secret signing key sk such that anybody can publicly verify that σ is a valid signature for m using the public verification key vk of the signer but nobody can feasibly forge a signature without knowing sk.

Definition 2.25 (Digital Signature Scheme)**.** A *digital signature scheme* $\Sigma \coloneqq (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ consists of the following PPT algorithms:

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ takes 1^λ as input and outputs a keypair $(\mathsf{vk}, \mathsf{sk})$ consisting of verification key vk and signing key sk
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ takes the signing key sk and a message m as input and outputs a signature σ
- $0/1 = \mathsf{Vfy}(\mathsf{vk}, m, \sigma)$ is a deterministic polynomial-time algorithm, which takes as input a verification key vk, a message m and a signature σ. It returns either 0 or 1. (Intuitively, returning 1 means that σ is a *valid* signature for message m under verification key vk. Returning 0 means it is invalid.)

We say that Σ is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\ \forall (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)\ \forall m \in \mathcal{M}\ \forall \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \ : \ \mathsf{Vfy}(\mathsf{vk}, m, \sigma) = 1$$

To define security for digital signature schemes, we use the standard notion of EUF-CMA-security (existential unforgeability under chosen message attacks).

Definition 2.26 (EUF-CMA-Security)**.** A digital signature scheme Σ is EUF-CMA-*secure* if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[\mathsf{Vfy}(\mathsf{vk}, m^*, \sigma^*) = 1 \;\middle|\; \begin{array}{c} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \\ m^* \notin \{m_1, \ldots, m_q\} \end{array} \right]$$

is negligible in λ, where $\mathsf{Sign}(\mathsf{sk}, \cdot)$ is an oracle that, on input m, returns $\mathsf{Sign}(\mathsf{sk}, m)$, and $\{m_1, \ldots, m_q\}$ denotes the set of messages queried by $\mathcal{A}$ to its oracle.

Since we also need the group-based versions of a signature scheme, we define it here as well, as well es the corresponding version of EUF-CMA-security.

Definition 2.27 (Group-Based Digital Signature Scheme and EUF-CMA-Security)**.** A *(group-based) digital signature scheme* $\mathsf{S} \coloneqq (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ consists of four PPT algorithms:

- $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)$ takes as input a security parameter 1^λ and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms.
- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp})$ takes gp as input and outputs a key pair $(\mathsf{vk}, \mathsf{sk})$ consisting of public verification key vk and secret signing key sk. The verification key vk and gp define a message space $\mathcal{M}$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ takes as input the signing key sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $0/1 = \mathsf{Vfy}(\mathsf{vk}, m, \sigma)$ takes as input the verification key vk, a message $m \in \mathcal{M}$, and a signature σ, and outputs a bit. (Intuitively, returning 1 means that σ is a *valid* signature for message m under verification key vk. Returning 0 means it is invalid.)

We say that S is *correct* if the following holds:

$$\forall \lambda \in \mathbb{N}\ \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)\ \forall (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp})\ \forall m \in \mathcal{M}\ \forall \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\ : \\ \mathsf{Vfy}(\mathsf{vk}, m, \sigma) = 1$$

We say that S is EUF-CMA-*secure* if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[\mathsf{Vfy}(\mathsf{vk}, m^*, \sigma^*) = 1 \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \\ m^* \notin \{m_1, \ldots, m_q\} \end{array} \right]$$

is negligible in λ, where $\mathsf{Sign}(\mathsf{sk}, \cdot)$ is an oracle that, on input m, returns $\mathsf{Sign}(\mathsf{sk}, m)$, and $\{m_1, \ldots, m_q\}$ denotes the set of messages queried by $\mathcal{A}$ to its oracle.

In Chapter 3 and Chapter 4 we will need EUF-CMA-secure signature schemes, while the one in Chapter 3 also needs to be structure-preserving. Structure-preserving signatures are pairing-based signatures where all the messages, signatures and verification keys are group elements.

2.5. Zero-Knowledge Proofs

Let $\mathcal{R}$ be a witness relation for some NP language

$$L = \{stmt \mid \exists\, wit \text{ s.t. } (stmt, wit) \in \mathcal{R}\}.$$

Informally, a zero-knowledge proof system allows a prover P to convince a verifier V that some *stmt* is contained in L without V learning anything beyond that fact. In particular, V learns nothing about *wit*. In a non-interactive zero-knowledge (NIZK) proof, only one message, the proof π, is sent from P to V for that purpose.

NIZKs come in many different flavors with different definitions. Therefore, here we limit ourselves to those definitions that we will actually need later.

First, we define a group-based NIZK proof system POK with perfect completeness and perfect soundness, which is a (dual-mode) NIZK where one can switch between composable zero-knowledge and perfect F_{gp}-extractability. This will be needed in Chapter 3, where we will use the SXDH-based Groth-Sahai proof system [EG14; GS08] as POK. Since here we are again in the bilinear group setting, the witness relation $\mathcal{R}_{\mathrm{gp}}$ is also defined over the bilinear group description gp: Here, $\mathcal{R}_{\mathrm{gp}}$ is a witness relation for some NP language

$$L_{\mathrm{gp}} = \{stmt \mid \exists\, wit \text{ s.t. } (\mathrm{gp}, stmt, wit) \in \mathcal{R}_{\mathrm{gp}}\}.$$

More precisely, our group-based NIZK proof system is defined as:

Definition 2.28 (Group-Based NIZK Proof System)**.** Let $\mathcal{R}_{\mathrm{gp}}$ be an efficiently verifiable relation containing triples $(\mathrm{gp}, stmt, wit)$. We call gp the group setup, *stmt* the statement, and *wit* the witness. Given some gp, let L_{gp} be the language containing all statements *stmt* such that $(\mathrm{gp}, stmt, wit) \in \mathcal{R}_{\mathrm{gp}}$. Let $\mathsf{POK} := (\mathsf{SetupGrp}, \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a tuple of four PPT algorithms such that

- $\mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)$ takes as input a security parameter 1^λ and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms.
- $\mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{Setup}(\mathrm{gp})$ takes as input gp and outputs a (public) common reference string $\mathsf{crs}_{\mathsf{pok}}$.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}_{\mathsf{pok}}, stmt, wit)$ takes as input the common reference string $\mathsf{crs}_{\mathsf{pok}}$, a statement *stmt*, and a witness *wit* with $(\mathrm{gp}, stmt, wit) \in \mathcal{R}_{\mathrm{gp}}$ and outputs a proof π.
- $1/0 = \mathsf{Verify}(\mathsf{crs}_{\mathsf{pok}}, stmt, \pi)$ takes as input the common reference string $\mathsf{crs}_{\mathsf{pok}}$, a statement *stmt*, and a proof π and outputs 1 or 0.

POK is called a *non-interactive zero-knowledge proof system for $\mathcal{R}_{\mathrm{gp}}$ with F_{gp}-extractability*, if the following properties are satisfied:

(1) *Perfect completeness:* For all $\mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)$, $\mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{Setup}(\mathrm{gp})$, $(\mathrm{gp}, stmt, wit) \in \mathcal{R}_{\mathrm{gp}}$, and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}_{\mathsf{pok}}, stmt, wit)$ we have that $\mathsf{Verify}(\mathsf{crs}_{\mathsf{pok}}, stmt, \pi) = 1$.

(2) *Perfect soundness:* For all (possibly unbounded) adversaries $\mathcal{A}$ we have that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}_{\mathsf{pok}}, stmt, \pi) = 0 \;\middle|\; \begin{array}{c} \mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda) \\ \mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{Setup}(\mathrm{gp}) \\ (stmt, \pi) \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{pok}}) \\ stmt \notin L_{\mathrm{gp}} \end{array}\right]$$

is 1.

(3) *Perfect F_{gp}-extractability:* There exists a polynomial-time extractor $(\mathsf{ExtSetup}, \mathsf{ExtractW})$ such that for all (possibly unbounded) adversaries $\mathcal{A}$

(a) we have that the advantage $\mathsf{Adv}^{\mathsf{pok\text{-}ext\text{-}setup}}_{\mathsf{POK},\mathcal{A}}(\lambda)$ defined by

$$\left| \begin{array}{r} \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{pok}}) \;\middle|\; \begin{array}{c} \mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda), \\ \mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{Setup}(\mathrm{gp}) \end{array}\right] \\ - \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}'_{\mathsf{pok}}) \;\middle|\; \begin{array}{c} \mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda), \\ (\mathsf{crs}'_{\mathsf{pok}}, \mathsf{td}^{\mathsf{ext}}_{\mathsf{pok}}) \leftarrow \mathsf{ExtSetup}(\mathrm{gp}) \end{array}\right] \end{array} \right|$$

is zero.

(b) we have that the advantage $\mathrm{Adv}^{\text{pok-ext}}_{\mathrm{POK},\mathcal{A}}(\lambda)$ defined by

$$\Pr\left[\begin{array}{c|c} \exists\, wit : & \mathrm{gp} \leftarrow \mathrm{SetupGrp}(1^\lambda) \\ F_{\mathrm{gp}}(wit) = WIT \wedge & (\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{ext}}_{\mathrm{pok}}) \leftarrow \mathrm{ExtSetup}(\mathrm{gp}) \\ (\mathrm{gp}, stmt, wit) \in \mathcal{R}_{\mathrm{gp}} & (stmt, \pi) \leftarrow \mathcal{A}(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{ext}}_{\mathrm{pok}}) \\ & 1 \leftarrow \mathrm{Verify}(\mathrm{crs}'_{\mathrm{pok}}, stmt, \pi) \\ & WIT \leftarrow \mathrm{ExtractW}(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{ext}}_{\mathrm{pok}}, stmt, \pi) \end{array}\right]$$

is 1.

(4) *Composable Zero-knowledge:* There exists a polynomial-time simulator (SimSetup, SimProve) and hint generator GenHint such that for all PPT adversaries $\mathcal{A}$

(a) we have that the advantage $\mathrm{Adv}^{\text{pok-zk-setup}}_{\mathrm{POK},\mathcal{A}}(\lambda)$ defined by

$$\left| \begin{array}{l} \Pr\left[1 \leftarrow \mathcal{A}(\mathrm{crs}_{\mathrm{pok}}) \;\middle|\; \begin{array}{c} \mathrm{gp} \leftarrow \mathrm{SetupGrp}(1^\lambda), \\ \mathrm{crs}_{\mathrm{pok}} \leftarrow \mathrm{Setup}(\mathrm{gp}) \end{array}\right] \\ - \Pr\left[1 \leftarrow \mathcal{A}(\mathrm{crs}'_{\mathrm{pok}}) \;\middle|\; \begin{array}{c} \mathrm{gp} \leftarrow \mathrm{SetupGrp}(1^\lambda), \\ hint \leftarrow \mathrm{GenHint}(\mathrm{gp}), \\ (\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}) \leftarrow \mathrm{SimSetup}(\mathrm{gp}, hint), \end{array}\right] \end{array} \right|$$

is negligible in λ.

(b) we have that the advantage $\mathrm{Adv}^{\text{pok-zk}}_{\mathrm{POK},\mathcal{A}}(\lambda)$ defined by

$$\left| \begin{array}{l} \Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathrm{SimProve}'(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}, \cdot, \cdot)}(1^\lambda, \mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}})\ \right] \\ - \Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathrm{Prove}(\mathrm{crs}'_{\mathrm{pok}}, \cdot, \cdot)}(1^\lambda, \mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}})\ \right] \end{array} \right|$$

is negligible in λ, where $\mathrm{gp} \leftarrow \mathrm{SetupGrp}(1^\lambda)$, $(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}) \leftarrow \mathrm{SimSetup}(\mathrm{gp})$, and $\mathrm{SimProve}'(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}, \cdot, \cdot)$ is an oracle which returns $\mathrm{SimProve}(\mathrm{crs}'_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}, stmt)$ on input $(stmt, z) \in \mathcal{R}$. Both SimProve′ and Prove return $\perp$ on input $(stmt, z) \notin \mathcal{R}$.

We wish to point out some remarks.

Remark 2.29.

(1) The considered language L_{gp} may depend on gp.

(2) F_{gp}-extractability actually implies soundness independent of F_{gp}: If there was a false statement *stmt* which verifies, violating soundness, then obviously there is no witness *wit* for *stmt*, which violates extractability.

(3) Extractability essentially means that ExtractW—given a trapdoor $\mathrm{td}^{\mathrm{ext}}_{\mathrm{pok}}$—is able to extract $F_{\mathrm{gp}}(wit)$ for an NP-witness *wit* for $stmt \in L_{\mathrm{gp}}$ from any valid proof π. If F_{gp} is the identity function, then the actual witness is extracted and the system is called a *proof of knowledge.*

For Chapter 4 we need a different flavor of NIZK: There we need the ability to simulate proofs and extract witnesses *with the same setup.* Therefore, the dual-mode NIZK just described is not sufficient. Furthermore, we do not need the group-based version in Chapter 4, the standard version is sufficient. Therefore, we now give another definition of NIZKs that satisfies our enhanced security needs, i.e., straight-line simulation-extractability.

Definition 2.30 (NIZKPoK)**.** A *non-interactive zero-knowledge proof of knowledge (NIZKPoK)* system $\mathsf{NIZK} \coloneqq (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext})$ for NP-relation $\mathcal{R}$ consists of the following five PPT algorithms:

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$ outputs $(\mathsf{crs}, \mathsf{td})$, where crs is the CRS and td is a trapdoor.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathit{stmt}, \mathit{wit})$ generates a proof π given the CRS crs and $(\mathit{stmt}, \mathit{wit}) \in \mathcal{R}$.
- $1/0 = \mathsf{Verify}(\mathsf{crs}, \mathit{stmt}, \pi)$ verifies a proof π for statement *stmt* and outputs 1 or 0.
- $\pi \leftarrow \mathsf{Sim}(\mathsf{td}, \mathit{stmt})$ outputs a proof π.
- $\mathit{wit}/\bot \leftarrow \mathsf{Ext}(\mathsf{td}, \mathit{stmt}, \pi)$ outputs a witness *wit* with $(\mathit{stmt}, \mathit{wit}) \in \mathcal{R}$ or $\bot$.

Note that we sometimes explicitly add the relation $\mathcal{R}$ as input to the algorithms to make it clear for which relation the NIZKPoK is. Completeness and soundness for NIZKPoKs are defined analogous to Definition 2.28.

Definition 2.31 (Straight-Line Simulation-Extractability)**.** Let NIZK be a NIZKPoK for NP-relation $\mathcal{R}$. We define the *straight-line simulation-extractability* game as follows: Let $\mathcal{A}$ be an adversary.

(1) Setup $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$.

(2) Run $\mathcal{A}(1^\lambda, \mathsf{crs})$, where $\mathcal{A}$ is given oracle access to oracle $(\mathit{stmt}, \pi) \mapsto \mathsf{Ext}(\mathsf{td}, \mathit{stmt}, \pi)$ and either

   - $(\mathit{stmt}, \mathit{wit}) \mapsto \mathsf{Prove}(\mathsf{crs}, \mathit{stmt}, \mathit{wit})$, or
   - $(\mathit{stmt}, \mathit{wit}) \mapsto \mathsf{Sim}(\mathsf{td}, \mathit{stmt})$,

   with the choice made uniformly at random. Let $\mathsf{L}_{\mathsf{Ext}}$ be the list of queries with outputs $(\mathit{stmt}, \pi, \mathit{wit})$ to Ext, and $\mathsf{L}_{\mathsf{Prove}}$ be the list of queries with outputs $(\mathit{stmt}, \mathit{wit}, \pi)$ to Prove or Sim.

(3) Eventually, $\mathcal{A}$ outputs a guess b, where $b = 0$ (resp. $b = 1$) indicates that it interacted with Prove (reps. Sim).

(4) $\mathcal{A}$ wins the game if:

   - $\mathcal{A}$ never queried (stmt, π) if π was previously returned from a query to Prove (or Sim) with statement *stmt*. (I.e., $\mathcal{A}$ may not trivially distinguish simulated proofs from real proofs.)
   - For any $(\mathit{stmt}, \mathit{wit}, \pi) \in Q_{Ext}$, we have $\mathit{wit} = \bot$, but $\mathsf{Verify}(\mathsf{crs}, \mathit{stmt}, \pi) = 1$. (I.e., extraction failed for an accepting proof.)
   - $\mathcal{A}$ guessed b correctly.

We say that NIZK is *straight-line simulation-extractable*, if for any PPT adversary, the probability that it wins the straight-line simulation-extractability game is negligible.

Note that weaker notions of simulation-extractability exist, which are not straight-line. But there are simple and efficient transformations to ensure straight-line extractability (e.g., by adding an encryption of the witness under a public key of a PKE which is put into the crs (and proving consistency as part of the relation)).

2.6. Pseudo-Random Functions

A pseudo-random function (PRF) $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a keyed function whose output cannot be distinguished from randomness, i.e., any PPT adversary given oracle access to either $F(k, \cdot)$ or a randomly chosen function $R : \mathcal{X} \to \mathcal{Y}$, cannot distinguish between them with non-negligible probability. A PRF (more precisely a family of PRF's in the security parameter 1^λ) is formally defined as follows. Note that we here directly define the group-based version, since we will only need this version.

Definition 2.32 (Group-Based Pseudo-Random Function)**.** A (group-based) *pseudo-random function* (PRF) $\mathsf{PRF} := (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Eval})$ consists of three PPT algorithms:

- $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda)$ takes as input a security parameter 1^λ and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms. The input domain $\mathcal{X}_{\mathsf{gp}}$, the key space $\mathcal{K}_{\mathsf{gp}}$, and the codomain $\mathcal{Y}_{\mathsf{gp}}$ may all depend on gp.
- $k \leftarrow \mathsf{Gen}(\mathsf{gp})$ takes gp as input and outputs a key $k \in \mathcal{K}_{\mathsf{gp}}$. (Typically, we have $k \overset{R}{\leftarrow} \mathcal{K}_{\mathsf{gp}}$.)
- $y \leftarrow \mathsf{Eval}(k, x)$ is a deterministic algorithm which takes as input a key $k \in \mathcal{K}_{\mathsf{gp}}$ and a value $x \in \mathcal{X}_{\mathsf{gp}}$, and outputs some $y \in \mathcal{Y}_{\mathsf{gp}}$. Usually, we simply write $y = \mathsf{PRF}(k, x)$ for short.

We say that PRF is secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{A}}(\lambda)$ defined by

$$\left| \begin{array}{r} \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{PRF}(k,\cdot)}(\mathsf{gp}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda), \\ k \leftarrow \mathsf{Gen}(\mathsf{gp}) \end{array} \right] \\ - \Pr\left[1 \leftarrow \mathcal{A}^{R(\cdot)}(\mathsf{gp}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^\lambda), \\ R \overset{R}{\leftarrow} \{R : \mathcal{X}_{\mathsf{gp}} \to \mathcal{Y}_{\mathsf{gp}}\} \end{array} \right] \end{array} \right|$$

is negligible in λ.

We will use a group-based pseudo-random function PRF in Chapter 3.

2.7. Introduction to the UC Framework

The *Universal Composability (UC)* framework [Can00; Can01; Can20] is a model to analyze cryptographic protocols which offers strong security guarantees. Security is proven by showing that two different worlds are indistinguishable from one another. More precisely, in the UC-framework, an ideal functionality $\mathcal{F}$, which is acting as a trusted third party (TTP), is defined which plainly solves the problem at hand in a secure and privacy-preserving manner. A protocol Π is a *secure realization* of this ideal functionality $\mathcal{F}$ if no PPT-machine $\mathcal{Z}$—called the *environment*—can distinguish between two experiments: the *real experiment* (running Π) and the *ideal experiment* (using $\mathcal{F}$).

In the *real experiment*, $\mathcal{Z}$ interacts with parties running the actual protocol Π and is supported by a real adversary $\mathcal{A}$. The environment $\mathcal{Z}$ specifies the input of the honest parties, receives their output and determines the overall course of action. The adversary $\mathcal{A}$ is instructed by $\mathcal{Z}$ and represents $\mathcal{Z}$'s interface to the network, e.g., $\mathcal{A}$ reports all messages generated by any party to $\mathcal{Z}$ and can manipulate, reroute, inject and/or suppress messages on $\mathcal{Z}$'s order. Moreover, $\mathcal{Z}$ may instruct $\mathcal{A}$ to corrupt parties. In this case, $\mathcal{A}$ takes over the role of the corrupted party, reports its internal state to $\mathcal{Z}$ and from then on may arbitrarily deviate from the protocol Π.

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

$\mathcal{F}_{\text{CRS}}$ proceeds as follows, when parameterized by a distribution $\mathcal{D}$.

Value: When activated for the first time on input $(\textsc{value}, sid)$, choose $d \leftarrow \mathcal{D}$ and send d back to the activating party. In each other activation return the value d to the activating party.

Figure 2.2.: The common reference string functionality $\mathcal{F}_{\text{CRS}}$ from [CF01]

In the *ideal experiment* the protocol parties are mere dummies that pass their (private) input to the TTP $\mathcal{F}$ and hand over $\mathcal{F}$'s output as their own output. The ideal functionality $\mathcal{F}$ is incorruptible and executes the task in a trustworthy manner. The real adversary $\mathcal{A}$ is replaced by a simulator $\mathcal{S}$. The simulator must mimic the behavior of $\mathcal{A}$, e.g., simulate appropriate network messages (there are no network messages in the ideal experiment), and come up with a convincing internal state for corrupted parties (dummy parties do not have an internal state).

If no environment $\mathcal{Z}$ can distinguish executions of the real and ideal experiment, any attack on the real experiment is also possible in the ideal one. Therefore the protocol Π guarantees the same level of security as the (inherently secure) ideal functionality $\mathcal{F}$. One then says that Π *UC-realizes* $\mathcal{F}$ and we denote this by

$$\Pi \geq_{\text{UC}} \mathcal{F}.$$

Regarding privacy, note that all parties (including the simulator) use the ideal functionality as a black-box and only know what it explicitly allows them to know as part of their output. This makes UC suitable to reason about privacy in a very nice way. As no additional information is unveiled, the achieved level of privacy can directly be deduced from the output of the ideal functionality.

2.8. Setup Assumptions in the UC Framework

The UC framework provides very strong security guarantees, but faces impossibility results in the plain model: In a seminal paper, Canetti and Fischlin [CF01] show that UC-secure commitment schemes are impossible to achieve in the plain model, i.e., without setup assumptions. Thus, most more involved UC-secure protocols rely on setup assumptions, and our constructions in Chapter 3 and Chapter 4 are no exception. Setup assumptions are ideal functionalities that remain ideal in the real experiment, e.g., their secure realization is left unspecified. Especially, the security of their realization has to be either proven outside of the (UC-)model or simply assumed.

We use several ideal functionalities as setup assumptions in our systems, which we will introduce now.

The CRS functionality. A *common reference string (CRS)* is a short piece of information that is shared between all parties. The trust assumptions are that the CRS has been generated honestly and that all parties possess the same CRS. The CRS functionality $\mathcal{F}_{\text{CRS}}$ is depicted in Fig. 2.2 and will be used in Chapter 3 as well as Chapter 4.

Functionality $\mathcal{F}_{\text{BB}}$

The functionality models a bulletin-board. Any party can register a *single* (uid, v) pair associated with its identity, where the *uid*s need to be unique. Any party can retrieve registered values. Initially the list of recorded values is empty, i.e., $\mathsf{L}_{\text{BB}} = \emptyset$.

Register: Upon receiving $(\textsc{Register}, uid, v)$ from party P, send $(\textsc{Register}, uid, v)$ to the adversary. Upon receiving $(\textsc{Ok})$ from the adversary, if there is no record $(P, \cdot, \cdot) \in \mathsf{L}_{\text{BB}}$ and no record $(\cdot, uid, \cdot) \in \mathsf{L}_{\text{BB}}$ record the pair (P, uid, v). Otherwise, do nothing.

Retrieve by Identity: Upon receiving $(\textsc{Retrieve}, P_i)$ from party P_j (including the adversary), generate public delayed output $(\textsc{Retrieve}, P_i, uid, v)$ to P_j where (uid, v) is such that $(P_i, uid, v) \in \mathsf{L}_{\text{BB}}$ or $(\perp, \perp)$ if no entry is recorded in L_{BB}.

Retrieve by UID: Upon receiving $(\textsc{Retrieve}, uid)$ from party P_j (including the adversary), generate public delayed output $(\textsc{Retrieve}, P_i, uid, v)$ to P_j where (P_i, v) is such that $(P_i, uid, v) \in \mathsf{L}_{\text{BB}}$ or $(\perp, \perp)$ if no entry is recorded in L_{BB}.

Figure 2.3.: Bulletin board functionality $\mathcal{F}_{\text{BB}}$, adapted from [KL11]

Functionality $\mathcal{G}_{\text{bb}}$

Register: Upon input $(\textsc{Register}, (v_1, v_2, \ldots))$ from a party $\mathcal{P}$ with PID $pid_{\mathcal{P}}$, send $(\textsc{Registering}, pid_{\mathcal{P}}, (v_1, v_2, \ldots))$ to the adversary. Upon receiving OK from the adversary proceed, else abort. If another entry $(pid_{\mathcal{P}}, (\ldots))$ has already been registered for $pid_{\mathcal{P}}$, abort. If $\exists\ pid_{\mathcal{P}'}$ and i, j such that $(pid_{\mathcal{P}'}, (\ldots, v'_i, \ldots))$ has been recorded and $v'_i = v_j$ holds, abort.

Record the pair $(pid_{\mathcal{P}}, (v_1, v_2, \ldots))$.

Retrieve: Upon input $(\textsc{Retrieve}, pid)$ from some party $\mathcal{P}$ (or the adversary), look up $(pid, (v_1, v_2, \ldots))$ and generate a delayed output $(pid, (v_1, v_2, \ldots))$ to $\mathcal{P}$ with $(v_1, v_2, \ldots) = \perp$ if no record exists.

Reverse Retrieve (Partial, reverse lookup): Upon input $(\textsc{ReverseRetrieve}, j, v)$ from some party $\mathcal{P}$ (or the adversary), look up $(pid, (v_1, v_2, \ldots, v_j, \ldots))$ with $v_j = v$ and generate a delayed output $(pid, (v_1, v_2, \ldots))$ to $\mathcal{P}$ or $(\perp, \perp)$ if no entry exists.

Figure 2.4.: Global bulletin board functionality $\mathcal{G}_{\text{bb}}$, adapted from [CSV16, Fig. 3]

The bulletin board functionality. A bulletin board can be depicted as a key registration service which associates (physical) party identifiers (PIDs) with (cryptographic) public keys. The assumptions about the bulletin board functionality are that upon registration the operator of the bulletin board checks the identity of the registering party in a trustworthy way and that every party can retrieve information from the bulletin board trustworthily. The bulletin board functionality $\mathcal{F}_{\text{BB}}$ is depicted in Fig. 2.3 and will be used in Chapter 4. For Chapter 3 we use a *global* bulletin board functionality $\mathcal{G}_{\text{bb}}$, which is depicted in Fig. 2.4.

The global clock functionality. To model the concept of *time* in the UC framework, a globally available "clock" functionality $\mathcal{G}_{\text{CLOCK}}$ is used. It should be noted that $\mathcal{G}_{\text{CLOCK}}$ does not model the time in a traditional sense, but rather models *rounds*. After all (honest) parties and functionalities in the respective session have sent a "clock-update" to $\mathcal{G}_{\text{CLOCK}}$, the next round starts. The round counter can always be read by any party. The global clock functionality $\mathcal{G}_{\text{CLOCK}}$ is depicted in Fig. 2.5 and will be used in Chapter 4.

Global Functionality $\mathcal{G}_{\text{CLOCK}}$

The functionality manages the set $\mathcal{P}$ of registered identities, i.e., parties $P = (pid, sid)$. It also manages the set F of functionalities (together with their session identifier). Initially, $\mathcal{P} \coloneqq \emptyset$ and $F \coloneqq \emptyset$.
For each session sid the clock maintains a variable τ_{sid}. For each identity $P \coloneqq (pid, sid) \in \mathcal{P}$ it manages a variable d_P. For each pair $(\mathcal{F}, sid) \in F$ it manages a variable $d_{\mathcal{F},sid}$ (all integer variables are initially 0).

Synchronization:

- Upon receiving (Clock-Update, sid_C) from some party $P \in \mathcal{P}$ set $d_P \coloneqq 1$, execute Round-Update and forward (Clock-Update, sid_C, P) to $\mathcal{A}$
- Upon receiving (Clock-Update, sid_C) from some functionality $\mathcal{F}$ in a session sid such that $(\mathcal{F}, sid) \in F$ set $d_{\mathcal{F},sid} \coloneqq 1$, execute Round-Update and return (Clock-Update, $sid_C, \mathcal{F}$) to this instance of $\mathcal{F}$.
- Upon receiving (Clock-Read, sid_C) from any participant (including the environment on behalf of a party, the adversary, or any ideal—shared or local—functionality), return (Clock-Read, sid_C, τ_{sid}) to the requestor (where sid is the session id of the calling instance).

Party Management:

- Upon receiving (Register, sid_C) from some party P_i (or from $\mathcal{A}$ on behalf of a corrupted P_i), set $\mathcal{P} = \mathcal{P} \cup \{P_i\}$. Return (Register, sid_C, P_i) to the caller.
- Upon receiving (De-Register, sid_C) from some party $P_i \in \mathcal{P}$, the functionality updates $\mathcal{P} \coloneqq \mathcal{P} \setminus \{P_i\}$ and returns (De-Register, sid_C, P_i) to P_i.
- Upon receiving (Is-Registered, sid_C) from some party P_i, return (Register, sid_C, b) to the caller, where the bit b is 1 if and only if $P_i \in \mathcal{P}$.
- Upon receiving (Get-Registered, sid_C) from $\mathcal{A}$, the functionality returns the response (Get-Registered, $sid_C, \mathcal{P}$) to $\mathcal{A}$.
- Upon receiving (Register, sid_C) from a functionality $\mathcal{F}$ (with session-id sid), update $F \coloneqq F \cup \{(\mathcal{F}, sid)\}$.
- Upon receiving (De-Register, sid_C) from a functionality $\mathcal{F}$ (with session-id sid), update $F \coloneqq F \setminus \{(\mathcal{F}, sid)\}$.
- Upon receiving (Get-Registered-F, sid_C) from $\mathcal{A}$, return (Get-Registered-F, sid_C, F) to $\mathcal{A}$

Procedure Round-Update: For each session sid do: If $d_{\mathcal{F},sid} = 1$ for all $\mathcal{F} \in F$ and $d_P = 1$ for all honest parties $P = (\cdot, sid) \in \mathcal{P}$, then set $\tau_{sid} \coloneqq \tau_{sid} + 1$ and reset $d_{\mathcal{F},sid} \coloneqq 0$ and $d_P \coloneqq 0$ for all parties $P = (\cdot, sid) \in \mathcal{P}$.

Figure 2.5.: The global clock functionality $\mathcal{G}_{\text{CLOCK}}$ from [Bad+17]

3. P4TC - Provably-Secure yet Practical Privacy-Preserving Toll Collection

This chapter is based on [Fet+18; Fet+20] and most parts of this chapter are either taken verbatim or rephrased from these works, along with some additional explanations. In both the conference version [Fet+20] and the ePrint version [Fet+18], significant parts (in particular the complete ideal functionality and the complete protocol) were only given in the appendix. In this thesis, the contents of [Fet+18; Fet+20] are thus reorganized to form a consistent overall structure. The formatting of the ideal functionality and protocol has also been adjusted to make the formatting consistent within the thesis. The exception to this is Section 3.6, which was not part of [Fet+18; Fet+20] and is an original contribution to this thesis.

In [Fet+18; Fet+20], but not part of this thesis, the real-world practicality of P4TC was evaluated by means of an implementation. We briefly sketch two results of the evaluation here to give an idea about the practicality of P4TC, but the detailed evaluation results will not be featured further in this thesis.

(1) The time-critical task Debt Accumulation has been benchmarked on realistic hardware, in particular on an embedded processor which is known to be used in currently available OBUs such as the Savari MobiWAVE [Sav17]. The implementation already shows quite practical performance figures due to carefully chosen precomputations, although there is still room for optimization. An interaction between an on-board unit and a road-side unit is estimated to take less than a second, which allows to collect tolls at 120 km/h, assuming one road-side unit per lane.

(2) To blacklist a user, the Dispute Resolver (DR) must compute a series of discrete logarithms to recover the wallet ID λ. Our choice of parameters splits λ into 32-bit values, resulting in the calculation of eight 32-bit DLOGs. Recovery of one λ takes the DR $\approx$12 s assuming certain precomputations have been done during system setup.

The implementation thus suggests that provably-secure ETC at full speed can indeed be realized using present-day hardware.

Structure of this Chapter. We first give an introduction to electronic toll collection in Section 3.1, discuss related work, and state the contribution. In Section 3.2 we introduce the scenario, give a high-level system overview, sketch some privacy implications of the system, and discuss the desired security properties of our system.

The concrete presentation of the privacy-preserving electronic toll collection scheme starts in Section 3.3 with the ideal functionality $\mathcal{F}_{\mathrm{P4TC}}$. In Section 3.4 we present the protocol Π_{P4TC} that UC-realizes $\mathcal{F}_{\mathrm{P4TC}}$. There we also state the main security theorem, along with a proof sketch. The complete security proof is deferred to Appendix A.1.

We discuss the information leakage of the system and the difficulty of actually linking a specific transaction or trip to a user in Section 3.5. We conclude the chapter by evaluating the impact of different pricing schemes on user privacy in Section 3.6.

3.1. Introduction

Electronic toll collection (ETC) is already deployed in many countries world-wide, not only to finance our road infrastructures, but also to realize advanced features like congestion management and pollution reduction by means of dynamic pricing. A study from 2017 [Mar17] predicts a Compound Annual Growth Rate (CAGR) for this market of 9.16% from 2017 to 2022 reaching 10.6 Billion USD by 2022. Europe plans to introduce the first implementation of a fully interoperable tolling system (EETS) by 2027 [Eur17]. As ETC will become the default option for paying tolls with no easy way to opt-out, privacy is a concern of particular importance. Unfortunately, the systems in use today are inherently violating user privacy. This encourages abuse: E-ZPass records have been used as evidence in divorce lawsuits [ABC08], E-ZPass transponders are abused to track drivers throughout New York City [Ame15], and the Norwegian AutoPASS system allowed anyone to obtain a transcript of visited toll booths [Dat07]. The only legitimate reason for a *toll service provider (TSP)* to store personalized location records is to bill customers. Personalized location records may also be needed for violation enforcement, but for this the *state authority (SA)* (cf. Section 3.2) is responsible and not the TSP. Thus, an efficient and cost-effective privacy-preserving mechanism which avoids data collection in the first place, but still enables the billing functionality, should be of interest to TSPs. In this way, there is no need to deploy costly technical and organizational measures to protect a large amount of sensitive data and there is no risk of a data breach resulting in costly law suits, fines, and loss of customer trust. This is especially interesting in view of the new EU General Data Protection Regulation (GDPR) [Eur16].

Classification of ETC. One can classify ETC systems based on what the user is charged for, where toll determination takes place, and how this determination is done [Eur14; Eur15]. Concerning what the user is charged for, we distinguish between two major types of charging schemes:

Distance-based: Toll is calculated based on the distance traveled by the vehicle and adapted by other parameters like the type of vehicle, discounts, etc.

Access-based: Tolls apply to a specific geographic area, e.g., part of a city, segment of a highway, tunnel, etc. As before, it can dynamically be adapted by other parameters. This charging scheme is typically used in urban areas—not only to finance road infrastructure but also for congestion and pollution management through dynamically adapted tolls.

There are two types of toll determination environments:

Toll plaza: This is the traditional environment where cars pass toll booths on physically separated lanes which may be secured by barriers or cameras to enforce honest behavior.

Open road: In open road tolling the traffic is not disrupted as tolls are collected in a seamless fashion without forcing cars to slow down. In the DSRC setting, this is enabled by equipping roads with toll gantries and cameras enforcing participation.

Several key technologies define how toll is determined:

Dedicated Short-Range Communication (DSRC): This is the most widely used ETC technology world-wide and the de facto standard in Europe [Eur14]. It is based on bidirectional radio communication between *road-side units (RSUs)* and *on-board units (OBUs)* installed in the vehicles. In conventional systems, the OBU just identifies the user to trigger a payment. However, more complex protocols (like ours) between OBU and RSU can be implemented.

Automatic Number Plate Recognition (ANPR): Video tolling inherently violates privacy.

Global Navigation Satellite System (GNSS): With GNSS the OBU keeps track of the vehicle's location and processes the necessary information to measure its road usage autonomously, i.e., without the aid of an RSU. GNSS is typically used in combination with the Global System for Mobile Communications (GSM) for communicating with the toll service provider.

Drawbacks of Existing Systems and Proposals. Independent of the technology used, systems deployed in practice build on identifying the user to charge him. Previous academic work on privacy-preserving toll collection mainly considers the GNSS setting and comes—apart from a few exceptions [Bal+10; Che+13; DDS11]—without any formal security analysis. Although DSRC is currently the most dominant setting in practice, and according to recent market forcasts [Glo19], predicting an exponential growth, it will stay like this at least for the near future, it did not receive much attention in the literature so far. Moreover, practical issues like "what happens if an OBU breaks down", are usually not taken into account by these proposals. See Section 3.1.1 for details.

P4TC. We propose a comprehensive security model as well as a provably-secure and efficient protocol for privacy-friendly ETC in the DSRC setting. Note that our instantiation is not post-quantum secure as it relies on pairing-based cryptography. Our definitional framework and the system are very flexible. We cover access-based and distance-based charging schemes as well as combinations of those. Our protocol works for toll plaza environments, but is (due to offline precomputations) efficient enough for open road tolling. Additionally, we also cope with several issues that may arise in practice, e.g., broken/stolen OBUs, RSUs with non-permanent internet connection, imperfections of violation enforcement technology, etc. To the best of our knowledge, we arguably did the most comprehensive formal treatment of ETC security and privacy overall.

3.1.1. Related Work

In this section, we review the academic literature on privacy-preserving ETC, split by GNSS and DSCR setting. So far, more elaborate systems have been proposed for the GNSS setting in comparison to the actually more widely used DSRC setting. See Table 3.1 for a comparison of the contemplated systems.

GNSS Setting. A variety of GNSS-based system proposals can be found in the literature. Here, the OBU equipped with GPS and GSM typically collects location-time data or road segment prices and sends this data to the TSP. To ensure that users behave honestly and, e.g., do not omit or forge data, unpredictable spot checks are assumed which force users to reveal the data they sent at a certain location and time. In a reconciliation phase, users calculates their toll based on the data sent and prove their calculation to be correct.

In VPriv [PBB09], the OBU anonymously sends tagged location-time data to the TSP while driving. The user previously committed to use exactly these random tags in a registration phase with the TSP. In an inefficient reconciliation phase, each user downloads the database of all tagged location-time tuples, calculates their total fee, and proves that for this purpose, they correctly used the tuples belonging to their tags without leaking those. In [DDS11], ProVerif is used to verify VPriv's privacy guarantees for honest-but-curious adversaries.

In the PrETP [Bal+10] scheme, the OBU non-anonymously sends payment tuples consisting of commitments to location, time, and the corresponding price that the OBU determined itself. During

System	Setting	Post-pay-ments	Dynam-ic Pricing	Offline RSUs	No TPM in OBU	Fraud Detec-tion	Black-listing Mecha-nism	Formal Model and Proof	Proto-type Imple-menta-tion
VPriv [DDS11; PBB09]	GNSS	✓	✓	n/a[4]	✓	(✓)[6]	✗	✓	✓
PrETP [Bal+10]	GNSS	✓	(✓)[3]	n/a[4]	✓	(✓)[6]	✗	✓	✓
Milo [Mei+11]	GNSS	✓	✓	n/a[4]	✓	(✓)[6]	✗	✗	✓
[Che+12; Che+13]	GNSS	✓	✓	n/a[4]	✓	(✓)[6]	✗	✓	✗
[GVJ11]	GNSS	✓	✗	n/a[4]	✗	(✓)[6]	✗	✓✗[7]	✗
[KL15]	GNSS	✓	✓	(✓)[5]	✓	✓	✗	✗	✓✗[8]
[Jar+14; Jar+16; JCV14]	DSRC	✗	(✓)[2]	✗	✗	✓	✗	✗	✗
SPEcTRe [Day+11]	DSRC	✓	✗	n/a[4]	✓	(✓)[6]	✗	✗	✓
[Bar+16]	DSRC	✓✗[1]	✓	✗	✓	✓	✓	✗	✓
P4TC	DSRC	✓	✓	(✓)[5]	✓	✓	✓	✓	✓

[1] Pre-payment scheme with refunds
[2] In [JCV14] the pricing is not dynamic, in [Jar+14; Jar+16] it is dynamic
[3] PrETP supports dynamic prices, but for practical reasons the set of possible prices must be kept "small"
[4] Not applicable, since these systems do no have RSUs
[5] RSUs are offline most of the time, they only sometimes need to be online to transmit data to a central server
[6] Fraud detection is probabilistic, so fraud will not always be detected
[7] Informal proof
[8] No implementation is given, put performance is estimated

Table 3.1.: Comparison of electronic toll collection systems

reconciliation with the TSP, users present their total toll and prove that this is indeed the sum of the individual prices they sent, using the homomorphic property of the commitment scheme. The authors prove their system secure using the ideal/real world paradigm.

In [Mei+11], the authors identify large-scale driver collusion as a potential threat to the security of PrETP (and other systems): As spot check locations are leaked to the drivers in the reconciliation phase, they may collude to cheat the system by sharing these locations and only sending correct payment tuples nearby. To this end, the Milo system is constructed. In contrast to PrETP, the location of the spot checks is not revealed during the monthly reconciliation phase. Therefore, drivers are less motivated to collude and cheat. However, if a cheating user is caught, the corresponding spot check location is still revealed. Thus, Milo does not protect against mass-collusion of *dishonest* drivers.

In [Che+12], the authors propose a system based on group signatures that achieves k-anonymity. A user is assigned to a group of drivers during registration. While driving, the user's OBU sends location, time, and group ID—signed using one of the group's signature keys—to the TSP. At the end of a billing period, each user is expected to pay their toll. If the sum of tolls payable by the group (calculated from the group's location-time data) is not equal to the total toll actually paid by the group, a dispute solving protocol is executed. Choosing an appropriate group size is difficult (the larger the size, the better the anonymity, but the computation overhead rises), as well as choosing a suited group division policy (users in a group should have similar driving regions and similar driving patterns). In [Che+13], the system's security and privacy properties are verified using ProVerif.

Another cell-based road pricing scheme is presented in [GVJ11]. Here, a roadpricing area is divided into cells and certain cells randomly selected as spot check cells. A trusted platform module (TPM) inside each OBU is aware of this selection. While driving, the OBU tells its TPM the current location and time. The TPM updates the total toll and generates a "proof of participation" which is sent to the TSP. This proof is the signed and encrypted location-time data under the TSP's public key if the user is inside a spot check cell and 0 otherwise. In this way, the TSP can easily verify that a user behaved honestly at spot check cells without leaking their locations to honest users. A security proof is sketched.

A key issue with all of the above systems is that their security relies on a strong assumption: invisible spot checks at unpredictable locations in each billing period. Otherwise users could easily collude and cheat. On the other hand, spot checks reveal a user's location. Hence, fixing the number of spot checks is a trade-off between ensuring honesty and preserving privacy. Clearly, the penalty a cheater faces influences the number of spot checks required to ensure a certain security level.

In [KL15], the authors argue that even mass surveillance, protecting no privacy at all, cannot prevent collusion under reasonable penalties. They present a protocol for a privacy-preserving spot checking device where the locations of these devices can be publicly known. Drivers interacting with the device identify themselves with a certain probability, but do not learn whether they do so, therefore being forced to honesty. To let users not benefit from turning off their OBUs between two spot check devices, such a device is needed at every toll segment. Furthermore, to encourage interaction with the device, an enforcement camera is required. However, since all these road-side devices are needed anyway, there is no advantage anymore in terms of infrastructure requirements compared to the DSRC setting.

DSRC (OBU/RSU) Setting. Previous work [Day+11; Jar+14; Jar+16; JCV14] in the DSCR setting mainly focus on a pre-pay scenario where some form of e-cash is used to spend coins when passing an RSU. None of them comes with a formal security model and proof.

General problems involved with an e-cash approach are the assurance of sufficient funds and exact payments. The first means that the user or some mechanism needs to ensure that during a trip a user never runs out of e-coins to pay their toll. The problem of "exact payments" is well-known in the e-cash community and led to research on divisible e-cash [BPS19] (so that coins can be split-up appropriately) and transferable e-cash [Bal+15] (so that the payee can return change). However, instantiations of these variants still are significantly more heavyweight than traditional e-cash which hampers the deployment in time-critical settings such as toll collection. Moreover, in the case of transferable e-cash there also is an impossibility result [CG08] negating "perfect anonymity", which translated to the ETC setting means: if the entity issuing e-coins (usually the TSP) colludes with the RSUs, then e-coins given out as change by an RSU to a user can be linked to their spendings at other RSUs. Certainly, this reduces the suitability for an ETC setting.[1]

In [Jar+14; Jar+16; JCV14] multiple electronic road pricing systems specifically tailored for Low Emission Zones (LEZ) are proposed. In [JCV14] a user drives by an RSU and directly pays some price depending on this RSU. In [Jar+14; Jar+16] the user receives some e-ticket from an Entry-RSU when entering the LEZ which they need to present again at an Exit-RSU when leaving the LEZ. For the actual payment in all these systems, some untraceable e-cash scheme that supports dynamic pricing is assumed but not specified. The systems require tamper-proof hardware for the OBU and are claimed to provide fraud protection and privacy for honest drivers.

SPEcTRe [Day+11] is a simple access-based toll collection system, where RSA-based tokens/coins are used to pay toll while driving. Some ideas are presented to keep the double-spending database small.

In [Bar+16], the authors propose an efficient scheme for distance-based toll collection. Here, a user obtains a coin, used as an entry ticket, which is worth the maximum toll in the system and can be reused a fixed number of times. The actual toll is calculated at the Exit-RSU, where a reimbursement is

[1] Note that relaxations of "perfect anonymity" can be achieved [Bal+15]. It would be interesting to see if those allow for practical ETC constructions with reasonable privacy guarantees.

done. Unfortunately, the description of their system misses some important details. For instance, it is unclear how the zero-knowledge proof generated by the user in their token issuance protocol can be efficiently instantiated as it mixes statements and variables over a Paillier ring and a discrete logarithm group. As opposed to our scheme, their system relies on online “over-spending” detection and lacks a formal model and security proof.[2]

3.1.2. Contribution

The contribution of this work is twofold:

Protocol. A major challenge of this work was to select, adapt, and combine the numerous cryptographic building blocks and techniques to design a system satisfying simulation-based security and practicality *at the same time*.

To this end, we started from a payment system building-block called black-box accumulation (BBA+) [Har+17]. BBA+[3] offers the core functionality of an unlinkable user wallet maintaining a balance. Values can be added to and subtracted from the wallet by an operator, where the use of an old wallet is detected offline by a double-spending mechanism.[4] The system guarantees that a wallet may only be used by its legitimate owner and with its legitimate balance.

While BBA+ provides us with some ((P1), (P3), (P4) and (P9) partially) of the desired properties identified in Section 3.2.4, it has not been designed to cope with the real-world issues of an ETC scenario. Thus, we have to enhance BBA+ in several ways to enable its usage in our scenario in the first place.

For instance, consider the case an OBU, which contains the BBA+ wallet, is (claimed to) be broken. As a user collects debt with this wallet and those transactions are perfectly anonymous, the TSP is not able to recalculate what the user owes (P7) nor blacklist the wallet (P6). As this could result in a serious loss of revenue for the TSP, it renders the original BBA+ scheme impractical for an ETC scenario. We solve this by having an individual anonymity revocation trapdoor for each wallet. In case of an incident (like a broken OBU or possibly court-ordered assistance to law enforcement) this makes its transactions forward- and backward-linkable with the help of a trusted *dispute resolver (DR)*. It is important to note that the trapdoor does not allow to link transactions of other wallets. To realize this, we adopt an idea from the e-cash literature [CHL05]. More precisely, we make use of a *pseudo-random function (PRF)* applied to a counter value to generate some fraud detection ID for a wallet state. To ensure security and privacy, we let both the user and the TSP jointly choose the PRF key with the key remaining unknown to the TSP. To make it accessible in case of an incident, the user deposits the key encrypted under the DR’s public key. The correctness of this deposit is ensured by a NIZK proof. This part is tricky due to the use of Groth-Sahai NIZKs for efficiency reasons and the lack of a compatible (i.e., algebraic) encryption scheme with message space $\mathbb{Z}_q$.

As a minor but very useful modification we added (user and RSU) attributes, which get signed along with the wallet to protect from forgery. This allows to bind wallets to a billing period encoded in the attribute. Having RSUs only accept wallets from the current period reduces the size of the blacklist it

[2] For the proofs the authors refer to the full version, which, however, does not seem to be publicly available.

[3] Note that we cannot build on BBW [Hof+20], which is a faster version of BBA+, as we aim for a proof in the UC model which collides with their need to rewind adversaries.

[4] Unfortunately, *online* double-spending detection is not feasible in this scenario (even if we required permanent online connection from RSUs) due to the strong time constraints of open road tolling and potentially simultaneously conducted interactions.

has to check, which enables faster transactions. Similarly, the database needed to recalculate balances can be kept small.

Another problem is the use of a single shared wallet certification key in the BBA+ scheme. Translated to our setting, the TSP and all RSUs would share the same secret key. Hence, if an adversary corrupted a single RSU, they could arbitrarily create new wallets, forge user attributes and balances, etc. In order to mitigate this problem (P8), we take the following measures:

(1) First, we separate user identity and attribute information, i.e., the fixed part of a wallet, from balance information, i.e., the updatable part. The first part is signed by a signature key only held by the TSP when the wallet is issued. The second part is signed by an RSU each time the balance of a wallet is updated. This prevents a corrupted RSU to issue new wallets or fake user attributes.

(2) Furthermore, the individual key of each RSU is certified by the TSP along with its attributes. In this way, a RSU may not forge its attributes (P2) but may still fake the balance.

(3) By including an expiration date into the RSU attributes, one can limit the potential damage involved with the latter issue of (2).

In view of the fact that RSUs are usually not easily accessible physically and key material is usually protected by a hardware security module (HSM), we believe that these measures are sufficient. We intentionally refrain from using key revocation mechanisms like cryptographic accumulators [Lap+10] in order to retain real-time interactions.

Finally, we added mechanisms for a user to prove their participation in certain transactions (P10) and to enable simulation-based security.

System Definition, Security Model and Proof. Having the scenario from Section 3.2 at the back of our mind, we propose a system definition and security model for post-payment toll collection systems based on the UC framework [Can+07]. Our work is one of very few combining a complex, yet practical crypto system with a thorough UC-security analysis.

The security of BBA+ has been modeled by individually formalizing and proving security properties from a list. This approach is common in the game-based setting but bears the intrinsic risk that important but non-obvious security aspects are overlooked, i.e., the list is incomplete. This danger is eliminated by our UC-approach where we do not formalize a list of individual properties but rather what an ideal system looks like.

A challenging task was to find a formalization of such an ideal functionality which accomplishes a reasonable trade-off between various aspects: On one hand, it needs to be sufficiently abstract to represent the semantics of the goal "toll collection" while still admitting a realization. On the other hand, keeping it too close to the concrete realization and declaring aspects as out-of-scope only provides weak security guarantees.

We decided to directly model the ETC system as a single functionality with (polynomially) many parties that reactively participate in (polynomially) many interactions. This leads to a clean interface but makes security proofs highly non-trivial. At first sight, it seems tempting to follow a different approach: Consider the system as a composition of individual two-party protocols, analyze their security separately and deduce the security of the system from the composition theorem. We refrain from this approach, as it entails a slew of technical subtleties due to the shared state between protocols.

Moreover, although our system uses cryptographic building blocks for which UC-formalizations exist (commitments, signatures, NIZK), these abstractions cannot be used. For example, UC-signatures are just random strings that are information-theoretic independent of the message they sign. Thus, it is impossible to prove in zero-knowledge any statement about message-signature-pairs. Hence, our security proof has to start almost from scratch, is very complex, and technically demanding.

3.2. Considered Scenario

This overview of the scenario shows the flexibility and complexity required of our system. We target post-payment toll collection in a DSRC setting which allows access- as well as distance-based charging and can be deployed in a toll-plaza as well as an open-road environment.

3.2.1. Parties

Our scenario involves the following entities:

TSP: A *Toll Collection Service Provider (TSP)* which may be a private company.

SA: A *State Authority (SA)*, e.g., Department of Transportation, which outsourced toll collection to the TSP but is responsible for violation enforcement.

DR: A *Dispute Resolver (DR)*, e.g., a privacy-protecting NGO. It is involved in case of incidents or disputes.[5]

User: Users participate in the system by means of a (portable or mounted) *On-Board Unit (OBU)*. OBUs are used for on-road transactions and in debt and dispute clearance periods. For the latter, it needs to establish a (3G) connection to the TSP/SA. Alternatively, a smartphone might be used for that purpose, which, however, needs access to the OBU.

RSU: *Road-Side Units (RSUs)* interact with OBUs and are typically managed by the TSP. To enable fast and reliable transactions with OBUs, we do *not* require RSUs to have a permanent connection to the TSP. We only assume them to be periodically online for a short duration (e.g., at night when there is not much traffic) to exchange data with the TSP.

Camera: Enforcement Cameras are triggered by the RSUs and used to make photos of license plates (and possibly drivers) *only in case anything goes wrong*. Alternatively, there might be barriers in a toll-plaza environment. Enforcement Cameras are typically owned by the SA.

3.2.2. Main Tasks

In the following, we sketch the main tasks of the system involving the parties from above. Figure 3.1 provides an overview of these interactions. The full description that also includes the remaining tasks can be found in Section 3.4. For simplicity, let us envision a system with monthly billing periods.

[5] Note that we assume the DR to be trusted by all other parties as it implements a key escrow mechanism.

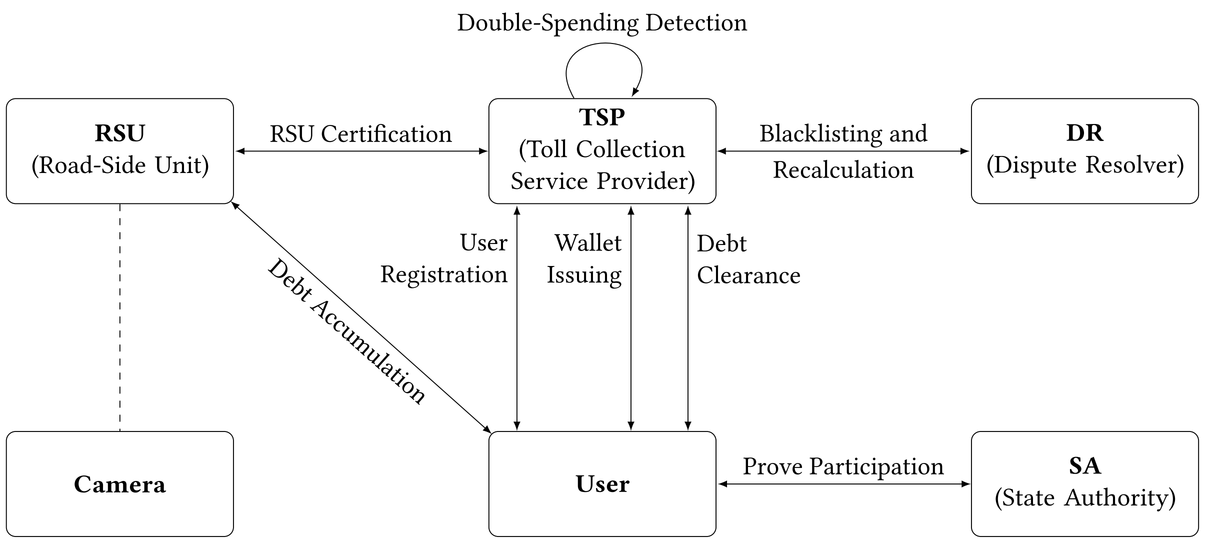


Figure 3.1.: The P4TC system model

User Registration: To participate in the system, a user needs to register with the TSP using some physical ID (e.g., passport, SSN) which is verified out-of-band. This is done once and makes the user accountable in case they cheat or refuse to pay their bills.

RSU Certification: The RSU gets a certificate from the TSP on its public signing key and its RSU attributes. This allows the RSU to later prove that it is a genuine RSU and that its attributes (which can influence the toll price) are correct.

Wallet Issuing: An (empty) wallet is issued to a user by the TSP at the beginning of each billing period. The wallet is bound to the user's ID and attributes in a privacy-preserving manner.

Debt Accumulation: Every time a user passes an RSU, their OBU and the RSU execute the Debt Accumulation task. The due toll is determined and added to the user's wallet—possibly along with public attributes of the RSU. The toll may be dynamic and depend on different factors like the current time of day, congestion level, some user attributes attached to the wallet as well as some attributes of the previous RSU the user drove by. Note that users are anonymous during this task.

The camera takes a photo of the license plate(s) in case the RSU reports any task failure or a car in the range of the RSU refuses to communicate at all. Due to technical limitations, it might be impossible to determine which car triggered the camera, especially in open-road tolling environments [Kap18]. In this case, photos of more than one car in the range of the RSU are taken and transmitted to the SA.

Prove Participation: After the SA has identified the users behind the license plates involved in an incident, it determines who caused the incident. Since we demand that Debt Accumulation transactions are anonymous, the users need to interact with the SA to help with this matching. To this end, an instance of the Prove Participation task is executed between the SA and each of these users consecutively. This prevents honest users who successfully ran Debt Accumulation from being penalized.

Debt Clearance: At the end of a billing period, users participate in an (asynchronous) clearance phase with the TSP. As this task is not anonymous, users can be penalized if they refuse to run the task within a certain time frame. In a task execution, a user presents their wallet and the accumulated debt to the TSP. Then they may clear their debt immediately or within a certain grace period. A successful task execution invalidates the wallet. Users can then get a new wallet by rerunning Wallet Issuing.[6]

Double-spending Detection: As the system is largely offline, a malicious user might re-use an old state of their wallet (e.g., with lower debt) and thus commit double-spending. To mitigate this problem, Wallet Issuing, Debt Accumulation and Debt Clearance generate so-called *double-spending information* that is eventually collected by the TSP. The TSP periodically runs the Double-Spending Detection task on its database to find pairs of matching double-spending information and to identify fraudulent users.

Blacklisting & Recalculation: With the help of the trusted DR, the TSP is able to blacklist fraudulent users and to recalculate what they owe.

3.2.3. Attributes, Pricing Function and Privacy Leakage

Our scenario involves two types of attribute vectors: user attributes and (previous) RSU attributes. To keep our framework flexible, we do not stipulate which kind of attributes are used. Those details depend on the concrete pricing model. We expect, however, that for most scenarios very little information needs to be encoded in these attributes. For instance, access-based toll collection can be realized with prices primarily depending on auxiliary input like location, time and possibly the congestion level—without any previous RSU attributes and only the current billing period as user attribute. Including the previous RSU's attributes into the toll calculation allows for distance-based charging, where RSUs are installed at each entry and exit of a highway. Running Debt Accumulation at the Entry-RSU does not add any debt to the wallet but only encodes the previous RSU's ID as attribute. At the Exit-RSU, the appropriate toll is calculated and added but no RSU attribute is set.[7] To mitigate the damage of a stolen RSU, one might want RSUs to have a common "expiration date" which is periodically renewed and encoded as RSU attribute. Likewise, to enforce that users eventually pay their debts, the user attributes should encode the billing period.[8]

Obviously, the concrete content of the attributes affects the "level" of user privacy in an instantiation of our system. Our goal is to provide provable privacy up to what can be possibly be deduced by an operator who *explicitly* learns (1) those attributes as part of its input to the pricing function and (2) the total debt of a user at the end of a billing period. Our framework guarantees that protocol runs of honest users do not leak anything (useful) beyond that (see Section 3.3.3 and Section 3.5).

In order to allow users to assess the privacy of a particular instantiation of our framework, we assume that all attributes, all possible values for those attributes and how they are assigned, as well as the pricing function are public. In this way, the TSP is also discouraged from running trivial attacks by tampering with an individual's attribute values (e.g., by assigning a billing period value not assigned to any other user). To this end, a user needs to check if the assigned attribute values appear reasonable. Such checks could also be conducted (at random) by a regulatory authority or often also automatically by the user's OBU. Likewise, a (corrupted) RSU could try to break the privacy of a user by charging a peculiar price that differs from the prescribed pricing function. We assume that the user verifies the validity of the price after each transaction and files a claim if the price was computed incorrectly.

[6] If a user plans to be inactive the upcoming billing periods, they would request a new wallet not until they plan to become active again. But this requires to plan inactivity periods in advance. Alternatively, the TSP could be allowed to request the user's trapdoor for such a period from the DR, which would reveal an empty transaction history.

[7] This links entry and exit points. Our system still ensures anonymity and that multiple entry/exit pairs are unlinkable.

[8] Clearly, for privacy reasons, unique expiration dates in attributes need to be avoided.

3.2.4. Desired Security and Privacy Properties

The following list informally summarizes some desirable high-level properties that one would reasonably expect from an electronic toll collection system. These properties inspire the eventual definition of the ideal functionality in Section 3.3. Note that the ideal functionality (and not this list) formally determines the security of our proposed protocol. In Section 3.3.3 we show how these high-level goals are reflected in the ideal functionality.

(P1) Owner-binding: A user may only use a wallet legitimately issued to them.

(P2) Attribute-binding: In Debt Accumulation, a user cannot pretend that they owe less by forging the attributes attached to their wallet.

(P3) Balance-binding: In Debt Clearance, a user cannot claim to owe less than the amount added to their wallet unless they have committed double-spending.

(P4) Double-spending Detection: If a user reuses an old copy of their wallet, they will be identified.[9]

(P5) Participation Enforcement: If a user fails to participate in Debt Accumulation, they will be identified.

(P6) Blacklisting: The TSP is able—with a hint from the DR—to efficiently blacklist wallets of individual users. This is important in practice, e.g., to mitigate the financial loss due to stolen or compromised OBUs or double-spending.

(P7) Debt Recalculation: The TSP is able—with a hint from the DR—to efficiently recalculate the debt for individual users during a billing period. This is important in practice, e.g., to mitigate the financial loss due to broken, stolen, or compromised OBUs. Furthermore, it allows to determine the actual debt of a double-spender. Also, in a dispute, a user may request a detailed invoice listing the toll points visited and the amounts being charged.

(P8) Renegade Expulsion: As an RSU's secrets enable tampering with user debt, there is a mechanism to mitigate financial loss due to compromised RSUs.

(P9) Unlinkability: If user attributes and (previous) RSU attributes are ignored, a collusion of TSP and RSUs may not be able to link a set of Wallet Issue, Debt Accumulation and Debt Clearance transactions of an honest user given that the user is not blacklisted nor committed double-spending. More precisely, Wallet Issuing, Debt Accumulation and Debt Clearance do not reveal any information (except for user attributes, RSU attributes and the final balance in case of Debt Clearance) that may help in linking transactions.

(P10) Participation Provability: Prove Participation enables the SA to deanonymize a single transaction of an honest user in case of an incident.

(P11) Protection Against False Accusation: The user is protected against false accusation of having committed double-spending (cf. (P4)).

[9] Note that this also applies to multiple instances of double-spending. Since the TSP only periodically checks for double-spendings, a user could reuse an old copy of their wallet several times, say 100 times, before the TSP runs the Double-spending Detection task. In that case all 100 double-spendings will be identified and the user can be punished accordingly. Note that hardware cloning is also captured by this property. An adversary cloning their own/stolen hardware (including secrets) may coordinate simultaneous double-spendings with many vehicles. All these double-spendings will be detected and penalized.

3.3. $\mathcal{F}_{\text{P4TC}}$: A Formal Model for Privacy-Preserving ETC

We model our system P4TC within the UC-framework by Canetti [Can01]. We first explain the ideas behind $\mathcal{F}_{\text{P4TC}}$, then informally present the ideal functionality in Section 3.3.1 before giving the full details in Section 3.3.2. We conclude the description of $\mathcal{F}_{\text{P4TC}}$ in Section 3.3.3 by explaining how $\mathcal{F}_{\text{P4TC}}$ fulfills the security properties from Section 3.2.4

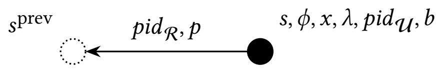
Main Idea: $\mathcal{F}_{\text{P4TC}}$ Models a Transaction Database. The key idea behind $\mathcal{F}_{\text{P4TC}}$ is to keep track of all transactions in a pervasive transaction database *TRDB*. Each entry *trdb* is of the form

$$trdb = (s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b).$$

It contains the *identities* $pid_{\mathcal{U}}$ and $pid_{\mathcal{R}}$ of the involved user and RSU (or TSP) respectively,[10] the *price* p (toll) of this particular transaction and the user wallet's total *balance* b, which is the accumulated sum of all transaction prices up to and including p. In other words, $\mathcal{F}_{\text{P4TC}}$ implements a trustworthy global bookkeeping service managing the wallets of all users. Each transaction entry is uniquely identified by a *serial number* s. Additionally, each entry contains a pointer s^{prev} to the logically previous transaction, a unique *wallet ID* λ, a *counter* x indicating the number of previous transactions with this particular wallet, and a *fraud detection ID* ϕ.

Some explanations are in order with respect to the different IDs. In a truly ideal world, $\mathcal{F}_{\text{P4TC}}$ would use the user identity $pid_{\mathcal{U}}$ to look up its most recent entry in the database and append a new entry. Such a scheme, however, could only be implemented by an online system. Since we require offline capabilities—allowing a user and RSU to interact without the help of other parties and without permanent access to a global database—the inherent restrictions of such a setting must be reflected in the ideal model: (Even formally honest) users can misbehave and commit double-spending without being noticed *instantly* and double-spending is eventually detected after-the-fact.

In order to accurately define security, these technicalities have to be incorporated into $\mathcal{F}_{\text{P4TC}}$, which causes the bookkeeping to be more involved. The transaction database *TRDB* is best depicted as a directed graph, in which each node represents the state of a user's wallet *after* a transaction. Each entry $trdb \in TRDB$ represents a node together with an edge pointing to its predecessor node. Nodes are identified by serial numbers s and additionally labeled with $(\phi, x, \lambda, pid_{\mathcal{U}}, b)$. Edges are given by (s^{prev}, s) and additionally labeled with $(pid_{\mathcal{R}}, p)$. It can be depicted like this:



A user's wallet is represented by the subgraph of all nodes with the same wallet ID λ and forms a connected component. Unless a user commits double-spending the particular subgraph is a linked, linear list. In this case, each transaction entry has a globally unique fraud detection ID ϕ. If a user misbehaves and reuses an old wallet state (i.e., there are edges (s^{prev}, s) and (s^{prev}, s')), the corresponding subgraph becomes a directed tree. In this case, all transaction entries that constitute double-spending, i.e., all nodes with the same predecessor, share the same fraud detection ID ϕ. To consistently manage

[10] The *party identifier (PID)* can best be depicted as the model's counterpart of a party's physical identity in the real world. For users, this could for example be a passport number or SSN. The "identity" of an RSU could be its geo-location. Generally, there is no necessary one-to-one correspondence between a PID and a cryptographic key. Also, the ideal functionality always knows the PID of the party it interacts with by definition of the UC framework.

fraud detection IDs, $\mathcal{F}_{\text{P4TC}}$ uses a counter x and an injective map $f_\Phi : \mathcal{L} \times \mathbb{N}_0 \to \Phi, (\lambda, x) \mapsto \phi$. For any newly issued wallet with ID λ, the counter x starts at zero and $x = x^{\text{prev}} + 1$ always holds. It counts the number of subsequent transactions of a wallet since its generation, i.e., x equals the depth of a node. The function f_Φ maps a transaction to its fraud detection ID ϕ, i.e., $f_\Phi(\lambda, x) = \phi$.

Besides storing transaction data, $\mathcal{F}_{\text{P4TC}}$ also keeps track of parties' attributes by internally storing RSU attributes $\mathbf{a}_{\mathcal{R}}$ upon certification and user (or rather wallet) attributes $\mathbf{a}_{\mathcal{U}}$ when the wallet is issued.

Setup Assumptions and Implicit Writing Conventions. As commonly found in the UC setting, we also draw from setup assumptions. We assume that the functionality $\mathcal{F}_{\text{P4TC}}$ has access to a globally available bulletin board, denoted $\mathcal{G}_{\text{bb}}$ (see Fig. 2.4). We also assume that our functionality uses the implicit writing conventions for ideal functionalities from [Can01]. In particular, our simulator can delay outputs and abort at any point. Beyond that, the simulator has the power to override the output to honest parties with an abort reason (e.g., "blacklisting") if it decides to abort.

3.3.1. Introduction to $\mathcal{F}_{\text{P4TC}}$

Before describing the ideal functionality in full detail in Section 3.3.2, we first give a condensed description here. We explain (a slightly simplified version of) Debt Accumulation in some detail to familiarize the reader with the concept of ideal functionalities and give only short sketches of the remaining tasks.

Individual tasks (e.g., Wallet Issuing, Debt Accumulation etc.) are not formalized as separate functionalities. Instead the whole system is given as one monolithic, highly reactive ideal functionality $\mathcal{F}_{\text{P4TC}}$ with polynomially many parties as users and RSUs. This allows for a shared state between individual interactions. An excerpt of $\mathcal{F}_{\text{P4TC}}$ is depicted in Fig. 3.2. For the ease of presentation, all instructions which are typically executed are printed in normal font, while some conditional side tracks are grayed out. The conditional branches deal with corrupted, misbehaving,[11] or blacklisted parties.

Task Debt Accumulation. In this task (cf. Fig. 3.2), the user provides a serial number s^{prev} as input to $\mathcal{F}_{\text{P4TC}}$, indicating which past wallet state they wish to use for this transaction. Of course, an honest user always uses the wallet state resulting from the previous transaction. The participating RSU inputs a list of fraud detection IDs that are blacklisted.

Firstly, $\mathcal{F}_{\text{P4TC}}$ randomly picks a fresh serial number s for the upcoming transaction. If the user is corrupted, $\mathcal{F}_{\text{P4TC}}$ allows the simulator to provide a different value for $pid_{\mathcal{U}}$ that belongs to a another corrupted user.[12] $\mathcal{F}_{\text{P4TC}}$ looks up the previous wallet state $trdb^{\text{prev}}$ in $TRDB$. The ideal functionality extracts the wallet ID from the previous record ($\lambda \coloneqq \lambda^{\text{prev}}$) and increases the counter for this particular wallet ($x \coloneqq x^{\text{prev}} + 1$). Then, it checks if a fraud detection ID ϕ has already been defined for the wallet ID λ and counter x (note that λ identifies the tree in $TRDB$ and x specifies the depth of the node). If so, the current transaction record will be assigned the same fraud detection ID ϕ. Otherwise, $\mathcal{F}_{\text{P4TC}}$ ties a fresh, uniformly and independently drawn fraud detection ID ($(\lambda, x) \mapsto \phi$) to the x'th transaction of the wallet

[11] Please note, that users do not need to be formally corrupted in order to commit double-spending. We call these users honest, but misbehaving.

[12] This is a required technicality as corrupted users might share their credentials and thus might use each other's wallet. Please note that this does not affect honest users.

Simplified Excerpt of Functionality $\mathcal{F}_{\text{P4TC}}$

Functionality State:

- Set *TRDB* of transaction entries *trdb*. Each entry has the form $trdb = (s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$.
- Mapping $f_{\Phi} : \mathcal{L} \times \mathbb{N}_0 \to \Phi, (\lambda, x) \mapsto \phi$ assigning a fraud detection ID ϕ to a wallet ID λ and counter x.
- Mapping $f_{\mathcal{A}_{\mathcal{U}}} : \mathcal{L} \to \mathcal{A}_{\mathcal{U}}, \lambda \mapsto \mathbf{a}_{\mathcal{U}}$ assigning user attributes to a given wallet ID λ.
- Mapping $f_{\mathcal{A}_{\mathcal{R}}} : \mathcal{PID}_{\mathcal{R}} \to \mathcal{A}_{\mathcal{R}}, pid_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}$ assigning RSU attributes to a given RSU PID $pid_{\mathcal{R}}$.

Debt Accumulation:

- Input User: $(\text{PayToll}, s^{\text{prev}})$
- Input RSU: $(\text{PayToll}, bl_{\mathcal{R}})$
- Behavior:

(1) Pick serial number $s \overset{\text{R}}{\leftarrow} S$ that has not previously been used.

(2) *User corrupted:* Ask the adversary if the PID $pid_{\mathcal{U}}$ of another corrupted user should be used.[a]

(3) Look up $(\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in$ *TRDB*, with $(s^{\text{prev}}, pid_{\mathcal{U}})$ being the unique key.

(4) Adopt previous walled ID $\lambda \coloneqq \lambda^{\text{prev}}$ and increase counter $x \coloneqq x^{\text{prev}} + 1$.

(5) *Double-spending/Blacklisted:* In this case $\phi \coloneqq f_{\Phi}(\lambda, x)$ has already been defined; continue with (6).

Pick fraud detection ID $\phi \overset{\text{R}}{\leftarrow} \Phi$ that has not previously been used.

User corrupted: Allow the adversary to choose a previously unused fraud detection ID ϕ.[b]

Append assignment $(\lambda, x) \mapsto \phi$ to f_{Φ}.

(6) If $\phi \in bl_{\mathcal{R}}$, output BLACKLISTED to both parties and abort.

(7) Look up attributes $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}) \coloneqq (f_{\mathcal{A}_{\mathcal{U}}}(\lambda), f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}), f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}}))$.

(8) Calculate price $p \coloneqq \mathrm{O}_{\text{pricing}}(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$.

RSU corrupted: Leak $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$ to the adversary and obtain a price p.

(9) Calculate new balance $b \coloneqq b^{\text{prev}} + p$.

(10) Append new transaction $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ to *TRDB*.

- Output User: $(s, \mathbf{a}_{\mathcal{R}}, p, b)$
- Output RSU: $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$

[a] If corrupted users collude, they might share their credentials and use each other's wallet.

[b] The ideal model only guarantees privacy for honest users. For corrupted users the fraud detection ID might be chosen adversarially.

Figure 3.2.: An excerpt of the ideal functionality $\mathcal{F}_{\text{P4TC}}$ just with the task Debt Accumulation.

λ. If this fraud detection ID is blacklisted, the task aborts.[13] If and only if the user is corrupted and ϕ has not previously been defined, the adversary is allowed to overwrite the fraud detection ID with another value.[14] Moreover, $\mathcal{F}_{\text{P4TC}}$ looks up the user's attributes bound to this particular wallet ($\mathbf{a}_{\mathcal{U}} \coloneqq f_{\mathcal{A}_{\mathcal{U}}}(\lambda)$) and the attributes of the current and previous RSU ($\mathbf{a}_{\mathcal{R}} \coloneqq f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}})$, $\mathbf{a}_{\mathcal{R}}^{\text{prev}} \coloneqq f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}})$).

Finally, the ideal functionality queries the pricing oracle $\mathrm{O}_{\text{pricing}}$ for the price p of this transaction, calculates the new balance of the wallet ($b \coloneqq b^{\text{prev}} + p$) and appends a new record to the transaction database. If and only if the RSU is corrupted, the adversary learns the involved attributes and is allowed to override the price. Please note that leaking the user/RSU attributes to the adversary does not weaken

[13] Note, that the probability to blacklist a freshly drawn fraud detection ID is negligible. Only if $f_{\Phi}(\lambda, x)$ has already been defined by a past task, this yields a chance to successfully blacklist a user.

[14] Again, this is a technical concession to the security proof. Corrupted users are not obliged to use "good" randomness. This might affect untrackability, but we do not aim to provide this guarantee for corrupted users.

the privacy guarantees as the (corrupted) RSU learns the attributes as an output anyway. The option to manipulate the price on the other hand was a design decision. It was made to enable implementations in which the pricing function is unilaterally evaluated by the RSU and the user initially just accepts the price. It is assumed that a user usually collects any toll willingly in order to proceed and (in case of a dispute) files an out-of-band claim later.

The user's output are the serial number s of the current transaction, the current RSU's attributes $\mathbf{a}_{\mathcal{R}}$, the price p to pay and the updated balance b of their wallet. The RSU's output are the serial number s of the current transaction, the fraud detection ID ϕ and the attributes $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ of the user and the previous RSU, respectively. Learning their mutual attributes is necessary, because RSU and user must evaluate the pricing function themselves in the real protocol without the help of a third party.

Remaining Tasks. There are auxiliary tasks for registration and certification of parties. They are modeled in the obvious way and append the appropriate mappings (e.g., $f_{\mathcal{A}_{\mathcal{R}}}$) with the given attributes for the PIDs.

In *Wallet Issuing* a new wallet ID λ is freshly, uniformly and independently drawn. A new transaction entry for the user is inserted into the database using λ and a zero balance. This entry can be depicted as the root node of a wallet tree.

The task *Debt Clearance* is very similar to Debt Accumulation described above except that the TSP additionally learns the user's party ID $pid_{\mathcal{U}}$ and the wallet balance b. Debt Clearance is identifying for the user to allow the operator to invoice them and check if they (physically) pay the correct amount. Also, the user does not obtain a new serial number from which it follows that this transaction entry becomes a leaf node of the wallet tree.

The task *Blacklisting & Recalculation* is run between the DR and TSP. The TSP inputs the PID of a user it wishes to blacklist and obtains the debt the user owes and a list of past and upcoming serial numbers. For the latter, $\mathcal{F}_{\text{P4TC}}$ draws a sequence of fresh, uniform and independent fraud detection IDs ϕ_i and prefills the mapping f_Φ for all wallets the user owns. This ensures that upcoming transactions use predetermined fraud detection IDs that are actually blacklisted.

The task *Prove Participation* checks if a TRDB record exists for the particular user and serial number.

The task *Double-Spending Detection* checks if the given fraud detection ID exists multiple times in the database, i.e., if double-spending has occurred with this ID. If so, $\mathcal{F}_{\text{P4TC}}$ leaks the identity of the corresponding user to the adversary and asks the adversary to provide an arbitrary bit string that serves as a proof of guilt. $\mathcal{F}_{\text{P4TC}}$ outputs both—the user's identity and the proof—to the TSP and records the proof as valid for the user.

The task *Guilt Verification* checks if the given proof is internally recorded as valid for the particular user.

3.3.2. Full Ideal Functionality

As explained before we define $\mathcal{F}_{\text{P4TC}}$ as a monolithic, reactive functionality with polynomially many parties. This is mainly due to a shared state that the system requires. We will therefore first explain how this state is recorded by $\mathcal{F}_{\mathrm{P4TC}}$ before we go on to describe its behavior in a modular way by explaining each task[15] it provides.

The main feature of $\mathcal{F}_{\mathrm{P4TC}}$ is that it keeps track of all conducted transactions in a global transaction database *TRDB*. Note that in this case by "transaction" we mean every instance of the tasks Wallet Issuing, Debt Accumulation or Debt Clearance, not just Debt Accumulation. Each transaction entry $trdb \in TRDB$ is of the form

$$trdb = (s^{\mathrm{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b).$$

It contains the identities $pid_{\mathcal{U}}$ and $pid_{\mathcal{R}}$ of the involved user and RSU (or TSP in the case of Wallet Issuing and Debt Clearance) respectively, the ID λ of the wallet that was used as well as the price p and total balance b of the wallet state after this transaction. Furthermore, each transaction entry is identified by a unique serial number s and links via s^{prev} to the previous transaction $trdb^{\mathrm{prev}}$ (which corresponds to the wallet state before *trdb*). Lastly, a fraud detection ID ϕ and a counter x are part of the transaction entry. The counter starts at zero for any newly registered wallet and $x = (x^{\mathrm{prev}} + 1)$ always holds. Hence, it is unique across all wallet states belonging to the same wallet λ if and only if no double-spending has been committed with this wallet. The fraud detection ID is constant for each pair (λ, x) of wallet ID and counter instead of being unique for each transaction, but unique for different pairs of wallet ID and counter. Therefore, fraud detection IDs are stored in a partially defined, but one-to-one, mapping $f_{\Phi} : (\mathcal{L} \times \mathbb{N}_0) \rightarrow \Phi$ within $\mathcal{F}_{\mathrm{P4TC}}$.

Remember that *TRDB* is a forest (i.e., a set of trees), with one tree per user and billing period, i.e., each wallet ID λ defines a tree. Full transaction entries *trdb* are only created by instances of Debt Accumulation. Wallet Issuing creates entries of the form

$$(\perp, s, \phi, 0, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, 0, 0)$$

that have no predecessor node identified by s^{prev}, thus creating a new root node. Debt Clearance creates entries of the form

$$(s^{\mathrm{prev}}, \perp, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, -b^{\mathrm{bill}}, 0)$$

that have no own serial number s and thus form a leaf node. Every other task does not alter *TRDB* but only queries it. Although this database and the mapping to fraud detection IDs contains most of the information our toll collection scheme needs, $\mathcal{F}_{\mathrm{P4TC}}$ stores three more partially defined mappings:

- $f_{\mathcal{A}_{\mathcal{U}}} : \mathcal{L} \rightarrow \mathcal{A}_{\mathcal{U}}, \lambda \mapsto \mathbf{a}_{\mathcal{U}}$ maps a wallet ID λ to user attributes $f_{\mathcal{A}_{\mathcal{U}}}$
- $f_{\mathcal{A}_{\mathcal{R}}} : \mathcal{PID}_{\mathcal{R}} \rightarrow \mathcal{A}_{\mathcal{R}}, pid_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}$ maps a RSU PID $pid_{\mathcal{R}}$ to RSU attributes $\mathbf{a}_{\mathcal{R}}$
- $f_{\Pi} : \mathcal{PID}_{\mathcal{U}} \times \Pi \rightarrow \{\mathrm{OK}, \mathrm{NOK}\}, pid_{\mathcal{U}} \times \pi \mapsto \mathrm{OK}/\mathrm{NOK}$ maps a user PID $pid_{\mathcal{U}}$ and a proof of guilt π to a validity bit

The ideal function $\mathcal{F}_{\mathrm{P4TC}}$ provides twelve different tasks in total which we divide up into three categories:

System Setup Tasks: all Registrations and RSU Certification

Basic Tasks: Wallet Issuing, Debt Accumulation and Debt Clearance

Feature Tasks: Prove Participation, Double-Spending Detection, Guilt Verification and Blacklisting & Recalculation

Identifier	Description
$\mathcal{PID}_{\text{corrupt}}$	set of corrupted party identifiers
f_Φ	(partial) mapping assigning a fraud detection ID ϕ to given wallet ID λ and counter x
$f_{\mathcal{A}_\mathcal{U}}$	(partial) mapping assigning user attributes $\mathbf{a}_\mathcal{U}$ to a given wallet ID λ
$f_{\mathcal{A}_\mathcal{R}}$	(partial) mapping assigning RSU attributes $\mathbf{a}_\mathcal{R}$ to a given RSU PID $pid_\mathcal{R}$
f_Π	(partial) mapping assigning a validity bit OK/NOK to a given pair of $pid_\mathcal{U}$ and π

Table 3.2.: Notation that only occurs in the ideal functionality $\mathcal{F}_{\text{P4TC}}$

Identifier	Abstract Domain	Instantiated Domain	Description
pid_{DR}	$\mathcal{PID}_{\text{DR}}$	$\{0,1\}^*$	party identifier of the DR
$pid_\mathcal{T}$	$\mathcal{PID}_\mathcal{T}$	$\{0,1\}^*$	party identifier of the TSP
$pid_\mathcal{R}$	$\mathcal{PID}_\mathcal{R}$	$\{0,1\}^*$	party identifier of a RSU
$pid_\mathcal{U}$	$\mathcal{PID}_\mathcal{U}$	$\{0,1\}^*$	party identifier of a user
pk_{DR}	$\mathcal{PK}_{\text{DR}}$	$G_1^3 \times G_2^3 \times (G_1^2)^{\ell+2} \times (G_2^2)^4 \times (G_2^2)^{\ell+2}$	public DR key
$\mathsf{pk}_\mathcal{T}$	$\mathcal{PK}_\mathcal{T}$	$G_1^{j+3} \times (G_1^3 \times G_2^{y+3}) \times (G_1^3 \times G_2)$	public TSP key
$\mathsf{vk}_\mathcal{R}$	$\mathcal{PK}_\mathcal{R}$	$G_1^3 \times G_2$	public RSU key
$\mathsf{pk}_\mathcal{U}$	$\mathcal{PK}_\mathcal{U}$	G_1	public user key
$\mathbf{a}_\mathcal{U}$	$\mathcal{A}_\mathcal{U}$	G_2^j	user attributes
$\mathbf{a}_\mathcal{R}$	$\mathcal{A}_\mathcal{R}$	G_1^y	RSU attributes
$\mathbf{a}_\mathcal{T}$	$\mathcal{A}_\mathcal{R}$	G_1^y	TSP attributes
b	$\mathbb{Z}_q$	$\mathbb{Z}_q$	balance
p	$\mathbb{Z}$	$\mathbb{Z}$	price to pay at an RSU
ϕ	Φ	G_1	fraud detection ID
s	S	G_1	serial number
λ	$\mathcal{L}$	$\mathbb{Z}_q$	wallet ID; is used as PRF seed
x	$\mathbb{N}_0$	$\{0,\ldots,\lambda_{\text{PRF}}\}$	(PRF) counter
$bl_\mathcal{R}$	list of Φ elements	list of G_1 elements	RSU blacklist
$x_{\text{bl}_\mathcal{R}}$	$\mathbb{N}$	$\mathbb{N}$	RSU blacklist parameter
$bl_\mathcal{T}$	list of $\mathcal{PK}_\mathcal{U}$ elements	list of G_1 elements	TSP blacklist

Table 3.3.: Notation that occurs in the ideal functionality $\mathcal{F}_{\text{P4TC}}$ and in the real protocol Π_{P4TC}

For better clarity of the following task descriptions, an overview of the variables used can be found in Table 3.2 and Table 3.3. The full ideal functionality $\mathcal{F}_{\text{P4TC}}$ is presented in Fig. 3.3, Fig. 3.4, and Fig. 3.5 and we will now explain each task in turn.

[15] Note that we are intentionally avoiding the word "phase"—which is commonly used in other composite functionalities—as it suggests a predefined order/number of executions.

3.3.2.1. System Setup Tasks

To set up the system two things are required: All parties—the DR, TSP, RSUs and users—have to register public keys with the bulletin board $\mathcal{G}_{\mathrm{bb}}$ to be able to participate in the toll collection system. As all of these registration tasks are similar, we will not describe them separately. In the special case of RSUs a certification conducted with the TSP also needs to take place.

Registrations. The tasks of DR, RSU and User Registration (cp. Fig. 3.3) are straightforward and analogous. They do not take any input apart from “Register”, but in the case of the user we assume the physical identity of the party has been verified out-of-band before this task is conducted. In each case a check is performed first whether the task has been run before for this party. If this does not lead to an abort, the adversary is asked to provide a public key pk for the respective party which is then registered with the bulletin board $\mathcal{G}_{\mathrm{bb}}$ and output to the newly registered party.

The registration of the TSP is slightly different (cp. Fig. 3.3). In addition to “Register” it takes an attribute vector $\mathbf{a}_{\mathcal{T}}$ as input, which—after a check if this task has been run before—is sent to the adversary together with $pid_{\mathcal{T}}$ when the public key $\mathrm{pk}_{\mathcal{T}}$ is obtained. In addition, all data structures of $\mathcal{F}_{\mathrm{P4TC}}$ are initialized as empty sets and empty (partial) mappings respectively. Again, the public key $\mathrm{pk}_{\mathcal{T}}$ is output to the TSP.

RSU Certification. RSU certification (cp. Fig. 3.3) is a two-party task between the TSP and an RSU in which the RSU is assigned an attribute vector $\mathbf{a}_{\mathcal{R}}$.[16] The content of attribute vectors is in no way restricted by $\mathcal{F}_{\mathrm{P4TC}}$ and can be used to implement different scenarios like location-based toll collection or entry/exit toll collection, but could also be maliciously used to void unlinkability. The attributes $\mathbf{a}_{\mathcal{R}}$ are input by the TSP, while the RSU only inputs its desire to be certified. $\mathcal{F}_{\mathrm{P4TC}}$ checks if there have already been attributes assigned to the RSU previously (in which case it aborts) and otherwise appends $f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}) \coloneqq \mathbf{a}_{\mathcal{R}}$ to the partial mapping $f_{\mathcal{A}_{\mathcal{R}}}$ which internally stores all RSU attributes already assigned. The identity $pid_{\mathcal{R}}$ and attributes $\mathbf{a}_{\mathcal{R}}$ are sent to the adversary before the attributes are output to the RSU.

3.3.2.2. Basic Tasks

Now we describe the basic tasks you would expect from any toll collection scheme. These tasks are *Wallet Issuing*, *Debt Accumulation*, and *Debt Clearance*. As mentioned before, those are the only tasks in which transaction entries are created.

Wallet Issuing. Wallet Issuing (cp. Fig. 3.4) is a two-party task between a user and the TSP in which a new and empty wallet is created for the user. The TSP inputs an attribute vector $\mathbf{a}_{\mathcal{U}}$ and a blacklist $bl_{\mathcal{T}}$ of user public keys that are not allowed to obtain any new wallets. First, $\mathcal{F}_{\mathrm{P4TC}}$ randomly picks a (previously unused) serial number s for the new transaction entry $trdb$. If the user is corrupted, the adversary may at this point choose another corrupted user's identity $pid_{\mathcal{U}}$ that is to be used for this wallet. Multiple corrupted users are allowed to have wallets issued for one another but are not able to request a new wallet for an honest user. The corresponding public key for the user ID $pid_{\mathcal{U}}$ is obtained

[16] Although only attributes are set in this task, we will later see in the task of Debt Accumulation that these also serve as a kind of certificate, as RSUs are only able to successfully participate in Debt Accumulation if they have been assigned attributes by the TSP.

Functionality $\mathcal{F}_{\text{P4TC}}$ (Part 1)

Functionality State:

- Set $TRDB = \{trdb\}$ of transactions

$$\begin{aligned} trdb = (s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b) \\ \in S \times S \times \Phi \times \mathbb{N}_0 \times \mathcal{L} \times \mathcal{PID}_{\mathcal{U}} \\ \times \mathcal{PID}_{\mathcal{R}} \times \mathbb{Z}_q \times \mathbb{Z}_q \end{aligned}$$

- A (partial) mapping f_Φ giving the fraud detection ID ϕ corresponding to given wallet ID λ and counter x:

$$f_\Phi : \mathcal{L} \times \mathbb{N}_0 \to \Phi, (\lambda, x) \mapsto \phi$$

- A (partial) mapping $f_{\mathcal{A}_\mathcal{U}}$ assigning user attributes to a given wallet ID λ:

$$f_{\mathcal{A}_\mathcal{U}} : \mathcal{L} \to \mathcal{A}_\mathcal{U}, \lambda \mapsto \mathrm{a}_\mathcal{U}$$

- A (partial) mapping $f_{\mathcal{A}_\mathcal{R}}$ assigning RSU attributes to a given RSU PID $pid_\mathcal{R}$:

$$f_{\mathcal{A}_\mathcal{R}} : \mathcal{PID}_\mathcal{R} \to \mathcal{A}_\mathcal{R}, pid_\mathcal{R} \mapsto \mathrm{a}_\mathcal{R}$$

- A (partial) mapping f_Π assigning a user PID $pid_\mathcal{U}$ and a proof of guilt π to a validity bit:

$$f_\Pi : \mathcal{PID}_\mathcal{U} \times \Pi \to \{\text{OK}, \text{NOK}\}$$

DR Registration:

- Input DR: (Register)
- Behavior:
  (1) If this task has been run before, then output $\bot$ and abort.
  (2) Send (RegisteringDR, pid_{DR}) to the adversary and obtain the key (pk_{DR}).[a]
  (3) Call $\mathcal{G}_{\text{bb}}$ with input (Register, pk_{DR}).
- Output DR: (pk_{DR})

TSP Registration:

- Input TSP: (Register, $\mathrm{a}_\mathcal{T}$)
- Behavior:
  (1) If this task has been run before, then output $\bot$ and abort.
  (2) $TRDB := \emptyset$
  (3) Send (RegisteringTSP, $pid_\mathcal{T}$, $\mathrm{a}_\mathcal{T}$) to the adversary and obtain the key ($\mathrm{pk}_\mathcal{T}$).[a]
  (4) Call $\mathcal{G}_{\text{bb}}$ with input (Register, $\mathrm{pk}_\mathcal{T}$).
- Output TSP: ($\mathrm{pk}_\mathcal{T}$)

RSU Registration:

- Input RSU: (Register)
- Behavior:
  (1) If this task has been run before for that RSU, then output $\bot$ and abort.
  (2) Send (RegisteringRSU, $pid_\mathcal{R}$) to the adversary and obtain the key ($\mathrm{vk}_\mathcal{R}$).[a]
  (3) Call $\mathcal{G}_{\text{bb}}$ with input (Register, $\mathrm{vk}_\mathcal{R}$).
- Output RSU: ($\mathrm{vk}_\mathcal{R}$)

User Registration:

- Input User: (Register)
- Behavior:
  (1) If this task has been run before for that user, then output $\bot$ and abort.
  (2) Send (RegisteringU, $pid_\mathcal{U}$) to the adversary and obtain the key ($\mathrm{pk}_\mathcal{U}$).[a]
  (3) Call $\mathcal{G}_{\text{bb}}$ with input (Register, $\mathrm{pk}_\mathcal{U}$).
- Output User: ($\mathrm{pk}_\mathcal{U}$)

RSU Certification:

- Input RSU: (Certify)
- Input TSP: (Certify, $\mathrm{a}_\mathcal{R}$)
- Behavior:
  (1) If $f_{\mathcal{A}_\mathcal{R}}(pid_\mathcal{R})$ is already defined, then output $\bot$ to both parties and abort; else append $f_{\mathcal{A}_\mathcal{R}}(pid_\mathcal{R}) := \mathrm{a}_\mathcal{R}$ to $f_{\mathcal{A}_\mathcal{R}}$.
  (2) Send (CertifyingRSU, $pid_\mathcal{R}$, $\mathrm{a}_\mathcal{R}$) to the adversary.
- Output RSU: ($\mathrm{a}_\mathcal{R}$)
- Output TSP: (OK)

[a] Giving the adversary the power to control the key generation serves two purposes: (1) It gives a stronger security guarantee, i.e., security for honest parties is retained, even if its keys are maliciously generated (due to bad random number generator). (2) It gives the simulator the lever to simulate faithfully.

Figure 3.3.: The ideal functionality $\mathcal{F}_{\text{P4TC}}$ (System Setup Tasks)

from the bulletin board $\mathcal{G}_{\text{bb}}$ and checked against the TSP's blacklist $bl_\mathcal{T}$. If this does not lead to an abort, a new wallet ID λ and fraud detection ID ϕ are uniquely and randomly picked, unless the user is corrupted in which case the adversary chooses ϕ. This may infringe upon the unlinkability of the user's transactions and we do not give any privacy guarantees for corrupted users. Finally, a transaction entry

$$trdb := (\bot, s, \phi, 0, \lambda, pid_\mathcal{U}, pid_\mathcal{T}, 0, 0)$$

Functionality $\mathcal{F}_{\text{P4TC}}$ (Part 2)

Wallet Issuing:

- Input User: (Issue)
- Input TSP: (Issue, $\mathbf{a}_{\mathcal{U}}$, $bl_{\mathcal{T}}$)
- Behavior:
  1. Pick serial number $s \xleftarrow{R} S$ that has not previously been used.
  2. If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then leak $(s, \mathbf{a}_{\mathcal{U}})$ to the adversary, and ask if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
  3. Receive $\text{pk}_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.$^{\perp}$
  4. If $\text{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$, then output blacklisted to both parties and abort.
  5. Pick wallet ID $\lambda \xleftarrow{R} \mathcal{L}$ that has not previously been used.
  6. If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$, then pick $\phi \xleftarrow{R} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID ϕ that has not previously been used.[b] Append $f_{\Phi}(\lambda, 0) := \phi$ to f_{Φ}.
  7. Append $trdb := (\perp, s, \phi, 0, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, 0, 0)$ to $TRDB$.
  8. Append $f_{\mathcal{A}_{\mathcal{U}}}(\lambda) := \mathbf{a}_{\mathcal{U}}$ to $f_{\mathcal{A}_{\mathcal{U}}}$.
- Output User: $(s, \mathbf{a}_{\mathcal{U}})$
- Output TSP: (s)

Debt Accumulation:

- Input User: (PayToll, s^{prev})
- Input RSU: (PayToll, $bl_{\mathcal{R}}$)
- Behavior:
  1. Pick serial number $s \xleftarrow{R} S$ that has not previously been used.
  2. If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then ask the adversary, if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
  3. Select $(\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in TRDB$ (with s^{prev}, $pid_{\mathcal{U}}$ being the unique key)$^{\perp}$.
  4. Set $\lambda := \lambda^{\text{prev}}$ and $x := x^{\text{prev}} + 1$.
  5. If $f_{\Phi}(\lambda, x)$ is already defined, then set $\phi := f_{\Phi}(\lambda, x)$. Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\phi \xleftarrow{R} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID ϕ that has not previously been used.[c] Append $f_{\Phi}(\lambda, x) := \phi$ to f_{Φ}.
  6. If $\phi \in bl_{\mathcal{R}}$, then output blacklisted to both parties and abort.
  7. Set $\mathbf{a}_{\mathcal{U}} := f_{\mathcal{A}_{\mathcal{U}}}(\lambda)$, $\mathbf{a}_{\mathcal{R}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}})$, and $\mathbf{a}_{\mathcal{R}}^{\text{prev}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}})$.$^{\perp}$
  8. Calculate price $p := \mathrm{O}_{\text{pricing}}(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$. If $pid_{\mathcal{R}} \in \mathcal{PID}_{\text{corrupt}}$, then leak $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$ to the adversary and obtain a price p.
  9. $b := b^{\text{prev}} + p$.
  10. Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ to $TRDB$.
- Output User: $(s, \mathbf{a}_{\mathcal{R}}, p, b)$
- Output RSU: $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$

Debt Clearance:

- Input User: (ClearDebt, s^{prev})
- Input TSP: (ClearDebt)
- Behavior:
  1. Pick serial number $s \xleftarrow{R} S$ that has not previously been used.
  2. If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then ask the adversary, if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
  3. Receive $\text{pk}_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.$^{\perp}$
  4. Select $(\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in TRDB$ (with s^{prev}, $pid_{\mathcal{U}}$ being the unique key).$^{\perp}$
  5. Set $\lambda := \lambda^{\text{prev}}$ and $x := x^{\text{prev}} + 1$.
  6. If $f_{\Phi}(\lambda, x)$ is already defined, then set $\phi := f_{\Phi}(\lambda, x)$. Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\phi \xleftarrow{R} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID ϕ that has not previously been used.[c] Append $f_{\Phi}(\lambda, x) := \phi$ to f_{Φ}.
  7. If $pid_{\mathcal{T}} \in \mathcal{PID}_{\text{corrupt}}$, then set $\mathbf{a}_{\mathcal{R}}^{\text{prev}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}})^{\perp}$, and leak $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ to the adversary.
  8. $b^{\text{bill}} := b^{\text{prev}}$.
  9. Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, -b^{\text{bill}}, 0)$ to $TRDB$.
- Output User: (b^{bill})
- Output TSP: $(\text{pk}_{\mathcal{U}}, \phi, b^{\text{bill}})$

$^{\perp}$If this does not exist, then output $\perp$ and abort.

[a] If a group of corrupted users collude, a correct mapping to a specific users cannot be guaranteed, because corrupted users might share their credentials.

[b] Picking the upcoming fraud detection ID randomly asserts untrackability for honest users. For corrupted user, we do not (and cannot) provide such a guarantee.

[c] The ideal model only guarantees privacy for honest users. For corrupted users the fraud detection ID might be chosen adversarially (cp. text body).

Figure 3.4.: The ideal functionality $\mathcal{F}_{\text{P4TC}}$ (Basic Tasks)

corresponding to the new and empty wallet is stored in *TRDB* and the wallet's attributes $f_{\mathcal{A}_\mathcal{U}}(\lambda) := \mathbf{a}_\mathcal{U}$ are appended to the partial mapping $f_{\mathcal{A}_\mathcal{U}}$. Both parties get the serial number s as output; the user also receives the attribute vector $\mathbf{a}_\mathcal{U}$ to check this has been assigned correctly and more importantly does not contain any identifying information.

Debt Accumulation. This two-party task (cp. Fig. 3.4) is conducted whenever a registered user passes an RSU and it serves the main purpose of adding toll to a previous wallet state of the user. In this task, the user only inputs a serial number s^{prev}, indicating which past wallet state shall be used for this transaction. The participating RSU in turn inputs a blacklist $bl_\mathcal{R}$ of fraud detection IDs. First, $\mathcal{F}_{\text{P4TC}}$ randomly picks a (previously unused) serial number s for the new transaction entry *trdb*. If the user is corrupted, the adversary may at this point choose another corrupted user's identity $pid_\mathcal{U}$ that is to be used for this transaction. $\mathcal{F}_{\text{P4TC}}$ looks up if a wallet state $trdb^{\text{prev}}$ in *TRDB* corresponds to the user input s^{prev} and belongs to the users $pid_\mathcal{U}$. This guarantees that each user can only accumulate debt on a wallet that was legitimately issued to them. Multiple corrupted users may choose to swap wallets between them but are not able to use an honest user's wallet. The ideal functionality uses part of the information from the previous wallet state

$$trdb^{\text{prev}} = (\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda, pid_\mathcal{U}, pid_\mathcal{R}^{\text{prev}}, \cdot, b^{\text{prev}})$$

to determine the content of the new transaction entry *trdb*. The user ID $pid_\mathcal{U}$ and wallet ID λ stay the same, $pid_\mathcal{R}$ is set to the identity of the participating RSU, and the counter x^{prev} is increased by one to obtain x. $\mathcal{F}_{\text{P4TC}}$ checks if there is already a fraud detection ID $\phi := f_\Phi(\lambda, x)$ assigned to the pair (λ, x) (either because the user committed double-spending or because it has been precalculated for blacklisting purposes). If not and the user is honest, it picks a new ϕ uniquely at random. If the user is corrupted, the fraud detection ID is not randomly drawn but picked by the adversary. This may infringe upon the unlinkability of the user's transactions, but, as mentioned before, we do not give any privacy guarantees for corrupted users. The fraud detection ID ϕ is checked against $bl_\mathcal{R}$.[17] The attributes $\mathbf{a}_\mathcal{U}$, $\mathbf{a}_\mathcal{R}$ and $\mathbf{a}_\mathcal{R}^{\text{prev}}$ are again looked up internally and sent to the adversary who chooses the price p of this transaction. Having the price determined in this way makes it clear that $\mathcal{F}_{\text{P4TC}}$ does not give any guarantees on the "right" amount of debt being added at this point. Instead, it gives the user enough information about the transaction to appeal out-of-band afterwards if the wrong amount of debt is added. We assume this detectability will keep RSUs in the real world from adding too much debt. Finally, the new balance b is calculated from the price and old balance. This leads to the new entry

$$trdb := (s^{\text{prev}}, s, \phi, x := x^{\text{prev}} + 1, \lambda, pid_\mathcal{U}, pid_\mathcal{R}, p, b := b^{\text{prev}} + p).$$

being stored in *TRDB*. Note that all information leading to the new wallet state came from data internally stored in $\mathcal{F}_{\text{P4TC}}$ itself, not from an input by the user or RSU, and can therefore not be compromised. The serial number, RSU attributes, price and balance are output to the user so that they can check that they paid the correct toll price. The RSU gets the serial number as well but also the fraud detection ID (to enable double-spending detection) and the attributes of the user and previous RSU.

Debt Clearance. As Debt Clearance (cp. Fig. 3.4) is very similar to the task of Debt Accumulation, we will refrain from describing it again in full detail but rather just highlight the differences to Debt Accumulation. The first difference is that it is conducted with the TSP rather than an RSU and no

[17] Note, that the probability to blacklist a freshly drawn fraud detection ID is negligible. Only if $f_\Phi(\lambda, x)$ has already been defined by a past task, this yields a chance to successfully blacklist a user.

blacklist is taken as input as we do not want to prevent anyone from paying their debt. Although this task results in a transaction entry $trdb$ as well, no new serial number s is picked. This emphasizes that the new wallet state is final and can not be updated again by using its serial number as input for another transaction. Instead of obtaining a price from the adversary, the attributes $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ of the previous RSU $pid_{\mathcal{R}}^{\text{prev}}$ are leaked to the adversary in case the TSP is corrupted. The (negative) price for the transaction entry is set to the billing amount b^{bill} which in turn is taken to be the previous balance b^{prev} of the wallet. As new transaction entry

$$trdb \coloneqq (s^{\text{prev}}, \bot, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, -b^{\text{bill}}, 0)$$

is added to $TRDB$ and the billing amount b^{bill} output to both parties. Furthermore, the TSP gets the user's ID $pid_{\mathcal{U}}$, as we assume Debt Clearance to be identifying, as well as the fraud detection ID ϕ to enable double-spending detection.

3.3.2.3. Feature Tasks

To obtain a more secure toll collection system we also provide the feature tasks *Prove Participation*, *Double-Spending Detection*, *Guilt Verification*, and *Blacklisting & Recalculation*. All of those tasks deal with different aspects arising from fraudulent user behavior.

Prove Participation. This is a two-party task involving a user and the SA (cp. Fig. 3.5) and assumed to be conducted with every user that has been physically caught by one of the SA's cameras. It allows an honest user to prove their successful participation in a transaction with the RSU where the photo was taken, while a fraudulent user will not be able to do so. The SA inputs the public key $\mathsf{pk}_{\mathcal{U}}$ of the user who is asked to prove their participation and a set $S_{\mathcal{R}}^{\text{pp}}$ of serial numbers in question. The user has no explicit input but simply expresses their consent by running the protocol.

First note, there is no guarantee that the user who participates in the protocol (with PID $pid_{\mathcal{U}}$) is the user the SA wants to prove their participation (with user key $\mathsf{pk}_{\mathcal{U}}$ and $pid'_{\mathcal{U}}$). Nonetheless, the ideal functionality checks if $TRDB$ contains a transaction for the requested PID $pid'_{\mathcal{U}}$ and a serial number s in $S_{\mathcal{R}}^{\text{pp}}$. Hence, the ideal functionality ensures that either the SA receives the correct answer—even for corrupted users—or aborts.

Some remarks are in order to the abort condition—which is only a technical concession to what can be practically realized. If $pid'_{\mathcal{U}} = pid_{\mathcal{U}}$ holds (i.e., the SA communicates with the expected user), everything is fine. If $pid'_{\mathcal{U}} \neq pid_{\mathcal{U}}$ holds (i.e., the SA communicates with the wrong user) and at least one of the users is honest, the ideal functionality aborts. This models the fact that a malicious user must neither be able to embody an honest user nor an honest user embodies a malicious user. But if $pid'_{\mathcal{U}} \neq pid_{\mathcal{U}}$ holds (i.e., the SA communicates with the wrong user) and both users are corrupted, the ideal functionality proceeds normally, but guarantees to return the correct result for the user in question (i.e., with $pid'_{\mathcal{U}}$). This models the limitation that two corrupted users can share their credentials. A corrupted user (with $pid'_{\mathcal{U}}$) can pass their transaction information to another corrupted user (with $pid_{\mathcal{U}}$) who then proves participation to the SA. However, the latter user can still only prove the participation of the original user and not misuse the information to falsely prove participation for themselves.

If the user is corrupted, the set of serial numbers in question is sent to the adversary. Note further that this task also deanonymizes the one transaction proven by the user and leaks the respective serial number. Although the SA only obtains a single bit of information, whether the user's serial number is a member of the set $S_{\mathcal{R}}^{\text{pp}}$ or not, this single bit of information is sufficient to restore the complete serial

Functionality $\mathcal{F}_{\text{P4TC}}$ (Part 3)

Prove Participation:

- Input User: (ProveParticipation)
- Input SA: (ProveParticipation, $\text{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}$)
- Behavior:
  1. Obtain $pid'_{\mathcal{U}}$ for $\text{pk}_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$.[a]
  2. If $(pid'_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}} \vee pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}) \wedge pid'_{\mathcal{U}} \neq pid_{\mathcal{U}}$, then abort.
  3. If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then leak $S^{\text{pp}}_{\mathcal{R}}$ to the adversary.
  4. If $\exists\ (\cdot, s, \cdot, \cdot, \cdot, pid'_{\mathcal{U}}, \cdot, \cdot, \cdot) \in \mathit{TRDB}$ such that $s \in S^{\text{pp}}_{\mathcal{R}}$,
     then $\text{OUT}_{\mathcal{U}} \coloneqq \text{OUT}_{SA} \coloneqq \text{OK}$
     else $\text{OUT}_{\mathcal{U}} \coloneqq \text{OUT}_{SA} \coloneqq \text{NOK}$.
- Output User: ($\text{OUT}_{\mathcal{U}}$)
- Output SA: (OUT_{SA})

Double-Spending Detection:

- Input TSP: (ScanForFraud, ϕ)
- Behavior:
  1. Pick $\mathit{trdb} \neq \mathit{trdb}'$ in TRDB such that $\mathit{trdb} = (\cdot, \cdot, \phi, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, \cdot, \cdot)$ and $\mathit{trdb}' = (\cdot, \cdot, \phi, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, \cdot, \cdot)$.$^{\perp}$
  2. Ask the adversary for a proof $\pi \in \Pi$ corresponding to $pid_{\mathcal{U}}$ and append $(pid_{\mathcal{U}}, \pi) \mapsto \text{OK}$ to f_{Π}.
  3. Receive $\text{pk}_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{U}}{}^{\perp}$.
- Output TSP: $(\text{pk}_{\mathcal{U}}, \pi)$

Guilt Verification:

- Input some party P: (VerifyGuilt, $\text{pk}_{\mathcal{U}}, \pi$)
- Behavior:
  1. Receive $pid_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for key $\text{pk}_{\mathcal{U}}$.$^{\perp}$
  2. If $f_{\Pi}(pid_{\mathcal{U}}, \pi)$ is defined, then set $\text{OUT} \coloneqq f_{\Pi}(pid_{\mathcal{U}}, \pi)$ and output (OUT).
  3. If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then leak $(pid_{\mathcal{U}}, \pi)$ to the adversary and obtain result OUT, else set $\text{OUT} \coloneqq \text{NOK}$.
  4. Append $(pid_{\mathcal{U}}, \pi) \mapsto \text{OUT}$ to f_{Π}.
- Output P: (OUT)

Blacklisting & Recalculation:

- Input DR: (BlacklistUser, $\text{pk}^{\text{DR}}_{\mathcal{U}}$)
- Input TSP: (BlacklistUser, $\text{pk}^{\mathcal{T}}_{\mathcal{U}}$)
- Behavior:
  1. If $\text{pk}^{\text{DR}}_{\mathcal{U}} \neq \text{pk}^{\mathcal{T}}_{\mathcal{U}}$, then abort.
  2. Receive $pid_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for $\text{pk}^{\mathcal{T}}_{\mathcal{U}}$.$^{\perp}$
  3. Distinguish 3 cases:
     a. *Case* $pid_{\mathcal{T}} \notin \mathcal{PID}_{\text{corrupt}}$:
        Set $\mathcal{L}_{\text{bl}} \coloneqq \{\lambda \mid (\cdot, \cdot, \cdot, \cdot, \lambda, pid_{\mathcal{U}}, \cdot, \cdot, \cdot) \in \mathit{TRDB}\}$.
     b. *Case* $pid_{\mathcal{T}} \in \mathcal{PID}_{\text{corrupt}} \wedge pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$:
        Obtain a set of serial numbers S_{root} from the adversary and set $\mathcal{L}_{\text{bl}} \coloneqq \{\lambda \mid (\perp, s, \cdot, \cdot, \lambda, pid_{\mathcal{U}}, \cdot, \cdot, \cdot) \in \mathit{TRDB}$ with $s \in S_{\text{root}}\}$.
     c. *Case* $pid_{\mathcal{T}} \in \mathcal{PID}_{\text{corrupt}} \wedge pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$:
        Let the adversary decide on the output for DR and TSP and stop.
  4. $\mathit{TRDB}_{\text{bl}} \coloneqq \{\mathit{trdb} \in \mathit{TRDB} \mid \mathit{trdb} = (\cdot, \cdot, \cdot, \cdot, \lambda, \cdot, \cdot, p, \cdot) \text{ s. t. } \lambda \in \mathcal{L}_{\text{bl}}\}$
  5. $b^{\text{bill}} \coloneqq \sum_{\mathit{trdb} \in \mathit{TRDB}_{\text{bl}}} p$.
  6. For each $\lambda \in \mathcal{L}_{\text{bl}}$:
     a. $x_{\lambda} \coloneqq \max\{\, x \mid f_{\Phi}(\lambda, x) \text{ is already defined}\}$.
     b. For $x \in \{x_{\lambda} + 1, \dots, x_{\text{bl}_{\mathcal{R}}}\}$:
        i. If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$, then pick $\phi \xleftarrow{\text{R}} \Phi$ that has not previously been used, otherwise leak (λ, x) to the adversary and obtain fraud detection ID ϕ that has not previously been used.
        ii. Append $(\lambda, x) \mapsto \phi$ to f_{Φ}.
  7. $\Phi_{\mathcal{U}} \coloneqq \{f_{\Phi}(\lambda, x) \mid \lambda \in \mathcal{L}_{\text{bl}}, 0 \leq x \leq x_{\text{bl}_{\mathcal{R}}}\}$.
- Output DR: (OK)
- Output TSP: $(b^{\text{bill}}, \Phi_{\mathcal{U}})$

$^{\perp}$If this does not exist, then output $\perp$ and abort.

[a] $pid_{\mathcal{U}}$ is the implicit PID of the user participating in the protocol. $pid_{\mathcal{U}}$ is not necessarily equal to $pid'_{\mathcal{U}}$ which denotes the user that is expected by the SA.

Figure 3.5.: The ideal functionality $\mathcal{F}_{\text{P4TC}}$ (Feature Tasks)

number by means of a bi-sectional search. The SA could repeatedly run the task and summon the user to prove its participation for a descending sequence of bi-sected sets until the last set only contains a single serial number. Nonetheless, this does not effect the anonymity or unlinkability of any other transactions.

Double-Spending Detection and Guilt Verification. Due to our requirement to allow offline RSUs, users are able to fraudulently collect debt on outdated states of their wallets. This double-spending can not be prevented but must be detected afterwards. To ensure this, $\mathcal{F}_{\mathrm{P4TC}}$ provides the tasks Double-Spending Detection (cp. Fig. 3.5) and Guilt Verification (cp. Fig. 3.5).

Double-Spending Detection is a one-party task performed by the TSP. It takes a fraud detection ID ϕ as input and checks the transaction database *TRDB* for two distinct entries containing this same fraud detection ID. In case such entries are present, the adversary is asked for a proof π to be issued for this instance of double-spending. The user ID and proof $(pid_{\mathcal{U}}, \pi)$ are appended to f_{Π} and marked as valid. Additionally, both are output to the TSP.

Guilt Verification is a one-party task as well but can be performed by any party. It takes a user ID $pid_{\mathcal{U}}$ and a double-spending proof π as input. First, it checks if this particular pair $(pid_{\mathcal{U}}, \pi)$ has already been defined and outputs whatever has been output before. This is necessary to ensure consistency across different invocations. If $(pid_{\mathcal{U}}, \pi)$ has neither been issued nor queried before *and* the affected user is corrupted, the adversary is allowed to decide if this proof should be accepted. This implies that we do not protect corrupted users from false accusations of guilt. If the user is honest and $(pid_{\mathcal{U}}, \pi)$ has neither been issued nor queried before, then the proof is marked as invalid. This protects honest users from being accused by made-up proofs which have not been issued by the ideal functionality itself. Finally, the result is recorded for the future and output to the party. This possibility of public verification is vital to prevent the TSP from wrongly accusing any user of double-spending and should for instance be utilized by the DR before it agrees to blacklist and therefore deanonymize a user on the basis of double-spending.

Blacklisting & Recalculation. Blacklisting & Recalculation (cp. Fig. 3.5) is a two-party task between the DR and TSP and serves two purposes: First, the debt b^{bill} owed by the user that is to be blacklisted is calculated. Second, fraud detection IDs for all of the user's wallets are determined and handed to the TSP so it may add them to the RSU blacklist $bl_{\mathcal{R}}$. Note that the generation of the blacklist $bl_{\mathcal{T}}$ of user public keys is handled internally by the TSP and not in the scope of this task or $\mathcal{F}_{\mathrm{P4TC}}$.

Both parties—the TSP and the DR—input the public key ($\mathrm{pk}_{\mathcal{U}}^{\mathrm{DR}}$ and $\mathrm{pk}_{\mathcal{U}}^{\mathcal{T}}$, resp.) of the user that is going to be blacklisted. We assume both parties to agree on the same key out-of-band before the protocol starts. We first describe the "normal" case (step (3a)) for an honest TSP. (Note that the DR is assumed to be always honest.) To calculate the user's outstanding debt, all transaction entries in *TRDB* containing $pid_{\mathcal{U}}$ are taken and their respective prices p summed up to obtain b^{bill}. Note that although this sum may contain the prices of transactions and wallets that have already been cleared, this does not falsify the value of b^{bill} as every successful execution of Debt Clearance creates an entry with the amount that was cleared as negative price. For the actual blacklisting, the set of all wallet IDs belonging to $pid_{\mathcal{U}}$ is looked up and the remainder of the task is conducted for every wallet λ separately. $\mathcal{F}_{\mathrm{P4TC}}$ checks how many values of $f_{\Phi}(\lambda, \cdot)$ are already defined and extends them to the first $x_{\mathrm{bl}_{\mathcal{R}}}$ fraud detection IDs, where $x_{\mathrm{bl}_{\mathcal{R}}}$ is a parameter we assume to be greater than the number of transactions a user would be involved in within one billing period. To that end, yet undefined fraud detection IDs $f_{\Phi}(\lambda, x)$ with $x \leq x_{\mathrm{bl}_{\mathcal{R}}}$ are uniquely and randomly drawn or—in case of a corrupted user—obtained from the adversary. Finally, all fraud detection IDs $\phi = f_{\Phi}(\lambda, x)$ for $x \leq x_{\mathrm{bl}_{\mathcal{R}}}$ and all wallets λ of the user are output to the TSP together with the outstanding debt b^{bill}.

It remains to describe the remaining two cases. If the TSP is corrupted but the user in question honest (step (3b)), the TSP is free to drop some of the user's wallets and only partially blacklist the user. This "attack"—a demented TSP—cannot be ruled out. In order to correctly map this in the ideal model, the adversary is asked to provide a set of associated serial numbers and only the wallets from the intersection are used for blacklisting. This ensures that a malicious TSP can only blacklist less wallets but not more wallets or even wallets of another user. If the TSP and the user in question are both corrupted (step (3c)), no guarantees are given. Please note that a corrupted TSP could even come up with an “imaginary” corrupted user (which only exists in the head of the TSP) and ask the DR to blacklist this user. Essentially, this is nothing else than a cumbersome way to evaluate the PRF at inputs chosen by the TSP. However, the TSP can do this by itself anyway. We stress that this does not affect the security or privacy of honest parties in the system.

3.3.3. Properties of $\mathcal{F}_{\text{P4TC}}$

In this section we discuss why the previously defined ideal functionality $\mathcal{F}_{\text{P4TC}}$ captures an ideal model of a secure and privacy-preserving ETC scheme. Especially, we illustrate how the high-level objectives of a toll collection scheme (cp. Section 3.2.4) are reflected in $\mathcal{F}_{\text{P4TC}}$. The properties (P1) to (P8) are consolidated under the term *Operator security*, while properties (P9) to (P11) are summed up under *User Security and Privacy.*

Operator Security. At the bottom line, operator security, especially correctness of billing, follows from the fact that $\mathcal{F}_{\text{P4TC}}$ represents an incorruptible accountant which manages all wallets and their associated transactions in a single, pervasive database. In Debt Accumulation and Debt Clearance a (possibly malicious) user only inputs a serial number to indicate which previous wallet state should be used. All relevant information is then looked up by $\mathcal{F}_{\text{P4TC}}$ internally.

(P1) Owner-binding: Given the serial number of a previous wallet state, $\mathcal{F}_{\text{P4TC}}$ checks that the associated wallet belongs to the calling user and thus that is has legitimately been issued to that user. If the user is malicious, the adversary is allowed to indicate another corrupted user instead. Only corrupted users are able to swap wallets, they cannot use wallets of honest users. Please note that this does not change the total amount due to the operator. It only changes the party liable, which is unavoidable if corrupted users collude and mutually share their credentials.

(P2) Attribute-binding: As the attributes $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ attached to the user's wallet are internally managed by $\mathcal{F}_{\text{P4TC}}$, the user is unable to claim their wallet contains any other information than it actually does.

(P3) Balance-binding: By the same argument as in (P2) a user is unable to claim an incorrect balance b^{bill} in Debt Clearance. For each transaction, given the previous serial number, $\mathcal{F}_{\text{P4TC}}$ looks up the previous balance b^{prev}, calculates the price p and creates a new entry in the transaction database with balance $b \coloneqq b^{\text{prev}} + p$. The user only learns the price and the new balance, but cannot tamper with it. Assuming that no double-spending occurred, the set of transactions for a particular wallet forms a linked, linear list and hence b^{bill} equals the sum of all prices.

(P4) Double-spending Detection: The same fraud detection ID ϕ occurs in multiple transaction entries if and only if double-spending was committed. In this case the task Double-Spending Detection provides the TSP with the identity $pid_{\mathcal{U}}$ of the respective user and a publicly verifiable proof π that this user has committed double-spending. Again, a fraudulent user cannot elude detection as $\mathcal{F}_{\text{P4TC}}$ internally asserts that all transactions which have the same predecessor share the same fraud detection ID ϕ.

(P5) Participation Enforcement: This is handled outside the scope of $\mathcal{F}_{\text{P4TC}}$. As discussed in Section 3.2 we assume users to be physically identified by cameras if they do not properly participate in Debt Accumulation.

(P6) Blacklisting: Given a user ID $pid_{\mathcal{U}}$, the task Blacklisting & Recalculation provides the TSP with a set of past and upcoming fraud detection IDs ϕ of all wallets of this user. In order to lock down the upcoming fraud detection IDs $\mathcal{F}_{\text{P4TC}}$ pre-fills the mapping f_Φ with fresh, uniform and independent fraud detection IDs ϕ_i. In Debt Accumulation $\mathcal{F}_{\text{P4TC}}$ uses these already determined fraud detection IDs from f_Φ for the new wallet state. Then, $\mathcal{F}_{\text{P4TC}}$ checks whether the fraud detection ID of the new wallet state is contained in the blacklist provided by the RSU. Hence, the user is successfully blacklisted, if the RSU inputs the "correct" blacklist, i.e., a list containing fraud detection IDs from the TSP.

(P7) Debt Recalculation: Within the Blacklisting & Recalculation task, $\mathcal{F}_{\text{P4TC}}$ sums up all prices of all past transactions of all wallets of the user in question and outputs the resulting amount b^{bill} to the TSP. As each instance of Debt Clearance results in a transaction entry $trdb$ with $p = -b^{\text{bill}}$ as the price, correctly cleared and paid wallets cancel out in this sum and the result accurately gives the amount of debt still owed by the user. Again, the user is not able to tamper with the resulting balance as the database of all transaction entries is internally controlled by $\mathcal{F}_{\text{P4TC}}$.

(P8) Renegade Expulsion: This is handled outside the scope of $\mathcal{F}_{\text{P4TC}}$ by encoding a limited time of validity into the RSU's attributes (cp. Section 3.2.3).

User Security and Privacy. The information leakage needed to assess the level of user privacy is directly determined by the in- and output of $\mathcal{F}_{\text{P4TC}}$. We stress that we only care about privacy for honest, well-behaving, non-blacklisted[18] users. Hence, the grayed out steps in Fig. 3.2 can be ignored.

(P9) Unlinkability: First note that the serial number of the previous transaction s^{prev} is a private input of the user and never output to any party. After Debt Accumulation the RSU only learns the serial number s and fraud detection ID ϕ of the current transaction which are both freshly, uniformly and independently drawn by $\mathcal{F}_{\text{P4TC}}$. Wallet Issue only outputs s and Debt Clearance additionally outputs the final balance b^{bill}. Hence, it is *information-theoretically impossible* to track an honest and well-behaving user across any pair of transactions using any of these numbers. The only "real" information leakage in Debt Accumulation is determined by the user's and the previous RSU's attributes $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ which need to be assessed separately (see below).

(P10) Participation Provability: As discussed in Section 3.2 we assume the user to be physically identified (out-of-scope) if they do not properly participate in Debt Accumulation. In Prove Participation the SA inputs a user public key and a set $S_{\mathcal{R}}^{\text{pp}}$ of serial numbers in question. $\mathcal{F}_{\text{P4TC}}$ checks whether the user participated in any of these transactions. If so, $\mathcal{F}_{\text{P4TC}}$ simply outputs OK to the SA who does not learn anything about any of the user's transactions beyond that.

(P11) Protection Against False Accusation: Given a user ID and a bit string the task Guilt Verification checks if the bit string has been recorded as a legitimate proof-of-guilt for this user. A proof-of-guilt can only be registered via successful invocation of Double-Spending Detection. Hence, soundness of the proof and thus protection against false accusation is guaranteed by the internal bookkeeping of $\mathcal{F}_{\text{P4TC}}$.

[18] Note that the TSP cannot blacklist users alone and the incorruptible DR only cooperates if the user agreed or misbehaved.

$\mathcal{F}_{\mathrm{P4TC}}$ provides unlinkability of transactions ((P9)) up to information gained from user attributes, RSU attributes, and the total debt. As discussed in Section 3.2 we assume the attributes to be sufficiently indistinct to not enable any tracking. This is not ensured within the scope of $\mathcal{F}_{\mathrm{P4TC}}$—apart from outputs to users, which enable them to check attributes. Their real world impact on the privacy level crucially depends on the concrete deployment of the system. Please note that this constitutes a line of research on its own.

3.4. Π_{P4TC}: A Protocol to Realize $\mathcal{F}_{\mathrm{P4TC}}$

In this section, we basically follow the same approach as in Section 3.3: we start by explaining the main ideas, continue by explaining the protocol Π_{P4TC} at a high level in Section 3.4.1, then describe each task in more detail in Section 3.4.2, before finally presenting the complete protocol in Section 3.4.3. Afterwards we describe how our building blocks can be instantiated in Section 3.4.4. Lastly, we briefly state the security theorem in Section 3.4.5.

We start with some remarks on the algebraic setting, the used cryptographic building blocks, the modeling of channels, and the used setup assumptions. For formal definitions we refer to Chapter 2.

Algebraic Setting: Our system instantiation is based on an asymmetric bilinear group setting $(G_1, G_2, G_T, e, \mathrm{q}, g_1, g_2)$. Here, G_1, G_2, G_T are cyclic groups of prime order q (where no efficiently computable homomorphisms between G_1 and G_2 are known); g_1, g_2 are generators of G_1, G_2, respectively, and $e\colon\ G_1 \times G_2 \to G_T$ is a (non-degenerated) bilinear map. We rely on the co-CDH assumption as well as on the security of the building blocks in this setting.

Cryptographic Building Blocks: Our construction makes use of non-interactive zero-knowledge (NIZK) proofs, equivocal and extractable homomorphic commitments, digital signatures, public-key encryption, and pseudo-random functions (PRFs). The latter building blocks need to be efficiently and securely combinable with the chosen NIZK proof (which is Groth-Sahai [GS08] in our case).

Secure (Authenticated) Channels: All messages are encrypted using CCA-secure encryption. To this end a new session key is chosen by the user and encrypted under the public key of the RSU/TSP for each interaction. We omit these encryptions in the following. Apart from Debt Accumulation, we furthermore assume all channels to be authenticated.

Setup Assumptions: As commonly found in the UC setting, we also draw from setup assumptions. For our protocol Π_{P4TC}, we assume a common reference string (CRS) functionality $\mathcal{F}_{\mathrm{CRS}}$ (see Fig. 2.2), and a globally available bulletin board functionality $\mathcal{G}_{\mathrm{bb}}$ (see Fig. 2.4).

A CRS is a short piece of information that is shared between all parties. The trust assumptions are that the CRS has been generated honestly and that all parties possess the same CRS.

A bulletin board can be depicted as a key registration service which associates (physical) party identifiers (PIDs) with (cryptographic) public keys. The assumptions about $\mathcal{G}_{\mathrm{bb}}$ are that upon registration the operator of the bulletin board checks the identity of the registering party in a trustworthy way and that every party can retrieve information from $\mathcal{G}_{\mathrm{bb}}$ trustworthily. As the PID establishes an entity's physical identity it cannot be captured by cryptographic means. In our scenario the PID of users could be a passport number or SSN. For RSUs the geo-location could be used as a PID.

Main Idea. The basic technique underlying P4TC is a commit-sign-rerandomize-prove approach: a wallet is essentially a commitment *com* to the balance b (and some important auxiliary information) which is initially certified, i.e., signed, by the TSP during Wallet Issuing. To update b during Debt Accumulation, *com* cannot be sent over to the RSU as is since this would allow to link transactions. Instead it has to be re-randomized, i.e., the user generates a fresh commitment com' on b to send it over for updating. However, com' is not certified (directly) but only the original commitment *com*. Here, a NIZK proof comes into play to show that com' is certified indirectly, i.e., that com' is just a new commitment to the same balance b as contained in *com* for which the user knows a signature. If the RSU is convinced that this proof is correct, the RSU updates com' by the price p the user has to pay using the homomorphic property of the commitment and signs com'. In this way the user obtains a new certified wallet state. Debt Clearance essentially works the same as Debt Accumulation except that the user reveals b to the TSP.

To incorporate blacklisting and recalculation as well as double-spending detection capabilities, *com* has to be augmented by additional information. For Blacklisting & Recalculation, the user must additionally commit to a PRF key λ and counter x as well as deposit λ with the DR during Wallet Issuing and prove that this has been done correctly. In the x-th transaction, the value $\mathsf{PRF}(\lambda, x)$ now serves as the wallet's fraud detection ID ϕ which is revealed to the RSU/TSP. In the same transaction, the new wallet state com' for the upcoming transaction $x + 1$ is generated in which b is increased by p and the old counter value x by one. This is done in a similar manner as for the simpler case before. The NIZK proof now additionally enforces that the correct PRF key and counter value is used. Computing the fraud detection ID using a PRF has the advantage that the different (previous and future) states of a wallet are traceable given λ but untraceable if λ is unknown.

To encourage the user to only use the most recent wallet state by means of double-spending detection, each wallet state is bound to a random straight line encoding the user's secret key as slope: $t = \mathsf{sk}_{\mathcal{U}} u_2 + u_1$. To this end, the user chooses some random u_1 for the wallet state and the parameters determining the straight line, i.e., $\mathsf{sk}_{\mathcal{U}}$ and u_1, are added to the commitment *com* in Wallet Issuing (resp. com' in Debt Accumulation). Now in each transaction using this wallet state, the user is enforced (by means of the NIZK) to reveal a point (u_2, t) for random u_2 on this straight line. Clearly, if only one such point is revealed, this leaks nothing about the slope. However, if two (different) points are revealed, so the wallet state is used twice, then the slope $\mathsf{sk}_{\mathcal{U}}$ can be determined. This allows to compute $\mathsf{pk}_{\mathcal{U}}$ identifying the cheating user. Note that transactions involving the same wallet state can be recognized by exhibiting the same fraud detection ID ϕ (see before).

Structure of Wallets. A user's wallet essentially consists of two signed commitments $(com_{\mathcal{T}}, com_{\mathcal{R}})$, where $com_{\mathcal{T}}$ represents the fixed part and $com_{\mathcal{R}}$ the updatable part. Accordingly, the fixed part is signed along with the user's attributes $\mathbf{a}_{\mathcal{U}}$ by the TSP using $\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$ during Wallet Issuing. Every time $com_{\mathcal{R}}$ is updated, it is signed along with the serial number s of the transaction by the RSU using $\mathsf{sk}_{\mathcal{R}}$. The fixed part $com_{\mathcal{T}} = \mathsf{Com}(\lambda, \mathsf{sk}_{\mathcal{U}})$ is a commitment on the PRF key λ (used as wallet ID) and the user's secret key $\mathsf{sk}_{\mathcal{U}}$. The updatable part $com_{\mathcal{R}} = \mathsf{Com}(\lambda, b, u_1, x)$ also contains λ (to link both parts), the current balance b (debt), some user-chosen randomness u_1 to generate double-spending tags for the current wallet state, and a counter value x being the input to the PRF.

3.4.1. Introduction to Π_{P4TC}

We now explain the main ideas behind each task. Note that this section simplifies some notation for readability.

System Setup. In the setup phase, a TTP generates the bilinear group and a public common reference string crs.[19] The latter contains parameters for the NIZK and commitment scheme(s) we use. Formally, we assume that generation and distribution of the CRS is handled by $\mathcal{F}_{\text{CRS}}$.

DR/TSP/RSU Key Generation. The DR generates a keypair $(\mathsf{pk}_{\text{DR}}, \mathsf{sk}_{\text{DR}})$ for an IND-CCA2-secure public key encryption scheme, where pk_{DR} is used to deposit a user-specific trapdoor (PRF-key) which allows to link the user's transactions in case of disputes. An individual signature keypair $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ is generated for each RSU to sign the updatable part (see below) of a user's wallet. Moreover, the TSP generates keypairs $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{T}})$, $(\mathsf{vk}_{\mathcal{T}}^{\text{cert}}, \mathsf{sk}_{\mathcal{T}}^{\text{cert}})$ and $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}})$ for an EUF-CMA-secure signature scheme. The key $\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$ is used to sign fixed user-specific information (see below) when a new wallet is issued. Using $\mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}$ the TSP plays the role of the initial RSU signing the updatable part of a new wallet.

RSU Certification. An RSU engages with the TSP in this task to get its certificate $\mathsf{cert}_{\mathcal{R}}$. It contains the RSU's public verification key $\mathsf{vk}_{\mathcal{R}}$, its attributes $\mathbf{a}_{\mathcal{R}}$ (that are assigned in this task by the TSP), and a signature on both, generated by the TSP using $\mathsf{sk}_{\mathcal{T}}^{\text{cert}}$.

User Registration. To participate in the system, a user needs to generate a keypair $(\mathsf{pk}_{\mathcal{U}} \coloneqq g_1^{\mathsf{sk}_{\mathcal{U}}}, \mathsf{sk}_{\mathcal{U}}) \in G_1 \times \mathbb{Z}_q$. We assume that the TSP out-of-band binds $\mathsf{pk}_{\mathcal{U}}$ to a verified physical user ID, in order to hold the user liable in case of a misuse. The keypair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ is used to bind a wallet to a user.

Wallet Issuing. This task is executed between user and TSP to create a new wallet with a fresh wallet ID λ and balance 0. Additionally, the PRF key λ is deposited under the DR's public encryption key.

To generate the wallet, the user encodes λ together with their other secrets into the wallet. The PRF key λ needs to be chosen jointly and remain secret to the TSP. If only the user chose the key, an adversary could tamper with Blacklisting & Recalculation, as well as with Double-Spending Detection (e.g., by choosing the same key for two different users). If only the TSP chose it, a user's transactions would be linkable. To this end, the user and the TSP engage in the first two messages of a Blum coin toss. After the second message $\lambda \coloneqq \lambda' + \lambda''$ is fixed and the user knows their own share λ' as well as the TSP's share λ''. Then the user computes the commitments $com_{\mathcal{T}} \coloneqq \mathsf{Com}(\lambda, \mathsf{sk}_{\mathcal{U}})$ and $com_{\mathcal{R}} \coloneqq \mathsf{Com}(\lambda, b \coloneqq 0, u_1, x \coloneqq 0)$.

Additionally, the user prepares the deposit of λ. This is a bit tricky, as the user needs to prove that the ciphertext they give to the TSP is actually an encryption of λ under pk_{DR}. For practical reasons, we use Groth-Sahai NIZKs [GS08] and the Dodis-Yampolskiy PRF [DY04]. Ideally, one would want an encryption scheme that is compatible with Groth-Sahai and whose message space equals the key space of the PRF, i.e., $\mathbb{Z}_q$. Unfortunately, we are not aware of any such scheme. Instead, we use a CCA-secure structure-preserving encryption scheme for vectors of G_1-elements [Cam+11] and the following workaround: The user splits up its share λ' into small chunks $\lambda'_i < \mathcal{B}$ (e.g., $\mathcal{B} = 2^{32}$) such that recovering the discrete logarithm of $\Lambda'_i \coloneqq g_1^{\lambda'_i}$ becomes feasible. All chunks $\Lambda_{i \in \{0,\ldots,\ell-1\}}$, the TSP's share $\Lambda'' \coloneqq g_1^{\lambda''}$, and the user's public key $\mathsf{pk}_{\mathcal{U}}$ are jointly encrypted as e^* under pk_{DR}. The CCA-secure ciphertext e^* unambiguously binds λ to the user's key $\mathsf{pk}_{\mathcal{U}}$ and rules out malleability attacks. Otherwise,

[19] If one does not want to assume the existence of a TTP, one can let distrusting parties perform a secure multi-party computation once to generate the common reference string (CRS). One could assume that the CRS (~ 7 KB) is distributed along with the OBU software, certified by a trusted certification authority.

a malicious TSP could potentially trick the DR into recovering the trapdoor of a different (innocent) user.

The user then sends over e^*, $com_\mathcal{T}$, $com_\mathcal{R}$ along with a NIZK π proving that everything has been generated honestly and the wallet is bound to the user owning $\mathsf{sk}_\mathcal{U}$. When the TSP receives this data, it verifies the NIZK first. If the check passed, the TSP signs $com_\mathcal{T}$ along with attributes $\mathbf{a}_\mathcal{U}$ using $\mathsf{sk}_\mathcal{T}^\mathcal{T}$, as well as signs $com_\mathcal{R}$ along with the serial number s using $\mathsf{sk}_\mathcal{T}^\mathcal{R}$.[20] The resulting signatures $\sigma_\mathcal{T}$, $\sigma_\mathcal{R}$ are sent to the user, who checks their correctness. The user finally stores the freshly generated token

$$\tau := (\mathbf{a}_\mathcal{U}, com_\mathcal{R}, decom_\mathcal{R}, \sigma_\mathcal{R}, com_\mathcal{T}, decom_\mathcal{T}, \sigma_\mathcal{T}, \lambda := \lambda' + \lambda'', b := 0, u_1, x := 1, s),$$

where $decom_\mathcal{R}$ and $decom_\mathcal{T}$ are the decommitment values required to open $com_\mathcal{R}$ and $com_\mathcal{T}$, respectively. The TSP stores $htd := (\mathsf{pk}_\mathcal{U}, s, \lambda'', e^*)$ as hidden user trapdoor to recover λ with help of DR.

Debt Accumulation. In this task (see Fig. 3.6 for a simplified version) executed between an anonymous user and an RSU, the toll p is determined and a new state of the user's wallet with a debt increased by p is created.

The user's main input is the token

$$\tau^\text{prev} := (\mathbf{a}_\mathcal{U}, com_\mathcal{R}^\text{prev}, decom_\mathcal{R}^\text{prev}, \sigma_\mathcal{R}^\text{prev}, com_\mathcal{T}, decom_\mathcal{T}, \sigma_\mathcal{T}, \lambda, b^\text{prev}, u_1^\text{prev}, x, s^\text{prev})$$

containing the state of their previous task run and wallet. To update the wallet state, the user computes a fresh commitment $com'_\mathcal{R}$ on $(\lambda, b^\text{prev}, u_1^\text{next}, x)$, where except for u_1^next the same values are contained in the previous wallet state. The double-spending mask u_1^next is freshly chosen by the user and is used to generate a double-spending tag for the new wallet state. In order to get the new wallet state certified by the RSU, the user needs to invalidate the previous state. To do so, they need to reveal the fraud detection ID and double-spending tag of the previous state. For the latter, the RSU sends a double-spending randomness u_2 (as "challenge") along with a commitment $com''_\text{ser} := \mathsf{Com}(s'')$ on its share of the serial number of this transaction (which is part of the Blum coin toss), and the certificate for $\mathsf{vk}_\mathcal{R}$ to the user. Upon receiving these values, the user computes the double-spending tag $t := u_2 \cdot \mathsf{sk}_\mathcal{U} + u_1^\text{prev} \bmod q$ (a linear equation in the unknowns u_1^prev and $\mathsf{sk}_\mathcal{U}$), the fraud detection ID $\phi := \mathsf{PRF}(\lambda, x)$, and a hidden user ID $com_\text{hid} := \mathsf{Com}(\mathsf{pk}_\mathcal{U})$. The latter is used in Prove Participation to associate this interaction with the user. As response, the user sends over com_hid, $com'_\mathcal{R}$, ϕ, t, $\mathbf{a}_\mathcal{U}$, $\mathbf{a}_\mathcal{R}^\text{prev}$, π, and s', where π is a NIZK proving that everything has been computed honestly, and s' is the user's share of the serial number. In particular, π shows that the user knows a certified wallet state involving commitments $com_\mathcal{T}$ and $com_\mathcal{R}^\text{prev}$ such that $com_\mathcal{R}^\text{prev}$ and $com'_\mathcal{R}$ are commitments on the same messages except for the double-spending randomness, that the (hidden) signature on $com_\mathcal{R}^\text{prev}$ verifies under some (hidden) RSU key $\mathsf{vk}_\mathcal{R}^\text{prev}$ certified by the TSP, and that t, ϕ, and com_hid have been computed using the values contained in $com_\mathcal{T}$ and $com_\mathcal{R}^\text{prev}$.

When receiving this data, the RSU first checks that ϕ is not on the blacklist and π is correct. Then it calculates the price p, adds it to the user's balance b^prev and increases the counter x by 1 using the homomorphic property of $com'_\mathcal{R}$. The resulting commitment $com_\mathcal{R}$ is signed along with the serial number $s := s' \cdot s''$ using $\mathsf{sk}_\mathcal{R}$. Then the RSU sends $com_\mathcal{R}$ to the user, along with its decommitment information $decom_\mathcal{R}$, its signature $\sigma_\mathcal{R}$, the added price p, RSU's share s'' of the serial number, and the decommitment $decom''_\text{ser}$ for com''_ser.

[20] A uniformly random serial number s for this transaction is jointly generated by user and TSP by means of a Blum-like coin toss (see also Debt Accumulation task for details).

Simplified Excerpt of Protocol Π_{P4TC}

Debt Accumulation:

- Input User: ($\textsc{PayToll}, s^{\text{prev}}$)
- Input RSU: ($\textsc{PayToll}, bl_{\mathcal{R}}$)
- Behavior:
  (1) RSU:
    (a) Sample double-spending randomness: $u_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$
    (b) Sample second half of the serial number: $s'' \overset{\text{R}}{\leftarrow} G_1$
    (c) Commit on second half of the serial number: $(com''_{\text{ser}}, decom''_{\text{ser}}) \leftarrow \mathsf{Com}(s'')$
    (d) Send $(u_2, com''_{\text{ser}}, \text{cert}_{\mathcal{R}})$ to the user
  (2) User:
    (a) Sample first half of the serial number: $s' \overset{\text{R}}{\leftarrow} G_1$
    (b) Sample double-spending mask: $u_1^{\text{next}} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$
    (c) Prepare first half of RSU commitment: $(com'_{\mathcal{R}}, decom'_{\mathcal{R}}) \leftarrow \mathsf{Com}(\lambda, b^{\text{prev}}, u_1^{\text{next}}, x)$
    (d) Compute double-spending tag: $t \coloneqq \text{sk}_{\mathcal{U}} u_2 + u_1^{\text{prev}}$
    (e) Compute fraud detection ID: $\phi \coloneqq \mathsf{PRF}(\lambda, x)$
    (f) Hide own identity: $(com_{\text{hid}}, decom_{\text{hid}}) \leftarrow \mathsf{Com}(\text{pk}_{\mathcal{U}})$
    (g) *Compute proof π*
    (h) Send $(s', \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, com_{\text{hid}}, com'_{\mathcal{R}}, t)$ to the RSU
  (3) RSU:
    (a) Check blacklist and proof: If $\big((\phi \in bl_{\mathcal{R}}) \ \vee \ (\pi \textit{ does not verify})\big)$, then return $\perp$
    (b) *Calculate price p*
    (c) Compute final serial number: $s \coloneqq s' \cdot s''$
    (d) Prepare second half of RSU commitment: $(com''_{\mathcal{R}}, decom''_{\mathcal{R}}) \leftarrow \mathsf{Com}(0, p, 0, 1)$
    (e) Assemble RSU commitment: $com_{\mathcal{R}} \coloneqq com'_{\mathcal{R}} \cdot com''_{\mathcal{R}}$
    (f) Sign RSU commitment: $\sigma_{\mathcal{R}} \leftarrow \mathsf{Sign}(\text{sk}_{\mathcal{R}}, (com_{\mathcal{R}}, s))$
    (g) Store $\omega^{\text{dsp}} \coloneqq (\phi, t, u_2)$, $\omega^{\text{rc}} \coloneqq (\phi, p)$, and $\omega_{\mathcal{R}}^{\text{pp}} \coloneqq (s, com_{\text{hid}})$
    (h) Send $(s'', decom''_{\text{ser}}, com_{\mathcal{R}}, decom''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to the user
  (4) User:
    (a) Assemble RSU decommitment: $decom_{\mathcal{R}} \coloneqq decom'_{\mathcal{R}} \cdot decom''_{\mathcal{R}}$
    (b) Perform sanity checks: If $\big((\mathsf{Open}(s'', com''_{\text{ser}}, decom''_{\text{ser}}) = 0) \ \vee \ (\mathsf{Vfy}(\text{vk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (com_{\mathcal{R}}, s)) = 0)\big)$, then return $\perp$
    (c) Assemble new token: $\tau \coloneqq (\mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, \lambda, b \coloneqq b^{\text{prev}} + p, u_1^{\text{next}}, x+1, s \coloneqq s' \cdot s'')$
    (d) Store τ and $\omega_{\mathcal{U}}^{\text{pp}} \coloneqq (s, com_{\text{hid}}, decom_{\text{hid}})$
- Output User: $(s, \mathbf{a}_{\mathcal{R}}, p, b)$
- Output RSU: $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$

Figure 3.6.: A simplified version of Debt Accumulation

The user checks if the received data is valid and ends up with an updated token

$$\tau \coloneqq (\mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, \lambda, b^{\text{prev}} + p, u_1^{\text{next}}, x + 1, s)$$

containing the new wallet state $com_{\mathcal{T}}, com_{\mathcal{R}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}}$. The user additionally stores $\omega_{\mathcal{U}}^{\text{pp}} \coloneqq (s, com_{\text{hid}}, decom_{\text{hid}})$, where $decom_{\text{hid}}$ allows to open the hidden ID, for the case that it is needed in a Prove Participation task. The RSU stores $(\phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, p, t, u_2, com_{\text{hid}}, s)$.

Debt Clearance. In this task conducted by a user and the TSP, the user identifies themselves using $\mathsf{sk}_{\mathcal{U}}$ and reveals the balance of their current wallet state. The wallet is terminated by not creating a new state. Upon receiving a double-spending randomness u_2 from the TSP, the user computes the double-spending tag t and fraud detection ID ϕ. This is the same as in Debt Accumulation. Then the user sends over $\mathsf{pk}_{\mathcal{U}}$, b^{prev}, ϕ, t and a NIZK proof π. This NIZK proof asserts that the user knows a certified wallet state with balance b^{prev}, fraud detection ID ϕ, and double-spending tag t which is bound to $\mathsf{pk}_{\mathcal{U}}$. Note that if the user does not make use of their latest wallet state, double-spending detection will reveal this. If the proof verifies, the balance and the double-spending information, i.e., $(b^{\mathrm{prev}}, \phi, t, u_2)$, is stored by the TSP.

Prove Participation. In this task executed between a user and the SA, the user proves that they participated in one of the Debt Accumulation transactions under audit by the SA. This task is identifying, as the SA retrieved the user's physical ID from their license plate number and, thus, is aware of their public key $\mathsf{pk}_{\mathcal{U}}$. First, the SA sends the list $S_{\mathcal{R}}^{\mathrm{pp}}$ of serial numbers observed by the RSU during the considered time frame to the user. If the user owns some $\omega_{\mathcal{U}}^{\mathrm{pp}} := (s, \mathit{com}_{\mathrm{hid}}, \mathit{decom}_{\mathrm{hid}})$ with $s \in S_{\mathcal{R}}^{\mathrm{pp}}$, they simply send over $\omega_{\mathcal{U}}^{\mathrm{pp}}$. The SA accepts if the commitment $\mathit{com}_{\mathrm{hid}}$ indeed opens to $\mathsf{pk}_{\mathcal{U}}$ and s is part of the corresponding prove participation information the RSU stored. Note that no other user may claim to have sent $\mathit{com}_{\mathrm{hid}}$, as this would imply to open $\mathit{com}_{\mathrm{hid}}$ with a different public key.

Double-Spending Detection. This algorithm is applied by the TSP to its database containing all double-spending information (ϕ, t, u_2) collected by the RSUs. The fraud detection ID ϕ is thereby used as the database index. If the same index appears twice in the TSP's database, a double-spending occurred and the cheater's keypair can be reconstructed from the two double-spending information as follows: Let us assume there exist two records (ϕ, t, u_2), (ϕ, t', u_2'). In this case, $\mathsf{sk}_{\mathcal{U}}$ can be recovered as $\mathsf{sk}_{\mathcal{U}} := (t - t')(u_2 - u_2')^{-1} \bmod q$ with overwhelming probability. The cheater's public key $\mathsf{pk}_{\mathcal{U}}$ can be computed from $\mathsf{sk}_{\mathcal{U}}$. As a consequence, the wallet bound to $\mathsf{pk}_{\mathcal{U}}$ could be blacklisted. The output of the algorithm is the fraudulent user's public key $\mathsf{pk}_{\mathcal{U}}$ along with a proof-of-guilt $\pi := \mathsf{sk}_{\mathcal{U}}$.

Guilt Verification. This algorithm can be executed by any party to verify the guilt of an accused double-spender. Given a public key $\mathsf{pk}_{\mathcal{U}}$ and a proof-of-guilt π, it checks if $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$.

Blacklisting & Recalculation. This is a task executed by the DR upon request of the TSP. We assume that DR and TSP agreed out-of-band that the user with public key $\mathsf{pk}_{\mathcal{U}}^{\mathrm{DR}}$ should be blacklisted before the task starts. Given e^* and λ'', the DR recovers the contained PRF key but only if it is bound to $\mathsf{pk}_{\mathcal{U}}^{\mathrm{DR}}$. To this end, DR decrypts e^* using $\mathsf{sk}_{\mathrm{DR}}$ to obtain Λ_i', Λ'', and $\mathsf{pk}_{\mathcal{U}}$. If $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}_{\mathcal{U}}^{\mathrm{DR}}$ or $\Lambda'' \neq g_1^{\lambda''}$ it aborts. Otherwise, it computes the (small) discrete logarithms of the Λ_i' to the base g_1 to recover the chunks λ_i' of the user's share of the PRF key. The key is computed as $\lambda := \lambda'' + \sum_{i=0}^{\ell-1} \lambda_i' \cdot \mathcal{B}^i$. Using λ, the fraud detection IDs belonging to the previous and upcoming states of a user's wallet can be computed. Thus, all interactions of the user (including double-spendings) in the TSP's database can be linked and the legitimate debt recalculated. Also, the fraud detection IDs for upcoming transactions with this wallet can be blacklisted.

3.4.2. Task Descriptions

In this and the next section we describe and define a real protocol Π_{P4TC} that implements our toll collection system $\mathcal{F}_{\mathrm{P4TC}}$. The proof that Π_{P4TC} is a UC-realization of $\mathcal{F}_{\mathrm{P4TC}}$ is deferred to Appendix A.1.

The style of the presentation of Π_{P4TC} roughly follows the same structure as the presentation of the ideal model $\mathcal{F}_{\mathrm{P4TC}}$ in Section 3.3: Although Π_{P4TC} is a single, monolithic protocol with different tasks, the individual tasks are presented as if they were individual protocols.

While in the ideal model all information is kept in a single, pervasive, trustworthy database, in the real model such a database does not exist. Instead, the state of the system is distributed across all parties. Each party locally stores a piece of information: The user owns a "user wallet", which is updated during each transaction, the RSU collects "double-spending information", "recalculation information", and "prove participation information", which are periodically sent to the TSP, and the TSP creates and keeps "hidden user trapdoors" for each wallet issued.

Secure Authenticated Channels. In our system, all protocol messages are encrypted using CCA-secure encryption. For this purpose, a new session key chosen by the user is encrypted under the public key of an RSU/TSP for each interaction. Furthermore, we make use of fully authenticated channels. The only exception to this is the task of Debt Accumulation where only the participating RSU authenticates itself to the user who in turn remains anonymous. We omit these encryptions and authentications when describing the protocol.

Wallets. A central component of our toll collection system is the wallet that is created during Wallet Issuing. It is of the form

$$\tau \coloneqq (s, \phi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathrm{cert}_{\mathcal{R}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathrm{next}})\,.$$

Some of the components are fixed after creation, some change after every transaction. The fixed components consist of the *wallet ID* λ (which is also used as the PRF seed), the *user attributes* $\mathbf{a}_{\mathcal{U}}$, the *TSP commitment* $com_{\mathcal{T}}$ (a commitment on λ and the secret user key $\mathrm{sk}_{\mathcal{U}}$), its corresponding opening $decom_{\mathcal{T}}$ and a signature $\sigma_{\mathcal{T}}$ on $com_{\mathcal{T}}$ and $\mathbf{a}_{\mathcal{U}}$ created by the TSP.

The alterable components consist of the *RSU commitment* $com_{\mathcal{R}}$ (a commitment on λ, b, u_1^{next} and x^{next}), its corresponding opening $decom_{\mathcal{R}}$, a signature $\sigma_{\mathcal{R}}$ on $com_{\mathcal{R}}$ and s created by a RSU, the *balance* b, the *double-spending mask* u_1^{next} for the next transaction, the *PRF counter* x^{next} for the next interaction, a *RSU certificate* $\mathrm{cert}_{\mathcal{R}}$ and the *serial number* s and *fraud detection ID* $\phi \coloneqq \mathsf{PRF}(\lambda, x^{\mathrm{next}} - 1)$ for the current transaction. These components change after each interaction with an RSU via the Debt Accumulation task.

In the following, a description of the operations of each task is given. We again group the tasks into "System Setup Tasks", "Basic Tasks", and "Feature Tasks". For a better overview, the variables used are summarized in Table 3.3 and Table 3.4. The complete protocol is afterwards presented in Section 3.4.3.

3.4.2.1. System Setup Tasks

To set up the system, the CRS needs to be created and most parties (DR, TSP, RSUs, and Users) need to create their keypair.It is assumed that the DR and the TSP register before the initial start of the system, while RSUs and users can register afterwards (but they need to register before they can participate in the system).

Identifier	**Domain**	**Description**
$\mathsf{sk}_{\mathrm{DR}}$	$\mathbb{Z}_q^{2\ell+8}$	secret DR key
$\mathsf{sk}_{\mathcal{T}}$	$\mathbb{Z}_q^{j+3} \times \mathbb{Z}_q^{y+6} \times \mathbb{Z}_q^4$	secret TSP key
$\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}$	G_1^{j+3}	public TSP commitment signing key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$	$\mathbb{Z}_q^{j+3}$	secret TSP commitment signing key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}}$	$G_1^3 \times G_2^{y+3}$	public certification key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathrm{cert}}$	$\mathbb{Z}_q^{y+6}$	secret certification key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}$	$G_1^3 \times G_2$	public TSP's RSU commitment signing key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}$	$\mathbb{Z}_q^4$	secret TSP's RSU commitment signing key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{R}}$	$\mathbb{Z}_q^4$	secret RSU key
$\mathsf{sk}_{\mathcal{U}}$	$\mathbb{Z}_q$	secret user key
$com_{\mathcal{T}}$	G_2	TSP commitment
$decom_{\mathcal{T}}$	G_1	decommitment of $com_{\mathcal{T}}$
$\sigma_{\mathcal{T}}$	$G_2^2 \times G_1$	signature on $com_{\mathcal{T}}$ and $\mathbf{a}_{\mathcal{U}}$
$com_{\mathcal{R}}$	G_2	RSU commitment
$decom_{\mathcal{R}}$	G_1	decommitment of $com_{\mathcal{R}}$
$\sigma_{\mathcal{R}}$	$G_2^2 \times G_1$	signature on $com_{\mathcal{R}}$ and s
$\mathsf{cert}_{\mathcal{R}}$	$(G_1^3 \times G_2) \times G_1^y \times (G_2^2 \times G_1)$	RSU certificate
$\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}$	$(G_1^3 \times G_2) \times G_1^y \times (G_2^2 \times G_1)$	TSP certificate
$\sigma_{\mathcal{R}}^{\mathrm{cert}}$	$G_2^2 \times G_1$	signature on $\mathsf{vk}_{\mathcal{R}}$ and $\mathbf{a}_{\mathcal{R}}$
$\sigma_{\mathcal{T}}^{\mathrm{cert}}$	$G_2^2 \times G_1$	signature on $\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}$ and $\mathbf{a}_{\mathcal{T}}$
com_{hid}	G_2	hidden ID
$decom_{\mathrm{hid}}$	G_1	opening of hidden ID
com'_{seed}	G_2	commitment on the user half of the wallet ID
$decom'_{\mathrm{seed}}$	G_1	opening of com'_{seed}
com''_{ser}	G_1^2	commitment on the RSU half of the serial number
$decom''_{\mathrm{ser}}$	$\mathbb{Z}_q^2$	opening of com''_{ser}
ω^{dsp}	$G_1 \times \mathbb{Z}_q \times \mathbb{Z}_q$	double-spending information
ω^{rc}	$G_1 \times \mathbb{Z}_q$	blacklisting information
$\omega_{\mathcal{U}}^{\mathrm{pp}}$	$G_1 \times G_2 \times G_1$	user's prove participation information
$\omega_{\mathcal{R}}^{\mathrm{pp}}$	$G_1 \times G_2$	RSU's prove participation information
u_1	$\mathbb{Z}_q$	double-spending mask
u_2	$\mathbb{Z}_q$	double-spending randomness
t	$\mathbb{Z}_q$	double-spending tag
htd	$G_1 \times G_1 \times \mathbb{Z}_q \times ((G_1^3 \times G_2^3) \times G_1^{\ell+2} \times G_{\mathrm{T}})$	hidden user trapdoor
HTD	set of $(G_1 \times G_1 \times \mathbb{Z}_q \times ((G_1^3 \times G_2^3) \times G_1^{\ell+2} \times G_{\mathrm{T}}))$ elements	set of hidden user trapdoors
λ_{PRF}	$\mathbb{N}$	maximum value of the PRF counter
$\mathcal{B}$	$\mathbb{N}$	splitting base

Table 3.4.: Notation that only occurs in the real protocol Π_{P4TC}

System Setup (CRS Generation). To setup the system once, the public parameter crs must be generated in a trustworthy way. The CRS crs consists of a description of the underlying algebraic framework gp, a *splitting base* $\mathcal{B}$ and the individual CRSs for the used commitments and zero-knowledge proofs. Formally, we assume that the hybrid functionality $\mathcal{F}_{\text{CRS}}$ handles CRS generation and distribution. In practice, a number of mutually distrusting parties can run a multi-party computation (using some other sort of setup assumption) to generate the CRS or a commonly trusted party is charged with this task. As a trusted third party (the DR) explicitly participates in our system (for resolving disputes), this party could also run the system setup.

In the following task descriptions, we assume that the CRS is available to all parties.

Registration. The registration algorithms of DR, TSP, RSUs and users mainly consist of the parties generating their keypairs (and obtaining the CRS).

The DR computes an encryption keypair $(\mathsf{pk}_{\text{DR}}, \mathsf{sk}_{\text{DR}})$ which can be used to remove the unlinkability of user transactions in case of a dispute between the user and the TSP. The DR could be a (non-governmental) organization trusted by both, users to protect their privacy and the TSP to protect operator security.

The TSP must also generate a keypair. Therefore, the TSP generates several signature keypairs $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{T}})$, $(\mathsf{vk}_{\mathcal{T}}^{\text{cert}}, \mathsf{sk}_{\mathcal{T}}^{\text{cert}})$, $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}})$, where $\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$ is used in the Wallet Issuing task to sign the TSP commitment $com_{\mathcal{T}}$ and the user attributes $\mathbf{a}_{\mathcal{U}}$, $\mathsf{sk}_{\mathcal{T}}^{\text{cert}}$ is used to sign RSU public keys in the RSU Certification task and $\mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}$ is used in the Wallet Issuing task to sign the RSU commitment $com_{\mathcal{R}}$ and the serial number s in place of an RSU. The TSP also generates a certificate $\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}$ for its own key $\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}$.

Each RSU must generate a keypair as well. For that purpose, each RSU generates a signature keypair $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ that is used in the Debt Accumulation task to sign the RSU commitment $com_{\mathcal{R}}$.

Each user also has to generate a keypair. The public key $\mathsf{pk}_{\mathcal{U}}$ will be used to identify the user in the system and is assumed to be bound to a physical ID such as a passport number, social security number, etc. Of course, for this purpose the public key needs to be unique. We assume that ensuring the uniqueness of user public keys as well as verifying and binding a physical ID to them is done "out-of-band" before participating in the Wallet Issuing task. A simple way to realize the latter could be to make use of external trusted certification authorities.

RSU Certification. The RSU Certification task is executed between a RSU and the TSP when a new RSU is deployed into the field. There, the TSP certifies the validity of the RSU public key and stores the certificate on the RSU.

Note that the public key of an RSU $\mathsf{vk}_{\mathcal{R}}$ and the associated certificate $\mathsf{cert}_{\mathcal{R}}$ has to be refreshed from time to time. For the ease of presentation we assume that the same RSU (identified by its PID $pid_{\mathcal{R}}$) can only be registered once. In other words, if the (physically identical) RSU is removed from the field, goes to maintenance and is re-deployed to the field, we consider this RSU a "new" RSU.

3.4.2.2. Basic Tasks

Next we describe main tasks of our toll collection scheme, namely *Wallet Issuing*, *Debt Accumulation*, and *Debt Clearance.*

Wallet Issuing. The Wallet Issuing task is executed between a user and the TSP. It is executed at the beginning of each billing period to generate a fresh wallet for the user. The task fulfills four objectives:

(1) Jointly computing a fresh and random wallet ID for the user that is only known to the user.

(2) Storing this wallet ID in a secret form at the TSP such that it can only be recovered by the DR in the case that the user conducts a fraud.

(3) Jointly computing a fresh and random serial number for this transaction.

(4) Creating a new wallet for the user.

For the first objective, both parties randomly choose shares of the wallet ID λ' and λ'', respectively, that together form the wallet ID $\lambda \coloneqq \lambda' + \lambda''$. To this end, the parties engage in the first two message of a Blum coin toss and omit the last message. This way, the wallet ID λ is fixed and known by the user, but remains secret to the TSP.

The second objective requires some sort of key-escrow mechanism. Ideally, the user would simply encrypt the wallet ID λ under the public key $\mathsf{pk}_{\mathrm{DR}}$ of the DR and prove to the TSP that the encrypted value is consistent to the committed shares in zero-knowledge. Unfortunately, we are unaware of a CCA-secure encryption scheme whose message space equals the key space of our PRF underlying the wallet (i.e., $\mathbb{Z}_{\mathrm{q}}$) and that is compatible to the GS-NIZK proof system (i.e., is algebraic). Moreover, it is impossible to recover λ from g_1^{λ} due to the hardness of the DLOG problem in G_1. Therefore, the user splits its share λ' into smaller values $\lambda'_0, \ldots, \lambda'_{\ell-1} \in \{0, \ldots, \mathcal{B} - 1\}$ s.t. $\lambda' \coloneqq \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$ for some base $\mathcal{B}$. The *splitting base* $\mathcal{B}$ is chosen in a way that it is feasible for the DR to recover λ'_i from $g_1^{\lambda'_i}$ in a reasonable amount of time (e.g., $\mathcal{B} = 2^{32}$). Then the user encrypts a vector of all $g_1^{\lambda'_i}$ together with the TSP's share $g_1^{\lambda''}$ and their own public key $\mathsf{pk}_{\mathcal{U}}$ under the public key $\mathsf{pk}_{\mathrm{DR}}$ of the DR and sends the ciphertext e^* to the TSP. The TSP's share $g_1^{\lambda''}$ and the public key $\mathsf{pk}_{\mathcal{U}}$ are embedded into the ciphertext in order to bind it to the user and to rule out malleability attacks. The TSP creates the hidden user trapdoor as $htd \coloneqq (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$. In the case that the user commits a fraud, the TSP sends the htd for each wallet of the fraudulent user to the DR and the DR recovers the wallet ID λ for each wallet. For more details, see the Blacklisting & Recalculation task.

The goal of the third objective is to create a truly random serial number $s \in G_1$ for this transaction. To ensure that the serial number is indeed random (and not maliciously chosen by either party), the user and the TSP engage in another (complete) Blum coin toss.

For the last objective, the user generates the TSP and RSU commitments, i.e., the fixed and the updatable part of the wallet. The user commits to the wallet ID λ and secret user key $\mathsf{sk}_{\mathcal{U}}$ for the TSP commitment $com_{\mathcal{T}}$. For the preliminary RSU commitment $com_{\mathcal{R}}$, the user commits to the wallet ID λ, the balance $b \coloneqq 0$, a fresh double-spending mask u_1^{next} and the PRF counter $x^{\mathrm{next}} \coloneqq 1$. Then the user computes a proof showing that these commitments are formed correctly. The proof also shows that the encryption

e^* has been honestly created and that each λ'_i is smaller than $\mathcal{B}$. More precisely, P1 is used to compute a proof π for a statement *stmt* from the language $L^{(1)}_{\text{gp}}$ defined by

$$L^{(1)}_{\text{gp}} \coloneqq \left\{ \begin{pmatrix} \text{pk}_{\mathcal{U}} \\ \text{pk}_{\text{DR}} \\ e^* \\ com_{\mathcal{T}} \\ com_{\mathcal{R}} \\ com'_{\text{seed}} \\ \Lambda'' \\ \lambda'' \end{pmatrix}^{\top} \middle| \begin{array}{l} \exists\, \lambda, \lambda', \lambda'_0, \ldots, \lambda'_{\ell-1}, r_1, r_2 \in \mathbb{Z}_q; \\ \Lambda, \Lambda', \Lambda'_0, \ldots, \Lambda'_{\ell-1}, U_1^{\text{next}}, decom_{\mathcal{T}}, decom_{\mathcal{R}}, decom'_{\text{seed}} \in G_1; \\ \text{SK}_{\mathcal{U}} \in G_2 : \\ e\left(\text{pk}_{\mathcal{U}}, g_2\right) = e\left(g_1, \text{SK}_{\mathcal{U}}\right) \\ \text{C1.Open}(\text{crs}^1_{\text{com}}, (\Lambda, \text{pk}_{\mathcal{U}}), com_{\mathcal{T}}, decom_{\mathcal{T}}) = 1 \\ \text{C1.Open}(\text{crs}^1_{\text{com}}, (\Lambda, 1, U_1^{\text{next}}, g_1), com_{\mathcal{R}}, decom_{\mathcal{R}}) = 1 \\ \text{C1.Open}(\text{crs}^1_{\text{com}}, \Lambda', com'_{\text{seed}}, decom'_{\text{seed}}) = 1 \\ e^* = \text{E.Enc}(\text{pk}_{\text{DR}}, (\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \text{pk}_{\mathcal{U}}); r_1, r_2) \\ \lambda = \lambda' + \lambda'' \\ \Lambda = g_1^{\lambda},\ \Lambda' = g_1^{\lambda'} \\ \lambda' = \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i \\ \forall i \in \{0, \ldots, \ell-1\} : \\ \qquad \lambda'_i \in \{0, \ldots, \mathcal{B}-1\} \\ \qquad \Lambda'_i = g_1^{\lambda'_i} \end{array} \right\} \tag{3.1}$$

Note that the first equation in Eq. (3.1) actually proves the knowledge of $g_2^{\text{sk}_{\mathcal{U}}}$ (rather than $\text{sk}_{\mathcal{U}}$ itself).[21] However, computing $g_2^{\text{sk}_{\mathcal{U}}}$ without knowing $\text{sk}_{\mathcal{U}}$ (only given $\text{pk}_{\mathcal{U}}$) is assumed to be a hard problem (Co-CDH).

In temporal order, the task proceeds as follows: In the first message (from user to TSP) the user sends its own public key $\text{pk}_{\mathcal{U}}$ and starts the Blum coin toss for the wallet ID by sending com'_{seed}. The TSP checks if the user's public key is contained in the TSP blacklist $bl_{\mathcal{T}}$ and potentially aborts. If not, the TSP replies with the second message of the Blum coin toss for the wallet ID by sending its own share λ'' and starts the other Blum coin toss for the serial number by sending com''_{ser}. Moreover, the TSP sends its own certificate $\text{cert}^{\mathcal{R}}_{\mathcal{T}}$ and the attributes $\mathbf{a}_{\mathcal{U}}$ the user is supposed to incorporate into its wallet. This completes the first Blum coin toss for the wallet ID. At this point the user knows all information to create the two commitments $com_{\mathcal{T}}$ and $com_{\mathcal{R}}$ of the wallet, the hidden user trapdoor e^* for the key-escrow mechanism and to create a proof π that everything is consistent. In the third message (again from user to TSP) the user sends all these elements (e^*, $com_{\mathcal{T}}$, $com_{\mathcal{R}}$ and π) to the TSP together with its share s' of the serial number as the second message of the second Blum coin toss. If the proof π verifies, the TSP creates two signatures: $\sigma_{\mathcal{T}}$ on $com_{\mathcal{T}}$ together with $\mathbf{a}_{\mathcal{U}}$ for the fixed part of the wallet, and $\sigma_{\mathcal{R}}$ on $com_{\mathcal{R}}$ together with s on the updatable part of the wallet. Please note that at this point the serial number $s \coloneqq s' \cdot s''$ is fixed and known to the TSP. In the fourth message (from TSP to user) the TSP sends the signatures $\sigma_{\mathcal{T}}$ and $\sigma_{\mathcal{R}}$ to the user and completes the Blum coin toss for the serial number by sending its share s'' together with the opening information $decom''_{\text{ser}}$. At this point the user assembles all part to obtain a fully functional wallet

$$\tau \coloneqq (s, \phi \coloneqq \text{PRF}(\lambda, 0), x^{\text{next}} \coloneqq 1, \lambda \coloneqq \lambda' + \lambda'', \mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}}, \text{cert}^{\mathcal{R}}_{\mathcal{T}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, b \coloneqq 0, u_1^{\text{next}})$$

and check its validity by executing the WalletVerification algorithm (see Section 3.4.2.4), which is presented as a separate algorithm to make it reusable.

At the end of the task, the user stores the wallet τ and outputs the serial number s of this transaction and their attributes $\mathbf{a}_{\mathcal{U}}$. The TSP stores the hidden user trapdoor *htd* and returns the serial number s of this transaction.

[21] Note that proving a statement $\exists \text{sk}_{\mathcal{U}} \in \mathbb{Z}_q : \text{pk}_{\mathcal{U}} = g_1^{\text{sk}_{\mathcal{U}}}$ instead would not help as we can only extract $g_1^{\text{sk}_{\mathcal{U}}}$ from the proof.

Debt Accumulation. When a driving car passes an RSU, the Debt Accumulation task is executed between the user and the RSU. For this task, the user needs their keypair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, the public key of the TSP $\mathsf{pk}_{\mathcal{T}}$ and their current wallet

$$\tau^{\mathrm{prev}} := (s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}^{\mathrm{prev}}, decom_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathrm{prev}}, u_1)$$

The RSU needs its own secret key $\mathsf{sk}_{\mathcal{R}}$ and certificate $\mathsf{cert}_{\mathcal{R}}$, the public key of the TSP $\mathsf{pk}_{\mathcal{T}}$ and the RSU blacklist $bl_{\mathcal{R}}$. It has also access to a pricing oracle $\mathrm{O}_{\mathrm{pricing}}(\cdot,\cdot,\cdot)$, which helps it to determine the price the user has to pay.

Analogous to the Wallet Issuing task, the RSU and the user utilize a Blum coin toss to jointly compute a fresh and random serial number s for this transaction. The detailed description of this coin toss is therefore omitted in the remaining description of this task.

The RSU starts the task by sending its certificate $\mathsf{cert}_{\mathcal{R}}$ and a fresh double-spending randomness u_2 to the user. The user checks the validity of the certificate and uses u_2 to calculate the double-spending tag $t := \mathsf{sk}_{\mathcal{U}} \cdot u_2 + u_1 \bmod \mathrm{q}$. They then compute the fraud detection ID for the current transaction as $\phi := \mathsf{PRF}(\lambda, x)$. The user then proceeds by preparing the updated wallet. Therefore, they first sample a fresh double-spending mask u_1^{next} and execute $(com'_{\mathcal{R}}, decom'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}_{\mathrm{com}}^1, (\lambda, b^{\mathrm{prev}}, u_1^{\mathrm{next}}, x))$ to commit to their wallet ID, the current balance, the fresh double-spending mask and the current counter. The user also executes $(com_{\mathrm{hid}}, decom_{\mathrm{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}_{\mathrm{com}}^1, \mathsf{sk}_{\mathcal{U}})$ to create a fresh commitment on the user's secret key. This commitment can be used at a later point in the Prove Participation task to prove to the TSP that the user behaved honestly in this transaction.

The user continues by using P2 to compute a proof π for a statement $stmt$ from the language $L_{\mathrm{gp}}^{(2)}$ defined by

$$L_{\mathrm{gp}}^{(2)} := \left\{ \begin{pmatrix} \mathsf{vk}_{\mathcal{T}}^{\mathcal{T}} \\ \mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}} \\ \phi \\ \mathbf{a}_{\mathcal{U}} \\ \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}} \\ com_{\mathrm{hid}} \\ com'_{\mathcal{R}} \\ t \\ u_2 \end{pmatrix}^{\top} \middle| \begin{array}{l} \exists\, x, \lambda, \mathsf{sk}_{\mathcal{U}}, u_1 \in \mathbb{Z}_{\mathrm{q}};\ s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, X, \Lambda, \mathsf{pk}_{\mathcal{U}}, \\ B^{\mathrm{prev}}, U_1, U_1^{\mathrm{next}}, decom_{\mathrm{hid}}, decom_{\mathcal{R}}^{\mathrm{prev}}, decom'_{\mathcal{R}}, decom_{\mathcal{T}} \in G_1; \\ \mathsf{vk}_{\mathcal{R}}^{\mathrm{prev}} \in G_1^3;\ com_{\mathcal{R}}^{\mathrm{prev}}, com_{\mathcal{T}} \in G_2;\ \sigma_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{cert\,prev}}, \\ \sigma_{\mathcal{T}} \in G_2^2 \times G_1 : \\ \\ \mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, (\Lambda, \mathsf{pk}_{\mathcal{U}}), com_{\mathcal{T}}, decom_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, \mathsf{pk}_{\mathcal{U}}, com_{\mathrm{hid}}, decom_{\mathrm{hid}}) = 1 \\ \mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, (\Lambda, B^{\mathrm{prev}}, U_1, X), com_{\mathcal{R}}^{\mathrm{prev}}, decom_{\mathcal{R}}^{\mathrm{prev}}) = 1 \\ \mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, (\Lambda, B^{\mathrm{prev}}, U_1^{\mathrm{next}}, X), com'_{\mathcal{R}}, decom'_{\mathcal{R}}) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \sigma_{\mathcal{T}}, (com_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, (com_{\mathcal{R}}^{\mathrm{prev}}, s^{\mathrm{prev}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}}, \sigma_{\mathcal{R}}^{\mathrm{cert\,prev}}, (\mathsf{vk}_{\mathcal{R}}^{\mathrm{prev}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}})) = 1 \\ \phi^{\mathrm{prev}} = \mathsf{PRF}(\lambda, x-1),\ \phi = \mathsf{PRF}(\lambda, x), \\ t = \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \\ \mathsf{pk}_{\mathcal{U}} = g_1^{\mathsf{sk}_{\mathcal{U}}},\ U_1 = g_1^{u_1},\ X = g_1^{x},\ \Lambda = g_1^{\lambda} \end{array} \right\} \tag{3.2}$$

This proof essentially shows that the wallet τ is valid, i.e., that the commitments $com_{\mathcal{T}}$ and $com_{\mathcal{R}}^{\mathrm{prev}}$ are valid, bound to the user and have valid signatures, that the certificate $\mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}$ from the previous RSU is valid and that the fraud detection ID ϕ^{prev} from the last transaction has been computed correctly. It also shows that com_{hid} is valid and contains the user's secret key, that $com_{\mathcal{R}}^{\mathrm{prev}}$ and $com'_{\mathcal{R}}$ contain the same values (except for the double-spending mask) and that the fraud detection ID ϕ and the double-spending tag t are computed correctly.

The user then sends $(\pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, com_{\mathrm{hid}}, com'_{\mathcal{R}}, t)$ to the RSU, who first checks whether the proof π verifies and the fraud detection ID ϕ is on the RSU blacklist $bl_{\mathcal{R}}$ or not. If one of the checks fails, the RSU aborts the communication with the user and takes certain measures. These measures should include instructing the connected camera to take a picture of the cheating vehicle.

If the tests have been passed, the RSU calculates with the help of the pricing oracle the price p the user has to pay, depending on factors like the user's attributes, the attributes of the current and previous RSU and auxiliary information (e.g., the time of the day, the current traffic volume). Then the RSU does its part to update the user's wallet. It blindly adds the price p to the wallet balance b and increases the PRF counter x by 1 by calculating $(com''_{\mathcal{R}}, decom''_{\mathcal{R}}) \coloneqq \mathsf{C1.Com}(\mathsf{crs}^1_{\mathrm{com}}, (0, p, 0, 1))$. Then it computes the new RSU commitment $com_{\mathcal{R}}$ by adding $com'_{\mathcal{R}}$ and $com''_{\mathcal{R}}$ (remember that C1 is homomorphic) and also signs it along with the serial number s. The RSU then sends $(com_{\mathcal{R}}, decom''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to the user. It also stores several information: The recalculation information $\omega^{\mathrm{rc}} \coloneqq (\phi, p)$ can be used at a later point in the Blacklisting & Recalculation task to recalculate the total fee of a fraudulent user. The double-spending information $\omega^{\mathrm{dsp}} \coloneqq (\phi, t, u_2)$ enables the TSP to identify the user if they use an old state of the wallet (with unchanged balance) in another transaction. The RSU's prove participation information $\omega^{\mathrm{pp}}_{\mathcal{R}} \coloneqq (s, com_{\mathrm{hid}})$ can be used later in the Prove Participation task.

The user can then calculate the remaining values needed to update the wallet, e.g., increasing the counter and the balance. They can thus construct the updated wallet

$$\tau \coloneqq (s, \phi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathrm{next}})$$

and check its validity by executing the WalletVerification algorithm (see Section 3.4.2.4).

At the end of this task, the user stores the updated wallet along with the user's prove participation information $\omega^{\mathrm{pp}}_{\mathcal{U}} \coloneqq (s, com_{\mathrm{hid}}, decom_{\mathrm{hid}})$. The RSU stores the three information ω^{rc}, ω^{dsp}, and $\omega^{\mathrm{pp}}_{\mathcal{R}}$, and outputs the user's attributes $\mathbf{a}_{\mathcal{U}}$ and the attributes $\mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}$ of the RSU from the previous transaction of the user.

Debt Clearance. After the end of a billing period, the Debt Clearance task is executed between a user and the TSP. For this task, the user needs the public key $\mathsf{pk}_{\mathcal{T}}$ of the TSP, their own keypair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ and their current wallet

$$\tau^{\mathrm{prev}} \coloneqq (s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, com^{\mathrm{prev}}_{\mathcal{R}}, decom^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathrm{prev}}, u_1).$$

The TSP needs its own public key $\mathsf{pk}_{\mathcal{T}}$.

This task is similar to the Debt Accumulation task, with the difference that the user here is not anonymous, the wallet balance is no secret and the TSP does not give the user an updated wallet. Like in Debt Accumulation, the TSP first sends a fresh double-spending randomness u_2 to the user and the user calculates the double-spending tag $t \coloneqq \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \bmod \mathrm{q}$ and the fraud detection ID ϕ for this transaction. The user continues by preparing a proof of knowledge. More precisely, P3 is used to compute a proof π for a statement $stmt$ from the language $L^{(3)}_{\mathrm{gp}}$ defined by

$$L^{(3)}_{\mathrm{gp}} \coloneqq \left\{ \begin{pmatrix} \mathsf{pk}_{\mathcal{U}} \\ \mathsf{vk}^{\mathcal{T}}_{\mathcal{T}} \\ \mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}} \\ \phi \\ \mathbf{a}_{\mathcal{U}} \\ \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}} \\ B^{\mathrm{prev}} \\ t \\ u_2 \end{pmatrix}^{\top} \middle| \begin{array}{l} \exists\, \lambda, x, u_1, \mathsf{sk}_{\mathcal{U}} \in \mathbb{Z}_{\mathrm{q}};\ \phi^{\mathrm{prev}}, s^{\mathrm{prev}}, X, \Lambda, U_1, \\ decom^{\mathrm{prev}}_{\mathcal{R}}, decom_{\mathcal{T}} \in G_1;\ \mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}} \in G_1^3;\ com^{\mathrm{prev}}_{\mathcal{R}}, com_{\mathcal{T}} \in G_2; \\ \sigma^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{cert\,prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}} \in G_2^2 \times G_1 : \\ \mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, (\Lambda, \mathsf{pk}_{\mathcal{U}}), com_{\mathcal{T}}, decom_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, (\Lambda, B^{\mathrm{prev}}, U_1, X), com^{\mathrm{prev}}_{\mathcal{R}}, decom^{\mathrm{prev}}_{\mathcal{R}}) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, (com_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, (com^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert\,prev}}_{\mathcal{R}}, (\mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}})) = 1 \\ \phi^{\mathrm{prev}} = \mathsf{PRF}(\lambda, x-1),\ \phi = \mathsf{PRF}(\lambda, x), \\ t = \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \\ \mathsf{pk}_{\mathcal{U}} = g_1^{\mathsf{sk}_{\mathcal{U}}},\ U_1 = g_1^{u_1},\ X = g_1^{x},\ \Lambda = g_1^{\lambda} \end{array} \right\} \tag{3.3}$$

The proof is a simplified version of the one in the Debt Accumulation task. The balance and the public user key are now in the statement and not in the witness and one does not need to prove anything about $com'_{\mathcal{R}}$ and com_{hid}.

The user then sends $(\text{pk}_{\mathcal{U}}, \pi, \phi, \text{a}_{\mathcal{U}}, \text{a}_{\mathcal{R}}^{\text{prev}}, b^{\text{prev}}, t)$ to the TSP. Unlike in Debt Accumulation, the balance and the user's public key are transmitted. The TSP then checks the validity of the proof and signals the user that the proof successfully verified.

At the end of the task, the TSP stores the recalculation information $\omega^{\text{rc}} \coloneqq (\phi, -b^{\text{prev}})$ and the double-spending information $\omega^{\text{dsp}} \coloneqq (\phi, t, u_2)$ and outputs the user's public key $\text{pk}_{\mathcal{U}}$ and final debt $b^{\text{bill}} \coloneqq b^{\text{prev}}$. The user just outputs their final debt $b^{\text{bill}} \coloneqq b^{\text{prev}}$ for this billing period.

Note that the wallet itself is discarded. It is expected that the user and the TSP execute the Wallet Issuing task next, to give the user a fresh wallet. After the task has ended, the TSP can issue an invoice to the user for the current billing period. The specifics of the actual payment process are out of scope.

3.4.2.3. Feature Tasks

The feature tasks *Prove Participation*, *Double-Spending Detection*, *Guilt Verification*, and *Blacklisting & Recalculation* all deal with different aspects arising from fraudulent user behavior.

Prove Participation. The Prove Participation task is used by a user to prove to the SA that they behaved honestly at a specific Debt Accumulation transaction. In the case that more than one vehicle is captured on a photograph taken by a RSU camera after a fraud occurred, this task can be used to identify the fraudulent driver.[22] The SA recovers the identities of the users captured on the photograph and executes the Prove Participation task which each of them. The user that is not able to prove that they honestly participated in a corresponding RSU transaction is found guilty.

For this task, the SA needs the set $\Omega_{\mathcal{R}}^{\text{pp}}$ of prove participation information. The entries of this set are collected by the RSUs, and we assume that each RSU periodically transmits its set to the TSP, which merges them into a large set $\Omega_{\mathcal{R}}^{\text{pp}}$. We also assume that the TSP periodically sends this set (or updates to it) to the SA, so that before a Prove Participation task is started, the SA's copy of $\Omega_{\mathcal{R}}^{\text{pp}}$ is up to date. Note that we assume that this information transfer is out-of-band, and do not model it explicitly. The SA also needs the set $S_{\mathcal{R}}^{\text{pp}}$ of all serial numbers that were recorded by the RSU that took the photo at roughly the time the photo was taken, as well as the public key $\text{pk}_{\mathcal{U}}$ of the user in question. The user needs their own internally recorded set $\Omega_{\mathcal{U}}^{\text{pp}}$ of all prove participation information.

The task itself is then simple. The SA first sends $S_{\mathcal{R}}^{\text{pp}}$ to the user who searches $\Omega_{\mathcal{U}}^{\text{pp}}$ for a prove participation information $\omega_{\mathcal{U}}^{\text{pp}}$ for which the serial number s in $\omega_{\mathcal{U}}^{\text{pp}}$ is also in $S_{\mathcal{R}}^{\text{pp}}$. If one is found, the user's output bit $\text{OUT}_{\mathcal{U}}$ is set to OK, if none is found, $\text{OUT}_{\mathcal{U}}$ is set to NOK ("not ok"). The user then sends $\omega_{\mathcal{U}}^{\text{pp}} \coloneqq (s, com_{\text{hid}}, decom_{\text{hid}})$ to the SA and the SA checks if $(s, com_{\text{hid}}) \in \Omega_{\mathcal{R}}^{\text{pp}}$, if $s \in S_{\mathcal{R}}^{\text{pp}}$ and if $decom_{\text{hid}}$ is the opening of com_{hid} under $\text{pk}_{\mathcal{U}}$. If all checks succeed, the SA's output bit OUT_{SA} is set to OK, if at least one check fails, OUT_{SA} is set to NOK. At the end of the task both parties output their bits $\text{OUT}_{\mathcal{U}}$ and

[22] Of course, the task can also be used in the case that only one vehicle was captured on the photograph to eliminate the possibility that the RSU falsely instructed the camera to take a photograph.

OUT_{SA}, respectively.[23] If the SA's output equals NOK, the user is found guilty and appropriate measures are taken (e.g., the user gets blacklisted).

Double-Spending Detection. The double-spending information ω^{dsp} collected by the RSUs are periodically transmitted to the TSP's database, which is regularly checked for two double-spending information associated with the same fraud detection ID. If the database contains two such records, the Double-Spending Detection task can be used by the TSP to extract the public key of the user these double-spending information belong to as well as a proof (such as their secret user key) that the user is guilty.

In particular, the task gets a fraud detection ID ϕ as input and searches the internal database Ω^{dsp} for two double-spending information $\omega^{\mathrm{dsp}} \coloneqq (\phi, t, u_2)$ and $\omega^{\mathrm{dsp}\prime} \coloneqq (\phi', t', u_2')$ that contain the same fraud detection ID $\phi = \phi'$ but not the same double-spending randomness $u_2 \neq u_2'$. Then the fraudulent user's secret key can be recovered as $\mathsf{sk}_{\mathcal{U}} \coloneqq (t - t') \cdot (u_2 - u_2)^{-1}$ mod q. The user's public key is then $\mathsf{pk}_{\mathcal{U}} \coloneqq g_1^{\mathsf{sk}_{\mathcal{U}}}$. The secret key $\mathsf{sk}_{\mathcal{U}}$ can be used as a proof of guilt in the Guilt Verification task, which will be described next.

Every user that is convicted of double-spending is added to the blacklist via the Blacklisting & Recalculation task and additional measures are taken (these are out of scope).

Guilt Verification. Whether a user is indeed guilty of double-spending can be verified using the Guilt Verification task.

This algorithm may be run by anyone, in particular by justice. Essentially, the algorithm checks if a given public user key $\mathsf{pk}_{\mathcal{U}}$ and a proof of guilt π match. This is easily accomplished because they match if and only if $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$ holds. This equation holds if and only if π equals the user's secret key $\mathsf{sk}_{\mathcal{U}}$ that was recovered using the Double-Spending Detection task.

Blacklisting & Recalculation. The Blacklisting & Recalculation task executed between the DR and the TSP is used to put a user on the blacklist as well as calculate their outstanding debt. There are several reasons why a user is entered on the blacklist:

(1) A user did not submit their balance at the end of the billing period.

(2) A user did not physically pay their debt after submitting their balance.

(3) A user has been convicted of double-spending.

(4) The wallet (or the vehicle including the wallet) of a user has been stolen and the user wants to prevent the thief from paying tolls from their account.

Blacklisted users are unable to get issued a new wallet and they also get photographed at every RSU they pass. They may also be punished by other means (which are out of scope).

For this task, the DR needs its keypair $(\mathsf{pk}_{\mathrm{DR}}, \mathsf{sk}_{\mathrm{DR}})$ and the public key $\mathsf{pk}_{\mathcal{U}}^{\mathrm{DR}}$ of the user to be blacklisted. The TSP needs the set $HTD_{\mathcal{U}}$ containing all hidden user trapdoors from that user for the current billing

[23] Note that after a successful (both parties output OK) execution of the Prove Participation task a single transaction can be linked to the user. But as long as the user does not get guiltlessly photographed at every RSU they pass, tracking is not possible.

period.[24] We assume that the DR and TSP agreed upon which user is going to be blacklisted before the task out-of-band.

At the beginning of the task the TSP sends $HTD_{\mathcal{U}}$ to the DR. The DR then recovers the corresponding wallet ID λ for every $htd \coloneqq (\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}, s, \lambda'', e^*) \in HTD_{\mathcal{U}}$. To this end, the DR decrypts e^* to get $(\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}^{\mathcal{T}}_{\mathcal{U}})$. Firstly, the DR checks if the decrypted public key $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}$ equals the expected public $\mathsf{pk}^{\mathsf{DR}}_{\mathcal{U}}$ of the correct user in question.[25] This way, the DR cannot be tricked into recovering the wallet ID of another (possibly innocent) user. Since each λ'_i is small ($\lambda'_i < \mathcal{B}$), the DR can compute the discrete logarithms of $(\Lambda'_0, \ldots, \Lambda'_{\ell-1})$ in a reasonable amount of time to recover $(\lambda'_0, \ldots, \lambda'_{\ell-1})$. This algorithm is also not time-critical and is expected to be executed only a few times per billing period. Therefore, the amount of required computation should be acceptable. Secondly, the DR checks if the claimed TSP's share λ'' of the wallet ID is consistent to the decrypted $\Lambda'' \overset{?}{=} g_1^{\lambda''}$. (Remember that λ'' is directly stored in htd.) The DR calculates the wallet ID as $\lambda \coloneqq \lambda'' + \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$. Finally, the DR sends $\Phi_{\mathcal{U}}$ to the TSP, which is the union of all sets $\{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\mathrm{bl}_{\mathcal{R}}})\}$ of fraud detection IDs for every $htd \in HTD_{\mathcal{U}}$. The blacklist parameter $x_{\mathrm{bl}_{\mathcal{R}}}$ is chosen in such a way that a user is expected to perform at most $x_{\mathrm{bl}_{\mathcal{R}}}$ executions of the Debt Accumulation task in a single billing period. The TSP then calculates the total toll amount the user owes for the billing period in question. To this end, the TSP calculates the subset $\Omega^{\mathrm{rc}}_{\mathcal{U}} \subseteq \Omega^{\mathrm{rc}}$ of all recalculation information that correspond to the unveiled fraud detection IDs in $\Phi_{\mathcal{U}}$. By summing up all the prices in $\Omega^{\mathrm{rc}}_{\mathcal{U}}$, the TSP can calculate the total fee b^{bill} the user owes. The DR's output of the task is simply OK, while the TSP's output is the $\Phi_{\mathcal{U}}$ of fraud detection IDs to be blacklisted and the debt b^{bill} of the user.

We assume that after the execution of this task, the fraud detection IDs $\Phi_{\mathcal{U}}$ are added to the RSU blacklist $bl_{\mathcal{R}}$ out-of-band and the user's public key $\mathsf{pk}_{\mathcal{U}}$ is added on the TSP blacklist $bl_{\mathcal{T}}$ out-of-band. In the Wallet Issuing task the TSP uses the blacklist $bl_{\mathcal{T}}$ to prevent a user that did not pay its invoice from receiving a fresh wallet. In the Debt Accumulation task a RSU checks if the fraud detection ID that the current user presents is on the RSU blacklist $bl_{\mathcal{R}}$.

3.4.2.4. Wallet Verification

The WalletVerification algorithm is depicted in Fig. 3.7. Users can verify with this algorithm that the wallet they store at the end of a transaction is valid. In particular, the algorithm verifies that the commitments $com_{\mathcal{T}}$ and $com_{\mathcal{R}}$ are valid and contain the values they are supposed to contain, that $\sigma_{\mathcal{T}}$ is a valid signature under $\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}$ of $com_{\mathcal{T}}$ and $\mathbf{a}_{\mathcal{U}}$, that $\sigma_{\mathcal{R}}$ is a valid signature under $\mathsf{sk}_{\mathcal{R}}$ of $com_{\mathcal{R}}$ and s, that the certificate $\mathsf{cert}_{\mathcal{R}}$ containing $\mathsf{vk}_{\mathcal{R}}$ is valid and that the fraud detection id ϕ was calculated using the correct values.

Of course, this algorithm can also be run by a third party to verify the validity of a wallet (since no secret keys are needed to run this algorithm).

[24] In the case that a user owns more than one vehicle they can have more than one wallet and hence more than one hidden user trapdoor is stored at the TSP for this user.

[25] Note: The public key $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}$ is stored redundantly: in clear as part of $htd \coloneqq (\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}, s, \lambda'', e^*)$ and encrypted as part of the ciphertext $e^* \coloneqq \mathsf{Enc}(\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}^{\mathcal{T}}_{\mathcal{U}})$. The DR only considers the latter version as this is protected by the non-malleability of the CCA encryption. The outer public key is only required as management information and utilized by the TSP which cannot look inside e^*. The DR ignores the outer $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}$.

```
WalletVerification($\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau$)
Parse $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}) \coloneqq \mathsf{pk}_{\mathcal{T}}$
Parse $(s, \phi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, com_{\mathcal{R}}, decom_{\mathcal{R}}, \sigma_{\mathcal{R}},$
        $\mathsf{cert}_{\mathcal{R}}, com_{\mathcal{T}}, decom_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathrm{next}}) \coloneqq \tau$
Parse $(\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\mathrm{cert}}) \coloneqq \mathsf{cert}_{\mathcal{R}}$
If
    $\mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, (g_1^{\lambda}, \mathsf{pk}_{\mathcal{U}}), com_{\mathcal{T}}, decom_{\mathcal{T}}) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \sigma_{\mathcal{T}}, (com_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 0 \vee$
    $\mathsf{C1.Open}(\mathsf{crs}_{\mathrm{com}}^1, (g_1^{\lambda}, g_1^{b}, g_1^{u_1^{\mathrm{next}}}, g_1^{x^{\mathrm{next}}}), com_{\mathcal{R}}, decom_{\mathcal{R}}) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (com_{\mathcal{R}}, s)) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}}, \sigma_{\mathcal{R}}^{\mathrm{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0 \vee$
    $\mathsf{PRF}(\lambda, x^{\mathrm{next}} - 1) \neq \phi$
then return 0
else return 1
```

Figure 3.7.: Algorithm for wallet verification

3.4.3. Full Protocol

We now present the complete protocol Π_{P4TC}. Afterwards we briefly outline how the used building blocks can be instantiated.

Protocol Π_{P4TC}

State of the Parties:

- The DR stores:
  - CRS $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}_{\mathrm{com}}^1, \mathsf{crs}_{\mathrm{com}}^2, \mathsf{crs}_{\mathrm{pok}})$
  - Encryption keypair $(\mathsf{pk}_{\mathrm{DR}}, \mathsf{sk}_{\mathrm{DR}})$
- The SA stores:
  - Set Ω^{dsp} of double-spending information (initially empty). We assume that the TSP periodically sends updates to Ω^{dsp} out-of-band to the SA, such that before a Prove Participation task is started, the SA's copy of Ω^{dsp} is up-to-date.
- The TSP stores:
  - CRS $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}_{\mathrm{com}}^1, \mathsf{crs}_{\mathrm{com}}^2, \mathsf{crs}_{\mathrm{pok}})$
  - Own public and private key $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) \coloneqq \big((\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\mathrm{cert}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}), (\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathrm{cert}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}})\big)$
  - A self-signed certificate $\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}} \coloneqq (\mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}, \mathbf{a}_{\mathcal{T}}, \sigma_{\mathcal{T}}^{\mathrm{cert}})$
  - A set *HTD* of hidden user trapdoors (initially empty)
  - A (partial) mapping $\{\mathsf{vk}_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}\}$ of RSU attributes (initially empty)
  - Sets Ω^{dsp}, Ω^{rc} and $\Omega_{\mathcal{R}}^{\mathrm{pp}}$ of double-spending information, recalculation information and prove participation information (all initially empty)
- Each RSU stores:
  - CRS $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}_{\mathrm{com}}^1, \mathsf{crs}_{\mathrm{com}}^2, \mathsf{crs}_{\mathrm{pok}})$
  - Own signing keypair $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$
  - A certificate $\mathsf{cert}_{\mathcal{R}}$ signed by the TSP

  - Sets Ω^{dsp}, Ω^{rc} and $\Omega^{\text{pp}}_{\mathcal{R}}$ of double-spending information, recalculation information and prove participation information (all initially empty). We assume that each RSU periodically transmits these sets out-of-band to the TSP, who then stores the entries in the TSP's copy of these sets.
- Each user stores:
  - CRS $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})$
  - Own public and private key $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
  - A set $\{\tau\}$ of all past tokens issued to them (initially empty)
  - A set $\Omega^{\text{pp}}_{\mathcal{U}}$ of prove participation information (initially empty)

System Setup ($\mathcal{F}_{\text{CRS}}$):
- Input: $(1^\lambda, \mathcal{B})$
- Behavior:
  (1) $\mathsf{gp} \coloneqq (G_1, G_2, G_T, e, q, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^\lambda)$
  (2) $\mathsf{crs}^1_{\mathsf{com}} \leftarrow \mathsf{C1.Gen}(\mathsf{gp})$
  (3) $\mathsf{crs}^2_{\mathsf{com}} \leftarrow \mathsf{C2.Gen}(\mathsf{gp})$
  (4) $\mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{POK.Setup}(\mathsf{gp})$
  (5) Store $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})$

DR Registration:
- Input DR: (Register)
- Behavior:
  (1) If a keypair $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}})$ has already been recorded, output $\perp$ and abort
  (2) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})$ and store crs
  (3) Generate encryption keypair: $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}}) \leftarrow \mathsf{E.Gen}(\mathsf{gp})$
  (4) Store $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}})$ and call $\mathcal{G}_{\text{bb}}$ with input (Register, $\mathsf{pk}_{\mathsf{DR}}$)
- Output DR: $(\mathsf{pk}_{\mathsf{DR}})$

TSP Registration:
- Input TSP: (Register, $\mathsf{a}_{\mathcal{T}}$)
- Behavior:
  (1) If a keypair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ has already been recorded, output $\perp$ and abort
  (2) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})$ and store crs
  (3) Generate signing keypair $(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$ for signing TSP commitments during Wallet Issuing
  (4) Generate signing keypair $(\mathsf{vk}^{\text{cert}}_{\mathcal{T}}, \mathsf{sk}^{\text{cert}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$ for certifying RSUs
  (5) Generate signing keypair $(\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$ for signing RSU commitments like a RSU
  (6) $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) \coloneqq \big((\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\text{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}), (\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\text{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}})\big)$
  (7) Certify own "RSU signing key":
    (a) $\sigma^{\text{cert}}_{\mathcal{T}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\text{cert}}_{\mathcal{T}}, (\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}))$
    (b) $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}} \coloneqq (\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{T}})$
  (8) Store $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ and $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$ and call $\mathcal{G}_{\text{bb}}$ with input (Register, $\mathsf{pk}_{\mathcal{T}}$)
- Output TSP: $(\mathsf{pk}_{\mathcal{T}})$

RSU Registration:
- Input RSU: (Register)
- Behavior:
  (1) If a keypair $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ has already been stored, output $\perp$ and abort

(2) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\text{crs} \coloneqq (\text{gp}, \mathcal{B}, \text{crs}^1_{\text{com}}, \text{crs}^2_{\text{com}}, \text{crs}_{\text{pok}})$ and store crs
(3) Generate signing keypair $(\text{vk}_{\mathcal{R}}, \text{sk}_{\mathcal{R}}) \leftarrow \text{S.Gen}(\text{gp})$
(4) Store $(\text{vk}_{\mathcal{R}}, \text{sk}_{\mathcal{R}})$ and call $\mathcal{G}_{\text{bb}}$ with input $(\textsc{Register}, \text{vk}_{\mathcal{R}})$

- Output RSU: $(\text{vk}_{\mathcal{R}})$

User Registration:

- Input User: $(\textsc{Register})$
- Behavior:

(1) If a keypair $(\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$ has already been stored, output $\perp$ and abort
(2) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\text{crs} \coloneqq (\text{gp}, \mathcal{B}, \text{crs}^1_{\text{com}}, \text{crs}^2_{\text{com}}, \text{crs}_{\text{pok}})$ and store crs
(3) Parse $(G_1, G_2, G_{\text{T}}, e, \text{q}, g_1, g_2) \coloneqq \text{gp}$
(4) Generate keypair:
  (a) $y \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\text{q}}$
  (b) $(\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}}) \coloneqq (g_1^y, y)$
(5) Store $(\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$ and call $\mathcal{G}_{\text{bb}}$ with input $(\textsc{Register}, \text{pk}_{\mathcal{U}})$

- Output User: $(\text{pk}_{\mathcal{U}})$

RSU Certification:

- Input RSU: $(\textsc{Certify})$
- Input TSP: $(\textsc{Certify}, \mathbf{a}_{\mathcal{R}})$
- Behavior:

(1) TSP:
  (a) Load $\text{crs} \coloneqq (\text{gp}, \mathcal{B}, \text{crs}^1_{\text{com}}, \text{crs}^2_{\text{com}}, \text{crs}_{\text{pok}})^{\perp}$
  (b) Load $(\text{pk}_{\mathcal{T}}, \text{sk}_{\mathcal{T}})^{\perp}$
  (c) Receive $\text{vk}_{\mathcal{R}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for PID $pid_{\mathcal{R}}{}^{\perp}$
  (d) Check that no mapping $\text{vk}_{\mathcal{R}} \mapsto \mathbf{a}'_{\mathcal{R}}$ has been registered before, else output $\perp$ and abort
  (e) Parse $\big((\text{vk}^{\mathcal{T}}_{\mathcal{T}}, \text{vk}^{\text{cert}}_{\mathcal{T}}, \text{vk}^{\mathcal{R}}_{\mathcal{T}}), (\text{sk}^{\mathcal{T}}_{\mathcal{T}}, \text{sk}^{\text{cert}}_{\mathcal{T}}, \text{sk}^{\mathcal{R}}_{\mathcal{T}})\big) \coloneqq (\text{pk}_{\mathcal{T}}, \text{sk}_{\mathcal{T}})$
  (f) Create RSU certificate:
    (i) $\sigma^{\text{cert}}_{\mathcal{R}} \leftarrow \text{S.Sign}(\text{sk}^{\text{cert}}_{\mathcal{T}}, (\text{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$
    (ii) $\text{cert}_{\mathcal{R}} \coloneqq (\text{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\text{cert}}_{\mathcal{R}})$
  (g) Store mapping of RSU attributes $\text{vk}_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}$
  (h) Send $(\text{cert}_{\mathcal{R}})$ to the RSU
(2) RSU:
  (a) Load $\text{crs} \coloneqq (\text{gp}, \mathcal{B}, \text{crs}^1_{\text{com}}, \text{crs}^2_{\text{com}}, \text{crs}_{\text{pok}})^{\perp}$
  (b) Load $(\text{vk}_{\mathcal{R}}, \text{sk}_{\mathcal{R}}).^{\perp}$
  (c) Receive $\text{pk}_{\mathcal{T}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for PID $pid_{\mathcal{T}}{}^{\perp}$
  (d) Parse $(\text{vk}^{\mathcal{T}}_{\mathcal{T}}, \text{vk}^{\text{cert}}_{\mathcal{T}}, \text{vk}^{\mathcal{R}}_{\mathcal{T}}) \coloneqq \text{pk}_{\mathcal{T}}$ and $(\text{vk}'_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\text{cert}}_{\mathcal{R}}) \coloneqq \text{cert}_{\mathcal{R}}$
  (e) Check validity of certificate: If $\text{S.Vfy}(\text{vk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{R}}, (\text{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$, then return $\perp$
  (f) Store certificate $\text{cert}_{\mathcal{R}}$

- Output RSU: $(\mathbf{a}_{\mathcal{R}})$
- Output TSP: (OK)

Wallet Issuing:

- Input User: $(\textsc{Issue})$
- Input TSP: $(\textsc{Issue}, \mathbf{a}_{\mathcal{U}}, bl_{\mathcal{T}})$
- Behavior:

```
(1) User:
    (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) := \mathsf{gp}$
    (b) Load $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})^{\perp}$
    (c) Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\mathcal{G}_{\mathrm{bb}}$ for PID $pid_{\mathcal{T}}{}^{\perp}$ and parse $(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$
    (d) Receive $\mathsf{pk}_{\mathsf{DR}}$ from the bulletin-board $\mathcal{G}_{\mathrm{bb}}$ for PID $pid_{\mathsf{DR}}{}^{\perp}$
    (e) Pick first half of the serial number: $s' \xleftarrow{R} G_1$
    (f) $\lambda'_i \xleftarrow{R} \{0, \dots, \mathcal{B} - 1\}$ for $i \in \{0, \dots, \ell - 1\}$
    (g) $\lambda' := \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$
    (h) $(com'_{\mathrm{seed}}, decom'_{\mathrm{seed}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}^1_{\mathsf{com}}, \lambda')$
    (i) Send $(\mathsf{pk}_{\mathcal{U}}, com'_{\mathrm{seed}})$ to the TSP
(2) TSP:
    (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) := \mathsf{gp}$
    (b) Load $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ and $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}{}^{\perp}$ and parse $(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{sk}_{\mathcal{T}}$
    (c) Receive $\mathsf{pk}_{\mathsf{DR}}$ from the bulletin-board $\mathcal{G}_{\mathrm{bb}}$ for PID $pid_{\mathsf{DR}}{}^{\perp}$
    (d) If $\mathsf{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$, then return BLACKLISTED
    (e) Pick second half of the serial number: $s'' \xleftarrow{R} G_1$
    (f) $(com''_{\mathrm{ser}}, decom''_{\mathrm{ser}}) \leftarrow \mathsf{C2.Com}(\mathsf{crs}^2_{\mathsf{com}}, s'')$
    (g) $\lambda'' \xleftarrow{R} \mathbb{Z}_{\mathsf{q}}$
    (h) $\Lambda'' := g_1^{\lambda''}$
    (i) Send $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}, com''_{\mathrm{ser}}, \lambda'')$ to the user
(3) User:
    (a) Check validity of "RSU" certificate:
        (i) Parse $(\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}}, \sigma^{\mathsf{cert}}_{\mathcal{T}}) := \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$
        (ii) If $\mathsf{S.Vfy}(\mathsf{vk}^{\mathsf{cert}}_{\mathcal{T}}, \sigma^{\mathsf{cert}}_{\mathcal{T}}, (\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}})) = 0$, then return $\perp$
    (b) $\lambda := \lambda' + \lambda''$
    (c) $\Lambda := g_1^{\lambda}, \quad \Lambda' := g_1^{\lambda'}, \quad \Lambda'' := g_1^{\lambda''}, \quad \Lambda'_i := g_1^{\lambda'_i}$ for $i \in \{0, \dots, \ell - 1\}$
    (d) $r_1, r_2 \xleftarrow{R} \mathbb{Z}_{\mathsf{q}}$
    (e) $e^* \leftarrow \mathsf{E.Enc}(\mathsf{pk}_{\mathsf{DR}}, (\Lambda'_0, \dots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}_{\mathcal{U}}); r_1, r_2)$
    (f) $u_1^{\mathrm{next}} \xleftarrow{R} \mathbb{Z}_{\mathsf{q}}$
    (g) $(com_{\mathcal{T}}, decom_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}^1_{\mathsf{com}}, (\lambda, \mathsf{sk}_{\mathcal{U}}))$
    (h) $(com_{\mathcal{R}}, decom_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}^1_{\mathsf{com}}, (\lambda, 0, u_1^{\mathrm{next}}, 1))$
    (i) $stmt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathsf{DR}}, e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, com'_{\mathrm{seed}}, \Lambda'', \lambda'')$
    (j) $wit := (\lambda, \lambda', \lambda'_0, \dots, \lambda'_{\ell-1}, r_1, r_2, \Lambda, \Lambda', \Lambda'_0, \dots, \Lambda'_{\ell-1}, g_1^{u_1^{\mathrm{next}}}, decom_{\mathcal{T}}, decom_{\mathcal{R}}, decom'_{\mathrm{seed}}, g_2^{\mathsf{sk}_{\mathcal{U}}})$
    (k) $\pi \leftarrow \mathsf{P1.Prove}(\mathsf{crs}_{\mathsf{pok}}, stmt, wit)$ (cp. Eq. (3.1) for $L^{(1)}_{\mathrm{gp}}$)
    (l) Send $(s', e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, \pi)$ to the TSP
(4) TSP:
    (a) $s := s' \cdot s''$
    (b) $stmt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathsf{DR}}, e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, com'_{\mathrm{seed}}, \Lambda'', \lambda'')$
    (c) If $\mathsf{P1.Verify}(\mathsf{crs}_{\mathsf{pok}}, stmt, \pi) = 0$, then return $\perp$
    (d) $\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (com_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}))$
    (e) $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (com_{\mathcal{R}}, s))$
    (f) $htd := (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$
    (g) Store $htd$ in $HTD$
```

   (h) Send $(s'', \mathit{decom}''_{\mathrm{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}})$ to the user

(5) User:

   (a) If $\mathsf{C2.Open}(\mathsf{crs}^2_{\mathrm{com}}, s'', \mathit{com}''_{\mathrm{ser}}, \mathit{decom}''_{\mathrm{ser}}) = 0$, then return $\perp$

   (b) $s := s' \cdot s''$

   (c) $\tau := (s, \mathsf{PRF}(\lambda, 0), 1, \lambda, \mathsf{a}_{\mathcal{U}}, \mathit{com}_{\mathcal{R}}, \mathit{decom}_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathit{com}_{\mathcal{T}}, \mathit{decom}_{\mathcal{T}}, \sigma_{\mathcal{T}}, 0, u_1^{\mathrm{next}})$

   (d) Run the code of $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$ (cp. Fig. 3.7)

   (e) If $\mathsf{WalletVerification}$ returns 0, output $\perp$ and abort

   (f) Store τ

- Output User: $(s, \mathsf{a}_{\mathcal{U}})$
- Output TSP: (s)

Debt Accumulation:

- Input User: (PayToll, s^{prev})
- Input RSU: (PayToll, $\mathit{bl}_{\mathcal{R}}$)
- Behavior:

(1) RSU:

   (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathrm{com}}, \mathsf{crs}^2_{\mathrm{com}}, \mathsf{crs}_{\mathrm{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) := \mathsf{gp}$

   (b) Load $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})^{\perp}$

   (c) Load $\mathsf{cert}_{\mathcal{R}}{}^{\perp}$ and parse $(\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$

   (d) Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\mathcal{G}_{\mathrm{bb}}$ for PID $\mathit{pid}_{\mathcal{T}}{}^{\perp}$ and parse $(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$

   (e) Pick second half of the serial number: $s'' \overset{\mathrm{R}}{\leftarrow} G_1$

   (f) $u_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathsf{q}}$

   (g) $(\mathit{com}''_{\mathrm{ser}}, \mathit{decom}''_{\mathrm{ser}}) \leftarrow \mathsf{C2.Com}(\mathsf{crs}^2_{\mathrm{com}}, s'')$

   (h) Send $(u_2, \mathit{com}''_{\mathrm{ser}}, \mathsf{cert}_{\mathcal{R}})$ to the user

(2) User:

   (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathrm{com}}, \mathsf{crs}^2_{\mathrm{com}}, \mathsf{crs}_{\mathrm{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) := \mathsf{gp}$

   (b) Load $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})^{\perp}$

   (c) Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\mathcal{G}_{\mathrm{bb}}$ for PID $\mathit{pid}_{\mathcal{T}}{}^{\perp}$ and parse $(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$

   (d) Load τ^{prev} for serial number $s^{\mathrm{prev}\perp}$ and parse $(s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x, \lambda, \mathsf{a}_{\mathcal{U}}, \mathit{com}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{decom}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}, \mathit{com}_{\mathcal{T}}, \mathit{decom}_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathrm{prev}}, u_1) := \tau^{\mathrm{prev}}$

   (e) Parse $(\mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \mathsf{a}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{cert\,prev}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}$

   (f) Check validity of RSU certificate:

      (i) Parse $(\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$

      (ii) If $\mathsf{S.Vfy}(\mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}, (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}})) = 0$, then return $\perp$

   (g) $\phi := \mathsf{PRF}(\lambda, x)$

   (h) Pick first half of the serial number: $s' \overset{\mathrm{R}}{\leftarrow} G_1$

   (i) $u_1^{\mathrm{next}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathsf{q}}$

   (j) $(\mathit{com}'_{\mathcal{R}}, \mathit{decom}'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}^1_{\mathrm{com}}, (\lambda, b^{\mathrm{prev}}, u_1^{\mathrm{next}}, x))$

   (k) $t := \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \bmod \mathsf{q}$

   (l) $(\mathit{com}_{\mathrm{hid}}, \mathit{decom}_{\mathrm{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}^1_{\mathrm{com}}, \mathsf{sk}_{\mathcal{U}})$

   (m) $\mathit{stmt} := (\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{com}_{\mathrm{hid}}, \mathit{com}'_{\mathcal{R}}, t, u_2)$

   (n) $\mathit{wit} := (x, \lambda, \mathsf{sk}_{\mathcal{U}}, u_1, s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, g_1^x, g_1^{\lambda}, \mathsf{pk}_{\mathcal{U}}, g_1^{b^{\mathrm{prev}}}, g_1^{u_1}, g_1^{u_1^{\mathrm{next}}}, \mathit{decom}_{\mathrm{hid}}, \mathit{decom}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{decom}'_{\mathcal{R}}, \mathit{decom}_{\mathcal{T}}, \mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{com}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{com}_{\mathcal{T}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{cert\,prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}})$

   (o) $\pi \leftarrow \mathsf{P2.Prove}(\mathsf{crs}_{\mathrm{pok}}, \mathit{stmt}, \mathit{wit})$ (cp. Eq. (3.2) for $L^{(2)}_{\mathrm{gp}}$)

   (p) Send $(s', \pi, \phi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\mathrm{prev}}_{\mathcal{R}}, \mathit{com}_{\mathrm{hid}}, \mathit{com}'_{\mathcal{R}}, t)$ to the RSU

(3) RSU:
   (a) $\mathit{stmt} := (\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathsf{prev}}, \mathit{com}_{\mathsf{hid}}, \mathit{com}'_{\mathcal{R}}, t, u_2)$
   (b) If $\mathsf{P2.Verify}(\mathsf{crs}_{\mathsf{pok}}, \mathit{stmt}, \pi) = 0$, then return $\perp$
   (c) If $\phi \in \mathit{bl}_{\mathcal{R}}$, then return BLACKLISTED
   (d) *Obtain the price from the pricing oracle based on* $\mathbf{a}_{\mathcal{U}}$, $\mathbf{a}_{\mathcal{R}}$, $\mathbf{a}_{\mathcal{R}}^{\mathit{prev}}$ *and possibly other public-verifiable, environmental information*: $p \leftarrow \mathsf{O}_{\mathsf{pricing}}(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\mathsf{prev}})$
   (e) $s := s' \cdot s''$
   (f) $(\mathit{com}''_{\mathcal{R}}, \mathit{decom}''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{crs}_{\mathsf{com}}^{1}, (0, p, 0, 1))$
   (g) $\mathit{com}_{\mathcal{R}} := \mathit{com}'_{\mathcal{R}} \cdot \mathit{com}''_{\mathcal{R}}$
   (h) $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}_{\mathcal{R}}, (\mathit{com}_{\mathcal{R}}, s))$
   (i) $\omega^{\mathsf{dsp}} := (\phi, t, u_2)$
   (j) $\omega^{\mathsf{rc}} := (\phi, p)$
   (k) $\omega_{\mathcal{R}}^{\mathsf{pp}} := (s, \mathit{com}_{\mathsf{hid}})$
   (l) Store ω^{dsp} in Ω^{dsp}, store ω^{rc} in Ω^{rc}, and store $\omega_{\mathcal{R}}^{\mathsf{pp}}$ in $\Omega_{\mathcal{R}}^{\mathsf{pp}}$
   (m) Send $(s'', \mathit{decom}''_{\mathsf{ser}}, \mathit{com}_{\mathcal{R}}, \mathit{decom}''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to the user

(4) User:
   (a) If $\mathsf{C2.Open}(\mathsf{crs}_{\mathsf{com}}^{2}, s'', \mathit{com}''_{\mathsf{ser}}, \mathit{decom}''_{\mathsf{ser}}) = 0$, then return $\perp$
   (b) $s := s' \cdot s''$
   (c) $\mathit{decom}_{\mathcal{R}} := \mathit{decom}'_{\mathcal{R}} \cdot \mathit{decom}''_{\mathcal{R}}$
   (d) $b := b^{\mathsf{prev}} + p$
   (e) $x^{\mathsf{next}} := x + 1$
   (f) $\tau := (s, \phi, x^{\mathsf{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, \mathit{com}_{\mathcal{R}}, \mathit{decom}_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, \mathit{com}_{\mathcal{T}}, \mathit{decom}_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathsf{next}})$
   (g) $\omega_{\mathcal{U}}^{\mathsf{pp}} := (s, \mathit{com}_{\mathsf{hid}}, \mathit{decom}_{\mathsf{hid}})$
   (h) Run the code of $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$ (cp. Fig. 3.7)
   (i) If WalletVerification returns 0, output $\perp$ and abort
   (j) Store τ and store $\omega_{\mathcal{U}}^{\mathsf{pp}}$ in $\Omega_{\mathcal{U}}^{\mathsf{pp}}$

- Output User: $(s, \mathbf{a}_{\mathcal{R}}, p, b)$
- Output RSU: $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathsf{prev}})$

Debt Clearance:

- Input User: (ClearDebt, s^{prev})
- Input TSP: (ClearDebt)
- Behavior:

(1) TSP:
   (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}_{\mathsf{com}}^{1}, \mathsf{crs}_{\mathsf{com}}^{2}, \mathsf{crs}_{\mathsf{pok}})^{\perp}$ and parse $(G_1, G_2, G_{\mathrm{T}}, e, \mathrm{q}, g_1, g_2) := \mathsf{gp}$
   (b) Load $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})^{\perp}$ and parse $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$
   (c) $u_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathrm{q}}$
   (d) Send (u_2) to the user

(2) User:
   (a) Load $\mathsf{crs} := (\mathsf{gp}, \mathcal{B}, \mathsf{crs}_{\mathsf{com}}^{1}, \mathsf{crs}_{\mathsf{com}}^{2}, \mathsf{crs}_{\mathsf{pok}})^{\perp}$ and parse $(G_1, G_2, G_{\mathrm{T}}, e, \mathrm{q}, g_1, g_2) := \mathsf{gp}$
   (b) Load $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})^{\perp}$
   (c) Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\mathcal{G}_{\mathsf{bb}}$ for PID $\mathit{pid}_{\mathcal{T}}{}^{\perp}$ and parse $(\mathsf{vk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$
   (d) Load τ^{prev} for serial number $s^{\mathsf{prev}\perp}$ and parse $(s^{\mathsf{prev}}, \phi^{\mathsf{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, \mathit{com}_{\mathcal{R}}^{\mathsf{prev}}, \mathit{decom}_{\mathcal{R}}^{\mathsf{prev}}, \sigma_{\mathcal{R}}^{\mathsf{prev}}, \mathsf{cert}_{\mathcal{R}}^{\mathsf{prev}}, \mathit{com}_{\mathcal{T}}, \mathit{decom}_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathsf{prev}}, u_1) := \tau^{\mathsf{prev}}$
   (e) Parse $(\mathsf{vk}_{\mathcal{R}}^{\mathsf{prev}}, \mathbf{a}_{\mathcal{R}}^{\mathsf{prev}}, \sigma_{\mathcal{R}}^{\mathsf{cert prev}}) := \mathsf{cert}_{\mathcal{R}}^{\mathsf{prev}}$
   (f) $\phi := \mathsf{PRF}(\lambda, x)$

(g) $t \coloneqq \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \bmod \mathsf{q}$
(h) $\mathit{stmt} \coloneqq (\mathsf{pk}_{\mathcal{U}}, \mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{prev}}}, t, u_2)$
(i) $\mathit{wit} \coloneqq (x, \lambda, \mathsf{sk}_{\mathcal{U}}, u_1, s^{\text{prev}}, \phi^{\text{prev}}, g_1^x, g_1^\lambda, g_1^{u_1}, \mathit{decom}^{\text{prev}}_{\mathcal{R}}, \mathit{decom}_{\mathcal{T}}, \mathsf{vk}^{\text{prev}}_{\mathcal{R}}, \mathit{com}^{\text{prev}}_{\mathcal{R}}, \mathit{com}_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{certprev}}_{\mathcal{R}}, \sigma_{\mathcal{T}})$
(j) $\pi \leftarrow \mathsf{P3.Prove}(\mathsf{crs}_{\text{pok}}, \mathit{stmt}, \mathit{wit})$ (cp. Eq. (3.3) for $L^{(3)}_{\text{gp}}$)
(k) Send $(\mathsf{pk}_{\mathcal{U}}, \pi, \phi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{prev}}, t)$ to the TSP

(3) TSP:
(a) $\mathit{stmt} \coloneqq (\mathsf{pk}_{\mathcal{U}}, \mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{prev}}}, t, u_2)$
(b) If $\mathsf{P3.Verify}(\mathsf{crs}_{\text{pok}}, \mathit{stmt}, \pi) = 0$, then return $\perp$
(c) $b^{\text{bill}} \coloneqq b^{\text{prev}}$
(d) $\omega^{\text{rc}} \coloneqq (\phi, -b^{\text{bill}})$
(e) $\omega^{\text{dsp}} \coloneqq (\phi, t, u_2)$
(f) Store ω^{rc} in Ω^{rc} and store ω^{dsp} in Ω^{dsp}
(g) Send (OK) to the user

(4) User:
(a) $b^{\text{bill}} \coloneqq b^{\text{prev}}$

- Output User: (b^{bill})
- Output TSP: $(\mathsf{pk}_{\mathcal{U}}, \phi, b^{\text{bill}})$

Prove Participation:

- Input User: (ProveParticipation)
- Input SA: (ProveParticipation, $\mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}$)
- Behavior:

(1) SA:
(a) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\text{com}}, \mathsf{crs}^2_{\text{com}}, \mathsf{crs}_{\text{pok}})$
(b) Load the set $\Omega^{\text{pp}}_{\mathcal{R}}$ of all prove participation transaction information
(c) Send $(S^{\text{pp}}_{\mathcal{R}})$ to the user

(2) User:
(a) Load the set $\Omega^{\text{pp}}_{\mathcal{U}}$ of all prove participation transaction information
(b) If $\exists\, \omega^{\text{pp}}_{\mathcal{U}} = (s, \mathit{com}_{\text{hid}}, \mathit{decom}_{\text{hid}}) \in \Omega^{\text{pp}}_{\mathcal{U}}$ with $s \in S^{\text{pp}}_{\mathcal{R}}$,
then $\textsc{out}_{\mathcal{U}} \coloneqq \text{OK}$
else $\textsc{out}_{\mathcal{U}} \coloneqq \text{NOK}$
(c) Send $(s, \mathit{com}_{\text{hid}}, \mathit{decom}_{\text{hid}})$ to the SA

(3) SA:
(a) $\textsc{out}_{\mathcal{SA}} \coloneqq \text{OK}$
(b) $\omega^{\text{pp}}_{\mathcal{R}} \coloneqq (s, \mathit{com}_{\text{hid}})$
(c) If $\omega^{\text{pp}}_{\mathcal{R}} \notin \{(s^*, \mathit{com}^*_{\text{hid}}) \mid (s^*, \mathit{com}^*_{\text{hid}}) \in \Omega^{\text{pp}}_{\mathcal{R}} \wedge s^* \in S^{\text{pp}}_{\mathcal{R}}\}$, then $\textsc{out}_{\mathcal{SA}} \coloneqq \text{NOK}$
(d) If $\mathsf{C1.Open}(\mathsf{crs}^1_{\text{com}}, \mathsf{pk}_{\mathcal{U}}, \mathit{com}_{\text{hid}}, \mathit{decom}_{\text{hid}}) = 0$, then $\textsc{out}_{\mathcal{SA}} \coloneqq \text{NOK}$

- Output User: $(\textsc{out}_{\mathcal{U}})$
- Output SA: $(\textsc{out}_{\mathcal{SA}})$

Double-Spending Detection:

- Input TSP: (ScanForFraud, ϕ)
- Behavior:

(1) Load $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\text{com}}, \mathsf{crs}^2_{\text{com}}, \mathsf{crs}_{\text{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathsf{q}, g_1, g_2) \coloneqq \mathsf{gp}$
(2) Load the set Ω^{dsp} of all double-spending information

(3) Pick double-spending information $\omega^{\text{dsp}} \coloneqq (\phi, t, u_2)$ and $\omega^{\text{dsp}\prime} \coloneqq (\phi', t', u_2')$ from the database Ω^{dsp}, such that $\phi = \phi'$ and $u_2 \neq u_2'$$^{\perp}$

(4) $\mathsf{sk}_{\mathcal{U}} \coloneqq (t - t') \cdot (u_2 - u_2')^{-1} \bmod \mathrm{q}$

(5) $\mathsf{pk}_{\mathcal{U}} \coloneqq g_1^{\mathsf{sk}_{\mathcal{U}}}$

(6) $\pi \coloneqq \mathsf{sk}_{\mathcal{U}}$

- Output TSP: $(\mathsf{pk}_{\mathcal{U}}, \pi)$

Guilt Verification:

- Input some party P: (VerifyGuilt, $\mathsf{pk}_{\mathcal{U}}, \pi$)
- Behavior:

(1) Query $\mathcal{F}_{\text{CRS}}$ to obtain $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})$ and parse $(G_1, G_2, G_T, e, \mathrm{q}, g_1, g_2) \coloneqq \mathsf{gp}$

(2) Receive $pid_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for key $\mathsf{pk}_{\mathcal{U}}$$^{\perp}$

(3) If $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$,
then OUT $\coloneqq$ OK
else OUT $\coloneqq$ NOK

- Output P: (OUT)

Blacklisting & Recalculation:

- Input DR: (BlacklistUser, $\mathsf{pk}^{\mathsf{DR}}_{\mathcal{U}}$)
- Input TSP: (BlacklistUser, $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}$)
- Behavior:

```
(1) TSP:
    (a) Load the set $HTD$ of all hidden user trapdoors
    (b) $HTD_{\mathcal{U}} \coloneqq \{htd \mid (\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}, \cdot, \cdot, \cdot) \in HTD\}$
    (c) Send $(HTD_{\mathcal{U}})$ to the DR
(2) DR:
    (a) Load $\mathsf{crs} \coloneqq (\mathsf{gp}, \mathcal{B}, \mathsf{crs}^1_{\mathsf{com}}, \mathsf{crs}^2_{\mathsf{com}}, \mathsf{crs}_{\mathsf{pok}})^{\perp}$ and parse $(G_1, G_2, G_T, e, \mathrm{q}, g_1, g_2) \coloneqq \mathsf{gp}$
    (b) Load $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}})^{\perp}$
    (c) Receive $\mathsf{pk}^{\mathsf{DR}}_{\mathcal{U}}$ from the bulletin-board $\mathcal{G}_{\text{bb}}$ for PID $pid^{\mathsf{DR}\,\perp}_{\mathcal{U}}$
    (d) $\Phi_{\mathcal{U}} \coloneqq \emptyset$
    (e) For all $htd \in HTD_{\mathcal{U}}$:
        (i) Parse $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}, s, \lambda'', e^*) \coloneqq htd$
        (ii) $(\Lambda'_0, \dots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}) \coloneqq \mathsf{E.Dec}(\mathsf{sk}_{\mathsf{DR}}, e^*)$
        (iii) If $\big((\text{decryption fails}) \vee (\Lambda'' \neq g_1^{\lambda''}) \vee (\mathsf{pk}^{\mathsf{DR}}_{\mathcal{U}} \neq \mathsf{pk}^{\mathcal{T}}_{\mathcal{U}})\big)$, then return $\perp$
        (iv) $\lambda \coloneqq \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$
        (v) $\Phi_{\mathcal{U}} \coloneqq \Phi_{\mathcal{U}} \cup \{\mathsf{PRF}(\lambda, 0), \dots, \mathsf{PRF}(\lambda, x_{\mathrm{bl}_{\mathcal{R}}})\}$
    (f) Send $(\Phi_{\mathcal{U}})$ to the TSP
(3) TSP:
    (a) Load the set $\Omega^{\text{rc}}$ of all recalculation information
    (b) Let $\Omega^{\text{rc}}_{\mathcal{U}}$ be the subset of entries $\omega^{\text{rc}} = (\phi, p)$ with fraud detection IDs $\phi \in \Phi_{\mathcal{U}}$
    (c) Recalculate the user's debt: $b^{\text{bill}} \coloneqq \sum_{\omega^{\text{rc}} \in \Omega^{\text{rc}}_{\mathcal{U}}} p$
```

- Output DR: (OK)
- Output TSP: $(b^{\text{bill}}, \Phi_{\mathcal{U}})$

$^{\perp}$If this does not exist, then output $\perp$ and abort.

3.4.4. Building Block Instantiation

P4TC is designed to use *Groth–Sahai (GS) proofs* [EG14; GS08] as NIZK proofs. Therefore, all building blocks about which statements are to be proved in zero-knowledge must be compatible with GS proofs. In particular, all these building blocks must be algebraic. We now present possible instantiations for all building blocks that satisfy the required security guarantees (cp. Theorem A.1) and, if necessary, are compatible with GS proofs.

NIZK Proofs. We choose the SXDH-based Groth–Sahai (GS) proof system [EG14; GS08] as our NIZK, as it allows for very efficient proofs (under standard assumptions). On the other hand, GS comes with some drawbacks, which makes applying it sometimes pretty tricky: It only works for algebraic languages containing certain types of equations,[26] it is not always zero-knowledge, and F_{gp} is not always the identity function. When choosing our remaining building blocks and forming equations we ensured that they fit into this framework. Likewise, we ensured that the ZK-property holds for the languages we consider.

For proving correctness of the computations taking place on the user's side we need three different instantiations of the GS proof system, denoted by P1, P2 and P3, respectively. The corresponding functions $F_{\text{gp}}^{(1)}$, $F_{\text{gp}}^{(2)}$ and $F_{\text{gp}}^{(3)}$ depend on the considered languages L_1, L_2 and L_3 (defined in Section 3.4.2) but they have the following in common: They behave as the identity function with respect to group elements and map elements from $\mathbb{Z}_\text{q}$ either to G_1 or G_2 (by exponentiation with basis g_1 or g_2) depending on whether these are used as exponents of a G_1 or G_2 element in the language.

Note that all proof systems share a common reference string, i.e., there is a shared setup algorithm.

Range Proofs. For the task Wallet Issuing we need range proofs in order to show that some $\mathbb{Z}_\text{q}$-element λ'_i is "smaller" than some fixed system parameter $\mathcal{B}$ with both elements being regarded as elements from $\{0, \ldots, \text{q} - 1\}$ and the normal $\leq$-relation from the integers. We realize these range proofs using Groth–Sahai by applying the signature-based technique from Camenisch, Chaabouni, and shelat [CCs08]. Here, the verifier initially chooses parameters q and t such that every possible λ'_i can be represented as $\lambda'_i = \sum_{j=0}^{t} d_j q^j$ with $0 \leq d_j < q$. He also generates a signature on every possible value of a digit, i.e., $0, \ldots, q - 1$. The prover then shows using a Groth–Sahai NIZK that each λ'_i can be indeed represented in this way and that he knows a signature by the verifier for each of its digits. Clearly, a structure-preserving signature scheme is needed for this purpose and we use the one from Abe et al. [Abe+11].

Signature Schemes. As we need to prove statements about signatures, the signature scheme S has to be algebraic. For our construction, we use the structure-preserving signature scheme from Abe et al. [Abe+11], which is currently the most efficient structure-preserving signature scheme. Its EUF-CMA-security proof is in the generic group model, a restriction we consider reasonable with respect to our goal of constructing a highly efficient P4TC scheme. An alternative secure in the plain model would be [KPW15]. For the scheme in [Abe+11], one needs to fix two additional parameters $\mu, \nu \in \mathbb{N}_0$ defining the actual message space $G_1^\nu \times G_2^\mu$. Then $\text{sk} \in \mathbb{Z}_\text{q}^{\mu+\nu+2}$, $\text{pk} \in G_1^{\mu+2} \times G_2^\nu$ and $\sigma \in G_2^2 \times G_1$.

We use the signature scheme from Abe et al. [Abe+11] in the following ways in our system:

[26] GS-NIZKs support "pairing-product equations", "multi-scalar equations over G_1", "multi-scalar equations over G_2", and "quadratic equations over $\mathbb{Z}_\text{q}$".

- In the Wallet Issuing and Debt Accumulation tasks we use S for messages from $G_2 \times G_1$ ($\nu = 1$ and $\mu = 1$).
- In the Wallet Issuing task we use S for messages from $G_1^{2\ell+2}$ ($\nu = 2\ell + 2$ and $\mu = 0$).
- In the RSU Certification and TSP Registration tasks we use S for messages from G_1^{3+y} ($\nu = 3 + y$ and $\mu = 0$).

Commitments Schemes. We make use of two commitment schemes that are both based on the SXDH assumption. We first use the shrinking α-message-commitment scheme from Abe et al. [Abe+15a]. This commitment scheme has message space $\mathbb{Z}_q^\alpha$, commitment space G_2 and opening value space G_1. It is statistically hiding, additively homomorphic, equivocal, and F'_{gp}-Binding, for $F'_{gp}(m_1, \dots, m_\alpha) \coloneqq (g_1^{m_1}, \dots, g_1^{m_\alpha})$. We use this commitment scheme as C1 with CRS $\mathsf{crs}_{\mathsf{com}}^1$ in the following ways in our system:

- In the Wallet Issuing task we use C1 for messages from $\mathbb{Z}_q$ ($\alpha \coloneqq 1$), $\mathbb{Z}_q^2$ ($\alpha \coloneqq 2$) and $\mathbb{Z}_q^4$ ($\alpha \coloneqq 4$).
- In the Debt Accumulation task we use C1 for messages from $\mathbb{Z}_q$ ($\alpha \coloneqq 1$) and $\mathbb{Z}_q^4$ ($\alpha \coloneqq 4$).

We also use the (dual-mode) equivocal and extractable commitment scheme from Groth and Sahai [GS08]. This commitment scheme has message space G_1, commitment space G_1^2 and opening value space $\mathbb{Z}_q^2$. It is equivocal, extractable, hiding and F'_{gp}-Binding for $F'_{gp}(m) \coloneqq m$. In our system, we use this commitment scheme as C2 with CRS $\mathsf{crs}_{\mathsf{com}}^2$ in the Wallet Issuing and Debt Accumulation tasks to realize the Blum coin toss on the serial number s.

Asymmetric Encryption. To instantiate the asymmetric encryption scheme E that is used for the deposit of shares of the wallet ID, we use a variant of the structure-preserving, IND-CCA2 secure encryption scheme by Camenisch et al. [Cam+11]. Note that the original scheme is formalized for a symmetric type-1 pairing, but we need a scheme that is secure in the asymmetric type-3 case. For the conversion we followed the generic transformation proposed by Abe et al. [Abe+14] with some additional, manual optimizations.

PRF. As we want to efficiently prove statements about PRF outputs, we use an efficient algebraic construction, namely the Dodis–Yampolskiy PRF [DY04]. This function is defined by $F(k, x) : \mathbb{Z}_q^2 \to G_1$, $(k, x) \mapsto g_1^{\frac{1}{x+k}}$, where $k \overset{R}{\leftarrow} \mathbb{Z}_q$ is the random PRF key. It is secure for inputs $\{0, \dots, \lambda_{\mathrm{PRF}}\} \subset \mathbb{Z}_q$ under the λ_{PRF}-DDHI assumption. This is a family of increasingly stronger assumptions which is assumed to hold for asymmetric bilinear groups.

Secure Channels. To encrypt the exchanged protocol messages, we combine an IND-CCA2-secure asymmetric encryption scheme with an IND-CCA2-secure symmetric encryption scheme in the usual KEM/DEM approach. As asymmetric encryption scheme we use the one by Cash, Kiltz, and Shoup [CKS08]. This scheme is based on the TWIN-DH assumption and is used to set up a session key for a symmetric encryption of all protocol messages. The symmetric encryption scheme can for example be instantiated with AES in CBC mode together with HMAC based on the SHA-256 hash function. The result will be IND-CCA2-secure if AES is a pseudo-random permutation and the SHA-256 compression function is a PRF when the data input is seen as the key [Bel15].

3.4.5. Security of Π_{P4TC}

To prove the security of our toll collection protocol Π_{P4TC}, we show that it is "indistinguishable" from the ideal functionality $\mathcal{F}_{\text{P4TC}}$. In particular, we show the following theorem:

Theorem 3.1 (informal)**.** *Assuming co-CDH is hard and all building blocks are secure, we prove that the toll collection protocol Π_{P4TC} from Section 3.4 UC-realizes the ideal functionality $\mathcal{F}_{\text{P4TC}}$ from Section 3.3 in the $(\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}})$-hybrid model, i.e.,*

$$\Pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}}} \geq_{UC} \mathcal{F}_{\text{P4TC}}^{\mathcal{G}_{\text{bb}}}.$$

Informally, this means the ideal functionality and our protocol are indistinguishable and therefore provide the same guarantees regarding security and privacy. The statement holds given a static corruption of either

(1) A subset of users.

(2) All users and a subset of RSUs, TSP and SA.

(3) A subset of RSUs, TSP and SA.

(4) All RSUs, TSP and SA as well as a subset of users.

Note that we assume the DR to be honest.

While full versions of the theorem and all proofs can be found in Appendix A.1, the following proof sketch explains the ideas behind them.

Proof Outline. For our security statement, we separately prove correctness, system security and user security and privacy.

Although proofs of correctness are often neglected for smaller UC-secure protocols, they are highly nontrivial for extensive ideal models as are required for a complex real world task like toll collection. Since it is not only instructive to understand the underlying functionality but also a helpful basis for our proofs of system and user security, we briefly sketch the idea behind our proof of correctness here.

First, note that the entries $trdb = (s^{\text{prev}}, s, \phi, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ of the ideal transaction database *TRDB* define a graph structure where serial numbers s are considered vertices and predecessors s^{prev} define edges (s^{prev}, s). Assigning a label $(pid_{\mathcal{R}}, p)$ to this edge and $(\phi, \lambda, pid_{\mathcal{U}}, b)$ to the vertex s results in a graph where every vertex represents the state of a wallet and the incoming edge represents the transaction that led to this wallet state. We call this perception of *TRDB* the *Ideal Transaction Graph* and give graph-theoretic proofs of its structural properties. These properties include that the graph as a whole is a directed forest where each tree corresponds to a wallet ID λ, double-spending corresponds to branching and different wallet states have the same fraud detection ID ϕ if and only if they have the same depth in the same tree.

Secondly we add in and out commitments $(com_{\mathcal{R}}^{\text{in}}, com_{\mathcal{T}}^{\text{in}})$ and $(com_{\mathcal{R}}^{\text{out}}, com_{\mathcal{T}}^{\text{out}})$ from the real protocol to each transaction vertex in the Ideal Transaction Graph. These commitments are the fixed and updatable part of the wallet before and after the transaction (cp. Section 3.4). This information gives a second set of edges where two transactions *trdb* and *trdb** are connected if $(com_{\mathcal{R}}^{\text{out}}, com_{\mathcal{T}}^{\text{out}})$ corresponds to $(com_{\mathcal{R}}^{\text{in}*}, com_{\mathcal{T}}^{\text{in}*})$. We call the resulting graph the *Augmented Transaction Graph*. Showing that in case of an honest execution of the toll collection protocol both of those graph structures coincide with overwhelming probability yields correctness.

The proofs of system security and user security and privacy are conducted by explicitly specifying a simulator and reducing the indistinguishability of real and ideal world to the security of our building blocks. During an execution of the protocol our simulator generates the Augmented Transaction Graph as explained above. After showing that all messages sent by the simulator are statistically close to the real messages, i.e., simulated perfectly, that leaves only two kinds of reasons the environment could be able to distinguish both worlds. The first kind are *failure events* where the two graph structures in the augmented transaction graph diverge and the second are *discrepancy events* where some party's outputs could be distinguished. We show that both of those cases only occur with negligible probability by various reductions to our cryptographic building blocks and hardness assumptions.

3.5. Information Leakage and Discussion on Privacy Implications

As briefly discussed in Section 3.2.3 and Section 3.3.3, possibly known background information and the leakage of the ideal functionality determines the level of user privacy in P4TC. Since we prove our real protocol Π_{P4TC} to be indistinguishable from the ideal functionality $\mathcal{F}_{\text{P4TC}}$, it is ensured that an adversary attacking Π_{P4TC} in the real world can only learn as much about a user as an adversary in the ideal model.

Table 3.5 summarizes what an adversary learns about the users in each task. We omitted the serial number s and the fraud detection ID ϕ in the table as these are independently and uniformly drawn randomness and thus cannot be exploited (see (P9) in Section 3.3.3). In all tasks except Debt Accumulation the user's public key $\mathsf{pk}_{\mathcal{U}}$ is leaked. The variables $\mathbf{a}_{\mathcal{U}}$, $\mathbf{a}_{\mathcal{R}}$, $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ and $\mathbf{a}_{\mathcal{T}}$ refer to attributes of the participating parties. The variable p denotes the price of a Debt Accumulation transaction, and b^{bill} is the total debt the user owes at the end of the task Debt Clearance.

For every billing period, the TSP collects all transaction information from every RSU. Hence, the TSP eventually possesses two datasets:

(1) A database of users that are identified by their public key $\mathsf{pk}_{\mathcal{U}}$ together with their attributes and total debt. This dataset comprises all information from every conducted task but Debt Accumulation.

(2) A database of anonymous transactions. This dataset stems from the Debt Accumulation task.

With respect to practical privacy considerations one can naturally pose several questions: Can a single transaction be linked to a specific user? Has a user passed by a particular RSU? Can a user be mapped to a complete trip, i.e., a sequence of consecutive transactions? A final answer to these questions crucially depends on the concrete instantiation of the attributes $\mathbf{a}_{\mathcal{U}}$, $\mathbf{a}_{\mathcal{R}}$ and the pricing function but also on "environmental" parameters that cannot be chosen by the system designer such as the total number of registered users, the average length of a trip, etc. An in-depth analysis would require plausible and justifiable assumptions about probability distributions for these parameters, and would constitute a separate line of research in its own right.

In the following, however, we will discuss some general aspects of the question how to link a user to a complete trip. This problem can be depicted as a graph-theoretical problem of finding a path in a directed, layered graph. The graph consists of initial nodes, inner nodes that are ordered in layers and terminal nodes. Initial nodes represent Wallet Issuing transactions and are linked to users. Terminal nodes represent Debt Clearance transactions and are also linked to users and final balances b^{bill}.

Protocol	**Leakage**					
	$\mathsf{pk}_{\mathcal{U}}$	$\mathbf{a}_{\mathcal{U}}$	$\mathbf{a}_{\mathcal{R}}/\mathbf{a}_{\mathcal{T}}$	$\mathbf{a}_{\mathcal{R}}^{\text{prev}}$	p	b^{bill}
User Registration	●					
Wallet Issuing	●	●	●			
Debt Accumulation		●	●	●	●	
Debt Clearance	●	(●)		●		●
Prove Participation	●					

Table 3.5.: Information an adversary learns about honest users

Inner nodes represent the (anonymous) transactions in between. Assuming that transactions can only occur at discrete points in time, the inner nodes can be ordered in layers. A directed edge connects two nodes if the target node is a plausible successor of the source node. As a bare minimum, this requires that the represented transactions have equal user attributes $\mathbf{a}_{\mathcal{U}}$, the attribute $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ of the target node equals $\mathbf{a}_{\mathcal{R}}$ of the source node and the target node is in a later layer than the source node (because time can only increase). Additionally, background knowledge such as the geo-position of the RSUs, the given road infrastructure, etc. can be utilized to only insert an edge between two nodes if, e.g., the corresponding RSUs are within a certain distance bound. Obviously, the average in- and out-degree of a node heavily depends on the distribution of distinct values for $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}$. Given this graph, the task is to find a path from the initial node to the terminal node for a particular user such that the sum of the prices of the transactions on this path equals the total balance. Moreover, if the transactions of all users are taken into account, there must be a path for each user such that all paths are pairwise disjoint and every node (i.e., transaction) lies on exactly one path.

For privacy, two characteristics are important: How many solutions do exist and what is the computational complexity to find one (or all) solutions? This results in a trade-off between two borderline cases:

(1) There is exactly one unique solution. At first glance, this contradicts privacy. However, the mere existence of a unique solution is worthless, if it is computationally infeasible to find it.

(2) Finding a solution is easy but there are many equally valid solutions. In this case privacy is preserved as well.

If additional background information is omitted, the problem can be cast as a specialized instance of various NP-complete problems, e.g., the parallel-version of the KNAPSACK problem. Parallel KNAPSACK is defined as follows:

Definition 3.2 (Parallel KNAPSACK). Let $\{u_i\}_{i\in\{1,\dots,n\}}$ be a finite set of knapsacks (users) with volume $b_i \coloneqq b(u_i)$ (total balance). Further let $\{t_j\}_{j\in\{1,\dots,m\}}$ be a finite set of items (transactions) with volumes $p_j \coloneqq p(t_j)$ (price) respectively. Let $x_{ij} \in \{0,1\}$ be variables that indicate if knapsack i contains item j. The task is to maximize the objective function

$$\sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij} v(t_j)$$

under the constraints

$$\sum_{j=1}^{m} x_{ij} p_j \leq b_i \text{ for } i \in \{1,\dots,m\} \tag{3.4}$$

and

$$\sum_{i=1}^{n} x_{ij} \leq 1 \text{ for } j \in \{1, \dots, m\}$$

Here, v denotes a value function that assigns a benefit to each item. In our case, we set $v(t_j) \coloneqq p(t_j)$, i.e., each item (transaction) has its volume (price) as value. Informally, parallel KNAPSACK means to pack items (transactions) into knapsacks (invoices) without exceeding the volume (balance) of each knapsack or assigning an item to more than one knapsack, such that the total value of the packed items is maximal.

Having an optimal solution means that each knapsack is filled to its limit, i.e., the total value of its items equals the total balance of a user. This is because we know that all knapsacks can be filled exactly without any item being left over, since every transaction must belong to a user. Hence we can deduce that any solution that does not completely use a knapsack up to its limit must omit an item and can thus never be optimal.

Setting the value function to the price of the transaction has the advantage that now every solution for filling each knapsack to its limit leads to a maximal total value, i.e., the total value of the items in a knapsack equals the total balance of a user. Hence, the set of optimal solutions is now the set off all solutions for the trip-finding problem.

Two remarks are in order. With the above explanation, we map an instance of our trip-finding problem onto an instance of the parallel KNAPSACK problem, i.e., this only shows that our problem at hand is not harder than parallel KNAPSACK. To be fully correct, we would have to show the inverse direction, namely that a *restricted set* of instances of the parallel KNAPSACK problem can be cast into our trip-finding problem (without background knowledge). The *general* KNAPSACK problem is NP-complete. This is beneficial as it implies that finding a solution is generally believed to be intractable. However, there might be good heuristics for all “natural” instances. Especially since we only need to consider those instances for which the optimal solution is a perfect solution (i.e., each knapsack is completely filled). Nonetheless, this restricted class of KNAPSACK problems is still NP-complete and can indeed be mapped onto our trip-finding problem. Moreover, depending on the concrete parameters (e.g., an upper bound on the maximum price p or the balance b^{bill}) the problem might become fixed-parameter tractable [GRR19]. In other words, although solving the general problem has super-polynomial runtime in the instance size, it might still be practically solvable for “real-world instances”. We stress again that an in-depth analysis requires to look at concrete distributions of these parameters which may be the basis for an independent work.

Nonetheless, there are indicators that—if finding one solution is easy—there might be a myriad of solutions, which again yields privacy. To examine this, we take the German Toll Collect[27] for trucks and analyze the statistics from January, 2018 [Deu18], as a more concrete example. This system uses 24 distinct[28] values for the user attributes. We assume that transaction times are recorded with 5 min resolution which is a little bit larger than the timespan a truck needs to travel between two subsequent RSUs.[29] Within this timeslot each RSU is passed by 28.4 other trucks. Picking a fixed RSU and a fixed

[27] https://www.toll-collect.de/en/

[28] The actual system uses more attribute values (i.e., 720) for statistical purposes. However, we only counted attribute values that the price depends on.

[29] Since the German toll collection system is GNSS-based and has no RSUs, the average distance between two RSUs is assumed to be 5 km as in the Austrian Toll Collection system [KDD15]. Additionally, we assumed 80 $\frac{\text{km}}{\text{h}}$ travel speed for trucks.

timeslot, this implies that the probability *pr* that at least one other truck with identical attributes passes by equals

$$pr = 1 - \left(\frac{23}{24}\right)^{28.4} \approx 0.7$$

This means, for a randomly chosen transaction in the graph the probability to find at least one other transaction in the same layer (i.e., timeslot) for the same RSU and for identical user attributes equals 0.7. In other words, for a randomly chosen node in the transaction graph the in- and out-degree is at least two with probability 0.7.

The average length of trips per month of a single vehicle equals 3 398 km and the vehicle passes 613 RSUs. This means the average trip consists of 613 transactions. For approximately $0.7 \times 613 = 429$ of these transactions there are at least two identically looking transactions that are a plausible predecessor or successor. However, it is completely wrong to assume this would yield 2^{429} identically looking paths that could be assigned to a particular user. This kind of analysis would assume an independence of the transactions which does not hold. Again, let us randomly pick a particular RSU and timeslot and consider the geographically next RSU in the following timeslot. If two identical transactions are observed at the first RSU, the *conditional probability* to observe two identical transactions at the next RSU again is assumedly much higher than 0.7, given that there are no entry points, exit points or intersections in between. Vice versa, if an transaction is unique at the first RSU, the *conditional probability* to observe a second, identical transaction at the next RSU is assumedly much lower. On the contrary, we expect to have some transaction nodes with a high in-/out-degree, namely those before an exit point or after an entry point. If a truck leaves the tolling system, it might either promptly re-enter the system at some nearby entry point or at some distant entry point in the remote future.

In summary, the analysis above is overly simplified as we assume uniform distributions of all values and an independence of random variables that is likely not given in reality. Nonetheless, this indicates that the solution space for mapping a particular user to a specific trip might be vast for long average trip lengths and it points out why an in-depth analysis would justify an independent work on its own. Please also note that this analysis is only based on trucks, not passenger cars, where we expect much higher probabilities for joint occurrence of identical attributes due to the higher number of participants and higher travel speeds.

Remark 3.3. In practice, several privacy notions like k-anonymity are established. For several reasons these notions are not directly applicable here. First of all, these notions evaluate the privacy level of a concrete dataset and we stress again that this is out of the scope of this work. While at first glance the calculations above might suggest that our system features k-anonymity [Swe02] for some yet to be determined k, the notion of k-anonymity is actually not applicable due to formal reasons. The definition of k-anonymity requires the database to have exactly one entry for each individual, but our transaction database features several entries per user. Therefore, the notion of k-anonymity is syntactically not applicable to the users of our system. While we could still discuss k-anonymity in this setting if the TSP combined all entries that pertain to the same user into one single entry, privacy of our system largely stems from the TSP not being able to link transactions of the same user in this way and hence such a discussion would largely undervalue the privacy protection P4TC provides.

3.6. On the Impact of Different Pricing Schemes on Privacy

In Section 3.5 we discussed the potential difficulty of mapping a specific transaction or a specific trip, i.e., a set of specific transactions, to a user. Now we want to look at this from a different angle, and first ask how easy it is to find out which RSUs a user has passed (independent of specific transactions). In [Jol+24] this question was answered for a specific ETC system, namely the ETC system currently deployed in Brisbane, Australia. There, the authors show that for users with a wallet balance of 10$ or less, it is possible to guess the toll stations visited by a user (but not necessarily the correct order) with an average probability of 94%.

In this section we ask whether this is an inherent problem with ETC systems or whether it depends on the *pricing scheme* used. To answer this question, we will first examine and compare several artificial pricing schemes. To get a more complete picture, we will then compare the artificial pricing schemes with the real pricing scheme from the Brisbane ETC system.

3.6.1. Analysis of Artificial Pricing Schemes and General Results

The artificial pricing schemes we will be examining are the following:

$$\begin{aligned}
\mathcal{P}_{\text{uniform}} &\coloneqq \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, \dots\}\\
\mathcal{P}_{\text{linear\_0.1}} &\coloneqq \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, \dots\}\\
\mathcal{P}_{\text{linear\_0.5}} &\coloneqq \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, \dots\}\\
\mathcal{P}_{\text{linear\_1.0}} &\coloneqq \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots\}\\
\mathcal{P}_{\text{quad\_0.1}} &\coloneqq \{1, 1.21, 1.44, 1.69, 1.96, 2.25, 2.56, 2.89, 3.24, 3.61, \dots\}\\
\mathcal{P}_{\text{quad\_0.5}} &\coloneqq \{1, 2.25, 4, 6.25, 9, 12.25, 16, 20.25, 25, 30.25, \dots\}
\end{aligned}$$

$\mathcal{P}_{\text{uniform}}$ simply has the same price for all toll stations; the $\mathcal{P}_{\text{linear\_}x}$ pricing schemes have a linear increase of prices (such that the difference between two consecutive prices is x); and the $\mathcal{P}_{\text{quad\_}x}$ pricing schemes have a quadratic increase (more specifically, they take the items from $\mathcal{P}_{\text{linear\_}x}$ and square them). Note that we will not examine $\mathcal{P}_{\text{quad\_1.0}}$ because the prices there quickly become too high to be useful.

For this set of pricing schemes, we want to explore the following question:

> *Given a pricing scheme and a wallet balance, how many different trips can lead to that wallet balance?*

Recall that a trip consists of a sequence of toll station visits. So we are interested not only in the toll stations visited (as in [Jol+24]), but also in the order in which they are visited. This is similar to the *subset sum problem*, which is a special case of the knapsack problem.

Definition 3.4 (Subset Sum Problem (SSP))**.** Given a set $\mathcal{I}$ of integers and a sum S, *is* there a subset of $\mathcal{I}$ whose sum is exactly S?

In our scenario the set $\mathcal{I}$ corresponds to a set of prices and the sum S is the wallet balance. However, answering the question above does not exactly map to the SSP, since we need the number of solutions a SSP instance has. But it does map exactly to the counting version of the SSP.

Definition 3.5 (Counting Version of the Subset Sum Problem (#SSP))**.** Given a set $\mathcal{I}$ of integers and a sum S, *how many* subsets of $\mathcal{I}$ exist whose sum is exactly S?

Note that #SSP is #P-complete and thus NP-hard [Val79]. However, the instances we are examining in this section are small enough to be solvable in reasonable time.

Translating this to the ETC setting again, solving the #SSP means finding out how many different trips can lead to the same wallet balance (sum). Intuitively, a large number of different trips yields more privacy for the user since then it is harder to find the correct trip. Now that we have a clearer picture of the problem domain, we can rephrase our question from above:

What influence do the different pricing schemes have on the #SSP?

To get closer to answering this question, we will first look at how the *number of toll stations* affects the #SSP. In Fig. 3.8 we compare the pricing schemes $\mathcal{P}_{\text{uniform}}$, $\mathcal{P}_{\text{linear\_0.1}}$, and $\mathcal{P}_{\text{quad\_0.1}}$ with 5 and 9 toll stations, respectively. To get from the pricing schemes from above to a price list for 5 (or 9) toll stations, we just take the first 5 (or 9) items from the list. In each graph in Fig. 3.8, for each possible wallet balance between 1 and 10, the output of the #SSP is plotted for that wallet balance (as sum S) and price list (as set $\mathcal{I}$). In other words, for each wallet balance, the number of distinct trips leading to that wallet balance is depicted. In the upper left corner the average number of different trips is also shown. If we now compare the 5 toll station variant with the 9 toll station variant for each pricing scheme, we can see that the number of trips is generally higher for the 9 toll station case. This fits with the intuition that the larger the set $\mathcal{I}$, the more solutions the SSP has. Thus, having more toll stations seems to benefit user privacy.

One can already see in Fig. 3.8 that $\mathcal{P}_{\text{uniform}}$ outperforms the other pricing schemes by far. It can also be seen that even for the currently weakest-looking pricing model ($\mathcal{P}_{\text{quad\_0.1}}$), the number of trips with a wallet balance of $\approx$10 is mostly so high that it is unlikely to filter out the correct trip among them. This leads to the conclusion that high wallet balances are beneficial for privacy. In the following, we will therefore look at a smaller maximum wallet balance in order to better analyze the vulnerable small wallet balances. Similarly, we will now examine the pricing schemes for 5 toll stations, as this also seems to be more privacy-critical.

In Fig. 3.9 we now compare all six pricing schemes ($\mathcal{P}_{\text{uniform}}$, $\mathcal{P}_{\text{linear\_0.1}}$, $\mathcal{P}_{\text{linear\_0.5}}$, $\mathcal{P}_{\text{linear\_1.0}}$, $\mathcal{P}_{\text{quad\_0.1}}$, and $\mathcal{P}_{\text{quad\_0.5}}$), all with 5 toll stations and wallet balances ranging from 1 to 6. When using the average number of possible trips as a metric, then the pricing schemes can be clearly sorted from best to worst as follows:

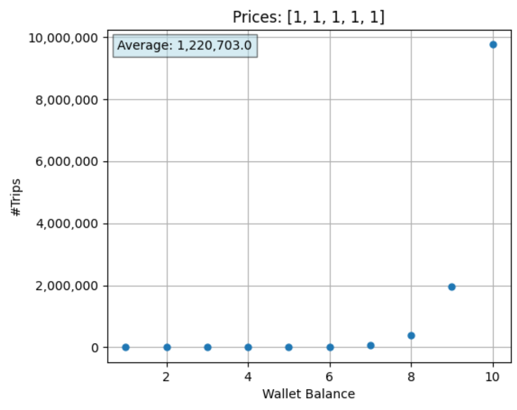
$$\mathcal{P}_{\text{uniform}} \gg \mathcal{P}_{\text{linear\_0.1}} > \mathcal{P}_{\text{linear\_0.5}} > \mathcal{P}_{\text{linear\_1.0}} > \mathcal{P}_{\text{quad\_0.1}} > \mathcal{P}_{\text{quad\_0.5}}$$

When using the number of trips for wallet balance 6 as a metric, then the pricing schemes are sorted a bit differently:

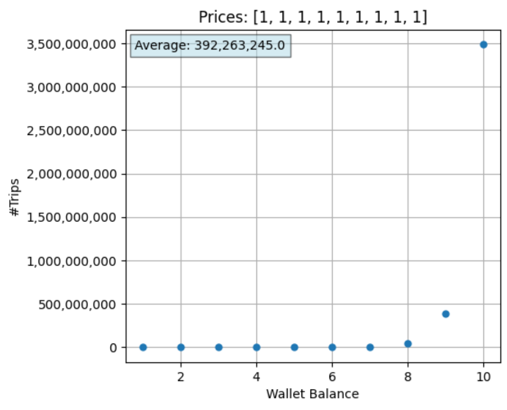
$$\mathcal{P}_{\text{uniform}} \gg \mathcal{P}_{\text{linear\_0.1}} > \mathcal{P}_{\text{linear\_0.5}} > \mathcal{P}_{\text{quad\_0.1}} > \mathcal{P}_{\text{linear\_1.0}} > \mathcal{P}_{\text{quad\_0.5}}$$

For both metrics, $\mathcal{P}_{\text{uniform}}$ is undoubtedly the best scheme from a privacy perspective. However, it is unsuitable for widespread practical use because it provides virtually no flexibility in the pricing scheme. Most toll providers will want to be able to assign different prices to different toll stations, e.g., to support traffic management. However, it is easy to see that having multiple toll stations with the same price has a positive impact on privacy, as it is then impossible to deduce the toll station visited from the price alone. Toll providers must therefore find the right balance between pricing flexibility and user privacy when deciding on the number of different prices to have in their system.

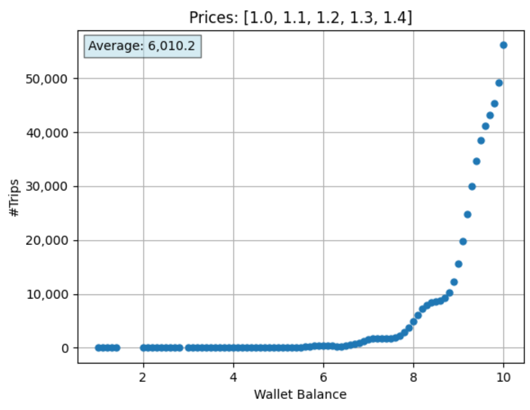
On average, the linear pricing schemes generally seem to be better than the quadratic ones from a privacy perspective. Since the linear schemes have more "even" prices, there are intuitively more



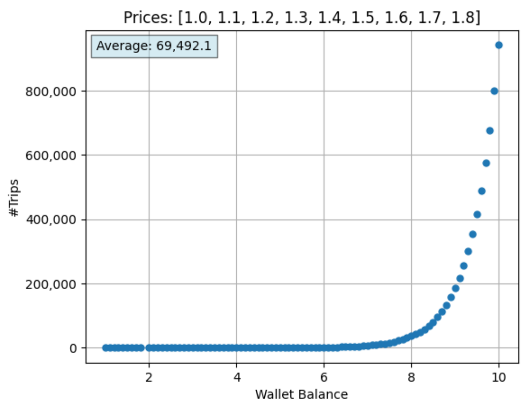
(a) $\mathcal{P}_{\text{uniform}}$ with length 5



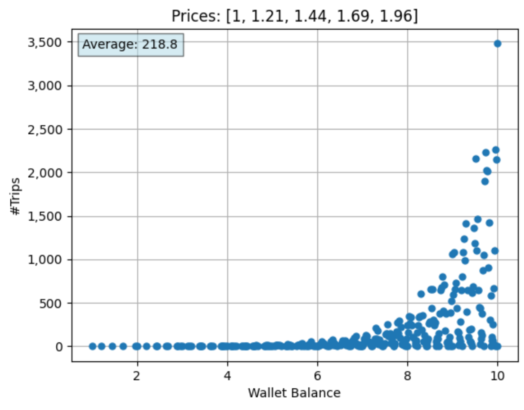
(b) $\mathcal{P}_{\text{uniform}}$ with length 9



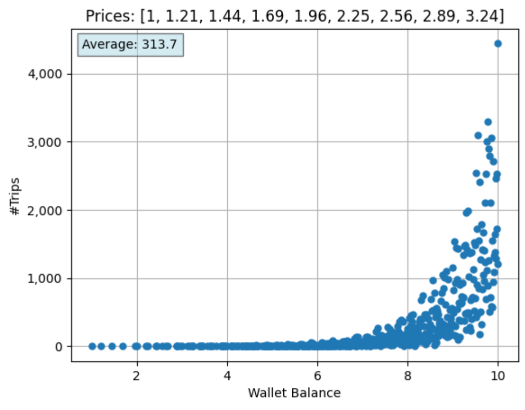
(c) $\mathcal{P}_{\text{linear\_0.1}}$ with length 5



(d) $\mathcal{P}_{\text{linear\_0.1}}$ with length 9

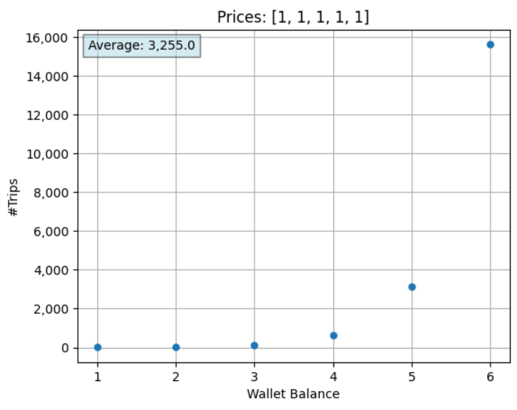


(e) $\mathcal{P}_{\text{quad\_0.1}}$ with length 5

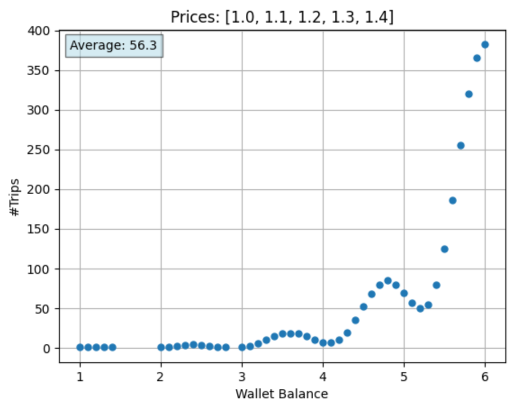


(f) $\mathcal{P}_{\text{quad\_0.1}}$ with length 9

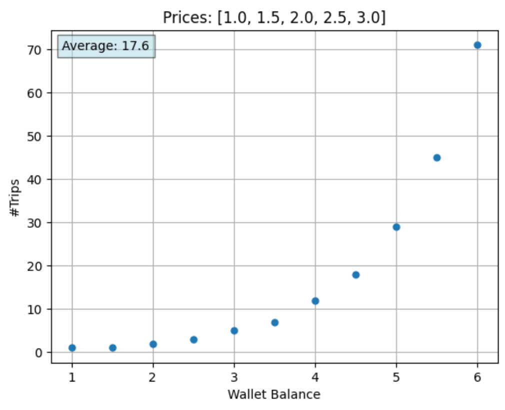
Figure 3.8.: Comparison between price lists of length 5 and length 9



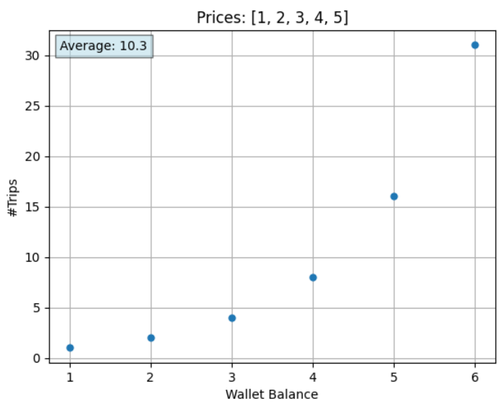
(a) $\mathcal{P}_{\text{uniform}}$



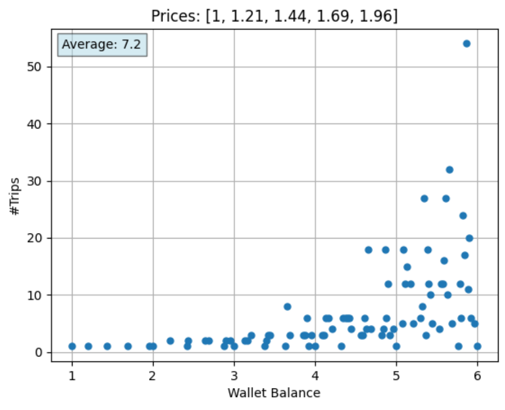
(b) $\mathcal{P}_{\text{linear\_0.1}}$



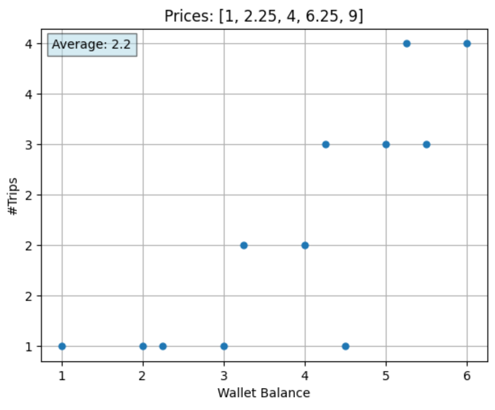
(c) $\mathcal{P}_{\text{linear\_0.5}}$



(d) $\mathcal{P}_{\text{linear\_1.0}}$



(e) $\mathcal{P}_{\text{quad\_0.1}}$



(f) $\mathcal{P}_{\text{quad\_0.5}}$

Figure 3.9.: Comparison between different artificial pricing schemes (wallet balance ≤ 6)

possible trips per wallet balance. In the case of quadratic pricing schemes (particularly evident in Fig. 3.9e), the number of possible trips can be in the low single digits even with high wallet balances. This is probably due to the irregular distribution of prices.

For $\mathcal{P}_{\text{linear\_0.5}}$ (Fig. 3.9c) and $\mathcal{P}_{\text{linear\_1.0}}$ (Fig. 3.9d) one can see that the number of possible trips is strictly monotonically increasing. This is not the case with $\mathcal{P}_{\text{linear\_0.1}}$, because prices 1.5 to 1.9 are missing and therefore some possible combinations for certain sums are not available. When comparing Fig. 3.8c and Fig. 3.8d, it can be seen that the number of possible trips for $\mathcal{P}_{\text{linear\_0.1}}$ approaches a monotonically increasing function as the number of toll stations increases.

Looking at the linear pricing schemes in Fig. 3.9 in detail, one can see that they are all above $\approx$30 possible trips from a fairly low wallet balance: With $\mathcal{P}_{\text{linear\_0.1}}$ the number of possible trips is 35 for a wallet balance of 4.4 and stays above this value from then on, with $\mathcal{P}_{\text{linear\_0.5}}$ the number of possible trips is 29 for a wallet balance of 5.0 and then increases monotonically, and with $\mathcal{P}_{\text{linear\_1.0}}$ we reach 31 possible trips with a wallet balance of 6.0. Just to compare, with $\mathcal{P}_{\text{uniform}}$ we have 25 possible trips with a wallet balance of 2, and with a wallet balance of 3 we already have 125 possible trips.

If we compare the quadratic pricing schemes with the linear pricing schemes again, we can see that they differ greatly in the number of possible wallet balances: The linear pricing schemes generally have quite a low number of possible wallet balances ($\mathcal{P}_{\text{linear\_0.1}}$ has 45 possible wallet balances and $\mathcal{P}_{\text{linear\_0.5}}$ has 11 possible wallet balances). On the other hand, $\mathcal{P}_{\text{quad\_0.1}}$ has 89 possible wallet balances, which is a comparatively large number.[30] It can be argued that pricing schemes with a small number of possible wallet balances have a positive effect on privacy. Intuitively, this is because when there are few possible wallet balances, there are generally more users with the same wallet balance. This means that per trip there are more users who could potentially have made that trip, making it even more difficult to associate a trip with a user. This is another reason why linear pricing seems to outperform quadratic pricing in terms of user privacy.

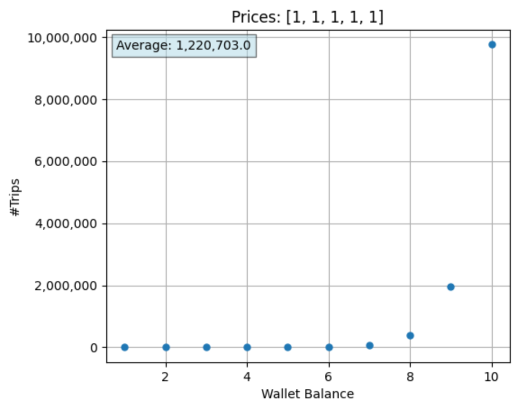
In Fig. 3.10, the same pricing schemes are shown as in Fig. 3.9, but now with a maximum wallet balance of 10. Since the trends observed in Fig. 3.9 continue in Fig. 3.10, we will not discuss Fig. 3.10 in detail and will simply provide Fig. 3.10 as additional information for the reader.

Summarizing all the points described above, we conclude that among the artificial pricing schemes analyzed, it is most beneficial for user privacy to choose a linear pricing scheme with a small distance between the different prices. This way, the number of possible wallet balances is small and it is not the case that there is only a low number of possible trips for higher wallet balances. On the other hand, the toll stations have different prices, so the toll provider has some flexibility in pricing. Ideally, some toll stations are assigned the same price.

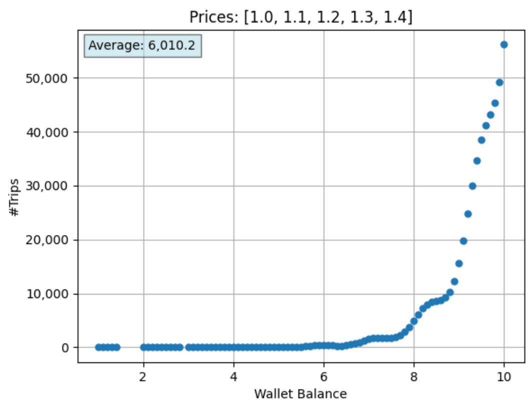
Remark 3.6. Of course, there are many more pricing schemes that could be analyzed. However, we have deliberately looked at only a limited selection here, as we only wanted to illustrate that the chosen pricing scheme has a significant impact on privacy.

Remark 3.7. As a side result it can be noted that pricing schemes that form a *superincreasing sequence* are not suitable pricing schemes.

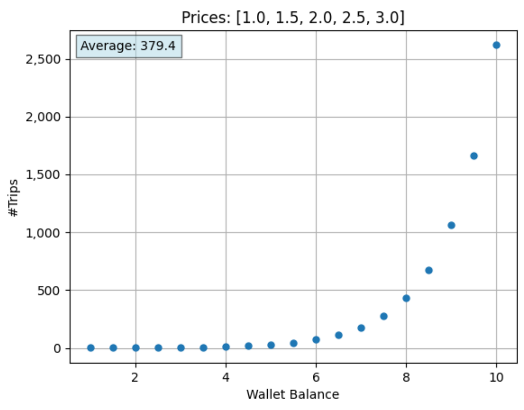
[30] Note that we are omitting $\mathcal{P}_{\text{linear\_1.0}}$ (6 possible wallet balances) and $\mathcal{P}_{\text{quad\_0.5}}$ (12 possible wallet balances) from this comparison, as there simply cannot be many wallet balances $\leq$6 due to the generally higher individual prices.



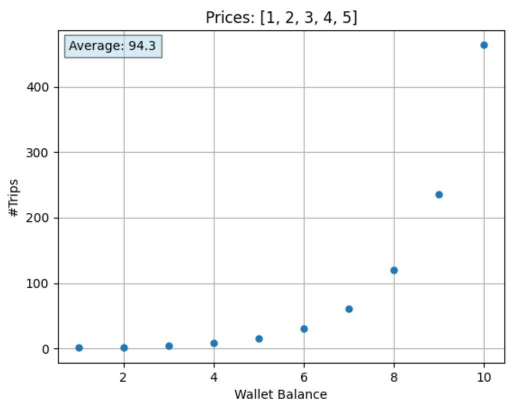
(a) $\mathcal{P}_{\text{uniform}}$



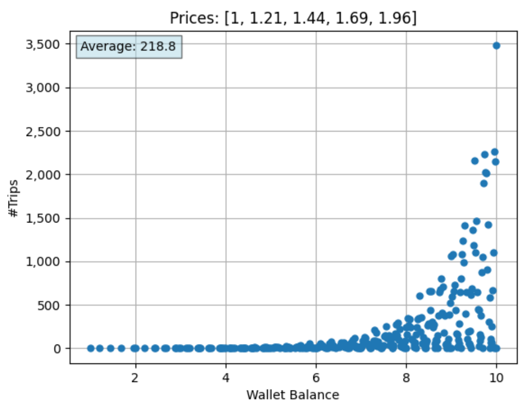
(b) $\mathcal{P}_{\text{linear\_0.1}}$



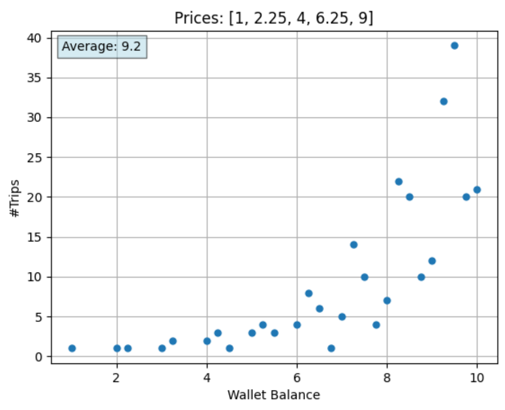
(c) $\mathcal{P}_{\text{linear\_0.5}}$



(d) $\mathcal{P}_{\text{linear\_1.0}}$



(e) $\mathcal{P}_{\text{quad\_0.1}}$



(f) $\mathcal{P}_{\text{quad\_0.5}}$

Figure 3.10.: Comparison between different artificial pricing schemes (wallet balance ≤10)

Definition 3.8 (Superincreasing Sequence (from [MV97, Def. 8.34])). A *superincreasing sequence* is a sequence $(b_1, b_2, \ldots, b_n)$ of positive integers with the property that $b_i > \sum_{j=1}^{i-1} b_j$ for each i with $2 \leq i \leq n$.

An example for a superincreasing pricing scheme would be $\mathcal{P}_{2^i} \coloneqq \{1, 2, 4, 8, 16, \ldots\}$. The subset sum problem for superincreasing sequences is solvable in polynomial (even in linear) time [MV97, Sec. 8.6.1], so finding all possible solutions is efficiently possible even for large instances. Also, superincreasing sequences tend to have a low number of possible solutions to the subset sum problem, which makes them unsuitable as a pricing scheme.

3.6.2. Analysis of the Brisbane ETC System

Having examined various artificial pricing schemes, we now turn to a real-world example: the Brisbane ETC system, which was also examined in [Jol+24]. The Brisbane ETC system is an access-based ETC system with 9 toll plazas. Like [Jol+24], we use the toll prices for passenger cars as of July 1, 2018, which are as follows [TC18, p. 23]:

$$\mathcal{P}_{\text{brisbane}} \coloneqq \{1.72, 2.68, 2.84, 3.19, 4.09, 4.55, 5.11, 5.11, 5.46\}$$

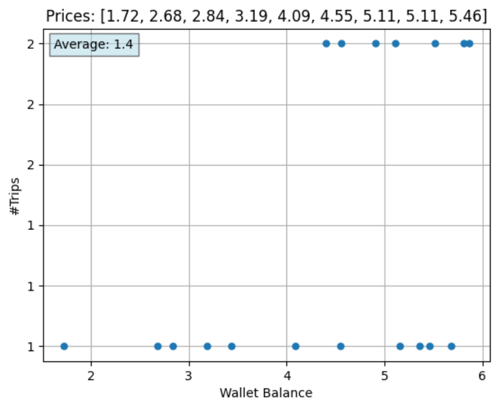
In Fig. 3.11 we show the distribution of the number of possible trips per wallet balance for different maximum wallet balances. Looking at the maximum wallet balance of 6, $\mathcal{P}_{\text{brisbane}}$ (cp. Fig. 3.11a) seems to perform even worse than the worst artificial pricing scheme examined, i.e., $\mathcal{P}_{\text{quad\_0.5}}$ (cp. Fig. 3.9f), as all wallet balances have only 1 or 2 possible trips. All graphs in Fig. 3.11 show that $\mathcal{P}_{\text{brisbane}}$ has the property that even for higher wallet balances, some wallet balances have only one possible trip. Since the prices in $\mathcal{P}_{\text{brisbane}}$ are very unevenly distributed, this is not surprising.

The linear pricing schemes in Fig. 3.9 all had the property of reaching ≥30 possible trips per wallet balance fairly quickly (the slowest of them was $\mathcal{P}_{\text{linear\_1.0}}$, which reached it at a wallet balance of 6.0). For the Brisbane ETC system, 30 possible trips is only reached in Fig. 3.11d. There, only 7 out of 174 wallet balances have ≥30 possible trips, and 11.33 is the smallest wallet balance with this property.

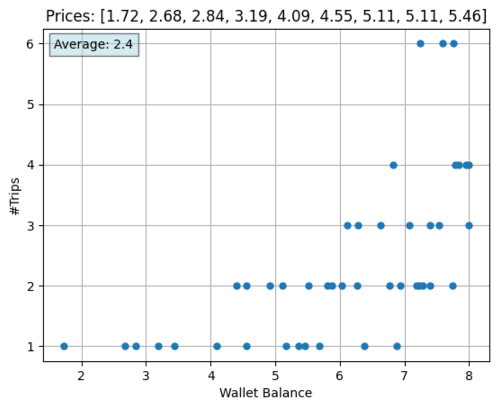
Thus, the Brisbane ETC system appears to perform worse than any of the artificial pricing schemes examined so far. We now turn to the question of whether we can "fix" this by slightly tweaking the pricing scheme. For that we examine two additional pricing schemes: $\mathcal{P}_{\text{brisbane\_0.1}}$, which rounds the prices from the Brisbane ETC system to the nearest 10 cents; and $\mathcal{P}_{\text{brisbane\_0.5}}$, where the prices from the Brisbane ETC system are rounded to the nearest 50 cents. More precisely, these pricing schemes look like this:

$$\mathcal{P}_{\text{brisbane\_0.1}} \coloneqq \{1.70, 2.70, 2.80, 3.20, 4.10, 4.60, 5.10, 5.10, 5.50\}$$
$$\mathcal{P}_{\text{brisbane\_0.5}} \coloneqq \{1.50, 2.50, 3.00, 3.00, 4.00, 4.50, 5.00, 5.00, 5.50\}$$

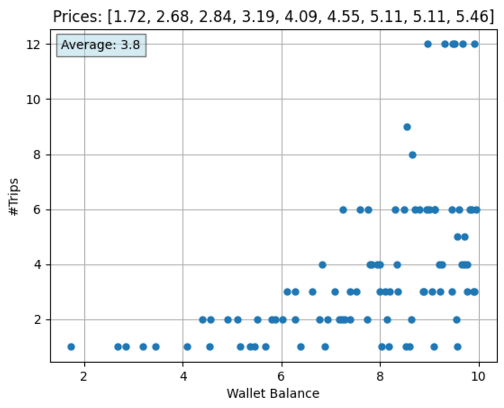
In Fig. 3.12 we now compare $\mathcal{P}_{\text{brisbane}}$ with $\mathcal{P}_{\text{brisbane\_0.1}}$ and $\mathcal{P}_{\text{brisbane\_0.5}}$, for maximum wallet balances of 6 and 10. It can be seen that with a maximum wallet balance of 6, all pricing schemes do not perform particularly well, but since the minimum price is ≥1.50, it is expected that not many different trips are possible with a wallet balance ≤6. Looking at the graphs on the right, where the maximum wallet balance is 10, we can see that $\mathcal{P}_{\text{brisbane}}$ and $\mathcal{P}_{\text{brisbane\_0.1}}$ sometimes still have a low number of possible trips even with higher wallet balances. This is not the case for $\mathcal{P}_{\text{brisbane\_0.5}}$, where the number of possible trips is 30 or more for all wallet balances ≥7.5. The number of possible wallet balances for $\mathcal{P}_{\text{brisbane\_0.5}}$ (with 16) is also significantly lower than for $\mathcal{P}_{\text{brisbane\_0.1}}$ (54) and for $\mathcal{P}_{\text{brisbane}}$ (93). Thus, $\mathcal{P}_{\text{brisbane\_0.5}}$ seems to be a good improvement of $\mathcal{P}_{\text{brisbane}}$ so far.



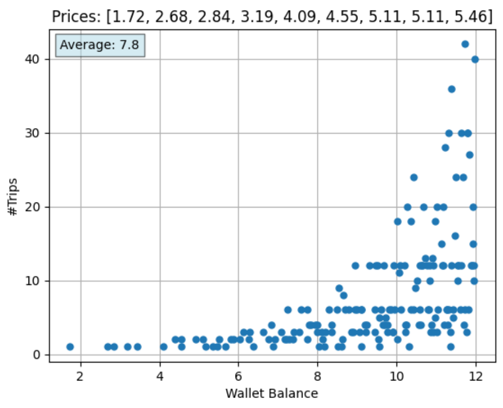
(a) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤6



(b) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤8



(c) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤10



(d) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤12

Figure 3.11.: The pricing scheme of Brisbane's ETC system for different maximum wallet balances

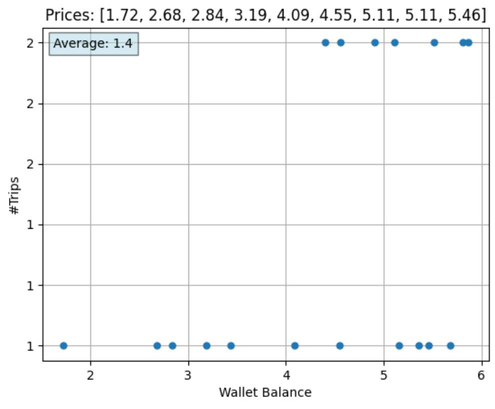
Note that $\mathcal{P}_{\text{brisbane\_0.5}}$ has the property that some of its prices can be expressed as the sum of other prices, while $\mathcal{P}_{\text{brisbane\_0.1}}$ does not. Since this property inherently increases the number of possible trips for a given wallet balance, having this property for a pricing scheme seems to be beneficial for privacy.

If we now compare $\mathcal{P}_{\text{brisbane\_0.5}}$ with the pricing schemes in Fig. 3.10, we see that $\mathcal{P}_{\text{brisbane\_0.5}}$ fits into the ranking of artificial pricing schemes as follows:

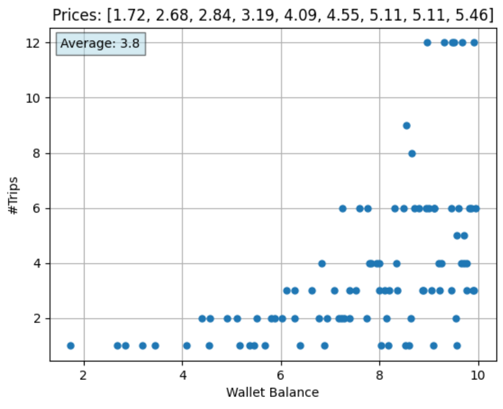
$$\mathcal{P}_{\text{uniform}} \gg \mathcal{P}_{\text{linear\_0.1}} > \mathcal{P}_{\text{linear\_0.5}} > \mathcal{P}_{\text{quad\_0.1}}/\mathcal{P}_{\text{linear\_1.0}} > \mathcal{P}_{\text{brisbane\_0.5}} > \mathcal{P}_{\text{quad\_0.5}}$$

Note that $\mathcal{P}_{\text{brisbane\_0.1}}$ and $\mathcal{P}_{\text{brisbane}}$ are both worse than $\mathcal{P}_{\text{quad\_0.5}}$.

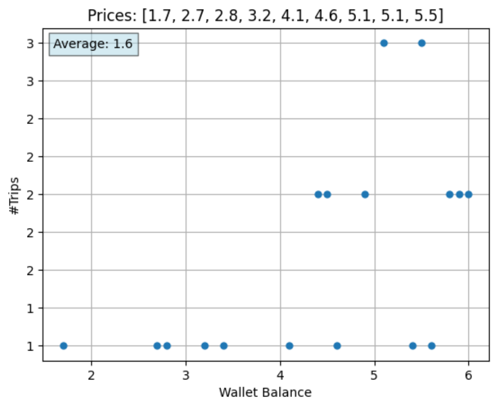
In conclusion, while $\mathcal{P}_{\text{brisbane\_0.5}}$ is a significant improvement over the original $\mathcal{P}_{\text{brisbane}}$, it does not compare well with most artificial pricing schemes. Although $\mathcal{P}_{\text{brisbane\_0.5}}$ is basically $\mathcal{P}_{\text{linear\_0.5}}$ with a few items missing, those missing items seem to have a noticeable impact on privacy. Nevertheless, switching from $\mathcal{P}_{\text{brisbane}}$ to $\mathcal{P}_{\text{brisbane\_0.5}}$ would already significantly improve privacy for most users, since for wallet balances ≥7.5 the number of possible trips is ≥30. And if switching to a pricing scheme with a difference of ≥50 cents between prices is not flexible enough for practical use (for example, because small annual price changes should be possible), then switching from $\mathcal{P}_{\text{brisbane}}$ to $\mathcal{P}_{\text{brisbane\_0.1}}$ would still be a noticeable improvement.



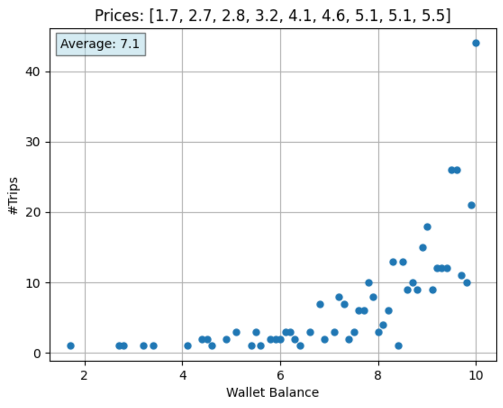
(a) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤6



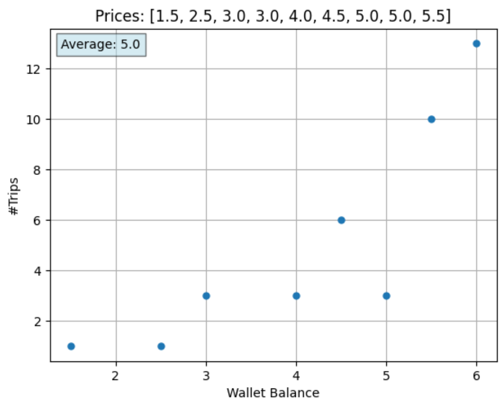
(b) $\mathcal{P}_{\text{brisbane}}$ with wallet balances ≤10



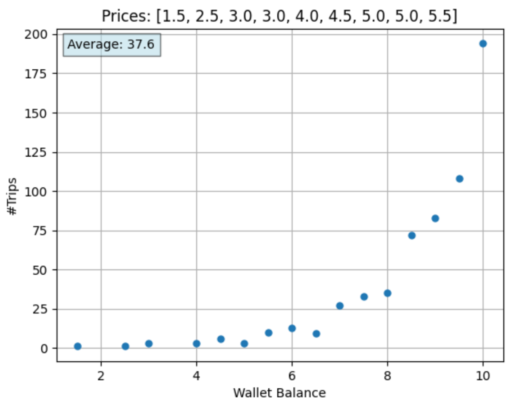
(c) $\mathcal{P}_{\text{brisbane\_0.1}}$ with wallet balances ≤6



(d) $\mathcal{P}_{\text{brisbane\_0.1}}$ with wallet balances ≤10



(e) $\mathcal{P}_{\text{brisbane\_0.5}}$ with wallet balances ≤6



(f) $\mathcal{P}_{\text{brisbane\_0.5}}$ with wallet balances ≤10

Figure 3.12.: Comparing the pricing scheme of Brisbane's ETC system with "similar" pricing schemes

4. Universally Composable Auditable Surveillance

This chapter is based on [Fet+23a; Fet+23b] and most parts of this chapter are either taken verbatim or rephrased from these works, along with some additional explanations. As the conference version [Fet+23b] had a page limit, significant details are missing in that version (in particular the ideal functionalities and protocols could only be presented there in abbreviated form), which have been added to the appendix of the ePrint version [Fet+23a]. In this thesis, the contents of [Fet+23a; Fet+23b] are now reorganized to form a consistent overall structure. The formatting of the ideal functionality and protocol has also been adjusted to make the formatting consistent within the thesis.

In [Fet+23a; Fet+23b], but not part of this thesis, an application was presented to showcase how the auditable surveillance functionality $\mathcal{F}_{\mathrm{AS}}$ can be used to enhance a system with auditable surveillance capabilities. We briefly sketch the contribution of this application here to give an idea about the intended use of $\mathcal{F}_{\mathrm{AS}}$, but the application itself will not be featured further in this thesis. An additional contribution of [Fet+23a; Fet+23b] is a (toy) ideal functionality, $\mathcal{F}_{\mathrm{ASTE}}$, for Auditably Sender-Traceable Encryption (ASTE). Effectively, $\mathcal{F}_{\mathrm{ASTE}}$ allows a registered user to *anonymously* encrypt a confidential message to another registered user, while ensuring that law enforcement is able to deanonymize the resulting ciphertext. In this toy example, law enforcement does not learn the message of a ciphertext, fully separating "content" from "identity". But it is easy to modify the example, so that law enforcement learns any function of message and identity. In [Fet+23a; Fet+23b], $\mathcal{F}_{\mathrm{ASTE}}$ also has a protocol that UC-realizes it utilizing $\mathcal{F}_{\mathrm{AS}}$.

Structure of this Chapter. We first give an introduction to the scenario in Section 4.1, clearly state the contribution, and discuss related work. In Section 4.2 we introduce the setting, give a high-level system overview, and discuss the desired security properties of our system.

The concrete presentation of the auditable surveillance system starts in Section 4.3 with the ideal functionality for auditable surveillance, $\mathcal{F}_{\mathrm{AS}}$. In Section 4.4 we present the protocol Π_{AS} that UC-realizes $\mathcal{F}_{\mathrm{AS}}$. Π_{AS} outsources the decryption of user secrets to a hybrid functionality $\mathcal{F}_{\mathrm{AD}}$, which is presented in Section 4.5. In Section 4.6 we give a protocol Π_{AD} that UC-realizes the auditable decryption functionality $\mathcal{F}_{\mathrm{AD}}$ in the YOSO model.

Finally, we discuss some design decisions of our system in Section 4.7 and conclude the chapter in Section 4.8. Note that the security proofs are deferred to Appendix B.

4.1. Introduction

In user-centric application scenarios such as communication services, electronic payments, internet search engines, etc. there is a strong tension between the need for user privacy and legal requirements or business interests that entail the monitoring of a user's (meta-)data. This tension is also reflected by the recent European Council Resolution on "Security through encryption and security despite encryption" [Cou20]. On the one hand, there is a strong demand for anonymity and confidentiality supported by the European General Data Protection Regulation. On the other hand, scenario-specific laws and regulations such as the European Council Resolution on the Lawful Interception of Telecommunications [Off95] or the EU Directive on Anti-Money Laundering and Countering the Financing of Terrorism (AML/CFT) [Off18], to name just a few, make it necessary to revoke a user's anonymity or disclose its encrypted transaction data or messages under certain well-defined circumstances, e.g., when a warrant has been issued for a suspect.

The security research community has recognized and addressed the necessity to balance confidentiality/anonymity with accountability. Most proposed solutions follow a variant of the key escrow paradigm [Arf+21; BCS05; BGK95; Daz+22; KL95; LRC14; PY00; Sav18; YY98]. In key escrow systems one or more, typically fixed, trusted authorities (TAs) aka escrow agents, are equipped with (shares of) a trapdoor key which can be used to recover encrypted messages, revoke transaction anonymity, etc. However, this holds the risk that, by corrupting the publicly known TAs, trapdoors can be *silently* misused, e.g., for mass surveillance or spying on lawful individuals of public interest (e.g., politicians, business leaders, celebrities). Due to these issues, policymakers, security researchers, and practitioners are concerned about deploying key escrow without further measures to prevent or detect misuse, e.g., [Abe+15b; Gro19]. Moreover, the lack of transparency concerning the lawful usage of trapdoors leaves citizens with the subconscious feeling of being under permanent surveillance.

To make surveillance actions more transparent and accountable, recent work [Fra+18; GP17; Pan+19] has discovered the usefulness of distributed ledgers. Here, judges, law enforcement, and companies publish commitments to information about surveillance measures on the ledger and can provide zero-knowledge proofs that they behave according to the laws. However, these proposals *do not enforce* that, in order to access user trapdoors, evidence must be put onto the ledger first. That means, a secure and accountable escrow and disclosure of end-to-end encryption keys is not considered. Hence, trapdoors kept by a company are still at risk of being covertly misused without leaving any trace.

Very recently, Green et al. propose a misuse-resistant surveillance scheme [GKV21] which compels law enforcement to leaving a warrant on the ledger in order to disclose the encrypted communication of a suspect. To this end, the authors build on extractable witness encryption (EWE) [GLW14], instead of a key escrow scheme, where the publication of a warrant serves as the key (aka witness) to decrypt the communication. Unfortunately, it seems implausible that extractable witness encryption (for general NP languages) does actually exist [Gar+14].[1] Moreover, they prove that any protocol secure in their model would already imply extractable witness encryption (for a highly non-trivial language). The goal of our system is to combine ledger-based auditability of surveillance actions with corruption-resilient key escrow mechanisms. Similar to [GKV21], leaving a warrant on the ledger to access trapdoors is enforced, but without relying on extractable witness encryption.

Goyal et al. [Goy+22] model storing secrets on a blockchain (via secret-sharing them among members of an evolving committee) and retrieving them under some condition. If we apply that idea to our scenario, we could secret-share user-specific trapdoors at committees who only release their shares if they find a warrant on the ledger requesting their specific shares. Benhamouda et al. [Ben+20] also model the storing of a secret on a blockchain, but additionally make committee members anonymous until they finished their work to prevent targeted corruption. Both methods [Ben+20; Goy+22] can result in a lot of overhead for the committee who potentially has to manage millions of secrets. To reduce the committee's workload during the handover phase, we take this idea and combine it with a key-escrow approach. Instead of secret-sharing millions of trapdoors on the blockchain, we only secret-share one item: A secret key for a threshold public key encryption scheme. The trapdoors are not

[1] Even (non-extractable) witness encryption currently has no efficient constructions.

secret-shared on the blockchain, but encrypted under the threshold public key and stored off-chain to reduce blockchain workload. The secret key for that ciphertext is secret-shared on the blockchain and instead of directly retrieving a secret from the blockchain, law enforcement can *request the decryption of the ciphertext.* To increase security, only a part of the trapdoor is decryptable by the blockchain committee; the other part is stored offline at the system operator. Thus, the system remains secure even if the blockchain committee's majority is corrupted. See Section 4.7 for a further discussion of design decisions.

While some other systems [GGM16; KV03] only enable *prospective* surveillance, where law enforcement must prepare surveillance of each user individually *before* this user conducts any transactions, we achieve *retrospective* surveillance, where law enforcement can also access transactions that were conducted in the past. We additionally model this functionality as a building block in the UC framework and prevent users from learning whether their trapdoor was retrieved or not (in contrast to [Ben+20; Goy+22]). Unlike [GKV21], we achieve retrospective surveillance without implying EWE. Moreover, the efficiency estimate of our application shows that our system is realizable and scales favorably: The judge is only involved in granting warrants, law enforcement is only involved when surveilling users and the work on the blockchain only depends on the number of surveilled users, not the number of registered users. We thus present the first UC-secure proof of concept.

Additionally, we add the possibility to *audit* the surveillance decisions of law enforcement. A special party, the *auditor*, has the power to retrospectively examine law enforcement's surveillance decisions and inform the public about possible abuse. Since this party is very powerful, care must be taken as to who takes on this role; for example, a neutral investigative committee might be suitable.

Application scenarios for our auditable surveillance technology range from mobile communication systems (e.g., 5G) and anonymous messaging services over anonymous electronic payments (e.g., Monero, ZCash, etc.) to privacy-preserving video surveillance.

4.1.1. Contribution

We formalize an auditable surveillance system as a building block which can be used to enhance many different (existing) applications with the capability to revoke a user's anonymity or to reveal a user's transaction data under the condition that a law enforcement agency has a court-signed warrant that legally empowers them to do so. Our auditable surveillance system is the first to combine key escrow with accountability without requiring unlikely assumptions (e.g., EWE) while scaling well. We accomplish this through the following contributions.

First and foremost, we provide an abstract ideal functionality for auditable surveillance, $\mathcal{F}_{\text{AS}}$. The functionality $\mathcal{F}_{\text{AS}}$ serves as a building block which allows to enhance protocols with auditable surveillance with relative ease. In particular, this provides a basic target functionality to realize and serves as a separation between the low-level implementation of the auditability mechanism, and the high-level decision of adding auditability to protocols, e.g., choosing what data to make available to law enforcement in anonymous electronic payment systems with auditable surveillance. We formalize our auditable surveillance system in the Universal Composability (UC) framework [Can00; Can20]. The standard approach to formalizing security is to formulate the security properties as a list of individual properties. This approach, however, carries the risk that important security and privacy aspects are overlooked, e.g., because the list may not be exhaustive. This problem does not exist in UC, because the objective is not to formalize a list of individual properties, but to define what an ideal system should look like. UC-security also assures that even if the system is executed in a large environment where many different protocols are executed concurrently, the system retains all security guarantees. Since there are many applications that would benefit from added auditable surveillance, we chose to model its security in the UC framework. This ensures that the system's security and privacy guarantees still hold if the system is run in combination with many different other protocols.

Our second contribution is the protocol Π_{AS} which UC-realizes $\mathcal{F}_{\mathrm{AS}}$. It uses several cryptographic building blocks like commitments, signatures and zero-knowledge proofs to achieve the same strong security and privacy guarantees as $\mathcal{F}_{\mathrm{AS}}$. The protocol Π_{AS} is based on an ideal functionality for auditable decryption, $\mathcal{F}_{\mathrm{AD}}$, for managing the secret-shared secret key for the threshold encryption scheme and answering decryption queries. Our modeling of this building block achieves auditability of requests for secrets and privacy regarding which secrets are released. The auditable decryption functionality $\mathcal{F}_{\mathrm{AD}}$ is also modeled in the UC framework to enable a flexible use of this building block and we provide a protocol Π_{AD} that UC-realizes it. We cast our protocol Π_{AD} in the YOSO (You-Only-Speak-Once) model [Gen+21] (see Section 4.6 for a brief introduction). In this model, protocols are executed by roles, where each role is only allowed to send messages once. Which party is executing a specific role is hidden until it sends its messages. This prevents targeted corruption of parties that comprise the current committee, and thus allows for leveraging global honest-majority (for the set of all nodes operating the blockchain) assumptions for smaller committee sizes, and achieving security against mobile adaptive adversaries.[2]

To summarize our contribution:

(1) We identify and define an ideal UC-functionality, called $\mathcal{F}_{\mathrm{AS}}$, which acts as a building block for auditable surveillance systems or more generally auditable access systems.

(2) We realize this functionality in a setting where the deployment of such systems is of interest (namely, the "blockchain space"). Therein we also specify a UC-functionality $\mathcal{F}_{\mathrm{AD}}$ for auditable decryption, which may be of independent interest. We stress however, that the realization of $\mathcal{F}_{\mathrm{AS}}$ is not in any way restricted to this setting.

4.1.2. Related Work

In this section, we review the academic literature on related topics.

Key escrow. Since the 1990s, many papers, e.g., [KL95; PY00; YY98], have been dealing with different variants of key escrow mechanisms in various domains, where key material is deposited with one or more trusted parties who can then decrypt targeted communications or access devices. In particular, key escrow has also been applied in the scope of e-cash [BGK95] to balance anonymity and accountability. Also, more recent work follows this paradigm. For instance, in [Arf+21] the authors propose protocols for secure-channel establishment of mobile communications that offer a session-specific opening mechanism. A session key is escrowed with n authorities which all need to agree for recovering the session key. The system comes with some addition of security guarantees, e.g., non-frameability. The work [Sav18] considers lawful device access while protecting from mass surveillance. The authors propose the use of self-escrow passcodes which are written to the device itself and can only be retrieved by means of physical access, e.g., via dedicated pins.

[2] Mobile adversaries can adaptively corrupt and uncorrupt parties as long as they do not exceed a certain threshold of simultaneously corrupted parties.

Paper	Lawful-only[1]	Retro-spective[2]	Silent[3]	No EWE needed[4]	Public Statis-tics[5]	UC-secure[6]	Flexible Frame-work[7]
[KV03], [GGM16]	no	no	yes	yes	no	no	no
[Daz+22]	no	yes	yes	yes	no	no	yes
[Bro+23]	no	yes	no	yes	yes	yes	no
[LRC14]	no	yes	yes	yes	yes	no	partially[8]
[JS04]	yes	yes	no	yes	no	no	partially[8]
[GKV21]	yes	yes	yes	no	yes	yes	no
Our System	yes	yes	yes	yes	yes	yes	yes

[1] Whether a warrant is needed for surveillance actions.
[2] Whether surveillance is retrospective or only prospective.
[3] Whether surveillance is silent, e.g., users are not aware that they are surveilled.
[4] Whether the system does not assume the existence of EWE (extractable witness encryption).
[5] Whether the system supports public statistics.
[6] Whether the system is UC-secure.
[7] Whether the work contains a framework that can be used for many different applications.
[8] The system support a limited set of applications.

Table 4.1.: Comparison of our system with the most relevant related work on accountable surveillance systems

Accountable access. Some works have tried to extend key escrow with basic accountability features. We compare ourselves with the most relevant of them in Table 4.1.

In [BCS05] an anonymous yet accountable access control system is proposed. Regarding accountability, the user needs to escrow its identity with a trusted third party (TTP) which is revealed by the TTP if some previously agreed condition bound to the ID using verifiable encryption with labels (where the condition is encoded as label) is satisfied. Liu et al. [LRC14] propose an accountable escrow system focusing on encrypted email communication. In their system users escrow their decryption capability (instead of their private key), essentially by means of a 3-party Diffie-Hellman key exchange, to trusted custodians. Custodians perform decryptions upon request by means of their own private keys and are trusted to log each decryption request to hold the government accountable. Still, the private keys of custodians can be stolen or they can be corrupted in particular as they are well-known to government organizations. The works [KV03] and [GGM16] deal with auditable tracing techniques in the context of e-cash and cryptocurrencies, respectively. The underlying idea is to provide a user either with a randomized version of the authority's public key, where the corresponding secret key is the revocation trapdoor, or a completely random key, which is useless for tracing. The user cannot tell which key it received until later when the authority is enforced to reveal this. The big disadvantage of their approach is that the authority has to decide in advance (e.g., at the beginning of each month) which users should be traced. This could result in practical issues, e.g., when money laundering is suspected, but the transactions of suspects cannot be traced since tracing was not turned on for them. In [Daz+22] a "mutual accountability layer" is added to systems to make operators accountable for opening key-escrowed user transactions. However, this accountability feature only results in the current key escrow committee learning that *some* transaction was opened. The lawfulness of opening a transaction is not verified and information about opening requests are not persistently and publicly stored.

Some proposals try to avoid key-escrow and its misuse potential from the outset. Very recently, a misuse-resistant surveillance scheme with retrospective exceptional access to end-to-end encrypted user communication has been proposed in [GKV21]. Instead of building on key escrow, users are forced to (additionally) encrypt their messages with extractable witness encryption [Gol+13]. Loosely speaking, witness encryption allows to define a policy under which a ciphertext can be decrypted. In their scheme, this is the case when a warrant for the corresponding user signed by a judge has been published on a public ledger. The major disadvantage of their approach is that it is implausible that extractable witness encryption schemes actually exist [Gar+14].

In [JS04], the authors also abstain from using key escrow. Instead, given a warrant, a user has to (verifiable) reveal the transactions of interest itself to the judge. Unfortunately, this prevents silent investigations and leaves a suspect with the option to deny cooperation.

Several works deal with the collection of auditable logs of surveillance actions. In [Bat+12] the authors propose a distributed auditing system for CALEA-compliant wiretaps. The idea is to add Encryptor devices to the wiretaps which send encrypted audit records to a log. With the help of the log, audit statistics can be computed from ciphertexts using homomorphic encryption. Kroll et al. in two not formally published manuscripts [KFB14; Kro+18] propose different systems for accountable access control to user data. Here, law enforcement needs to interact with a set of decryption authorities to decrypt user records, for which a single encryption key is used. Accesses are logged by an auditor party with whom the other parties need to interact continuously in order to confirm the different protocol steps. No end-to-end encryption of user data or the revocation of anonymous records is considered. Also, formal security model and proofs are missing. The work [Fra+18] extending [GP17] uses ledgers to collect accountable information about surveillance action and a hierarchical form of MPC to compute aggregate statistics. However, a secure and accountable escrow and disclosure of user trapdoors is not considered. In particular, if such trapdoors are kept by a company, they can still be misused by the company, stolen by an intelligence agency, etc. without leaving any trace on the ledger. The work [Pan+19] addresses some of the issues of [Fra+18] like, e.g., that government agencies and companies are trusted to regularly post (correct) information on the ledger. This is done by introducing an independent party called Enforcer who serves as the conduit for interactions between them and ensure compliance. However, for this it is assumed that government agencies and companies do never directly communicate with each other. Moreover, they do not make use of ledgers to control access to user trapdoors to disclose end-to-end encrypted communication or revoke anonymous transactions.

Finally, there are number of rather unconventional proposals to impede mass surveillance. In [BR99], the authors introduce the concept of translucent cryptography which, based on oblivious transfer and without relying on key escrow, allows law enforcement to access encrypted messages with a certain probability p. In [WV18], a system is proposed where law enforcement needs to provide a hash-based proof of work in order to recover an encrypted message, intentionally resulting in a high monetary cost (e.g., \$1K-\$1M per message). The work [SW19] considers exceptional access schemes for unlocking devices such as smartphones. To unlock a device, law enforcement needs to get approval by a set of custodians and subsequently locate and get physical access to a randomly selected set of delegate devices to obtain an unlock token. This requires both human and monetary resources of the law enforcement agency. In [Sca19] Scafuro introduces the concept of break-glass encryption for cloud storage where the confidentiality of encrypted cloud data can be revoked exactly once for emergency reasons. This "break" is detectable by the data owner. The author's construction relies on trusted hardware and is a feasibility result rather than a practical solution. Persiano, Phan and Yung [PPY22] recently introduce the concept of *anamorphic encryption*. The idea behind this is that even an attacker who can dictate the messages sent and demand the surrender of a decryption key (e.g., a government), cannot prevent a second, hidden, message from being sent along that only the dedicated recipient can decipher. While this means that one cannot prevent the exchange of hidden messages despite of surveillance, law enforcement agencies still consider surveillance as a useful measure as not all criminals have the knowledge or skill set to exploit this fact or find it convenient.

Storing secrets on a blockchain. There are prior works which model the capabilities of storing secrets and retrieving them under certain conditions thereby replacing the need for extractable witness encryption by relying on a blockchain.

In the recently published "eWEB" system [Goy+22] a dynamic proactive secret sharing (PSS) scheme with an efficient handover phase is constructed and used in a black-box way to store and retrieve secrets on a blockchain. Their system is secure against an adversary that statically corrupts less than half of the blockchain nodes. However, the members of the current committee are publicly known and the identifier of a retrieved secret is revealed to the general public. Hence, in our case, a user would learn that law enforcement is exposing its transactions, which should be prevented.

The system by Benhamouda et al. [Ben+20] uses a similar approach. They introduce an evolving-committee PSS scheme instead, where the selection of the next committee is a part of the secret-sharing scheme itself and the members of the current committee remain anonymous (to anyone, including the members of the previous committee) until they finished their work and hand over the role to the next committee. This enables the system to handle a mobile adversary corrupting $\approx$ 29% of blockchain nodes. However, they do not clearly state how their PSS scheme can be used to retrieve secrets from the blockchain, in particular it is unclear who learns which secrets are retrieved, whether the current committee learns this witness and whether the secret is revealed to the public or not.

Our usage of a blockchain has similarities to [Ben+20; Goy+22], in particular we also use the anonymous committees from [Ben+20], but we additionally model the system as a building block in the UC framework, which [Ben+20; Goy+22] do not. Also, we prevent users from learning whether their secret was decrypted or not (in most cases).

Erwig, Faust and Riahi [EFR21] propose a standalone protocol for threshold decryption in the YOSO model. Our building block for auditable decryption is similar to their protocol, but augmented to satisfy our auditing needs and formalized in the UC-model instead of using game-based security notions.

Brorsson et al. [Bro+23], in a concurrent and independent work, recently published PAPR, an anonymous credential scheme with a retrospective auditable surveillance feature. PAPR and our work both model retrospective auditable surveillance with a detailed UC framework, utilizing techniques such as bulletin boards/blockchains, YOSO, secret-sharing, ZK, and TPKE. Regarding the committee structure, our system differs from PAPR as the sets of users and committee member candidates are distinct, while PAPR's sets are identical. However, it is worth mentioning that PAPR proposes this separation as future work. Notably, our system supports silent anonymity revocation, whereas anonymity revocation in PAPR is inherently non-silent. Additionally, we implement different validity periods, allowing law enforcement to revoke a user's anonymity only for specific periods, offering a more nuanced approach compared to PAPR, where revocation affects all credential showings.

4.2. System Overview

In this section, we provide an overview of our system. We start by introducing the parties, followed by a high-level description of their interactions. Lastly, we give a brief discussion of its (intuitively) captured security properties.

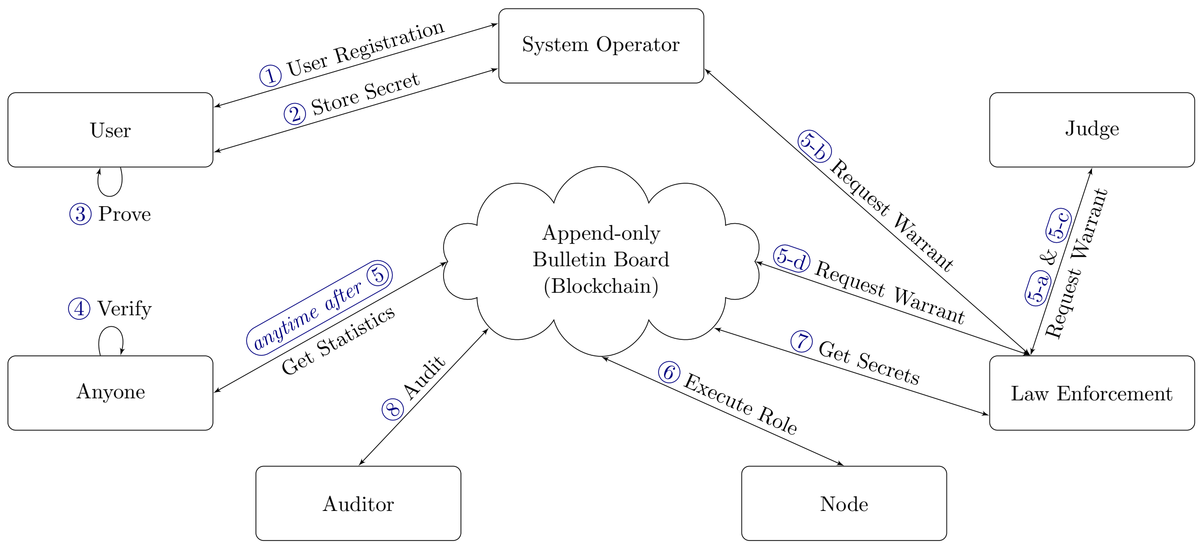


Figure 4.1.: High level system overview. The numbers represent the intended order of task execution.

4.2.1. Parties

Our system consists of the following parties:

System Operator: The system operator SO operates the system (e.g., anonymous payment system, confidential instant messenger service) that is enhanced with auditable surveillance.

Law Enforcement: Law enforcement LE can access a user's escrowed secrets if it is in possession of a valid warrant.

Judge: The judge J grants or rejects warrants.

Auditor: The auditor AU can access information about all used warrants.

Users: The set $\{U_i\}$ of users who want to make use of the application provided.

Nodes: The set $\{N_i\}$ of nodes that is available to execute assigned roles on the blockchain (the shareholder committee).

There is one each of system operator, law enforcement, judge and auditor, but there may be arbitrary many users and blockchain nodes. We consider static corruption of SO, LE and the users U_i. For the nodes N_i we achieve mobile adaptive corruption since each role a node can play is cast in the YOSO model [Gen+21] and only sends messages once. J and AU[3] are assumed to always be honest. Apart from the explicitly modeled parties, some portions of our system are available for *everyone*, for example reading information from the blockchain.

[3] While a corrupt AU would learn all warrants, it is not possible for a corrupt AU to convince a third party of false claims about those warrants.

4.2.2. High-Level System Overview

An overview of the tasks the different parties can execute can be found in Fig. 4.1. We now describe the intended usage of our system. In the following we focus on a specific application and instantiation to make the description clearer. We apply our functionality $\mathcal{F}_{AS}$ to an anonymous message transfer service and instantiate it with the protocols Π_{AS} and Π_{AD} that utilize a blockchain.[4] Note that of course different applications and instantiations are possible as well.

To initialize the system, each party generates their respective cryptographic keys. The public keys of SO, J and AU are posted on the blockchain and available to everyone. The blockchain selects its first committees and creates a threshold encryption keypair, whose public key is available to everyone and whose secret key is secret-shared among the committee members.

Users need to register themselves with SO and create an account to use the system. Each period (e.g., monthly) the user and SO create together a fresh escrow secret that could be used to expose the user's identity in messages sent during that period. The secret is then secret-shared into two *partial secrets*: One part is directly stored at SO[5] and the other part (only known to the user) is encrypted under the committee's threshold public key. The ciphertext itself reveals nothing about the user's identity, but it is linked to the user's identity and the period the secret is valid for with a zero-knowledge (ZK) proof. After depositing a secret for the current period, the user can use the messenger service during that period. To use the anonymous messenger application, the sender would *prove* in zero-knowledge that it has stored a secret for the current period and appended its identity (but encrypted under its escrow secret for the current period) to the current message. This proof can be *verified* by anyone, in particular the receiver and SO, where the latter can block non-complying messages.

If the law enforcement agency LE has a legitimate interest in disclosing one or multiple user secrets, it can request a warrant from the judge J. Each warrant consists of a set of sub-warrants and each sub-warrant states which user should be surveilled in which period. Additional information is included in each sub-warrant, for example the name of the court, the reason for surveillance, etc. If J signs the warrant, LE can get information about each user from SO, i.e., the partial secret stored by SO and the ciphertexts containing the other part of the secret. With a signed warrant and the additional information from SO, LE is able to retrieve the partial secrets for the users and periods in that warrant from the blockchain. In particular, LE convinces the current committee in zero-knowledge that it indeed has a warrant signed by J for that set of ciphertexts. Upon verifying the proof, the committee then decrypts the ciphertexts and reveals the other partial secrets to LE, who can then reconstruct the users' full secrets. Given the (anonymous) message data of the messenger system operator, LE can now match the previously anonymous messages to users in the warrant (but only for messages sent during the period specified in the warrant).

Security and Privacy Protection Mechanisms. To prevent misuse of the system or even mass surveillance, several countermeasures are supported by the system. Firstly, a system-wide policy function prevents any warrants violating that policy. For example, it may disallow surveillance of an individual user for more than 12 months (by a single warrant). Secondly, LE needs to be in possession of a valid warrant signed by J to request a secret and the blockchain committee first verifies the validity of this warrant before decrypting a secret. There exists the risk that the signing key of the judge may get

[4] The features of a blockchain include publicly viewable and non-editable information as well as easy committee formation. Alternatively, an append-only bulletin board can be used. In this case, a committee has to be formed by some other means.

[5] Since we use a Blum coin toss to jointly generate the secret, SO knows a share of the secret anyway.

stolen and an attacker could use the stolen key to sign a warrant itself and thus retrieve some secrets. We cannot prevent this entirely (other than suggest that judges secure their secret keys properly), but have mitigation measures. To retrieve a decrypted secret from the blockchain, the request needs to publish some information about the respective warrant on the blockchain, thus leaving visible proof that such a request took place. What information about each warrant should be publicly available is declared by a system-wide transparency function. Choosing a suitable transparency function for the system can be a challenging task since one needs to balance the public's desire for information, the privacy of surveilled users, and LE's need for silent surveillance. The transparency function should, for example, never publish the names of the surveilled individuals. But it may, for example, publish the number of different users in the warrant, the periods in which the retrieved secrets are valid and the name of the court that signed the warrant. Assuming the latter information is published, each court can monitor the blockchain and upon seeing more warrants signed by its key than they actually signed, they can detect key theft and henceforth take appropriate measures.[6] Thirdly, assuming an appropriate transparency function, anyone can get useful information about all enforced warrants through publicly available statistics. For example, if the transparency functions reveals the number of different users in the warrant, the number of currently surveilled users becomes public. If this number suddenly skyrockets, it is an indication of misuse of power. Fourthly, the auditor AU has the capability (as it can access the "full" warrant information on the blockchain) to conduct a detailed investigation of enforced warrants. It may check for any peculiarities and prove statements about the stored warrants to an arbitrary entity without revealing any further information. For instance, such a statement might be that none of the warrants issued within a certain period of time involve a certain user.

On the Necessity of the Prove and Verify Tasks. To be used in an application, the auditable surveillance functionality $\mathcal{F}_{\mathrm{AS}}$ must provide a suitable interface. In most applications, users need to be able to *prove* that they escrowed a secret, e.g., a secret for the current period in the messenger application ASTE. A first idea to realize that, is for $\mathcal{F}_{\mathrm{AS}}$ to provide a digital signature on user identity, escrowed secret and current period to the user. However, formulating a usable (let alone zero-knowledge-compatible) UC signature scheme turns out to be a daunting task, since signatures in UC are riddled with subtleties [BH04; Cam+16; Can04; KF08], and indeed many modeling artifacts occur.

To circumvent such problems, $\mathcal{F}_{\mathrm{AS}}$ directly provides the possibility for users to prove statements about their identity *uid*, their escrowed secret *secret*, some validity period *vper* and other information. Such a proof can show a statement of interest w.r.t. *vper*, for a witness which includes $(uid, secret)$, *and* the proof ensures that the secret *secret* for *uid* is stored for period *vper*. Overall, this approach is more general and easier to use.

4.2.3. Security Properties and Trust Assumptions

We now summarize the desired security and privacy properties of our building block for auditable surveillance in an informal and intuitive manner:

- (Non-colluding[7]) users are not aware whether LE issued a request to recover one of their secrets.

[6] Our system actually only supports a single non-revocable judge key to keep the model from being overly complex, but the extension to several different and revocable judge keys is straightforward.

[7] This security property holds for a user colluding with other users and blockchain nodes but not one colluding with SO or LE.

- To use the Prove task (cf. Fig. 4.1) for an application on top, e.g., to prove some statement involving the user's identity, the user needs to have escrowed a secret for the respective period.
- The privacy guarantees of the application on top are only breached if a warrant was granted. And even in that case, only the users covered by the warrant have (some of) their data exposed.
- LE can only request secrets for warrants that comply with the system policy.
- After LE requested a decryption by the blockchain, the publicly available information about the warrant (for statistical purposes) and the information about the warrant that is only available to AU are permanently stored on the blockchain and can not be modified or removed afterwards.
- Anyone is able to retrieve publicly available statistics on all enforced warrants.
- AU has the capability to provide the general public with *provably correct* statistics about the enforced warrants.[8]
- Even if SO and LE collude, they can not expose any escrowed user secrets.
- Likewise, even if a majority of the blockchain nodes collude, they can not reconstruct a user's secret.

We assume the judge J and auditor AU to be honest. Since AU has the power to read all warrants in the clear, it is a very powerful entity that could potentially misuse that power. In reality, one could decentralize that trust by having multiple auditor parties that utilize threshold cryptography or multi-party computation. Since we trust the jurisdiction, we also model J as an honest party and assume that it honestly follows the protocol. Warrants that were granted but legally not justified can be detected upon request to AU. The existence of judge-signed warrants also ensures that LE can also only request user information from SO for which it has a signed warrant.

Although SO is corruptible in our system, we implicitly have to trust it in some aspects: We assume that SO cooperates with other parties and responds to messages. A SO that ignores messages from other parties could bring the system to a halt or deny individual users' participation in the system. Since SO has a monetary motivation to keep the system running for a long time, we believe these assumptions to be reasonable. Note that while a corrupt SO cannot send false data (e.g., pretend that a ciphertext belongs to another honest user), it can omit some data (e.g., data from colluding corrupt users) when sending it to LE. This is a general problem that can be discouraged through laws.

4.3. $\mathcal{F}_{\text{AS}}$: A Formal Model for Auditable Surveillance Systems

In this section we introduce the functionality $\mathcal{F}_{\text{AS}}$, our formal model for our auditable surveillance system, which is independent of the application. We first introduce some variables that are essential for the functionality in Table 4.2.

We model our system in the Universal Composability (UC) framework [Can00; Can20]. For a very brief introduction to UC, see Section 2.7. Before we start, we introduce some writing conventions.

Remark 4.1 (Writing Conventions). We use the following conventions when describing the functionalities $\mathcal{F}_{\text{AS}}$ and $\mathcal{F}_{\text{AD}}$:

[8] Since AU has access to the full warrants, its statistics can be more detailed than those the general public can compute. AU could even prove to third parties (e.g., a parliament) facts about specific warrants without revealing the full warrant.

Identifier	Description
pid	The party identifier for the UC framework (e.g., a physical identifier)
uid	The (self-chosen) identity of a user
secret	The escrowed secret
vper	The validity period for a single secret (e.g., the current month)
W	A warrant from LE to retrieve (multiple) user secrets with $W = (W_1, \dots, W_v)$ and $W_i = (uid_i, vper_i, meta_i)$
v	number of subwarrants W_i inside a warrant W
meta	Meta information about a warrant. Some of it is public information and some is only meant for the auditor
W^{pub}	The information about the warrant that should be known to the public

Table 4.2.: Variables used in $\mathcal{F}_{\text{AS}}$

- As usual, we assume authenticated channels between all parties.
- We omit (sub)session identifiers, written *sid* (resp. *ssid*), which are used to identify separate protocol instances (resp. separate instances of tasks in a protocol instance). Adding *sid*s and *ssid*s is straightforward.
- When we write "Input P: (...)" for P $\in$ {SO, AU, J, LE}, the functionality first checks that the party calling the functionality has indeed that role.[9] If that is not the case, the functionality ignores the message.
- The first task that has to be called is the System Init task. If any parties send any other messages to the functionality before the System Init task has been completed, those messages are ignored.
- We use *delayed output* to some party P as a short-hand notation for the following protocol: Send (Output, TaskName, P) to the adversary and wait for message (Allow, TaskName, P) from the adversary. Then send (*value*) to P.
- When we write "Output P: (*value*)", this is a shorthand for "generate delayed output (*value*) to party P".
- If, in contrast, we write "immediately output (*value*) to P", the message is directly sent to party P without notifying the adversary.
- We use notion of "urgent request"/"immediate response" from [Cam+16], as well as responsive environments, to model that the adversary can not make any other calls to functionalities or hybrid functionalities before sending a response to the current request. By UC-realization, we mean realization w.r.t. to responsive environments.
- The notation "Send (*value*) to the adversary" is also a shorthand for "Send (*value*) to the adversary and wait for message (Ok) from the adversary before continuing".

We now describe how our auditable surveillance system is modeled in the UC framework by presenting the ideal functionality $\mathcal{F}_{\text{AS}}$ in Figs. 4.2 and 4.3, as well as sketching the individual tasks.

System Init. This task initializes the system and is thus the first task that must be invoked. The ideal functionality does not respond to other messages until this task has been successfully executed.

[9] The functionality can do that because it knows the party identifiers (*pid*s) of all parties in the system

Functionality $\mathcal{F}_{\text{AS}}$

System Parameters:
- f_t — Transparency function. Outputs public information of a warrant $W^{\text{pub}} \leftarrow f_t(W)$.
- f_p — Policy Function. Checks whether a given warrant is allowed by system policy. Interface is $\{0,1\} \leftarrow f_p(W)$.
- $\mathcal{S}$ — Space of secrets
- $\mathcal{R}$ — an NP relation for statements about stored secrets: Contains $(stmt_R, wit_R) \in \mathcal{R}$ pairs where $stmt_R$ is a statement and wit_R is a witness for that statement
- System pids: pid_{SO}, pid_{J}, pid_{AU}

Functionality State:
- L_I: List of initialized parties (initially empty). Contains (pid) entries.
- L_U: List of registered users (initially empty). Contains (pid, uid) pairs.
- L_S: List of stored secrets (initially empty). Contains $(uid_i, vper_i, secret_i)$ entries.
- L_W: List of warrants that were requested by LE from J (initially empty). Entries are of the form (W, b), where b is a bit that states whether the warrant was granted or denied by the judge.
- L_π: List of proofs (initially empty). Entries are of the form $(stmt_R, wit_R, \pi, b)$, where the bit b states whether the relation is fulfilled or not.

System Init:
- Input SO & J & AU: (Init, SO/J/AU)
  - If this is the first time that this task is invoked, then do whatever is stated in "Behavior". Else, ignore the messages.
- Behavior:
  (1) Create empty lists L_I, L_U, L_S, L_W, and L_π
  (2) Send (Init) to the adversary
  (3) Add pid_{SO} and pid_{J} to L_I
- Output SO & J & AU: (InitFinished)

Party Init:
- Input some Party P: (PInit)
- Behavior:
  (1) If $pid \notin \mathsf{L}_\mathsf{I}$, then store pid in L_I
  (2) Send (PInit) to the adversary
- Output to P: (PInitFinished)

User Registration:
- Input U: (Register, uid)
- Input SO: (Register, uid')
- Behavior:
  (1) As soon as U gave input, send (Register, pid, uid) to the adversary. Wait for (Ok) from the adversary and input from SO before continuing.
  (2) If $uid \neq uid'$, then abort (Wrong inputs)
  (3) If $(pid, \cdot) \in \mathsf{L}_\mathsf{U}$, then abort (User already registered)
  (4) If $(\cdot, uid) \in \mathsf{L}_\mathsf{U}$, then abort (Identity already taken)
  (5) Add (pid, uid) to list of registered users L_U
  (6) Store pid in L_I
- Output U & SO: (Registered)

Store Secret:
- Input U: (StoreSecret, uid, $vper$)
- Input SO: (StoreSecret, $vper'$)
- Behavior:
  (1) If SO is corrupted (and U is honest): As soon as U gave input, send (StoreSecret, $vper$) to the adversary and then wait for input from SO
  (2) If $vper \neq vper'$, then abort
  (3) Check if user is registered: If $(pid, uid) \notin \mathsf{L}_\mathsf{U}$, then abort
  (4) Check if user already registered a secret for the current validity period: If there exists an entry $(uid, vper, \cdot)$ in the list of stored secrets L_S, then abort
  (5) Generate secret: $secret \overset{R}{\leftarrow} \mathcal{S}$
  (6) Store $(uid, vper, secret)$ in the list of stored secrets L_S
- Output U: (SecretStored, $secret$)
- Output SO: (SecretStored, uid)

Request Warrant:
- Input LE: (RequestWarrant, W)
- Behavior:
  (1) Check if policy function allows that warrant: If $0 \leftarrow f_p(W)$, then abort (Warrant not allowed by policy function)
- Output J: (RequestWarrant, W)
- Input J: (b)
- Behavior:
  (1) If there already exists an entry $(W, \cdot) \in \mathsf{L}_W$, then abort (Warrant already processed)
  (2) If SO is honest: If $b = 1$, then send (RequestWarrant, $f_t(W)$, $\left|\widetilde{W}\right|$, v) to the adversary and wait for message (Ok) from the adversary
  (3) If SO is corrupted: Send (RequestWarrant, W, b) to the adversary and wait for message (Ok) from the adversary
  (4) Append (W, b) to L_W
- Output LE: (RequestWarrant, b)

Figure 4.2.: The ideal functionality $\mathcal{F}_{\text{AS}}$

Functionality $\mathcal{F}_{\mathrm{AS}}$ (continued)

Get Secrets:

- Input LE: (GetSecrets, W)
- Behavior:
  1. Check if warrant was granted by Judge: If no entry $(W, 1)$ exists in L_W, then abort (Warrant either not requested or not granted)
  2. Parse warrant: $(W_1, \dots, W_v) \leftarrow W$ and $(uid_i, vper_i, meta_i) \leftarrow W_i$
  3. For each i from 1 to v:
     (a) Check if secret is stored: Get $secret_i$ for which an entry $(uid_i, vper_i, secret_i)$ exists in the list of stored secrets L_S. If none exists, then set $secret_i = \bot$.
  4. If LE is honest: Send (GetSecrets) to the adversary
- Output LE: (GotSecrets, $(secret_1, \dots, secret_v)$)

Get Statistics:

- Input some Party P: (GetStatistics)
- Behavior:
  1. Send (GetStatistics) to the adversary
  2. Initialize empty list $\mathsf{L}_{\mathsf{Stats}}$
  3. For every $(W, b) \in \mathsf{L}_W$ with $b = 1$:
     (a) Apply transparency function to warrant: $W^{\mathrm{pub}} \leftarrow f_t(W)$
     (b) Append W^{pub} to $\mathsf{L}_{\mathsf{Stats}}$
- Output to P: (GotStatistics, $\mathsf{L}_{\mathsf{Stats}}$)

Audit:

- Input AU: (AuditRequest)
- Behavior:
  1. Send (AuditRequest) to the adversary
  2. Initialize empty list L_{AU}
  3. For every $(W, b) \in \mathsf{L}_W$ with $b = 1$, append W to L_{AU}
- Output to AU: (AuditAnswer, L_{AU})

Prove:

- Input U: (Prove, $stmt_R$, wit_R)
- Behavior:
  1. Parse $(uid, secret, wit') \coloneqq wit_R$
  2. Parse $(vper, stmt') \coloneqq stmt_R$
  3. Let pid_{U} be the pid of the calling party
  4. If $\mathcal{R}(stmt_R, wit_R) \neq 1$ or $(pid_{\mathrm{U}}, uid) \notin \mathsf{L}_{\mathsf{U}}$ or $(uid, vper, secret) \notin \mathsf{L}_{\mathsf{Secrets}}$, then abort
  5. Send urgent request (Prove, $stmt_R$) to the adversary
  6. Wait for immediate response (Proof, π) from the adversary
  7. Store $(stmt_R, wit_R, \pi, 1)$ in L_π
- Output U (immediate): (Proof, $stmt_R$, wit_R, π)

Verify:

- Input some party P: (Verify, $stmt_R$, π)
- Behavior:
  1. If $pid \notin \mathsf{L}_{\mathsf{I}}$, then abort (Verifier did not initialize)
  2. For $(stmt_R, \pi)$, check if there exists $(stmt_R, wit_R, \pi, b) \in \mathsf{L}_\pi$
  3. If entry exists:
     (a) (Immediately) output (Verification, $stmt_R$, π, b) to P
  4. If entry does not exist:
     (a) Send urgent request (Verify, $stmt_R$, π) to the adversary
     (b) Wait for immediate response (witness, wit_R) from the adversary
     (c) Set $b \leftarrow \mathcal{R}(stmt_R, wit_R)$
     (d) Only if SO is honest:
        (i) Parse $(uid, secret, wit') \coloneqq wit_R$
        (ii) Parse $(vper, stmt') \coloneqq stmt_R$
        (iii) If $(uid, vper, secret) \notin \mathsf{L}_{\mathsf{Secrets}}$, then set $b = 0$ (User has not stored a secret)
     (e) Store $(stmt_R, wit_R, \pi, b)$ in L_π
     (f) Prepare output message (Verification, $stmt_R$, π, b)
- Output to P (immediate): (Verification, $stmt_R$, π, b)

Figure 4.3.: The ideal functionality $\mathcal{F}_{\mathrm{AS}}$ (continued)

Party Init. If parties want to use the Verify task, they have to initialize with the system first. While in the ideal world this is achieved by the ideal functionality just adding them to the list L_{I}, this will correspond to the parties getting all needed CRSs and public keys in the real world. Note that users do not have to explicitly call this task, as they are already added to L_{I} during User Registration.

User Registration. To participate in the auditable surveillance system (and to use the application on top) the user needs to create an account with SO. It is ensured that each user can only create one account. The user can choose an unique identity (uid) under which it will be known henceforth.

Store Secret. Each period (e.g., monthly) the user needs to deposit a new secret. Since $\mathcal{F}_{\text{AS}}$ is a trustworthy incorruptible entity, we do not need to encrypt it and directly store the secret inside $\mathcal{F}_{\text{AS}}$. $\mathcal{F}_{\text{AS}}$ first checks that the user is registered in the system and has not yet stored a secret for the current period. Then, $\mathcal{F}_{\text{AS}}$ draws a fresh secret (so the user can not influence what his secret will be) and stores it in its internal storage. After this task is finished, the user can now use the application, which is not modeled directly in $\mathcal{F}_{\text{AS}}$.

Request Warrant. In this task LE can request a warrant from J. The functionality itself first checks if the proposed warrant W complies with the system's policy function f_p. Then it gives J the opportunity to approve ($b = 1$) or deny ($b = 0$) that warrant. $\mathcal{F}_{\text{AS}}$ also ensures that each warrant is only processed once (this ensures that the statistics calculated later will be correct).

Get Secrets. With a granted warrant LE can now retrieve the secrets for all $(uid, vper)$ pairs inside the warrant. $\mathcal{F}_{\text{AS}}$ first verifies that the warrant was approved by J and then outputs all secrets corresponding to that warrant to LE.

Get Statistics. Every party of the system can query this task to enable the general public to access statistics about the warrants. For every warrant granted by J the transparency function f_t is computed to get the publicly available information W^{pub} about that warrant. The publicly available information about each warrant are then returned to the party asking for statistics.

Audit. When AU gets tasked with calculating detailed statistics or with investigating the case of a specific user, it can call this task. AU is then provided with all warrants that were approved by J. Since we assume AU to be a trustworthy entity (or a group of entities that perform this task in a multi-party computation) we can provide AU with the warrants in the clear. The actual execution of AU's task takes place outside our system, we only provide AU with the necessary information.

Prove and Verify. The zero-knowledge proof interface is used to build applications on top of $\mathcal{F}_{\text{AS}}$. The Prove task allows a prover to generate a proof π for a statement $stmt_R$ in some NP-relation $\mathcal{R}$, where the witness wit_R includes the user's identity and escrow secret for the chosen period. The Verify task allows to check the validity of a proof. As an example, a user may prove that it correctly encrypted its identity under its escrow secret key. Since we do not limit our system to a single application, this generic proof/verify interface enables the flexible use of different applications.

Remark 4.2 (Discussion of the leaked information). In the description of $\mathcal{F}_{\text{AS}}$ (c.f. Figs. 4.2 and 4.3) there are several messages of the form "Send ($value$) to the adversary". This is due to modeling the system in the UC framework: In UC, privacy guarantees are modeled by explicitly sending all information that an adversary could learn in the real world to the adversary. Note that the information the adversary learns in $\mathcal{F}_{\text{AS}}$ are all either information any party could learn (by monitoring public changes on the blockchain or the bulletin board) or information a corrupted party learns during the protocol Π_{AS} either way. While the information the adversary could learn in $\mathcal{F}_{\text{AS}}$ might seem like more information than necessary, this is again due to the nature of the UC framework: In the UC framework, there exists only a *single* adversarial party that can corrupt several other parties. In the real world, this corresponds to the scenario that *all* dishonest parties collude and share all the information they gathered with each other. Consequently, a UC-adversary learns a lot of information. However, one should note that in reality dishonest parties learn significantly less information if they do not cooperate.

4.4. Π_{AS}: A Protocol to Realize $\mathcal{F}_{\text{AS}}$

The functionality $\mathcal{F}_{\text{AS}}$ represents an *ideal* version of the system we want to achieve. Since in practice we do not want to rely on trusted third parties to perform our calculations for us, we build a protocol Π_{AS} *in the real world* that achieves the same security guarantees as $\mathcal{F}_{\text{AS}}$. We later prove that our constructed protocol Π_{AS} UC-realizes $\mathcal{F}_{\text{AS}}$ in the $\{\mathcal{F}_{\text{AD}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{BB}}, \mathcal{G}_{\text{CLOCK}}\}$-hybrid model. Since setup assumptions are common when modeling UC functionalities due to strong impossibility results [CF01], we also need to rely on several setup assumptions: We use the well-known functionality $\mathcal{F}_{\text{CRS}}$ (cf. Fig. 2.2) which enables access for all parties to a *common reference string* (CRS), set up by a trusted party with a given distribution. We also use an external bulletin board functionality $\mathcal{F}_{\text{BB}}$ (cf. Fig. 2.3), where any party can register a single (uid, v) pair associated with its identity, where the *uid*s need to be unique. Any party can retrieve registered values v, which in our case are public encryption keys.

We modularize our realization by outsourcing the auditable decryption of ciphertexts to another hybrid functionality $\mathcal{F}_{\text{AD}}$, which we describe in Section 4.5. This hybrid functionality idealizes the primitive of a blockchain with an evolving set of committees where the committee members are anonymous until they finished their work. The functionality is parameterized by a (threshold) PKE scheme and provides everyone with access to a public key under which secrets can be encrypted. Then, given suitable auditing and authorization information, decryption of a ciphertext can be requested. In our case this information will be a judge-signed warrant for that ciphertext. We outsource the auditable decryption for several reasons:

(1) A primitive for auditable threshold decryption is an interesting building block in itself and to the best of our knowledge no UC formalization of that primitive exists yet.

(2) It simplifies the security analysis of our system, since we can first assume that the decryption is handled by a trustworthy party. In a second step we then replace the auditable decryption functionality $\mathcal{F}_{\text{AD}}$ with a protocol that realizes it.

Since blockchains with evolving committees generally require some concept of *time* (e.g., to ensure that the committee changes daily), the functionality $\mathcal{F}_{\text{AD}}$ utilizes the global clock functionality $\mathcal{G}_{\text{CLOCK}}$ (cf. Fig. 2.5) from [Bad+17] to model time.

We now first elaborate on some of the core techniques we use in Π_{AS} and give the full description of Π_{AS} afterwards. Note that the hybrid functionalities (except $\mathcal{F}_{\text{AD}}$) are given in Section 2.8. The hybrid functionality $\mathcal{F}_{\text{AD}}$ will be described in Section 4.5.

System Init. SO and J each create a signing keypair. Then SO, J, and AU initialize the auditable decryption functionality $\mathcal{F}_{\text{AD}}$ together. The functionality $\mathcal{F}_{\text{AD}}$ is then used to provide all parties with access to the needed CRSs and public keys.

Party Init. If parties want to use the Verify task, they have to initialize with the system first. Thus, in this task they utilize $\mathcal{F}_{\text{AD}}$ to get all needed CRSs and public keys Note that users do not have to explicitly call this task, as they get all needed CRSs and public keys during User Registration.

User Registration. Users create a signing keypair $(\mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}})$ during registration. To ensure at most one account per user, we use an idealized bulletin board $\mathcal{F}_{\mathrm{BB}}$, where any party can register a single $(uid, \mathsf{vk}_{\mathrm{U}})$ pair associated with its identity, where the *uid*s need to be unique. Any party can retrieve registered values.

Store Secret. To store a fresh secret with validity period *vper*, user and SO jointly create a fresh secret *secret* with a Blum coin toss, where each party draws a partial secret: SO draws sec_1 and the user draws sec_2. The full secret is then $secret \coloneqq sec_1 \oplus sec_2$, but only learned by the user. The operator directly stores the partial secret sec_1 (along with the user's identity *uid* and the current period *vper*). The user encrypts the partial secret sec_2 under the public threshold encryption key pk of $\mathcal{F}_{\mathrm{AD}}$, sends the resulting ciphertext *ct* to SO and proves in zero-knowledge that it calculated all values honestly. SO also stores the ciphertext *ct* and provides the user with a (blinded) signature on $(uid, vper, secret)$ (without learning *secret*). The user can then utilize this signature in the application on top to prove to another party that it indeed stored a secret for that validity period.

Request Warrant. First, J signs the warrant W proposed by LE to convince third parties that it indeed has approved the warrant. Since the auditable decryption functionality $\mathcal{F}_{\mathrm{AD}}$ needs to know which ciphertexts should be decrypted, we need J to also sign all ciphertexts *ct* containing the partial secrets sec_2 associated with the warrant W. Therefore, we additionally include SO in this protocol: LE sends the signed warrant to SO and asks for the corresponding ciphertexts *ct* along with the stored partial secrets sec_1.[10] Afterwards, LE provides J with the ciphertexts *ct* to get a signature on $\widetilde{W}$, which is the warrant W *including* the ciphertexts.[11] LE can now utilize the hybrid functionality $\mathcal{F}_{\mathrm{AD}}$ to request the decryption of all ciphertexts corresponding to the warrant $\widetilde{W}$.

Get Secrets. After $\mathcal{F}_{\mathrm{AD}}$ processed the decryption request, LE can retrieve the partial secrets sec_2 for a warrant from $\mathcal{F}_{\mathrm{AD}}$. Of course, $\mathcal{F}_{\mathrm{AD}}$ verified the validity of all requests and partial secrets for invalid requests can not be retrieved. Then, LE uses the already stored partial secrets sec_1 (obtained from SO) to reconstruct the full user secrets $secret \coloneqq sec_1 \oplus sec_2$ for all users in the warrant.

Get Statistics and Audit. Since $\mathcal{F}_{\mathrm{AD}}$ knows all requested[12] warrants, $\mathcal{F}_{\mathrm{AD}}$ can directly give the desired information for those tasks.

Prove and Verify. The Prove task is a local task in which the user uses a NIZKPoK to create the proof π itself. Similarly, the Verify task is also a local task in which the validity of the statement is verified using the NIZKPoK.

Before delving into the full details of the protocol, we introduce the NP-relations needed in Π_{AS} for the zero-knowledge proofs:

[10] This of course enables SO to guess which users are or will be tracked by LE. But in practice this could be amended either by SO just sending *all* its information to LE or by LE using private information retrieval (PIR) to get just the ciphertexts for the current warrant without SO learning which ciphertexts were retrieved.

[11] Before sending the request to J, LE checks the users' signatures. Before answering the request, J checks the (same) signatures as well. Since we assume that J is always honest, it would be sufficient for only J to check the signatures. But we intentionally let LE check the signatures first to filter out invalid requests before forwarding them to J, to reduce J's workload.

[12] Note that these tasks provide the parties with information about all *requested* warrants, independently of whether the secrets were actually retrieved or not.

Details of the relation $\mathcal{R}_{ss}^{AS}$

$wit \coloneqq (secret, sec_2, decom_{\text{sec}}, decom_{sec_2}, r)$
$stmt \coloneqq (sec_1, com_{\text{sec}}, com_{sec_2}, \mathsf{pk}, ct, \mathsf{crs}_{\text{com}})$

$\mathcal{R}_{ss}^{AS} \coloneqq \{(stmt, wit) \mid$

- $secret = sec_1 \oplus sec_2$
- $\mathsf{Open}(\mathsf{crs}_{\text{com}}, com_{\text{sec}}, decom_{\text{sec}}, secret) = 1$
- $\mathsf{Open}(\mathsf{crs}_{\text{com}}, com_{sec_2}, decom_{sec_2}, sec_2) = 1$
- $ct = \mathsf{Enc}(\mathsf{pk}, sec_2; r)$

$\}$

Figure 4.4.: ZK-relation $\mathcal{R}_{ss}^{AS}$

Details of the relation $\mathcal{R}_{zk}^{AS}$

$wit \coloneqq (uid, secret, wit', \sigma_{\text{ss}}, com_{\text{uid}}, decom_{\text{uid}}, com_{\text{sec}}, decom_{\text{sec}}, com_{\text{vper}}, decom_{\text{vper}})$
$stmt \coloneqq (vper, stmt', \mathsf{vk}_{\text{SO}}, \mathsf{crs}_{\text{com}})$

$\mathcal{R}_{zk}^{AS} \coloneqq \{(stmt, wit) \mid$

- $\mathsf{Vfy}(\mathsf{vk}_{\text{SO}}, (com_{\text{uid}}, com_{\text{sec}}, com_{\text{vper}}), \sigma_{\text{ss}}) = 1$
- $\mathsf{Open}(\mathsf{crs}_{\text{com}}, com_{\text{uid}}, decom_{\text{uid}}, uid) = 1$
- $\mathsf{Open}(\mathsf{crs}_{\text{com}}, com_{\text{sec}}, decom_{\text{sec}}, secret) = 1$
- $\mathsf{Open}(\mathsf{crs}_{\text{com}}, com_{\text{vper}}, decom_{\text{vper}}, vper) = 1$
- $wit_R = (uid, secret, wit')$
- $stmt_R = (vper, stmt')$
- $\mathcal{R}(stmt_R, wit_R) = 1$

$\}$

Figure 4.5.: ZK-relation $\mathcal{R}_{zk}^{AS}$

- Relation $\mathcal{R}_{ss}^{AS}$ (see Fig. 4.4) is used in Store Secret by a user to prove to the system operator that they honestly encrypted the partial secret sec_2.
- Relation $\mathcal{R}_{zk}^{AS}$ (see Fig. 4.5) is the NP-relation used in the Prove and Verify tasks to prove some statement about a user's identity, secret and the current period.

We are now ready to give the complete description of the protocol Π_{AS} that UC-realizes the auditable surveillance functionality $\mathcal{F}_{\text{AS}}$.

Protocol Π_{AS}

System Parameters:

- f_t — Transparency function. Gets a preliminary warrant W as input and outputs what should be publicly known about that warrant. Interface is $W^{\text{pub}} \leftarrow f_t(W)$.
- f_p — Policy Function. Checks whether a given warrant is allowed by system policy. Interface is $\{0, 1\} \leftarrow f_p(W)$.
- n — Number of committee members
- $\mathcal{S}$ — Space from which the secrets are drawn
- Signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, where all algorithms are PPT and Vfy is deterministic

- Commitment scheme $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$, where all algorithms are PPT and Open is deterministic
- Re-randomizable Threshold Public Key Encryption scheme $\mathsf{TPKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TDec}, \mathsf{ShareVer}, \mathsf{Combine}, \mathsf{Rand}, \mathsf{Sk2Pk})$, where all are PPT and Dec, TDec, ShareVer, Combine and Sk2Pk are deterministic
- Non-interactive Zero-Knowledge Proof of Knowledge scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where all are PPT and Verify is deterministic
- $\mathcal{R}$ — an NP relation (The code threats $\mathcal{R}$ as a binary function)
- ZK-Relations $\mathcal{R}_{ss}^{AS}$ (see Fig. 4.4), and $\mathcal{R}_{zk}^{AS}$ (see Fig. 4.5)
- System pids: pid_{SO}, pid_{J}, pid_{AU}

States of the Parties:

- Each user stores:
  - CRS: $\mathsf{crs} := (\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{crs}_{\mathrm{com}})$
  - SO public key: $\mathsf{vk}_{\mathrm{SO}}$ (to verify signatures from SO)
  - Public threshold encryption key: pk
  - Own signing keypair: $(\mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}})$
  - Registration data: $(uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}})$
  - List of own secrets. Entries are of the form $(secret, com_{\mathrm{sec}}, decom_{\mathrm{sec}}, vper, com_{\mathrm{vper}}, decom_{\mathrm{vper}}, \sigma_{\mathrm{ss}})$
- LE stores:
  - List of warrants. Entries are of the form $(W, b, \widetilde{W}, \sigma_{\widetilde{W}})$
  - List of ciphertexts requested for decryption. Entries are of the form $(uid_i, vper_i, ct_i, sec_{1,i})$
- J stores:
  - Own signature keypair: $(\mathsf{vk}_{\mathrm{J}}, \mathsf{sk}_{\mathrm{J}})$ (to sign warrants)
  - CRS: $\mathsf{crs}_{\mathrm{com}}$
  - List of already processed preliminary warrants. Entries are of the form (W, b)
  - List of already processed enhanced warrants. Entries are of the form $(\widetilde{W}, \sigma_{\widetilde{W}})$
- SO stores:
  - CRS: $\mathsf{crs} := (\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{crs}_{\mathrm{com}})$
  - Public verification key of the judge: vk_{J}
  - Public threshold encryption key: pk
  - Own signature keypair: $(\mathsf{vk}_{\mathrm{SO}}, \mathsf{sk}_{\mathrm{SO}})$. Messages are either of the form com_{uid} or of the form $(com_{\mathrm{uid}}, com_{\mathrm{sec}}, com_{\mathrm{vper}})$
  - List of registered users. Entries are of the form $(uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}}, \mathsf{vk}_{\mathrm{U}})$
  - List of stored partial secrets. Entries are of the form $(uid, vper, ct, \sigma_{\mathrm{U}}, sec_1, com_{sec_1}, decom_{sec_1}, \sigma_{sec_1})$

System Setup ($\mathcal{F}_{\mathrm{CRS}}$):

- $(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{td}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
- $\mathsf{crs}_{\mathrm{com}} \leftarrow \mathsf{COM.Setup}(1^\lambda)$,
- Return $\left(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{crs}_{\mathrm{com}}\right)$

System Init:

- Note:
  - Each of SO, J, AU only executes the following the first time this input is received and ignores all further inputs of this form.

- Each of SO, J, AU ignores all other inputs and messages until this has been executed.

- Input SO: (Init, SO)
- Input J: (Init, J)
- Input AU: (Init, AU)
- Behavior:

(1) AU:
  (a) Initialize $\mathcal{F}_{\text{AD}}$:
    AU $\to \mathcal{F}_{\text{AD}}$: (Init, AU)
    $\mathcal{F}_{\text{AD}} \to$ AU: (InitFinished)

(2) J:
  (a) Create signing keypair $(\mathsf{vk_J}, \mathsf{sk_J}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ and store it
  (b) Initialize $\mathcal{F}_{\text{AD}}$:
    J $\to \mathcal{F}_{\text{AD}}$: (Init, J, $\mathsf{vk_J}$)
    $\mathcal{F}_{\text{AD}} \to$ J: (InitFinished)
  (c) Get CRS:
    J $\to \mathcal{F}_{\text{CRS}}$: (value)
    $\mathcal{F}_{\text{CRS}} \to$ J: $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$
  (d) Store $\mathsf{crs}_{\text{com}}$

(3) SO:
  (a) Create signing keypair $(\mathsf{vk_{SO}}, \mathsf{sk_{SO}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ and store it
  (b) Get CRS and keys:
    (i) SO $\to \mathcal{F}_{\text{CRS}}$: (value)
      $\mathcal{F}_{\text{CRS}} \to$ SO: $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$
    (ii) SO $\to \mathcal{F}_{\text{AD}}$: (GetPK)
      $\mathcal{F}_{\text{AD}} \to$ SO: (GotPK, pk)
    (iii) SO $\to \mathcal{F}_{\text{AD}}$: (GetSKeys)
      $\mathcal{F}_{\text{AD}} \to$ SO: (GotSKeys, $\mathsf{vk_{SO}}, \mathsf{vk_J}$)
  (c) Store $\mathsf{crs} \coloneqq (\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$, pk and $\mathsf{vk_J}$
  (d) Initialize $\mathcal{F}_{\text{AD}}$:
    SO $\to \mathcal{F}_{\text{AD}}$: (Init, SO, $\mathsf{vk_{SO}}$)
    $\mathcal{F}_{\text{AD}} \to$ SO: (InitFinished)

- Output SO: (InitFinished)
- Output J: (InitFinished)
- Output AU: (InitFinished)

Party Init:

- Input some Party P: (PInit)
- Behavior P:

(1) Get CRS and keys:
  (a) P $\to \mathcal{F}_{\text{CRS}}$: (value)
    $\mathcal{F}_{\text{CRS}} \to$ P: $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$
  (b) P $\to \mathcal{F}_{\text{AD}}$: (GetSKeys)
    $\mathcal{F}_{\text{AD}} \to$ P: (GotSKeys, $\mathsf{vk_{SO}}, \mathsf{vk_J}$)
  (c) P $\to \mathcal{F}_{\text{AD}}$: (GetPK)
    $\mathcal{F}_{\text{AD}} \to$ P: (GotPK, pk)

(2) Store $\mathsf{crs} \coloneqq (\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$ and $(\mathsf{pk}, \mathsf{vk_J}, \mathsf{vk_{SO}})$

- Output to P: (PInitFinished)

User Registration:

- Input U: (Register, *uid*)
- Input SO: (Register, *uid′*)
- Behavior:
  (1) U:
    (a) If no keypair is stored yet, generate keypair $(\mathsf{vk}_{\text{U}}, \mathsf{sk}_{\text{U}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ and store it
    (b) Send (uid) to SO
  (2) SO:
    (a) If $uid \neq uid'$, then abort (Wrong inputs)
    (b) If there is already an entry $(uid, \cdot, \cdot, \cdot)$ in the list of registered users, then abort (User already successfully registered)
    (c) Send (uid_ok) to U
  (3) U:
    (a) Register on bulletin board:
    $\text{U} \rightarrow \mathcal{F}_{\text{BB}}$: (Register, uid, vk_{U})
    (b) Check bulletin board to see if registration was successful:
    $\text{U} \rightarrow \mathcal{F}_{\text{BB}}$: (Retrieve, uid)
    $\mathcal{F}_{\text{BB}} \rightarrow \text{U}$: (Retrieve, ${pid_{\text{U}}}^*$, uid, vk^*_{U})
    (c) If ${pid_{\text{U}}}^* \neq pid_{\text{U}}$ or $\mathsf{vk}^*_{\text{U}} \neq \mathsf{vk}_{\text{U}}$, then abort (UID already taken by someone else)
    (d) Send (Ok) to SO
  (4) SO:
    (a) Check bulletin board to see if registration was successful:
    $\text{SO} \rightarrow \mathcal{F}_{\text{BB}}$: (Retrieve, uid)
    $\mathcal{F}_{\text{BB}} \rightarrow \text{SO}$: (Retrieve, ${pid_{\text{U}}}^*$, uid, vk^*_{U})
    (b) If ${pid_{\text{U}}}^* \neq pid_{\text{U}}$, then abort (UID already taken by another user)
    (c) Set $\mathsf{vk}_{\text{U}} := \mathsf{vk}^*_{\text{U}}$
    (d) Commit-and-sign uid:
    $(com_{\text{uid}}, decom_{\text{uid}}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, uid)$ and $\sigma_{\text{reg}} := \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{SO}}, com_{\text{uid}})$
    (e) Store $(uid, com_{\text{uid}}, decom_{\text{uid}}, \sigma_{\text{reg}}, \mathsf{vk}_{\text{U}})$
    (f) Send $(\sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}})$ to U
  (5) U:
    (a) Get public key of SO:
    $\text{U} \rightarrow \mathcal{F}_{\text{AD}}$: (GetSKeys)
    $\mathcal{F}_{\text{AD}} \rightarrow \text{U}$: (GotSKeys, vk_{SO}, vk_{J})
    (b) Store vk_{SO}
    (c) Get CRS:
    $\text{U} \rightarrow \mathcal{F}_{\text{CRS}}$: (value)
    $\mathcal{F}_{\text{CRS}} \rightarrow \text{U}$: $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\mathsf{com}})$
    (d) Store $\mathsf{crs} := (\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\mathsf{com}})$
    (e) Get public key of TPKE:
    $\text{U} \rightarrow \mathcal{F}_{\text{AD}}$: (GetPK)
    $\mathcal{F}_{\text{AD}} \rightarrow \text{U}$: (GotPK, pk)
    (f) Store pk
    (g) Verify commitment and signature:
    If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{\text{uid}}, decom_{\text{uid}}, uid) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{SO}}, \sigma_{\text{reg}}, com_{\text{uid}}) = 0$, then abort

(h) Store $(uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}})$

- Output U: (Registered)
- Output SO: (Registered)

Store Secret:

- Input U: (StoreSecret, *uid*, *vper*)
- Input SO: (StoreSecret, *vper′*)
- Behavior:

(1) U:

(a) Draw part of the secret: $sec_2 \overset{R}{\leftarrow} \mathcal{S}$

(b) Commit on sec_2: $(com_{sec_2}, decom_{sec_2}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, sec_2)$

(c) Commit on *vper*: $(com_{\mathrm{vper}}, decom_{\mathrm{vper}}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, vper)$

(d) Send $(uid, \sigma_{\mathrm{reg}}, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, com_{sec_2}, com_{\mathrm{vper}}, decom_{\mathrm{vper}})$ to SO

(2) SO:

(a) Verify com_{vper}: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{\mathrm{vper}}, decom_{\mathrm{vper}}, vper') = 0$, then abort

(b) If $(uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}}, \cdot)$ is not in the list of registered users, then abort

(c) If there already exists an entry $(uid, vper, \cdot, \cdot)$ in the list of stored secrets, then abort (User already registered a secret for the current validity period)

(d) Draw part of the secret: $sec_1 \overset{R}{\leftarrow} \mathcal{S}$

(e) Send (sec_1) to U

(3) U:

(a) Compute $secret \coloneqq sec_1 \oplus sec_2$ (*Remark:* $\oplus$ can be XOR or $+$, but it must hold that $secret \in \mathcal{S}$)

(b) Commit on *secret*: $(com_{\mathrm{sec}}, decom_{\mathrm{sec}}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, secret)$

(c) Commit-and-sign sec_1:
$(com_{sec_1}, decom_{sec_1}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, sec_1)$ and $\sigma_{sec_1} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{U}}, com_{sec_1})$

(d) Encrypt part of the secret: $ct \leftarrow \mathsf{TPKE.Enc}(\mathsf{pk}, sec_2; r)$ for fresh randomness r

(e) Sign ciphertext: $\sigma_{\mathrm{U}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{U}}, (ct, uid, vper))$

(f) Assemble ZK-witness: $wit_{\mathrm{ss}} \coloneqq (secret, sec_2, decom_{\mathrm{sec}}, decom_{sec_2}, r)$

(g) Assemble ZK-statement: $stmt_{\mathrm{ss}} \coloneqq (sec_1, com_{\mathrm{sec}}, com_{sec_2}, \mathsf{pk}, ct, \mathsf{crs}_{\mathsf{com}})$

(h) Compute proof: $\pi_{\mathrm{ss}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{\mathrm{AS}}_{\mathsf{zk}}, stmt_{\mathrm{ss}}, wit_{\mathrm{ss}}, \mathcal{R}^{AS}_{ss})$

(i) Send $(ct, com_{\mathrm{sec}}, \pi_{\mathrm{ss}}, \sigma_{\mathrm{U}}, \sigma_{sec_1}, com_{sec_1}, decom_{sec_1})$ to SO

(4) SO:

(a) Assemble ZK-statement: $stmt_{\mathrm{ss}} \coloneqq (sec_1, com_{\mathrm{sec}}, com_{sec_2}, \mathsf{pk}, ct, \mathsf{crs}_{\mathsf{com}})$

(b) Verify proof: If $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AS}}_{\mathsf{zk}}, stmt_{\mathrm{ss}}, \pi_{\mathrm{ss}}, \mathcal{R}^{AS}_{ss}) = 0$, then abort

(c) For *uid*, retrieve entry $(uid, \cdot, \cdot, \cdot, \mathsf{vk}_{\mathrm{U}})$ from list of registered users

(d) Verify commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{sec_1}, decom_{sec_1}, sec_1) = 0$, then abort

(e) Verify signatures: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U}}, (ct, uid, vper), \sigma_{\mathrm{U}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U}}, com_{sec_1}, \sigma_{sec_1}) = 0$, then abort

(f) Sign the commitments: $\sigma_{\mathrm{ss}} \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathrm{SO}}, (com_{\mathrm{uid}}, com_{\mathrm{sec}}, com_{vper}))$

(g) Store $(uid, vper, ct, \sigma_{\mathrm{U}}, sec_1, com_{sec_1}, decom_{sec_1}, \sigma_{sec_1})$

(h) Send (σ_{ss}) to U

(5) U:

(a) Verify signature: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{SO}}, (com_{\mathrm{uid}}, com_{\mathrm{sec}}, com_{\mathrm{vper}}), \sigma_{\mathrm{ss}}) = 0$, then abort

(b) Store $(secret, com_{\mathrm{sec}}, decom_{\mathrm{sec}}, vper, com_{\mathrm{vper}}, decom_{\mathrm{vper}}, \sigma_{\mathrm{ss}})$

- Output U: (SecretStored, *secret*)
- Output SO: (SecretStored, *uid*)

Request Warrant:

- Input LE: (RequestWarrant, W)
- Behavior:
  (1) LE:
    (a) Send message (W) to J
  (2) J:
    (a) Check if policy function allows that warrant:
        If $0 \leftarrow f_p(W)$, then abort (Warrant not allowed by policy function).
- Output J: (RequestWarrant, W)
- Input J: (b)
- Behavior:
  (1) J:
    (a) If $b = 0$, then set $\sigma_W := \bot$. Else, set $\sigma_W \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{J}}, W)$.
    (b) Store (W, b) in list of already requested preliminary warrants
    (c) Send (b, σ_W) to LE
  (2) LE:
    (a) Store $(W, b, \bot, \bot)$ in list of warrants
    (b) If $b = 0$, then skip all other steps and directly go to output
    (c) Get CRS:
        LE $\rightarrow \mathcal{F}_{\text{CRS}}$: (value)
        $\mathcal{F}_{\text{CRS}} \rightarrow$ LE: $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$
    (d) Get public key of J:
        LE $\rightarrow \mathcal{F}_{\text{AD}}$: (GetSKeys)
        $\mathcal{F}_{\text{AD}} \rightarrow$ LE: (GotSKeys, $\mathsf{vk}_{\text{SO}}, \mathsf{vk}_{\text{J}}$)
    (e) Verify signature: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{J}}, W, \sigma_W) = 0$, then abort
    (f) Send (W, σ_W) to SO
  (3) SO:
    (a) Verify signature: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{J}}, W, \sigma_W) = 0$, then abort
    (b) Parse preliminary warrant: $(W_1, \ldots, W_v) := W$ and $(uid_i, vper_i, meta_i) := W_i$
    (c) For each $(uid_i, vper_i, \cdot)$ entry in W:
      (i) Retrieve the matching $(uid_i, vper_i, ct_i, \sigma_{\text{U,i}}, sec_{1,i}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}})$ entry from internal storage. If none exists, then abort (User has not stored a secret).
    (d) Build enhanced warrant $\widetilde{W}$ by adding ct_i to each W_i
    (e) Send $(\widetilde{W}, \{\sigma_{\text{U,i}}, sec_{1,i}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}\}_{i\in\{1,\ldots,v\}})$ to LE
  (4) LE:
    (a) Parse enhanced warrant: $(\widetilde{W_1}, \ldots, \widetilde{W_v}) := \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) := \widetilde{W_i}$
    (b) Build preliminary warrant W' out of $\widetilde{W}$ by deleting ct_i from each $\widetilde{W_i}$
    (c) If $W' \neq W$, then abort (SO sent wrong $\widetilde{W}$)
    (d) For i from 1 to v:
      (i) Get verification key of the ith user:
          LE $\rightarrow \mathcal{F}_{\text{BB}}$: (Retrieve, uid_i)
          $\mathcal{F}_{\text{BB}} \rightarrow$ LE: (Retrieve, $uid_i, \mathsf{vk}_{\text{U,i}}$)
      (ii) Check signatures:
          If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{U,i}}, (ct_i, uid_i, vper_i), \sigma_{\text{U,i}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{U,i}}, com_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}) = 0$, then abort
      (iii) Check commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\text{com}}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, sec_{1,i}) = 0$, then abort
    (e) Send $(\widetilde{W}, \{\sigma_{\text{U,i}}, sec_{1,i}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}\}_{i\in\{1,\ldots,v\}})$ to J

(5) J:
   (a) Build preliminary warrant W'' out of $\widetilde{W}$ by deleting ct_i from each $\widetilde{W_i}$
   (b) If there is no entry $(W'', 1)$ in the list of already processed preliminary warrants, then abort
   (c) Check if $(\widetilde{W}, \cdot)$ is already in list of stored enhanced warrants. If yes, then abort (Warrant already processed).
   (d) Parse enhanced warrant: $(\widetilde{W_1}, \dots, \widetilde{W_v}) \coloneqq \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \coloneqq \widetilde{W_i}$
   (e) For i from 1 to v:
      (i) Get verification key of the ith user:
      $\mathrm{J} \rightarrow \mathcal{F}_{\mathrm{BB}}$: (Retrieve, uid_i)
      $\mathcal{F}_{\mathrm{BB}} \rightarrow \mathrm{J}$: (Retrieve, uid_i, $\mathsf{vk}_{\mathrm{U,i}}$)
      (ii) Check signatures:
      If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, (ct_i, uid_i, vper_i), \sigma_{\mathrm{U,i}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, com_{sec_1,\mathrm{i}}, \sigma_{sec_1,\mathrm{i}}) = 0$, then abort
      (iii) Check commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{sec_1,\mathrm{i}}, decom_{sec_1,\mathrm{i}}, sec_{1,i}) = 0$, then abort
   (f) Sign enhanced warrant: $\sigma_{\widetilde{W}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{J}}, \widetilde{W})$
   (g) Store $(\widetilde{W}, \sigma_{\widetilde{W}})$ in list of stored enhanced warrants
   (h) Send $(\sigma_{\widetilde{W}})$ to LE

(6) LE:
   (a) Update the entry $(W, b, \perp, \perp)$ in the list of warrants to $(W, b, \widetilde{W}, \sigma_{\widetilde{W}})$
   (b) Request decryption of secrets:
   $\mathrm{LE} \rightarrow \mathcal{F}_{\mathrm{AD}}$: (Request, $\widetilde{W}, \sigma_{\widetilde{W}}$)
   $\mathcal{F}_{\mathrm{AD}} \rightarrow \mathrm{LE}$: (Request)
   (c) Store each $(uid_i, vper_i, ct_i, sec_{1,i})$ pair for $i \in \{1, \dots, v\}$

- Output LE: (RequestWarrant, b)

Get Secrets:

- Input LE: (GetSecrets, W)
- Behavior LE:

(1) Retrieve the entry $(W, 1, \widetilde{W}, \sigma_{\widetilde{W}})$ from the list of warrants. If none exists, then abort (Warrant not requested or not granted)
(2) Retrieve all $(uid_i, vper_i, ct_i)$ tuples from $\widetilde{W}$
(3) For all $(uid_i, vper_i, ct_i)$, retrieve the corresponding entry $(uid_i, vper_i, ct_i, \sigma_{\mathrm{U,i}}, sec_{1,i}, com_{sec_{1,i}}, decom_{sec_{1,i}}, \sigma_{sec_1,\mathrm{i}})$ from the list of stored partial secrets
(4) Retrieve (partial) secrets:
   $\mathrm{LE} \rightarrow \mathcal{F}_{\mathrm{AD}}$: (Retrieve)
   $\mathcal{F}_{\mathrm{AD}} \rightarrow \mathrm{LE}$: (Retrieve, $\mathsf{L}^{\mathsf{Ready}}_{\mathsf{Requests}}$)
(5) For each $(uid_i, vper_i, ct_i, sec_{1,i})$:
   (a) Search for an entry $(ct_j, sec_{2,j})$ in $\mathsf{L}^{\mathsf{Ready}}_{\mathsf{Requests}}$ where $ct_i = ct_j$
   (b) If one exists, then set $secret_i \coloneqq sec_{1,i} \oplus sec_{2,j}$
   (c) If none exists, then set $secret_i \coloneqq \perp$

- Output LE: (GotSecrets, $(secret_1, \dots, secret_v)$)

Get Statistics:

- Input some Party P: (GetStatistics)
- Behavior P:

(1) $\mathrm{P} \rightarrow \mathcal{F}_{\mathrm{AD}}$: (GetStatistics)
   $\mathcal{F}_{\mathrm{AD}} \rightarrow \mathrm{P}$: (GotStatistics, $\mathsf{L}_{\mathsf{Stats}}$)

- Output to P: (GotStatistics, $\mathsf{L}_{\mathsf{Stats}}$)

Audit:

- Input AU: (AuditRequest)
- Behavior AU:
  (1) If the party invoking this task has *not* the pid pid_{AU}, then abort
  (2) Get list of warrants (including ciphertexts):
  AU $\rightarrow \mathcal{F}_{\mathrm{AD}}$: (AuditRequest)
  $\mathcal{F}_{\mathrm{AD}} \rightarrow$ AU: (AuditAnswer, $\mathsf{L}_{\mathsf{Warrants}}$)
  (3) Build L_{AU} out of $\mathsf{L}_{\mathsf{Warrants}}$ by deleting ct_i from each W_i
- Output AU: (AuditAnswer, L_{AU})

Prove:

- Input U: (Prove, $stmt_R$, wit_R)
- Behavior U:
  (1) Parse $(uid, secret, wit') \coloneqq wit_R$ and $(vper, stmt') \coloneqq stmt_R$
  (2) If $\mathcal{R}(stmt_R, wit_R) \neq 1$, then abort
  (3) For uid, if there is no registration data $(uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \cdot)$ stored, abort (User not registered)
  (4) For $(secret, vper)$, if there is no entry $(secret, com_{\mathrm{sec}}, decom_{\mathrm{sec}}, vper, com_{\mathrm{vper}}, decom_{\mathrm{vper}}, \sigma_{\mathrm{ss}})$ in the internal list of encrypted secrets, abort (User has not encrypted a secret for the period)
  (5) Assemble witness: $wit \coloneqq (uid, secret, wit', \sigma_{\mathrm{ss}}, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, com_{\mathrm{sec}}, decom_{\mathrm{sec}}, com_{\mathrm{vper}}, decom_{\mathrm{vper}})$
  (6) Assemble statement: $stmt \coloneqq (vper, stmt', \mathsf{vk}_{\mathrm{SO}}, \mathsf{crs}_{\mathrm{com}})$
  (7) Compute proof: $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, stmt, wit, \mathcal{R}_{zk}^{AS})$
- Output U: (Proof, $stmt_R$, wit_R, π)

Verify:

- Input some party P: (Verify, $stmt_R$, π)
- Behavior P:
  (1) If the values $\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{crs}_{\mathrm{com}}, \mathsf{vk}_{\mathrm{SO}}$ are not stored internally, then abort
  (2) Parse $(vper, stmt') \coloneqq stmt_R$
  (3) Assemble statement: $stmt \coloneqq (vper, stmt', \mathsf{vk}_{\mathrm{SO}}, \mathsf{crs}_{\mathrm{com}})$
  (4) Verify proof: $b \leftarrow \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, stmt, \pi, \mathcal{R}_{zk}^{AS})$
- Output to P: (Verification, $stmt_R$, π, b)

Security. To prove the security of the just described protocol Π_{AS}, we show that it is "indistinguishable" from the ideal functionality $\mathcal{F}_{\mathrm{AS}}$. More precisely, we show that Π_{AS} UC-realizes $\mathcal{F}_{\mathrm{AS}}$, which is captured by the following theorem:

Theorem 4.3. Π_{AS} *UC-realizes* $\mathcal{F}_{\mathrm{AS}}$ *in the* $\{\mathcal{F}_{\mathrm{AD}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BB}}, \mathcal{G}_{\mathrm{CLOCK}}\}$*-hybrid model under the assumptions that*

- COM *is a (computationally) hiding, (statistically) binding and (dual-mode) extractable and equivocable commitment scheme,*
- Σ *is a* EUF-CMA*-secure signature scheme,*
- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- *and* TPKE *is* IND-CPA*-secure*

against all PPT*-adversaries* $\mathcal{A}$ *who statically corrupt either*

(1) a subset of the users,

(2) LE and a subset of the users,

(3) SO and a subset of the users, or

(4) SO, LE and a subset of the users.

The proof of this theorem is given in Appendix B.1.

4.5. $\mathcal{F}_{\mathrm{AD}}$: A Formal Model for Auditable Decryption Systems

We now want to describe our ideal auditable decryption functionality $\mathcal{F}_{\mathrm{AD}}$, which is used to outsource the auditable decryption of ciphertexts in Π_{AS}.

To enable protocols realizing this functionality based on the YOSO approach (cf. Section 4.6), our functionality makes use of the global clock functionality $\mathcal{G}_{\mathrm{CLOCK}}$ (cf. Fig. 2.5) and proceeds in rounds, where a round lasts a predefined amount of time units and decryptions only become available in following rounds. The functionality is also parameterized by a (threshold) PKE scheme TPKE and a signature scheme Σ. To ensure compatibility with Π_{AS}, the schemes TPKE and Σ used in $\mathcal{F}_{\mathrm{AD}}$ must be the same as in Π_{AS}.

We now describe how our auditable decryption system is modeled in the UC framework by presenting the ideal functionality $\mathcal{F}_{\mathrm{AS}}$ in Figs. 4.6 and 4.7, as well as sketching the key points of the individual tasks. Note that we again use the UC writing conventions introduced in Remark 4.1.

Init, Get Tasks and Interaction with $\mathcal{G}_{\mathrm{CLOCK}}$. $\mathcal{F}_{\mathrm{AD}}$ is initialized by creating an encryption keypair and registering with the global clock $\mathcal{G}_{\mathrm{CLOCK}}$. In Π_{AS}, SO and J each create a signing keypair and pass their public verification keys to $\mathcal{F}_{\mathrm{AD}}$, where they are stored. There are also tasks to provide the parties in Π_{AS} with the public keys of SO, J and the public encryption key of the functionality. This ensures that both systems use the same keys.

Request Decryption and Retrieve Secret. LE can send a signed warrant to $\mathcal{F}_{\mathrm{AD}}$ to request decryption of all ciphertexts *ct* listed in that warrant. $\mathcal{F}_{\mathrm{AD}}$ checks if that warrant is valid and then adds the listed ciphertexts *ct* to a list of pending decryption requests. To allow our implementation Π_{AD} to be YOSO, requests for decryption are only processed during committee handovers. To emulate that behavior in the ideal world, $\mathcal{F}_{\mathrm{AD}}$ separates the *request* of a secret and the actual *retrieval* of a secret into two different inquiries by LE, with the requirement that the committee switches between the two calls.

Role Execute. To emulate the passing of time in the real world and the fact that YOSO parties can only send a message once, $\mathcal{F}_{\mathrm{AD}}$ interacts with the clock $\mathcal{G}_{\mathrm{CLOCK}}$: After all honest nodes have activated $\mathcal{F}_{\mathrm{AD}}$, it advances the current time. After the required amount of “time” passes, $\mathcal{F}_{\mathrm{AD}}$ emulates a committee handover as follows: $\mathcal{F}_{\mathrm{AD}}$ handles all *pending* decryption requests and adds the decrypted (partial) secrets to a list of *processed* decryption requests. After this “committee handover”, the list of processed decryption requests contains all (partial) secrets that are ready for retrieval by LE.

Functionality $\mathcal{F}_{\text{AD}}$

System Parameters:
- t_{com} — Number of clock ticks between committee handovers
- f_t — Transparency function. Gets a warrant W as input and outputs what should be publicly known about that warrant. Interface is $W^{\text{pub}} \leftarrow f_t(W)$.
- Signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, where all are PPT and Vfy is deterministic
- Re-randomizable Threshold Public Key Encryption scheme $\mathsf{TPKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TDec}, \mathsf{ShareVer}, \mathsf{Combine}, \mathsf{Rand}, \mathsf{Sk2Pk})$, where all are PPT and Dec, TDec, ShareVer, Combine and Sk2Pk are deterministic
- System pids: pid_{SO}, pid_{J}, pid_{AU}

The functionality has access to a global clock $\mathcal{G}_{\text{CLOCK}}$.

Functionality State:
- t_{last}: Time of last committee handover
- $(\mathsf{pk}, \mathsf{sk})$: Keypair for TPKE
- $\mathsf{L}_{\text{Warrants}}$: List of warrants $\widetilde{W}$ on the blockchain
- $\mathsf{L}_{\text{Requests}}$: List of requests for decryption not yet ready for retrieval
- $\mathsf{L}^{\text{Ready}}_{\text{Requests}}$: Lists of requests for decryption ready to be retrieved
- vk_{SO}: Public signing key of SO (This key is not needed for this functionality, it only exists so parties in Π_{AS} can get it)
- vk_{J}: Public signing key of J (Used to verify warrant signatures and so parties in Π_{AS} can get it)
- $\{d_{pid}\}$, for $pid \in \{N_i\}$: Flag whether the node pid has called ExecuteRole since the last clock tick

Init
Note: For each party only execute this task once. Ignore messages other than (Init) until SO, J and AU each have called this task.
- Input SO: (Init, SO, vk_{SO})
- Behavior:
  1. Send (Register) to $\mathcal{G}_{\text{CLOCK}}$
  2. Set $cnum \coloneqq 0$
  3. Generate encryption key pair: $(\mathsf{pk}, \mathsf{sk}, \{\mathsf{vk}_i\}, \{\mathsf{sk}_i\}) \leftarrow \mathsf{TPKE.Gen}(1^\lambda)$ and store $(\mathsf{pk}, \mathsf{sk})$
  4. Set $t_{\text{last}} \coloneqq 0$
  5. Send (Init, SO, vk_{SO}, pk) to the adversary
  6. Store vk_{SO}
- Output SO: (InitFinished)

- Input J: (Init, J, vk_{J})
- Behavior:
  1. Send (Init, J, vk_{J}) to the adversary
  2. Store vk_{J}
- Output J: (InitFinished)

- Input AU: (Init, AU)
- Behavior:
  1. Send (Init, AU) to the adversary
- Output AU: (InitFinished)

Get Public Key:
- Input some Party P: (GetPK)
- Behavior:
  1. Send (GetPK) to the adversary
  2. Retrieve $(\mathsf{pk}, \mathsf{sk})$
- Output P: (GotPK, pk)

Get System Keys:
- Input some Party P: (GetSKeys)
- Behavior:
  1. Send (GetSKeys) to the adversary
  2. Retrieve vk_{SO} and vk_{J}
- Output P: (GotSKeys, vk_{SO}, vk_{J})

Request Decryption:
- Input LE: (Request, $\widetilde{W}$, $\sigma_{\widetilde{W}}$)
- Behavior:
  1. Check warrant signature: $b \coloneqq \Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{J}}, \widetilde{W}, \sigma_{\widetilde{W}})$
  2. If $b = 0$, then abort (Warrant not valid)
  3. Parse warrant: $\left(\widetilde{W_1}, \ldots, \widetilde{W_v}\right) \coloneqq \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \coloneqq \widetilde{W_i}$
  4. For each i, store ct_i in list of requests $\mathsf{L}_{\text{Requests}}$
  5. Send (Request, pid, $f_t(W)$, $\left|\widetilde{W}\right|$, v) to the adversary, where pid is the id of the calling party
  6. When the adversary allows to deliver output, store $\widetilde{W}$ in the list of warrants $\mathsf{L}_{\text{Warrants}}$
- Output LE: (Request)

Retrieve Secret:
- Input LE: (Retrieve)
- Behavior:
  1. If LE is honest: Send (Retrieve) to the adversary
  2. If LE is corrupted: Send (Retrieve, $\mathsf{L}^{\text{Pending}}_{\text{Requests}}$) to the adversary
- Output LE: (Retrieve, $\mathsf{L}^{\text{Ready}}_{\text{Requests}}$)

Figure 4.6.: The ideal functionality $\mathcal{F}_{\text{AD}}$

Functionality $\mathcal{F}_{\text{AD}}$ (continued)

Get Statistics:
- Input some Party P: (GetStatistics)
- Behavior:
  (1) Send (GetStatistics) to the adversary
  (2) Initialize empty list $\mathsf{L}_{\mathsf{Stats}}$
  (3) Retrieve list of warrants $\mathsf{L}_{\mathsf{Warrants}}$, entries are of the form $\widetilde{W}$
  (4) For each $\widetilde{W} \in \mathsf{L}_{\mathsf{Warrants}}$: Build W from $\widetilde{W}$, calculate $W^{\text{pub}} \leftarrow f_t(W)$ and append W^{pub} to $\mathsf{L}_{\mathsf{Stats}}$
- Output P (immediately): (GotStatistics, $\mathsf{L}_{\mathsf{Stats}}$)

Audit:
- Input AU: (AuditRequest)
- Behavior:
  (1) If the party invoking this task has *not* the pid pid_{AU}, then abort
  (2) Send (AuditRequest) to the adversary
  (3) Retrieve list of warrants $\mathsf{L}_{\mathsf{Warrants}}$
- Output AU (immediately): (AuditAnswer, $\mathsf{L}_{\mathsf{Warrants}}$)

RoleExecute:
- Input N: (ExecuteRole)
- Behavior:
  (1) Send (ExecuteRole, pid_N) to the adversary
  (2) Set $d_N \coloneqq 1$
  (3) If for all honest nodes N_i it holds that $d_{N_i} = 1$, then execute ClockTick
- Output N: (ExecuteRole)

Handling corruptions:
- Upon receiving a message (corrupt, pid) for $pid \in \{N_i\}$:
  (1) Mark N_i as corrupted
  (2) If as a result of this operation for all remaining honest nodes N_i it holds that $d_{N_i} = 1$, then execute ClockTick

ClockTick:
  (1) Reset $d_{pid} \coloneqq 0$ for all nodes
  (2) Send (Clock-Update, sid) to $\mathcal{G}_{\text{CLOCK}}$
  (3) Send (Clock-Read, sid) to $\mathcal{G}_{\text{CLOCK}}$ and receive (Clock-Read, sid, t_{Now})
  (4) If $t_{\text{last}} + t_{\text{com}} \leq t_{\text{Now}}$, then execute Handover

Handover:
  (1) Set $\mathsf{L}^{\mathsf{Ready}}_{\mathsf{Requests}} \coloneqq \mathsf{L}^{\mathsf{Ready}}_{\mathsf{Requests}} \cup \mathsf{L}^{\mathsf{Pending}}_{\mathsf{Requests}}$
  (2) Clear $\mathsf{L}^{\mathsf{Pending}}_{\mathsf{Requests}}$
  (3) For each entry $ct_i \in \mathsf{L}_{\mathsf{Requests}}$:
    (a) $secret_i \coloneqq \mathsf{TPKE.Dec}(\mathsf{sk}, ct_i)$
    (b) Add $(ct_i, secret_i)$ to $\mathsf{L}^{\mathsf{Pending}}_{\mathsf{Requests}}$
  (4) Clear $\mathsf{L}_{\mathsf{Requests}}$
  (5) Set time of last handover: $t_{\text{last}} \coloneqq t_{\text{Now}}$

Figure 4.7.: The ideal functionality $\mathcal{F}_{\text{AD}}$ (continued)

Get Statistics and Audit. $\mathcal{F}_{\text{AD}}$ keeps track of all warrants for which LE requested secrets. If AU initiates an investigation, $\mathcal{F}_{\text{AD}}$ provides it with all valid warrants. Likewise, $\mathcal{F}_{\text{AD}}$ can also provide a party asking for statistics with the outputs of the transparency function for all warrants. Therefore, $\mathcal{F}_{\text{AD}}$ provides the same statistics and audit information as $\mathcal{F}_{\text{AS}}$.

4.6. Π_{AD}: A Protocol to Realize $\mathcal{F}_{\text{AD}}$

We cast our protocol Π_{AD} in the YOSO model, which we now introduce briefly.

The YOSO Model. In the YOSO (You-Only-Speak-Once) model introduced by [Gen+21], protocols are run between roles, where each role is only allowed to send one message and has no lasting state. These roles can then be assigned to actual machines executing them through some form of role assignment mechanism. With a way to anonymously receive messages (e.g., by reading ciphertexts stored on a public blockchain) and a role assignment mechanism that privately assigns roles, this prevents targeted attack against roles: The identity of a machine executing a role can only be learned when it sends its message, but at that point it finished execution and is no longer in possession of any secret state.

Functionality $\mathcal{F}_{\text{BCRA}}$

The functionality is parameterized by an encryption scheme PKE, a signature scheme Σ, a threshold ϵ, a maximum delay δ_{Post} and a set MACHINE which is the set of parties allowed to use it. It has also access to a global clock $\mathcal{G}_{\text{CLOCK}}$. Upon receiving any input, $\mathcal{F}_{\text{BCRA}}$ first queries $\mathcal{G}_{\text{CLOCK}}$ and sets t_{Now} to the value returned.

Init: Let POSTED be the empty set, let ORDERED be the empty sequence.

POST: On input (POST, m) from $pid \in \text{MACHINE}$, add (pid, t_{Now}, m) to the set POSTED and output (POST, pid, m) to the adversary.

ORDER: On input (ORDER, pid, m) from the adversary where some $(pid, t, m) \in \text{POSTED}$ and no $(pid, t', m) \in \text{ORDERED}$, append (pid, t_{Now}, m) to ORDERED.

READ: On input (READ) from $pid \in \text{MACHINE}$, leak (READ) to the adversary. For each entry $(pid, t, m) \in \text{POSTED}$ for which no $(pid, t', m) \in \text{ORDERED}$ and $t = t_{\text{Now}} - \delta_{\text{Post}}$, append (pid, t_{Now}, m) to ORDERED. Then output (ORDERED) to pid.

NEXTCOMMITTEE: On input $(\text{NEXTCOMMITTEE}, R, n, S)$ from the adversary, with $|S| < \epsilon n$ containing entries of the form $(M_i \in \text{MACHINE}, (\text{ek}_i, \text{vk}_i))$, do the following:

- For $i \in (1, \ldots, |S|)$:
  (1) Mark M_i as corrupted
  (2) Set $m := (\text{NEXTCOMMITTEE}, R, \text{ek}_i, \text{vk}_i)$, add $(\text{roleassign}, t_{\text{Now}}, m)$[a] to POSTED and leak m to the adversary
- For $i \in (|S| + 1, \ldots, n)$:
  (1) Sample $(\text{ek}_{R_i}, \text{dk}_{R_i}) \leftarrow \text{PKE.Gen}(1^\lambda)$
  (2) Sample $(\text{vk}_{R_i}, \text{sk}_{R_i}) \leftarrow \Sigma.\text{Gen}(1^\lambda)$
  (3) Sample a uniformly random $M_{R_i} \in \text{MACHINE}$
  (4) Set $m := (\text{NEXTCOMMITTEE}, R, \text{ek}_{R_i}, \text{vk}_{R_i})$, add $(\text{roleassign}, t_{\text{Now}}, m)$ to POSTED and leak m to the adversary

When $(\text{roleassign}, t', m)$ is later added to ORDERED, output $(\text{GENERATE}, R, \text{ek}_{R_i}, \text{dk}_{R_i}, \text{vk}_{R_i}, \text{sk}_{R_i})$ to M_{R_i} for $i \in (|S| + 1, \ldots, n)$

Forward Security: When M_R becomes corrupted, output $(\text{GENERATE}, R, \text{ek}_R, \text{dk}_R, \text{vk}_R, \text{sk}_R)$ to the adversary if $(\text{NEXTCOMMITTEE}, R, \text{ek}_R, \text{vk}_R) \notin \text{ORDERED}$.

[a] roleassign is a special pid used to represent role-assignment messages

Figure 4.8.: Blockchain with role assignment functionality, loosely based on [Gen+21]

This allows both resilience against denial-of-service attacks as well as against strong adversaries trying to corrupt roles that are part of a protocol execution of interest. Assuming a large enough pool of machines willing to execute roles, an attacker able to corrupt any machine of its choosing, but limited in the number of machines it can corrupt at once, cannot break the security of a protocol even when run between only a number of roles smaller than the corruption limit.

To achieve this in our protocol, we make use of a *blockchain with role assignment functionality* $\mathcal{F}_{\text{BCRA}}$, which provides a public append-only ledger together with a mechanism that anonymously selects parties for the next committee by posting public encryption and verification keys for the individual roles on the ledger and privately sending the corresponding decryption and signing keys to the assigned party. For more details, see Fig. 4.8. Note that since it is unclear how to realize the role assignment functionality provided in [Gen+21], our functionality differs from theirs in the following ways:

(1) We integrated the global clock $\mathcal{G}_{\text{CLOCK}}$.

(2) $\mathcal{F}_{\text{BCRA}}$ only allows assigning roles of the next committee, not roles at an arbitrary time in the future.

(3) The adversary can control a portion of the public keys (and $\mathcal{F}_{\text{BCRA}}$ does not get to know the corresponding secret keys).

We deem it plausible that the committee selection protocol from [Ben+20] with suitable corruption thresholds in combination with a suitable blockchain can be used to implement this,[13] although more efficient approaches such as described in [Cam+22] are also possible. We want to stress here that we are in the "near future" setting of [Cam+22], compared to the "far future" setting that would imply witness encryption. Given that the mechanisms for assigning roles and the criteria by which parties should be chosen are subject to ongoing research, we believe using $\mathcal{F}_{\text{BCRA}}$ to abstract from the details is a suitable approach that allows incorporation of future research.

We now first elaborate on some of the core techniques we use in Π_{AD} and give the full description of Π_{AD} afterwards. Our instantiation Π_{AD} UC-realizes $\mathcal{F}_{\text{AD}}$ in the $\{\mathcal{F}_{\text{BCRA}}, \mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{CLOCK}}\}$-hybrid model. We use $\mathcal{F}_{\text{CRS}}$ (cf. Fig. 2.2) to set up the CRS for the NIZK and to distribute it to all parties. To achieve our YOSO modeling we need some form of (generalized) time, which is why the protocol used the global clock functionality $\mathcal{G}_{\text{CLOCK}}$ (cf. Fig. 2.5) to model rounds/epochs.

Init, Get Tasks and Interaction with $\mathcal{G}_{\text{CLOCK}}$. During initialization, AU creates an encryption keypair and SO, J and AU post their public keys to the ledger $\mathcal{F}_{\text{BCRA}}$. The common reference string is obtained by querying $\mathcal{F}_{\text{CRS}}$, the other Get Tasks are handled by reading from the ledger $\mathcal{F}_{\text{BCRA}}$. All honest nodes N register with the clock $\mathcal{G}_{\text{CLOCK}}$ upon first activation. Additionally, the distributed key generation protocol from [EFR21] is run by the first roles assigned through $\mathcal{F}_{\text{BCRA}}$, resulting in a public encryption key pk and a threshold-sharing of the corresponding decryption key among the first committee.

Request Decryption and Retrieve Secret. To request decryption of ciphertexts and subsequently receive user secrets, LE needs to be in possession of a judge-signed warrant listing the relevant ciphertexts ct. To ensure privacy of the warrant $\widetilde{W}$, instead of simply posting the warrant to $\mathcal{F}_{\text{BCRA}}$, LE instead posts an encryption W^{enc} of the warrant $\widetilde{W}$ under AU's public key, the output W^{pub} of the transparency function and a NIZK-proof that it knows a valid signature under J's public key on $\widetilde{W}$ and both W^{enc} and W^{pub} were computed correctly. Additionally, instead of posting the ciphertexts ct directly, LE re-randomizes them and also proves in zero-knowledge that the ciphertexts $\widehat{ct}$ are indeed re-randomizations of the ciphertexts ct listed in the warrant. This ensures that the users under surveillance can not be identified from the ciphertexts. The request additionally contains a public encryption key of LE under which the responses from the committee members will be encrypted.

After the responses have been posted, LE again reads the content of $\mathcal{F}_{\text{BCRA}}$, decrypts all responses using its decryption key, and combines the partial decryptions ct^* to obtain the secret for each ciphertext ct.

[13] Alternatively, a suitable variant of the committee selection protocol from [Ben+20] or the "encryption to the current winner" scheme from [Cam+22] are good candidates as well.

Role Execute. Nodes N read the content of $\mathcal{F}_{\mathrm{BCRA}}$ at least once per round (this is ensured by them only sending an update-message to $\mathcal{G}_{\mathrm{CLOCK}}$ after having done so). Afterwards, they check if they were assigned a role in the current round. If this is the case, they proceed as follows by parsing the content of the ledger:

- They gather all required encryption/verification keys for relevant previous and the next committee
- They gather all messages with key shares of the threshold decryption key
- They gather all requests for decryption

After gathering all relevant messages, they fulfill their role as committee member of the current round. For all messages gathered from the ledger, they validate the signature and accompanying proofs and ignore the message if they are invalid.

They gather all resharings of the threshold decryption key that were made by the previous committee and addressed to the current role and combine them to obtain their share sk_i of the threshold decryption key sk. To enable the committee in the next round to fulfill their duties as well, they reshare sk_i again and encrypt each reshare to a committee member from the next round. This is again done in the same way as in [EFR21].

For each valid decryption request, a (partial) threshold-decryption of the ciphertext $\widehat{ct}$ is performed using sk_i. The answer (including the partial decryption ct^* and a proof of correct decryption) is encrypted under the public key of LE contained in the request.

All messages to be sent[14] are signed using the role's signing key. Before sending any message, all state except for the prepared messages is deleted. Finally, all messages are posted to $\mathcal{F}_{\mathrm{BCRA}}$.

Get Statistics and Audit. Obtaining statistics is achieved by reading the content of $\mathcal{F}_{\mathrm{BCRA}}$ and gathering all W^{pub} accompanied by valid NIZK-proofs. Similarly, the audit is performed by AU reading the content of $\mathcal{F}_{\mathrm{BCRA}}$ and decrypting all W^{enc} accompanied by valid NIZK-proofs.

Before delving into the full details of the protocol, we introduce the NP-relations needed in Π_{AD} for the zero-knowledge proofs:

- Relations $\mathcal{R}^{AD}_{KG1}$ and $\mathcal{R}^{AD}_{KG2}$ (see Fig. 4.9) are used during initial key generation to ensure the resulting keypair is generated properly
- Relation $\mathcal{R}^{AD}_{KS}$ (see Fig. 4.10) is used during each committee handover to ensure correct resharing of the secret key
- Relation $\mathcal{R}^{AD}_{W}$ (see Fig. 4.11) is used in Request Decryption by law enforcement to prove that they possess a warrant signed by the judge and correctly evaluated the transparency function
- Relation $\mathcal{R}^{AD}_{Dec}$ (see Fig. 4.12) is used when answering decryption requests to prove correct partial decryption

[14] These include messages to the next committee and decryption answers to LE

Details of the relation $\mathcal{R}^{AD}_{KG1}$

$wit \coloneqq \left(\mathsf{sk}_i, \{r_j\}_{j\in[n]}, F\right)$

$stmt \coloneqq \left(\mathsf{pk}_i, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}\right)$

$\mathcal{R}^{AD}_{KG1} \coloneqq \{(stmt, wit) \mid$

- F is a degree t polynomial
- For each $j \in [n]$:
  - $ct_j = \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, F(j); r_j)$

}

Details of the relation $\mathcal{R}^{AD}_{KG2}$

$wit \coloneqq \left(\mathsf{sk}_i, \{r_j\}_{j\in[n]}, \mathsf{dk}_R, F\right)$

$stmt \coloneqq \left(\mathsf{pk}_i, \{ct^k\}_{k\in[t+1]}, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}, \mathsf{ek}_R\right)$

$\mathcal{R}^{AD}_{KG2} \coloneqq \{(stmt, wit) \mid$

- $\mathsf{PKE.Sk2Pk}(\mathsf{dk}_R) = \mathsf{ek}_R$
- $\mathsf{sk}_i = \sum_{k=1}^{t+1} \mathsf{PKE.Dec}(\mathsf{dk}_R, ct^k)$
- $\mathsf{TPKE.Sk2Pk}(sk_i) = \mathsf{pk}_i$
- F is a degree t polynomial with $F(0) = \mathsf{sk}_i$
- For each $j \in [n]$:
  - $ct_j = \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, F(j); r_j)$

}

Figure 4.9.: ZK-relations $\mathcal{R}^{AD}_{KG1}$ and $\mathcal{R}^{AD}_{KG2}$

Details of the relation $\mathcal{R}^{AD}_{KS}$

$wit \coloneqq \left(\mathsf{dk}_R, \mathsf{sk}_i, \{r_j\}_{j\in[n]}, F, G\right)$

$stmt \coloneqq \left(\mathsf{ek}_R, \{ct^k\}_{k\in[t+1]}, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}, \mathsf{pk}_i\right)$

$\mathcal{R}^{AD}_{KS} \coloneqq \{(stmt, wit) \mid$

- $\mathsf{PKE.Sk2Pk}(\mathsf{dk}_R) = \mathsf{ek}_R$
- $\mathsf{TPKE.Sk2Pk}(\mathsf{sk}_i) = \mathsf{pk}_i$
- G and F are both degree t polynomials
- $G(i) = F(0) = \mathsf{sk}_i$
- For each $k \in [t+1]$:
  - $sh_k = \mathsf{PKE.Dec}(\mathsf{dk}_R, ct^k)$
- G is obtained by interpolating $sh_1, \ldots, sh_{t+1}$
- For each $j \in [n]$:
  - $ct_j = \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, G(j); r_j)$

}

Figure 4.10.: ZK-relation $\mathcal{R}^{AD}_{KS}$

Details of the relation $\mathcal{R}^{AD}_{W}$

$wit := \left(\widetilde{W}, r, \sigma_{\widetilde{W}}, \{ct_i, r_i\}_{i\in[v]}\right)$
$stmt := \left(\mathsf{ek}_{\text{AU}}, \mathsf{vk}_{\text{J}}, W^{\text{enc}}, W^{\text{pub}}, \{\widehat{ct}_i\}\right)$

$\mathcal{R}^{AD}_{W} := \{(stmt, wit) \mid$

- $W^{\text{enc}} = \mathsf{PKE.Enc}(\mathsf{ek}_{\text{AU}}, \widetilde{W}; r)$
- $W^{\text{pub}} = f_t(W)$
- $\mathsf{Vfy}(\mathsf{vk}_{\text{J}}, \widetilde{W}, \sigma_{\widetilde{W}})$
- W is $\widetilde{W}$ but without the ciphertexts
- For each $i \in [v]$:
  - $\widehat{ct}_i = \mathsf{TPKE.Rand}(ct_i; r_i)$

$\}$

Figure 4.11.: ZK-relation $\mathcal{R}^{AD}_{W}$

Details of the relation $\mathcal{R}^{AD}_{Dec}$

$wit := (\mathsf{sk}_i)$
$stmt := (\mathsf{vk}_i, ct, ct^*)$

$\mathcal{R}^{AD}_{Dec} := \{(stmt, wit) \mid$

- $\mathsf{TPKE.Sk2Pk}(\mathsf{sk}_i) = \mathsf{vk}_i$
- $ct^* = \mathsf{TPKE.TDec}(\mathsf{sk}_i, ct)$

$\}$

Figure 4.12.: ZK-relation $\mathcal{R}^{AD}_{Dec}$

We now give the complete description of the protocol Π_{AD} that UC-realizes the auditable decryption functionality $\mathcal{F}_{\text{AD}}$.

Protocol Π_{AD}

System Parameters:

- n: Number of committee members
- t_{com}: Number of clock ticks between committee handovers
- Transparency function f_t. Gets a preliminary warrant W as input and outputs what should be publicly known about that warrant. Interface is $W^{\text{pub}} \leftarrow f_t(W)$.
- Signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, where all algorithms are PPT and Vfy is deterministic
- Re-randomizable Threshold Public Key Encryption scheme $\mathsf{TPKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{TDec}, \mathsf{ShareVer}, \mathsf{Combine}, \mathsf{Rand}, \mathsf{Sk2Pk})$, where all algorithms are PPT and Dec, TDec, ShareVer, Combine and Sk2Pk are deterministic
- Public Key Encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where all are PPT and Dec is deterministic
- Non-interactive Zero-Knowledge scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where all are PPT and Verify is deterministic
- ZK-Relations $\mathcal{R}^{AD}_{W}$ (see Fig. 4.11), $\mathcal{R}^{AD}_{KG1}$, $\mathcal{R}^{AD}_{KG2}$ (see Fig. 4.9), $\mathcal{R}^{AD}_{KS}$ (see Fig. 4.10) and $\mathcal{R}^{AD}_{Dec}$ (see Fig. 4.12)
- System pids: pid_{SO}, pid_{J}, pid_{AU}, pid_{LE}

State of the Parties:

- LE stores:
  - Own encryption keypair $(\mathsf{ek}_{\mathrm{LE}}, \mathsf{sk}_{\mathrm{LE}})$
- AU stores:
  - Own encryption keypair $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}})$

System Setup ($\mathcal{F}_{\mathrm{CRS}}$):

(1) $\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AD}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
(2) Return $\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AD}}$

Init:

- Note:
  - Each of SO, J, AU only executes the following the first time this input is received and ignores all further inputs of this form.
  - Each of SO, J, AU ignores all other inputs and messages until this has been executed.

- Input SO: $(\textsc{Init}, \mathrm{SO}, \mathsf{vk}_{\mathrm{SO}})$
- Behavior SO:
(1) Send $(\textsc{Post}, (\textsc{OperatorKey}, \mathsf{vk}_{\mathrm{SO}}))$ to $\mathcal{F}_{\mathrm{BCRA}}$
- Output SO: $(\textsc{InitFinished})$

- Input J: $(\textsc{Init}, \mathrm{J}, \mathsf{vk}_{\mathrm{J}})$
- Behavior J:
(1) Send $(\textsc{Post}, (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}}))$ to $\mathcal{F}_{\mathrm{BCRA}}$
- Output J: $(\textsc{InitFinished})$

- Input AU: $(\textsc{Init}, \mathrm{AU})$
- Behavior AU:
(1) Generate $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$
(2) Send $(\textsc{Post}, (\textsc{AuditorKey}, \mathsf{ek}_{\mathrm{AU}}))$ to $\mathcal{F}_{\mathrm{BCRA}}$
(3) Store $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}})$
- Output AU: $(\textsc{InitFinished})$

Get Public Key:

- Input some Party P: $(\textsc{GetPK})$
- Behavior P:
(1) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$
(2) For each role R_i in the first key generation committee do the following:
  (a) Find $(\mathrm{roleassign}, t, (\textsc{Generate}, R_i, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$ and retrieve vk_R
  (b) Find (R, msg, σ) with $msg = (\textsc{KeyGen}1, (\{ct_j\}_{j\in[n]}, \pi_{KG}))$ in $\textsc{Ordered}$
  (c) Verify that $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, msg, \sigma) = 1$, otherwise ignore this role
  (d) Assemble ZK-statement: $stmt_{KG} := (\mathsf{pk}_i, \{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]})$
  (e) Verify that $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AD}}, stmt_{KG}, \pi_{KG}, \mathcal{R}_{KG1}^{AD}) = 1$, otherwise ignore this role
  (f) Add $\{ct_j\}_{j\in[n]}$ to $\mathsf{L}_{\mathsf{Qual}}^{ct}$
(3) Sort $\mathsf{L}_{\mathsf{Qual}}^{ct}$ lexicographically
(4) For each role R_i in the second key generation committee do the following:
  (a) Find $(\mathrm{roleassign}, t, (\textsc{Generate}, R_i, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$ and retrieve $\mathsf{ek}_R, \mathsf{vk}_R$

   (b) Find (R, msg, σ) with $msg = (\textsc{KeyGen2}, (\mathsf{pk}_i, \{ct_j\}_{j \in [n]}, \pi_{KG}))$ in $\textsc{Ordered}$
   (c) Verify that $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, msg, \sigma) = 1$, otherwise ignore this role
   (d) Retrieve the first $t + 1$ ciphertexts $ct_i^1, \ldots, ct_i^{t+1}$ for this role from $\mathsf{L}^{ct}_{\mathrm{Qual}}$ as $\{ct^k\}_{k \in [t+1]}$
   (e) Assemble ZK-statement: $stmt_{KG} \coloneqq (\mathsf{pk}_i, \{ct^k\}_{k \in [t+1]}, \{ct_i\}_{i \in [n]}, \mathsf{ek}_R)$
   (f) Verify that $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KG}, \pi_{KG}, \mathcal{R}^{AD}_{KG2}) = 1$, otherwise ignore this role
   (g) Add pk_i to $\mathsf{L}_{\mathrm{Qual}}$
 (5) Sort $\mathsf{L}_{\mathrm{Qual}}$ lexicographically
 (6) Reconstruct pk via Lagrange-interpolation from the first $t + 1$ entries in $\mathsf{L}_{\mathrm{Qual}}$
- Output P: $(\textsc{GotPK}, \mathsf{pk})$

Get System Keys:
- Input some Party P: $(\textsc{GetSKeys})$
- Behavior P:
 (1) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$
 (2) Find $(pid_{\mathrm{SO}}, t_1, (\textsc{OperatorKey}, \mathsf{vk}_{\mathrm{SO}}))$ in $\textsc{Ordered}$, otherwise abort
 (3) Find $(pid_{\mathrm{J}}, t_2, (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}}))$ in $\textsc{Ordered}$, otherwise abort
- Output P: $(\textsc{GotSKeys}, \mathsf{vk}_{\mathrm{SO}}, \mathsf{vk}_{\mathrm{J}})$

Request Decryption:
- Input LE: $(\textsc{Request}, \widetilde{W}, \sigma_{\widetilde{W}})$
- Behavior LE:
 (1) Initialize empty list $\mathsf{L}_{\mathrm{CommitteePK}}$
 (2) If $\mathsf{sk}_{\mathrm{LE}} = \bot$, then run $(\mathsf{ek}_{\mathrm{LE}}, \mathsf{sk}_{\mathrm{LE}}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$
 (3) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$
 (4) Find $(pid_{\mathrm{J}}, t_2, (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}}))$ in $\textsc{Ordered}$, otherwise abort
 (5) Find $(pid_{\mathrm{AU}}, t_3, (\textsc{AuditorKey}, \mathsf{ek}_{\mathrm{AU}}))$ in $\textsc{Ordered}$, otherwise abort
 (6) Build preliminary warrant W out of $\widetilde{W}$ by deleting the ciphertext from each entry
 (7) Compute transparency function of warrant: $W^{\mathrm{pub}} \leftarrow f_t(W)$
 (8) Encrypt warrant to the auditor: $W^{\mathrm{enc}} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\mathrm{AU}}, \widetilde{W}; r)$ with randomness $r \overset{\mathrm{R}}{\leftarrow} \mathcal{R}$
 (9) Parse warrant: $(\widetilde{W}_1, \ldots, \widetilde{W}_v) \coloneqq \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \coloneqq \widetilde{W}_i$
 (10) For each $i \in [v]$, re-randomize $\widehat{ct}_i \leftarrow \mathsf{TPKE.Rand}(ct_i; r_i)$
 (11) Assemble ZK-witness: $wit_{\widetilde{W}} \coloneqq (\widetilde{W}, r, \sigma_{\widetilde{W}}, \{ct_i, r_i\}_{i \in [v]})$
 (12) Assemble ZK-statement: $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}}, \{\widehat{ct}_i\}_{i \in [v]})$
 (13) Compute proof: $\pi_{\widetilde{W}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{\widetilde{W}}, wit_{\widetilde{W}}, \mathcal{R}^{AD}_{W})$
 (14) Send $(\textsc{Post}, (\textsc{Request}, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \{\widehat{ct}_i\}_{i \in [v]}, \pi_{\widetilde{W}}, \mathsf{ek}_{\mathrm{LE}}))$ to $\mathcal{F}_{\mathrm{BCRA}}$
- Output LE: $(\textsc{Request})$

Retrieve Secret:
- Input LE: $(\textsc{Retrieve})$
- Behavior LE:
 (1) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$
 (2) For each entry $(pid, t, R, (\textsc{Request}, C), \sigma)$ in $\textsc{Ordered}$:
   (a) Find $(\mathsf{roleassign}, t, R, \mathsf{ek}_R, \mathsf{vk}_R)$ in $\textsc{Ordered}$
   (b) Verify that $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, (\textsc{Request}, C), \sigma) = 1$, otherwise skip the following steps and proceed to next entry
   (c) Decrypt $(ct, ct^*, \pi_{Dec}) \coloneqq \mathsf{PKE.Dec}(\mathsf{sk}_{\mathrm{LE}}, C)$

(d) Assemble ZK-statement: $stmt_{Dec} \coloneqq (\mathsf{vk}, ct, ct^*)$

(e) Verify π_{Dec}: Check that $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{Dec}, \pi_{Dec}, \mathcal{R}_{Dec}^{AD}) = 1$, otherwise skip the following steps and proceed to next entry

(f) Add ct^* to $\mathsf{L}_{\mathsf{Qual}}^{ct}$

(3) Sort all lists $\mathsf{L}_{\mathsf{Qual}}^{ct}$ lexicographically

(4) For each list $\mathsf{L}_{\mathsf{Qual}}^{ct}$:

(a) Retrieve the first $t + 1$ values ct_k^* from $\mathsf{L}_{\mathsf{Qual}}^{ct}$, ignore this list if $|\mathsf{L}_{\mathsf{Qual}}^{ct}| < t + 1$

(b) Obtain decryption: $secret \coloneqq \mathsf{TPKE.Combine}(\{ct_k^*\}_{k\in[t+1]}, ct)$

(c) Add $(ct, secret)$ to $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Ready}}$

- Output LE: $(\textsc{Retrieve}, \mathsf{L}_{\mathsf{Requests}}^{\mathsf{Ready}})$

Get Statistics:

- Input some Party P: $(\textsc{GetStatistics})$
- Behavior P:

(1) Initialize empty list $\mathsf{L}_{\mathsf{Stats}}$

(2) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$

(3) Find $(pid_{\mathrm{J}}, t_2, (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}}))$ in $\textsc{Ordered}$, otherwise abort

(4) Find $(pid_{\mathrm{AU}}, t_3, (\textsc{AuditorKey}, \mathsf{ek}_{\mathrm{AU}}))$ in $\textsc{Ordered}$, otherwise abort

(5) For each entry $(pid, t, (id, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \pi_{\widetilde{W}}))$:

(a) Assemble ZK-statement: $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}})$

(b) Verify proof: $b \coloneqq \mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{\widetilde{W}}, \pi_{\widetilde{W}})$

(c) If $b = 1$ and $W^{\mathrm{pub}} \notin \mathsf{L}_{\mathsf{Stats}}$, then store W^{pub} in $\mathsf{L}_{\mathsf{Stats}}$. Else, skip this entry and proceed to next entry.

- Output P: $(\textsc{GotStatistics}, \mathsf{L}_{\mathsf{Stats}})$

Audit:

- Input AU: $(\textsc{AuditRequest})$
- Behavior AU:

(1) Initialize empty list $\mathsf{L}_{\mathsf{Warrants}}$

(2) Retrieve stored $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}})$

(3) Send $(\textsc{Read})$ to $\mathcal{F}_{\mathrm{BCRA}}$ and receive $(\textsc{Ordered})$

(4) Find $(pid_{\mathrm{J}}, t_2, (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}}))$ in $\textsc{Ordered}$, otherwise abort

(5) For each entry $(pid, t, (id, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \pi_{\widetilde{W}}))$ in $\textsc{Ordered}$:

(a) Assemble ZK-statement: $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}})$

(b) Verify proof: If $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{\widetilde{W}}, \pi_{\widetilde{W}}) = 0$, then skip the following steps and proceed to next entry

(c) Decrypt W^{enc}: $\widetilde{W} \coloneqq \mathsf{PKE.Dec}(\mathsf{sk}_{\mathrm{AU}}, W^{\mathrm{enc}})$

(d) If $\widetilde{W} \notin \mathsf{L}_{\mathsf{Warrants}}$, then store $\widetilde{W}$ in $\mathsf{L}_{\mathsf{Warrants}}$. Else, skip this entry and proceed to next entry.

- Output AU: $(\textsc{AuditAnswer}, \mathsf{L}_{\mathsf{Warrants}})$

Behavior of Nodes:

- Upon being activated for the first time, send $(\textsc{Register})$ to $\mathcal{G}_{\mathrm{CLOCK}}$
- Upon receiving a message $(\textsc{Generate}, R, \mathsf{ek}_R, \mathsf{dk}_R, \mathsf{vk}_R, \mathsf{sk}_R)$ from $\mathcal{F}_{\mathrm{BCRA}}$, store $(R, \mathsf{ek}_R, \mathsf{dk}_R, \mathsf{vk}_R, \mathsf{sk}_R)$ in $\mathsf{L}_{\mathsf{Roles}}$
- Upon receiving input $(\textsc{ExecuteRole})$:

(1) Send $(\textsc{Clock-Read})$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive (t_{Now})

```
  (2) Set $cnum \coloneqq \left\lfloor \frac{t_{\text{Now}}}{t_{\text{com}}} \right\rfloor$
  (3) Send (Read) to $\mathcal{F}_{\text{BCRA}}$ and receive (Ordered)
  (4) Find $(pid_{\text{J}}, t, (\text{JudgeKey}, \mathsf{vk}_{\text{J}}))$ in Ordered, otherwise abort
  (5) Find $(pid_{\text{AU}}, t, (\text{AuditorKey}, \mathsf{ek}_{\text{AU}}))$ in Ordered, otherwise abort
  (6) For each role in $\mathsf{L}_{\text{Roles}}$, check if it should be executed this round, if yes execute it
  (7) After having executed all roles for this round, send (Clock-Update) to $\mathcal{G}_{\text{CLOCK}}$
• Executing Role **KeyGen1**:
  (1) Retrieve $\mathsf{sk}_R$ associated with this role $R_i$ from $\mathsf{L}_{\text{Roles}}$
  (2) Initialize empty list $\mathsf{L}_{\text{CommitteePK}}{}^{\text{next}}$
  (3) *Get encryption keys for next committee.*
      For each role $R_j$ in the next committee:
      (a) Find $(\mathsf{roleassign}, t, (\text{Generate}, R_j, \mathsf{ek}_{R_j}, \mathsf{vk}_{R_j}))$ in Ordered
      (b) Insert $\mathsf{ek}_{R_j}$ into $\mathsf{L}_{\text{CommitteePK}}{}^{\text{next}}$
  (4) Generate key share: $s_i \xleftarrow{\text{R}} \mathbb{Z}_p$
  (5) Share secret key: Choose a random degree $t$ polynomial $F(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_t * x^t$ with
      $F(0) = s_i$
  (6) For each role $R_j$ in the next committee:
      (a) Set $sh_j \coloneqq F(j)$
      (b) Generate ciphertext: $ct_j \longleftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, sh_j; r_j)$
  (7) Assemble ZK-witness: $wit_{KG} \coloneqq (s_i, \{r_j\}_{j\in[n]}, F)$
  (8) Assemble ZK-statement: $stmt_{KG} \coloneqq (\{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]})$
  (9) Compute proof: $\pi_{KG} \longleftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\text{zk}}^{\text{AD}}, stmt_{KG}, wit_{KG}, \mathcal{R}_{KG1}^{AD})$
 (10) Prepare message: $msg \coloneqq (\text{KeyGen1}, (\{ct_j\}_{j\in[n]}, \pi_{KG}))$
 (11) Sign message: $\sigma \longleftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
 (12) Remove the entry for this role from $\mathsf{L}_{\text{Roles}}$
 (13) Delete all state except $msg, \sigma$
 (14) Send $(\text{Post}, (R_i, msg, \sigma))$ to $\mathcal{F}_{\text{BCRA}}$
• Executing Role **KeyGen2**:
  (1) Retrieve $\mathsf{sk}_R$ associated with this role $R_i$ from $\mathsf{L}_{\text{Roles}}$
  (2) Initialize empty lists $\mathsf{L}_{\text{CommitteePK}}{}^{\text{next}}, \mathsf{L}_{\text{CommitteePK}}{}^{\text{current}}, \mathsf{L}_{\text{CommitteeVK}}$
  (3) *Get encryption keys for next committee.*
      For each role $R_j$ in the next committee:
      (a) Find $(\mathsf{roleassign}, t, (\text{Generate}, R_j, \mathsf{ek}_{R_j}, \mathsf{vk}_{R_j}))$ in Ordered
      (b) Insert $\mathsf{ek}_{R_j}$ into $\mathsf{L}_{\text{CommitteePK}}{}^{\text{next}}$
  (4) *Get encryption keys for the current committee.*
      For each role $R_j$ in the current committee:
      (a) Find $(\mathsf{roleassign}, t, (\text{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in Ordered
      (b) Insert $\mathsf{ek}_R$ into $\mathsf{L}_{\text{CommitteePK}}{}^{\text{current}}$
  (5) *Get verification keys for the previous committee.*
      For each role $R_j$ in the previous committee:
      (a) Find $(\mathsf{roleassign}, t, (\text{Generate}, R_j, \mathsf{ek}_{R_j}, \mathsf{vk}_{R_j}))$ in Ordered
      (b) Insert $\mathsf{vk}_{R_j}$ into $\mathsf{L}_{\text{CommitteeVK}}$
  (6) *Read messages from KeyGen phase 1:*
      For each entry $(pid, t, (R_j, (\text{KeyGen1}, (\{ct_j\}_{j\in[n]}, \pi_{KG})), \sigma))$ in Ordered
      (a) Retrieve $\mathsf{vk}_{R_j}$ from $\mathsf{L}_{\text{CommitteeVK}}$
```

```
    (b) Check if $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, (\textsc{KeyGen1}, (\{ct_j\}_{j\in[n]}, \pi_{KG})), \sigma) = 1$, otherwise skip this entry.
    (c) Assemble ZK-statement: $stmt_{KG} \coloneqq (\{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]})$
    (d) If $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KG}, \pi_{KG}, \mathcal{R}^{AD}_{KG1}) \neq 1$, skip this entry
    (e) Add $ct_i$ to $\mathsf{L}_{\mathsf{Qual}}$ (the one addressed to the currently executing role)
(7) Sort $\mathsf{L}_{\mathsf{Qual}}$ lexicographically
(8) Obtain the first $t+1$ entries $ct_i^1, \ldots, ct_i^{t+1}$ from $\mathsf{L}_{\mathsf{Qual}}$
(9) Decrypt $ct_i^k$ as $sh_k \coloneqq \mathsf{PKE.Dec}(\mathsf{dk}_R, ct_i^k)$
(10) Set $\mathsf{sk}_i \coloneqq \sum_{k=1}^{t+1} sh_k$ and $\mathsf{pk}_i \coloneqq \mathsf{TPKE.Sk2Pk}(\mathsf{sk}_i)$
(11) Share secret key: Choose a random degree $t$ polynomial $F(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_t * x^t$ with
     $F(0) = \mathsf{sk}_i$
(12) For each role $R_j$ in the next committee:
    (a) Set $sh_j \coloneqq F(j)$
    (b) Generate ciphertext: $ct_j \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, sh_j; r_j)$
(13) Assemble ZK-witness: $wit_{KG} \coloneqq (\mathsf{sk}_i, \{r_j\}_{j\in[n]}, \mathsf{dk}_R, F)$
(14) Assemble ZK-statement: $stmt_{KG} \coloneqq (\mathsf{pk}_i, \{ct^k\}_{k\in[t+1]}, \{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]}, \mathsf{ek}_R)$
(15) Compute proof: $\pi_{KG} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KG}, wit_{KG}, \mathcal{R}^{AD}_{KG2})$
(16) Prepare message: $msg \coloneqq (\textsc{KeyGen2}, (\mathsf{pk}_i, \{ct_j\}_{j\in[n]}, \pi_{KG}))$
(17) Sign message: $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
(18) Remove the entry for this role from $\mathsf{L}_{\mathsf{Roles}}$
(19) Delete all state except $msg, \sigma$
(20) Send $(\textsc{Post}, (R_i, msg, \sigma))$ to $\mathcal{F}_{\mathrm{BCRA}}$
• Executing Role **Handover**:
(1) Retrieve $\mathsf{dk}_R, \mathsf{sk}_R$ associated with this role $R_i$ from $\mathsf{L}_{\mathsf{Roles}}$
(2) Initialize empty lists $\mathsf{L}_{\mathsf{CommitteePK}}{}^{\mathrm{next}}$, $\mathsf{L}_{\mathsf{CommitteePK}}{}^{\mathrm{current}}$, $\mathsf{L}_{\mathsf{CommitteeVK}}$, $\mathsf{L}_{\mathsf{Shares}}$ and $\mathsf{L}_{\mathsf{Requests}}$
(3) *Get encryption keys for next committee.*
    For each role $R_j$ in the next committee:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$
    (b) Insert $\mathsf{ek}_R$ into $\mathsf{L}_{\mathsf{CommitteePK}}{}^{\mathrm{next}}$
(4) *Get encryption keys for the current committee.*
    For each role $R_j$ in the current committee:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$
    (b) Insert $\mathsf{ek}_R$ into $\mathsf{L}_{\mathsf{CommitteePK}}{}^{\mathrm{current}}$
(5) *Get verification keys for the previous two committees.*
    For each role $R_j$ in the previous two committees:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_{R_j}, \mathsf{vk}_{R_j}))$ in $\textsc{Ordered}$
    (b) Insert $\mathsf{vk}_{R_j}$ into $\mathsf{L}_{\mathsf{CommitteeVK}}$
(6) *Get proofs for key resharings from last epoch.*
    For each entry $(pid, t, (R_j, msg, \sigma))$ with $msg = (\textsc{KeyShare}, (\mathsf{vk}_i, \{ct_j\}, \pi_{KG}))$ for the **previous** com-
    mittee in $\textsc{Ordered}$:
    (a) Retrieve $\mathsf{vk}_{R_j}$ from $\mathsf{L}_{\mathsf{CommitteeVK}}$
    (b) Check $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, msg, \sigma) = 1$, otherwise ignore this message
    (c) Assemble ZK-statement: $stmt_{KS} \coloneqq (\mathsf{vk}_i, \{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]})$
    (d) Check $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KS}, \pi_{KS}, \mathcal{R}^{AD}_{KS}) = 1$, otherwise ignore this message
    (e) Add $\{ct_j\}_{j\in[n]}$ to $\mathsf{L}^{ct}_{\mathsf{Qual}}$
(7) Sort $\mathsf{L}^{ct}_{\mathsf{Qual}}$ lexicographically
```

(8) *Read reshared key.*

For each entry $(pid, t, (R_j, msg, \sigma))$ with $msg = (\textsc{KeyShare}, (\mathsf{vk}_i, \{ct_k\}, \pi_{KS}))$ for the **current** committee in $\textsc{Ordered}$:

- (a) Retrieve vk_{R_j} from $\mathsf{L}_{\mathsf{CommitteeVK}}$
- (b) Check if $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{R_j}, (\textsc{KeyShare}, msg, \sigma) = 1$, otherwise skip this entry.
- (c) Retrieve from $\mathsf{L}^{ct}_{\mathsf{Qual}}$ from the first $t+1$ entries the ciphertext ct_j for R_j each, as $\mathsf{L}_{\mathsf{KCom}} \coloneqq \{ct^1_j, \ldots, ct^n_j\}$
- (d) Assemble ZK-statement: $stmt_{KS} \coloneqq (\mathsf{pk}_j, \{ct_k\}_{k\in[n]}, \mathsf{L}_{\mathsf{KCom}})$
- (e) Check if $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KS}, \pi_{KS}, \mathcal{R}^{AD}_{KS}) = 1$, otherwise skip this entry.
- (f) Add ct_i (the one addressed to the currently executing role) to $\mathsf{L}_{\mathsf{Qual}}$

(9) Sort $\mathsf{L}_{\mathsf{Qual}}$ lexicographically

(10) Reconstruct and reshare secret key share:

- (a) Obtain the first $t+1$ entries $ct^1_i, \ldots, ct^{t+1}_i$ from $\mathsf{L}_{\mathsf{Qual}}$
- (b) Decrypt ct^k_i as $sh_k \coloneqq \mathsf{PKE.Dec}(\mathsf{dk}_R, ct^k_i)$
- (c) Lagrange-interpolate $G(x)$ from $sh_1, \ldots, sh_{t+1}$ to obtain secret key share $\mathsf{sk}_i \coloneqq G(i)$ and set $\mathsf{vk}_i \coloneqq \mathsf{TPKE.Sk2Pk}(\mathsf{sk}_i)$
- (d) Choose a random degree t polynomial $F(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_t * x^t$ with $F(0) = \mathsf{sk}_i$
- (e) For each role R_j in the next committee:
  - (i) Generate ciphertext: $ct_j \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, F(j); r_j)$
- (f) Assemble ZK-witness: $wit_{KS} \coloneqq (\mathsf{dk}_R, \mathsf{sk}_i, \{r_j\}_{j\in[n]}, F, G)$
- (g) Assemble ZK-statement: $stmt_{KS} \coloneqq (\mathsf{ek}_R, \{ct^k_i\}_{k\in[t+1]}, \{ct_j, \mathsf{ek}_{R_j}\}_{j\in[n]}, \mathsf{vk}_i)$
- (h) Compute proof: $\pi_{KS} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{KS}, wit_{KS}, \mathcal{R}^{AD}_{KS})$
- (i) Add $msg \coloneqq (\textsc{KeyShare}, (\mathsf{vk}_i, \{ct_j\}_{j\in[n]}, \pi_{KS}))$ to L_{M}

(11) *Parse requests for decryption and store them in* $\mathsf{L}_{\mathsf{Requests}}$.

For each entry $(pid, t, (\textsc{Request}, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \{\widehat{ct}_i\}_{i\in[v]}, \pi_{\widetilde{W}}, \mathsf{ek}_{\mathrm{LE}}))$ in $\textsc{Ordered}$:

- (a) Assemble ZK-statement: $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}}, \{\widehat{ct}_i\}_{i\in[v]})$
- (b) If $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{\widetilde{W}}, \pi_{\widetilde{W}}, \mathcal{R}^{AD}_{W}) \neq 1$, abort
- (c) For each $i \in [v]$:
  - (i) Store $(\mathsf{ek}_{\mathrm{LE}}, \widehat{ct}_i)$ in $\mathsf{L}_{\mathsf{Requests}}$

(12) *Process decryption requests.*

For each entry $(\mathsf{ek}_{\mathrm{LE}}, ct)$ in $\mathsf{L}_{\mathsf{Requests}}$:

- (a) Partially decrypt ct to $ct^* \coloneqq \mathsf{TPKE.TDec}(\mathsf{sk}_i, ct)$
- (b) Assemble ZK-witness: $wit_{Dec} \coloneqq (\mathsf{sk}_i)$
- (c) Assemble ZK-statement: $stmt_{Dec} \coloneqq (\mathsf{vk}_i, ct, ct^*)$
- (d) Compute proof: $\pi_{Dec} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{\mathrm{AD}}_{\mathrm{zk}}, stmt_{Dec}, wit_{Dec}, \mathcal{R}^{AD}_{Dec})$
- (e) Encrypt answer: $msg \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\mathrm{LE}}, (ct, ct^*, \pi_{Dec}))$
- (f) Add $(\textsc{Request}, msg)$ to L_{M}

(13) *Sign outgoing messages.*

For each entry (msg) in L_{M}:

- (a) Generate signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
- (b) Update $msg \coloneqq (R_i, msg, \sigma)$

(14) Remove the entry for this role from $\mathsf{L}_{\mathsf{Roles}}$

(15) Delete all state except for L_{M}

(16) *Send messages.*

For each entry (msg) in L_{M}:

- (a) Send $(\textsc{Post}, msg)$ to $\mathcal{F}_{\mathrm{BCRA}}$

Security. In Appendix B.2 we show that our protocol Π_{AD} UC-realizes $\mathcal{F}_{\mathrm{AD}}$. In particular, we show the following theorem.

Theorem 4.4. Π_{AD} *UC-realizes* $\mathcal{F}_{\mathrm{AD}}$ *in the* $\{\mathcal{F}_{\mathrm{BCRA}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{G}_{\mathrm{CLOCK}}\}$*-hybrid model under the assumptions that*

- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- Σ *is an* EUF-CMA*-secure signature scheme,*
- *the* PKE *scheme used by LE and AU is an* IND-CPA*-secure public key encryption scheme,*
- *the* PKE *scheme that is a parameter of* $\mathcal{F}_{\mathrm{BCRA}}$ *is a* RIND-SO*-secure public key encryption scheme,*
- *and* TPKE *is an* IND-CPA*-secure, simulatable, re-randomizable and binding* (t, n)*-threshold PKE*

against all PPT*-adversaries* $\mathcal{A}$ *who may statically corrupt SO and/or LE as well as mobile adaptively corrupt at most a fraction* $\frac{t}{n} - \epsilon$ *of nodes N.*

Efficiency. Given the need for a suitable incentive for nodes to participate, it is important to limit the amount of work roles have to do. In our protocol, the work of roles only depends on the number of decryption requests (which correspond to the number of users under surveillance in the combined system), but not on the number of ciphertexts created (which corresponds to the number of registered users in the combined system). A current bottleneck is the role assignment process and the communication required to transmit the shared secret key to the next committee. This is an active area of research and improvements along the lines of [Cam+21; Cas+22; GHL22] are promising.

4.7. Discussion of Design Decisions

In this section we want to elaborate on some decisions made during the design of the whole system.

A first question would be why we used the threshold crypto approach (secret-sharing a single decryption key and encrypting the user's escrow secrets with the corresponding public key) instead of directly secret-sharing the user escrow secrets with the blockchain committee. Since there is more literature related to the secret-sharing approach (e.g., [Ben+20; Goy+22]), this approach might seem more natural at first glance. But a big drawback of this approach is *scalability*. Since there very well may be millions of escrow secrets (one for each user and validity period), the workload for the committee is immense since they need to hand-over all those secrets. While there has been effort to speed up the hand-over process [Cas+22], the workload for the committee remains high. This raises the question on how to incentivize parties to apply for a committee position, which is an entirely different research direction.

One idea to reduce the workload for committee members would be to not have *one* committee that handles *all* secrets, but *multiple* committees that each handle one (or a constant number of) secrets. While this reduces the workload per committee member, it significantly increases the number of nodes needed for all the committees. If not enough nodes are available to fill the millions of committee member positions, some nodes will have to apply for multiple committees – and thus their workload increases again. Despite the glaring disadvantage of workload, the multiple-committee approach has one advantage: if an attacker were to succeed in corrupting the majority of one of the committees, only the secrets of that committee would be at risk, not all secrets.

With our threshold crypto approach, the committee only stores *one* item (the decryption key), so the hand-over phase should be more efficient than the secret-sharing approach.

One disadvantage of our current approach is that *if* an attacker manages to compromise a majority of the blockchain committee, it can reconstruct the threshold decryption key. If that attacker then *also* corrupts (or is) the system operator, the attacker would be able to compute all escrowed user secrets. We deem this scenario unlikely, but not impossible. To prevent that, the threshold encryption keypair would need to be changed periodically, which would lead to a significant efficiency decrease. Also, then the question arises how law enforcement would be able to receive decrypted user secrets that were encrypted with a *previous* threshold key. The committee could keep copies of all previous keypairs to be able to decrypt ciphertexts created during previous key epochs. This would lead to a forward-secure but not backward-secure system.

Another question is why we put a lot of trust on the system operator. As explained in Section 4.2.3 we implicitly have to trust SO in some aspects, although SO is modeled as corruptible in our system. In particular, we assume that SO honestly stores the partial secrets and ciphertexts and releases them upon LE's request because if SO cheats at this step, we can only detect it but not prevent it. We believe this assumption to be reasonable since, to earn revenue, SO has an intrinsic motivation to keep the system running for a long time. However, it would be possible to remove that trust assumption. The user could encrypt the ciphertext containing the escrow secret again, this time under LE's public encryption key. The resulting ciphertext would then directly be sent to and stored by LE. While this approach removes some trust from SO, it has two disadvantages:

(1) LE is now directly involved in every Store Secret task and thus possibly in millions of interactions each validity period (e.g., month).

(2) This directly limits the role of LE to be played by a single party (or multiple parties sharing a single encryption keypair).

In our system we wanted to minimize the involvement of LE (in our system LE's workload only scales with the number of surveilled users, not with the number of existing users) and to keep the possibility open to expand the system to multiple law enforcement agencies (each with their own key pair). For these two reasons, we decided to put the additional trust in SO.

This second reason is also the reason why there is no fixed LE keypair in our system (neither in $\mathcal{F}_{\text{AS}}$ nor in $\mathcal{F}_{\text{AD}}$): We wanted the system to be easily expandable to multiple LE keys. Therefore, every time LE requests the decryption of a partial secret on the blockchain, it can specify a public key under which the result will be encrypted. When the role of LE is played by multiple parties, each party can specify its own key pair there.

Having the idea of multiple LE parties in mind, the question arises why we limit our system to a single auditor party with a single and fixed encryption key pair. Would it not be preferable to be able to selectively determine a different auditor for each warrant request? We deliberately chose to stick to a single auditor key pair to keep the system from being overly complex. Nonetheless, the system is designed in a way so that it is easy to extend it to multiple auditor parties (for example, there might be a separate AU party for each LE party).

A note on GDPR's "right to be forgotten". We also consider user rights, such as the "right to be forgotten" (Article 17 of the European General Data Protection Regulation (GDPR)), which enables users to request the deletion of all their personal data. Publicly accessible within our system are only ciphertexts that are stored on the blockchain and thus cannot simply be deleted, as blockchains are write-only. There are two types of ciphertexts.

Firstly, there are ciphertexts containing a secret-share of the user's escrow secret. The other secret-share of this secret is held by the system operator, who has the ability to delete its share upon receiving a deletion request from the user. Without the system operator's share, it is not possible to reconstruct the user's escrow secret.

Secondly, court orders encrypted under the auditor's key are stored on the blockchain. To avoid granting infinite access to these orders, our system can be expanded to incorporate annually rotating auditor keys. Then, upon the expiration of the retention obligation, the auditor's decryption key for the warrants pertaining to that specific year can simply be deleted.

The concept of deleting an encryption key instead of deleting the ciphertext is commonly known as "crypto-shredding". This approach is widespread in the IT security community, e.g., in HDDs, SSDs, file systems and databases.

4.8. Conclusion and Future Work

In this chapter, we developed a building block to enhance different applications with auditable surveillance. For that matter, we gave an ideal formalization that easily visualizes the security and privacy guarantees we achieve and described how this building block could be instantiated.

A realistic protocol must also cover the existence of many judges, law enforcement agencies, and auditors. Once the judge (and auditor) are not modeled as trusted parties anymore, key-revocation mechanisms become absolutely necessary, as otherwise a single key compromise allows a (state-level) adversary permanent unauthorized surveillance. Although our system ensures that such unauthorized surveillance will be noticed, it does not prevent it. See also Section 4.7 for a more fine-grained discussion of system limitations.

Moreover, to harden security, one should distribute trust over multiple parties, especially for the auditor, for example by using threshold cryptography or secure multi-party computation.

Also, it is an interesting question how to achieve more flexibility w.r.t. warrants, e.g., surveillance based on metadata similar to [GKV21], but without resorting to implausible primitives such as EWE.

Currently, realizations of the $\mathcal{F}_{\text{BCRA}}$ hybrid functionality are still novel and experimental [Ben+20; Cam+21; Cas+22], so the actual guarantees of such a realization may differ from our assumptions.

Bibliography

[ABC08] ABC News. *Toll Records Catch Unfaithful Spouses.* 2008. URL: https://abcnews.go.com/Technology/story?id=3468712&page=1 (visited on 04/19/2019).

[Abe+11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. "Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups". In: *CRYPTO 2011.* Vol. 6841. LNCS. Springer, Berlin, Heidelberg, 2011, pp. 649–666. DOI: 10.1007/978-3-642-22792-9_37.

[Abe+14] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. "Converting Cryptographic Schemes from Symmetric to Asymmetric Bilinear Groups". In: *CRYPTO 2014, Part I.* Vol. 8616. LNCS. Springer, Berlin, Heidelberg, 2014, pp. 241–260. DOI: 10.1007/978-3-662-44371-2_14.

[Abe+15a] Masayuki Abe, Markulf Kohlweiss, Miyako Ohkubo, and Mehdi Tibouchi. "Fully Structure-Preserving Signatures and Shrinking Commitments". In: *EUROCRYPT 2015, Part II.* Vol. 9057. LNCS. Springer, Berlin, Heidelberg, 2015, pp. 35–65. DOI: 10.1007/978-3-662-46803-6_2.

[Abe+15b] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. "Keys under doormats: mandating insecurity by requiring government access to all data and communications". In: *J. Cybersecur.* 1.1 (2015), pp. 69–79. DOI: 10.1093/cybsec/tyv009.

[ACL15] ACLU. *Christie Use of Tollbooth Data and Why Location Privacy Must Be Protected.* 16, 2015. URL: https://www.aclu.org/news/free-future/christie-use-tollbooth-data-and-why-location-privacy-must-be-protected (visited on 08/19/2024).

[Ame15] American Civil Liberties Union. *Newly Obtained Records Reveal Extensive Monitoring of E-ZPass Tags Throughout New York.* 2015. URL: https://www.aclu.org/blog/privacy-technology/location-tracking/newly-obtained-records-reveal-extensive-monitoring-e-zpass (visited on 04/19/2019).

[Arf+21] Ghada Arfaoui, Olivier Blazy, Xavier Bultel, Pierre-Alain Fouque, Thibaut Jacques, Adina Nedelcu, and Cristina Onete. "How to (Legally) Keep Secrets from Mobile Operators". In: *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Proceedings, Part I.* Vol. 12972. Lecture Notes in Computer Science. Springer, 2021, pp. 23–43. DOI: 10.1007/978-3-030-88418-5_2.

[Ate+00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. "A Practical and Provably Secure Coalition-Resistant Group Signature Scheme". In: *CRYPTO 2000.* Vol. 1880. LNCS. Springer, Berlin, Heidelberg, 2000, pp. 255–270. DOI: 10.1007/3-540-44598-6_16.

[Bad+17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. *Bitcoin as a Transaction Ledger: A Composable Treatment.* Cryptology ePrint Archive, Report 2017/149. 2017. URL: https://eprint.iacr.org/2017/149.

[Bal+10] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. “PrETP: Privacy-Preserving Electronic Toll Pricing”. In: *USENIX Security 2010.* USENIX Association, 2010, pp. 63–78. URL: https://www.usenix.org/legacy/event/sec10/tech/full_papers/Balasch.pdf.

[Bal+15] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. “Anonymous Transferable E-Cash”. In: *PKC 2015.* Vol. 9020. LNCS. Springer, Berlin, Heidelberg, 2015, pp. 101–124. DOI: 10.1007/978-3-662-46447-2_5.

[Bar+16] Amira Barki, Solenn Brunet, Nicolas Desmoulins, Sébastien Gambs, Saïd Gharout, and Jacques Traoré. “Private eCash in Practice (Short Paper)”. In: *FC 2016.* Vol. 9603. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 99–109. DOI: 10.1007/978-3-662-54970-4_6.

[Bat+12] Adam M. Bates, Kevin R. B. Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan S. Wallach. “Accountable Wiretapping -or- I know they can hear you now”. In: *NDSS 2012.* The Internet Society, 2012. URL: https://www.ndss-symposium.org/ndss2012/accountable-wiretapping-or-i-know-they-can-hear-you-now.

[BCS05] Michael Backes, Jan Camenisch, and Dieter Sommer. “Anonymous yet accountable access control”. In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005.* ACM, 2005, pp. 40–46. DOI: 10.1145/1102199.1102208.

[Bel15] Mihir Bellare. “New Proofs for NMAC and HMAC: Security without Collision Resistance”. In: *Journal of Cryptology* 28.4 (2015), pp. 844–878. DOI: 10.1007/s00145-014-9185-x.

[Ben+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. “Can a Public Blockchain Keep a Secret?” In: *TCC 2020, Part I.* Vol. 12550. LNCS. Springer, Cham, 2020, pp. 260–290. DOI: 10.1007/978-3-030-64375-1_10.

[BGK95] Ernest F. Brickell, Peter Gemmell, and David W. Kravitz. “Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change”. In: *6th SODA.* ACM-SIAM, 1995, pp. 457–466. URL: http://dl.acm.org/citation.cfm?id=313651.313790.

[BH04] Michael Backes and Dennis Hofheinz. “How to Break and Repair a Universally Composable Signature Functionality”. In: *Information Security, 7th International Conference, ISC 2004, Proceedings.* Vol. 3225. Lecture Notes in Computer Science. Springer, 2004, pp. 61–72. DOI: 10.1007/978-3-540-30144-8_6.

[BPS19] Florian Bourse, David Pointcheval, and Olivier Sanders. “Divisible E-Cash from Constrained Pseudo-Random Functions”. In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I.* Vol. 11921. Lecture Notes in Computer Science. Springer, 2019, pp. 679–708. DOI: 10.1007/978-3-030-34578-5_24.

[BR99] Mihir Bellare and Ronald L. Rivest. “Translucent Cryptography - An Alternative to Key Escrow, and Its Implementation via Fractional Oblivious Transfer”. In: *Journal of Cryptology* 12.2 (1999), pp. 117–139.
DOI: 10.1007/PL00003819.

[Bro+18] Brandon Broadnax, Valerie Fetzer, Jörn Müller-Quade, and Andy Rupp. “Non-malleability vs. CCA-Security: The Case of Commitments”. In: *PKC 2018, Part II*. Vol. 10770. LNCS. 2018, pp. 312–337.
DOI: 10.1007/978-3-319-76581-5_11.

[Bro+23] Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner. “PAPR: Publicly Auditable Privacy Revocation for Anonymous Credentials”. In: *Topics in Cryptology - CT-RSA 2023 - Cryptographers’ Track at the RSA Conference 2023, Proceedings*. Vol. 13871. Lecture Notes in Computer Science. Springer, 2023, pp. 163–190.
DOI: 10.1007/978-3-031-30872-7_7.

[Cam+11] Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. “Structure Preserving CCA Secure Encryption and Applications”. In: *ASIACRYPT 2011*. Vol. 7073. LNCS. Springer, Berlin, Heidelberg, 2011, pp. 89–106.
DOI: 10.1007/978-3-642-25385-0_5.

[Cam+16] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. “Universal Composition with Responsive Environments”. In: *ASIACRYPT 2016, Part II*. Vol. 10032. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 807–840.
DOI: 10.1007/978-3-662-53890-6_27.

[Cam+21] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. *Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees*. Cryptology ePrint Archive, Report 2021/1423. 2021.
URL: https://eprint.iacr.org/2021/1423.

[Cam+22] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. “Encryption to the Future - A Paradigm for Sending Secret Messages to Future (Anonymous) Committees”. In: *ASIACRYPT 2022, Part III*. Vol. 13793. LNCS. Springer, Cham, 2022, pp. 151–180.
DOI: 10.1007/978-3-031-22969-5_6.

[Can+07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. “Universally Composable Security with Global Setup”. In: *TCC 2007*. Vol. 4392. LNCS. Springer, Berlin, Heidelberg, 2007, pp. 61–85.
DOI: 10.1007/978-3-540-70936-7_4.

[Can00] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. 2000.
URL: https://eprint.iacr.org/2000/067.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *42nd FOCS*. IEEE Computer Society Press, 2001, pp. 136–145.
DOI: 10.1109/SFCS.2001.959888.

[Can04] Ran Canetti. “Universally Composable Signature, Certification, and Authentication”. In: *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004)*. IEEE Computer Society, 2004, p. 219.
DOI: 10.1109/CSFW.2004.24.

[Can20] Ran Canetti. “Universally Composable Security”. In: *Journal of the ACM* 67.5 (2020), 28:1–28:94.
DOI: 10.1145/3402457.

[Cas+22] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. *YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model.* Cryptology ePrint Archive, Report 2022/242. 2022.
URL: https://eprint.iacr.org/2022/242.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. “Efficient Protocols for Set Membership and Range Proofs”. In: *ASIACRYPT 2008.* Vol. 5350. LNCS. Springer, Berlin, Heidelberg, 2008, pp. 234–252.
DOI: 10.1007/978-3-540-89255-7_15.

[CF01] Ran Canetti and Marc Fischlin. “Universally Composable Commitments”. In: *CRYPTO 2001.* Vol. 2139. LNCS. Springer, Berlin, Heidelberg, 2001, pp. 19–40.
DOI: 10.1007/3-540-44647-8_2.

[CG08] Sébastien Canard and Aline Gouget. “Anonymity in Transferable E-cash”. In: *ACNS 08International Conference on Applied Cryptography and Network Security.* Vol. 5037. LNCS. Springer, Berlin, Heidelberg, 2008, pp. 207–223.
DOI: 10.1007/978-3-540-68914-0_13.

[Che+12] Xihui Chen, Gabriele Lenzini, Sjouke Mauw, and Jun Pang. “A Group Signature Based Electronic Toll Pricing System”. In: *Seventh International Conference on Availability, Reliability and Security.* ARES 2012. IEEE Computer Society, 2012, pp. 85–93.
DOI: 10.1109/ARES.2012.67.

[Che+13] Xihui Chen, Gabriele Lenzini, Sjouke Mauw, and Jun Pang. “Design and Formal Analysis of A Group Signature Based Electronic Toll Pricing System”. In: *JoWUA* 4.1 (2013), pp. 55–75.
URL: http://isyou.info/jowua/papers/jowua-v4n1-3.pdf.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. “Compact E-Cash”. In: *EUROCRYPT 2005.* Vol. 3494. LNCS. Springer, Berlin, Heidelberg, 2005, pp. 302–321.
DOI: 10.1007/11426639_18.

[CKS08] David Cash, Eike Kiltz, and Victor Shoup. “The Twin Diffie-Hellman Problem and Applications”. In: *EUROCRYPT 2008.* Vol. 4965. LNCS. Springer, Berlin, Heidelberg, 2008, pp. 127–145.
DOI: 10.1007/978-3-540-78967-3_8.

[CMS96] Jan Camenisch, Ueli M. Maurer, and Markus Stadler. “Digital Payment Systems with Passive Anonymity-Revoking Trustees”. In: *ESORICS’96.* Vol. 1146. LNCS. Springer, Berlin, Heidelberg, 1996, pp. 33–43.
DOI: 10.1007/3-540-61770-1_26.

[Col+21] Silvia Colabianchi, Francesco Costantino, Giulio Di Gravio, Fabio Nonino, and Riccardo Patriarca. “Discussing resilience in the context of cyber physical systems”. In: *Computers & Industrial Engineering* 160 (2021), p. 107534.
DOI: 10.1016/j.cie.2021.107534.

[Cou20] Council of the European Union. *Council Resolution on Encryption – Security through encryption and security despite encryption.* 2020.
URL: https://data.consilium.europa.eu/doc/document/ST-13084-2020-REV-1/en/pdf (visited on 04/19/2018).

[CSO23] CSO Online. *The Biggest Data Breaches of the 21st Century*. 2023.
URL: https://www.csoonline.com/article/534628/the-biggest-data-breaches-of-the-21st-century.html (visited on 08/19/2024).

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. "Universally Composable Authentication and Key-Exchange with Global PKI". In: *PKC 2016, Part II*. Vol. 9615. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 265–296.
DOI: 10.1007/978-3-662-49387-8_11.

[Dam99] Ivan Damgård. "Commitment Schemes and Zero-Knowledge Protocols". In: *Lectures on Data Security: Modern Cryptology in Theory and Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 63–86.
DOI: 10.1007/3-540-48969-X_3.

[Dat07] Datatilsynet. *Statens Vegvesen Holdt Tilbake Viktig AutoPASS-Informasjon (Press release)*. 2007.
URL: http://www.datatilsynet.no/ (visited on 04/19/2019).

[Day+11] Jeremy Day, Yizhou Huang, Edward Knapp, and Ian Goldberg. "SPEcTRe: Spot-Checked Private Ecash Tolling at Roadside". In: *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, WPES 2011*. ACM, 2011, pp. 61–68.
DOI: 10.1145/2046556.2046565.

[Daz+22] Vanesa Daza, Abida Haque, Alessandra Scafuro, Alexandros Zacharakis, and Arantxa Zapico. "Mutual Accountability Layer: Accountable Anonymity Within Accountable Trust". In: *Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML 2022, Proceedings*. Vol. 13301. Lecture Notes in Computer Science. Springer, 2022, pp. 318–336.
DOI: 10.1007/978-3-031-07689-3_24.

[DDS11] Morten Dahl, Stéphanie Delaune, and Graham Steel. "Formal Analysis of Privacy for Anonymous Location Based Services". In: *Theory of Security and Applications - Joint Workshop, TOSCA 2011, Revised Selected Papers*. Vol. 6993. Lecture Notes in Computer Science. Springer, 2011, pp. 98–112.
DOI: 10.1007/978-3-642-27375-9_6.

[de +13] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. "Unique in the Crowd: The Privacy Bounds of Human Mobility". In: *Scientific Reports* 3.1 (2013), p. 1376.
DOI: 10.1038/srep01376.

[Deu18] Deutsches Bundesamt für Güterverkehr. *Monatliche Mautstatistik für Januar 2018*. 2018.
URL: https://www.bag.bund.de/SharedDocs/Downloads/DE/Statistik/Lkw-Maut/18_Monatstab_01.html (visited on 01/09/2019).

[DY04] Yevgeniy Dodis and Aleksandr Yampolskiy. *A Verifiable Random Function With Short Proofs and Keys*. Cryptology ePrint Archive, Report 2004/310. 2004.
URL: https://eprint.iacr.org/2004/310.

[EFR21] Andreas Erwig, Sebastian Faust, and Siavash Riahi. *Large-Scale Non-Interactive Threshold Cryptosystems Through Anonymity*. Cryptology ePrint Archive, Report 2021/1290. 2021.
URL: https://eprint.iacr.org/2021/1290.

[EG14] Alex Escala and Jens Groth. "Fine-Tuning Groth-Sahai Proofs". In: *PKC 2014*. Vol. 8383. LNCS. Springer, Berlin, Heidelberg, 2014, pp. 630–649.
DOI: 10.1007/978-3-642-54631-0_36.

[Ele24] Electronic Frontier Foundation. *Senators Expose Car Companies' Terrible Data Privacy Practices.* 2024.
URL: https://www.eff.org/deeplinks/2024/07/senators-expose-car-companies-terrible-data-privacy-practices (visited on 08/22/2024).

[Eur14] European Parliament. *Technology Options for the European Electronic Toll Service.* 2014.
URL: http://www.europarl.europa.eu/RegData/etudes/STUD/2014/529058/IPOL_STUD(2014)529058_EN.pdf (visited on 04/19/2019).

[Eur15] European Commission. *Study on State of the Art of Electronic Road Tolling.* 2015.
URL: https://ec.europa.eu/transport/sites/transport/files/modes/road/road_charging/doc/study-electronic-road-tolling.pdf (visited on 04/19/2019).

[Eur16] European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).* GDPR. 27, 2016.
URL: https://eur-lex.europa.eu/eli/reg/2016/679 (visited on 06/12/2024).

[Eur17] European Commission. *Proposal for a Directive of the European Parliament and of the Council on the Interoperability of Electronic Road Toll Systems and Facilitating Crossborder Exchange of Information on the Failure to Pay Road Fees in the Union (recast).* 2017.
URL: https://ec.europa.eu/transport/sites/transport/files/com20170280-eets-directive.pdf (visited on 04/19/2019).

[Eur18] European Union. *Directive (EU) 2018/843 of the European Parliament and of the Council of 30 May 2018 amending Directive (EU) 2015/849 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing, and amending Directives 2009/138/EC and 2013/36/EU.* 5th Anti-Money Laundering Directive. 30, 2018.
URL: http://data.europa.eu/eli/dir/2018/843/oj (visited on 08/24/2024).

[Fau+25] Dennis Faut, Valerie Fetzer, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. "POBA: Privacy-Preserving Operator-Side Bookkeeping and Analytics". In: *IACR Communications in Cryptology* 2.2 (7, 2025).
DOI: 10.62056/av11zo-3y.

[Fet+18] Valerie Fetzer, Max Hoffmann, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. *P4TC – Provably-Secure yet Practical Privacy-Preserving Toll Collection.* Cryptology ePrint Archive, Report 2018/1106. 2018.
URL: https://eprint.iacr.org/2018/1106.

[Fet+20] Valerie Fetzer, Max Hoffmann, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. "P4TC – Provably-Secure yet Practical Privacy-Preserving Toll Collection". In: *PoPETs* 2020.3 (2020), pp. 62–152.
DOI: 10.2478/popets-2020-0046.

[Fet+22] Valerie Fetzer, Marcel Keller, Sven Maier, Markus Raiber, Andy Rupp, and Rebecca Schwerdt. "PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics". In: *PoPETs* 2022.2 (2022), pp. 447–516.
DOI: 10.2478/popets-2022-0054.

[Fet+23a] Valerie Fetzer, Michael Klooß, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. *Universally Composable Auditable Surveillance.* Cryptology ePrint Archive, Report 2023/1343. 2023.
URL: https://eprint.iacr.org/2023/1343.

[Fet+23b] Valerie Fetzer, Michael Klooß, Jörn Müller-Quade, Markus Raiber, and Andy Rupp. "Universally Composable Auditable Surveillance". In: *ASIACRYPT 2023, Part II*. Vol. 14439. LNCS. 2023, pp. 453–487.
DOI: 10.1007/978-981-99-8724-5_14.

[FMN16] Valerie Fetzer, Jörn Müller-Quade, and Tobias Nilges. "A Formal Treatment of Privacy in Video Data". In: *ESORICS 2016, Part II*. Vol. 9879. LNCS. 2016, pp. 406–424.
DOI: 10.1007/978-3-319-45741-3_21.

[Fra+18] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. "Practical Accountability of Secret Processes". In: *USENIX Security 2018*. USENIX Association, 2018, pp. 657–674.
URL: https://www.usenix.org/conference/usenixsecurity18/presentation/frankle.

[Gar+14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. "On the Implausibility of Differing-Inputs Obfuscation and Extractable Witness Encryption with Auxiliary Input". In: *CRYPTO 2014, Part I*. Vol. 8616. LNCS. Springer, Berlin, Heidelberg, 2014, pp. 518–535.
DOI: 10.1007/978-3-662-44371-2_29.

[Gen+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. "YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles". In: *CRYPTO 2021, Part II*. Vol. 12826. LNCS. Virtual Event: Springer, Cham, 2021, pp. 64–93.
DOI: 10.1007/978-3-030-84245-1_3.

[GGM16] Christina Garman, Matthew Green, and Ian Miers. "Accountable Privacy for Decentralized Anonymous Payments". In: *FC 2016*. Vol. 9603. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 81–98.
DOI: 10.1007/978-3-662-54970-4_5.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties". In: *EUROCRYPT 2022, Part I*. Vol. 13275. LNCS. Springer, Cham, 2022, pp. 458–487.
DOI: 10.1007/978-3-031-06944-4_16.

[GKV21] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. "Abuse Resistant Law Enforcement Access Systems". In: *EUROCRYPT 2021, Part III*. Vol. 12698. LNCS. Springer, Cham, 2021, pp. 553–583.
DOI: 10.1007/978-3-030-77883-5_19.

[Glo19] Global Markets Insights. *ETC Market Report 2019*. 2019.
URL: https://www.marketwatch.com/press-release/electronic-toll-collection-market-2019-in-depth-industry-analysis-by-product-technology-application-opportunities-and-growth-forecast-by-2025-2019-11-12 (visited on 11/12/2019).

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. "Witness Encryption from Instance Independent Assumptions". In: *CRYPTO 2014, Part I*. Vol. 8616. LNCS. Springer, Berlin, Heidelberg, 2014, pp. 426–443.
DOI: 10.1007/978-3-662-44371-2_24.

[Gol+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. "How to Run Turing Machines on Encrypted Data". In: *CRYPTO 2013, Part II*. Vol. 8043. LNCS. Springer, Berlin, Heidelberg, 2013, pp. 536–553.
DOI: 10.1007/978-3-642-40084-1_30.

[Goy+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. “Storing and Retrieving Secrets on a Blockchain”. In: *PKC 2022, Part I*. Vol. 13177. LNCS. Springer, Cham, 2022, pp. 252–282.
DOI: 10.1007/978-3-030-97121-2_10.

[GP09] Philippe Golle and Kurt Partridge. “On the Anonymity of Home/Work Location Pairs”. In: *Pervasive Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 390–397.
DOI: 10.1007/978-3-642-01516-8_26.

[GP17] Shafi Goldwasser and Sunoo Park. “Public Accountability vs. Secret Laws: Can They Coexist?: A Cryptographic Proposal”. In: *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. ACM, 2017, pp. 99–110.
DOI: 10.1145/3139550.3139565.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. “Pairings for cryptographers”. In: *Discrete Applied Mathematics* 156.16 (2008). Applications of Algebra to Cryptography, pp. 3113–3121.
DOI: 10.1016/j.dam.2007.12.010.

[Gro19] Encryption Working Group. *Moving the Encryption Policy Conversation Forward*. Tech. rep. Carnegie Endowment for International Peace, 2019.
URL: https://coilink.org/20.500.12592/vmg7cd.

[GRR19] Frank Gurski, Carolin Rehs, and Jochen Rethmann. “Knapsack problems: A parameterized point of view”. In: *Theor. Comput. Sci.* 775 (2019), pp. 93–108.
DOI: 10.1016/J.TCS.2018.12.019.

[GS08] Jens Groth and Amit Sahai. “Efficient Non-interactive Proof Systems for Bilinear Groups”. In: *EUROCRYPT 2008*. Vol. 4965. LNCS. Springer, Berlin, Heidelberg, 2008, pp. 415–432.
DOI: 10.1007/978-3-540-78967-3_24.

[GVJ11] Flavio D. Garcia, Eric R. Verheul, and Bart Jacobs. “Cell-Based Roadpricing”. In: *Public Key Infrastructures, Services and Applications - 8th European Workshop, EuroPKI 2011, Revised Selected Papers*. Vol. 7163. Lecture Notes in Computer Science. Springer, 2011, pp. 106–122.
DOI: 10.1007/978-3-642-29804-2_7.

[Har+17] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. “BBA+: Improving the Security and Applicability of Privacy-Preserving Point Collection”. In: *ACM CCS 2017*. ACM Press, 2017, pp. 1925–1942.
DOI: 10.1145/3133956.3134071.

[Hof+20] Max Hoffmann, Michael Klooß, Markus Raiber, and Andy Rupp. “Black-Box Wallets: Fast Anonymous Two-Way Payments for Constrained Devices”. In: *Proceedings on Privacy Enhancing Technologies* 2020.1 (2020), pp. 165–194.
DOI: 10.2478/popets-2020-0010.

[HPW15] Carmit Hazay, Arpita Patra, and Bogdan Warinschi. “Selective Opening Security for Receivers”. In: *ASIACRYPT 2015, Part I*. Vol. 9452. LNCS. Springer, Berlin, Heidelberg, 2015, pp. 443–469.
DOI: 10.1007/978-3-662-48797-6_19.

[IT 23] IT Governance. *List of Data Breaches and Cyber Attacks in 2023*. 2023.
URL: https://www.itgovernance.co.uk/blog/list-of-data-breachesand-cyber-attacks-in-2023 (visited on 08/19/2024).

[Jar+14] Roger Jardí-Cedó, Macià Mut Puigserver, Magdalena Payeras-Capellà, Jordi Castellà-Roca, and Alexandre Viejo. "Electronic Road Pricing System for Low Emission Zones to Preserve Driver Privacy". In: *Modeling Decisions for Artificial Intelligence - 11th International Conference, MDAI 2014. Proceedings.* Vol. 8825. Lecture Notes in Computer Science. Springer, 2014, pp. 1–13.
DOI: 10.1007/978-3-319-12054-6_1.

[Jar+16] Roger Jardí-Cedó, Macià Mut Puigserver, Jordi Castellà-Roca, Magdalena Payeras-Capellà, and Alexandre Viejo. "Privacy-preserving Electronic Road Pricing System for Multifare Low Emission Zones". In: *Proceedings of the 9th International Conference on Security of Information and Networks.* SIN 2016. ACM, 2016, pp. 158–165.
DOI: 10.1145/2947626.2947653.

[JCV14] Roger Jardí-Cedó, Jordi Castellà-Roca, and Alexandre Viejo. "Privacy-Preserving Electronic Toll System with Dynamic Pricing for Low Emission Zones". In: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance - 9th International Workshop, DPM 2014, 7th International Workshop, SETOP 2014, and 3rd International Workshop, QASA 2014. Revised Selected Papers.* Vol. 8872. Lecture Notes in Computer Science. Springer, 2014, pp. 327–334.
DOI: 10.1007/978-3-319-17016-9_22.

[Jol+24] Amirhossein Adavoudi Jolfaei, Andy Rupp, Stefan Schiffner, and Thomas Engel. "Why Privacy-Preserving Protocols Are Sometimes Not Enough: A Case Study of the Brisbane Toll Collection Infrastructure". In: *Proc. Priv. Enhancing Technol.* 2024.1 (2024), pp. 232–257.
DOI: 10.56553/POPETS-2024-0014.

[Jol+25] Amirhossein Adavoudi Jolfaei, Andy Rupp, Stefan Schiffner, and Valerie Fetzer. "Differential Privacy to the Rescue? On Obfuscating Tolls in Privacy-Preserving ETC Systems". In Submission. 2025.

[JS04] Stanislaw Jarecki and Vitaly Shmatikov. "Handcuffing Big Brother: an Abuse-Resilient Transaction Escrow Scheme". In: *EUROCRYPT 2004.* Vol. 3027. LNCS. Springer, Berlin, Heidelberg, 2004, pp. 590–608.
DOI: 10.1007/978-3-540-24676-3_35.

[Kap18] Kapsch (Toll Collection System Integrator and Supplier). Personal Communication. 2018.

[KDD15] Sebastian Kummer, Maria Dieplinger, and Mario Dobrovnik. *Endbericht der Studie "Flächendeckende Schwerverkehrs-Maut in Österreich".* 2015.
URL: https://www.wko.at/branchen/stmk/transport-verkehr/gueterbefoerderungsgewerbe/Endbericht_FlaechendeckendeMaut_final.pdf (visited on 01/09/2019).

[KF08] Kaoru Kurosawa and Jun Furukawa. "Universally Composable Undeniable Signature". In: *ICALP 2008, Part II.* Vol. 5126. LNCS. Springer, Berlin, Heidelberg, 2008, pp. 524–535.
DOI: 10.1007/978-3-540-70583-3_43.

[KFB14] Joshua A. Kroll, Edward W. Felten, and Dan Boneh. *Secure protocols for accountable warrant execution.* 2014.
URL: https://www.jkroll.com/papers/warrant_paper.pdf (visited on 04/19/2018).

[KL11] Dafna Kidron and Yehuda Lindell. "Impossibility Results for Universal Composability in Public-Key Models and with Fixed Inputs". In: *Journal of Cryptology* 24.3 (2011), pp. 517–544.
DOI: 10.1007/s00145-010-9069-7.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[KL15] Florian Kerschbaum and Hoon Wei Lim. "Privacy-Preserving Observation in Public Spaces". In: *ESORICS 2015, Part II.* Vol. 9327. LNCS. Springer, Cham, 2015, pp. 81–100.
DOI: 10.1007/978-3-319-24177-7_5.

[KL95] Joe Kilian and Frank Thomson Leighton. "Fair Cryptosystems, Revisited: A Rigorous Approach to Key-Escrow (Extended Abstract)". In: *CRYPTO'95.* Vol. 963. LNCS. Springer, Berlin, Heidelberg, 1995, pp. 208–221.
DOI: 10.1007/3-540-44750-4_17.

[KP98] Joe Kilian and Erez Petrank. "Identity Escrow". In: *CRYPTO'98.* Vol. 1462. LNCS. Springer, Berlin, Heidelberg, 1998, pp. 169–185.
DOI: 10.1007/BFb0055727.

[KPW15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. "Structure-Preserving Signatures from Standard Assumptions, Revisited". In: *CRYPTO 2015, Part II.* Vol. 9216. LNCS. Springer, Berlin, Heidelberg, 2015, pp. 275–295.
DOI: 10.1007/978-3-662-48000-7_14.

[Kro+18] Joshua A. Kroll, Joe Zimmerman, David J. Wu, Valeria Nikolaenko, Edward W. Felten, and Dan Boneh. *Accountable Cryptographic Access Control.* 2018.
URL: https://www.cs.yale.edu/homes/jf/kroll-paper.pdf (visited on 04/19/2018).

[Kru07] John Krumm. "Inference Attacks on Location Tracks". In: *Pervasive Computing.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 127–143.
DOI: 10.1007/978-3-540-72037-9_8.

[KV03] Dennis Kügler and Holger Vogt. "Offline Payments with Auditable Tracing". In: *FC 2002.* Vol. 2357. LNCS. Springer, Berlin, Heidelberg, 2003, pp. 269–281.
DOI: 10.1007/3-540-36504-4_19.

[Lap+10] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. "Performance Analysis of Accumulator-Based Revocation Mechanisms". In: *Security and Privacy - Silver Linings in the Cloud - 25th IFIP TC-11 International Information Security Conference, SEC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings.* 2010, pp. 289–301.
DOI: 10.1007/978-3-642-15257-3_26.

[LRC14] Jia Liu, Mark Dermot Ryan, and Liqun Chen. "Balancing Societal Security and Individual Privacy: Accountable Escrow System". In: *CSF 2014 Computer Security Foundations Symposium.* IEEE Computer Society Press, 2014, pp. 427–440.
DOI: 10.1109/CSF.2014.37.

[Mar17] Markets and Markets. *Electronic Toll Collection Market Study.* 2017.
URL: https://www.marketsandmarkets.com/Market-Reports/electronic-toll-collection-system-market-224492059.html (visited on 04/19/2018).

[Mei+11] Sarah Meiklejohn, Keaton Mowery, Stephen Checkoway, and Hovav Shacham. "The Phantom Tollbooth: Privacy-Preserving Electronic Toll Collection in the Presence of Driver Collusion". In: *USENIX Security 2011.* USENIX Association, 2011.
URL: https://www.usenix.org/legacy/events/sec11/tech/full_papers/Meiklejohn.pdf.

[MV97] Alfred J. Menezes and Paul C. Van Oorschot. *Handbook of applied cryptography.* 1st edition. CRC Press, 1997.
URL: https://cacr.uwaterloo.ca/hac/.

[Off18] Official Journal of the European Communities. *Directive (EU) 2018/843 of the European Parliament and of the Council amending Directive (EU) 2015/849 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing, and amending Directives 2009/138/EC and 2013/36/EU.* 2018.
URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31996G1104&from=EN (visited on 04/19/2018).

[Off95] Official Journal of the European Communities. *Council Resolution on on the lawful interception of telecommunications.* 1995.
URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31996G1104&from=EN (visited on 04/19/2018).

[Pan+19] Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Austin Bos. “SAMPL: Scalable Auditability of Monitoring Processes using Public Ledgers”. In: *ACM CCS 2019.* ACM Press, 2019, pp. 2249–2266.
DOI: 10.1145/3319535.3354219.

[PBB09] Raluca A. Popa, Hari Balakrishnan, and Andrew J. Blumberg. “VPriv: Protecting Privacy in Location-Based Vehicular Services”. In: *USENIX Security 2009.* USENIX Association, 2009, pp. 335–350.
URL: https://www.usenix.org/legacy/events/sec09/tech/full_papers/popa.pdf.

[PPY22] Giuseppe Persiano, Duong Hieu Phan, and Moti Yung. “Anamorphic Encryption: Private Communication Against a Dictator”. In: *EUROCRYPT 2022, Part II.* Vol. 13276. LNCS. Springer, Cham, 2022, pp. 34–63.
DOI: 10.1007/978-3-031-07085-3_2.

[Ps10] Raphael Pass and abhi shelat. *A Course in Cryptography.* available online. 2010.
URL: http://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf.

[PY00] Pascal Paillier and Moti Yung. “Self-Escrowed Public-Key Infrastructures”. In: *ICISC 99.* Vol. 1787. LNCS. Springer, Berlin, Heidelberg, 2000, pp. 257–268.
DOI: 10.1007/10719994_20.

[Sav17] Savari.net. *MobiWAVE On-Board-Unit (OBU).* 2017.
URL: http://savari.net/wp-content/uploads/2017/05/MW-1000_April2017.pdf (visited on 02/05/2018).

[Sav18] Stefan Savage. “Lawful Device Access without Mass Surveillance Risk: A Technical Design Discussion”. In: *ACM CCS 2018.* ACM Press, 2018, pp. 1761–1774.
DOI: 10.1145/3243734.3243758.

[Sca19] Alessandra Scafuro. “Break-glass Encryption”. In: *PKC 2019, Part II.* Vol. 11443. LNCS. Springer, Cham, 2019, pp. 34–62.
DOI: 10.1007/978-3-030-17259-6_2.

[Sch+19] Rebecca Schwerdt, Matthias Nagel, Valerie Fetzer, Tobias Gräf, and Andy Rupp. “P6V2G: A Privacy-Preserving V2G Scheme for Two-Way Payments and Reputation”. In: *Energy Informatics* 2.1 (27, 2019), p. 32.
DOI: 10.1186/s42162-019-0075-1.

[SW19] Sacha Servan-Schreiber and Archer Wheeler. “Judge, Jury & Encryptioner: Exceptional Access with a Fixed Social Cost”. In: *CoRR* abs/1912.05620 (2019).
URL: http://arxiv.org/abs/1912.05620.

[Swe02] Latanya Sweeney. “k-Anonymity: A Model for Protecting Privacy”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pp. 557–570. DOI: 10.1142/S0218488502001648.

[TC18] Transport and Public Works Committee. *Inquiry into the operations of toll roads in Queensland.* 2018. URL: https://www.transurban.com/content/dam/transurban-pdfs/02/news/transurban-submission-inquiry-qld.pdf (visited on 09/25/2024).

[Tie10] Lee Tien. *New “Smart Meters” for Energy Use Put Privacy at Risk.* 2010. URL: https://www.eff.org/deeplinks/2010/03/new-smart-meters-energy-use-put-privacy-risk (visited on 06/12/2024).

[US 86] U.S. Congress. *18 U.S. Code §2703 – Required disclosure of customer communications or records.* Stored Communications Act. As amended. 1986. URL: https://www.law.cornell.edu/uscode/text/18/2703 (visited on 08/24/2024).

[Val79] Leslie G. Valiant. “The Complexity of Enumeration and Reliability Problems”. In: *SIAM Journal on Computing* 8.3 (1979), pp. 410–421. DOI: 10.1137/0208032.

[WV18] Charles V. Wright and Mayank Varia. “Crypto Crumple Zones: Enabling Limited Access without Mass Surveillance”. In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018.* IEEE, 2018, pp. 288–306. DOI: 10.1109/EuroSP.2018.00028.

[YY98] Adam Young and Moti Yung. “Auto-Recoverable Auto-Certifiable Cryptosystems”. In: *EUROCRYPT’98.* Vol. 1403. LNCS. Springer, Berlin, Heidelberg, 1998, pp. 17–31. DOI: 10.1007/BFb0054114.

A. Appendix for Chapter 3

A.1. Security Proof: Π_{P4TC} UC-realizes $\mathcal{F}_{\text{P4TC}}$

In this appendix we show that Π_{P4TC} UC-realizes $\mathcal{F}_{\text{P4TC}}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}})$-hybrid model for static corruption. More precisely, we show the following theorem:

Theorem A.1 (Security of Π_{P4TC}). *Under the assumptions that the Co-CDH problem is hard for (G_1, G_2), the DLOG-problem is hard for G_1, the SXDH-problem*[1] *is hard for* $\text{gp} \coloneqq (G_1, G_2, G_T, e, \text{q}, g_1, g_2)$*, the λ_{PRF}-DDHI problem is hard for G_1, and our building blocks (encryption scheme* E*, commitment schemes* C1 *and* C2*, signature scheme* S*, NIZK* POK*, and PRF* PRF*) are instantiated as described in Section 3.4.4 it holds that*

$$\Pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}}} \geq_{UC} \mathcal{F}_{\text{P4TC}}^{\mathcal{G}_{\text{bb}}}$$

under static corruption of either

(1) a subset of users,

(2) all users and a subset of RSUs, TSP and SA,

(3) a subset of RSUs, TSP and SA, or

(4) all RSUs, TSP and SA as well as a subset of users.

Why this corruption model was chosen and why it does not represent a severe restriction from a practical point of view is explained below.

Remark A.2. When instantiating the building blocks as described in Section 3.4.4, the building blocks have the following properties:

- E is a structure-preserving IND-CCA2-secure asymmetric encryption scheme
- C1 is a group-based commitment scheme that is statistically hiding, additively homomorphic, equivocal, and F_{gp}-Binding for $F_{\text{gp}}(m_1, \ldots, m_\alpha) \coloneqq (g_1^{m_1}, \ldots, g_1^{m_\alpha})$
- C2 is a (dual-mode) equivocal and extractable commitment scheme (which is, depending on the mode, either statistically hiding or statistically binding)
- S is a structure-preserving EUF-CMA-secure signature scheme
- POK is a (dual-mode) non-interactive proof system, where one can switch between composable zero-knowledge and perfect F_{gp}-extractability, with three different instantiations, denoted by P1, P2, and P3, which share a common reference string
- PRF is a group-based pseudo-random function

[1] Please note that the hardness of the Co-CDH problem and DLOG-problem is already implied by the SXDH-assumption.

We prove Theorem A.1 in three steps:

- In Appendix A.1.1, we first show some structural properties of $\mathcal{F}_{\text{P4TC}}^{\mathcal{G}_{\text{bb}}}$.
- In Appendix A.1.2, Theorem A.11 proves Theorem A.1 for the corruption scenarios (1) and (2). We call this case "Operator Security".
- In Appendix A.1.3, Theorem A.26 proves Theorem A.1 for the corruption scenarios (3) and (4). We call this case "User Security and Privacy".

Proof of Theorem A.1. The theorem is immediately implied by Theorem A.11 and Theorem A.26. □

Before giving the proof in full detail and complexity in Appendices A.1.1 to A.1.3, we explain our adversarial model.

Restricted Corruption. Firstly, we only consider security under *static* corruption. This is a technical necessity to enable the use of PRFs to generate fraud detection IDs. With adaptive corruption the simulator would be required to come up with a consistent PRF that could explain the up to the point of corruption uniformly and randomly drawn fraud detection IDs. We deem static corruption to provide a sufficient level of security as a statically corrupted party may always decide to interact honestly first and then deviate from the protocol later. Adaptive corruption is tightly related to deniability which is not part of our desired properties.

Secondly, we only consider adversaries $\mathcal{Z}$ that corrupt one of the following sets[2]:

(1) A subset of users.

(2) All users and a subset of RSUs, TSP and SA.

(3) A subset of RSUs, TSP and SA.

(4) All of RSUs, TSP and SA as well as a subset of users.

We subsume the cases (1) and (2) under the term *Operator Security* and the cases (3) and (4) under the term *User Security*. For both Operator Security and User Security the two subordinate cases are collectively treated by the same proof. It is best to picture the cases inversely: To prove Operator Security we consider a scenario in which at least some parties at the operator's side remain honest; to prove User Security we consider a scenario in which at least some users remain honest. Please note that both scenarios also commonly cover the case in which all parties are corrupted, however, this extreme case is tedious as it is trivially simulatable.

One might believe that the combination of all cases above should already be sufficient to guarantee privacy, security and correctness under arbitrary corruption. For example, case (4) guarantees that privacy and correctness of accounting are still provided for honest users, even if all of the operator's side and some fellow users are corrupted. This ought to be the worst case from a honest user's perspective. Further note that the proof of indistinguishability quantifies over all environments $\mathcal{Z}$. This includes environments that—still in case (4)—first corrupt all the operator's side but then let some (formally corrupted) parties follow the protocol honestly.

[2] Note that "subset" also includes the empty or full set.

However, consider a scenario in which a party acts as a *Man-in-the-Middle (MitM)* playing the roles of a user and an RSU at the same time while interacting with an honest user in the left interaction and an honest RSU in the right interaction. The MitM simply relays messages back and forth unaltered. If the MitM approaches the RSU and the RSU requests the MitM to participate in Debt Accumulation, the MitM relays all messages of the RSU to the honest user (possibly driving the same road behind the MitM). The honest user replies and the MitM forwards the messages to the honest RSU. The MitM passes by the RSU unnoticed and untroubled, while the honest user pays for the MitM.

This scenario is not captured by any of the above cases and is the missing gap towards arbitrary corruption. As the MitM is corrupted and plays both roles of a user and RSU, this falls into case (2) or (4). But either all users are corrupted in case (2), which contradicts the existence of an honest user in the left interaction, or all of RSUs, TSP and SA are corrupted in case (4), which does not allow for an honest RSU in the right interaction.

This attack is known as *relay attack*. Please note that the MitM does not need to break any cryptographic assumption for this kind of attack as it just poses as a prolonged communication channel. There are some possible countermeasures that can be applied in the real world. For example, using distance-bounding the honest user could refuse to participate in the protocol, if the RSU is known to be to far away. However, these are physical counter measures and thus are not captured by the UC notion nor any other cryptographic notion. Actually, it is a strength of the UC model that this gap is made explicit. For example, a set of game-based security notions that cover a list of individual properties would most likely not unveil this issue.

Channel Model. Most of the time we assume channels to be secure and authenticated. The only exception is Debt Accumulation, which uses a secure but only half-authenticated channel. Half-authenticated channel means only the RSU authenticates itself and the user does not. These channels exempt us from the burden of defining a simulator for the case where only honest parties interact with each other and rules out some trivial replay attacks. Of course, the authentication of the channels must be tied to the parties' credentials used in the toll collection system. In other words, the same key registration service that registers the public keys for the toll collection system must also be used to register the public keys to authenticate the communication.

A.1.1. Proof of Correctness

Many papers that show some protocol to be UC-secure consider rather simple cases (e.g., a commitment, an oblivious transfer, a coin toss) and correctness of the ideal functionality is mostly obvious. In contrast, our ideal functionality $\mathcal{F}_{\text{P4TC}}$ is already a complex system on its own with polynomially many parties that can reactively interact forever, i.e., $\mathcal{F}_{\text{P4TC}}$ itself has no inherent exit point except that at some point the polynomially bounded runtime of the environment is exhausted. In this appendix no security reduction occurs, because we only consider the ideal functionality $\mathcal{F}_{\text{P4TC}}$. We start with a series of simple lemmas which also help to develop a good conception about how the individual tasks/transactions are connected. Moreover, these lemmas are closely associated to the desired properties of a toll collection scheme (cp. Section 3.2.4 and Section 3.3.3).

Internally, $\mathcal{F}_{\text{P4TC}}$ stores a pervasive database *TRDB* whose entries *trdb* are of the form

$$trdb = (s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b).$$

This set can best be visualized as a directed graph in which each node represents the state of a user *after* the respective transaction, i.e., at the end of an execution of Wallet Issuing, Debt Accumulation or Debt

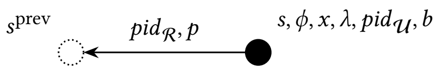


Figure A.1.: An entry $trdb \in TRDB$ visualized as an element of a directed graph

Clearance, and the edges correspond to the transition from the previous to the next state. Each *trdb* entry represents a node together with an edge pointing to its predecessor node. The node is labeled with $(s, \phi, x, \lambda, pid_{\mathcal{U}}, b)$ and identified by s. The edge to the predecessor is identified by (s^{prev}, s) and labeled with $(pid_{\mathcal{R}}, p)$. See Fig. A.1 for a depiction. Transaction entries or nodes that are inserted by Wallet Issuing do not have a predecessor, therefore $s^{\text{prev}} = \bot$ and also $p = 0$ holds. All other tasks besides Wallet Issuing, Debt Accumulation and Debt Clearance do not alter the graph but only query it. We show that the graph is a directed forest, i.e., a set of directed trees. Wallet Issuing creates a new tree by inserting a new root node. Debt Accumulation and Debt Clearance extend a tree. Debt Clearance results in a leaf node from where no further extension is possible. As long as no double spending occurs, each tree is a path graph.

Definition A.3 (Ideal Transaction Graph (Informal))**.** The transaction database $TRDB = \{trdb_i\}$ with $trdb = (s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ is a directed, labeled graph as defined above. This graph is called the *Ideal Transaction Graph.*

Lemma A.4 (Ideal Transaction Forest)**.** *The Ideal Transaction Graph TRDB is a forest.*

Proof. *TRDB* is a forest, if and only if it is cycle-free and every node has in-degree at most one. A new node is only inserted in the scope of Wallet Issuing, Debt Accumulation or Debt Clearance.

Proof by Induction: The statement is correct for the empty *TRDB*. If Wallet Issuing is invoked, a new node with no predecessor is inserted. Moreover, the serial number s of the new node is randomly chosen from the set of unused serial numbers, i.e., it is unique and no existing node can point to the new node as its predecessor. If Debt Accumulation or Debt Clearance is invoked, a new node is inserted that points to an existing node. Again, the serial number s of the new node is randomly chosen from the set of unused serial numbers, i.e., it is unique and no existing node can point to the new node as its predecessor. Hence, no cycle can be closed. Since the only incoming edge of a node is defined by the stated predecessor s^{prev} (which may also be $\bot$), each vertex has in-degree at most one. □

Lemma A.5 (Tree-Wise Uniqueness of the Wallet ID)**.** *The wallet ID λ maps one-to-one and onto a connected component (i.e., tree) of the Ideal Transaction Graph.*

Proof. " $\Longleftarrow$ ": Let $trdb_i$ be an arbitrary node in *TRDB* and λ be its wallet ID. Furthermore let $trdb_i^*$ be the root of the tree containing $trdb_i$. Then on the (unique) path from $trdb_i^*$ to $trdb_i$, every node apart from $trdb_i^*$ was inserted by means of either Debt Accumulation or Debt Clearance, both of which ensure the inserted node has the same λ as its predecessor. By induction over the length of the path, $trdb_i$ has the same wallet ID as $trdb_i^*$ and hence the wallet ID is a locally constant function on *TRDB*.

" $\Longrightarrow$ ": For contradiction assume there are two nodes $trdb_i$ and $trdb_j$ with equal wallet IDs $\lambda_i = \lambda_j$ in two different connected components. Pick the root nodes $trdb_i^*$ and $trdb_j^*$ of their respective trees. By " $\Longleftarrow$ " we get $\lambda_i^* = \lambda_i = \lambda_j = \lambda_j^*$, i.e., the root nodes have equals wallet IDs, too. Both root nodes are inserted in the scope of Wallet Issuing and the wallet ID is randomly drawn from the set of *unused* wallet IDs, i.e., they can not both have the same wallet ID. Contradiction! □

Lemma A.6. *Within a tree of the Ideal Transaction Graph, the PID $pid_{\mathcal{U}}$ of the corresponding user is constant.*

Proof. Same proof as " $\Longleftarrow$ " in the proof of Lemma A.5. ☐

In other words, Lemma A.6 states that a wallet (a tree in *TRDB*) is always owned by a distinct user. But a user can own multiple wallets.

Lemma A.7. *Within a tree of TRDB, every node $trdb = (s^{prev}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ has depth x and all nodes of the same depth in the same tree have the same fraud detection ID ϕ. Conversely, nodes with the same fraud detection ID are in the same tree and have the same depth within this tree.*

Proof. Proof by Induction: The statement is true for the empty *TRDB*. In the scope of Wallet Issuing a new root node is inserted, Wallet Issuing sets $x \coloneqq 0$ and an unused ϕ is chosen. In the scope of Debt Accumulation or Debt Clearance, x is calculated as $x \coloneqq x^{\text{prev}} + 1$, where by induction x^{prev} is the depth of its predecessor. With respect to ϕ we note that when inserted, every node gets as fraud detection ID the value stored in $f_{\Phi}(\lambda, x)$ which only depends on the node's wallet ID and depth. When this value is set (in either Wallet Issuing, Debt Accumulation, Debt Clearance or Blacklisting & Recalculation) it is chosen from the set of *unused* fraud detection IDs and therefore unique for given λ and x. ☐

Lemma A.8 (Billing Correctness)**.** *Let $trdb = (s^{prev}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ be an arbitrary but fixed node. If trdb is not a root, let $trdb^{prev} = (s^{prev,prev}, s^{prev}, \phi^{prev}, x^{prev}, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{prev}, p^{prev}, b^{prev})$ be its predecessor. Then $b = b^{prev} + p$ holds for non-root nodes and $p = \perp, b = 0$ for root nodes.*

Proof. Same induction argument as in the proof of Lemma A.7. ☐

Before we start to show that Π_{P4TC} correctly implements $\mathcal{F}_{\text{P4TC}}$, we make note of two additional simple statements about the ideal functionality itself.

Lemma A.9 (Protection Against False Accusation)**.**

(1) The task Double-Spending Detection returns a proof $\pi \neq \perp$ if and only if the user has committed double-spending.

(2) The task Verify Guilt returns OK if and only if its input $(pid_{\mathcal{U}}, \pi)$ has been output at a previous invocation of Double-Spending Detection.

Proof. The first part obviously follows by the definition of the task Double-Spending Detection (cp. Task Double-Spending Detection in Fig. 3.5). Note for the second part that users are assumed to be honest. If $f_{\Pi}(pid_{\mathcal{U}}, \pi)$ is undefined, then OUT is set to NOK and the result is recorded (cp. steps (3) and (4) in the Task Guilt Verification in Fig. 3.5). Guilt Verification only returns OK, if $f_{\Pi}(pid_{\mathcal{U}}, \pi) = \text{OK}$ has already been defined (cp. step (2)). As (in case of honest users) Guilt Verification extends f_{Π} by nothing but invalid proofs, Double-Spending Detection exclusively sets $f_{\Pi}(pid_{\mathcal{U}}, \pi) = \text{OK}$ (cp. step (2) of Task Double-Spending Detection Fig. 3.5). ☐

Lemma A.10 (Correctness of Blacklisting)**.** *Let $\mathcal{U}$ be an arbitrary but fixed user with $pid_{\mathcal{U}}$. Under the assumption that $\mathcal{U}$ participates in less then $x_{bl_{\mathcal{R}}}$ transactions, i.e., in less then $x_{bl_{\mathcal{R}}}$ invocations of Wallet Issuing, Debt Accumulation and Debt Clearance, the following two statements hold:*

(1) Let the TSP be honest. The set $\Phi_{\mathcal{U}}$ returned to the TSP by Blacklisting & Recalculation contains all fraud detection IDs that have ever been used by $\mathcal{U}$.

(2) Any invocation of Debt Accumulation for $\mathcal{U}$ with input $bl_{\mathcal{R}} = \Phi_{\mathcal{U}}$ aborts with message BLACKLISTED*.*

Proof. We prove both claims separately.

(1) Let $TRDB_{\mathcal{U}} \subseteq TRDB$ be the subset of all transaction entries $trdb = (\cdot, \cdot, \cdot, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, p, \cdot)$ corresponding to $pid_{\mathcal{U}}$ and let $\mathcal{L}_{\mathcal{U}}$ denote the set of wallet IDs occurring in $TRDB_{\mathcal{U}}$. For $\lambda \in \mathcal{L}_{\mathcal{U}}$ the depth of the tree associated to the wallet id λ is given by $x_\lambda \coloneqq \max\{x \mid f_\Phi(\lambda, x) \neq \bot\}$. If $\mathcal{U}$ with $pid_{\mathcal{U}}$ participated in less than $x_{\mathrm{bl}_{\mathcal{R}}}$ transaction, then the maximum depth $x_{\max} = \max_{\lambda \in \mathcal{L}_{\mathcal{U}}} x_\lambda$ is smaller than $x_{\mathrm{bl}_{\mathcal{R}}}$. The set of used fraud detection IDs is given by $\{f_\Phi(\lambda, x) \mid \lambda \in \mathcal{L}_{\mathcal{U}}, 0 \leq x \leq x_\lambda\}$ which is a subset of $\Phi_{\mathcal{U}} \coloneqq \{f_\Phi(\lambda, x) \mid \lambda \in \mathcal{L}_{\mathcal{U}}, 0 \leq x \leq x_{\mathrm{bl}_{\mathcal{R}}}\}$.

(2) Let s^{prev} denote the serial number for which Debt Accumulation is invoked and let $trdb^{\mathrm{prev}} = (\cdot, s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x^{\mathrm{prev}}, \lambda, \ldots)$ be the corresponding transaction entry. By assumption $x^{\mathrm{prev}} < x_{\mathrm{bl}_{\mathcal{R}}}$ holds. As Blacklisting & Recalculation has previously been called, $\phi = f_\Phi(\lambda, x^{\mathrm{prev}} + 1)$ is already fixed. Moreover, $\phi \in \Phi_{\mathcal{U}} = bl_{\mathcal{R}}$ holds and thus Debt Accumulation aborts.

□

A.1.2. Proof of Operator Security

In this appendix we show the following theorem.

Theorem A.11 (Operator Security)**.** *Under the assumptions of Theorem A.1*

$$\Pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{G}_{\mathrm{bb}}} \geq_{UC} \mathcal{F}_{\mathrm{P4TC}}^{\mathcal{G}_{\mathrm{bb}}}$$

holds under static corruption of

(1) a subset of users, or

(2) all users and a subset of RSUs, TSP and SA.

Before giving the UC-simulator $\mathcal{S}_{P4TC}^{\text{sys-sec}}$, a few remarks are in order. Please note that while the real protocol Π_{P4TC} lives in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{G}_{\mathrm{bb}})$-model the ideal functionality $\mathcal{F}_{\mathrm{P4TC}}$ has no CRS. Hence, the CRS (but not the bulletin board) is likewise simulated by $\mathcal{S}_{P4TC}^{\text{sys-sec}}$, giving it a lever to extract the ZK proofs P1, P2, and P3 and to equivoke the commitment C2.

While the protocol executes, the simulator $\mathcal{S}_{P4TC}^{\text{sys-sec}}$ records certain information similar to what the parties or the ideal functionality internally record, namely the set of simulated double-spending detection information $\overline{\Omega}^{\mathrm{dsp}}$, the set of simulated prove participation information $\overline{\Omega}_{\mathcal{U}}^{\mathrm{pp}}$, and the simulated transaction graph $\overline{TRDB}$. Basically, $\overline{\Omega}^{\mathrm{dsp}}$, $\overline{\Omega}_{\mathcal{U}}^{\mathrm{pp}}$ and $\overline{TRDB}$ correspond to Ω^{dsp}, $\Omega_{\mathcal{U}}^{\mathrm{pp}}$ and $TRDB$ resp., but exist in the head of the simulator and are augmented by additional information. The simulator uses them as "lookup tables" to keep up a consistent simulation in later parts of the protocol. Obviously, this implies information is stored redundantly: In the head of $\mathcal{S}_{P4TC}^{\text{sys-sec}}$ as $\overline{\Omega}_{\mathcal{U}}^{\mathrm{pp}}$ and $\overline{TRDB}$ and inside the ideal functionality $\mathcal{F}_{\text{P4TC}}$ (in case of *TRDB*) or the environment (in case of $\Omega^{\text{pp}}_{\mathcal{U}}$ for a corrupted user).[3] A crucial part of the security proof is to show that these sets stay in sync.

We now present the UC-simulator $\mathcal{S}^{\text{sys-sec}}_{P4TC}$ for Theorem A.11.

Simulator $\mathcal{S}^{\text{sys-sec}}_{P4TC}$ for System Security

System Setup:

(1) Set $\overline{TRDB} := \emptyset$

(2) Set $\overline{\Omega}^{\text{dsp}} := \emptyset$

(3) Set $\overline{\Omega}^{\text{pp}}_{\mathcal{U}} := \emptyset$

System Setup ($\mathcal{F}_{\text{CRS}}$):

(1) Run a modified version of $\mathcal{F}_{\text{CRS}}$ setup with
  (1) $\mathsf{crs}^2_{\mathsf{com}} \leftarrow \mathsf{C2.Gen}(\mathsf{gp})$ being replaced by $(\mathsf{crs}^2_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}) \leftarrow \mathsf{C2.SimGen}$, and
  (2) $\mathsf{crs}_{\mathsf{pok}} \leftarrow \mathsf{Setup}$ being replaced by $(\mathsf{crs}_{\mathsf{pok}}, \mathsf{td}^{\mathsf{ext}}_{\mathsf{pok}}) \leftarrow \mathsf{ExtSetup}$

(2) Record crs, $\mathsf{td}_{\mathsf{eqcom}}$ and $\mathsf{td}^{\mathsf{ext}}_{\mathsf{pok}}$

Simulation of $\mathcal{F}_{\text{CRS}}$:
Corrupt parties P can issue calls to $\mathcal{F}_{\text{CRS}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\text{CRS}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\text{CRS}}$ for all tasks in the following.

$\mathcal{S}$ answers all calls honestly.

DR Registration:

(1) Upon receiving (RegisteringDR, pid_{DR}) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate encryption keypair: $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}}) \leftarrow \mathsf{E.Gen}(\mathsf{gp})$
  (b) Record $pid_{\text{DR}} \mapsto (\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}})$
  (c) Return $\mathsf{pk}_{\mathsf{DR}}$ to $\mathcal{F}_{\text{P4TC}}$

TSP Registration (honest TSP):

(1) Upon receiving (RegisteringTSP, $pid_{\mathcal{T}}$, $\mathsf{a}_{\mathcal{T}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate all signing keys and the certificate:
    (i) $(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
    (ii) $(\mathsf{vk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
    (iii) $(\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
    (iv) $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) := \big((\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{vk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}), (\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}})\big)$
    (v) $\sigma^{\mathsf{cert}}_{\mathcal{T}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}, (\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}))$
    (vi) $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}} := (\mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}, \sigma^{\mathsf{cert}}_{\mathcal{T}})$
  (b) Record $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$
  (c) Return $\mathsf{pk}_{\mathcal{T}}$ to $\mathcal{F}_{\text{P4TC}}$

[3] Note, that we get rid of Ω^{dsp} during the proof, because honest users are only dummy parties and only $\overline{\Omega}^{\text{dsp}}$ remains.

TSP Registration (corrupted TSP):
Nothing to do as this is a local algorithm for a corrupted TSP

RSU Registration (honest RSU):

(1) Upon receiving (RegisteringRSU, $pid_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate signing keypair: $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
  (b) Record $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$
  (c) Return $\mathsf{vk}_{\mathcal{R}}$ to $\mathcal{F}_{\text{P4TC}}$

RSU Registration (corrupted RSU):
Nothing to do as this is a local algorithm for a corrupted RSU

User Registration (honest user):

(1) Upon receiving (RegisteringU, $pid_{\mathcal{U}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate keypair:
    (i) $y \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\mathsf{q}}$
    (ii) $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \coloneqq (g_1^y, y)$
  (b) Record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
  (c) Return $\mathsf{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$

User Registration (corrupted user):
Nothing to do as this is a local algorithm for a corrupted user

RSU Certification (honest TSP, honest RSU):

(1) Upon receiving (CertifyingRSU, $pid_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
  (b) Generate $\mathsf{cert}_{\mathcal{R}} \coloneqq (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\text{cert}})$ with $\sigma_{\mathcal{R}}^{\text{cert}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}_{\mathcal{T}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}))$ faithfully
  (c) Update record $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$

RSU Certification (honest TSP, corrupted RSU):

(1) Upon receiving (CertifyingRSU, $pid_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}})$, and obtain $\mathsf{vk}_{\mathcal{R}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{R}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
  (b) Generate $\mathsf{cert}_{\mathcal{R}} \coloneqq (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\text{cert}})$ with $\sigma_{\mathcal{R}}^{\text{cert}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}_{\mathcal{T}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}))$ faithfully
  (c) Record $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \perp, \mathsf{cert}_{\mathcal{R}})$
  (d) Output cert to $\mathcal{Z}^{\text{user-sec}}$ as message from the TSP to the RSU

RSU Certification (corrupted TSP, honest RSU):

(1) Upon receiving ($\mathsf{cert}_{\mathcal{R}}$) from $\mathcal{Z}^{\text{user-sec}}$ in the name of the TSP with $pid_{\mathcal{T}}$
  (a) Load the recorded $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{T}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
  (b) Parse $\mathsf{a}_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}^{\text{cert}}$ from $\mathsf{cert}_{\mathcal{R}}$
  (c) If $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\text{cert}}, \sigma_{\mathcal{R}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}})) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort
  (d) Call $\mathcal{F}_{\text{P4TC}}$ with input (Certify, $\mathsf{a}_{\mathcal{R}}$) in the name of the TSP with $pid_{\mathcal{T}}$

RSU Certification (corrupted TSP, corrupted RSU):
Nothing to do as $\mathcal{Z}^{\text{user-sec}}$ plays both parties

---

Wallet Issuing (corrupted user, honest TSP):

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, $pid_{\text{DR}} \mapsto (\mathsf{pk}_{\text{DR}}, \mathsf{sk}_{\text{DR}})$; if any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort
(2) Upon receiving $(\mathsf{pk}_{\mathcal{U}}, com'_{\text{seed}})$ from $\mathcal{Z}^{\text{sys-sec}}$ in the name of the user with $pid^*_{\mathcal{U}}$
  (a) Look up $pid_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$ for which $\mathsf{pk}_{\mathcal{U}}$ has been recorded; if no $pid_{\mathcal{U}}$ exists, then abort
  (b) If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$, then give up simulation
  (c) Call $\mathcal{F}_{\text{P4TC}}$ with input (ISSUE)
(3) Upon receiving the leak $(s, \mathsf{a}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$
  (a) $(com''_{\text{ser}}, \overline{decom}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{crs}^2_{\text{com}})$.
  (b) $\lambda'' \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$.
  (c) Send $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}, com''_{\text{ser}}, \lambda'')$ to $\mathcal{Z}^{\text{sys-sec}}$ as message from the TSP to the user.
(4) Upon receiving $(s', e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, \pi)$ from $\mathcal{Z}^{\text{sys-sec}}$ in the name of the user with $pid^*_{\mathcal{U}}$
  (a) $stmt \coloneqq (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\text{DR}}, e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, \pi)$
  (b) If $\mathsf{P1.Verify}(\mathsf{crs}_{\text{pok}}, stmt, \pi) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort
  (c) Extract witness as $WIT = (\Lambda, \Lambda', \Lambda'_0, \ldots, \Lambda'_{\ell-1}, \ldots, U^{\text{next}}_1, decom_{\mathcal{T}}, decom_{\mathcal{R}}, decom'_{\text{seed}}, \mathsf{SK}_{\mathcal{U}}) \leftarrow \mathsf{P1.ExtractW}(\mathsf{crs}, \mathsf{td}^{\text{ext}}_{\text{pok}}, stmt, \pi)$
  (d) Assert that $(stmt, WIT)$ fulfills the projected equations from $L^{(1)}_{\text{gp}}$, else abort (event *E1*)
  (e) $\lambda \coloneqq \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$
  (f) Provide alternative user PID $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$
(5) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide a fraud detection ID ϕ
  (a) $\phi \coloneqq \mathsf{PRF}(\lambda, x)$ with $x \coloneqq 0$
  (b) Provide ϕ to $\mathcal{F}_{\text{P4TC}}$
(6) Upon receiving output $(s, \mathsf{a}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ for the user
  (a) $s'' \coloneqq s \cdot s'^{-1}$
  (b) Equivoke $decom''_{\text{ser}} \leftarrow \mathsf{C2.Equiv}(\mathsf{crs}^2_{\text{com}}, \mathsf{td}_{\text{eqcom}}, s'', com''_{\text{ser}}, \overline{decom}_{\text{ser}})$
  (c) $\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (com_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}))$
  (d) $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (com_{\mathcal{R}}, s))$
  (e) Set $s^{\text{prev}} \coloneqq \bot$, $p \coloneqq 0$, $b \coloneqq 0$, $com^{\text{in}}_{\mathcal{T}} \coloneqq \bot$, $decom^{\text{in}}_{\mathcal{T}} \coloneqq \bot$, $M^{\text{in}}_{\mathcal{T}} \coloneqq \bot$, $com^{\text{in}}_{\mathcal{R}} \coloneqq \bot$, $decom^{\text{in}}_{\mathcal{R}} \coloneqq \bot$, $M^{\text{in}}_{\mathcal{R}} \coloneqq \bot$, $com^{\text{out}}_{\mathcal{T}} \coloneqq com_{\mathcal{T}}$, $decom^{\text{out}}_{\mathcal{T}} \coloneqq decom_{\mathcal{T}}$, $M^{\text{out}}_{\mathcal{T}} \coloneqq (\Lambda, \mathsf{pk}_{\mathcal{U}})$, $com^{\text{out}}_{\mathcal{R}} \coloneqq com_{\mathcal{R}}$, $decom^{\text{out}}_{\mathcal{R}} \coloneqq decom_{\mathcal{R}}$, and $M^{\text{out}}_{\mathcal{R}} \coloneqq (\Lambda, g^b_1, U_1, g^{x+1}_1)$
  (f) Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, p, b, com^{\text{in}}_{\mathcal{T}}, decom^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, com^{\text{in}}_{\mathcal{R}}, decom^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}, com^{\text{out}}_{\mathcal{T}}, decom^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, com^{\text{out}}_{\mathcal{R}}, decom^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})$ to $\overline{TRDB}$
  (g) Send $(s'', decom''_{\text{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}})$ to $\mathcal{Z}^{\text{sys-sec}}$ as message from the TSP to the user

Wallet Issuing (other corruption scenarios):
Nothing to do

Debt Accumulation (corrupted user, honest RSU):

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$; if any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort
(2) Pick $u_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$
(3) $(com''_{\text{ser}}, \overline{decom}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{crs}^2_{\text{com}})$

(4) Send $(u_2, com''_{\mathrm{ser}}, \mathrm{cert}_{\mathcal{R}})$ to $\mathcal{Z}$ as message from the RSU to the user

(5) Upon receiving $(s', \pi, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}}, com_{\mathrm{hid}}, com'_{\mathcal{R}}, t)$ from $\mathcal{Z}$ as message from the user to the RSU

  (a) $stmt := (\mathrm{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathrm{vk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}}, com_{\mathrm{hid}}, com'_{\mathcal{R}}, t, u_2)$

  (b) If $\mathsf{P2.Verify}(\mathrm{crs}_{\mathrm{pok}}, stmt, \pi) = 0$, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort

  (c) Extract witness as $WIT = (X, \Lambda, \mathrm{pk}_{\mathcal{U}}, U_1, s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, X, \Lambda, \mathrm{pk}_{\mathcal{U}}, B^{\mathrm{prev}}, U_1, U_1^{\mathrm{next}}, decom_{\mathrm{hid}}, decom^{\mathrm{prev}}_{\mathcal{R}}, decom'_{\mathcal{R}}, decom_{\mathcal{T}}, \mathrm{vk}^{\mathrm{prev}}_{\mathcal{R}}, com^{\mathrm{prev}}_{\mathcal{R}}, com_{\mathcal{T}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{certprev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) \leftarrow \mathsf{P2.ExtractW}(\mathrm{crs}, \mathrm{td}^{\mathrm{ext}}_{\mathrm{pok}}, stmt, \pi)$

  (d) Assert that $(stmt, WIT)$ fulfills the projected equations from $L^{(2)}_{\mathrm{gp}}$, else abort (event *E1*)

  (e) Look up $\overline{trdb}^{*} := (s^{\mathrm{prev},*}, s^{*}, \phi^{*}, x^{*}, \lambda^{*}, pid^{*}_{\mathcal{U}}, pid^{*}_{\mathcal{T}}, p^{*}, b^{*}, com^{\mathrm{in}\,*}_{\mathcal{T}}, decom^{\mathrm{in}\,*}_{\mathcal{T}}, M^{\mathrm{in}\,*}_{\mathcal{T}}, com^{\mathrm{in}\,*}_{\mathcal{R}}, decom^{\mathrm{in}\,*}_{\mathcal{R}}, M^{\mathrm{in}\,*}_{\mathcal{R}}, com^{\mathrm{out}\,*}_{\mathcal{T}}, decom^{\mathrm{out}\,*}_{\mathcal{T}}, M^{\mathrm{out}\,*}_{\mathcal{T}}, com^{\mathrm{out}\,*}_{\mathcal{R}}, decom^{\mathrm{out}\,*}_{\mathcal{R}}, M^{\mathrm{out}\,*}_{\mathcal{R}})$ with $s^{*} = s^{\mathrm{prev}}$ being used as key; if no unique entry exists, give up simulation (event *E2*)

  (f) Give up simulation if any of these conditions meet: $com^{\mathrm{out}\,*}_{\mathcal{R}} \neq com^{\mathrm{prev}}_{\mathcal{R}}$ (event *E3*), $\Lambda \neq g_1^{\lambda^{*}}$ (event *E4*), $com^{\mathrm{out}\,*}_{\mathcal{T}} \neq com_{\mathcal{T}}$ (event *E5*), $\mathrm{pk}_{\mathcal{U}} \neq \mathrm{pk}^{*}_{\mathcal{U}}$ with $(\Lambda^{*}, \mathrm{pk}^{*}_{\mathcal{U}}) := M^{\mathrm{out}\,*}_{\mathcal{T}}$ (event *E6*), $B^{\mathrm{prev}} \neq g_1^{b^{*}}$ (event *E7*), or $X \neq g_1^{x^{*}+1}$ (event *E8*).

  (g) Retrieve $pid_{\mathcal{U}}$ from $\mathcal{G}_{\mathrm{bb}}$ for $\mathrm{pk}_{\mathcal{U}}$

  (h) Call $\mathcal{F}_{\mathrm{P4TC}}$ with input $(\textsc{PayToll}, s^{\mathrm{prev}})$ in the name of the user

(6) Upon being asked by $\mathcal{F}_{\mathrm{P4TC}}$ to provide an alternative PID, return $pid_{\mathcal{U}}$ to $\mathcal{F}_{\mathrm{P4TC}}$

(7) If being asked by $\mathcal{F}_{\mathrm{P4TC}}$ to provide ϕ, return $\phi := \mathsf{PRF}(\lambda, x)$ with $x := x^{*} + 1$ to $\mathcal{F}_{\mathrm{P4TC}}$[a]

(8) Upon being asked by $\mathcal{F}_{\mathrm{P4TC}}$ to provide a price for $(\mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}})$, return a price p as the dummy adversary would do

(9) Upon receiving output $(s, \mathrm{a}_{\mathcal{R}}, p, b)$ from $\mathcal{F}_{\mathrm{P4TC}}$ for the user

  (a) Set $\overline{\omega}^{\mathrm{pp}}_{\mathcal{U}} := (s, com_{\mathrm{hid}}, decom_{\mathrm{hid}}, \mathrm{pk}_{\mathcal{U}})$ and append $\overline{\omega}^{\mathrm{pp}}_{\mathcal{U}}$ to $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$

  (b) Run $(com''_{\mathcal{R}}, decom''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathrm{crs}^{1}_{\mathrm{com}}, (0, p, 0, 1))$, $com_{\mathcal{R}} := com'_{\mathcal{R}} \cdot com''_{\mathcal{R}}$, and $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sign}(\mathrm{sk}_{\mathcal{R}}, (com_{\mathcal{R}}, s))$ honestly as the real protocol would do

  (c) Set $s'' := s \cdot s'^{-1}$ and equivoke $decom''_{\mathrm{ser}} \leftarrow \mathsf{C2.Equiv}(\mathrm{crs}^{2}_{\mathrm{com}}, s'', com''_{\mathrm{ser}}, \overline{decom}_{\mathrm{ser}})$

  (d) Set $com^{\mathrm{in}}_{\mathcal{T}} := com_{\mathcal{T}}$, $decom^{\mathrm{in}}_{\mathcal{T}} := decom_{\mathcal{T}}$, $M^{\mathrm{in}}_{\mathcal{T}} := (\Lambda, \mathrm{pk}_{\mathcal{U}})$, $com^{\mathrm{in}}_{\mathcal{R}} := com^{\mathrm{prev}}_{\mathcal{R}}$, $decom^{\mathrm{in}}_{\mathcal{R}} := decom^{\mathrm{prev}}_{\mathcal{R}}$, $M^{\mathrm{in}}_{\mathcal{R}} := (\Lambda, B^{\mathrm{prev}}, U_1, X)$, $com^{\mathrm{out}}_{\mathcal{T}} := com_{\mathcal{T}}$, $decom^{\mathrm{out}}_{\mathcal{T}} := decom'_{\mathcal{T}} \cdot decom''_{\mathcal{T}}$, $M^{\mathrm{out}}_{\mathcal{T}} := (\Lambda, \mathrm{pk}_{\mathcal{U}})$, $com^{\mathrm{out}}_{\mathcal{R}} := com_{\mathcal{R}}$, $decom^{\mathrm{out}}_{\mathcal{R}} := decom'_{\mathcal{R}} \cdot decom''_{\mathcal{R}}$, and $M^{\mathrm{out}}_{\mathcal{R}} := (\Lambda, g_1^{b}, U_1, g_1^{x+1})$

  (e) Append $(s^{\mathrm{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b, com^{\mathrm{in}}_{\mathcal{T}}, decom^{\mathrm{in}}_{\mathcal{T}}, M^{\mathrm{in}}_{\mathcal{T}}, com^{\mathrm{in}}_{\mathcal{R}}, decom^{\mathrm{in}}_{\mathcal{R}}, M^{\mathrm{in}}_{\mathcal{R}}, com^{\mathrm{out}}_{\mathcal{T}}, decom^{\mathrm{out}}_{\mathcal{T}}, M^{\mathrm{out}}_{\mathcal{T}}, com^{\mathrm{out}}_{\mathcal{R}}, decom^{\mathrm{out}}_{\mathcal{R}}, M^{\mathrm{out}}_{\mathcal{R}})$ to $\overline{TRDB}$

  (f) Append $\overline{\omega}^{\mathrm{dsp}} = (\phi, t, u_2)$ to $\overline{\Omega}^{\mathrm{dsp}}$

  (g) Check if $\overline{\omega}^{\mathrm{dsp}\,\ddagger} = (\phi^{\ddagger}, t^{\ddagger}, u_2^{\ddagger}) \in \overline{\Omega}^{\mathrm{dsp}}$ exists with $\phi = \phi^{\ddagger}$ and $u_2 \neq u_2^{\ddagger}$; in this case

    (i) $\mathrm{sk}_{\mathcal{U}} := (t - t^{\ddagger}) \cdot (u_2 - u_2^{\ddagger})^{-1} \bmod \mathrm{q}$

    (ii) Record $pid_{\mathcal{U}} \mapsto (\mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}})$ internally

  (h) Send $(s'', decom''_{\mathrm{ser}}, com_{\mathcal{R}}, decom''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to $\mathcal{Z}$ as message from the RSU to the user

Debt Accumulation (other corruption scenarios):
Nothing to do

Debt Clearance (corrupted user, honest TSP):

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathrm{pk}_{\mathcal{T}}, \mathrm{sk}_{\mathcal{T}}, \mathrm{cert}^{\mathcal{R}}_{\mathcal{T}})$ if this does not exist, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort.

(2) Pick $u_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathrm{q}}$

(3) Send u_2 to $\mathcal{Z}$ as message from the TSP to the user

(4) Upon receiving $(\mathrm{pk}_{\mathcal{U}}, \pi, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}}, b^{\mathrm{prev}}, t)$ from $\mathcal{Z}$ as message from the user to the TSP

  (a) $stmt := (\mathrm{pk}_{\mathcal{U}}, \mathrm{vk}^{\mathcal{T}}_{\mathcal{T}}, \mathrm{vk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}}, g_1^{b^{\mathrm{prev}}}, t, u_2)$

  (b) If $\mathsf{P3.Verify}(\mathrm{crs}_{\mathrm{pok}}, stmt, \pi) = 0$, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort

(c) Extract witness as $WIT = (X, \Lambda, \text{pk}_{\mathcal{U}}, U_1, s^{\text{prev}}, \phi^{\text{prev}}, X, \Lambda, U_1, decom_{\mathcal{R}}^{\text{prev}}, decom_{\mathcal{T}}, \text{vk}_{\mathcal{R}}^{\text{prev}}, com_{\mathcal{R}}^{\text{prev}}, com_{\mathcal{T}}, \sigma_{\mathcal{R}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{certprev}}, \sigma_{\mathcal{T}}) \leftarrow \text{P3.ExtractW}(\text{crs}, \text{td}_{\text{pok}}^{\text{ext}}, stmt, \pi)$

(d) Assert that $(stmt, WIT)$ fulfills the projected equations from $L_{\text{gp}}^{(3)}$, else abort (event *E1*)

(e) Look up $\overline{trdb}^{*} := (s^{\text{prev},*}, s^{*}, \phi^{*}, x^{*}, \lambda^{*}, pid_{\mathcal{U}}^{*}, pid_{\mathcal{T}}^{*}, p^{*}, b^{*}, com_{\mathcal{T}}^{\text{in}*}, decom_{\mathcal{T}}^{\text{in}*}, M_{\mathcal{T}}^{\text{in}*}, com_{\mathcal{R}}^{\text{in}*}, decom_{\mathcal{R}}^{\text{in}*}, M_{\mathcal{R}}^{\text{in}*}, com_{\mathcal{T}}^{\text{out}*}, decom_{\mathcal{T}}^{\text{out}*}, M_{\mathcal{T}}^{\text{out}*}, com_{\mathcal{R}}^{\text{out}*}, decom_{\mathcal{R}}^{\text{out}*}, M_{\mathcal{R}}^{\text{out}*})$ with $s^{*} = s^{\text{prev}}$ being used as key; if no unique entry exists, give up simulation (event *E2*)

(f) Give up simulation if any of these conditions meet: $com_{\mathcal{R}}^{\text{out}*} \neq com_{\mathcal{R}}^{\text{prev}}$ (event *E3*), $\Lambda \neq g_1^{\lambda^{*}}$ (event *E4*), $com_{\mathcal{T}}^{\text{out}*} \neq com_{\mathcal{T}}$ (event *E5*), $\text{pk}_{\mathcal{U}} \neq \text{pk}_{\mathcal{U}}^{*}$ with $(\Lambda^{*}, \text{pk}_{\mathcal{U}}^{*}) := M_{\mathcal{T}}^{\text{out}*}$ (event *E6*), or $B^{\text{prev}} \neq g_1^{b^{*}}$ (event *E7*)

(g) Retrieve $pid_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$ for $\text{pk}_{\mathcal{U}}$

(h) Call $\mathcal{F}_{\text{P4TC}}$ with input (ClearDebt, s^{prev}) in the name of the user

(5) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide an alternative PID, return $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$

(6) If being asked by $\mathcal{F}_{\text{P4TC}}$ to provide ϕ, return $\phi := \text{PRF}(\lambda, x)$ with $x := x^{\text{prev}} + 1$ to $\mathcal{F}_{\text{P4TC}}$[a]

(7) Upon receiving output (b^{bill}) from $\mathcal{F}_{\text{P4TC}}$ for the user

(a) Set $com_{\mathcal{T}}^{\text{in}} := com_{\mathcal{T}}$, $decom_{\mathcal{T}}^{\text{in}} := decom_{\mathcal{T}}$, $M_{\mathcal{T}}^{\text{in}} := (\Lambda, \text{pk}_{\mathcal{U}})$, $com_{\mathcal{R}}^{\text{in}} := com_{\mathcal{R}}^{\text{prev}}$, $decom_{\mathcal{R}}^{\text{in}} := decom_{\mathcal{R}}^{\text{prev}}$, $M_{\mathcal{R}}^{\text{in}} := (\Lambda, g_1^{b^{\text{prev}}}, U_1, X)$, $com_{\mathcal{T}}^{\text{out}} := \bot$, $decom_{\mathcal{T}}^{\text{out}} := \bot$, $M_{\mathcal{T}}^{\text{out}} := \bot$, $com_{\mathcal{R}}^{\text{out}} := \bot$, $decom_{\mathcal{R}}^{\text{out}} := \bot$, and $M_{\mathcal{R}}^{\text{out}} := \bot$

(b) Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b, com_{\mathcal{T}}^{\text{in}}, decom_{\mathcal{T}}^{\text{in}}, M_{\mathcal{T}}^{\text{in}}, com_{\mathcal{R}}^{\text{in}}, decom_{\mathcal{R}}^{\text{in}}, M_{\mathcal{R}}^{\text{in}}, com_{\mathcal{T}}^{\text{out}}, decom_{\mathcal{T}}^{\text{out}}, M_{\mathcal{T}}^{\text{out}}, com_{\mathcal{R}}^{\text{out}}, decom_{\mathcal{R}}^{\text{out}}, M_{\mathcal{R}}^{\text{out}})$ to $\overline{TRDB}$

(c) Append $\overline{\omega}^{\text{dsp}} = (\phi, t, u_2)$ to $\overline{\Omega}^{\text{dsp}}$

(d) Check if $\overline{\omega}^{\text{dsp}\ddagger} = (\phi^{\ddagger}, t^{\ddagger}, u_2^{\ddagger}) \in \overline{\Omega}^{\text{dsp}}$ exists with $\overline{\phi} = \overline{\phi}^{\ddagger}$ and $u_2 \neq u_2^{\ddagger}$; in this case

(i) $\text{sk}_{\mathcal{U}} := (t - t^{\ddagger}) \cdot (u_2 - u_2^{\ddagger})^{-1} \bmod \text{q}$

(ii) Record $pid_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$ internally

(e) Send (OK) to $\mathcal{Z}$ as message from the TSP to the user

Debt Clearance (other corruption scenarios):
Nothing to do

Prove Participation (corrupted user, honest SA):

(1) Call $\mathcal{F}_{\text{P4TC}}$ with input (ProveParticipation) in the name of the user

(2) Obtain leaked set $S_{\mathcal{R}}^{\text{pp}}$ of serial numbers from $\mathcal{F}_{\text{P4TC}}$

(3) Upon receiving output ($\text{OUT}_{\mathcal{U}}$) from $\mathcal{F}_{\text{P4TC}}$ for the user

(a) Delay the output of $\mathcal{F}_{\text{P4TC}}$ to the SA

(b) Send $S_{\mathcal{R}}^{\text{pp}}$ to $\mathcal{Z}$ as message from the SA to the user

(4) Upon receiving $(s, com_{\text{hid}}, decom_{\text{hid}})$ from $\mathcal{Z}$ as message from the user to the SA

(a) If $(s, com_{\text{hid}}, \cdot, \cdot) \notin \overline{\Omega}_{\mathcal{U}}^{\text{pp}}$, then let $\mathcal{F}_{\text{P4TC}}$ abort

(b) Look up $\text{pk}_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$; if no $\text{pk}_{\mathcal{U}}$ has been recorded, then abort

(c) If $\text{C1.Open}(\text{crs}_{\text{com}}^{1}, \text{pk}_{\mathcal{U}}, com_{\text{hid}}, decom_{\text{hid}}) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort

(d) If $\text{OUT}_{\mathcal{U}} = \text{NOK}$, then give up simulation

(e) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the SA

Prove Participation (other corruption scenarios):
Nothing to do

Double-Spending Detection (honest TSP):

(1) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide a proof for $pid_{\mathcal{U}}$
  (a) Look up $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
  (b) Return $\mathsf{sk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$

Double-Spending Detection (corrupted TSP):
Nothing to do as this is a local algorithm for a corrupted TSP

Guilt Verification (honest P):

(1) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide OUT for $(pid_{\mathcal{U}}, \pi)$
  (a) Receive $\mathsf{pk}_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{U}}$
  (b) If $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$, then set OUT $\coloneqq$ OK, else set OUT $\coloneqq$ NOK
  (c) Return OUT to $\mathcal{F}_{\text{P4TC}}$

Guilt Verification (corrupted P):
Nothing to do as this is a local algorithm for a corrupted P

Blacklisting & Recalculation (corrupted TSP):

(1) Upon receiving $HTD_{\mathcal{U}}$ from $\mathcal{Z}^{\text{user-sec}}$ as message from the TSP to the DR
  (a) Parse $HTD_{\mathcal{U}}$ to get a set of user public keys. If they are not all the same, abort. If they are all the same, denote this key with $\mathsf{pk}_{\mathcal{U}}$.
  (b) Call $\mathcal{F}_{\text{P4TC}}$ with input (BlacklistUser, $\mathsf{pk}_{\mathcal{U}}$) in the name of the TSP
(2) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to decide on the output for both parties
  (a) Set (OK) as DR output and $(\bot, \bot)$ as TSP output
  (b) Let $\mathcal{F}_{\text{P4TC}}$ deliver output

Blacklisting & Recalculation (honest TSP):

(1) Upon receiving (λ, x) from $\mathcal{F}_{\text{P4TC}}$ and being asked to provide a serial number
  (a) Return $\phi \coloneqq \mathsf{PRF}(\lambda, x)$ to $\mathcal{F}_{\text{P4TC}}$

[a] N.b.: $\mathcal{F}_{\text{P4TC}}$ does not always ask for the next serial number. If the corrupted user re-uses an old token, then $\mathcal{F}_{\text{P4TC}}$ internally picks the next serial number which has already been determined in some earlier interaction. Hence, the simulator only needs to provide the next serial number if the chain of transactions is extended.

Before starting with the security proof we explain the *Simulated Transaction Graph* $\overline{TRDB}$ and the additional information beyond the Ideal Transaction Graph (cp. Definition A.3) in more details. The *Ideal Transaction Graph* is a theoretical construct that helps us to link the interactions of the parties across the various tasks our protocol provides. An *Simulated Transaction Entry* $\overline{trdb}$ has the form

$$\begin{aligned} \overline{trdb} = (&s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b, \\ &com_{\mathcal{T}}^{\text{in}}, decom_{\mathcal{T}}^{\text{in}}, M_{\mathcal{T}}^{\text{in}}, com_{\mathcal{R}}^{\text{in}}, decom_{\mathcal{R}}^{\text{in}}, M_{\mathcal{R}}^{\text{in}}, \\ &com_{\mathcal{T}}^{\text{out}}, decom_{\mathcal{T}}^{\text{out}}, M_{\mathcal{T}}^{\text{out}}, com_{\mathcal{R}}^{\text{out}}, decom_{\mathcal{R}}^{\text{out}}, M_{\mathcal{R}}^{\text{out}}) \end{aligned} \tag{A.1}$$

with c, d and M with equal suffixes denoting a commitment, its decommitment information and the opening in the implicit message space (see Fig. A.2). At the beginning of a transaction in the scope of Debt Accumulation or Debt Clearance, the user loads their token τ^{prev} which contains two commitments $com_{\mathcal{T}}$ and $com_{\mathcal{R}}^{\text{prev}}$, randomizes the commitments and at the end the user possesses two

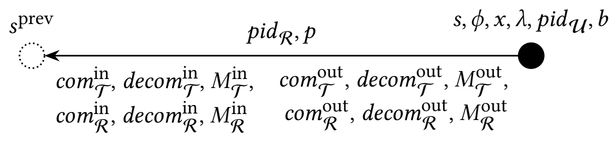

s^{prev}
$pid_{\mathcal{R}}, p$
$s, \phi, x, \lambda, pid_{\mathcal{U}}, b$
$com_{\mathcal{T}}^{\text{in}}, decom_{\mathcal{T}}^{\text{in}}, M_{\mathcal{T}}^{\text{in}},$
$com_{\mathcal{R}}^{\text{in}}, decom_{\mathcal{R}}^{\text{in}}, M_{\mathcal{R}}^{\text{in}}$
$com_{\mathcal{T}}^{\text{out}}, decom_{\mathcal{T}}^{\text{out}}, M_{\mathcal{T}}^{\text{out}},$
$com_{\mathcal{R}}^{\text{out}}, decom_{\mathcal{R}}^{\text{out}}, M_{\mathcal{R}}^{\text{out}}$

Figure A.2.: An entry $\overline{trdb} \in \overline{TRDB}$ visualized as an element of a directed graph

updated commitments $com_{\mathcal{T}}$, $com_{\mathcal{R}}$ which are stored in τ again. We call the initial commitments the *in*-commitments of the transaction and the resulting commitments the *out*-commitments.

Definition A.12 (Simulated Transaction Graph (Informal))**.** The set $\overline{TRDB} = \{\overline{trdb}_i\}$ with $\overline{trdb}_i$ defined as in Eq. (A.1) is called the *Simulated Transaction Graph*. It inherits the graph structure of the Ideal Transaction Graph and augments each edge by additional labels, called the *in-commitments* and *out-commitments*.

Two remarks are in order: Firstly, none of the (commitment, decommitment, message)-triples is neither completely received nor sent by the RSU or TSP, respectively. The RSU receives a randomized version of the in-commitment and no decommitment at all. In the reverse direction, the RSU sends the out-commitment and a share of the decommitment. The complete triples only exist inside the user's token. Secondly, it is tempting but misleading to assume that $com_{\mathcal{R}}^{\text{in}} = com_{\mathcal{R}}^{\text{prev}}$ (or similar equations) hold. Note that we do not make any of these assumptions for the definition. Hence we decided on a new notion and coined the term in-/out-commitments instead of re-using the term "previous commitment". Actually, these kind of equalities is what we have to show.

The overall proof idea is to define a sequence of hybrid experiments H_i together with simulators $\mathcal{S}_i$ and protocols Π_i such that the first hybrid H_0 is identical to the real experiment and the last hybrid H_{16} is identical to the ideal experiment. Each hybrid is of the form

$$\mathsf{H}_i := \mathsf{EXEC}_{\Pi_i, \mathcal{G}_{\text{bb}}, \mathcal{S}_i, \mathcal{Z}^{\text{sys-sec}}}(1^\lambda).$$

Instead of directly proving indistinguishability of the real and ideal experiment we can break the proof down into showing indistinguishability of each pair of consecutive hybrids. We achieve this by demonstrating that whenever $\mathcal{Z}^{\text{sys-sec}}$ can distinguish between two consecutive hybrids with non-negligible probability this yields an efficient adversary against one of the underlying cryptographic assumptions. The general idea is that the protocol Π_i that honest parties perform gradually declines from the real protocol $\Pi_0 = \Pi_{\text{P4TC}}$ to a dummy protocol Π_{16}, which does nothing but relay in- and outputs. At the same time $\mathcal{S}_i$ progresses from a dummy adversary $\mathcal{S}_0$ to the final simulator $\mathcal{S}_{16}$ which can be split up into the ideal functionality $\mathcal{F}_{\text{P4TC}}$ and $\mathcal{S}_{P4TC}^{\text{sys-sec}}$.

We proceed by giving concrete (incremental) definitions of all hybrids H_i. Please note that *input privacy* for honest TSP, RSU, DR and SA does not pose a difficulty for the definition of the sequence of the simulators. The users learns most information as part of their prescribed output anyway. In other words, the simulator $\mathcal{S}_{P4TC}^{\text{sys-sec}}$ mimicking the role of an honest operator can perfectly simulate most messages towards the (malicious) user after it has received the user's output from the ideal functionality. The essential part is to ensure that no malicious user can make the Simulated Transaction Graph to deviate from the Ideal Transaction Graph and thereby cause a different (wrong) output at some later point in the protocol. To this end, most hybrids introduce additional "sanity checks" to the simulation: if the sanity check holds, both transaction graphs are still in sync and the simulator proceeds; if the sanity check fails, the adversary has caused the transaction graphs to fall apart and the simulator immediately gives up the simulation. Each sanity check is related to the security of one of the building blocks or cryptographic assumptions. Finally, after the last hybrid, all sanity checks collectively assert that no efficient adversary can deviate from the Ideal Transaction Graph.

Hybrid H_0 The hybrid H_0 is defined as

$$\mathsf{H}_0 \coloneqq \mathsf{EXEC}_{\Pi_0,\mathcal{G}_{\mathrm{bb}},\mathcal{S}_0,\mathcal{Z}^{\text{sys-sec}}}(1^\lambda)$$

with $\mathcal{S}_0 = \mathcal{A}$ being identical to the dummy adversary and $\Pi_0 = \Pi_{\mathrm{P4TC}}$. Hence, H_0 denotes the real experiment.

Hybrid H_1 In hybrid H_1 we modify $\mathcal{S}_1$ such that $\mathsf{crs}_{\mathrm{pok}}$ is generated by $\mathsf{ExtSetup}$, and $\mathsf{crs}^2_{\mathrm{com}}$ is generated by $\mathsf{C2.SimGen}$. Additionally, $\mathcal{S}_1$ initializes the internal sets $\overline{TRDB}$, $\overline{\Omega}^{\mathrm{dsp}}$, and $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$ as empty sets.

Hybrid H_2 Hybrid H_2 replaces the code in the tasks DR/TSP/RSU/User Registration of the protocol Π_2 such that the simulator $\mathcal{S}_2$ is asked for the keys instead. This equals the method in which the keys are generated in the ideal experiment.

Hybrid H_3 In hybrid H_3 the task RSU Certification is modified. The protocol Π_3 is modified such that the simulator $\mathcal{S}_3$ receives the message $(\textsc{CertifyingRSU}, pid_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$, creates the certificate $\sigma^{\mathrm{cert}}_{\mathcal{R}}$ and records it.

Whenever the honest TSP or honest RSU running Π_3 would send $\sigma^{\mathrm{cert}}_{\mathcal{R}}$ as part of their messages in the scope of Wallet Issuing or Debt Accumulation, they omit $\sigma^{\mathrm{cert}}_{\mathcal{R}}$. Instead, the simulator $\mathcal{S}_3$ injects $\sigma^{\mathrm{cert}}_{\mathcal{R}}$ into the message.

Hybrid H_4 Hybrid H_4 replaces the code in the tasks Wallet Issuing and Debt Accumulation of the protocol Π_4 such that the RSU/TSP do not create signatures, but the simulator $\mathcal{S}_4$ creates the signatures $\sigma_{\mathcal{T}}$, $\sigma_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}$ resp. and injects them into the messages instead.

Moreover, in Debt Accumulation the RSU running Π_4 does not send $com_{\mathcal{R}}$ and $decom''_{\mathcal{R}}$ in its final message, but reports the price p to $\mathcal{S}_4$, $\mathcal{S}_4$ creates $com_{\mathcal{R}}$ and $decom''_{\mathcal{R}}$ honestly and injects them into the message.

Hybrid H_5 Hybrid H_5 modifies the tasks of Wallet Issuing and Debt Accumulation. The code of Π_5 for the TSP/RSU is modified such that it does not send com''_{ser} in the scope of Wallet Issuing or Debt Accumulation. Instead $\mathcal{S}_5$ runs $(com''_{\mathrm{ser}}, \overline{decom}_{\mathrm{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{crs}^2_{\mathrm{com}})$ and injects com''_{ser} into the message. Moreover, Π_5 for the TSP/RSU is modified such that it uniformly and independently picks $s \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$ and passes s to $\mathcal{S}_5$ as part of the final message. $\mathcal{S}_5$ calculates $s'' \coloneqq s \cdot (s')^{-1}$, executes $decom''_{\mathrm{ser}} \leftarrow \mathsf{C2.Equiv}(\mathsf{crs}^2_{\mathrm{com}}, \mathsf{td}_{\mathrm{eqcom}}, s'', com''_{\mathrm{ser}}, \overline{decom}_{\mathrm{ser}})$ and injects s'' together with $decom''_{\mathrm{ser}}$ into the messages from TSP/RSU to the user.

Hybrid H_6 When $\mathcal{S}_6$ receives a NIZK proof π in the scope of Wallet Issuing, Debt Accumulation and Debt Clearance, it extracts the witness, restores $\lambda \coloneqq \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$, assembles $\overline{trdb}$ and appends it to $\overline{TRDB}$. Additionally, $\mathcal{S}_6$ also assembles $\overline{\omega}^{\mathrm{pp}}_{\mathcal{U}}$, $\overline{\omega}^{\mathrm{dsp}}$ in the scope of Debt Accumulation and appends entries to $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$, $\overline{\Omega}^{\mathrm{dsp}}$ resp. If $\overline{\Omega}^{\mathrm{dsp}}$ already contains an entry $\overline{\omega}^{\mathrm{dsp}^{\ddagger}}$ with matching fraud detection ID, the secret key $\mathsf{sk}_{\mathcal{U}}$ is immediately reconstructed and the pair $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ is also recorded.

Moreover, the verification of the proof is moved from Π_6 for the honest TSP/RSU to the simulator. If the verification fails, $\mathcal{S}_6$ aborts as the TSP/RSU running the real protocol would do.

Additionally, $\mathcal{S}_6$ checks if the pair of the statement and the extracted witness fulfills the languages $L^{(1)}_{\mathrm{gp}}$, $L^{(2)}_{\mathrm{gp}}$, and $L^{(3)}_{\mathrm{gp}}$ resp. If not, $\mathcal{S}_6$ abort with failure event (*E1*).

Hybrid H_7 This hybrid modifies the code Π_7 for TSP, SA and DR in the scope of the tasks Prove Participation, Double-Spending Detection, Guilt Verification and Blacklisting & Recalculation. The honest parties become dummy parties, the code is moved to the simulator and $\mathcal{S}_7$ resorts to its "lookup tables" $\overline{TRDB}$, $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$, and $\overline{\Omega}^{\text{dsp}}$ that have been introduced by the previous hybrid.

More precisely, in Prove Participation the SA becomes a dummy party and simply forwards the set of serial numbers $S^{\text{pp}}_{\mathcal{R}}$ to $\mathcal{S}_7$. The simulator uses its own set $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ to validate the response of the environment (in the name of the malicious user) and returns the result to the SA.

In the task Double-Spending Detection the honest TSP becomes a dummy party, too. It simply asks the simulator $\mathcal{S}_7$ to provide a proof. To this end, $\mathcal{S}_7$ checks if $pid_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$ has been recorded and returns $\text{sk}_{\mathcal{U}}$.

The same applies to the task Guilt Verification. The honest party does not locally run the algorithm itself, but simply forwards its input to the simulator (as the dummy party would do) and $\mathcal{S}_7$ actually checks if $g_1^{\pi} = \pi$ holds.

The task Blacklisting & Recalculation is modified accordingly. The dispute resolver DR becomes a dummy party and simply sends it input $(\textsc{BlacklistUser}, \text{pk}^{\text{DR}}_{\mathcal{U}})$ to the simulator $\mathcal{S}_7$ in order to signal its consent to blacklist the user. The simulator $\mathcal{S}_7$ utilizes the Simulated Transaction Graph $\overline{TRDB}$ and runs the code as the ideal functionality $\mathcal{F}_{\text{P4TC}}$ would do eventually.

Hybrid H_8 Hybrid H_8 replaces the code in the tasks Wallet Issuing and Debt Accumulation of the protocol Π_8 such that the RSU/TSP do not neither send λ'' nor u_2. Instead $\mathcal{S}_8$ draws λ'' and u_2 and injects them into the appropriate messages. Consequently, the code of the TSP is modified such that it does not longer record htd. Likewise, the code of the RSU is modified such that it does not longer record ω^{dsp}.

Hybrid H_9 In the scope of Debt Accumulation or Debt Clearance, the simulator $\mathcal{S}_9$ looks up the predecessor entry with s^{prev} being used as the unique key. If this fails, $\mathcal{S}_9$ gives up the simulation with event *E2*.

Hybrid H_{10} The simulator $\mathcal{S}_{10}$ additionally checks for $com^{\text{out}*}_{\mathcal{R}} \neq com^{\text{prev}}_{\mathcal{R}}$. If the check succeeds, it gives up the simulation with event *E3*.

Hybrid H_{11} The simulator $\mathcal{S}_{11}$ additionally checks for $\Lambda \neq g_1^{\lambda^*}$. If the check succeeds, it gives up the simulation with event *E4*.

Hybrid H_{12} The simulator $\mathcal{S}_{12}$ additionally checks for $com^{\text{out}*}_{\mathcal{T}} \neq com_{\mathcal{T}}$. If the check succeeds, it gives up the simulation with event *E5*.

Hybrid H_{13} The simulator $\mathcal{S}_{13}$ parses $(\Lambda^*, \text{pk}^*_{\mathcal{U}}) \coloneqq M^{\text{out}*}_{\mathcal{T}}$ and checks for $\text{pk}_{\mathcal{U}} \neq \text{pk}^*_{\mathcal{U}}$. If the check succeeds, it gives up the simulation with event *E6*.

Hybrid H_{14} The simulator $\mathcal{S}_{14}$ additionally checks for $B^{\text{prev}} \neq g_1^{b^*}$. If the check succeeds, it gives up the simulation with event *E7*.

Hybrid H_{15} The simulator $\mathcal{S}_{15}$ additionally checks for $X \neq g_1^{x^*+1}$. If the check succeeds, it gives up the simulation with event *E8*.

For the proof of Theorem A.11 we show the indistinguishability of subsequent hybrids by a series of lemmas. The Lemmas A.13 to A.15 are rather trivial and thus Lemma A.14 handles various hybrids at once.

Lemma A.13 (Indistinguishability between H_0 and H_1)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_0 \overset{c}{\equiv} \mathsf{H}_1$ holds.*

Proof. This hop solely changes how the crs is created during the setup phase. This is indistinguishable for $\mathsf{crs}_{\mathsf{pok}}$ (see the extractability property of Definition 2.28, Item (3a)) and $\mathsf{crs}^2_{\mathsf{com}}$ (see the equivocality property of Definition 2.24, Item (3a)). ☐

Lemma A.14 (Indistinguishability $\mathsf{H}_1 \rightarrow \mathsf{H}_4$ and $\mathsf{H}_6 \rightarrow \mathsf{H}_8$)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_1 \overset{c}{\equiv} \mathsf{H}_2$, $\mathsf{H}_2 \overset{c}{\equiv} \mathsf{H}_3$, $\mathsf{H}_3 \overset{c}{\equiv} \mathsf{H}_4$, $\mathsf{H}_6 \overset{c}{\equiv} \mathsf{H}_7$, and $\mathsf{H}_7 \overset{c}{\equiv} \mathsf{H}_8$ holds.*

Proof. The hops are all indistinguishable as they do not change anything in the view of $\mathcal{Z}^{\text{sys-sec}}$. Please note that $\mathcal{Z}^{\text{sys-sec}}$ only sees the in-/output of honest parties and these hops only syntactically change what parts of the code are executed by the parties or by the simulator. With each hop the parties degrade more to a dummy party while at the same time more functionality is put into the simulator. ☐

Lemma A.15 (Indistinguishability between H_4 and H_5)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_4 \overset{c}{\equiv} \mathsf{H}_5$ holds.*

Proof. This hop is indistinguishable as the equivoked decommitment information is perfectly indistinguishable from a decommitment that has originally been created with the correct message (cp. Definition 2.24, Item (3)). ☐

So far, none of hops between two consecutive hybrids changes anything from the environment's perspective: either the hops are only syntactical or the modification is perfectly indistinguishable. Hence, no reduction argument is required. In the contrary, each of the upcoming security proofs roughly follows the same lines of argument. If the environment $\mathcal{Z}^{\text{sys-sec}}$ can efficiently distinguish between two consecutive hybrids, then we can construct an efficient adversary $\mathcal{B}$ against one of the underlying cryptographic building blocks. To this end, $\mathcal{B}$ plays the adversary against the binding property in the outer game and internally executes the UC-experiment in its head while mimicking the role of the simulator. It is important to note that although $\mathcal{B}$ emulates the environment internally, it only has *black-box access* to it. In other words, although everything happens inside "the head of $\mathcal{B}$" it cannot somehow magically extract $\mathcal{Z}$'s attack strategy.

Lemma A.16 (Indistinguishability between H_5 and H_6)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_5 \overset{c}{\equiv} \mathsf{H}_6$ holds.*

Proof. First note that the only effective change between H_5 and H_6 are the additional checks that abort the simulation with event *E1*, if the extracted witnesses are invalid. Again, the other modification are purely syntactical. To proof indistinguishability between H_5 and H_6 we split this hop into three sub-hybrids. Each sub-hybrid introduces the check for one of the languages $L^{(1)}_{\text{gp}}$, $L^{(2)}_{\text{gp}}$ and $L^{(3)}_{\text{gp}}$, resp. In the following only the sub-hybrid for the language $L^{(1)}_{\text{gp}}$ is considered, the indistinguishability of the remaining two is proved analogously. Further note, that the view of $\mathcal{Z}^{\text{sys-sec}}$ is perfectly indistinguishable, if the simulation does not abort.

Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that triggers the event *E1* in the first sub-hybrid with non-negligible advantage. This immediately yields an efficient adversary $\mathcal{B}$ against the extraction property of the NIZK scheme. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the adversary in Definition 2.28, Item (3b). If the event *E1* occurs internally, $\mathcal{B}$ outputs the corresponding pair $(stmt, \pi)$. In the second and third sub-hybrid $\mathcal{B}$ internally extracts the witness for the previous sub-hybrid using the extraction trapdoor $\mathsf{td}^{\mathrm{ext}}_{\mathrm{pok}}$ which $\mathcal{B}$ obtains as part of its input. □

Remark A.17. We observe that Lemma A.16 implies
with $m = (\Lambda, \mathsf{pk}_{\mathcal{U}})$:

$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, m, com_{\mathcal{T}}, decom_{\mathcal{T}}) = 1,$$

with $m = (\Lambda, 1, U_1^{\mathrm{next}}, g_1)$:

$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, m, com_{\mathcal{R}}, decom_{\mathcal{R}}) = 1,$$

with $m = (\Lambda, B^{\mathrm{prev}}, U_1, X)$:

$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, m, com_{\mathcal{R}}^{\mathrm{prev}}, decom_{\mathcal{R}}^{\mathrm{prev}}) = 1,$$

with $m = (\Lambda, B^{\mathrm{prev}}, U_1^{\mathrm{next}}, X)$:

$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, m, com'_{\mathcal{R}}, decom'_{\mathcal{R}}) = 1,$$

furthermore

$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, \Lambda', com'_{\mathrm{seed}}, decom'_{\mathrm{seed}}) = 1,$$
$$\mathsf{C1.Open}(\mathsf{crs}^1_{\mathrm{com}}, \mathsf{pk}_{\mathcal{U}}, com_{\mathrm{hid}}, decom_{\mathrm{hid}}) = 1,$$

with $m = (com_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})$:

$$\mathsf{S.Vfy}(\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, m) = 1,$$

with $m = (com^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}})$:

$$\mathsf{S.Vfy}(\mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, m) = 1,$$

with $m = (\mathsf{vk}^{\mathrm{prev}}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}})$:

$$\mathsf{S.Vfy}(\mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert\,prev}}_{\mathcal{R}}, m) = 1,$$

and that all variables can efficiently be extracted. Remember, that F_{gp} acts as the identity function on group elements. Moreover, given the extracted chunks of the wallet ID $\Lambda'_0, \ldots, \Lambda'_{\ell-1}$ the unique wallet ID λ can be reconstructed. The projection F_{gp} becomes injective if the pre-image is restricted to $\mathbb{Z}_q$ and the inverse, i.e., DLOG, can be efficiently computed as $\lambda'_0, \ldots, \lambda'_{\ell-1}$ are sufficiently "small".

Up to this point, we already know that $\mathsf{H}_0 \overset{c}{\equiv} \mathsf{H}_8$ holds. Except for two small changes (from H_4 to H_5 and from H_5 to H_6) all hops are only syntactical. Moreover, the simulator $\mathcal{S}_8$ of hybrid H_8 is indeed sufficient to simulate an indistinguishable view for $\mathcal{Z}^{\mathrm{sys\text{-}sec}}$ in the ideal model. Note, that all subsequent hybrids from H_{10} to H_{15} only add more sanity checks but do not change any messages. Actually, even the modification introduced by H_6 is not required for a indistinguishable simulation, as H_6 only records $\overline{TRDB}$, but $\overline{TRDB}$ is not used yet. However, only $\overline{TRDB}$ and the upcoming sanity checks enable a reduction to cryptographic assumptions and thus are vital to proof the indistinguishably between H_8 and the ideal model.

To this end, two additional lemmas about the structure of $\overline{TRDB}$ are necessary. These lemmas are in the same spirit as Lemmas A.4 and A.5. Intuitively, the commitments $com_{\mathcal{T}}$, $com_{\mathcal{R}}$ induce a graph structure onto $\overline{TRDB}$ comparable to the wallet ID λ and serial number s.

Lemma A.18 (Simulated Transaction Forest)**.**

(1) Every $\overline{trdb} = (s^{prev}, s, \dots) \in \overline{TRDB}$ is uniquely identified by s with overwhelming probability.

(2) The Simulated Transaction Graph TRDB is a forest with edges defined by (s^{prev}, s).

Proof. We prove both claims separately.

(1) A new entry is only inserted in the scope of Wallet Issuing, Debt Accumulation or Debt Clearance. Proof by Induction: The statement is correct for the empty $\overline{TRDB}$. For each insertion, the simulator $\mathcal{S}_6$ (and every following simulator) draws s uniformly and independently. The chance to pick a serial number that has already been used is negligible.

(2) As the serial number s of the new node is randomly chosen, no existing node can point to the new node as its predecessor and thus no cycle is closed with overwhelming probability.

□

Lemma A.19 (Indistinguishability between H_8 and H_9)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_8 \overset{c}{\equiv} \mathsf{H}_9$ holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that triggers the event *E2* with non-negligible advantage. This immediately yields an efficient adversary $\mathcal{B}$ against the EUF-CMA-security of S. We only need to deal with the case that s^* does not exist. If it exists, Lemma A.18, Item (1) implies its uniqueness. We need to distinguish two cases. On an abstract level these cases correspond to the following scenarios: Either the previous RSU exists. Then the signature $\sigma_{\mathcal{R}}^{\text{prev}}$ on $(com_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ is a forgery. Or alternatively, the allegedly previous RSU does not exits but has been imagined by the user. Then $(com_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ may have a honest, valid signature (because the user feigned the RSU), but the certificate $\mathsf{cert}_{\mathcal{R}}{}^{\text{prev}}$ for the fake RSU constitutes a forgery. Please note that the simulator always records an entry $\overline{trdb}$ when it legitimately issues a signature $\sigma_{\mathcal{R}}$ and vice versa.

(1) *A record $pid_{\mathcal{R}}^{prev} \mapsto (\mathsf{vk}_{\mathcal{R}}^{prev}, \mathsf{sk}_{\mathcal{R}}^{prev})$ has been recorded:*
In other words, $(com_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ has never been legitimately issued by the allegedly previous RSU.[4] We construct an efficient adversary $\mathcal{B}$ against the EUF-CMA-security of S. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head and plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the EUF-CMA-security experiment with a challenger C and a signing oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. $\mathcal{B}$ needs to guess for which $pid_{\mathcal{R}}^{\text{prev}}$ the event $(E2)$ eventually occurs. When the RSU with $pid_{\mathcal{R}}^{\text{prev}}$ registers itself, and $\mathcal{B}$ playing $\mathcal{S}_9$ needs to provide $\mathsf{vk}_{\mathcal{R}}^{\text{prev}}$ it embeds the challenge as $\mathsf{vk}_{\mathcal{R}}^{\text{prev}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}_9$ needs to issue a signature with respect to $\mathsf{vk}_{\mathcal{R}}^{\text{prev}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. When the event $(E2)$ occurs, $\mathcal{B}$ extracts $(com_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ and $\sigma_{\mathcal{R}}^{\text{prev}}$ from the proof and outputs the forgery. Note that $(com_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ has never been signed with respect to $\mathsf{vk}_{\mathcal{R}}^{\text{prev}} = \mathsf{pk}_C$ by assumption.

[4] Note: RSU may also denote the TSP, if the transaction at hand happens to be the first after a Wallet Issuing and thus s^* has been signed by the TSP playing the role of an RSU. For brevity, we only consider RSUs here.

(2) *A record* $pid_{\mathcal{R}}^{prev} \mapsto (\mathsf{vk}_{\mathcal{R}}^{prev}, \mathsf{sk}_{\mathcal{R}}^{prev}, \mathsf{cert}_{\mathcal{R}})$ *has not been recorded:*
We construct an efficient adversary $\mathcal{B}$ against the EUF-CMA-security of S along the same lines as above. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head and plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the EUF-CMA-security experiment with a challenger C and a signing oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. When the adversary $\mathcal{B}$ has to internally provide $\mathsf{pk}_{\mathcal{T}} = (\mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{vk}_{\mathcal{T}}^{\mathcal{T}})$ playing the role of $\mathcal{S}_9$ in the scope of the TSP Registration, $\mathcal{B}$ embeds the external challenge as $\mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}} \coloneqq \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}_9$ in the scope of RSU Certification needs to issue signatures with respect to $\mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. When the event ($E2$) occurs, $\mathcal{B}$ extracts $\mathsf{cert}_{\mathcal{R}}^{\mathrm{prev}} = (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\mathsf{cert}})$ from the proof and outputs $(\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$ together with $\sigma_{\mathcal{R}}^{\mathsf{cert}}$ as the forgery. Note that $(\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$ has never been signed by the TSP with respect to $\mathsf{vk}_{\mathcal{T}}^{\mathsf{cert}} = \mathsf{pk}_C$ as otherwise a mapping $pid_{\mathcal{R}}^{\mathrm{prev}} \mapsto (\mathsf{vk}_{\mathcal{R}}^{\mathrm{prev}}, \mathsf{sk}_{\mathcal{R}}^{\mathrm{prev}}, \mathsf{cert}_{\mathcal{R}})$ would have been recorded.

The forgeries are indeed valid due to Remark A.17. □

Remark A.20. Without Lemma A.19 it is unclear in Lemma A.18, Item (2) if the denoted predecessor of edge (s^{prev}, s) actually exists. The simulator extracts the serial number s^{prev} of the predecessor from the proof and puts this serial number into the newly added $\overline{trdb}$. With this in mind Lemma A.18, Item (2) would have to be interpreted such that an edge (s^{prev}, s) is ignored, if the predecessor did not exist. Nonetheless, $\overline{TRDB}$ is still a forest and Lemma A.18, Item (2) remains correct. Anyway, this oddity is ruled out by Lemma A.19.

Lemma A.21 (Indistinguishability between H_9 and H_{10}). *Under the assumptions of Theorem A.11,* $\mathsf{H}_9 \overset{c}{\equiv} \mathsf{H}_{10}$ *holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that triggers the event $E3$ with non-negligible advantage. This immediately yields an efficient adversary $\mathcal{B}$ against the EUF-CMA-security of S by the same argument as in the proof of Lemma A.19 as $(com_{\mathcal{R}}^{\mathrm{prev}}, s^{\mathrm{prev}})$ are jointly signed by the same signature $\sigma_{\mathcal{R}}$. □

Lemma A.22 (Indistinguishability between H_{10} and H_{11}). *Under the assumptions of Theorem A.11,* $\mathsf{H}_{10} \overset{c}{\equiv} \mathsf{H}_{11}$ *holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that triggers the event $E4$ with non-negligible advantage. We construct an efficient adversary $\mathcal{B}$ against the binding property of C1. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head and plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the role of the adversary as defined by Definition 2.24, Item (2). When the event ($E3$) occurs, $\mathcal{B}$ sets

$$M_{\mathcal{R}}^{\mathrm{prev}} \coloneqq (\Lambda, B^{\mathrm{prev}}, U_1, X)$$

from the extracted witness and obtains

$$M_{\mathcal{R}}^{\mathrm{out}*} = (\Lambda^*, B^*, U_1^*, X^*)$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(com_{\mathcal{R}}^{\mathrm{out}*}, M_{\mathcal{R}}^{\mathrm{prev}}, decom_{\mathcal{R}}^{\mathrm{prev}}, M_{\mathcal{R}}^{\mathrm{out}*}, decom_{\mathcal{R}}^{\mathrm{out}*})$ to the external game. By assumption $\Lambda \neq \Lambda^*$ holds and Remark A.17 asserts that both openings are valid. □

Lemma A.23 (Tree-Wise Uniqueness of the Wallet ID). *The wallet ID λ maps one-to-one and onto a connected component (i.e., tree) of the Simulated Transaction Graph.*

Proof. Same argument as in the proof of Lemma A.5. □

Lemma A.24 (Indistinguishability between H_{11} and H_{12})**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_{11} \overset{c}{\equiv} \mathsf{H}_{12}$ holds.*

Proof. We introduce a sub-hybrid that splits between two cases why event *E5* is triggered:

(1) $com_{\mathcal{T}}^{\mathrm{out}*} \neq com_{\mathcal{T}}$ and $com_{\mathcal{T}}$ is not recorded in any $\overline{trdb} \in \overline{TRDB}$

(2) $com_{\mathcal{T}}^{\mathrm{out}*} \neq com_{\mathcal{T}}$ and $com_{\mathcal{T}}$ is recorded in some record $\overline{trdb}^{\ddagger} \in \overline{TRDB}$

An environment $\mathcal{Z}^{\text{sys-sec}}$ that can differentiate between H_{11} and the sub-hybrid yields an efficient adversary $\mathcal{B}$ against the EUF-CMA security of S. An environment $\mathcal{Z}^{\text{sys-sec}}$ that can differentiate between the sub-hybrid and H_{12} yields an efficient adversary $\mathcal{B}$ against the binding property of $\mathsf{C1}$.

(1) We construct an efficient adversary $\mathcal{B}$ against the EUF-CMA-security of S. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head, and plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the EUF-CMA-security experiment with a challenger C and a signing oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. When $\mathcal{B}$ must internally provide $\mathsf{pk}_{\mathcal{T}} = (\mathsf{vk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{vk}^{\mathcal{T}}_{\mathcal{T}})$ playing the role of $\mathcal{S}_{12}$ in the scope of the TSP Registration, $\mathcal{B}$ embeds the external challenge as $\mathsf{vk}^{\mathcal{T}}_{\mathcal{T}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}_{12}$ needs to issue signatures with respect to $\mathsf{pk}_{\mathcal{T}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\Sigma}_{\mathsf{pk},\mathsf{sk}}$. When the event (*E5*) occurs, $\mathcal{B}$ extracts $com_{\mathcal{T}}$ and $\sigma_{\mathcal{T}}$ from the proof and outputs the forgery.

(2) We construct an efficient adversary $\mathcal{B}$ against the binding property of $\mathsf{C1}$. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head and plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the role of the adversary as defined by Definition 2.24, Item (2). As (*E5*) has not been raised earlier, $com_{\mathcal{T}}^{\mathrm{out}(i)} = com_{\mathcal{T}}^{\mathrm{out}*} \neq com_{\mathcal{T}}$ holds for all $com_{\mathcal{T}}^{\mathrm{out}(i)}$ in the same tree. Consequently, $\overline{trdb}^{\ddagger}$ with $com_{\mathcal{T}}^{\mathrm{out}\ddagger} = com_{\mathcal{T}}$ is part of a different tree in $\overline{TRDB}$ and thus $\Lambda^{\ddagger} \neq \Lambda^{*} = \Lambda$ follows by Lemma A.23. $\mathcal{B}$ sets

$$M_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$$

from the extracted witness and obtains

$$M_{\mathcal{T}}^{\mathrm{out}\ddagger} = (\Lambda^{\ddagger}, \mathsf{pk}^{\ddagger}_{\mathcal{U}})$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(com_{\mathcal{T}}, M_{\mathcal{T}}, decom_{\mathcal{T}}, M_{\mathcal{T}}^{\mathrm{out}\ddagger}, decom_{\mathcal{T}}^{\mathrm{out}\ddagger})$ to the external game.

Remark A.17 asserts that the forgery in (1) and both openings in (2) are indeed valid. □

Lemma A.25 (Indistinguishability $\mathsf{H}_{12} \to \mathsf{H}_{15}$)**.** *Under the assumptions of Theorem A.11, $\mathsf{H}_{12} \overset{c}{\equiv} \mathsf{H}_{13} \overset{c}{\equiv} \mathsf{H}_{14} \overset{c}{\equiv} \mathsf{H}_{15}$ holds.*

Proof. If an environment $\mathcal{Z}^{\text{sys-sec}}$ can distinguish between any of the hops from H_{12} to H_{15} this yields an efficient adversary $\mathcal{B}$ against the binding property of $\mathsf{C1}$. As usual, $\mathcal{B}$ runs $\mathcal{Z}^{\text{sys-sec}}$ in its head and internally plays the role of the simulator and all honest parties. Externally, $\mathcal{B}$ plays the role of the adversary as defined by Definition 2.24, Item (2). If event $(E7)$ or $(E8)$ occurs, $\mathcal{B}$ sets

$$M_{\mathcal{R}}^{\text{prev}} = (\Lambda, B^{\text{prev}}, U_1, g_1 X)$$

from the extracted witness and obtains

$$M_{\mathcal{R}}^{\text{out}*} := (\Lambda^*, B^*, U_1^*, X^*)$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(com_{\mathcal{R}}, M_{\mathcal{R}}^{\text{prev}}, decom_{\mathcal{R}}^{\text{prev}}, M_{\mathcal{R}}^{\text{out}*}, decom_{\mathcal{R}}^{\text{out}*})$ to the external game. If the event $(E6)$ is triggered, $\mathcal{B}$ proceeds analogous but for the fixed part of wallet $com_{\mathcal{T}}$. □

Taking all the aforementioned statements together, Theorem A.11 from the beginning of this appendix follows. For the sake of formal completeness we recall it again.

Theorem A.11 (Operator Security). *Under the assumptions of Theorem A.1*

$$\Pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}}} \geq_{UC} \mathcal{F}_{\text{P4TC}}^{\mathcal{G}_{\text{bb}}}$$

holds under static corruption of

(1) a subset of users, or

(2) all users and a subset of RSUs, TSP and SA.

Proof of Theorem A.11. A direct consequence of Lemmas A.13 to A.16, A.19, A.21, A.22, A.24 and A.25. □

A.1.3. Proof of User Security and Privacy

In this appendix we show the following theorem.

Theorem A.26 (User Security and Privacy). *Under the assumptions of Theorem A.1*

$$\Pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}}} \geq_{UC} \mathcal{F}_{\text{P4TC}}^{\mathcal{G}_{\text{bb}}}$$

holds under static corruption of

(1) a subset of RSUs, TSP and SA, or

(2) all RSUs, TSP and SA as well as a subset of users.

Before giving the UC-simulator $\mathcal{S}_{P4TC}^{\text{user-sec}}$, a remark is in order. Please note that while the real protocol Π_{P4TC} lives in the $(\mathcal{F}_{\text{CRS}}, \mathcal{G}_{\text{bb}})$-model, the ideal functionality $\mathcal{F}_{\text{P4TC}}$ has no CRS. Hence, the CRS (but not the bulletin board) is likewise simulated, providing $\mathcal{S}_{P4TC}^{\text{user-sec}}$ with a lever to simulate the ZK proofs P1, P2, and P3, to equivoke C1, and to extract C2.

We now present the UC-simulator $\mathcal{S}_{P4TC}^{\text{user-sec}}$ for Theorem A.26.

Simulator $\mathcal{S}^{\text{user-sec}}_{P4TC}$ for User Security and Privacy

System Setup:

(1) Set $\overline{\Omega}^{\text{dsp}} \coloneqq \emptyset$
(2) Set $\overline{\Omega}^{\text{pp}}_{\mathcal{U}} \coloneqq \emptyset$
(3) Set $\overline{HTD} \coloneqq \emptyset$

System Setup ($\mathcal{F}_{\text{CRS}}$):

(1) Run a modified version of $\mathcal{F}_{\text{CRS}}$ setup with
  (a) $\text{crs}^1_{\text{com}} \leftarrow \text{C1.Gen}$ being replaced by $(\text{crs}^1_{\text{com}}, \text{td}_{\text{eqcom}}) \leftarrow \text{C1.SimGen}$,
  (b) $\text{crs}^2_{\text{com}} \leftarrow \text{C2.Gen}$ being replaced by $(\text{crs}^2_{\text{com}}, \text{td}_{\text{extcom}}) \leftarrow \text{C2.ExtGen}$, and
  (c) $\text{crs}_{\text{pok}} \leftarrow \text{Setup}$ being replaced by $(\text{crs}_{\text{pok}}, \text{td}^{\text{sim}}_{\text{pok}}) \leftarrow \text{SimSetup}$
(2) Record crs, td_{eqcom}, $\text{td}_{\text{extcom}}$, and $\text{td}^{\text{sim}}_{\text{pok}}$

Simulation of $\mathcal{F}_{\text{CRS}}$:
Corrupt parties P can issue calls to $\mathcal{F}_{\text{CRS}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\text{CRS}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\text{CRS}}$ for all tasks in the following.
$\mathcal{S}$ answers all calls honestly.

DR Registration:

(1) Upon receiving (RegisteringDR, pid_{DR}) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate encryption keypair: $(\text{pk}_{\text{DR}}, \text{sk}_{\text{DR}}) \leftarrow \text{E.Gen}(\text{gp})$
  (b) Record $pid_{\text{DR}} \mapsto (\text{pk}_{\text{DR}}, \text{sk}_{\text{DR}})$
  (c) Return pk_{DR} to $\mathcal{F}_{\text{P4TC}}$

TSP Registration (honest TSP):

(1) Upon receiving (RegisteringTSP, $pid_{\mathcal{T}}$, $\text{a}_{\mathcal{T}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate all signing keys and the certificate:
    (i) $(\text{vk}^{\mathcal{T}}_{\mathcal{T}}, \text{sk}^{\mathcal{T}}_{\mathcal{T}}) \leftarrow \text{S.Gen}(\text{gp})$
    (ii) $(\text{vk}^{\text{cert}}_{\mathcal{T}}, \text{sk}^{\text{cert}}_{\mathcal{T}}) \leftarrow \text{S.Gen}(\text{gp})$
    (iii) $(\text{vk}^{\mathcal{R}}_{\mathcal{T}}, \text{sk}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \text{S.Gen}(\text{gp})$
    (iv) $(\text{pk}_{\mathcal{T}}, \text{sk}_{\mathcal{T}}) \coloneqq \big((\text{vk}^{\mathcal{T}}_{\mathcal{T}}, \text{vk}^{\text{cert}}_{\mathcal{T}}, \text{vk}^{\mathcal{R}}_{\mathcal{T}}), (\text{sk}^{\mathcal{T}}_{\mathcal{T}}, \text{sk}^{\text{cert}}_{\mathcal{T}}, \text{sk}^{\mathcal{R}}_{\mathcal{T}})\big)$
    (v) $\sigma^{\text{cert}}_{\mathcal{T}} \leftarrow \text{S.Sign}(\text{sk}^{\text{cert}}_{\mathcal{T}}, (\text{vk}^{\mathcal{R}}_{\mathcal{T}}, \text{a}_{\mathcal{T}}))$
    (vi) $\text{cert}^{\mathcal{R}}_{\mathcal{T}} \coloneqq (\text{vk}^{\mathcal{R}}_{\mathcal{T}}, \text{a}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{T}})$
  (b) Record $pid_{\mathcal{T}} \mapsto (\text{pk}_{\mathcal{T}}, \text{sk}_{\mathcal{T}}, \text{cert}^{\mathcal{R}}_{\mathcal{T}})$
  (c) Return $\text{pk}_{\mathcal{T}}$ to $\mathcal{F}_{\text{P4TC}}$

TSP Registration (corrupted TSP):
Nothing to do as this is a local algorithm for a corrupted TSP

RSU Registration (honest RSU):

(1) Upon receiving (RegisteringRSU, $pid_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
  (a) Honestly generate signing keypair: $(\text{vk}_{\mathcal{R}}, \text{sk}_{\mathcal{R}}) \leftarrow \text{S.Gen}(\text{gp})$
  (b) Record $pid_{\mathcal{R}} \mapsto (\text{vk}_{\mathcal{R}}, \text{sk}_{\mathcal{R}})$

(c) Return $\mathsf{vk}_{\mathcal{R}}$ to $\mathcal{F}_{\text{P4TC}}$

RSU Registration (corrupted RSU):
Nothing to do as this is a local algorithm for a corrupted RSU

User Registration (honest user):

(1) Upon receiving (RegisteringU, $pid_{\mathcal{U}}$) from $\mathcal{F}_{\text{P4TC}}$
   (a) Honestly generate keypair:
      (i) $y \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$
      (ii) $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) := (g_1^y, y)$
   (b) Record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
   (c) Return $\mathsf{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$

User Registration (corrupted user):
Nothing to do as this is a local algorithm for a corrupted user

RSU Certification (honest TSP, honest RSU):

(1) Upon receiving (CertifyingRSU, $pid_{\mathcal{R}}$, $\mathbf{a}_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
   (a) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
   (b) Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\text{cert}})$ with $\sigma_{\mathcal{R}}^{\text{cert}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}_{\mathcal{T}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$ faithfully
   (c) Update record $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$

RSU Certification (honest TSP, corrupted RSU):

(1) Upon receiving (CertifyingRSU, $pid_{\mathcal{R}}$, $\mathbf{a}_{\mathcal{R}}$) from $\mathcal{F}_{\text{P4TC}}$
   (a) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}})$, and obtain $\mathsf{vk}_{\mathcal{R}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{R}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
   (b) Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\text{cert}})$ with $\sigma_{\mathcal{R}}^{\text{cert}} \leftarrow \mathsf{S.Sign}(\mathsf{sk}_{\mathcal{T}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$ faithfully
   (c) Record $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \bot, \mathsf{cert}_{\mathcal{R}})$
   (d) Output cert to $\mathcal{Z}^{\text{user-sec}}$ as message from the TSP to the RSU

RSU Certification (corrupted TSP, honest RSU):

(1) Upon receiving ($\mathsf{cert}_{\mathcal{R}}$) from $\mathcal{Z}^{\text{user-sec}}$ in the name of the TSP with $pid_{\mathcal{T}}$
   (a) Load the recorded $pid_{\mathcal{R}} \mapsto (\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{T}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
   (b) Parse $\mathbf{a}_{\mathcal{R}}$ and $\sigma_{\mathcal{R}}^{\text{cert}}$ from $\mathsf{cert}_{\mathcal{R}}$
   (c) If $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{T}}^{\text{cert}}, \sigma_{\mathcal{R}}^{\text{cert}}, (\mathsf{vk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort
   (d) Call $\mathcal{F}_{\text{P4TC}}$ with input (Certify, $\mathbf{a}_{\mathcal{R}}$) in the name of the TSP with $pid_{\mathcal{T}}$

RSU Certification (corrupted TSP, corrupted RSU):
Nothing to do as $\mathcal{Z}^{\text{user-sec}}$ plays both parties

---

Wallet Issuing (corrupted TSP, honest user):

(1) Load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{T}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.
(2) $(com'_{\text{seed}}, decom'^{\text{sim}}_{\text{seed}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{crs}^1_{\text{com}})$

```
(3) Send $(\mathrm{pk}_{\mathcal{U}}, com'_{\mathrm{seed}})$ to $\mathcal{Z}^{\text{user-sec}}$ as message from the user to the TSP
(4) Upon receiving $(\mathrm{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathrm{a}_{\mathcal{U}}, com''_{\mathrm{ser}}, \lambda'')$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of the TSP with $pid_{\mathcal{T}}$ [a]
    (a) Parse $(\mathrm{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathrm{a}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}) := \mathrm{cert}^{\mathcal{R}}_{\mathcal{T}}$
    (b) If $\mathsf{S.Vfy}(\mathrm{vk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}, (\mathrm{vk}^{\mathcal{R}}_{\mathcal{T}}, \mathrm{a}_{\mathcal{T}})) = 0$, then abort
    (c) $\Lambda'' := g_1^{\lambda''}$
    (d) $s'' \leftarrow \mathsf{C2.Extract}(\mathrm{crs}^2_{\mathrm{com}}, \mathrm{td}_{\mathrm{extcom}}, com''_{\mathrm{ser}})$
    (e) Call $\mathcal{F}_{\mathrm{P4TC}}$ with input $(\textsc{Issue}, \mathrm{a}_{\mathcal{U}}, \emptyset)$ in the name of the TSP with $pid_{\mathcal{T}}$ [b]
(5) Upon receiving output $(s)$ from $\mathcal{F}_{\mathrm{P4TC}}$ for the TSP
    (a) $s' := s \cdot s''^{-1}$
    (b) $r_1, r_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$
    (c) $e^* \leftarrow \mathsf{E.Enc}(\mathrm{pk}_{\mathrm{DR}}, \underbrace{(1, \dots, 1)}_{\ell+2}; r_1, r_2)$
    (d) $(com_{\mathcal{T}}, decom_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathrm{crs}^1_{\mathrm{com}}, (0, 0))$
    (e) $(com_{\mathcal{R}}, decom_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathrm{crs}^1_{\mathrm{com}}, (0, 0, 0, 0))$
    (f) $stmt := (\mathrm{pk}_{\mathcal{U}}, \mathrm{pk}_{\mathrm{DR}}, e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, com'_{\mathrm{seed}}, \Lambda'', \lambda'')$
    (g) $\pi \leftarrow \mathsf{P1.SimProve}(\mathrm{crs}_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}, stmt)$
    (h) Send $(s', e^*, com_{\mathcal{T}}, com_{\mathcal{R}}, \pi)$ to $\mathcal{Z}^{\text{user-sec}}$ as message from the user to the TSP
(6) Upon receiving $(s'', decom''_{\mathrm{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}})$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of the TSP with $pid_{\mathcal{T}}$
    (a) If $\mathsf{C2.Open}(\mathrm{crs}^2_{\mathrm{com}}, s'', com''_{\mathrm{ser}}, decom''_{\mathrm{ser}}) = 0$, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort
    (b) Set $\overline{htd} := (\mathrm{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ and insert $\overline{htd}$ into $\overline{HTD}$
    (c) Create real token $\tau$ faithfully
    (d) If $\mathsf{WalletVerification}(\mathrm{pk}_{\mathcal{T}}, \mathrm{pk}_{\mathcal{U}}, \tau) = 0$, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort
    (e) Let $\mathcal{F}_{\mathrm{P4TC}}$ return the delayed output to the user
```

Wallet Issuing (other corruption scenarios):
Nothing to do

Debt Accumulation (corrupted RSU, honest user):

```
(1) Obtain $\mathrm{pk}_{\mathcal{T}}$ from $\mathcal{G}_{\mathrm{bb}}$ for $pid_{\mathcal{T}}$. If it does not exist, then let $\mathcal{F}_{\mathrm{P4TC}}$ abort.
(2) Upon receiving $u_2, com''_{\mathrm{ser}}, \mathrm{cert}_{\mathcal{R}}$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of the RSU with $pid_{\mathcal{R}}$
    (a) Parse $(\mathrm{vk}_{\mathcal{R}}, \mathrm{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathrm{cert}_{\mathcal{R}}$
    (b) If $\mathsf{S.Vfy}(\mathrm{vk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}, (\mathrm{vk}_{\mathcal{R}}, \mathrm{a}_{\mathcal{R}})) = 0$, then abort
    (c) $s'' \leftarrow \mathsf{C2.Extract}(\mathrm{crs}^2_{\mathrm{com}}, com''_{\mathrm{ser}})$
    (d) Call $\mathcal{F}_{\mathrm{P4TC}}$ with input $(\textsc{PayToll}, \emptyset)$ [c] in the name of the RSU with $pid_{\mathcal{R}}$
    (e) Obtain RSU's output $(s, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}})$ from $\mathcal{F}_{\mathrm{P4TC}}$, and delay the output of the user
    (f) $s' := s \cdot s''^{-1}$
(3) Upon being requested by $\mathcal{Z}^{\text{user-sec}}$ to provide the second message
    (a) Run $(com_{\mathrm{hid}}, \overline{decom_{\mathrm{hid}}}) \leftarrow \mathsf{C1.SimCom}(\mathrm{crs}^1_{\mathrm{com}})$ and append $(s, com_{\mathrm{hid}}, \overline{decom_{\mathrm{hid}}})$ to $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$
    (b) $(com'_{\mathcal{R}}, decom'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathrm{crs}^1_{\mathrm{com}}, (0, 0, 0, 0))$
    (c) Check if any $(\phi, t', u'_2) \in \overline{\Omega}^{\mathrm{dsp}}$ has been recorded previously with $(\phi)$ being used as key. If no, pick
        $t \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$. If yes, load the recorded $pid_{\mathcal{U}} \mapsto (\mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}})$ and set $t := t' + \mathrm{sk}_{\mathcal{U}}(u_2 - u'_2)$. Insert $(\phi, t, u_2)$
        into $\overline{\Omega}^{\mathrm{dsp}}$.
    (d) $stmt := (\mathrm{pk}_{\mathcal{T}}, \mathrm{vk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\mathrm{prev}}_{\mathcal{R}}, com_{\mathrm{hid}}, com'_{\mathcal{R}}, t, u_2)$
    (e) $\pi \leftarrow \mathsf{P2.SimProve}(\mathrm{crs}_{\mathrm{pok}}, \mathrm{td}^{\mathrm{sim}}_{\mathrm{pok}}, stmt)$
```

(f) Output $(s', \pi, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}}^{\text{prev}}, \mathit{com}_{\text{hid}}, \mathit{com}'_{\mathcal{R}}, t)$ to $\mathcal{Z}^{\text{user-sec}}$

(4) Upon being ask to provide a price for $(\mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}}, \mathrm{a}_{\mathcal{R}}^{\text{prev}})$

(a) Return a price p as the dummy adversary would do

(5) Upon receiving $(s'', \mathit{decom}''_{\text{ser}}, \mathit{com}_{\mathcal{R}}, \mathit{decom}''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ from $\mathcal{Z}^{\text{user-sec}}$ [a]

(a) $\mathit{decom}_{\mathcal{R}} := \mathit{decom}'_{\mathcal{R}} \cdot \mathit{decom}''_{\mathcal{R}}$

(b) If $\mathsf{C1.Open}(\mathsf{crs}^1_{\text{com}}, (1, g_1^p, 1, g_1), \mathit{com}_{\mathcal{R}}, \mathit{decom}_{\mathcal{R}}) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort

(c) If $\mathsf{S.Vfy}(\mathsf{vk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (\mathit{com}_{\mathcal{R}}, s)) = 0$, then let $\mathcal{F}_{\text{P4TC}}$ abort

(d) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the user

Debt Accumulation (other corruption scenarios):
Nothing to do

Debt Clearance (corrupted TSP, honest user):

(1) Load the recorded $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\mathcal{G}_{\text{bb}}$ for $\mathit{pid}_{\mathcal{T}}$. If any of these do not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Upon receiving u_2 from $\mathcal{Z}^{\text{user-sec}}$ in the name of the TSP with $\mathit{pid}_{\mathcal{T}}$

(a) Call $\mathcal{F}_{\text{P4TC}}$ with input (ClearDebt) in the name of the TSP with $\mathit{pid}_{\mathcal{T}}$

(b) Obtain leaked $\mathrm{a}_{\mathcal{R}}^{\text{prev}}$

(c) Obtain TSP's output $(\mathit{pid}_{\mathcal{U}}, \phi, b^{\text{bill}})$ from $\mathcal{F}_{\text{P4TC}}$, and delay the output of the user

(3) Upon being requested by $\mathcal{Z}^{\text{user-sec}}$ to provide the second message

(a) Check if any $(\phi, t', u'_2) \in \overline{\Omega}^{\text{dsp}}$ has been recorded previously with (ϕ) being used as key. If no, then pick $t \xleftarrow{R} \mathbb{Z}_q$. If yes, then load the recorded $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ and set $t := t' + \mathsf{sk}_{\mathcal{U}}(u_2 - u'_2)$. Insert (ϕ, t, u_2) into $\overline{\Omega}^{\text{dsp}}$.

(b) $\mathit{stmt} := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}^{\text{cert}}, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}}^{\text{prev}}, g_1^{b^{\text{bill}}}, t, u_2)$

(c) $\pi \leftarrow \mathsf{P3.SimProve}(\mathsf{crs}_{\text{pok}}, \mathsf{td}^{\text{sim}}_{\text{pok}}, \mathit{stmt})$

(d) Output $(\mathsf{pk}_{\mathcal{U}}, \pi, \phi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}_{\mathcal{R}}^{\text{prev}}, b^{\text{bill}}, t)$ to $\mathcal{Z}^{\text{user-sec}}$

(4) Upon receiving (OK) from $\mathcal{Z}^{\text{user-sec}}$ [d]

(a) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the user

Debt Clearance (other corruption scenarios):
Nothing to do

Prove Participation (corrupted SA, honest user):

(1) Load the recorded $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$. If this does not exist, then let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Upon receiving $S_{\mathcal{R}}^{\text{pp}}$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of the SA

(a) Call $\mathcal{F}_{\text{P4TC}}$ with input (ProveParticipation, $\mathit{pid}_{\mathcal{U}}, S_{\mathcal{R}}^{\text{pp}}$)

(b) Obtain the SA's output (out) from $\mathcal{F}_{\text{P4TC}}$

(c) If out = NOK, then abort

(d) Pick $\overline{\omega}_{\mathcal{U}}^{\text{pp}} = (s, \mathit{com}_{\text{hid}}, \overline{\mathit{decom}_{\text{hid}}})$ from $\overline{\Omega}_{\mathcal{U}}^{\text{pp}}$ such that $s \in S_{\mathcal{R}}^{\text{pp}}$. If this does not exist, then abort.

(e) Equivoke $\mathit{decom}_{\text{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{crs}^1_{\text{com}}, \mathsf{sk}_{\mathcal{U}}, \mathit{com}_{\text{hid}}, \overline{\mathit{decom}_{\text{hid}}})$

(f) Output $(s, \mathit{com}_{\text{hid}}, \mathit{decom}_{\text{hid}})$ to $\mathcal{Z}^{\text{user-sec}}$ as message from the user to the SA

Prove Participation (other corruption scenarios):
Nothing to do

Double-Spending Detection (honest TSP):

(1) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide a proof for $pid_{\mathcal{U}}$
  (a) Look up $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
  (b) Return $\mathsf{sk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$

Double-Spending Detection (corrupted TSP):
Nothing to do as this is a local algorithm for a corrupted TSP

Guilt Verification (honest P):

(1) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide OUT for $(pid_{\mathcal{U}}, \pi)$
  (a) Receive $\mathsf{pk}_{\mathcal{U}}$ from $\mathcal{G}_{\text{bb}}$ for $pid_{\mathcal{U}}$
  (b) If $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$, then set OUT $:=$ OK, else set OUT $:=$ NOK
  (c) Return OUT to $\mathcal{F}_{\text{P4TC}}$

Guilt Verification (corrupted P):
Nothing to do as this is a local algorithm for a corrupted P

Blacklisting & Recalculation (honest TSP):
Nothing to do

Blacklisting & Recalculation (corrupted TSP):

(1) Load the recorded $pid_{\text{DR}} \mapsto (\mathsf{pk}_{\text{DR}}, \mathsf{sk}_{\text{DR}})$
(2) Upon receiving $HTD_{\mathcal{U}}$ from $\mathcal{Z}^{\text{user-sec}}$
  (a) $HTD_{\mathcal{U}}^{\text{genuine}} = \left\{ (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \,\middle|\, (\cdot, \cdot, \lambda'', e^*) \in HTD_{\mathcal{U}} \wedge (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \in \overline{HTD} \right\}$
  (b) $HTD_{\mathcal{U}}^{\text{fake}} = \{ (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \mid (\cdot, s, \lambda'', e^*) \in HTD_{\mathcal{U}} \wedge (\cdot, \cdot, \lambda'', e^*) \notin \overline{HTD} \wedge$
  $(\ldots, \mathsf{pk}_{\mathcal{U}}) \leftarrow \mathsf{E.Dec}(\mathsf{sk}_{\text{DR}}, e^*) \}$
  (c) Assert $\mathsf{pk}_{\mathcal{U}}^{(1)} = \mathsf{pk}_{\mathcal{U}}^{(2)}$ for all $(\mathsf{pk}_{\mathcal{U}}^{(1)}, \cdot, \cdot, \cdot), (\mathsf{pk}_{\mathcal{U}}^{(2)}, \cdot, \cdot, \cdot) \in HTD_{\mathcal{U}}^{\text{genuine}} \cup HTD_{\mathcal{U}}^{\text{fake}}$ and set $\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}} := \mathsf{pk}_{\mathcal{U}}^{(1)}$, else abort
  (d) Call $\mathcal{F}_{\text{P4TC}}$ with (BlacklistUser, $\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}$)
(3) Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide S_{root}
  (a) $S_{\text{root}} := \left\{ s \,\middle|\, (\cdot, s, \cdot, \cdot) \in HTD_{\mathcal{U}}^{\text{genuine}} \right\}$
  (b) Provide S_{root} to $\mathcal{F}_{\text{P4TC}}$
(4) Upon receiving the TSP's output $(b^{\text{bill}}, \Phi_{\text{bl}})$ from $\mathcal{F}_{\text{P4TC}}$
  (a) For $HTD_{\mathcal{U}}^{\text{fake}}$, recover Φ^{fake} as the real DR would do
  (b) Send $\Phi_{\mathcal{U}} := \Phi_{\text{bl}} \cup \Phi^{\text{fake}}$ to $\mathcal{F}_{\text{P4TC}}$ as the message from the DR to the TSP

[a] If no message is received, let $\mathcal{F}_{\text{P4TC}}$ abort; if BLACKLISTED is received, override $\mathcal{F}_{\text{P4TC}}$'s delayed output for the user with BLACKLISTED.
[b] Use empty set as blacklist. If the TSP intended to blacklist the user, the TSP would not have sent the previous message.
[c] Use empty set as blacklist.
[d] If no message is received, let $\mathcal{F}_{\text{P4TC}}$ abort.

The overall proof idea is to define a sequence of hybrid experiments H_i together with simulators $\mathcal{S}_i$ and protocols Π_i such that the first hybrid H_0 is identical to the real experiment and the last hybrid H_{12} is identical to the ideal experiment. Each hybrid is of the form

$$\mathsf{H}_i := \mathsf{EXEC}_{\Pi_i, \mathcal{G}_{\text{bb}}, \mathcal{S}_i, \mathcal{Z}^{\text{user-sec}}}(1^{\lambda}).$$

Instead of directly proving indistinguishability of the real and ideal experiment we can break the proof down into showing indistinguishability of each pair of consecutive hybrids. We achieve this by demonstrating that whenever $\mathcal{Z}^{\text{user-sec}}$ can distinguish between two consecutive hybrids with non-negligible probability this yields an efficient adversary against one of the underlying cryptographic assumptions. The general idea is that the protocol Π_i that honest parties perform gradually declines from the real protocol $\Pi_0 = \Pi_{\text{P4TC}}$ to a dummy protocol Π_{12}, which does nothing but relay in- and outputs. At the same time $\mathcal{S}_i$ progresses from a dummy adversary $\mathcal{S}_0$ to the final simulator $\mathcal{S}_{12}$ which can be split up into the ideal functionality $\mathcal{F}_{\text{P4TC}}$ and $\mathcal{S}^{\text{user-sec}}_{P4TC}$. We proceed by giving concrete (incremental) definitions of all hybrids H_i.

Hybrid H_0 The hybrid H_0 is defined as

$$\mathsf{H}_0 \coloneqq \mathsf{EXEC}_{\Pi_0, \mathcal{G}_{\text{bb}}, \mathcal{S}_0, \mathcal{Z}^{\text{user-sec}}}(1^\lambda)$$

with $\mathcal{S}_0 = \mathcal{A}$ being identical to the dummy adversary and $\Pi_0 = \Pi_{\text{P4TC}}$. Hence, H_0 denotes the real experiment.

Hybrid H_1 In hybrid H_1 we modify $\mathcal{S}_1$ such that crs_{pok} is generated by SimSetup, $\text{crs}^1_{\text{com}}$ is generated by C1.SimGen and $\text{crs}^2_{\text{com}}$ is generated by C2.ExtGen. Additionally, $\mathcal{S}_1$ initializes the internal sets $\overline{\Omega}^{\text{dsp}}$, $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ and $\overline{HTD}$ as empty sets and records the respective entries as the final simulator $\mathcal{S}^{\text{user-sec}}_{P4TC}$ does.

Hybrid H_2 Hybrid H_2 replaces the code in the tasks DR/TSP/RSU/User Registration of the protocol Π_2 such that the simulator $\mathcal{S}_2$ is asked for the keys instead. This equals the method in which the keys are generated in the ideal experiment.

Hybrid H_3 In hybrid H_3 the task RSU Certification is modified. For an honest TSP or an honest RSU the code of Π_3 is replaced by the code of a dummy party. The simulator $\mathcal{S}_3$ behaves in this case as the final simulator $\mathcal{S}^{\text{user-sec}}_{P4TC}$ would.

Hybrid H_4 H_4 modifies the tasks of Wallet Issuing and Debt Accumulation. The code of Π_4 for the user is modified such that it does not send s' but randomly picks s and sends it to $\mathcal{S}_4$. Then $\mathcal{S}_4$ extracts $s'' \leftarrow \text{C2.Extract}(\text{crs}^2_{\text{com}}, com''_{\text{ser}})$, calculates $s' \coloneqq s \cdot (s'')^{-1}$ and inserts s' into the message from the user to the TSP or RSU respectively.

Hybrid H_5 This hybrid modifies Π_5 such that the honest parties do not send any proofs. Instead, the simulator $\mathcal{S}_5$ appends a simulated proof to the message from a user to a TSP or RSU without knowing the witness.

Hybrid H_6 H_6 modifies Π_6 such that honest users do not send the commitments com'_{seed}, $com_{\mathcal{T}}$ and $com_{\mathcal{R}}$ in the Wallet Issuing task and $com'_{\mathcal{R}}$ in the Debt Accumulation task. Instead, $\mathcal{S}_6$ injects suitable commitments to vectors of zeros. This equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{P4TC}$.

Hybrid H_7 This hybrid introduces a lookup table that links hidden user trapdoors to their origin. More precisely, if the task Wallet Issuing is executed, $\mathcal{S}_7$ records $\overline{htd} = (\text{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ in $\overline{HTD}$. Please note that $\mathcal{S}_7$ knows s due to the change in H_4.

Moreover, if the task Blacklisting & Recalculation is invoked, $\mathcal{S}_7$ partitions the set of hidden user trapdoors $HTD_{\mathcal{U}}$ provided by the environment into two "subsets"[5] $HTD^{\text{genuine}}_{\mathcal{U}}$ and $HTD^{\text{fake}}_{\mathcal{U}}$ (cp. Items (2a) to (2c) in the task Blacklisting & Recalculation of $\mathcal{S}^{\text{user-sec}}_{P4TC}$). If for a hidden user trapdoor $htd = (\text{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ a corresponding entry $(\cdot, \cdot, \lambda'', e^*)$ is recorded in $\overline{HTD}$, then we call it a *genuine* hidden user trapdoor and the public key $\text{pk}_{\mathcal{U}}$ and the serial number s are set to the originally recorded value.

[5] The sets $HTD^{\text{genuine}}_{\mathcal{U}}$ and $HTD^{\text{fake}}_{\mathcal{U}}$ might be no actual "subsets". The hidden user trapdoors are classified with respect to $(\cdot, \cdot, \lambda'', e^*)$ which are left unmodified, but the first two components $(\text{pk}_{\mathcal{U}}, s, \cdot, \cdot)$ are sanitized.

Genuine hidden user trapdoors are those which have legitimately been created in the scope of Wallet Issue. Else, we call it a *fake* hidden user trapdoor. In this case, the provided serial number s is left as is and the public key $\mathsf{pk}_{\mathcal{U}}$ is set to the decrypted value from e^*.[6] $\mathcal{S}_7$ additionally checks if the hidden user trapdoors of both sets $HTD_{\mathcal{U}}^{\text{genuine}}$ and $HTD_{\mathcal{U}}^{\text{fake}}$ belong to the same user public key $\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}$ or else aborts. This equals the behavior of the final simulator.

$\mathcal{S}_7$ runs the code of an honest DR on $HTD_{\mathcal{U}}^{\text{genuine}} \cup HTD_{\mathcal{U}}^{\text{fake}}$ to recover $\Phi_{\mathcal{U}}$, i.e., it decrypts every hidden user trapdoor and evaluates the PRF itself. For the DR the code of Π_7 is changed such that it simply signals its consent by forwarding its input $\mathsf{pk}_{\mathcal{U}}^{\mathsf{DR}}$ to $\mathcal{S}_7$.

Hybrid H_7 is a preparative step to eventually free the simulator from having to actually decrypt genuine hidden user trapdoors in case a user is blacklisted. Decryption of genuine hidden user trapdoors becomes impossible for the final simulator $\mathcal{S}_{P4TC}^{\text{user-sec}}$ as hybrid H_9 replaces all hidden user trapdoors with an encryption of zeros. However, a hidden user trapdoor is not bound to any user secrets. This allows a (malicious) TSP to pick an arbitrary chosen wallet ID λ^{fake} and create a syntactically valid hidden user trapdoor for λ^{fake} and any public user key $\mathsf{pk}_{\mathcal{U}}$. If the DR and the (malicious) TSP agree to blacklist a user, the TSP may send these fake hidden user trapdoors in addition to the genuine hidden user trapdoors to the DR. In the real protocol the DR simply decrypts both types of hidden user trapdoors and returns a list of pseudo-random fraud detection IDs for each of them.[7] The final simulator needs to mimic this behavior. The ideal functionality always returns a list of uniformly drawn fraud detection IDs of the correct length *only* for legitimately issued wallets. Hence, the final simulator extends this list by fraud detection IDs for each of the fake hidden user trapdoors.

Hybrid H_8 This hybrid introduces a new incorruptible entity $\mathcal{F}_{\phi\text{-rand}}$ into the experiment that is only accessible by honest users and the simulator through subroutine input/output tapes.[8]

$\mathcal{F}_{\phi\text{-rand}}$ provides the following functionality: Internally, $\mathcal{F}_{\phi\text{-rand}}$ manages a partial map f_Φ, mapping pairs of wallet IDs λ and counters x to fraud detection IDs. Whenever an as yet undefined value $f_\Phi(\lambda, x)$ is required, $\mathcal{F}_{\phi\text{-rand}}$ defines $f_\Phi(\lambda, x) \coloneqq \mathsf{PRF}(\lambda, x)$. If a user requests a fraud detection ID ϕ for (λ, x), $\mathcal{F}_{\phi\text{-rand}}$ returns $f_\Phi(\lambda, x)$ to the user. If an honest user inquires $\mathcal{F}_{\phi\text{-rand}}$ for the first time for a fresh λ, the user has to also provide the corresponding serial number s of the current transaction. $\mathcal{F}_{\phi\text{-rand}}$ internally records that s is associated with λ. If $\mathcal{F}_{\phi\text{-rand}}$ is invoked by the simulator with input s, $\mathcal{F}_{\phi\text{-rand}}$ looks up the associated wallet ID λ and returns the set $\{f_\Phi(\lambda, 0), \ldots, f_\Phi(\lambda, x_{\mathrm{bl}_{\mathcal{R}}})\}$.

An honest user running Π_8 does not evaluate the PRF to obtain a fraud detection ID ϕ, but requests $\mathcal{F}_{\phi\text{-rand}}$ to provide one.

If Blacklisting & Recalculation is invoked, the simulator $\mathcal{S}_8$ proceeds as follows: For hidden user trapdoors in the set $HTD_{\mathcal{U}}^{\text{fake}}$ it still decrypts the seed and evaluates the PRF. However, for hidden user trapdoors $(\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ in the set $HTD_{\mathcal{U}}^{\text{genuine}}$, $\mathcal{S}_8$ it does not decrypt e^*, but requests $\mathcal{F}_{\phi\text{-rand}}$ for the corresponding set of fraud detection IDs using s.

Hybrid H_9 In the scope of Wallet Issuing Π_9 is modified such that honest users do not send e^*. Instead, $\mathcal{S}_6$ injects an encrypted zero-vector as e^*.

6 We assume that $\mathsf{E.Dec}$ returns $\perp$ if e^* cannot be decrypted.

7 Skipping ahead, please note that this is not a "real" attack but only a very cumbersome way for the TSP to evaluate the PRF on self-chosen seeds.

8 I.e., communication is confidential, reliable and trustworthy. One might think of this entity as a preliminary version of the eventual ideal functionality.

Hybrid H_{10} Hybrid H_{10} replaces the PRF inside $\mathcal{F}_{\phi\text{-rand}}$ by truly random values. Whenever an as yet undefined value $f_\Phi(\lambda, x)$ is required, $\mathcal{F}_{\phi\text{-rand}}$ independently and uniformly draws a fresh random fraud detection id ϕ and sets $f_\Phi(\lambda, x) \coloneqq \phi$.

Hybrid H_{11} In hybrid H_{11} Debt Accumulation and Debt Clearance are modified such that the simulator $\mathcal{S}_{11}$ replaces t in the message from the user. If no $(\phi, t', u_2') \in \overline{\Omega}^{\mathrm{dsp}}$ has been recorded previously, $\mathcal{S}_{11}$ picks $t \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathrm{q}}$, else $\mathcal{S}_{11}$ sets $t \coloneqq t' + \mathsf{sk}_{\mathcal{U}}(u_2 - u_2')$. Finally, $\mathcal{S}_{11}$ inserts (ϕ, t, u_2) into $\overline{\Omega}^{\mathrm{dsp}}$. This equals the behavior of the final simulator $\mathcal{S}_{P4TC}^{\text{user-sec}}$.

Hybrid H_{12} The hybrid H_{12} modifies Debt Accumulation and Prove Participation. In Debt Accumulation the simulator $\mathcal{S}_{12}$ runs $(com_{\mathrm{hid}}, \overline{decom}_{\mathrm{hid}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{crs}^1_{\mathsf{com}})$ and appends $(s, com_{\mathrm{hid}}, \overline{decom}_{\mathrm{hid}})$ to $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$. In Prove Participation the code of $\mathcal{S}_{12}$ for the honest user is replaced by a code that just checks if the user has a matching and correct $(s^*, com^*_{\mathrm{hid}}, decom^*_{\mathrm{hid}})$ and respectively sends OK or NOK to $\mathcal{S}_{12}$. If $\mathcal{S}_{12}$ receives OK from the user, then it picks $\omega^{\mathrm{pp}}_{\mathcal{U}} = (s, com_{\mathrm{hid}}, \overline{decom}_{\mathrm{hid}})$ from $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$ such that $s \in S^{\mathrm{pp}}_{\mathcal{R}}$. Furthermore, it runs $decom_{\mathrm{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{crs}^1_{\mathsf{com}}, \mathsf{sk}_{\mathcal{U}}, com_{\mathrm{hid}}, \overline{decom}_{\mathrm{hid}})$ and sends $(s, com_{\mathrm{hid}}, decom_{\mathrm{hid}})$ to the TSP. Again, this equals the behavior of the final simulator $\mathcal{S}_{P4TC}^{\text{user-sec}}$.

The combinations of all modifications from H_0 to H_{12} yields

$$\begin{aligned} \mathsf{H}_{12} &= \mathsf{EXEC}_{\Pi_{12}, \mathcal{G}_{\mathrm{bb}}, \mathcal{S}_{12}, \mathcal{Z}^{\text{user-sec}}}(1^\lambda) \\ &= \mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \mathcal{G}_{\mathrm{bb}}, \mathcal{S}^{\text{user-sec}}_{P4TC}, \mathcal{Z}^{\text{user-sec}}}(1^\lambda). \end{aligned}$$

With these hybrids we can now give the proof of Theorem A.26. We do not spell out all steps of the proofs in full detail, but rather sketch the necessary reductions.

Proof of Theorem A.26.

From H_0 *to* H_1*:* This hop solely changes how the crs is created during the setup phase. This is indistinguishable for $\mathsf{crs}_{\mathsf{pok}}$, $\mathsf{crs}^1_{\mathsf{com}}$, and $\mathsf{crs}^2_{\mathsf{com}}$ (see the composable zero-knowledge property of Definition 2.28, the equivocality property and the extractability property of Definition 2.24, resp., condition (a) each).

From H_1 *to* H_2*:* This hop does not change anything in the view of $\mathcal{Z}^{\text{user-sec}}$ as $\mathcal{S}_2$ runs the same key generation algorithm as the real protocol does for honest parties.

From H_2 *to* H_3*:* Again, this hop only changes which party runs which part of the code, but has no effect on the view of $\mathcal{Z}^{\text{user-sec}}$.

From H_3 *to* H_4*:* This hop does not change anything from the perspective of $\mathcal{Z}^{\text{user-sec}}$ as C2 is perfectly extractable. The change is a purely syntactical one to push the simulator closer to $\mathcal{S}_{P4TC}^{\text{user-sec}}$.

From H_4 *to* H_5*:* This game hop replaces the real proofs by simulated proofs. To show indistinguishability despite this change, we actually have to consider a sequence of sub-hybrids—one for each of the different ZK proof systems P1, P2 and P3. In the first sub-hybrid all proofs for P1 are replaced by simulated proofs, in the second sub-hybrid all proofs for P2 are replaced and finally all proofs for P3. Assume there exists $\mathcal{Z}^{\text{user-sec}}$ that notices a difference between H_4 and the first sub-hybrid. Then we can construct an adversary $\mathcal{B}$ that has a non-negligible advantage $\mathsf{Adv}^{\mathsf{pok\text{-}zk}}_{\mathsf{POK},\mathcal{B}}(\lambda)$. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the protocol and simulator for $\mathcal{Z}^{\text{user-sec}}$. All calls of the simulator to P1.Prove are forwarded by $\mathcal{B}$ to its own oracle in the external challenge game which is either P1.Prove or P1.SimProve. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs. The second and third sub-hybrid follow the same line, but this time $\mathcal{B}$ internally needs to generate simulated proofs for the proof system that has already been replaced in the previous sub-hybrid. As $\mathcal{B}$ gets the simulation trapdoor as part of its input in the external challenge game, $\mathcal{B}$ can do so.

From H_5 *to* H_6*:* In this hop the commitments com'_{seed}, $com_{\mathcal{T}}$, $com_{\mathcal{R}}$ and $com'_{\mathcal{R}}$ are replaced with commitments to zero-messages for every honest user. Again, the hop from H_5 to H_6 is further split into a sequence of sub-hybrids with each sub-hybrid replacing a single commitment in reverse order of appearance. Assume $\mathcal{Z}^{\text{user-sec}}$ can distinguish between H_5 and H_6 with non-negligible advantage. This yields an efficient adversary $\mathcal{B}$ against the hiding property of C1. Please note that none of the commitments are ever opened, hence in each sub-hybrid only a single message is replaced. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the role of all parties and the simulator for $\mathcal{Z}^{\text{user-sec}}$. Externally, $\mathcal{B}$ plays the hiding game. First, $\mathcal{B}$ guesses the index i of the sub-hybrid which lets $\mathcal{Z}^{\text{user-sec}}$ distinguish. For the first $(i-1)$ commitments, $\mathcal{B}$ commits to the true message. For the i^{th} commitment, $\mathcal{B}$ sends the actual message and an all-zero message to the external challenger. $\mathcal{B}$ embeds the external challenge commitment (either to the actual message or the all-zero message) as the i^{th} commitment. All remaining commitments are replaced by commitments to zeros. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs.

From H_6 *to* H_7*:* This hop is perfectly indistinguishable from the environment's perspective as the additional code executed by $\mathcal{S}_7$ does not change the output. Note that the hidden user trapdoors are still recovered in the same way the real DR would. For hidden user trapdoors $htd = (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ in $HTD^{\text{genuine}}_{\mathcal{U}}$ the (outer) public key $\mathsf{pk}_{\mathcal{U}}$ is replaced by the public key that has originally been recorded for $(\cdot, \cdot, \lambda'', e^*)$. However, due to the correctness of E the ciphertext e^* determines a unique message (for a fix key pair $\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}}$) and thus the originally recorded $\mathsf{pk}_{\mathcal{U}}$ equals the one that e^* decrypts to. The additional, pairwise equality check for all public keys triggers an abort if and only if the real DR aborts as well.

From H_7 *to* H_8*:* Again, this hop is purely syntactical. The inserted entity $\mathcal{F}_{\phi\text{-rand}}$ is invisible for $\mathcal{Z}^{\text{user-sec}}$. Moreover, $\mathcal{F}_{\phi\text{-rand}}$ still uses the real PRF to generate fraud detection IDs. However, this hop frees $\mathcal{S}_8$ from the decryption of genuine hidden user trapdoors. Instead, $\mathcal{S}_8$ uses the originally recorded serial number s of the associated Wallet Issuing task to look up the set $\{\mathsf{PRF}(\lambda, 0), \dots, \mathsf{PRF}(\lambda, x_{\mathrm{bl}_{\mathcal{R}}})\}$, if required. Again, this is possible due to the correctness of E, i.e., e^* uniquely determines λ and thus maps to a unique s.

From H_8 *to* H_9*:* In this hop every encryption e^* of a wallet ID λ is replaced by an encryption of a 1-vector for every honest user. We further split this hop into a sequence of sub-hybrids, with each sub-hybrid replacing a single encryption in reverse order of appearance. Assume $\mathcal{Z}^{\text{user-sec}}$ can distinguish between H_8 and H_9 with non-negligible advantage. This yields an efficient adversary $\mathcal{B}$ against the IND-CCA security of the encryption scheme E. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the role of all parties and the simulator for $\mathcal{Z}^{\text{user-sec}}$. Externally, $\mathcal{B}$ plays the IND-CCA2 game. When $\mathcal{B}$—playing the role of the simulator—needs to provide the public key in the scope of DR Registration, it embeds the challenge key $\mathsf{pk}_{\mathsf{DR}} \coloneqq \mathsf{pk}^{\mathcal{C}}$. $\mathcal{B}$ needs to guess the index of the sub-hybrid that causes a non-negligible difference, i.e., $\mathcal{B}$ needs to guess which (user) wallet causes distinguishability. For the first $(i-1)$ invocations of Wallet Issuing, $\mathcal{B}$ encrypts the true seed, in the i^{th} invocation $\mathcal{B}$ embeds the external challenge and $\mathcal{B}$ encrypts a 1-vector for the remaining invocations of Wallet Issuing. If $\mathcal{Z}^{\text{user-sec}}$ invokes the task Blacklisting & Recalculation and $\mathcal{B}$ needs to restore the wallet ID, the following two cases may occur:

(1) The presented hidden user trapdoor is a genuine trapdoor.

(2) The presented hidden user trapdoor is a fake trapdoor.

In case (1) $\mathcal{B}$ uses its lookup table to recover the correct set of fraud detection IDs. In case (2) $\mathcal{B}$ uses its decryption oracle of the external IND-CCA2 game to restore the wallet ID λ and to create a set of fraud detection IDs. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs.

From H_9 *to* H_{10}*:* In this hop the pseudo-random fraud detection IDs for honest users are replaced by uniformly drawn random IDs. Again, we proceed by introducing a sequence of sub-hybrids. In each sub-hybrid the fraud detection IDs for one particular wallet ID λ are replaced. If $\mathcal{Z}^{\text{user-sec}}$ can distinguish between two of the sub-hybrids, this immediately yields an efficient adversary $\mathcal{B}$ against the pseudo-random game as defined in Definition 2.32. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the protocol and simulator for $\mathcal{Z}^{\text{user-sec}}$. Externally, $\mathcal{B}$ interacts with an oracle that is either a true random function $R(\cdot)$ or a pseudo-random function $\mathsf{PRF}(\hat{\lambda}, \cdot)$ for an unknown seed $\hat{\lambda}$. Whenever $\mathcal{B}$ playing $\mathcal{F}_{\phi\text{-rand}}$ internally needs to draw a fraud detection ID for the particular wallet λ, $\mathcal{B}$ uses its external oracle. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs. Please note that this argument crucially uses the fact that $\mathcal{Z}^{\text{user-sec}}$ is information-theoretically independent of λ. The hidden user trapdoors e^* have already been replaced by encryptions of 1-vectors in the previous hybrid H_9. This enables the external challenger to pick any seed $\hat{\lambda}$.

From H_{10} *to* H_{11}*:* The two hybrids are statistically identical. As long as no double-spending occurs, the user chooses a fresh u_1 in every transaction and thus a single point (u_2, t) is information-theoretically independent from $\mathsf{sk}_{\mathcal{U}}$.

From H_{11} *to* H_{12}*:* In this hop the simulator $\mathcal{S}_{12}$ sends simulated commitments com_{hid} for the hidden user ID instead of commitments to the true values. Later, $\mathcal{S}_{12}$ equivokes these commitments on demand to the correct $\mathsf{pk}_{\mathcal{U}}$, if $\mathcal{Z}^{\text{user-sec}}$ triggers Prove Participation. Again, if $\mathcal{Z}^{\text{user-sec}}$ has a non-negligible advantage to distinguish between H_{11} and H_{12}, then an efficient adversary $\mathcal{B}$ can be constructed against the hiding property and equivocality of C1. The reduction follows the same lines as in the hop from hybrid H_5 to H_6. □

B. Appendix for Chapter 4

B.1. Security Proof: Π_{AS} UC-realizes $\mathcal{F}_{\mathrm{AS}}$

Here we prove that the protocol Π_{AS} from Section 4.4 UC-realizes the functionality $\mathcal{F}_{\mathrm{AS}}$ from Section 4.3. In particular, we prove the following theorem.

Theorem 4.3. *Π_{AS} UC-realizes $\mathcal{F}_{\mathrm{AS}}$ in the $\{\mathcal{F}_{\mathrm{AD}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BB}}, \mathcal{G}_{\mathrm{CLOCK}}\}$-hybrid model under the assumptions that*

- COM *is a (computationally) hiding, (statistically) binding and (dual-mode) extractable and equivocable commitment scheme,*
- Σ *is a* EUF-CMA*-secure signature scheme,*
- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- *and* TPKE *is* IND-CPA*-secure*

against all PPT*-adversaries* $\mathcal{A}$ *who statically corrupt either*

(1) a subset of the users,

(2) LE and a subset of the users,

(3) SO and a subset of the users, or

(4) SO, LE and a subset of the users.

To simplify the security proof a bit, we split it into two cases, depending on whether SO is honest or corrupted. We call the first and second corruption scenario (where SO is honest) *System Security* and the third and fourth corruption scenario (where SO is corrupted) *User Security* and prove them separately.

B.1.1. System Security

In the *System Security* case, we handle the following corruptions:

- U can be statically corrupted, so some U are corrupted and some U are honest.
- SO is honest here
- LE can be statically corrupted
- J and AU are always honest

We now first describe a simulator $\mathcal{S}$ for those corruptions and then proceed to prove the security. The simulator uses the following writing conventions:

- If $\mathcal{F}_{\text{AS}}$ wants to deliver outputs to honest parties, delay them until the simulator explicitly allows them.
- Deny any other calls until the System Init task has been successfully completed.
- Hybrid functionalities $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{BB}}$ are simulated honestly. We assume in the following that whenever calls to those functionalities are made, $\mathcal{S}$ simulates them honestly.
- Hybrid functionality $\mathcal{F}_{\text{AD}}$ is mostly simulated honestly, except the tasks Request Decryption, Get Statistics and Handover (for details see the paragraph “simulation of $\mathcal{F}_{\text{AD}}$”).

The simulator $\mathcal{S}$ for System Security is defined as follows.

Simulator for System Security

State of the Simulator:

- Setup
  - TPKE keypair: $(\mathsf{pk}, \mathsf{sk})$
  - SO signing keypair: $(\mathsf{vk}_{\text{SO}}, \mathsf{sk}_{\text{SO}})$
  - J signing keypair: $(\mathsf{vk}_{\text{J}}, \mathsf{sk}_{\text{J}})$
  - Common reference strings $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{crs}_{\text{com}})$
  - Trapdoor $\mathsf{td}^{\text{ext}}_{\text{com}}$ for extracting commitments
  - Trapdoor $\mathsf{td}_{\text{NIZK}}$ for simulating and extracting NIZKs
- $\mathsf{L}^{\text{sys-sec}}_{\text{Registered}}$ with $(uid, \sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}})$ entries for *corrupted* users. Filled in the task User Registration. This is the same list that SO keeps in Π_{AS}.
- $\mathsf{L}^{\text{sys-sec}}_{\text{HonestU}}$ with $(pid, uid, \mathsf{vk}_{\text{U}}, \mathsf{sk}_{\text{U}})$ entries for *honest* users. Filled in the task User Registration.
- $\mathsf{L}^{\text{sys-sec}}_{\text{StoreSecret}}$ with $(uid, vper, secret, sec_1, com_{sec_1}, decom_{sec_1}, \sigma_{sec_1}, sec_2, ct, \sigma_{\text{U}})$ entries for *corrupted* users. Filled during StoreSecret.

- If LE is corrupted:
  - The list $\mathsf{L}^{\text{sys-sec}}_{\text{Warrants}}$ with $(W, 1, \widetilde{W}, \sigma_{\widetilde{W}})$ entries. Stored in Request Warrant (like J in Π_{AS})
  - The list $\mathsf{L}^{\text{sys-sec}}_{\text{HonestSecrets}}$ stores $(uid_i, vper_i, sec_{1,i})$ entries for *honest* users. Filled during Request Warrant and used in $\mathcal{F}_{\text{AD}}$.RequestDecryption
  - The list $\mathsf{L}^{\text{sys-sec}}_{\text{ReqNotReady}}$ temporarily stores $(ct_i, sec_{2,i})$ entries for *honest* users. Filled during the simulation of $\mathcal{F}_{\text{AD}}$.RequestDecryption and cleared during the simulation of $\mathcal{F}_{\text{AD}}$.Handover.
- If LE is honest:
  - The lists $\mathsf{L}_{\text{Warrants}}$, $\mathsf{L}_{\text{Requests}}$, $\mathsf{L}^{\text{Ready}}_{\text{Requests}}$ and $\mathsf{L}_{\text{Stats}}$ in $\mathcal{F}_{\text{AD}}$ are not used. Instead, manage a separate list for statistics: $\mathsf{L}^{\text{sys-sec}}_{\text{Stats}}$ with $f_t(W)$ entries. This list is filled during RequestWarrant.
- List for simulation of $\mathcal{F}_{\text{BB}}$: L_{BB} (initially empty).

System Setup ($\mathcal{F}_{\text{CRS}}$):

(1) $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{td}_{\text{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
(2) $(\mathsf{crs}_{\text{com}}, \mathsf{td}^{\text{ext}}_{\text{com}}) \leftarrow \mathsf{COM.ExtSetup}(1^\lambda)$
(3) Store all generated values

System Setup ($\mathcal{F}_{\mathsf{BB}}$):

(1) Create empty list L_{BB}

System Setup ($\mathcal{F}_{\mathsf{AD}}$):

(1) Execute this during the System Init task
(2) Create signing keypair for J: $(\mathsf{vk}_{\mathsf{J}}, \mathsf{sk}_{\mathsf{J}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$ and store it
(3) Create signing keypair for SO: $(\mathsf{vk}_{\mathsf{SO}}, \mathsf{sk}_{\mathsf{SO}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$ and store it
(4) Honestly simulate the Init task of $\mathcal{F}_{\mathsf{AD}}$ with the generated public keys as inputs. Store all generated variables and lists. Leak all necessary information to the adversary.

System Setup ($\mathcal{F}_{\mathsf{AS}}$):

(1) Choose the same system parameters for $\mathcal{F}_{\mathsf{AS}}$ as Π_{AS}

Simulation of $\mathcal{F}_{\mathsf{CRS}}$:
Corrupt P can issue calls to $\mathcal{F}_{\mathsf{CRS}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\mathsf{CRS}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\mathsf{CRS}}$ for most tasks in the following.

$\mathcal{S}$ answers all calls honestly.

Simulation of $\mathcal{F}_{\mathsf{BB}}$:
Corrupted U can issue calls to $\mathcal{F}_{\mathsf{BB}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\mathsf{BB}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\mathsf{BB}}$ in the following.

$\mathcal{S}$ answers all calls honestly.

Simulation of $\mathcal{F}_{\mathsf{AD}}$:
Corrupt P can issue calls to $\mathcal{F}_{\mathsf{AD}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\mathsf{AD}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\mathsf{AD}}$ for most tasks in the following.

$\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{AD}}$ honestly (including taking track of all internal lists and leaking information to the adversary), except for the following tasks:

- **Request Decryption**
  (1) Retrieve $(\textsc{Request}, \widetilde{W}, \sigma_{\widetilde{W}})$ as input from LE
  (2) Check Warrant Signature: $b \leftarrow \Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathsf{J}}, \widetilde{W}, \sigma_{\widetilde{W}})$
  (3) If $b = 0$, abort (Warrant not valid)
  (4) Find the entry $(W, 1, \widetilde{W}, \sigma_{\widetilde{W}}) \in \mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{Warrants}}$ containing $\widetilde{W}$. If none exists, abort (warrant not granted)
  (5) Call $\mathcal{F}_{\mathsf{AS}}$ in the name of LE with input $(\textsc{GetSecrets}, W)$ and get output $(\textsc{GotSecrets}, (secret_1, \ldots, secret_v))$ for LE
  (6) Parse enhanced warrant: $(\widetilde{W}_1, \ldots, \widetilde{W}_v) \leftarrow \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \leftarrow \widetilde{W}_i$
  (7) For each i from 1 to v:
    (a) Check if there is an entry $(\cdot, uid_i, \cdot, \cdot) \in \mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{HonestU}}$. If yes (case: user is honest) go to step (7b), if not (case: user is corrupted) go to step (7c).
    (b) *Case: User is honest*
      (i) For uid_i, retrieve $(uid_i, vper_i, sec_{1,i})$ from $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{HonestSecrets}}$
      (ii) Compute $sec_{2,i}$ such that $sec_{1,i} \oplus sec_{2,i} = secret_i$
      (iii) Add $(ct_i, sec_{2,i})$ to $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{ReqNotReady}}$
      (iv) Go back to step (7) and continue with next item
    (c) *Case: User is corrupted*

        (i) Append ct_i to $\mathsf{L}_{\text{Requests}}$ of $\mathcal{F}_{\text{AD}}$
        (ii) Go back to step (7) and continue with next item
    (8) Send $(\textsc{Request}, pid_{\text{LE}}, f_t(W), |\widetilde{W}|, v)$ to $\mathcal{A}$
    (9) When $\mathcal{A}$ allows to deliver output, store $\widetilde{W}$ in list of warrants $\mathsf{L}_{\text{Warrants}}$ of $\mathcal{F}_{\text{AD}}$
    (10) Output $(\textsc{Request})$ to LE

- **Get Statistics**
  - If LE is corrupted:
    (1) Simulate task honestly
  - If LE is honest:
    (1) Send $(\textsc{GetStatistics})$ to the adversary
    (2) Output $(\textsc{GotStatistics}, \mathsf{L}^{\text{sys-sec}}_{\text{Stats}})$
- **Handover**
  Simulate honestly, but additionally execute the following steps at the end:
  (1) For each item $(ct_i, sec_{2,i})$ in $\mathsf{L}^{\text{sys-sec}}_{\text{ReqNotReady}}$, append $(ct_i, sec_{2,i})$ to $\mathsf{L}^{\text{Pending}}_{\text{Requests}}$
  (2) Clear $\mathsf{L}^{\text{sys-sec}}_{\text{ReqNotReady}}$

System Init (honest SO, honest J, honest AU):

(1) Upon receiving the leak $(\textsc{Init})$ from $\mathcal{F}_{\text{AS}}$
  (a) Initialize $\mathcal{F}_{\text{AD}}$ as described above
  (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to all parties

Party Init (honest P):

(1) Upon receiving the leak $(\textsc{PInit})$ from $\mathcal{F}_{\text{AS}}$
  (a) Send $(\textsc{GetSKeys})$ and $(\textsc{GetPK})$ to the adversary
  (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to all parties

Party Init (corrupted P):
This is handled by simulation of $\mathcal{F}_{\text{AD}}$.

User Registration (honest U, honest SO):

(1) Upon receiving the leak $(\textsc{Register}, pid, uid)$ from $\mathcal{F}_{\text{AS}}$
  (a) Send $(\textsc{Ok})$ to $\mathcal{F}_{\text{AS}}$
(2) When $\mathcal{F}_{\text{AS}}$ wants to deliver output,
  (a) Generate keypair for the user: $(\mathsf{vk}_{\text{U}}, \mathsf{sk}_{\text{U}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
  (b) Store $(pid, uid, \mathsf{vk}_{\text{U}}, \mathsf{sk}_{\text{U}})$ in list of honest users $\mathsf{L}^{\text{sys-sec}}_{\text{HonestU}}$
  (c) Honestly simulate $\textsc{Register}$ and $\textsc{Retrieve}$ calls to $\mathcal{F}_{\text{BB}}$ including leaks to the adversary
  (d) Send $(\textsc{GetSKeys})$ and $(\textsc{GetPK})$ to the adversary
  (e) Allow $\mathcal{F}_{\text{AS}}$ to deliver output

User Registration (corrupted U, honest SO):

(1) Upon receiving the message (uid) from U with pid pid_{U}'
  (a) If there exists an entry $(uid, \cdot, \cdot, \cdot)$ in $\mathsf{L}^{\text{sys-sec}}_{\text{Registered}}$, let $\mathcal{S}$ abort in the name of SO (User already successfully registered)
  (b) Call $\mathcal{F}_{\text{AS}}$ in the name of U with $(\textsc{Register}, uid)$ under the pid pid_{U}'.
    (i) After receiving $(\textsc{Register}, pid_{\text{U}}', uid)$ from $\mathcal{F}_{\text{AS}}$, send $(\textsc{Ok})$ to $\mathcal{F}_{\text{AS}}$

(ii) If $\mathcal{F}_{\text{AS}}$ aborts, $\mathcal{S}$ aborts as well.

(2) Upon receiving output (Registered) from $\mathcal{F}_{\text{AS}}$ to U
  (a) Note that receiving this output implies that $uid = uid'$, where uid' is the secret SO input.
  (b) Report message (uid_ok) from SO to U
(3) Upon receiving (Ok) from U to SO
  (a) Honestly simulate call (Retrieve, uid) to $\mathcal{F}_{\text{BB}}$ including leaks to the adversary
  (b) Receive answer (Retrieve, $pid_{\text{U}}{}^{*}$, uid, $\mathsf{vk}^{*}_{\text{U}}$) from $\mathcal{F}_{\text{BB}}$. If $pid_{\text{U}}{}^{*} \neq pid_{\text{U}}{}'$, abort (UID already taken by another user or not registered).
  (c) $(com_{\text{uid}}, decom_{\text{uid}}) = \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, uid)$
  (d) $\sigma_{\text{reg}} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{SO}}, com_{\text{uid}})$
  (e) Store $(uid, \sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}})$ in $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{Registered}}$
  (f) Report message $(\sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}})$ from SO to U
  (g) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to SO

Store Secret (honest U, honest SO):
Nothing to do, since $\mathcal{F}_{\text{AS}}$.StoreSecret does not leak anything, Π_{AS}.StoreSecret has no calls to subfunctionalities and all involved parties are honest.

Store Secret (corrupted U, honest SO):

(1) Upon receiving $(uid, \sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}}, com_{sec_2}, com_{\text{vper}}, decom_{\text{vper}})$ from U to SO:
  (a) Extract Commitment com_{vper} to get $vper$: $vper \leftarrow \mathsf{COM.Extract}(\mathsf{td}^{\mathsf{ext}}_{\mathsf{com}}, com_{\text{vper}})$
  (b) Call $\mathcal{F}_{\text{AS}}$ with input (StoreSecret, uid, $vper$) in the name of U
  (c) If $\mathcal{F}_{\text{AS}}$ aborts, let $\mathcal{S}$ abort as well (in the name of SO)
(2) Upon receiving (SecretStored, $secret$) as output from $\mathcal{F}_{\text{AS}}$ to U
  (a) Extract Commitment com_{sec_2} to get sec_2: $sec_2 \leftarrow \mathsf{COM.Extract}(\mathsf{td}^{\mathsf{ext}}_{\mathsf{com}}, com_{sec_2})$
  (b) Calculate sec_1 such that $secret = sec_1 \oplus sec_2 \in \mathcal{S}$
  (c) Report message (sec_1) from SO to U
(3) Upon receiving $(ct, com_{\text{sec}}, \pi_{\text{ss}}, \sigma_{\text{U}}, \sigma_{sec_1}, com_{sec_1}, decom_{sec_1})$ from U to SO:
  (a) Compute signature $\sigma_{\text{ss}} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{SO}}, (com_{\text{uid}}, com_{\text{sec}}, com_{vper}))$
  (b) Report message (σ_{ss}) from SO to U
  (c) Store $(uid, vper, secret, sec_1, com_{sec_1}, decom_{sec_1}, \sigma_{sec_1}, sec_2, ct, \sigma_{\text{U}})$ in $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{StoreSecret}}$
  (d) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to SO

Request Warrant (corrupted LE, honest J, honest SO):

(1) Upon receiving the message (W) from LE to J
  (a) Call $\mathcal{F}_{\text{AS}}$ in the name of LE with input (RequestWarrant, W)
  (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to J
(2) *Case 1:* If then $\mathcal{F}_{\text{AS}}$ delivers the output (RequestWarrant, 0) to LE:
  (a) Report message $(0, \perp)$ from J to LE
  Note that this concludes the simulation for this task in this case
(3) *Case 2:* If then $\mathcal{F}_{\text{AS}}$ leaks (RequestWarrant, $f_t(W)$, $|\widetilde{W}|$, v):
  (a) Set $b \coloneqq 1$
  (b) Set $\sigma_W = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{J}}, W)$
  (c) Store $(W, b, \perp, \perp)$ in $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{Warrants}}$
  (d) Report message (b, σ_W) from J to LE
(4) Upon receiving the message (W, σ'_W) from LE to SO

- (a) Verify Signature: If $\sigma_W \neq \sigma'_W$, abort in the name of SO
- (b) Parse warrant: $(W_1, \dots, W_v) \leftarrow W$ and $(uid_i, vper_i, meta_i) \leftarrow W_i$
- (c) Create empty lists $\mathsf{L}_{\mathsf{C}}^{\mathsf{sys-sec}}$ and $\mathsf{L}_{\mathsf{MsgReqW}}^{\mathsf{sys-sec}}$
- (d) For each i from 1 to v:
    - (i) Check if there is an entry $(\cdot, uid_i, \cdot, \cdot) \in \mathsf{L}_{\mathsf{HonestU}}^{\mathsf{sys-sec}}$. If yes (case: user is honest) go to step (4d-ii), if not (case: user is corrupted) go to step (4d-iii)
    - (ii) *Case: User is honest*
        - (A) Fake ciphertext $ct_i \leftarrow \mathsf{TPKE.Enc}(\mathsf{pk}, 0)$ and append ct_i to $\mathsf{L}_{\mathsf{C}}^{\mathsf{sys-sec}}$
        - (B) For uid_i, retrieve entry $(pid_i, uid_i, \mathsf{vk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}}) \in \mathsf{L}_{\mathsf{HonestU}}^{\mathsf{sys-sec}}$.
        - (C) Create signature $\sigma_{\mathrm{U,i}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{U}}, (ct_i, uid_i, vper_i))$
        - (D) Draw random $sec_{1,i} \xleftarrow{R} \mathcal{S}$ and commit to it: $(com_{sec_{1,i}}, decom_{sec_{1,i}}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathsf{com}}, sec_{1,i})$
        - (E) Create signature $\sigma_{sec_{1},\mathrm{i}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{U}}, com_{sec_{1,i}})$
        - (F) Append $(\sigma_{\mathrm{U,i}}, sec_{1,i}, com_{sec_{1,i}}, decom_{sec_{1,i}}, \sigma_{sec_{1},\mathrm{i}})$ to $\mathsf{L}_{\mathsf{MsgReqW}}^{\mathsf{sys-sec}}$
        - (G) Append $(uid_i, vper_i, sec_{1,i})$ to $\mathsf{L}_{\mathsf{HonestSecrets}}^{\mathsf{sys-sec}}$
        - (H) Go back to step (4d) and continue with next item
    - (iii) *Case: User is corrupted*
        - (A) For uid_i retrieve the entry $(uid_i, vper_i, secret_i, sec_{1,i}, com_{sec_{1,i}}, decom_{sec_{1,i}}, \sigma_{sec_{1},\mathrm{i}}, sec_{2,i}, ct_i, \sigma_{\mathrm{U,i}})$ from $\mathsf{L}_{\mathsf{StoreSecret}}^{\mathsf{sys-sec}}$. If none exists, abort in the name of SO (User has not stored a secret)
        - (B) Append ct_i to $\mathsf{L}_{\mathsf{C}}^{\mathsf{sys-sec}}$ and $(\sigma_{\mathrm{U,i}}, sec_{1,i}, com_{sec_{1,i}}, decom_{sec_{1,i}}, \sigma_{sec_{1},\mathrm{i}})$ to $\mathsf{L}_{\mathsf{MsgReqW}}^{\mathsf{sys-sec}}$
        - (C) Go back to step (4d) and continue with next item
- (e) Build enhanced warrant $\widetilde{W}$ by adding ct_i (from $\mathsf{L}_{\mathsf{C}}^{\mathsf{sys-sec}}$) to each W_i
- (f) Report message $(\widetilde{W}, \mathsf{L}_{\mathsf{MsgReqW}}^{\mathsf{sys-sec}})$ from SO to LE

(5) Upon receiving the message $(\widetilde{W}, \{\sigma_{\mathrm{U,i}}, sec_{1,i}, com_{sec_{1,i}}, decom_{sec_{1,i}}, \sigma_{sec_{1},\mathrm{i}}\}_{i\in\{1,\dots,v\}})$ from LE to J

- (a) Build W out of $\widetilde{W}$ by deleting ct_i from each $\widetilde{W_i}$
- (b) If there is no entry $(W, 1, \perp, \perp)$ in $\mathsf{L}_{\mathsf{Warrants}}^{\mathsf{sys-sec}}$, abort in the name of J
- (c) Parse enhanced warrant: $(\widetilde{W_1}, \dots, \widetilde{W_v}) \leftarrow \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \leftarrow \widetilde{W_i}$
- (d) For i from 1 to v:
    - (i) For uid_i, retrieve entry $(pid_{\mathrm{U,i}} uid_i, \mathsf{vk}_{\mathrm{U,i}}) \in \mathsf{L}_{\mathsf{BB}}$
    - (ii) Check signatures: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, (ct_i, uid, vper_i), \sigma_{\mathrm{U,i}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, com_{sec_{1},\mathrm{i}}, \sigma_{sec_{1},\mathrm{i}}) = 0$, abort in the name of J
    - (iii) Check commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{sec_{1},\mathrm{i}}, decom_{sec_{1},\mathrm{i}}, sec_{1,i}) = 0$, abort in the name of J
- (e) Sign enhanced warrant: $\sigma_{\widetilde{W}} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{J}}, \widetilde{W})$
- (f) Change the entry $(W, 1, \perp, \perp)$ in $\mathsf{L}_{\mathsf{Warrants}}^{\mathsf{sys-sec}}$ to $(W, 1, \widetilde{W}, \sigma_{\widetilde{W}})$
- (g) Report message $(\sigma_{\widetilde{W}})$ from J to LE

(6) When the adversary allows $\mathcal{F}_{\mathrm{AD}}$ to deliver output in $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption

- (a) Finish simulation of $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption
- (b) Send (Ok) to $\mathcal{F}_{\mathrm{AS}}$
- (c) Receive output (WarrantAnswer, b) from $\mathcal{F}_{\mathrm{AS}}$

Request Warrant (honest LE, honest J, honest SO):

(1) Upon being notified that $\mathcal{F}_{\mathrm{AS}}$ wants to deliver output to J, allow the output

(2) Upon receiving the leak (RequestWarrant, $f_t(W), |\widetilde{W}|, v$) from $\mathcal{F}_{\mathrm{AS}}$

- (a) Send (Request, $pid_{\mathrm{LE}}, f_t(W), |\widetilde{W}|, v$) to $\mathcal{A}$

(3) When $\mathcal{A}$ allows to deliver output in $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption

- (a) Store $f_t(W)$ in $\mathsf{L}_{\mathsf{Stats}}^{\mathsf{sys-sec}}$ for managing statistics in $\mathcal{F}_{\mathrm{AD}}$

   (b) Send (Ok) to $\mathcal{F}_{\text{AS}}$
   (c) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to LE

Get Secrets (corrupted LE):
This is handled by simulation of $\mathcal{F}_{\text{AD}}$.

Get Secrets (honest LE):

(1) Upon receiving the leak (GetSecrets) from $\mathcal{F}_{\text{AS}}$
   (a) Leak (Retrieve) to $\mathcal{A}$
   (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output

Get Statistics (corrupted P):
This is handled by simulation of $\mathcal{F}_{\text{AD}}$.

Get Statistics (honest P):

(1) Upon receiving the leak (GetStatistics) from $\mathcal{F}_{\text{AS}}$
   (a) Send (GetStatistics) to $\mathcal{A}$ and wait for okay from $\mathcal{A}$
   (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output

Audit (honest AU):

(1) Upon receiving the leak (AuditRequest) from $\mathcal{F}_{\text{AS}}$
   (a) Send (AuditRequest) to $\mathcal{A}$ and wait for okay from $\mathcal{A}$
   (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output

Prove (corrupted U):
Nothing to do since this is a completely local task.

Prove (honest U):

(1) Upon receiving the message (Prove, $stmt_R$) from $\mathcal{F}_{\text{AS}}$
   (a) Simulate proof: $\pi \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\text{NIZK}}, stmt_R, \mathcal{R}_{zk}^{AS})$
   (b) Send (Proof, π) to $\mathcal{F}_{\text{AS}}$

Verify (corrupted P):
Nothing to do since this is a completely local task.

Verify (honest P):

(1) Upon receiving the message (Verify, $stmt_R, \pi$) from $\mathcal{F}_{\text{AS}}$
   (a) Extract Witness: $wit_R \leftarrow \mathsf{NIZK.Ext}(\mathsf{td}_{\text{NIZK}}, stmt_R, \pi, \mathcal{R}_{zk}^{AS})$
   (b) Send (witness, wit_R) to $\mathcal{F}_{\text{AS}}$

We now prove the following theorem:

Theorem B.2 (System Security). Π_{AS} *UC-realizes* $\mathcal{F}_{\text{AS}}$ *in the* $\{\mathcal{F}_{\text{AD}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{BB}}, \mathcal{G}_{\text{CLOCK}}\}$*-hybrid model under the assumptions that*

- COM *is a (computationally) hiding, (statistically) binding and (dual-mode) extractable and equivocable commitment scheme,*

- Σ *is a* EUF-CMA-*secure signature scheme,*
- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- *and* TPKE *is* IND-CPA-*secure*

against all PPT-*adversaries* $\mathcal{A}$ *who statically corrupt either*

(1) a subset of the users, or

(2) LE and a subset of the users.

To prove Theorem B.2, we proceed in a series of games, where $\mathcal{F}_{\text{AS}}^{i+1}$ behaves the same as $\mathcal{F}_{\text{AS}}^{i}$ except for the stated changes and $\mathcal{S}^{i+1}$ behaves the same as $\mathcal{S}^{i}$ except for the stated changes.

Game 0. Game 0 is equivalent to the real experiment. That is,

$$H_0 \coloneqq \mathsf{Exp}^{\Pi_{\text{AS}}}_{\mathcal{F}_{\text{AD}},\mathcal{F}_{\text{CRS}},\mathcal{F}_{\text{BB}},\mathcal{G}_{\text{CLOCK}},\mathcal{S}^0,\mathcal{Z}}$$

with $\mathcal{S}^0$ being the dummy adversary. This means that all parties execute the real protocol.

Game 1. In this game, the simulator $\mathcal{S}^1$ takes control over $\mathcal{F}_{\text{CRS}}$, $\mathcal{F}_{\text{BB}}$ and $\mathcal{F}_{\text{AD}}$, but executes them honestly. Additionally, $\mathcal{F}_{\text{AS}}^{1}$ is introduced, and all honest parties are replaced by dummy parties that forward their inputs to $\mathcal{F}_{\text{AS}}^{1}$ and when receiving output from $\mathcal{F}_{\text{AS}}^{1}$ forward it to the environment. $\mathcal{F}_{\text{AS}}^{1}$, upon receiving any message from a dummy party forwards it to the simulator $\mathcal{S}^1$ and asks it for output. $\mathcal{S}^1$ executes the real protocol for all honest parties (using the inputs received from $\mathcal{F}_{\text{AS}}^{1}$) and instructs $\mathcal{F}_{\text{AS}}^{1}$ to deliver the resulting outputs. Note that $\mathcal{S}^1$ simulates $\mathcal{F}_{\text{BB}}$ like $\mathcal{S}$ does.

Proof Sketch: Game 0 ≈ Game 1.
$\mathcal{S}^1$ executes the same code as the real parties on the same inputs (that have been forwarded by $\mathcal{F}_{\text{AS}}^{1}$). Thus, Game 0 and Game 1 are identical from the environment's view.

Game 2. In this game, $\mathcal{S}^2$ generates the common reference string for the commitment scheme with an extraction trapdoor $\mathsf{td}_{\mathsf{com}}^{\mathsf{ext}}$ as $(\mathsf{crs}_{\mathsf{com}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{ext}}) \leftarrow \mathsf{COM.ExtSetup}(1^\lambda)$. $\mathcal{S}^2$ also generates the common reference string for the non-interactive proof system with a trapdoor for simulating and extracting proofs, i.e., $(\mathsf{crs}_{\mathsf{zk}}^{\mathsf{AS}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. Thus, $\mathcal{S}^2$ simulates $\mathcal{F}_{\text{CRS}}$ the same way $\mathcal{S}$ does.

Proof Sketch: Game 1 ≈ Game 2.
Indistinguishability follows from the extractability of COM and the simulation-extractability of NIZK.

Game 3. During setup of $\mathcal{F}_{\text{AD}}$, $\mathcal{S}^3$ is asked for the keypairs for SO, J and AU. $\mathcal{S}^3$ generates the keypairs honestly and stores them.

Proof Sketch: Game 2 ≈ Game 3.
$\mathcal{S}^3$ runs the same key generation algorithm as the real protocol does for honest parties, therefore $\mathcal{Z}$'s view remains unchanged.

Game 4. $\mathcal{S}^4$ keeps track of additional information. For each honest U, it stores $(uid, vper, sec_1)$ in the list $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{HonestSecrets}}$ at the end of the Store Secret task.
For each corrupted U, it stores $(uid, vper, \perp, sec_1, com_{sec_1}, decom_{sec_1}, \sigma_{sec_1}, sec_2, ct, \sigma_{\text{U}})$ in the list $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{StoreSecret}}$ at the end of the Store Secret task. Note that normally instead of $\perp$ the user's secret should be stored in the list. But at this point we do not know the secret yet.

Proof Sketch: Game 3 ≈ Game 4.
$\mathcal{S}^4$ only keeps track of more information (that the honest parties acquire during the real protocol), nothing to distinguish for $\mathcal{Z}$.

Game 5. In this game, $\mathcal{F}^5_{\mathrm{AS}}$ additionally handles the tasks System Init and Party Init the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^5$ handles these task the same way $\mathcal{S}$ does.

Proof Sketch: Game 4 ≈ Game 5.
Since SO, J and AU are honest, the games are identical for the System Init task.
Party Init for honest Parties: $\mathcal{Z}$'s view remains identical, since $\mathcal{S}^5$ only needs to leak (GetSKeys) and (GetPK) to $\mathcal{A}$ to emulate the call to $\mathcal{F}_{\mathrm{AD}}$.
Party Init for corrupted Parties: Only the simulation of hybrid functionalities needs to be handled, which are already simulated honestly.

Game 6. In this game, $\mathcal{F}^6_{\mathrm{AS}}$ additionally handles the Audit task the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^6$ handles this task the same way $\mathcal{S}$ does. Note that since the auditor AU is assumed to be honest, simulation only requires a simulated (AuditRequest) to $\mathcal{F}_{\mathrm{BCRA}}$.

Proof Sketch: Game 5 ≈ Game 6.
Since AU is honest, these games are identical.

Game 7. In this game, $\mathcal{F}^7_{\mathrm{AS}}$ additionally handles the User Registration task the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^7$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 6 ≈ Game 7.
For honest U: $\mathcal{Z}$'s view remains identical, since $\mathcal{S}^7$ only needs to leak (GetSKeys) and (GetPK) to $\mathcal{A}$ to emulate the calls to $\mathcal{F}_{\mathrm{AD}}$ and to honestly emulate the calls to $\mathcal{F}_{\mathrm{BB}}$. Note that additionally some information about honest users (*pid*, *uid* and keys) are stored in a designated list $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{HonestU}}$.
For corrupted U: Since $\mathcal{S}$ essentially simulates SO in the User Registration task honestly, the hybrids are indistinguishable for $\mathcal{Z}$. Note that $\mathcal{S}$ keeps track of the list of registered users like SO does in Π_{AS}.

Game 8. If LE is honest, $\mathcal{F}^8_{\mathrm{AS}}$ additionally handles the Request Warrant task the same way $\mathcal{F}_{\mathrm{AS}}$ does for honest LE, and $\mathcal{S}^8$ handles this task the same way $\mathcal{S}$ does for honest LE. Additionally, $\mathcal{S}^8$ simulates $\mathcal{F}_{\mathrm{AD}}$ as $\mathcal{S}$ does for the Get Statistics task in $\mathcal{F}_{\mathrm{AD}}$. Note that if LE is corrupted, this game changes nothing.

Proof Sketch: Game 7 ≈ Game 8.
$\mathcal{F}^8_{\mathrm{AS}}$ sends (RequestWarrant, $f_t(W)$, $|\widetilde{W}|$, v) to $\mathcal{S}^8$. $\mathcal{S}^8$ then simulates the execution of $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption by sending (Request, pid_{LE}, $f_t(W)$, $|\widetilde{W}|$, v) to $\mathcal{A}$, which yields an indistinguishable execution of the Request Warrant task. But now the Get Statistics task has to be adapted as well, since we now do not store a warrant in $\mathsf{L}_{\mathsf{Warrants}}$ (since we do not actually know the warrant). This is fixed by directly storing $f_t(W)$ in a separate list $\mathsf{L}^{\mathsf{sys-sec}}_{\mathsf{Stats}}$ during Request Warrant and using this list in $\mathcal{F}_{\mathrm{AD}}$.GetStatistics to return the same statistics as the real game does. The task $\mathcal{F}_{\mathrm{AD}}$.GetStatistics now behaves like $\mathcal{S}$ does, since there we only changed the behavior for honest LE and the task is executed honestly for corrupted LE.

Game 9. If LE is corrupted, $\mathcal{F}^9_{\mathrm{AS}}$ additionally handles the Request Warrant task the same way $\mathcal{F}_{\mathrm{AS}}$ does for corrupted LE, and $\mathcal{S}^9$ handles this task the same way $\mathcal{S}$ does for corrupted LE. Additionally, $\mathcal{S}^9$ simulates $\mathcal{F}_{\mathrm{AD}}$ as $\mathcal{S}$ does for the Request Decryption and Handover tasks in $\mathcal{F}_{\mathrm{AD}}$. Note that if LE is honest, this game changes nothing. Also note that now $\mathcal{S}^9$ simulates $\mathcal{F}_{\mathrm{AD}}$ as $\mathcal{S}$ does (for every task and every corruption scenario).

Proof Sketch: Game 8 ≈ Game 9.
$\mathcal{S}^9$ receives the necessary information from LE to call $\mathcal{F}^9_{\mathrm{AS}}$ with the correct input. After receiving output

from $\mathcal{F}_{\text{AS}}^{9}$, $\mathcal{S}^{9}$ mostly executes the real protocol for J and SO. For corrupted users, the information needed to send the correct messages can be extracted from the list $\mathsf{L}_{\text{StoreSecret}}^{\text{sys-sec}}$ that was filled during Store Secret (for corrupted users). For honest users, the strategy is a bit different, since the list $\mathsf{L}_{\text{StoreSecret}}^{\text{sys-sec}}$ has no entries for honest users. For honest users, $\mathcal{S}^{9}$ creates a fake ciphertext (by encrypting 0) and then honestly creates the remainder of the information needed to send the correct messages. Since TPKE is IND-CPA-secure, this change can not be detected. If the signature scheme Σ is EUF-CMA-secure and the commitment scheme COM is binding, $\mathcal{S}^{9}$ aborts in the same cases as SO/J would do in Π_{AS}. Note that $\mathcal{S}^{9}$ stores the granted warrants in $\mathsf{L}_{\text{Warrants}}^{\text{sys-sec}}$ during the Request Warrant task.
Now, the simulation of $\mathcal{F}_{\text{AD}}$.RequestDecryption needs to be handled as well. Since Σ is EUF-CMA-secure, $\mathcal{S}^{9}$ aborts in the same cases as the real $\mathcal{F}_{\text{AD}}$ would. $\mathcal{S}^{9}$ has access to the list $\mathsf{L}_{\text{Warrants}}^{\text{sys-sec}}$ that is filled during Request Warrant (which needs to be called before this task). Using $\mathsf{L}_{\text{Warrants}}^{\text{sys-sec}}$, $\mathcal{S}^{9}$ can call $\mathcal{F}_{\text{AS}}$ with the correct inputs and gets the corresponding outputs. Next, the secrets need to be prepared for retrieval. Here we again need to distinguish between secrets of honest and corrupted users. For corrupted users, we already know the correct ciphertext (from the input of this task) and we can just append the ciphertext to $\mathsf{L}_{\text{Requests}}$ of $\mathcal{F}_{\text{AD}}$ like an honest $\mathcal{F}_{\text{AD}}$. The remainder of the secret retrieval (for corrupted users) is then handled by the honest simulation of $\mathcal{F}_{\text{AD}}$. For honest users, we retrieve the entry $(uid, vper, sec_1)$ from $\mathsf{L}_{\text{HonestSecrets}}^{\text{sys-sec}}$ and use the output of $\mathcal{F}_{\text{AS}}^{9}$ to calculate the secret sec_2 (in the clear) that needs to be retrieved. We then store this secret sec_2 directly to a special list $\mathsf{L}_{\text{ReqNotReady}}^{\text{sys-sec}}$. Then the $\mathcal{F}_{\text{AD}}$.Handover task has to be adapted as well. It now additionally moves all entries from $\mathsf{L}_{\text{ReqNotReady}}^{\text{sys-sec}}$ to $\mathsf{L}_{\text{Requests}}^{\text{Pending}}$. This ensures that now for honest users the correct secrets can be retrieved as well. Note that for honest users, the ciphertext encrypting the user's secret in Store Secret is not used during retrieval anymore.

Game 10. In this game, $\mathcal{F}_{\text{AS}}^{10}$ additionally handles the tasks Get Secrets and Get Statistics the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^{10}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 9 ≈ Game 10.
Get Secrets: For honest LE, $\mathcal{Z}$'s view remains identical, since $\mathcal{S}^{10}$ only needs to leak (Retrieve) to $\mathcal{A}$ to emulate the $\mathcal{F}_{\text{AD}}$.RetrieveSecret task. For corrupted LE, only the simulation of $\mathcal{F}_{\text{AD}}$.RetrieveSecret needs to be handled, which is already simulated like the final simulator does and returns the correct value.
Get Statistics: For an honest party sending that message, $\mathcal{S}^{10}$ only needs to send (GetStatistics) to $\mathcal{A}$. For a corrupted party, this game is identical to Game 9 since this task is already handled by the simulation of $\mathcal{F}_{\text{AD}}$.

Game 11. In this game, $\mathcal{F}_{\text{AS}}^{11}$ additionally handles the Store Secret task the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^{11}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 10 ≈ Game 11.
For corrupted U: $\mathcal{S}^{11}$ can obtain the required inputs to call $\mathcal{F}_{\text{AS}}^{11}$ from the message $(uid, \sigma_{\text{reg}}, com_{\text{uid}}, decom_{\text{uid}}, com_{sec_2}, com_{\text{vper}}, decom_{\text{vper}})$ from U by extracting $vper$ from com_{vper}. Since COM is extractable, this is possible with overwhelming probability and allows $\mathcal{S}^{11}$ to call $\mathcal{F}_{\text{AS}}^{11}$ with the correct input. Using the output (SecretStored, $secret$) of $\mathcal{F}_{\text{AS}}^{11}$ and by extracting sec_2 from com_{sec_2} (which is again possible with overwhelming probability since COM is extractable), $\mathcal{S}^{11}$ can calculate the correct sec_1 and then honestly execute the remainder of the protocol for SO.
For honest U: nothing to do in this case.

Game 12. In this game, $\mathcal{F}_{\text{AS}}^{12}$ additionally handles the Prove and Verify tasks the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^{12}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 11 ≈ Game 12.
Since NIZK is a simulation extractable zero-knowledge proof system, $\mathcal{Z}$ can not distinguish between the outputs it gets from $\mathcal{F}_{\mathrm{AS}}^{12}$ and the outputs it gets from $\mathcal{F}_{\mathrm{AS}}$.

Note that $\mathcal{S}^{12}$ equals the final simulator $\mathcal{S}$. Since we showed that Game i is indistinguishable from Game $i+1$ for $i \in \{0, \ldots, 11\}$, it follows that Game 0 (the real protocol) is indistinguishable from Game 12 (ideal execution) and thus Theorem B.2 follows. □

B.1.2. User Security

In the *User Security* case, we handle the following corruptions:

- U can be statically corrupted, so some U are corrupted and some U are honest.
- SO is corrupted here (static corruption)
- LE can be corrupted here (static corruption)
- J and AU are always honest

We now first describe a simulator $\mathcal{S}$ for those corruptions and then proceed to prove the security. The simulator uses the following writing conventions:

- If $\mathcal{F}_{\mathrm{AS}}$ wants to deliver outputs to honest parties, delay them until the simulator explicitly allows them.
- Deny any other calls until the System Init task has been successfully completed
- Hybrid functionalities $\mathcal{F}_{\mathrm{CRS}}$, $\mathcal{F}_{\mathrm{BB}}$ are simulated honestly. We assume in the following that whenever calls to those functionalities are made, $\mathcal{S}$ simulates them honestly.
- Hybrid functionality $\mathcal{F}_{\mathrm{AD}}$ is mostly simulated honestly, except the tasks Request Decryption and Handover (for details see the paragraph "simulation of $\mathcal{F}_{\mathrm{AD}}$")

The simulator $\mathcal{S}$ for User Security is defined as follows.

Simulator for User Security

State of the Simulator:

- Setup:
  - TPKE keypair: $(\mathsf{pk}, \mathsf{sk})$
  - Verification key of SO: $\mathsf{vk}_{\mathrm{SO}}$
  - J signing keypair: $(\mathsf{vk}_{\mathrm{J}}, \mathsf{sk}_{\mathrm{J}})$
  - Common reference strings $(\mathsf{crs}_{\mathrm{zk}}^{\mathrm{AS}}, \mathsf{crs}_{\mathrm{com}}^{\mathrm{eq}})$
  - Trapdoor $\mathsf{td}_{\mathrm{NIZK}}$ for simulating and extracting proofs
  - Trapdoor $\mathsf{td}_{\mathrm{com}}^{\mathrm{eq}}$ for equivocating commitments
- Stored in User Registration:
  - $\mathsf{L}_{\mathrm{Registered}}^{\mathrm{user-sec}}$ with $(pid_{\mathrm{U}}, uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}}, \mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}})$ entries. Note that only honest users are in that list.
- Stored in Request Warrant:
  - List of already requested preliminary warrants with (W, b) entries

  - List of stored enhanced warrants with $(\widetilde{W}, \sigma_{\widetilde{W}})$ entries
- Stored for simulation of $\mathcal{F}_{\text{AD}}$:
  - $\mathsf{L}^{\text{user-sec}}_{\text{Secrets}}$ with $(uid, vper, sec_1, ct)$ entries. Entries are stored in AS.StoreSecret (for honest U) and used in $\mathcal{F}_{\text{AD}}$.RequestSecret.
  - Only for corrupted LE: The list $\mathsf{L}^{\text{user-sec}}_{\text{ReqNotReady}}$ temporarily stores $(ct_i, sec_{2,i})$ entries (for honest U). Filled during the simulation of $\mathcal{F}_{\text{AD}}$.RequestDecryption and cleared during the simulation of $\mathcal{F}_{\text{AD}}$.Handover.
  - Everything else that is stored inside $\mathcal{F}_{\text{AD}}$
- List for simulation of $\mathcal{F}_{\text{BB}}$:
  - L_{BB} (initially empty). Entries are of the form $(pid_{\text{U}}, uid, \mathsf{vk}_{\text{U}})$

System Setup ($\mathcal{F}_{\text{CRS}}$):

(1) $(\mathsf{crs}^{\text{AS}}_{\text{zk}}, \mathsf{td}_{\text{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
(2) $(\mathsf{crs}^{\text{eq}}_{\text{com}}, \mathsf{td}^{\text{eq}}_{\text{com}}) \leftarrow \mathsf{COM.SimSetup}(1^\lambda)$
(3) Store all values

System Setup ($\mathcal{F}_{\text{BB}}$):

(1) Create empty list L_{BB}.

System Setup ($\mathcal{F}_{\text{AD}}$):

(1) See simulation of Init-Task

System Setup ($\mathcal{F}_{\text{AS}}$):

(1) Choose the same system parameters for $\mathcal{F}_{\text{AS}}$ as Π_{AS}

Simulation of $\mathcal{F}_{\text{CRS}}$:
Corrupted P can issue calls to $\mathcal{F}_{\text{CRS}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\text{CRS}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\text{CRS}}$ for most tasks in the following.

$\mathcal{S}$ answers all calls honestly.

Simulation of $\mathcal{F}_{\text{BB}}$:
Corrupted P can issue calls to $\mathcal{F}_{\text{BB}}$ whenever they want, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\text{BB}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\text{BB}}$ in the following.

$\mathcal{S}$ answers all calls honestly.

Simulation of $\mathcal{F}_{\text{AD}}$:
Corrupted P can issue calls to $\mathcal{F}_{\text{AD}}$ whenever they wants, not just at the intended points in the protocol. We specify here once how all calls from P to $\mathcal{F}_{\text{AD}}$ to all tasks are handled and ignore calls to $\mathcal{F}_{\text{AD}}$ for most tasks in the following.

$\mathcal{S}$ simulates $\mathcal{F}_{\text{AD}}$ honestly (including taking track of all internal lists and leaking information to the adversary), except for the following tasks:

- **Request Decryption**
  (1) Receive (Request, $\widetilde{W}, \sigma_{\widetilde{W}}$) as input from LE
  (2) Check Warrant Signature: $b \leftarrow \Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{J}}, \widetilde{W}, \sigma_{\widetilde{W}})$
  (3) If $b = 0$, abort (Warrant not valid)
  (4) Build preliminary warrant W out of $\widetilde{W}$ by deleting ct_j from each $\widetilde{W}_j$

(5) Call $\mathcal{F}_{\text{AS}}$ in the name of LE with input ($\textsc{GetSecrets}$, W)
(6) If $\mathcal{F}_{\text{AS}}$ aborts, $\mathcal{S}$ lets $\mathcal{F}_{\text{AD}}$ abort as well (Warrant either not requested or not granted)
(7) Receive output ($\textsc{GotSecrets}$, $(secret_1, \dots, secret_v)$) from $\mathcal{F}_{\text{AS}}$ to LE
(8) Parse warrant: $(\widetilde{W}_1, \dots, \widetilde{W}_v) \leftarrow \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \leftarrow \widetilde{W}_i$
(9) For i from 1 to v:
   (a) Check if there is an entry $(uid_i, vper_i, \cdot, ct_i)$ in $\mathsf{L}^{\text{user-sec}}_{\text{Secrets}}$. If yes (case: user is honest) go to step (9b), if not (case: user is corrupted) go to step (9c).
   (b) *Case: User is honest*
      (i) Retrieve the entry $(uid_i, vper_i, \cdot, ct_i)$ from $\mathsf{L}^{\text{user-sec}}_{\text{Secrets}}$.
      (ii) Compute $sec_{2,i}$ such that $sec_{1,i} \oplus sec_{2,i} = secret_i$
      (iii) Add $(ct_i, sec_{2,i})$ to $\mathsf{L}^{\text{user-sec}}_{\text{ReqNotReady}}$
      (iv) Go back to step (9) and continue with next item
   (c) *Case: User is corrupted*
      (i) Append ct_i to $\mathsf{L}_{\text{Requests}}$ of $\mathcal{F}_{\text{AD}}$
      (ii) Go back to step (9) and continue with next item
(10) Send ($\textsc{Request}$, pid_{LE}, $f_t(W)$, $|\widetilde{W}|$, v) to $\mathcal{A}$
(11) When $\mathcal{A}$ allows to deliver output, store $\widetilde{W}$ in list of warrants $\mathsf{L}_{\text{Warrants}}$ of $\mathcal{F}_{\text{AD}}$
(12) Give output ($\textsc{Request}$) to LE

- **Handover**
  Simulate honestly, but additionally execute the following steps at the end:
  (1) For each item $(ct_i, sec_{2,i})$ in $\mathsf{L}^{\text{user-sec}}_{\text{ReqNotReady}}$, append $(ct_i, sec_{2,i})$ to $\mathsf{L}^{\text{Pending}}_{\text{Requests}}$ of $\mathcal{F}_{\text{AD}}$
  (2) Clear $\mathsf{L}^{\text{user-sec}}_{\text{ReqNotReady}}$

System Init (corrupted SO, honest J, honest AU):

(1) Upon receiving ($\textsc{Init}$, SO, vk_{SO}) from SO to $\mathcal{F}_{\text{AD}}$
   (a) Create signing keypair for J: $(\mathsf{vk}_{\text{J}}, \mathsf{sk}_{\text{J}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ and store it
   (b) Store vk_{SO}
   (c) Honestly simulate the Init-Task of $\mathcal{F}_{\text{AD}}$ with the following inputs:
       J $\rightarrow \mathcal{F}_{\text{AD}}$: ($\textsc{Init}$, J, vk_{J})
       AU $\rightarrow \mathcal{F}_{\text{AD}}$: ($\textsc{Init}$, AU)
       SO $\rightarrow \mathcal{F}_{\text{AD}}$: ($\textsc{Init}$, SO, vk_{SO})
   (d) During simulation of the Init-Task, store all generated variables and lists and leak all necessary information to the adversary.
   (e) Call $\mathcal{F}_{\text{AS}}$ in the name of SO with input ($\textsc{Init}$, SO)
(2) Upon receiving output ($\textsc{InitFinished}$) from $\mathcal{F}_{\text{AS}}$ to SO
   (a) Report message ($\textsc{InitFinished}$) from $\mathcal{F}_{\text{AD}}$ to SO
   (b) Send ($\textsc{GetSKeys}$) and ($\textsc{GetPK}$) to the adversary
   (c) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to all honest parties

Party Init (honest P):

(1) Upon receiving the leak ($\textsc{PInit}$) from $\mathcal{F}_{\text{AS}}$
   (a) Send ($\textsc{GetSKeys}$) and ($\textsc{GetPK}$) to the adversary
   (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to all parties

Party Init (corrupted P):
This is handled by simulation of $\mathcal{F}_{\mathrm{AD}}$.

User Registration (honest U, corrupted SO):

(1) Upon receiving the leak (Register, *pid*, *uid*) from $\mathcal{F}_{\mathrm{AS}}$
  (a) Report message (*uid*) from U to SO
(2) Upon receiving the message (uid_ok) from SO to U
  (a) Call $\mathcal{F}_{\mathrm{AS}}$ in the name of SO with input (Register, *uid*) and send (Ok) to $\mathcal{F}_{\mathrm{AS}}$
  (b) If $\mathcal{F}_{\mathrm{AS}}$ aborts, let $\mathcal{S}$ abort as well (User already registered or UID already taken)
  (c) Generate keypair for U: $(\mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
  (d) Honestly simulate U's Register and Retrieve calls to $\mathcal{F}_{\mathrm{BB}}$ with the correct inputs
  (e) When $\mathcal{F}_{\mathrm{AS}}$ wants to deliver output to U, delay it and report message (Ok) from U to SO
(3) Upon receiving the message $(\sigma_{\mathrm{reg}}, com_{\mathrm{uid}}, decom_{\mathrm{uid}})$ from SO to U
  (a) Send (GetSKeys) and (GetPK) to the adversary
  (b) Verify Commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathrm{com}}, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, uid) = 0$, abort.
  (c) Verify Signature: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{SO}}, \sigma_{\mathrm{reg}}, com_{\mathrm{uid}}) = 0$, abort.
  (d) Store $(pid_{\mathrm{U}}, uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}}, \mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}})$ in $\mathsf{L}^{\mathsf{user-sec}}_{\mathsf{Registered}}$
  (e) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output to U

User Registration (corrupted U, corrupted SO):
This is handled by simulation of $\mathcal{F}_{\mathrm{AD}}$.

Store Secret (honest U, corrupted SO):

(1) Upon receiving the leak (StoreSecret, *vper*) from $\mathcal{F}_{\mathrm{AS}}$
  (a) Call $\mathcal{F}_{\mathrm{AS}}$ in the name of SO with input (StoreSecret, *vper*)
  (b) If $\mathcal{F}_{\mathrm{AS}}$ aborts, abort as well (Either wrong validity period or user is not registered yet or user has already stored a secret for the current validity period)
  (c) Retrieve output (SecretStored, *uid*) from $\mathcal{F}_{\mathrm{AS}}$ to SO, but delay output to U
  (d) For *uid*, retrieve entry $(pid_{\mathrm{U}}, uid, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \sigma_{\mathrm{reg}}, \mathsf{vk}_{\mathrm{U}}, \mathsf{sk}_{\mathrm{U}})$ from $\mathsf{L}^{\mathsf{user-sec}}_{\mathsf{Registered}}$
  (e) Create fake commitment on sec_2: $(\widetilde{com_{sec_2}}, eq_2) \leftarrow \mathsf{COM.SimCom}(\mathsf{td}^{\mathrm{eq}}_{\mathrm{com}})$
  (f) Create real commitment on *vper*: $(com_{\mathrm{vper}}, decom_{\mathrm{vper}}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}^{\mathrm{eq}}_{\mathrm{com}}, vper)$
  (g) Report message $(uid, \sigma_{\mathrm{reg}}, com_{\mathrm{uid}}, decom_{\mathrm{uid}}, \widetilde{com_{sec_2}}, com_{\mathrm{vper}}, decom_{\mathrm{vper}})$ from U to SO
(2) Upon receiving the message (sec_1) from SO to U
  (a) Create fake commitment on *secret*: $(\widetilde{com_{\mathrm{sec}}}, eq_s) \leftarrow \mathsf{COM.SimCom}(\mathsf{td}^{\mathrm{eq}}_{\mathrm{com}})$
  (b) Fake ciphertext $ct \leftarrow \mathsf{TPKE.Enc}(\mathsf{pk}, 0)$
  (c) Honestly commit and sign sec_1: $(com_{sec_1}, decom_{sec_1}) \leftarrow \mathsf{COM.Com}(\mathsf{crs}_{\mathrm{com}}, sec_1)$ and $\sigma_{sec_1} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{U}}, com_{sec_1})$
  (d) Create signature $\sigma_{\mathrm{U}} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{U}}, (ct, uid, vper))$
  (e) $stmt_{\mathrm{ss}} \coloneqq (sec_1, \widetilde{com_{\mathrm{sec}}}, \widetilde{com_{sec_2}}, \mathsf{pk}, ct, \mathsf{crs}_{\mathrm{com}})$
  (f) Fake proof: $\widetilde{\pi_{\mathrm{ss}}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathrm{NIZK}}, stmt_{\mathrm{ss}}, \mathcal{R}^{AS}_{ss})$
  (g) Report message $(ct, com_{\mathrm{sec}}, \pi_{\mathrm{ss}}, \sigma_{\mathrm{U}}, \sigma_{sec_1}, com_{sec_1}, decom_{sec_1})$ from U to SO
(3) Upon receiving the message (σ_{ss}) from SO to U
  (a) Verify signature: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{SO}}, (com_{\mathrm{uid}}, \widetilde{com_{\mathrm{sec}}}, com_{\mathrm{vper}}), \sigma_{\mathrm{ss}}) = 0$, let $\mathcal{S}$ abort in the name of U
  (b) Store $(uid, vper, sec_1, ct)$ in $\mathsf{L}^{\mathsf{user-sec}}_{\mathsf{Secrets}}$
  (c) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output to U

Store Secret (corrupted U, corrupted SO):
Nothing to do, since Π_{AS}.StoreSecret has no calls to subfunctionalities and all involved parties are corrupted.

Request Warrant (corrupted LE, honest J, corrupted SO):

(1) Upon receiving the message (W) from LE to J
  (a) Call $\mathcal{F}_{\text{AS}}$ in the name of LE with input (RequestWarrant, W)
  (b) Allow $\mathcal{F}_{\text{AS}}$ to deliver output to J
(2) Upon receiving the message (RequestWarrant, W, b) from $\mathcal{F}_{\text{AS}}$
  (a) If $b = 0$, set $\sigma_W = \bot$. Else, set $\sigma_W = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{J}}, W)$
  (b) Store (W, b) in list of already requested preliminary warrants
  (c) Report message (b, σ_W) from J to LE
(3) Upon receiving the message $(\widetilde{W}, \{\sigma_{\text{U,i}}, sec_{1,i}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}\}_{i\in\{1,\dots,v\}})$ from LE to J
  (a) Build preliminary warrant W'' out of $\widetilde{W}$ by deleting ct_i from each $\widetilde{W_i}$
  (b) If there is no entry $(W'', 1)$ in the list of already processed preliminary warrants, abort in the name of J
  (c) Check if $(\widetilde{W}, \cdot)$ is already in list of stored enhanced warrants. If yes, abort in the name of J (Warrant already processed).
  (d) Parse enhanced warrant: $(\widetilde{W_1}, \dots, \widetilde{W_v}) \leftarrow \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \leftarrow \widetilde{W_i}$
  (e) For i from 1 to v:
    (i) For uid_i, honestly simulate a call to $\mathcal{F}_{\text{BB}}$ in the name of J with (Retrieve, uid_i) and get answer (Retrieve, $pid_{\text{U,i}}, uid_i, \mathsf{vk}_{\text{U,i}}$)
    (ii) Check signatures: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{U,i}}, (ct_i, uid_i, vper_i), \sigma_{\text{U,i}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\text{U,i}}, com_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}) = 0$, abort in the name of J
    (iii) Check commitment: If $\mathsf{COM.Open}(\mathsf{crs_{com}}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, sec_{1,i}) = 0$, abort in the name of J
  (f) Sign enhanced warrant: $\sigma_{\widetilde{W}} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{J}}, \widetilde{W})$
  (g) Store $(\widetilde{W}, \sigma_{\widetilde{W}})$ in list of stored enhanced warrants
  (h) Report message $(\sigma_{\widetilde{W}})$ from J to LE
(4) When the adversary allows $\mathcal{F}_{\text{AD}}$ to deliver output in $\mathcal{F}_{\text{AD}}$.RequestDecryption
  (a) Finish simulation of $\mathcal{F}_{\text{AD}}$.RequestDecryption (like described above)
  (b) Send (Ok) to $\mathcal{F}_{\text{AS}}$
  (c) Receive output (WarrantAnswer, b) from $\mathcal{F}_{\text{AS}}$

Request Warrant (honest LE, honest J, corrupted SO):

(1) Upon $\mathcal{F}_{\text{AS}}$ asking to deliver output to J, allow the output
(2) Upon receiving the leak (RequestWarrant, W, b) from $\mathcal{F}_{\text{AS}}$
  (a) If $b = 0$, set $\sigma_W = \bot$. Else, set $\sigma_W = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\text{J}}, W)$
  (b) Store (W, b) in list of already requested preliminary warrants
  (c) If $b = 0$, directly go to step (4b)
  (d) Send (GetSKeys) to the adversary
  (e) Report message (W, σ_W) from LE to SO
(3) Upon receiving the message $(\widetilde{W}, \{\sigma_{\text{U,i}}, sec_{1,i}, com_{sec_1,\text{i}}, decom_{sec_1,\text{i}}, \sigma_{sec_1,\text{i}}\}_{i\in\{1,\dots,v\}})$ from SO to LE
  (a) Parse enhanced warrant: $(\widetilde{W_1}, \dots, \widetilde{W_v}) \leftarrow \widetilde{W}$ and $(uid_i, vper_i, meta_i, ct_i) \leftarrow \widetilde{W_i}$
  (b) Build preliminary warrant W' out of $\widetilde{W}$ by deleting ct_i from each $\widetilde{W_i}$
  (c) If $W' \neq W$, abort in the name of LE. (SO sent wrong $\widetilde{W}$)
  (d) For i from 1 to v:

(i) For uid_i, honestly simulate a call to $\mathcal{F}_{\mathrm{BB}}$ in the name of LE with (Retrieve, uid_i) and get answer (Retrieve, $pid_{\mathrm{U,i}}$, uid_i, $\mathsf{vk}_{\mathrm{U,i}}$)

(ii) Check signatures: If $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, (ct_i, uid_i, vper_i), \sigma_{\mathrm{U,i}}) = 0$ or $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{\mathrm{U,i}}, com_{sec_1,\mathrm{i}}, \sigma_{sec_1,\mathrm{i}}) = 0$, abort in the name of LE

(iii) Check commitment: If $\mathsf{COM.Open}(\mathsf{crs}_{\mathsf{com}}, com_{sec_1,\mathrm{i}}, decom_{sec_1,\mathrm{i}}, sec_{1,i}) = 0$, abort in the name of LE

(iv) For uid_i, honestly simulate a call to $\mathcal{F}_{\mathrm{BB}}$ in the name of J with (Retrieve, uid_i)

(e) Sign enhanced warrant: $\sigma_{\widetilde{W}} = \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathrm{J}}, \widetilde{W})$

(f) Store $(\widetilde{W}, \sigma_{\widetilde{W}})$ in list of stored enhanced warrants

(g) Simulate call to $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption by sending (Request, $pid_{\mathrm{LE}}, f_t(W), |\widetilde{W}|, v$) to $\mathcal{A}$

(4) When the adversary allows $\mathcal{F}_{\mathrm{AD}}$ to deliver output in $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption

(a) Store $\widetilde{W}$ in list of warrants $\mathsf{L}_{\mathsf{Warrants}}$ of $\mathcal{F}_{\mathrm{AD}}$

(b) Send (Ok) to $\mathcal{F}_{\mathrm{AS}}$

(c) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output to LE

Get Secrets (honest LE):

(1) Upon receiving the leak (GetSecrets) from $\mathcal{F}_{\mathrm{AS}}$

(a) Leak (Retrieve) to $\mathcal{A}$

(b) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output

Get Secrets (corrupted LE):
This is handled by simulation of $\mathcal{F}_{\mathrm{AD}}$.

Get Statistics (honest P):

(1) Upon receiving the leak (GetStatistics) from $\mathcal{F}_{\mathrm{AS}}$

(a) Leak (GetStatistics) to $\mathcal{A}$ and wait for okay from $\mathcal{A}$

(b) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output

Get Statistics (corrupted P):
This is handled by simulation of $\mathcal{F}_{\mathrm{AD}}$.

Audit (honest AU):

(1) Upon receiving the leak (AuditRequest) from $\mathcal{F}_{\mathrm{AS}}$

(a) Leak (AuditRequest) to $\mathcal{A}$ and wait for okay from $\mathcal{A}$

(b) Allow $\mathcal{F}_{\mathrm{AS}}$ to deliver output

Prove (corrupted U):
Nothing to do since this is a completely local task.

Prove (honest U):

(1) Upon receiving the message (Prove, $stmt_R$) from $\mathcal{F}_{\mathrm{AS}}$

(a) Simulate proof: $\pi \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathrm{NIZK}}, stmt_R, \mathcal{R}_{zk}^{AS})$

(b) Send (Proof, π) to $\mathcal{F}_{\mathrm{AS}}$

Verify (corrupted P):
Nothing to do since this is a completely local task.

Verify (honest P):

(1) Upon receiving the message (VERIFY, $stmt_R, \pi$) from $\mathcal{F}_{\mathrm{AS}}$
  (a) Extract Witness: $wit_R \leftarrow \mathsf{NIZK.Ext}(\mathsf{td}_{\mathsf{NIZK}}, stmt_R, \pi, \mathcal{R}_{zk}^{AS})$
  (b) Send (WITNESS, wit_R) to $\mathcal{F}_{\mathrm{AS}}$

We now prove the following theorem:

Theorem B.3 (User Security). Π_{AS} *UC-realizes* $\mathcal{F}_{\mathrm{AS}}$ *in the* $\{\mathcal{F}_{\mathrm{AD}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BB}}, \mathcal{G}_{\mathrm{CLOCK}}\}$*-hybrid model under the assumptions that*

- COM *is a (computationally) hiding, (statistically) binding and (dual-mode) extractable and equivocable commitment scheme,*
- Σ *is a* EUF-CMA*-secure signature scheme,*
- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- *and* TPKE *is* IND-CPA*-secure*

against all PPT*-adversaries* $\mathcal{A}$ *who statically corrupt either*

(1) SO and a subset of the users, or

(2) SO, LE and a subset of the users.

To prove Theorem B.3, we proceed in a series of games, where $\mathcal{F}_{\mathrm{AS}}^{i+1}$ behaves the same as $\mathcal{F}_{\mathrm{AS}}^{i}$ except for the stated changes and $\mathcal{S}^{i+1}$ behaves the same as $\mathcal{S}^{i}$ except for the stated changes.

Game 0. Game 0 is equivalent to the real experiment. That is,

$$H_0 \coloneqq \mathsf{Exp}^{\Pi_{\mathrm{AS}}}_{\mathcal{F}_{\mathrm{AD}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BB}}, \mathcal{G}_{\mathrm{CLOCK}}, \mathcal{S}^0, \mathcal{Z}}$$

with $\mathcal{S}^0$ being the dummy adversary. This means that all parties execute the real protocol.

Game 1. In this game, the simulator $\mathcal{S}^1$ takes control over $\mathcal{F}_{\mathrm{CRS}}$, $\mathcal{F}_{\mathrm{BB}}$ and $\mathcal{F}_{\mathrm{AD}}$, but executes them honestly. Additionally, $\mathcal{F}_{\mathrm{AS}}^1$ is introduced, and all honest parties are replaced by dummy parties that forward their inputs to $\mathcal{F}_{\mathrm{AS}}^1$ and when receiving output from $\mathcal{F}_{\mathrm{AS}}^1$ forward it to the environment. $\mathcal{F}_{\mathrm{AS}}^1$, upon receiving any message from a dummy party forwards it to the simulator $\mathcal{S}^1$ and asks it for output. $\mathcal{S}^1$ executes the real protocol for all honest parties (using the inputs received from $\mathcal{F}_{\mathrm{AS}}^1$) and instructs $\mathcal{F}_{\mathrm{AS}}^1$ to deliver the resulting outputs. Note that $\mathcal{S}^1$ simulates $\mathcal{F}_{\mathrm{BB}}$ and $\mathcal{F}_{\mathrm{CRS}}$ like $\mathcal{S}$ does.

Proof Sketch: Game 0 ≈ Game 1.
$\mathcal{S}^1$ executes the same code as the real parties on the same inputs (that have been forwarded by $\mathcal{F}_{\mathrm{AS}}^1$). Thus, Game 0 and Game 1 are identical from the environments view.

Game 2. In this game, $\mathcal{S}^2$ generates the common reference string for the commitment scheme with an equivocation trapdoor $\mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}$ as $(\mathsf{crs}_{\mathsf{com}}, \mathsf{td}_{\mathsf{com}}^{\mathsf{eq}}) \leftarrow \mathsf{COM.ExtSetup}(1^\lambda)$. $\mathcal{S}^2$ also generates the common reference string for the non-interactive proof system with a trapdoor for simulating and extracting proofs, i.e., $(\mathsf{crs}_{\mathsf{zk}}^{\mathsf{AS}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. Thus, $\mathcal{S}^2$ simulates $\mathcal{F}_{\mathrm{CRS}}$ the same way $\mathcal{S}$ does.

Proof Sketch: Game 1 ≈ Game 2.
Indistinguishability follows from the equivocality of COM and the simulation-extractability of NIZK.

Game 3. During setup of $\mathcal{F}_{\text{AD}}$, $\mathcal{S}^3$ is asked for the keypairs for J and AU. $\mathcal{S}^3$ generates the keypairs honestly and stores them.

Proof Sketch: Game 2 ≈ Game 3.
$\mathcal{S}^3$ runs the same key generation algorithm as the real protocol does for honest parties, therefore $\mathcal{Z}$'s view remains unchanged.

Game 4. $\mathcal{S}^4$ keeps track of additional information. For J it stores a list of (W, b) elements and a list of $(\widetilde{W}, \sigma_{\widetilde{W}})$ elements in the Request Warrant task. For each honest U, it stores $(\mathit{pid}_{\text{U}}, \mathit{uid}, \mathit{com}_{\text{uid}}, \mathit{decom}_{\text{uid}}, \sigma_{\text{reg}}, \mathsf{vk}_{\text{U}}, \mathsf{sk}_{\text{U}})$ in the list $\mathsf{L}^{\mathsf{user-sec}}_{\mathsf{Registered}}$ at the end of the User Registration task and $(\mathit{uid}, \mathit{vper}, \mathit{sec}_1, \mathit{ct})$ in the list $\mathsf{L}^{\mathsf{user-sec}}_{\mathsf{Secrets}}$ at the end of the Store Secret task.

Proof Sketch: Game 3 ≈ Game 4.
$\mathcal{S}^4$ only keeps track of more information (that the honest parties acquire during the real protocol), nothing to distinguish for $\mathcal{Z}$.

Game 5. In this game, $\mathcal{F}^5_{\text{AS}}$ additionally handles the tasks System Init and Party Init the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^5$ handles these task the same way $\mathcal{S}$ does.

Proof Sketch: Game 4 ≈ Game 5.
$\mathcal{S}^5$ simulates the System Init task honestly, therefore $\mathcal{Z}$s view remains unchanged.
Party Init for honest Parties: $\mathcal{Z}$s view remains identical, since $\mathcal{S}^5$ only needs to leak (GetSKeys) and (GetPK) to $\mathcal{A}$ to emulate the call to $\mathcal{F}_{\text{AD}}$.
Party Init for corrupted Parties: Only the simulation of hybrid functionalities needs to be handled, which are already simulated honestly.

Game 6. In this game, $\mathcal{F}^6_{\text{AS}}$ additionally handles the Audit task the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^6$ handles this task the same way $\mathcal{S}$ does. Note that since the auditor AU is assumed to be honest, simulation only requires a simulated (AuditRequest) to $\mathcal{F}_{\text{BCRA}}$.

Proof Sketch: Game 5 ≈ Game 6.
Since AU is honest, these games are identical.

Game 7. In this game, $\mathcal{F}^7_{\text{AS}}$ additionally handles the User Registration task the same way $\mathcal{F}_{\text{AS}}$ does, and $\mathcal{S}^7$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 6 ≈ Game 7.
For honest U, $\mathcal{S}^7$ learns the *uid* the environment has chosen for the user and can thus invoke $\mathcal{F}_{\text{AS}}$ with the correct input. Apart from that, $\mathcal{S}^7$ plays the role of the user honestly. If the signature scheme Σ is EUF-CMA-secure and the commitment scheme COM is binding, $\mathcal{S}^7$ aborts in the same cases as U would do in Π_{AS}.
For corrupted U, this game is identical to Game 6 since this task is already handled by the simulation of $\mathcal{F}_{\text{AD}}$.

Game 8. If LE is honest, $\mathcal{F}^8_{\text{AS}}$ additionally handles the Request Warrant task the same way $\mathcal{F}_{\text{AS}}$ does for honest LE, and $\mathcal{S}^8$ handles this task the same way $\mathcal{S}$ does for honest LE. Note that if LE is corrupted, this game changes nothing.

Proof Sketch: Game 7 ≈ Game 8.
$\mathcal{S}^8$ essentially simulates the real protocol for honest parties (only the call to $\mathcal{F}_{\text{AD}}$ is simulated instead of executed honestly), therefore $\mathcal{Z}$ can not distinguish those games. The simulated call to $\mathcal{F}_{\text{AD}}$ is indistinguishable since $\mathcal{S}^8$ leaks the correct information to the adversary. Note that if the signature scheme Σ is EUF-CMA-secure and the commitment scheme COM is binding, $\mathcal{S}^8$ aborts in the same cases as LE or J would do in Π_{AS}.

Game 9. If LE is corrupted, $\mathcal{F}_{\mathrm{AS}}^{9}$ additionally handles the Request Warrant task the same way $\mathcal{F}_{\mathrm{AS}}$ does for corrupted LE, and $\mathcal{S}^{9}$ handles this task the same way $\mathcal{S}$ does for corrupted LE. Additionally, $\mathcal{S}^{9}$ simulates $\mathcal{F}_{\mathrm{AD}}$ as $\mathcal{S}$ does for the Request Decryption and Handover tasks in $\mathcal{F}_{\mathrm{AD}}$. Note that if LE is honest, this game changes nothing. Also note that now $\mathcal{S}^{9}$ simulates $\mathcal{F}_{\mathrm{AD}}$ as $\mathcal{S}$ does (for every task and every corruption scenario).

Proof Sketch: Game 8 ≈ Game 9.
In Request Warrant, $\mathcal{S}^{9}$ receives the necessary information from LE to call $\mathcal{F}_{\mathrm{AS}}^{9}$ with the correct input. After receiving output from $\mathcal{F}_{\mathrm{AS}}^{9}$, $\mathcal{S}^{9}$ essentially executes the real protocol for J and SO. Now, the simulation of $\mathcal{F}_{\mathrm{AD}}$.RequestDecryption needs to be handled as well. Since Σ is EUF-CMA-secure, $\mathcal{S}^{9}$ aborts in the same cases as the real $\mathcal{F}_{\mathrm{AD}}$ would. $\mathcal{S}^{9}$ can call $\mathcal{F}_{\mathrm{AS}}^{9}$ with the correct inputs and gets the corresponding outputs. Next, the secrets need to be prepared for retrieval. Here we need to distinguish between secrets of honest and corrupted users. For corrupted users, we already know the correct ciphertext (from the input of this task) and we can just append the ciphertext to $\mathsf{L}_{\mathsf{Requests}}$ of $\mathcal{F}_{\mathrm{AD}}$ like an honest $\mathcal{F}_{\mathrm{AD}}$. The remainder of the secret retrieval (for corrupted users) is then handled by the honest simulation of $\mathcal{F}_{\mathrm{AD}}$. For honest users, we retrieve the entry $(uid, vper, sec_1, ct)$ from $\mathsf{L}_{\mathsf{Secrets}}^{\mathsf{user-sec}}$ and use the output of $\mathcal{F}_{\mathrm{AS}}^{9}$ to calculate the secret sec_2 (in the clear) that needs to be retrieved. We then store this secret directly to a special list $\mathsf{L}_{\mathsf{ReqNotReady}}^{\mathsf{user-sec}}$. Then, the $\mathcal{F}_{\mathrm{AD}}$.Handover task has to be adapted as well. It now additionally moves all entries from $\mathsf{L}_{\mathsf{ReqNotReady}}^{\mathsf{user-sec}}$ to $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Pending}}$. This ensures that now for honest users the correct secrets can be retrieved as well. Note that for honest users, the ciphertext encrypting the user's secret in Store Secret is not used during retrieval anymore.

Game 10. In this game, $\mathcal{F}_{\mathrm{AS}}^{10}$ additionally handles the tasks Get Secrets and Get Statistics the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^{10}$ handles these tasks the same way $\mathcal{S}$ does.

Proof Sketch: Game 9 ≈ Game 10.
For a corrupted party calling those tasks, this game is identical to Game 9 since those tasks are already handled by the simulation of $\mathcal{F}_{\mathrm{AD}}$. For honest parties, $\mathcal{S}^{10}$ only needs to leak the same messages to the adversary as $\mathcal{F}_{\mathrm{AD}}$ would do. All the information $\mathcal{S}^{10}$ needs for that are provided by leaks from $\mathcal{F}_{\mathrm{AS}}$.

Game 11. In this game, $\mathcal{F}_{\mathrm{AS}}^{11}$ additionally handles the Store Secret task the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^{11}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 10 ≈ Game 11.
For corrupted U: nothing to do in this case.
For honest U: With the message (StoreSecret, *vper*) $\mathcal{S}^{11}$ is able to invoke $\mathcal{F}_{\mathrm{AS}}^{11}$ with the correct inputs for SO. Instead of computing real commitments on sec_2 and *secret*, $\mathcal{S}^{11}$ computes equivocal commitments on them. Since COM is hiding and equivocable, $\mathcal{Z}$ can not differentiate between the real and the simulated commitments. Also, $\mathcal{S}^{11}$ simulates the proof π_{ss} instead of computing a real proof for honest users. Due to the zero-knowledge property of NIZK, $\mathcal{Z}$ can not detect this. $\mathcal{S}^{11}$ also creates a fake ciphertext (by encrypting 0) which is not detected by $\mathcal{Z}$ since TPKE is IND-CPA-secure and we do not use this ciphertext during retrieval anymore. Apart from these changes, $\mathcal{S}^{11}$ simulates the role of the user honestly.

Game 12. In this game, $\mathcal{F}_{\mathrm{AS}}^{12}$ additionally handles the Prove and Verify tasks the same way $\mathcal{F}_{\mathrm{AS}}$ does, and $\mathcal{S}^{12}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 11 ≈ Game 12.
For corrupted parties invoking this task, there is nothing to do for $\mathcal{S}^{12}$, since both tasks are *local tasks.* So let us assume the calling parties are honest in the following. Since NIZK is a simulation extractable zero-knowledge proof system, $\mathcal{Z}$ can not distinguish between the outputs it gets from $\mathcal{F}_{\mathrm{AS}}^{12}$ and the outputs it gets from $\mathcal{F}_{\mathrm{AS}}$. Note that in the $\mathcal{F}_{\mathrm{AS}}$.Verify task, it may be the case that no corresponding

entry exists in $\mathsf{L}_{\mathsf{Secrets}}$. But since SO is corrupted, the additional check in $\mathcal{F}_{\mathrm{AS}}$.Verify that involves $\mathsf{L}_{\mathsf{Secrets}}$ is skipped and the correct output is given.

Note that $\mathcal{S}^{12}$ equals the final simulator $\mathcal{S}$. Since we showed that Game i is indistinguishable from Game $i + 1$ for $i \in \{0, \ldots, 11\}$, it follows that Game 0 (the real protocol) is indistinguishable from Game 12 (ideal execution) and thus Theorem B.3 follows. ☐

Proof of Theorem 4.3. The theorem is directly implied by Theorem B.2 and Theorem B.3. ☐

B.2. Security Proof: Π_{AD} UC/YOSO-realizes $\mathcal{F}_{\mathrm{AD}}$

Here we prove that the protocol Π_{AD} from Section 4.6 UC-realizes the functionality $\mathcal{F}_{\mathrm{AD}}$ from Section 4.5 in the YOSO model. In particular, we prove the following theorem:

Theorem 4.4. Π_{AD} *UC-realizes* $\mathcal{F}_{\mathrm{AD}}$ *in the* $\{\mathcal{F}_{\mathrm{BCRA}}, \mathcal{F}_{\mathrm{CRS}}, \mathcal{G}_{\mathrm{CLOCK}}\}$*-hybrid model under the assumptions that*

- NIZK *is a straight-line simulation-extractable non-interactive zero-knowledge proof system,*
- Σ *is an* EUF-CMA*-secure signature scheme,*
- *the* PKE *scheme used by LE and AU is an* IND-CPA*-secure public key encryption scheme,*
- *the* PKE *scheme that is a parameter of* $\mathcal{F}_{\mathrm{BCRA}}$ *is a* RIND-SO*-secure public key encryption scheme,*
- *and* TPKE *is an* IND-CPA*-secure, simulatable, re-randomizable and binding* (t, n)*-threshold PKE*

against all PPT*-adversaries* $\mathcal{A}$ *who may statically corrupt SO and/or LE as well as mobile adaptively corrupt at most a fraction* $\frac{t}{n} - \epsilon$ *of nodes N.*

First, note that Π_{AD} is YOSO when considering SO, LE, J and AU as input/output roles: Nodes N_i only send one (batch of) outgoing message(s) and do not retain state after doing so. Thus, we can simplify our corruption treatment in the following security proof to static corruptions.

We first describe the simulator $\mathcal{S}$ and then proceed to prove indistinguishability between the real and ideal worlds. Again, we use the convention that when $\mathcal{F}_{\mathrm{AD}}$ wants to deliver output to an honest party, the adversary is notified about this (including the relevant task name, but not the other output) and output is only delivered after the adversary allows to do so.

The Simulator $\mathcal{S}$ is defined as follows.

Simulator $\mathcal{S}$

State of the Simulator:

- Setup:
  - SO verification key $\mathsf{vk}_{\mathrm{SO}}$
  - J verification key vk_{J}
  - AU encryption keypair $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}})$
  - Functionality public key pk
  - Common reference string $\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}$
  - Trapdoor td for extracting and simulating zero-knowledge proofs

- Lists to keep track of simulation:
  - $\mathsf{L}_{\mathsf{Secrets}}$ to store shares of retrieved secrets
  - ${\mathsf{L}_{\mathsf{CommitteePK}}}^{i}$: List of public keys of the committee with number i. Entries are of the form (ek_j).
  - ${\mathsf{L}_{\mathsf{Committee}}}^{i}$: List of keys for the committee with number i. Entries are of the form $(\mathsf{ek}_j, \mathsf{dk}_j, \mathsf{vk}_j, \mathsf{sk}_j)$.
  - For each committee a set $\mathcal{H}$ of honest roles, a set $\mathcal{B}$ of corrupted roles and a set $\mathcal{SH}$ of corruptible roles.
- State of $\mathcal{F}_{\text{BCRA}}$:
  - Set Posted, initialized to $\emptyset$
  - Sequence Ordered, initialized to ()

Simulation of $\mathcal{F}_{\text{CRS}}$:
A corrupted party P can issue calls to $\mathcal{F}_{\text{CRS}}$ whenever it wants. Since simulation of $\mathcal{F}_{\text{CRS}}$ is straightforward, we specify here once how all calls from corrupted P to $\mathcal{F}_{\text{CRS}}$ are handled and omit calls to $\mathcal{F}_{\text{CRS}}$ in the following description of the simulator.

- Upon receiving (value) from corrupted P to $\mathcal{F}_{\text{CRS}}$:
  (1) If this is the first time such a message is received:
    (a) Generate and store $(\mathsf{crs}_{\mathsf{zk}}^{\text{AD}}, \mathsf{td}) \leftarrow \mathsf{NIZK.Setup}()$
  (2) Report message $\mathsf{crs}_{\mathsf{zk}}^{\text{AD}}$ from $\mathcal{F}_{\text{CRS}}$ to P

Simulation of $\mathcal{F}_{\text{BCRA}}$:

- Upon receiving (Post), (Order) or (Read) from corrupted P, simulate $\mathcal{F}_{\text{BCRA}}$ honestly
- Upon receiving (NextCommittee, R, n, S) with $|S| < \epsilon n$ containing entries of the form ($M_i \in$ Machine, $(\mathsf{ek}_i, \mathsf{vk}_i)$) from the adversary
  (1) Let j be the id of the next committte. If ${\mathsf{L}_{\mathsf{Committee}}}^{j} \neq \emptyset$, ignore this message.
  (2) Otherwise, send (Clock-Read) to $\mathcal{G}_{\text{CLOCK}}$ and receive t_{Now}
    - For $i \in (1, \ldots, |S|)$:
      (a) Mark M_i as corrupted
      (b) Add $(R_i, (\mathsf{ek}_{R_i}, \bot, \mathsf{vk}_{R_i}, \bot))$ to ${\mathsf{L}_{\mathsf{Committee}}}^{j}$
      (c) Set m = (NextCommittee, $R, \mathsf{ek}_i, \mathsf{vk}_i$), add $(\mathsf{roleassign}, t_{\text{Now}}, m)$ to Posted and send m to the adversary on behalf of $\mathcal{F}_{\text{BCRA}}$
    - For $i \in (|S| + 1, \ldots, n)$:
      (a) Sample $(\mathsf{ek}_{R_i}, \mathsf{dk}_{R_i}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$
      (b) Sample $(\mathsf{vk}_{R_i}, \mathsf{sk}_{R_i}) \leftarrow \Sigma\mathsf{.Gen}(1^\lambda)$
      (c) Add $(R_i, (\mathsf{ek}_{R_i}, \mathsf{dk}_{R_i}, \mathsf{vk}_{R_i}, \mathsf{sk}_{R_i}))$ to ${\mathsf{L}_{\mathsf{Committee}}}^{j}$
      (d) Set m = (NextCommittee, $R, \mathsf{ek}_{R_i}, \mathsf{vk}_{R_i}$), add $(\mathsf{roleassign}, t_{\text{Now}}, m)$ to Posted and send m to the adversary on behalf of $\mathcal{F}_{\text{BCRA}}$

Init (honest SO):

- Upon receiving (Init, SO, vk_{SO}, pk) from $\mathcal{F}_{\text{AD}}$
  (1) Store pk as the functionality public key
  (2) Send (Clock-Read) to $\mathcal{G}_{\text{CLOCK}}$ and receive t_{Now}
  (3) Set $m \coloneqq$ (OperatorKey, vk_{SO})
  (4) Add $(pid_{\text{SO}}, t_{\text{Now}}, m)$ to Posted and send (Post, pid_{SO}, m) to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BCRA}}$
  (5) Allow $\mathcal{F}_{\text{AD}}$ to deliver output

Init (corrupted SO):

- Upon receiving $(\textsc{Post}, \textsc{OperatorKey}, \mathsf{vk}_{\mathrm{SO}})$ from SO to $\mathcal{F}_{\mathrm{BCRA}}$
  (1) Send $(\textsc{Clock-Read})$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive t_{Now}
  (2) Set $m \coloneqq (\textsc{OperatorKey}, \mathsf{vk}_{\mathrm{SO}})$
  (3) Add $(pid_{\mathrm{SO}}, t_{\mathrm{Now}}, m)$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid_{\mathrm{SO}}, m)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (4) Send $(\textsc{Init}, \mathrm{SO}, \mathsf{vk}_{\mathrm{SO}})$ to $\mathcal{F}_{\mathrm{AD}}$ on behalf of SO
  (5) Upon receiving $(\textsc{Init}, \mathrm{SO}, \mathsf{vk}_{\mathrm{SO}}, \mathsf{pk})$ from $\mathcal{F}_{\mathrm{AD}}$, store pk and allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Init (honest J):

- Upon receiving $(\textsc{Init}, \mathrm{J}, \mathsf{vk}_{\mathrm{J}})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Clock-Read})$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive t_{Now}
  (2) Set $m \coloneqq (\textsc{JudgeKey}, \mathsf{vk}_{\mathrm{J}})$
  (3) Add $(pid_{\mathrm{J}}, t_{\mathrm{Now}}, m)$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid_{\mathrm{J}}, m)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (4) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Init (honest AU):

- Upon receiving $(\textsc{Init}, \mathrm{AU})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Clock-Read})$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive t_{Now}
  (2) Generate $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}}) \leftarrow \mathsf{PKE.Gen}(1^{\lambda})$ and store $(\mathsf{ek}_{\mathrm{AU}}, \mathsf{sk}_{\mathrm{AU}})$
  (3) Set $m \coloneqq (\textsc{AuditorKey}, \mathsf{ek}_{\mathrm{AU}})$
  (4) Add $(pid_{\mathrm{AU}}, t_{\mathrm{Now}}, m)$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid_{\mathrm{AU}}, m)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (5) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Get Public Key (honest P):

- Upon receiving $(\textsc{GetPK})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Read})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (2) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Get Public Key (corrupted P):

- This is handled by simulation of key generation roles and $\mathcal{F}_{\mathrm{BCRA}}$

Get System Keys (honest P):

- Upon receiving $(\textsc{GetSKeys})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Read})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (2) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Get System Keys (corrupted P):

- This is handled by simulation of $\mathcal{F}_{\mathrm{BCRA}}$

Request Decryption (honest LE):

- Upon receiving $(\textsc{Request}, pid, W^{\mathrm{pub}}, len, v)$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Clock-Read})$ to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive t_{Now}
  (2) If $\mathsf{sk}_{\mathrm{LE}} = \perp$ set $(\mathsf{ek}_{\mathrm{LE}}, \mathsf{sk}_{\mathrm{LE}}) \leftarrow \mathsf{PKE.Gen}(1^{\lambda})$
  (3) Send $(\textsc{Read})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (4) Generate $W^{\mathrm{enc}} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\mathrm{AU}}, 0^{len})$

(5) For $i = 1, \ldots, v$ do:
  (a) $ct_i \leftarrow \mathsf{TPKE.Enc}(\mathsf{pk}, 0)$
(6) $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}}, \{ct_i\}_{i\in[v]})$
(7) $\pi_{\widetilde{W}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{\widetilde{W}}, \mathsf{td})$
(8) Set $msg \coloneqq (\textsc{Request}, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \{ct_i\}_{i\in[v]}, \pi_{\widetilde{W}}, \mathsf{ek}_{\mathrm{LE}})$, add $(pid, t_{\mathrm{Now}}, msg)$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid, msg)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
(9) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Request Decryption (corrupted LE):

- Upon receiving $(\textsc{Post}, (\textsc{Request}, W^{\mathrm{pub}}, W^{\mathrm{enc}}, \{ct_i\}_{i\in[v]}, \pi_{\widetilde{W}}, \mathsf{ek}_{\mathrm{LE}}))$ from LE to $\mathcal{F}_{\mathrm{BCRA}}$
  (1) $stmt_{\widetilde{W}} \coloneqq (\mathsf{ek}_{\mathrm{AU}}, \mathsf{vk}_{\mathrm{J}}, W^{\mathrm{enc}}, W^{\mathrm{pub}}, \{ct_i\}_{i\in[v]})$
  (2) If $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt, \pi_{\widetilde{W}}) \neq 1$ ignore this message
  (3) Otherwise, extract $wit_{\widetilde{W}} \coloneqq (\widetilde{W}, r, \sigma_{\widetilde{W}}, \{ct_i, r_i\}_{i\in[v]}) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, \pi_{\widetilde{W}}, \mathsf{td})$
  (4) Send $(\textsc{Request}, \widetilde{W}, \sigma_{\widetilde{W}})$ to $\mathcal{F}_{\mathrm{AD}}$ on behalf of LE

Retrieve Secret (honest LE):

- Upon receiving $(\textsc{Retrieve})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Read})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (2) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Retrieve Secret (corrupted LE):

- This is handled during simulation of handover and $\mathcal{F}_{\mathrm{BCRA}}$

Get Statistics (honest P):

- Upon receiving $(\textsc{GetStatistics})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Read})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (2) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

Get Statistics (corrupted P):

- This is handled by simulation of $\mathcal{F}_{\mathrm{BCRA}}$

Auditor Audit (honest AU):

- Upon receiving $(\textsc{AuditRequest})$ from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send $(\textsc{Read})$ to $\mathcal{A}$
  (2) Allow $\mathcal{F}_{\mathrm{AD}}$ to deliver output

RoleExecute (honest N):
For each committee, after $\mathcal{F}_{\mathrm{BCRA}}$ delivered all secret keys, the simulator fills the three sets $\mathcal{H}, \mathcal{B}, \mathcal{SH}$ as follows:

- All roles that are already corrupted are assigned to $\mathcal{B}$
- $t - |\mathcal{B}|$ random roles that are not corrupted are assigned to $\mathcal{SH}$
- The remaining roles are assigned to $\mathcal{H}$

It holds that $\mathcal{SH} \bigcap \mathcal{B} = \emptyset$ and $\mathcal{H} \bigcap (\mathcal{SH} \bigcup \mathcal{B}) = \emptyset$ and $\mathcal{H} \bigcup \mathcal{SH} \bigcup \mathcal{B}$ contains all roles of that committee. Whenever the adversary corrupts a party that is assigned a role in $\mathcal{H}$ and has not executed yet, the simulator picks a random role $R_i \in \mathcal{SH}$ instead, sets $\mathcal{SH} \coloneqq \mathcal{SH} \setminus \{R_i\}$, $\mathcal{B} \coloneqq \mathcal{B} \bigcup \{R_i\}$ and returns the state of role R_i instead (which consists of that roles secret keys). When the adversary corrupts a party that is assigned a role R in $\mathcal{SH}$, the simulator sets $\mathcal{SH} \coloneqq \mathcal{SH} \setminus \{R\}$, $\mathcal{B} \coloneqq \mathcal{B} \bigcup \{R\}$ and returns the state of that role.

- Upon receiving ($\textsc{ExecuteRole}, pid_N$) from $\mathcal{F}_{\mathrm{AD}}$
  (1) Send ($\textsc{Read}$) to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
  (2) Send ($\textsc{Clock-Read}$) to $\mathcal{G}_{\mathrm{CLOCK}}$ and receive t_{Now}
  (3) If $t_{\mathrm{last}} + t_{\mathrm{com}} \leq t_{\mathrm{Now}}$, then execute **PrepareExecution**
  (4) If this node is scheduled to execute a role R_i this round, then execute the respective task **KeyGen1**, **KeyGen2** or **Handover** below.
- **PrepareExecution:**
  (1) Set $t_{\mathrm{last}} = t_{\mathrm{Now}}$
  (2) Initialize empty lists $\mathsf{L}_{\mathsf{CommitteePK}}$ and $\mathsf{L}_{\mathsf{CommitteeVK}}$
  (3) *Get encryption keys for next committee.*
  For each role R_j in the next committee:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$
    (b) Insert ek_R into $\mathsf{L}^{\mathsf{next}}_{\mathsf{CommitteePK}}$
  (4) *Get encryption keys for the current committee.*
  For each role R_j in the current committee:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$
    (b) Insert ek_R into $\mathsf{L}_{\mathsf{CommitteePK}}$
  (5) *Get verification keys for the previous two committees.*
  For each role R_j in the previous two committees:
    (a) Find $(\mathsf{roleassign}, t, (\textsc{Generate}, R_j, \mathsf{ek}_R, \mathsf{vk}_R))$ in $\textsc{Ordered}$
    (b) Insert vk_R into $\mathsf{L}_{\mathsf{CommitteeVK}}$
  (6) Assign the roles of the current committee to $\mathcal{H}, \mathcal{SH}, \mathcal{B}$ as described above
  (7) If KeyGen1 has been executed, but KeyGen2 not, do the following:
    (a) For each entry $(pid, t, (R_j, (\textsc{KeyGen1}, (\{ct_j\}_{j\in[n]}, \pi_{KG})), \sigma))$ in $\textsc{Ordered}$:
      (i) Retrieve vk_{R_j} from $\mathsf{L}_{\mathsf{CommitteeVK}}$
      (ii) Check if $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, (\textsc{KeyGen1}, (\{ct_j\}_{j\in[n]}, \pi_{KG})), \sigma) = 1$, otherwise skip this entry.
      (iii) Assemble ZK-Statement: $stmt_{KG} \coloneqq \left(\left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}\right)$
      (iv) If $\mathsf{NIZK.Verify}(\mathsf{crs}^{\mathrm{AD}}_{\mathsf{zk}}, stmt_{KG}, \pi_{KG}, \mathcal{R}^{AD}_{KG1}) \neq 1$ skip this entry
      (v) Add ct_i to $\mathsf{L}_{\mathsf{Qual}}$ (the one addressed to the currently executing role)
    (b) Sort $\mathsf{L}_{\mathsf{Qual}}$ lexicographically
    (c) For each of the first $t+1$ entries in $\mathsf{L}_{\mathsf{Qual}}$ do the following:
      (i) Decrypt the ciphertexts for all roles in $\mathcal{H}$ (by assumption of $n > 2t+1$ this results in at least $t+1$ shares) and use the resulting shares to interpolate all $t+1$ polynomials $G_1, \ldots, G_{t+1}$
      (ii) For each role $R_j \in \mathcal{SH} \bigcup \mathcal{B}$ compute $\mathsf{sk}_j \coloneqq \sum_{k=1}^{t+1} G_k(j)$ and $\mathsf{pk}_j \coloneqq \mathsf{TPKE.Sk2Pk}(\mathsf{sk}_j)$
      (iii) For each role $R_j \in \mathcal{H}$ set $\mathsf{pk}_j \coloneqq \mathsf{pk}^{\lambda_{j,0}} \cdot \prod_{k\in\mathcal{SH}\bigcup\mathcal{B}} \mathsf{pk}_k^{\lambda_{j,k}}$, where $\lambda_{i,j}$ are appropriate Lagrange coefficients.
  (8) If key generation already finished, do the following:
    (a) *Get proofs for key resharings from last epoch.*
    For each entry $(pid, t, (R_j, msg, \sigma))$ with $msg \coloneqq (\textsc{KeyShare}, (\mathsf{vk}_i, \{ct_j\}, \pi_{KG}))$ for the **previous** committee in $\textsc{Ordered}$:
      (i) Retrieve vk_{R_j} from $\mathsf{L}_{\mathsf{CommitteeVK}}$
      (ii) Check $\Sigma.\mathsf{Vfy}(\mathsf{vk}_R, msg, \sigma) = 1$, otherwise ignore this message

(iii) Assemble ZK-Statement: $stmt_{KS} \coloneqq \left(\mathsf{vk}_i, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}\right)$

(iv) Check $\mathsf{NIZK.Verify}(\mathsf{crs}^{\text{AD}}_{\text{zk}}, stmt_{KS}, \pi_{KS}, \mathcal{R}^{AD}_{KS}) = 1$, otherwise ignore this message

(v) Add $\left\{ct_j\right\}_{j\in[n]}$ to $\mathsf{L}^{ct}_{\text{Qual}}$

(b) Sort $\mathsf{L}^{ct}_{\text{Qual}}$ lexicographically

(c) *Read reshared key.*

For each entry $(pid, t, (R_j, msg, \sigma))$ with $msg \coloneqq (\textsc{KeyShare}, (\mathsf{vk}_i, \{ct_k\}, \pi_{KS}))$ for the **current** committee in $\textsc{Ordered}$:

(i) Retrieve vk_{R_j} from $\mathsf{L}_{\text{CommitteeVK}}$

(ii) Check if $\Sigma.\mathsf{Vfy}(\mathsf{vk}_{R_j}, (\textsc{KeyShare}, msg, \sigma) = 1$, otherwise skip this entry.

(iii) Retrieve from $\mathsf{L}^{ct}_{\text{Qual}}$ from the first $t+1$ entries the ciphertext ct_j for R_j each, as $\mathsf{L}_{\text{KCom}} \coloneqq \{ct^1_j, \ldots, ct^n_j\}$

(iv) Assemble ZK-Statement: $stmt_{KS} \coloneqq \left(\mathsf{pk}_j, \{ct_k\}_{k\in[n]}, \mathsf{L}_{\text{KCom}}\right)$

(v) Check if $\mathsf{NIZK.Verify}(\mathsf{crs}^{\text{AD}}_{\text{zk}}, stmt_{KS}, \pi_{KS}, \mathcal{R}^{AD}_{KS}) = 1$, otherwise skip this entry.

(vi) Add ct_i (the one addressed to the currently executing role) to L_{Qual}

(d) Sort L_{Qual} lexicographically

(e) For each of the first $t+1$ entries in L_{Qual} do the following:

(i) Decrypt the ciphertexts for all roles in $\mathcal{H}$ (by assumption of $n > 2t+1$ this results in at least $t+1$ shares) and use the resulting shares to interpolate all $t+1$ polynomials $G_1, \ldots, G_{t+1}$

(ii) For each role $R_j \in \mathcal{SH} \bigcup \mathcal{B}$ compute $\mathsf{sk}_j \coloneqq \sum_{k=1}^{t+1} G_k(j)$ and $\mathsf{vk}_j \coloneqq \mathsf{TPKE.Sk2Pk}(\mathsf{sk}_j)$

(iii) For each role $R_j \in \mathcal{H}$ set $\mathsf{vk}_j \coloneqq \mathsf{pk}^{\lambda_{j,0}} \cdot \prod_{k\in\mathcal{SH}\bigcup\mathcal{B}} \mathsf{pk}_k^{\lambda_{j,k}}$, where $\lambda_{i,j}$ are appropriate Lagrange coefficients.

(f) *Read requests for decryption and store them in* $\mathsf{L}_{\text{Requests}}$.

For each entry $(pid, t, (\textsc{Request}, W^{\text{pub}}, W^{\text{enc}}, \{\widehat{ct}_i\}_{i\in[v]}, \pi_{\widetilde{W}}, \mathsf{ek}_{\text{LE}}))$ in $\textsc{Ordered}$:

(i) Assemble ZK-Statement: $stmt_{\widetilde{W}} \coloneqq \left(\mathsf{ek}_{\text{AU}}, \mathsf{vk}_{\text{J}}, W^{\text{enc}}, W^{\text{pub}}, \{\widehat{ct}_i\}_{i\in[v]}\right)$

(ii) If $\mathsf{NIZK.Verify}(\mathsf{crs}^{\text{AD}}_{\text{zk}}, stmt_{\widetilde{W}}, \pi_{\widetilde{W}}, \mathcal{R}^{AD}_{W}) \neq 1$, ignore this message

(iii) For each $i \in [v]$:

(A) Store $(\mathsf{ek}_{\text{LE}}, ct_i, \perp)$ in $\mathsf{L}_{\text{Requests}}$

(g) If LE is corrupted:

(i) Send $(\textsc{Retrieve})$ to $\mathcal{F}_{\text{AD}}$ and receive $(\textsc{Retrieve}, \mathsf{L}^{\text{Pending}}_{\text{Requests}})$

(ii) For each entry $(\mathsf{ek}_{\text{LE}}, ct_i, \perp)$ in $\mathsf{L}_{\text{Requests}}$:

(A) Find $(ct_i, secret_i)$ in $\mathsf{L}^{\text{Pending}}_{\text{Requests}}$

(B) For each role R_j in $\mathcal{SH} \bigcup \mathcal{B}$, compute $\widehat{ct}^j_i \leftarrow \mathsf{TPKE.TDec}(\mathsf{sk}_j, ct_i)$, where sk_j was computed earlier

(C) Run $\left\{\widehat{ct}^k_i\right\}_{k\in\mathcal{H}} \leftarrow \mathsf{TPKE.SimTDec}(\mathsf{pk}, ct_i, secret_i, \left\{\widehat{ct}^j_i\right\}_{i\in(\mathcal{SH}\bigcup\mathcal{B})})$

(D) Update the entry in $\mathsf{L}_{\text{Requests}}$ to $(\mathsf{ek}_{\text{LE}}, ct_i, \left\{\widehat{ct}^k_i\right\}_{k\in\mathcal{H}})$

- **KeyGen1:**

(1) Retrieve the respective $\mathsf{dk}_R, \mathsf{sk}_R$

(2) Generate key share: $s_i \overset{R}{\leftarrow} \mathbb{Z}_p$

(3) Share secret key: choose a random degree t polynomial $F(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_t * x^t$ with $F(0) = s_i$

(4) For each role R_j in the next committee:

```
        (a) Set $sh_j \coloneqq F(j)$
        (b) Generate ciphertext $ct_j \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, sh_j; r_j)$
    (5) Assemble ZK-Witness: $wit_{KG} \coloneqq \left(s_i, \left\{r_j\right\}_{j\in[n]}, F\right)$
    (6) Assemble ZK-Statement: $stmt_{KG} \coloneqq \left(\left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}\right)$
    (7) Compute Proof: $\pi_{KG} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{KG}, wit_{KG}, \mathcal{R}_{KG1}^{AD})$
    (8) Prepare message: $msg \coloneqq (\textsc{KeyGen1}, (\left\{ct_j\right\}_{j\in[n]}, \pi_{KG}))$
    (9) Sign message: $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
   (10) Add $(pid, t_{\mathrm{Now}}, (R_i, msg, \sigma))$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid, (R_i, msg, \sigma))$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
 • **KeyGen2:**
    (1) If $R_i \in \mathcal{SH}$ execute this role honestly
    (2) Otherwise ($R_i \in \mathcal{H}$) do the following:
        (a) Choose random degree-$t$ polynomial $F(x)$
        (b) For each role $R_j$ in the next committee:
            (i) Set $sh_j \coloneqq F(j)$
           (ii) Generate ciphertext $ct_j \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, sh_j; r_j)$
        (c) Retrieve $\mathsf{pk}_i$ computed during preparation
        (d) Assemble ZK-Statement: $stmt_{KG} \coloneqq \left(\mathsf{pk}_i, \left\{ct^k\right\}_{k\in[t+1]}, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}, \mathsf{ek}_R\right)$
        (e) Simulate Proof: $\pi_{KG} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{KG}, \mathsf{td}, \mathcal{R}_{KG2}^{AD})$
        (f) Prepare message: $msg \coloneqq (\textsc{KeyGen2}, (\mathsf{pk}_i, \left\{ct_j\right\}_{j\in[n]}, \pi_{KG}))$
        (g) Sign message: $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
        (h) Add $(pid, t_{\mathrm{Now}}, (R_i, msg, \sigma))$ to $\textsc{Posted}$ and send $(\textsc{Post}, pid, (R_i, msg, \sigma))$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{BCRA}}$
 • **Handover:**
    (1) If $R_i \in \mathcal{SH}$, then execute this role honestly
    (2) Else ($R_i \in \mathcal{H}$), do the following:
        (a) Reshare secret key share:
            (i) Retrieve $\mathsf{vk}_i$ and the first $t+1$ entries of $\mathsf{L}_{\mathsf{Qual}}^{ct}$ as $\left\{ct_i^k\right\}_{k\in[t+1]}$ computed during preparation
           (ii) Choose a random degree $t$ polynomial $F(x) = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_t * x^t$
          (iii) For each role $R_j$ in the next committee:
                (A) Generate ciphertext $ct_j \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{R_j}, F(j); r_j)$
           (iv) Assemble ZK-Statement: $stmt_{KS} \coloneqq \left(\mathsf{ek}_R, \left\{ct_i^k\right\}_{k\in[t+1]}, \left\{ct_j, \mathsf{ek}_{R_j}\right\}_{j\in[n]}, \mathsf{vk}_i\right)$
            (v) Simulate Proof: $\pi_{KS} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{KS}, \mathsf{td}, \mathcal{R}_{KS}^{AD})$
           (vi) Add $msg \coloneqq (\textsc{KeyShare}, (\mathsf{vk}_i, \left\{ct_j\right\}_{j\in[n]}, \pi_{KS}))$ to $\mathsf{L}_\mathsf{M}$
        (b) *Process decryption requests.*
            For each entry $(\mathsf{ek}_{\mathrm{LE}}, ct, \mathsf{L}_{\mathsf{Decryptions}})$ in $\mathsf{L}_{\mathsf{Requests}}$:
            – If LE is honest:
                (i) Partially decrypt $ct$ to $ct^* \leftarrow \mathsf{TPKE.TDec}(\mathsf{sk}_i, ct)$
               (ii) Assemble ZK-Statement: $stmt_{Dec} \coloneqq \left(\mathsf{vk}_i, ct, ct^*\right)$
              (iii) Simulate Proof: $\pi_{Dec} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}_{\mathsf{zk}}^{\mathrm{AD}}, stmt_{Dec}, \mathsf{td}, \mathcal{R}_{Dec}^{AD})$
               (iv) Generate response ciphertext: $msg \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\mathrm{LE}}, (0^{|ct|}, 0^{|ct^*|}, 0^{|\pi_{Dec}|}))$
                (v) Add $(\textsc{Request}, msg)$ to $\mathsf{L}_\mathsf{M}$
            – If LE is corrupted:
                (i) Retrieve $ct^*$ for this role from $\mathsf{L}_{\mathsf{Decryptions}}$
```

(ii) Assemble ZK-Statement: $stmt_{Dec} := (\mathsf{vk}_i, ct, ct^*)$
(iii) Simulate proof: $\pi_{Dec} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}_{\text{zk}}^{\text{AD}}, stmt_{Dec}, \mathsf{td}, \mathcal{R}_{Dec}^{AD})$
(iv) Encrypt answer: $msg \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\text{LE}}, (ct, ct^*, \pi_{Dec}))$
(v) Add (Request, msg) to L_M

(c) *Sign outgoing messages.*
For each entry (msg) in L_M:
(i) Generate signature $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_R, msg)$
(ii) Update $msg := (R_i, msg, \sigma)$

(d) Remove the entry for this role from $\mathsf{L}_{\mathsf{Roles}}$
(e) Delete all state except for L_M
(f) *Send messages.*
For each entry (msg) in L_M:
(i) Send (Post, msg) to $\mathcal{F}_{\text{BCRA}}$

RoleExecute (corrupted N):

- This is handled by simulation of $\mathcal{F}_{\text{BCRA}}$

To prove Theorem 4.4, we proceed in a series of games, where $\mathcal{F}_{\text{AD}}^{i+1}$ behaves the same as $\mathcal{F}_{\text{AD}}^{i}$ except for the stated changes and $\mathcal{S}^{i+1}$ behaves the same as $\mathcal{S}^i$ except for the stated changes.

Game 0. Game 0 is equivalent to the real experiment. That is,

$$H_0 := \mathsf{Exp}^{\Pi_{\text{AD}}}_{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{BCRA}}, \mathcal{G}_{\text{CLOCK}}, \mathcal{S}^0, \mathcal{Z}}$$

with $\mathcal{S}^0$ being the dummy adversary. This means that all parties execute the real protocol.

Game 1. In this game, the simulator $\mathcal{S}^1$ takes control over $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{BCRA}}$, but executes them honestly. Additionally, $\mathcal{F}_{\text{AD}}^1$ is introduced, and all honest parties are replaced by dummy parties that forward their inputs to $\mathcal{F}_{\text{AD}}^1$ and when receiving output from $\mathcal{F}_{\text{AD}}^1$ forward it to the environment. $\mathcal{F}_{\text{AD}}^1$, upon receiving any message from a dummy party forwards it to the simulator $\mathcal{S}^1$ and asks it for output. $\mathcal{S}^1$ executes the real protocol for all honest parties (using the inputs received from $\mathcal{F}_{\text{AD}}^1$) and instructs $\mathcal{F}_{\text{AD}}^1$ to deliver the resulting outputs.

Proof Sketch: Game 0 ≈ Game 1.
$\mathcal{S}^1$ executes the same code as the real parties on the same inputs (that have been forwarded by $\mathcal{F}_{\text{AD}}^1$). Thus, Game 0 and Game 1 are identical from the environments view.

Game 2. In this game, $\mathcal{S}^2$ generates the common reference string for the zero-knowledge proof system with an extraction and simulation trapdoor td as $(\mathsf{crs}_{\text{zk}}^{\text{AD}}, \mathsf{td}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

Proof Sketch: Game 1 ≈ Game 2.
Indistinguishability follows from the simulation extractability of NIZK.

Game 3. In this game, $\mathcal{F}_{\text{AD}}^3$ now handles (Init, ...) messages the same way $\mathcal{F}_{\text{AD}}$ does, except that during SO init it also sends the generated secret key sk to $\mathcal{S}^3$, and $\mathcal{S}^3$ handles this task the same way $\mathcal{S}$ does, except that it also stores the received secret key sk. Note that $\mathcal{S}^3$ still chooses the outputs for (GetPK) and can thus still return the public encryption key pk′ generated during the protocol.

Proof Sketch: Game 2 ≈ Game 3.
If SO is honest, $\mathcal{S}^3$ receives the message (Init, SO, vk_{SO}, pk) from $\mathcal{F}_{\text{AD}}^3$ and can thus perfectly simulate

the real protocol for this task. If SO is corrupted, $\mathcal{S}^3$ can wait for a message (Post, OperatorKey, $\mathsf{vk}_{\mathrm{SO}}$) from SO to $\mathcal{F}_{\mathrm{BCRA}}$ and then send (Init, SO, $\mathsf{vk}_{\mathrm{SO}}$) to $\mathcal{F}^3_{\mathrm{AD}}$ on behalf of SO. Since both J and AU are assumed to be honest, $\mathcal{S}^3$ receives the respective messages from $\mathcal{F}^3_{\mathrm{AD}}$ and can perfectly simulate the real protocol for the respective task.

Game 4. In this game, $\mathcal{F}^4_{\mathrm{AD}}$ now handles (GetSKeys) the same way $\mathcal{F}_{\mathrm{AD}}$ does, and $\mathcal{S}^4$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 3 ≈ Game 4.
For corrupted parties, this is perfectly simulated by executing $\mathcal{F}_{\mathrm{BCRA}}$. For honest parties, $\mathcal{F}^4_{\mathrm{AD}}$ outputs the keys if they are available via $\mathcal{F}_{\mathrm{BCRA}}$. Thus, these games are statistically indistinguishable.

Game 5. In this game, after simulation of $\mathcal{F}_{\mathrm{BCRA}}$ resulted in all secret keys for the next committee being delivered, $\mathcal{S}^5$ fills the three sets $\mathcal{H}, \mathcal{B}, \mathcal{SH}$ as follows:

- All roles that are already corrupted are assigned to $\mathcal{B}$
- $t - |\mathcal{B}|$ random roles that are not corrupted are assigned to $\mathcal{SH}$
- The remaining roles are assigned to $\mathcal{H}$

It holds that $\mathcal{SH} \bigcap \mathcal{B} = \emptyset$ and $\mathcal{H} \bigcap (\mathcal{SH} \bigcup \mathcal{B}) = \emptyset$ and $\mathcal{H} \bigcup \mathcal{SH} \bigcup \mathcal{B}$ contains all roles of that committee.

Proof Sketch: Game 4 ≈ Game 5.
This change is only syntactical.

Game 6. In this game, whenever the adversary corrupts a party that is assigned a role in $\mathcal{H}$ and has not executed yet, the simulator picks a random role $R_i \in \mathcal{SH}$ instead, sets $\mathcal{SH} \coloneqq \mathcal{SH} \setminus \{R_i\}, \mathcal{B} \coloneqq \mathcal{B} \bigcup \{R_i\}$ and returns the state of role R_i instead (which consists of that roles secret keys). When the adversary corrupts a party that is assigned a role R in $\mathcal{SH}$, the simulator sets $\mathcal{SH} \coloneqq \mathcal{SH} \setminus \{R\}, \mathcal{B} \coloneqq \mathcal{B} \bigcup \{R\}$ and returns the state of that role.

Proof Sketch: Game 5 ≈ Game 6.
Since all roles in a committee have the same description, and the assignment of a role to a node only becomes visible after it executed, these games are indistinguishable.

Game 7. In this game, the simulator uses the shares of uncorrupted roles to reconstruct the secret key sk' associated with the public encryption key pk' generated during the key generation protocol. Additionally, $\mathcal{S}^7$ performs the steps listed under **PrepareExecution** in $\mathcal{S}$, except that in step (8g-i) instead of querying $\mathcal{F}_{\mathrm{AD}}$ for decrypted secrets it uses sk' to decrypt ciphertexts ct_i to $secret_i$.

Proof Sketch: Game 6 ≈ Game 7.
This change only affects the internal view of the simulator.

Game 8. This is the same as the previous game, except that during **KeyGen2** and **Handover** the simulation trapdoor td is used to generate the zero-knowledge proofs.

Proof Sketch: Game 7 ≈ Game 8.
Indistinguishability follows from the zero-knowledge property of NIZK.

Game 9. In this game, if LE is corrupted, then when processing requests for decryption during Handover while executing roles in $\mathcal{H}$, $\mathcal{S}^9$ uses $\widehat{ct}$ computed during **PrepareExecution** instead of executing TPKE.TDec. (This corresponds to step (2b), LE corrupted in $\mathcal{S}$)

Proof Sketch: Game 8 ≈ Game 9.
If LE is honest, these two games are the same. Otherwise, $\widehat{ct}$ has been computed to ensure that TPKE.Combine outputs the correct plaintext. Since honest roles delete all internal state before sending their message, these two games are indistinguishable.

Game 10. In this game, if LE is honest, then when processing requests for decryption during Handover while executing roles in $\mathcal{H}$, $\mathcal{S}^{10}$ encrypts (the appropriate amount of) zeroes instead. (This corresponds to step (2b), LE honest in $\mathcal{S}$)

Proof Sketch: Game 9 ≈ Game 10.
If LE is corrupted, these two games are the same. Otherwise, indistinguishability follows from IND-CPA security of PKE.

Game 11. In this game, $\mathcal{S}^{11}$ simulates the distributed key generation protocol to output pk (received from $\mathcal{F}_{\text{AD}}^{11}$ during SO Init) as resulting public encryption key. This is achieved by following the steps listed under **KeyGen2** and **Handover** in $\mathcal{S}$. Additionally, sk (received from $\mathcal{F}_{\text{AD}}^{11}$ during SO Init) is used during **PrepareExecution** to decrypt ciphertexts. $\mathcal{F}_{\text{AD}}^{11}$ now handles (ExecuteRole) messages the same way $\mathcal{F}_{\text{AD}}$ does.

Proof Sketch: Game 10 ≈ Game 11.
Since by assumption at most t roles in each committee are corrupted, the shares received by corrupted roles are statistically close to those in Game 10. Since honest parties delete all internal state before sending their message, corrupting them after they executed does not reveal anything. Thus, only ciphertexts addressed to uncorrupted roles differentiate these two games, and RIND-SO security of PKE ensures Indistinguishability.

Game 12. In this game, $\mathcal{F}_{\text{AD}}^{12}$ now handles (GetPK) the same way $\mathcal{F}_{\text{AD}}$ does, and $\mathcal{S}^{12}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 11 ≈ Game 12.
Since the previous games already ensured that the same public key pk is used in both $\mathcal{F}_{\text{AD}}^{12}$ and the protocol simulation, these two games are perfectly indistinguishable.

Game 13. In this game, when receiving (Request, $\widetilde{W}, \sigma_{\widetilde{W}}$) messages, $\mathcal{F}_{\text{AD}}^{13}$ checks if the signature is valid. If the signature is valid, it forwards (Request, $\widetilde{W}$) to $\mathcal{S}^{13}$ (but not the signature $\sigma_{\widetilde{W}}$!), otherwise it ignores the message. $\mathcal{S}^{13}$ follows the protocol honestly except for the NIZK proof, where it uses the simulation trapdoor to generate $\pi_{\widetilde{W}} \leftarrow \mathsf{NIZK.Sim}(\text{crs}_{\text{zk}}^{\text{AD}}, stmt_{\widetilde{W}}, \text{td}, \mathcal{R}_{W}^{AD})$ without knowing the witness.

Proof Sketch: Game 12 ≈ Game 13.
If LE is corrupted, these two games are identical. If LE is honest, then indistinguishability follows from zero-knowledge of NIZK.

Game 14. In this game, when receiving (Post, (Request, W^{pub}, W^{enc}, $\{ct_i\}_{i\in[v]}, \pi_{\widetilde{W}}, \text{ek}_{\text{LE}}$)) from LE to $\mathcal{F}_{\text{BCRA}}$, if the zero-knowledge proof verifies, $\mathcal{S}^{14}$ uses the trapdoor td to extract a witness and aborts the simulation of this fails.

Proof Sketch: Game 13 ≈ Game 14.
This game only differs from Game 13 if the abort happens. Due to the simulation extractability of NIZK, this only happens with negligible probability, and thus the games are indistinguishable.

Game 15. In this game, when receiving (Post, (Request, W^{pub}, W^{enc}, $\{ct_i\}_{i\in[v]}$, $\pi_{\widetilde{W}}$, ek_{LE})) from LE to $\mathcal{F}_{\text{BCRA}}$, $\mathcal{S}^{15}$ sends (Request, $\widetilde{W}$, $\sigma_{\widetilde{W}}$) to $\mathcal{F}_{\text{AD}}^{15}$ in the name of LE. $\mathcal{F}_{\text{AD}}^{15}$ now fills the lists $\mathsf{L}_{\mathsf{Warrants}}$, $\mathsf{L}_{\mathsf{Requests}}$ the same way $\mathcal{F}_{\text{AD}}$ does, and as a consequence also fills the lists $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Pending}}$ and $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Ready}}$ during Handover.

Proof Sketch: Game 14 ≈ Game 15.
This only affects the internal view of $\mathcal{F}_{\text{AD}}^{15}$, but no output depends on that view (yet). Thus, these games are perfectly indistinguishable.

Game 16. In this game, $\mathcal{F}_{\text{AD}}^{16}$ handles (Retrieve) messages the same way as $\mathcal{F}_{\text{AD}}$, and $\mathcal{S}^{16}$ handles this task the same way as $\mathcal{S}$.

Proof Sketch: Game 15 ≈ Game 16.
If LE is corrupted, this changes nothing. If LE is honest, the list $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Ready}}$ has been correctly filled in Game 15. Since by assumption, at least $t + 1$ roles in each committee are not corrupted, every ciphertext that $\mathcal{F}_{\text{AD}}$ decrypts also results in at least $t + 1$ decryption shares in the real protocol. As such, the same set of decryptions is output in both games and indistinguishability follows.

Game 17. This game is the same as Game 16 except that $\mathcal{F}_{\text{AD}}^{17}$ now stores $\widetilde{W}$ in $\mathsf{L}_{\mathsf{Warrants}}$ when $\mathcal{S}^{17}$ instructs it to deliver output while handling (Request) messages.

Proof Sketch: Game 16 ≈ Game 17.
This change is only syntactical.

Game 18. In this game, $\mathcal{F}_{\text{AD}}^{18}$ additionally handles (AuditRequest) messages the same way $\mathcal{F}_{\text{AD}}$ does, and $\mathcal{S}^{18}$ handles this task the same way $\mathcal{S}$ does. Note that since the auditor AU is assumed to be honest, simulation only requires a simulated (Read) to $\mathcal{F}_{\text{BCRA}}$.

Proof Sketch: Game 17 ≈ Game 18.
If LE is honest, these games are identical, since $\mathcal{S}^{17}$ generated entries in Ordered with valid proofs for exactly the warrants that $\mathcal{F}_{\text{AD}}^{18}$ has in $\mathsf{L}_{\mathsf{Warrants}}$, and simulation of AU is perfect. If LE is corrupted, then these games are also identical: In Game 17 AU outputs exactly those warrants stored in Ordered for which the zero-knowledge proof is valid. For exactly those warrants, $\mathcal{S}^{18}$ sends (Request, . . .) messages to $\mathcal{F}_{\text{AD}}^{18}$, resulting in them being in $\mathsf{L}_{\mathsf{Warrants}}$. Thus, AU outputs the same set of warrants in Game 18 as in Game 17.

Game 19. This game is the same as Game 18 except that $\mathcal{S}^{19}$ computes W^{enc} as $W^{\text{enc}} \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}_{\text{AU}}, 0^{|\widetilde{W}|})$ instead of encrypting $\widetilde{W}$.

Proof Sketch: Game 18 ≈ Game 19.
If LE is corrupted, these two games are again identical. If LE is honest, indistinguishability follows directly from the IND-CPA security of PKE.

Game 20. In this game, $\mathcal{F}_{\text{AD}}^{20}$ no longer sends the secret decryption key sk to $\mathcal{S}^{20}$ during Init. Instead of using sk to decrypt ciphertexts during **PrepareExecute** handling, $\mathcal{S}^{20}$ now queries $\mathcal{F}_{\text{AD}}^{20}$ for the decryption as described in $\mathcal{S}$.

Proof Sketch: Game 19 ≈ Game 20.
The only place sk was used was to decrypt ciphertexts in step (8g-i) of **PrepareExecute** (instead of querying $\mathcal{F}_{\text{AD}}$). Thus, if LE is honest, these games are the same. If LE is corrupted, these two games only differ if there is an entry (ek_{LE}, ct_i, $\perp$) in $\mathsf{L}_{\mathsf{Requests}}$ for which no matching entry (ct_i, $secret_i$) exists in $\mathsf{L}_{\mathsf{Requests}}^{\mathsf{Pending}}$ returned by $\mathcal{F}_{\text{AD}}^{20}$. This can not be the case: Only ciphertexts ct_i corresponding to valid zero-knowledge proofs are inserted into $\mathsf{L}_{\mathsf{Requests}}$. But for each valid zero-knowledge proof, the

extracted warrant (and thus all listed ciphertexts) was send as input to $\mathcal{F}_{\mathrm{AD}}^{20}$, and thus a valid decryption is returned. Therefore, these games are indistinguishable.

Game 21. In this game, $\mathcal{F}_{\mathrm{AD}}^{21}$ handles (Request) messages the same way as $\mathcal{F}_{\mathrm{AD}}$, and $\mathcal{S}^{21}$ handles this task the same way as $\mathcal{S}$.

Proof Sketch: Game 20 ≈ Game 21.
For corrupted LE, this changes nothing. For honest LE, $\mathcal{S}^{21}$ no longer receives $\widetilde{W}$ but only $f_t(W)$, $\left|\widetilde{W}\right|$, v, i.e., the output of the transparency function, the length of the warrant and the number of listed ciphertexts. $\widetilde{W}$ was used to compute the transparency function and obtain the listed ciphertexts before, since encryption of the warrant was already replaced by an encryption of zeroes in Game 19. Since the output of the transparency function was passed by $\mathcal{F}_{\mathrm{AD}}^{21}$, the only missing thing are the listed ciphertexts, which the simulator replaces by ciphertexts of zeroes. By assumption, at most t roles in each committee are corrupted, and up to t decryption shares obtained by the environment in this way reveal no information about the plaintexts. Note that the decryption of these ciphertexts has not been used by $\mathcal{S}^{21}$ since Game 10, except for parties in $\mathcal{SH}$, for which it is ensured that there are at most t parties in $\mathcal{SH} \bigcup \mathcal{B}$. Thus, indistinguishability follows from IND-CPA security of TPKE.

Game 22. In this game, $\mathcal{F}_{\mathrm{AD}}^{22}$ additionally handles (GetStatistics) messages the same way $\mathcal{F}_{\mathrm{AD}}$ does, and $\mathcal{S}^{22}$ handles this task the same way $\mathcal{S}$ does.

Proof Sketch: Game 21 ≈ Game 22.
These games are identical for the same reason as above, the set of warrants in Ordered with valid proofs is the same as the set of warrants in $\mathcal{F}_{\mathrm{AD}}^{18}$

Note that Game 22 equals the ideal world. Since we showed that Game i is indistinguishable from Game $i+1$ for $i \in \{0, \dots, 21\}$, it follows that Game 0 (the real protocol) is indistinguishable from Game 22 (ideal execution) and thus Theorem 4.4 follows. □

List of Figures

List of Tables

List of Definitions