P4TC—Provably-Secure yet Practical Privacy-Preserving Toll Collection

MAX HOFFMANN[*], Ruhr-Universität Bochum, Germany
VALERIE FETZER[†], MATTHIAS NAGEL[‡], ANDY RUPP[§], and REBECCA SCHWERDT[¶], Karlsruhe Institute of Technology, Germany

Electronic toll collection (ETC) is widely used all over the world not only to finance our road infrastructures, but also to realize advanced features like congestion management and pollution reduction by means of dynamic pricing. Unfortunately, existing systems rely on user identification and allow tracing a user's movements. Several abuses of this personalized location data have already become public. In view of the planned European-wide interoperable tolling system EETS and the new EU General Data Protection Regulation, location privacy becomes of particular importance.

In this paper, we propose a flexible security model and crypto protocol framework designed for privacy-preserving toll collection in the most dominant setting, i.e., Dedicated Short Range Communication (DSRC) ETC. As opposed to our work, most related cryptographic proposals target a less popular type of toll collection based on Global Navigation Satellite Systems (GNSS), or do not come with a thorough security model and proof. A major challenge in designing the framework at hand was to combine provable security and practicality, where the latter includes practical performance figures and a suitable treatment of real-world issues, like broken on-board units etc. To the best of our knowledge, our work is the first in the DSRC setting with a rigorous security model and proof and arguably the most comprehensive formal treatment of ETC security and privacy overall.

For our ETC system, we make use of and significantly extend a payment protocol building block, called BBA+, introduced at ACM CCS 2017. Additionally, we provide a prototypical implementation on realistic hardware. This implementation already features fairly practical performance figures, even though there is still room for optimizations. An interaction between an on-board unit and a road-side unit is estimated to take less than a second allowing for toll collection at full speed assuming one road-side unit per lane.

Contents

[*] The author is supported by DFG grant PA 587/10-1.

[†] The author is supported by DFG grant RU 1664/3-1.

[‡] The author is supported by the German Federal Ministry of Education and Research within the framework of the project "Sicherheit vernetzter Infrastrukturen (SVI)" in the Competence Center for Applied Security Technology (KASTEL).

[§] The author is supported by DFG grant RU 1664/3-1 and the Competence Center for Applied Security Technology (KASTEL).

[¶] The author is supported by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data-Informatics Methods for its Collection, Analysis and Exploitation.

1 INTRODUCTION

Electronic toll collection (ETC) is already deployed in many countries all over the world. A recent study [57] conducted by "Markets and Markets" predicts a 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) until 2027 [29]. Hence, ETC has been and will be an important technology deserving a careful analysis of security and privacy issues as well as custom-tailored secure solutions.

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 do not protect the location privacy of their users. This encourages abuse: EZ-Pass records, for instance, have been used as evidence in divorce lawsuits [60], EZ-pass transponders are abused to track drivers throughout New York City [65], and the Norwegian AutoPASS system allowed anyone to obtain a transcript of which toll booths a car visited [31].

However, the only legitimate reason to store personalized location records in these systems is to bill customers. As opposed to other systems, as for instance loyalty systems, this data should not be used for other business relevant purposes like, e.g., personalized advertising. 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 ETC providers as well. 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) [37] which is in effect since May 2018. The GDPR stipulates comprehensive protection measures and heavy fines in case of non-compliance.

1.1 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 [28, 61].

Concerning what the user is charged for, we distinguish between two major types of charging schemes:

Distance-based: The 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 also be dynamically adapted by other parameters. This charging scheme is typically used in urban areas—not only to finance the road infrastructure but also for congestion management and pollution reduction by adapting tolls dynamically.

There are two main 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): Today, this is the most widely used ETC technology worldwide and the de facto standard in Europe [61]. It is based on bidirectional radio communication between a road-side unit (RSU) and a mobile device aka on-board unit (OBU) installed in the vehicle. In todays 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): ANPR, aka video tolling, inherently violates privacy.

Global Navigation Satellite System (GNSS): In a GNNS-based system the OBU keeps track of the vehicle's location (e.g., by means of GPS) 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 GSM for communicating with the toll service provider.

1.2 Drawbacks of Existing Systems and Proposals

Independent of the technology used, systems deployed in practice build on identifying the user to charge him. Previous work on privacy-preserving toll collection mainly considers the GNSS setting and comes—apart from a few exceptions [6, 27, 30]—without any formal security analysis. Although DSRC is the most dominant setting in practice, 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. We elaborate on related work in Section 1.4.

1.3 P4TC

We propose a comprehensive security model as well as provably-secure and efficient protocols for privacy-friendly ETC in the DSRC setting. Our definitional framework and the system are very flexible. We cover access-based and distant-based charging schemes as well as combinations of those. Our protocols work for toll plaza environments, but are (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 treament of ETC security and privacy overall.

Section 1.5 provides an overview on the toll collection scenario and the desired properties we consider. Section 1.6 gives details on our contribution.

1.4 Related Work

In this section, we review proposals for privacy-preserving ETC. So far, more elaborate systems have been proposed for the GNSS setting in comparison to the actually more widely used DSRC setting.

1.4.1 DSRC (OBU/RSU) Setting. Previous work [32, 47–49] in this setting mainly focuses on a pre-pay scenario where some form of e-cash is used to spend coins when passing an RSU. This mode is not preferred by users due to its inconveniences. A user needs to ensure that he always has enough e-coins to pay the tolls. This is particularly inconvenient when prices change dynamically. In addition, using standard offline e-cash (e.g., [17]), the user may not overpay since he cannot get change from the RSU in a privacy-preserving way. Also transferrable e-cash [7] does not solve the problem of change as the impossibility result of [19] applies to the ETC setting. So a user would not only need enough money but the right denomination of e-coins.

In [47–49] multiple electronic road pricing systems specifically tailored for Low Emission Zones (LEZ) are proposed. In [47] a user drives by an RSU and directly pays some price depending on this RSU. For [48, 49] the price a user has to pay depends on the time spent inside the LEZ. To this end, the user receives some e-ticket from an Entry-RSU when entering the LEZ which he needs 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.

In [32], a simple access-based toll collection system based on RSA blind signatures is sketched. Users buy a set of tokens/coins during registration which are used to pay toll while driving. Double-spending detection can be detected with high probability and some ideas are presented to keep the double-spending database small.

In [10], the authors propose an interesting 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 the user is reimbursed for the difference. The system supports revocation of a user's anonymity and token revocation. The instantiation mixes cryptographic primitives from the Paillier and DLog setting. Zero-knowledge proofs for languages which mix statements (and share variables) of the two settings are also required. The actual proofs are not specified in the paper. Instead they are just claimed to be "quite standard", what we doubt. As opposed to ours, their system relies on online "over-spending" detection and does not come with any formal model or security proof.

GNSS Setting. A variety of GNSS-based system proposals can be found in the literature, including a series of papers [6, 58, 62] published at USENIX Security. Here, the OBU equipped with GPS and GSM typically collects location-time data or road segment prices and sends this data to the toll service provider (TSP). To ensure that the user behaves honestly and, e.g., does not omit or forge data, unpredictable spot checks are assumed which force a user to reveal the data he sent at a certain location and time. In a reconciliation phase, the user calculates his toll based on the data sent and proves that his calculation is correct.

One example for the GNSS approach is the simple and flexible electronic road pricing system in [33]. There the server collects hash values of the trip records from the OBUs in form of a hash tree. In the reconciliation phase the consistency of the hash tree is verified. However, the whole system is only described imprecisely and the security and privacy of the system are not proven.

In VPriv [62], 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 needs to download the database of all tagged location-time tuples, calculate his total fee, and prove that for this purpose, he correctly used the tuples belonging to his tags without leaking those. In [30] ProVerif is used to show that VPriv is privacy-preserving for honest-but-curious adversaries.

In [44] a road pricing system, where the OBU anonymously sends location data to a central server, is presented. The system is based on splitting a single trip into several unlinkable segments (called "legs") and on distributing the calculation of the fee for a trip between several distinct entities. During the reconciliation phase, every OBU learns the locations of the spot check devices that recorded its corresponding vehicle. The security and privacy requirements are only proved informally.

In the PrETP [6] scheme, the OBU non-anonymously sends payment tuples consisting of commitments to location, time, and the corresponding price that the OBU determined itself. During reconciliation with the TSP, the user presents his total toll and proves that this is indeed the sum of the individual prices he sent, using the homomorphic property of the commitment scheme.

The authors prove their system secure using the ideal/real world paradigm.

In [58], 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. No security or privacy proofs are given.

In [26], 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 his 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. During dispute solving, the group manager recalculates each user's toll and, thus, catches the cheater. Choosing an appropriate group size is difficult (the larger the size, the better the anonymity is protected, 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 [27], the system's security and privacy properties are verified using ProVerif.

Another cell-based road pricing scheme is presented in [38]. Here, a roadpricing area is divided into cells, where certain cells are 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 also 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 main issue with all systems described above is that their security relies on a strong assumption in practice: invisible spot checks at locations which are unpredictable 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 [51], 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 a user not benefit from turning off his OBU 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 in terms of infrastructure requirements compared to the DSRC setting.

1.5 Considered Scenario

Here, we sketch the considered scenario, giving an idea of the required flexibility and complexity of our framework. We target a post-payment toll collection system in the DSRC setting which allows access-based as well as distance-based charging and can be deployed in a toll-plaza as well as an open-road environment. It involves the following entities:

- The *Toll Collection Service Provider (TSP)* which might be a privately owned company.
- The *State Authority (SA)*, e.g., the Department of Transportation, which outsourced the toll collection operations to the TSP but is responsible for violation enforcement.
- A *Dispute Resolver (DR)*, e.g., an NGO protecting privacy, which is involved in case of an incident or dispute.[1]
- *Users* who participate in the system by means of a (portable or mounted) On-Board Unit (OBU). The OBU is used for on-road transactions and in the scope of 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's data.
- *Road-Side Units (RSUs)* which 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

[1] Note that we assume the DR to be trusted by all other parties. It implements an *optional* "kind of key escrow" mechanism.

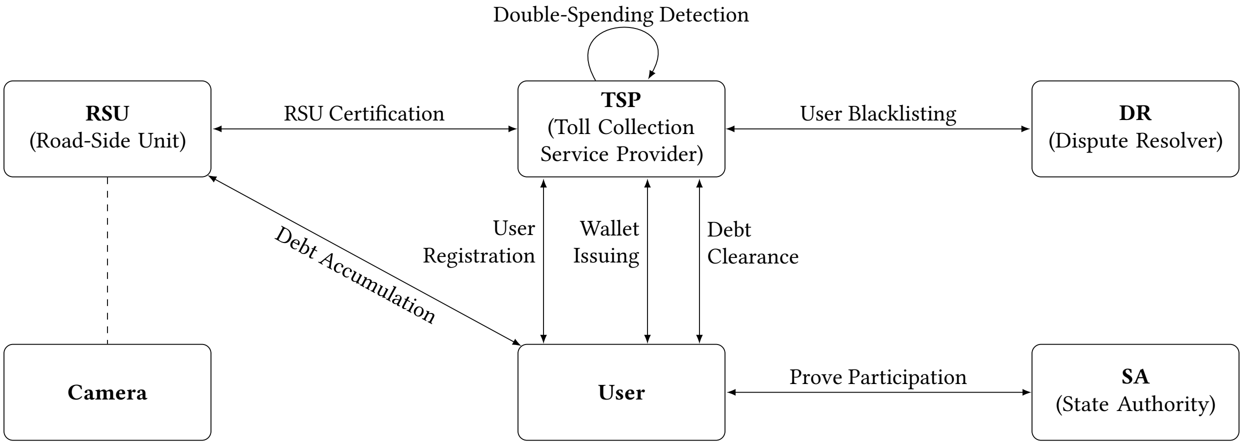


Fig. 1. The P4TC System Model

only assume that they are periodically online for a short duration (presumably at night when there is not much traffic) to exchange data for fraud detection with the TSP.

- *Enforcement Cameras* triggered by the RSUs which are typically owned by the SA 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 which are controlled by the RSU.

1.5.1 Main Protocols. In the following, we sketch the main protocols of the system involving the parties from above. Fig. 1 provides an overview of these interactions. A more detailed description that also includes the remaining protocols can be found in Section 4. For simplicity, let us envision a system with (e.g., monthly) billing periods, although this is not mandatory but suggested for our framework.

User Registration: To participate in the system, a user needs to register with the TSP using some physical ID (e.g., passport number, SSN) which is verified out-of-band. This is done once and makes the user accountable in case he cheats or refuses to pay his bill.

Wallet Issuing: A wallet is issued to a user by the TSP which is bound to the user's ID and a set of user attributes (discussed later on). The wallet is used to accumulate debt.

Debt Accumulation: Every time a user passes an RSU, his OBU and the RSU execute the Debt Accumulation protocol. First, the toll the user owes is determined and then this toll is added to the user's wallet—possibly along with some public attributes of the RSU. The toll may be dynamic and depend on different factors like the current time and congestion, the number of axles (recognized by sensors attached to the RSU), some user attributes attached to the wallet as well as some attributes of the previous RSU the user drove by. The camera takes a photo of the license plate(s) in case the RSU reports any protocol failure, aborts (e.g., if a wallet is outdated) or a car in the range of the RSU refuses to communicate at all. Due to technical limitations, it might not always be possible to determine exactly which car triggered the camera, especially in open-road tolling environments [45]. In these cases, photos of more than one car being 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 protocol 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, a user is requested to participate in a (asynchronous) clearance phase with the TSP. As the user is not anonymous in this protocol, he can be penalized if he refuses to run the protocol within a certain time frame. In a protocol execution, he presents his wallet and the accumulated debt to the TSP. Then he may clear his debt immediately or within a certain grace period. After a successful protocol execution, his wallet is invalid. He can get a new one by running Wallet Issuing again.

1.5.2 Attributes, Pricing Function and Privacy Leakage. Our scenario involves two types of attribute vectors: user attributes as well as (previous) RSU attributes. To keep our framework flexible, we do not stipulate which kind of attributes or how many of them are used. Those details depend on the concrete pricing model with a pricing function depending on the user attributes, the current and previous RSU attributes and auxiliary, publicly verifiable input. However, we expect that for most scenarios very little information needs to be encoded into these attributes. For instance, one could realize access-based toll collection with a pricing function primarily depending on auxiliary input like location, time and congestion without any previous RSU attributes and only encoding the current billing period as a user attribute. Including the previous RSU's attributes into the toll calculation also allows to cover 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.[2] 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 an RSU attribute. Likewise, to enforce that a user eventually pays his debt, the user attributes should at least encode the billing period they are valid for.[3]

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. In Appendix A, we analyze the privacy leakage of our system in more detail.

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, if a (corrupted) RSU tries to break privacy by charging a peculiar price that differs from the prescribed pricing function, the user is assumed to file a claim.

1.5.3 Desired Properties. The following list summarizes the desired security properties we had in mind in an informal manner. In Section 2 we then propose an ideal world model formally covering what we achieve.

(P1) A user may only use a wallet legitimately issued to him.

(P2) In Debt Accumulation, a user may not pretend that he owes less by forging the attributes attached to his wallet.

(P3) In Debt Clearance, a user should not be able to claim that he owes less than the amount added to his wallet, provided he never used an old copy of his wallet (double-spending) or aborted Debt Accumulation.

[2] In this way, the entry point of a user's trip can be linked to his exit point. However, our system ensures that the user is still anonymous and multiple entry/exit pairs are unlinkable.

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

(P4) If a user reuses an old copy of his wallet (double-spending) he will be identified.
(P5) If a user fails to participate in or prematurely aborts Debt Accumulation he will be identified.
(P6) A Debt Accumulation transaction of an honest user does not leak anything to an RSU/TSP beyond the user's attributes and the attributes of the previous RSU.
(P7) Prove Participation only enables the SA to deanonymize a single transaction of an honest user in case of an incident involving this user. His remaining transactions stay unlinkable.
(P8) The TSP is—with a hint from the DR—able 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.
(P9) The TSP is—with a hint from the DR—able 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 he visited and the amounts being charged.
(P10) As the secrets of an RSU might enable a user to tamper with his debt, there is a mechanism to mitigate the financial loss due to a stolen or compromised RSU.

1.6 Our Contribution

Our contribution is threefold:

1.6.1 Protocols. While the overall idea underlying our construction might appear intuitive and we partly draw from known techniques, a major challenge was to twist and combine all these techniques to achieve simulation-based security and practicality *at the same time.*

To this end, we start from a payment system building-block called black-box accumulation (BBA+), recently introduced in [42]. BBA+ 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. Besides unlinkability, 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 (P6) partially) of the desired properties identified in Section 1.5.3, significant modifications and enhancements had to be made to fully suit our needs.

For instance, the basic BBA+ mechanism does not allow for efficient blacklisting of individual wallets (P8) nor to recalculate individual balances (P9). We solve this by having an individual trapdoor for each wallet (accessible to the DR in case of an incident) which makes transactions involving this wallet forward and backward linkable. The trapdoor does not allow to link wallets of other users.[4] To realize this, we adopt and adjust an idea from the e-cash literature [18]. More precisely, we make use of a PRF applied to a counter value (which is bound to a wallet) 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 is forced to deposit the key encrypted under the DR's public key. The correctness of this deposit is ensured to the TSP by means of 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 $\mathbb{Z}_p$. By letting a user choose a new PRF key for each billing period, his transactions from previous/upcoming periods stay unlinkable.

As a minor but very useful modification, we added (user and RSU) attributes to a wallet, which, of course, get signed along with the wallet to protect from forgery. This allows us to bind wallets to a billing period encoded as attribute. By making RSUs only accept wallets from the current period, the size of the blacklist checked by the

[4] To be precise, in the BBA+ system, there is a TTP-owned trapdoor that allows to link a user's transactions. However, this is not a user- or wallet-specific trapdoor, which means, that if handed to the TSP, it could track each and every user in the system.

RSU can be limited to enable fast 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, he could create new wallets for fake users, forge user attributes, balances, etc. In order to mitigate this problem (P10), we take the following measures: 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. 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. By including an expiration date into the RSU attributes, one can limit the potential damage involved with the latter issue. In view of the fact that RSUs are usually not easily accessible physically and key material is usually protected by a HSM, we believe that these measures are sufficient. We intentionally refrain from using key revocation mechanisms like cryptographic accumulators [55] in order to retain real-time interactions.

Finally, we added mechanisms to prove the participation in Debt Accumulation interactions (P7) and to enable a simulation-based security proof.

1.6.2 System Definition, Security Model and Proof. Having the scenario from Section 1.5 at the back of our mind, we propose a system definition and security model for post-payment toll collection systems based on the real/ideal-paradigm. More precisely, we build upon the GUC framework [22].

Our work is one of very few that combines a complex, yet practical crypto system with a thorough UC security analysis. Typically, the standard approach is to cast a complex system as an MPC problem and then resort to a *generic but inefficient* UC-secure MPC protocol [23, 46].

The security of BBA+ has been modeled by formalizing each security property from a list of individual properties as it is usually done in the game-based setting. This approach bears the intrinsic risk that important and expedient security aspects are overlooked, e.g., the list is incomplete. This danger is eliminated by our UC-approach where we do not aim to formalize a list of individual properties but rather how an ideal system should look like.

A challenging task was to find a formalization of such an ideal system (aka ideal functionality) such that a reasonable trade-off between various aspects is accomplished: On the one hand, it needs to be sufficiently abstract to represent the semantics of the eventual goal “toll collection” while it should still admit 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 the security analysis 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 argue about the security of the whole system using the UC composition theorem. We refrain from this approach, however, as it entails a slew of technical subtleties due to the shared state between the individual two-party 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. Although parts of it are inspired by proofs from the literature, it is very complex and technically demanding in its entirety.

1.6.3 Implementation. In addition to our theoretical framework, we also evaluate the real-world practicality of P4TC by means of an implementation. We specifically benchmarked our protocols on an embedded processor which is known to be used in currently available OBUs such as the Savari MobiWAVE [63]. The major advantage for real-world deployment originates in the use of non-interactive zero-knowledge proofs, where major parts of the proofs can be precomputed and verification equations can be batched efficiently. This effectively minimizes the computations which have to be performed by the OBU and the RSU during an actual protocol run. Our implementation suggests that provably-secure ETC at full speed can indeed be realized using present-day hardware.

2 SECURITY MODEL

Our toll collection system $\mathcal{F}_{\text{P4TC}}$ is defined within the (G)UC-framework by Canetti [22], which is a simulation-based security notion. We briefly explain the UC-framework in this section; for a more detailed introduction to UC and its variant GUC see [20, 22].

2.1 Introduction to UC

In the UC-framework security is defined by indistinguishability of two experiments: the *ideal experiment* and the *real experiment*. In the ideal experiment the task at hand is carried out by dummy parties with the help of an ideal incorruptible entity—called the ideal functionality $\mathcal{F}$—which plainly solves the problem at hand in a secure and privacy preserving manner. In the real experiment the parties execute a protocol π in order to solve the prescribed tasks themselves. A protocol π is said to be a (secure) *realization* $\mathcal{F}$ if no PPT-machine $\mathcal{Z}$, called the *environment*, can distinguish between two experiments.

2.1.1 The Basic Model of Computation. The basic model of computation consists of a set of instances (ITIs) of interactive Turing machines (ITMs). An ITM is the description of a Turing machine with additional tapes for its identity, for subroutine input and output and for incoming and outgoing network messages. An ITI is a tangible instantiation of an ITM and is identified by the content of its identity tape. The order of activation of the ITIs is message-driven. If the ITI provides subroutine output or writes an outgoing message, the activation of the ITI completes and the ITI to whom the message has been delivered to gets activated next. Each experiment has two special ITIs: the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ (in the real experiment) or the simulator $\mathcal{S}$ (in the ideal experiment). The environment $\mathcal{Z}$ is the ITI that is initially activated. If the environment $\mathcal{Z}$ provides subroutine output, the whole experiments stops. The output of the experiment is the output of $\mathcal{Z}$.

2.1.2 The (Dummy) Adversary. The adversary $\mathcal{A}$ is instructed by $\mathcal{Z}$ and represents $\mathcal{Z}$'s interface to the network, e.g., $\mathcal{A}$ reports all network messages generated by any party to $\mathcal{Z}$ and can manipulate, reroute, inject and/or suppress messages on $\mathcal{Z}$'s order. This modeling reflects the idea of an unreliable and untrusted network. Please note: Only incoming/outgoing messages are under the control of $\mathcal{A}$. Subroutine input/output is authenticated, confidential and of integrity. Moreover, $\mathcal{Z}$ may instruct $\mathcal{A}$ to corrupt a party. 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 in the name of the corrupted party as requested by $\mathcal{Z}$.

2.1.3 The Real Experiment. In the real experiment, denoted by $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(1^n)$, the environment $\mathcal{Z}$ 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.

2.1.4 The Ideal Experiment. In the ideal experiment, denoted by $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^n)$, the protocol parties are mere dummies that pass their input to a trusted third party $\mathcal{F}$ and hand over $\mathcal{F}$'s output as their own output. The ideal functionality $\mathcal{F}$ executes the task at hand in a trustworthy manner and is incorruptible. 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).

2.1.5 Definition of Security. A protocol π is said to securely UC-realize an ideal functionality $\mathcal{F}$, denoted by $\pi \geq_{\mathrm{UC}} \mathcal{F}$, if and only if

$$\exists \mathcal{S} \, \forall \mathcal{Z} : \mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(1^n) \overset{\mathrm{c}}{\equiv} \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^n)$$

holds whereby the randomness is taken over the initial input of $\mathcal{Z}$ and the random tapes of all ITIs.[5] If no environment $\mathcal{Z}$ can tell executions of the real and the ideal experiment apart, then any successful attack existing in the real experiment would also exist in the ideal experiment. Therefore the real protocol π guarantees the same level of security as the (inherently secure) ideal functionality $\mathcal{F}$.

2.2 Remarks on Privacy

Regarding privacy, please note that all parties (including the simulator) use the ideal functionality as a black-box and only know what the ideal functionality explicitly allows them to know as part of their prescribed output. The output to the simulator is also called leakage. 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 defined output of the ideal functionality. In other words, the privacy assessment can be conducted onto the ideal functionality and is completely decoupled from the analysis of the protocol implementation. The proof of indistinguishability asserts that any secure realization of the functionality provides the same level of privacy.

2.3 UC Model Conventions

The bare UC model does not specify many important aspects. For example, it leaves open which ITIs are allowed to communicate with each other, how parties are corrupted and which ITI is allowed to invoke what kind of new ITIs. In this section we clarify these aspects. Our conventions are probably the mostly used one and quite natural.

- Each party is identified by its party identifier (PID) *pid* which is unique to the party and is the UC equivalent of the physical identity of this party. A party runs a protocol π by means of an ITI which is called the main party of this instance of π. An ITI can invoke subsidiary ITIs to execute sub-protocols. A subsidiary and its parent use their *subroutine input/output tapes* to communicate with each other. The set of ITIs taking part in the same protocol but for different parties communicate through their *message tapes*. An instance of a protocol is identified by its session identifier (SID) *sid*. All ITIs taking part in the same protocol instance share the same SID. A specific ITI is identified by its ID $id = (pid, sid)$.
- Ideal functionalities are an exception to this rule. An instance of an ideal functionality $\mathcal{F}$ owns a SID but no PID. Input to and output from $\mathcal{F}$ is performed through dummy parties which share the same SID as $\mathcal{F}$, but additionally have the PIDs of the parties they belong to.
- As already stated in Section 2.1.2 subroutine input/output is identifying. This implies in particular that an ideal functionality knows the *pid* of the party to/from whom it writes/reads input/output, because an ideal functionality communicates with its calling ITI by means of intermediary dummy ITIs and subroutine input/output.
- We assume PID-wise corruption: either all ITIs of party are corrupted or none.

All these conventions capture the natural intuition that ITIs with the same PID represent processes within the same physical computer and therefore can communicate with each other directly. But communication between

[5] N.b.: To streamline this short introduction we directly defined the so-called dummy adversary. The original definition all-quantifies over all adversaries in the first step. In the second step it is shown that the dummy adversary—as defined here—is the most severe one and complete.

different physical computers (i.e., parties) is only possible through an unreliable network under the control of an adversary.

2.4 Setup Assumptions — CRS $\mathcal{F}_{\text{CRS}}$ and Bulletin-Board $\overline{\mathcal{G}}_{\text{bb}}$

As commonly found in the UC setting, we also draw from constitutive trust assumptions, or setup assumptions in the UC terminology. Setup assumptions are ideal functionalities that remain ideal in the real experiment, e.g., their secure realization is left unspecified. In our scenario, we assume a common reference string (CRS), denoted $\mathcal{F}_{\text{CRS}}$, and a globally available bulletin board, $\overline{\mathcal{G}}_{\text{bb}}$ [24, Fig. 3], which is sometimes also referred to as key registration service in the literature [8, 21].

A CRS is a short piece of information that has been honestly generated and is shared between all parties. A bulletin board can be depicted as the abstract formalization of a globally available key registration service which associates (physical) party identifiers (PIDs) with (cryptographic) public keys. Every party has the option to register a key for itself once and $\overline{\mathcal{G}}_{\text{bb}}$ ensures that the registering party cannot lie about its (physical) identity *pid*. 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. The key can be any value v, i.e., $\overline{\mathcal{G}}_{\text{bb}}$ is totally agnostic to the data. Moreover, every party can obtain keys that have been registered by any other party in a trustworthy way. Our augmentations to $\overline{\mathcal{G}}_{\text{bb}}$ are only syntactical bagatelles. Parties can also perform a reverse lookup, i.e., lookup the PID of the party that registered a value, and parties can not only register an opaque string of bits but an ordered tuple of strings. The latter is required to support reverse searches on substrings.

2.5 Generalized UC

In the plain UC model the environment $\mathcal{Z}$ is only allowed to invoke a single session of the challenge protocol and is only allowed to exchange messages with the respective main parties. This implies that many real-world scenarios cannot be appropriately captured with plain UC. For example, a globally available PKI that also provides its services to other protocols cannot be modeled in UC. Generalized UC [22] and its simplification Externalied UC (EUC) relaxes this restriction such that a particular, designated ITI—the global functionality—can be accessed by any other ITI including the environment. In our setting, the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ is allowed to be global.

3 SYSTEM DEFINITION

We now give a condensed description of our ideal functionality $\mathcal{F}_{\text{P4TC}}$ and point out how it ensures the desired properties given in Section 1.5.3. We do not formalize each task (e.g., Wallet Issuing, Debt Accumulation, …) as an individual ideal functionality, but the whole system as a monolithic, highly reactive functionality $\mathcal{F}_{\text{P4TC}}$ with polynomially many parties as users and RSUs. This allows for a shared state between the individual interactions and to define correctness and security more easily.

An excerpt of $\mathcal{F}_{\text{P4TC}}$ is depicted in Fig. 2. For better understanding we refrain from giving a complete description of all the tasks $\mathcal{F}_{\text{P4TC}}$ provides at this point. Instead we focus on the most important task, namely Debt Accumulation, and only sketch the remaining tasks. We also state why the ideal model reflects the security and privacy level we wish to achieve. Again, we only concentrate on Debt Accumulation. The full-fledged ideal functionality can be found in Appendix B.

For the ease of presentation, all instructions in Fig. 2 that are typically executed are printed in normal font, while some conditional side tracks are grayed out. The normal execution path defines the level of security and privacy achieved for honest, well-behaving parties. The conditional branches deal with corrupted, misbehaving,[6] or blacklisted parties. The key idea behind $\mathcal{F}_{\text{P4TC}}$ is to keep track of all conducted transactions in a pervasive

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

Functionality $\mathcal{F}_{\text{P4TC}}$

I. State

- A set *TRDB* of transaction entries *trdb* having the form

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

- A (partial) mapping f_Φ assigning a wallet ID λ and a counter x to a fraud detection ID ϕ:

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

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

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

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

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

II. Behavior (Task Debt Accumulation only)

User input: (`pay_toll`, s^{prev})
RSU input: (`pay_toll`, $bl_{\mathcal{R}}$)

(1) Pick serial number $s \overset{R}{\leftarrow} S$ 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) If $\phi^{\text{prev}} \in bl_{\mathcal{R}}$, output `blacklisted` to both parties and abort.
(5) Adopt previous walled ID $\lambda := \lambda^{\text{prev}}$ and increase counter $x := x^{\text{prev}} + 1$.
(6) *Double-spending/Blacklisted:* In this case $\phi := f_\Phi(\lambda, x)$ has already been defined; continue with (7).

Pick fraud detection ID $\phi \overset{R}{\leftarrow} \Phi$ 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.

(7) Look up attributes $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}) := \left(f_{\mathcal{A}_{\mathcal{U}}}(\lambda), f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}), f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}})\right)$.
(8) Calculate price $p := \mathrm{O}_{\text{pricing}}(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$.
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 $b := b^{\text{prev}} + p$.
(10) Append new transaction $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ to *TRDB*.

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

[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.

Fig. 2. An excerpt of the ideal functionality $\mathcal{F}_{\text{P4TC}}$.

transaction database *TRDB* (cp. Fig. 2). Each transaction entry $trdb \in TRDB$ 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,[7] the price p (toll) associated with this particular transaction and the total balance b of the user's wallet, i.e., the accumulated sum of all prices so far including this transaction. In other words, $\mathcal{F}_{\text{P4TC}}$ implements a trustworthy global bookkeeping service that manages 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, reuse old wallet states and commit double-spending without being noticed *instantly*.
- Double-spending is eventually detected after-the-fact.

In order to accurately define security, these technicalities have to be incorporated into the ideal functionality, which causes the bookkeeping to be more involved. The whole transaction database is best depicted as a directed graph. 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 $(pid_{\mathcal{R}}, p)$. A user's wallet is represented by the subgraph of all nodes with the same wallet ID λ and forms a connected component. As long as a user does not commit 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 degenerates into a directed tree. In this case, all transaction entries that constitute a double-spending, i.e., all nodes with the same predecessor, should share the same fraud detection ID ϕ. To this end, the counter x and the injective map $f_\Phi : \mathcal{L} \times \mathbb{N}_0 \to \Phi$ have been introduced in order to manage fraud detection IDs consistently: 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, given its wallet (aka tree) λ and its depth x.

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. To give a better understanding of how $\mathcal{F}_{\text{P4TC}}$ works, we explain the task Debt Accumulation in more detail.

3.1 Task "Debt Accumulation"

In this task, the user provides a serial number s^{prev} as input to $\mathcal{F}_{\text{P4TC}}$, indicating which past wallet state he wishes to use for this 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 (and only 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. This is a required technicality as corrupted users might share their credentials and thus might use each

[7] The party identifier (PID) can best be depicted as the model's counterpart of a party's physical identity in the real world. E.g., the user's physical identity could 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 the definition of the UC framework.

other's wallet. Please note that this does not affect honest users. $\mathcal{F}_{\text{P4TC}}$ looks up the previous wallet state $trdb^{\text{prev}}$ in *TRDB* and asserts that its fraud detection ID is not blacklisted; otherwise it aborts. The ideal functionality extracts the wallet ID from the previous record ($\lambda := \lambda^{\text{prev}}$) and increases the counter for this particular wallet ($x := x^{\text{prev}} + 1$). Then, it checks if a fraud detection ID ϕ has already been defined for the wallet ID λ and counter x (n.b.: tree and depth of 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 λ. 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.[8] Moreover, $\mathcal{F}_{\text{P4TC}}$ looks up the user's attributes bound to this particular wallet ($\mathbf{a}_{\mathcal{U}} := f_{\mathcal{A}_{\mathcal{U}}}(\lambda)$) and the attributes of the current and previous RSU ($\mathbf{a}_{\mathcal{R}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}})$, $\mathbf{a}^{\text{prev}}_{\mathcal{R}} := f_{\mathcal{A}_{\mathcal{R}}}(pid^{\text{prev}}_{\mathcal{R}})$). 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 ($b := b^{\text{prev}} + p$) 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 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 (before making the actual payment).

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 his 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 $\mathbf{a}^{\text{prev}}_{\mathcal{R}}$ of the user and the previous RSU, respectively. Learning their mutual attributes is necessary, because the RSU and user must evaluate the pricing function themselves in the real implementation without the help of a third party.[9]

3.2 Correctness and Operator Security

Both are immediately asserted by the ideal functionality. As $\mathcal{F}_{\text{P4TC}}$ represents an incorruptible bookkeeper, the user has no chance to cheat. The only input of a (possibly malicious) user is his choice of which previous wallet state should be used. Hence, the user has no chance to lie about his attributes $\mathbf{a}_{\mathcal{U}}$, the balance b^{prev} associated with this state, the calculated new balance b nor anything else. If the user is malicious, the adversary has the additional options to choose an alternative (malicious) user that is being charged and to choose a different fraud detection ID. Both do not effect operator security. The former does not change the total amount due to the operator. It only changes the party liable which is unavoidable, if a group of corrupted users collude. The latter does not affect the operator at all.

3.3 User Security and Privacy Leakage

User security follows by the same arguments as for operator security.

The information leakage that needs to be considered for an assessment of user privacy directly follows from the in- and output of the ideal functionality. At this point, we exemplarily consider the leakage of Debt Accumulation only. We refer to Appendix A for an overview of the leakage of the remaining tasks.

[8] 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.

[9] At least, it is unavoidable for any *practical* implementation given our scenario. Due to the offline capabilities there is no third party available. Hence, the only alternative approach would be to use some generic SFE/2PC techniques which are prohibitively inefficient for an open-road ETC system using low-end hardware. Moreover, the legal framework of most countries requires that the value being charged can be checked manually.

We stress that we only care about privacy for honest, well-behaving, non-blacklisted[10] users. Hence, the grayed out steps can be ignored. 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. The RSU only learns the serial number s and the fraud detection ID ϕ of the current transaction which are both freshly, uniformly and independently drawn by $\mathcal{F}_{\text{P4TC}}$. Hence, it is *information-theoretically impossible* to track the honest and well-behaving user across any pair of transactions using any of these numbers. The only "real" information leakage (to the RSU) in this task is determined by the user's and the previous RSU's attributes $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$.

Independence of the fraud detection ID might be lost for two cases:

- The user is corrupted and the adversary playing a corrupted user knows all his secrets anyway. In this case, the adversary is allowed to pick the fraud detection ID if not previously defined.
- The mapping $(\lambda, x) \mapsto \phi$ is already defined, because the user repeatedly runs Debt Accumulation with the same inputs and thus tries to attempt double-spending or the user has been blacklisted. (Our blacklisting task—sketched later—prefills f_Φ and creates a pool of fraud detection IDs to be used.)

In neither of these cases we aim to provide untrackability.

3.4 Remaining Tasks

We briefly sketch the remaining tasks of the ideal functionality omitting some details regarding corrupted parties. Full details are given in Appendix B.

Registration and Certification: There are auxiliary tasks for registration/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.

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

Debt Clearance: 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 him and check if he (physically) pays the correct amount. Also, the user does not obtain a new serial number such that this the transaction entry becomes a leaf node of the wallet tree.

User Blacklisting: The task User Blacklisting 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 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.

Prove Participation: The task Prove Participation simply checks if a record exists in the database for the particular user and serial number.

Double-Spending Detection: 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. Please note, correctness, completeness and soundness is is guaranteed by the ideal bookkeeping service.

Guilt Verification: The task Guilt Verification checks if the given proof is internally recorded as being issued for the particular user.

[10] Note that the TSP may only blacklist users with the help of the incorruptible DR who only cooperates if the user agreed or misbehaved.

3.5 Mapping of the Desired Properties to the Ideal Model

In this section, we illustrate how the definition of our ideal model $\mathcal{F}_{\text{P4TC}}$ ensures the desired properties of a toll collection scheme (cp. Section 1.5.3, (P1)-(P10)).

(P1) to (P3): In the tasks of Debt Accumulation and Debt Clearance, the user only ever inputs the serial number of the wallet state he wants to use. All relevant information is then looked up internally by $\mathcal{F}_{\text{P4TC}}$. Therefore, the user is not able to pretend his wallet state contains any other information than it actually does (like different attributes in Debt Accumulation or a lower balance in Debt Clearance). $\mathcal{F}_{\text{P4TC}}$ also checks that the wallet actually belongs to this user. Only corrupted users are able to use wallets of one another, but not wallets of honest users. For a more formal proof of property (P3) see Lemma F.7.

(P4): As explained above, the same fraud detection ID ϕ occurs in multiple transaction entries if and only if double-spending was committed. The TSP can check this using the task Double-Spending Detection. If double-spending is detected, $\mathcal{F}_{\text{P4TC}}$ provides the TSP with the identity $pid_{\mathcal{U}}$ of the respective user and a publicly verifiable proof π that this user has actually committed double-spending. This prevents the TSP from falsely accusing a user of double-spending. For a more formal proof see Lemmas F.6 and F.8.

(P5) and (P7): As discussed in Section 1.5, we assume that if a user does not properly participate in Debt Accumulation, he is physically identified outside the scope of $\mathcal{F}_{\text{P4TC}}$. The important feature $\mathcal{F}_{\text{P4TC}}$ provides for this property is the task Prove Participation. In this task the SA inputs a set $S_{\mathcal{R}}^{\text{pp}}$ of serial numbers and the ID of the user in question. With the users consent $\mathcal{F}_{\text{P4TC}}$ then checks whether the user successfully participated in any of the transactions from this list. If so, it just responds with OK to the SA who does not learn anything about any of the user's transactions beyond that.

(P6): After an instance of Debt Accumulation, an RSU gets the information $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})$ as output. As already discussed in Section 1.5, we assume user and RSU attributes to be sufficiently indistinct that they do not enable tracking of the user. This is not ensured within the scope of $\mathcal{F}_{\text{P4TC}}$, apart from attribute outputs to the user which enable him to check they are not identifying. Unless double-spending is committed, the serial numbers s and fraud detection IDs ϕ are fresh and randomly drawn for every transaction and can therefore not compromise unlinkability. In case of double-spending, the TSP is able to directly link transactions with the same fraud detection ID and also find out the ID of the user responsible, but is still only able to link any other transactions of this user with help from the DR.

(P8): Given a user ID $pid_{\mathcal{U}}$, the task User Blacklisting provides the TSP a set of fraud detection IDs ϕ of all wallets λ of this user. In Debt Accumulation, the RSU inputs a blacklist $bl_{\mathcal{R}}$ (containing fraud detection IDs from the TSP). Then, $\mathcal{F}_{\text{P4TC}}$ checks whether the fraud detection ID of the wallet state the user wishes to use for the transaction is contained in this list. For a more formal proof see Lemma F.9.

(P9): Within the task of User Blacklisting, $\mathcal{F}_{\text{P4TC}}$ 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.

(P10): 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 1.5.2).

4 PROTOCOL OVERVIEW

For the sake of brevity, this section gives only a *simplified* description of the main protocols illustrating the ideas behind P4TC. For the full protocols see Appendix D.

Before describing our protocols, some remarks about the used crypto building blocks, secure channels and the structure of user wallets are in order.

4.1 Crypto Building Blocks and Algebraic Setting

Our construction makes use of (F_{gp}-extractable) 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 (which is Groth-Sahai [39] in our case). For readers not familiar with these building blocks, we refer to Appendix C for a detailed description.

Our system instantiation is based on an asymmetric bilinear group setting $(G_1, G_2, G_{\mathrm{T}}, e, \mathfrak{p}, g_1, g_2)$. Here, G_1, G_2, G_{T} are cyclic groups of prime order $\mathfrak{p}$ (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 $e\colon\ G_1 \times G_2 \to G_{\mathrm{T}}$ is a (non-degenerated) bilinear map. We rely on the co-CDH assumption (Definition C.3) as well as on the security of the building blocks in this setting.

In this section we use a simpler notation than the one introduced in Appendix C in order to improve the comprehension. Here, a commitment scheme consists of the PPT algorithms Com and Open, where Com returns a commitment c together with an opening information d for a given message m and Open verifies if a commitment c opens to a particular message m given the opening information d. A digital signature scheme consists of the PPT algorithms Sgn and Vfy, where Sgn takes as input the secret key sk and a message $m \in \mathcal{M}$, and outputs a signature σ and Vfy takes as input the public key pk, a message $m \in \mathcal{M}$, and a purported signature σ, and outputs a bit indicating whether or not the signature is valid.

4.2 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. We omit these encryptions when describing the protocols. Apart from the task Debt Accumulation we furthermore assume all channels to be authenticated.

4.3 Wallets

A user wallet essentially consists of two signed commitments $(c_{\mathcal{T}}, c_{\mathcal{R}})$, where $c_{\mathcal{T}}$ represents the fixed part of a wallet and $c_{\mathcal{R}}$ the updatable part. Accordingly, the fixed part is signed along with the user's attributes $\mathbf{a}_{\mathcal{U}}$ by the TSP using $\mathrm{sk}_{\mathcal{T}}^{\mathcal{T}}$ during wallet issuing. Every time $c_{\mathcal{R}}$ is updated, it is signed along with the serial number s of the transaction by the RSU using $\mathrm{sk}_{\mathcal{R}}$. The fixed part $c_{\mathcal{T}} = \mathrm{Com}(\lambda, \mathrm{sk}_{\mathcal{U}})$[11] is a commitment on the PRF key λ (which is used as wallet ID) and the user's secret key $\mathrm{sk}_{\mathcal{U}}$. The updatable part $c_{\mathcal{R}} = \mathrm{Com}(\lambda, b, u_1, x)$ 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 state of the wallet, and a counter value x being the input to the PRF. The value $\mathrm{PRF}(\lambda, x)$ serves as the wallet's fraud detection ID ϕ. This choice of the fraud detection ID has the advantage that the different versions of a wallet are traceable given λ but untraceable if λ is unknown.

4.4 The P4TC Protocols

We stress again that the following description of the P4TC protocols is a simplified version. The actual protocols, including protocol interaction diagrams, can be found in Appendix D.

4.4.1 System Setup. In the setup phase, a trusted third party generates the bilinear group and a public common reference string for the system. The latter contains parameters for some of the building blocks, namely the NIZK and commitment scheme(s) we use.

[11] Note that by abuse of notation, we sometimes ignore the opening or decommitment value which is also an output of $\mathrm{Com}(\cdot)$.

4.4.2 DR/TSP/RSU Key Generation. The DR generates a key pair $(\mathrm{pk}_{DR}, \mathrm{sk}_{DR})$ for an IND-CCA secure encryption scheme, where pk_{DR} is used to deposit a user-specific trapdoor (a PRF-key) which allows to link this user's transactions in case of a dispute. An individual signature key pair $(\mathrm{pk}_{\mathcal{R}}, \mathrm{sk}_{\mathcal{R}})$ is generated for each RSU to sign the updatable part of a user's wallet. Moreover, the TSP generates several key pairs $(\mathrm{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathrm{sk}_{\mathcal{T}}^{\mathcal{T}})$, $(\mathrm{pk}_{\mathcal{T}}^{\mathrm{cert}}, \mathrm{sk}_{\mathcal{T}}^{\mathrm{cert}})$, $(\mathrm{pk}_{\mathcal{T}}^{\mathcal{R}}, \mathrm{sk}_{\mathcal{T}}^{\mathcal{R}})$ for an EUF-CMA secure signature scheme. The key $\mathrm{sk}_{\mathcal{T}}^{\mathcal{T}}$ is used to sign fixed user-specific information when a new wallet is issued. Using $\mathrm{sk}_{\mathcal{T}}^{\mathcal{R}}$, the TSP can play the role of the initial RSU to sign the updatable part of a new wallet.

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

4.4.4 User Registration. Before a user can participate in the system, he needs to generate a key pair $(\mathrm{pk}_{\mathcal{U}} = g_1^{\mathrm{sk}_{\mathcal{U}}}, \mathrm{sk}_{\mathcal{U}}) \in G_1 \times \mathbb{Z}_\mathrm{p}$. We assume that binding $\mathrm{pk}_{\mathcal{U}}$ to a verified physical user ID, in order to hold the user liable in case of a misuse, is done out-of-band. For instance, this could be done by some trusted (external) certification authority. Throughout the protocols the key pair $(\mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}})$ is used to bind a wallet to a user.

4.4.5 Wallet Issuing. This protocol is executed between a user and the 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 his 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 recalculations and blacklisting, 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 λ is fixed and the user knows his own share λ' as well as the TSP's share λ''. Then the user computes the commitments $c_{\mathcal{T}} = \mathrm{Com}(\lambda, \mathrm{sk}_{\mathcal{U}})$ and $c_{\mathcal{R}} := \mathrm{Com}(\lambda, b := 0, u_1, x := 0)$. Additionally, he prepares the deposit of λ. This is a bit tricky, as the user needs to prove that the ciphertext he gives to the TSP is actually an encryption of λ under pk_{DR}. For practical reasons, we use Groth-Sahai NIZKs and the Dodis-Yampolski PRF. 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., $\mathbb{Z}_\mathrm{p}$. Unfortunately, we are not aware of any such scheme.[12] Instead, we use a CCA-secure structure-preserving encryption scheme for vectors of G_1-elements 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 $\Lambda'_i := g_1^{\lambda'_i}$ becomes feasible. All chunks $\Lambda_{i \in \{0,\ldots,\ell-1\}}$, the TSP's share $\Lambda'' := g_1^{\lambda''}$, and the user's public key $\mathrm{pk}_{\mathcal{U}}$ are jointly encrypted as e^* under pk_{DR}. The CCA-secure ciphertext e^* unambiguously binds the PRF key λ to the user's key $\mathrm{pk}_{\mathcal{U}}$ and rules out malleability attacks. Otherwise, a malicious TSP could potentially trick the DR into recovering the trapdoor for a different (innocent) user. It is not necessary, to split the TSP share λ'', because it is known to the TSP anyway and thus can additionally be stored in the clear alongside e^*. The user then sends over e^*, $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ along with a NIZK π proving that everything has been generated honestly and the wallet is bound to the user owning $\mathrm{sk}_{\mathcal{U}}$. When the TSP receives this data, it verifies the NIZK first. If the check passed, the TSP signs $c_{\mathcal{T}}$ along with attributes $\mathbf{a}_{\mathcal{U}}$ and $c_{\mathcal{R}}$ along with s using $\mathrm{sk}_{\mathcal{T}}^{\mathcal{R}}$.[13] The resulting signatures $\sigma_{\mathcal{T}}$, $\sigma_{\mathcal{R}}$ are sent to the user, who checks their correctness. The user finally stores his freshly generated state token

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

[12] Note that Paillier encryption works in a different algebraic setting and cannot easily be combined with Groth-Sahai proofs.

[13] During the protocol run, a uniformly random serial number s for this transaction was jointly generated by user and TSP by means of a Blum-like coin toss (see Debt Accumulation for a description).

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

4.4.6 User Blacklisting. This is a protocol 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 $\mathsf{pk}_{\mathcal{U}}^{DR}$ should be blacklisted before the protocol starts. For example, the TSP might have presented a proof of guilt to the DR or the user affirmed to be voluntarily blacklisted (for example due to a stolen car). Given e^* and λ'', the DR recovers the contained PRF key but only if it is bound to $\mathsf{pk}_{\mathcal{U}}^{DR}$. To this end, DR decrypts e^* using sk_{DR} to obtain Λ'_i, Λ'', and $\mathsf{pk}_{\mathcal{U}}$. If $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}_{\mathcal{U}}^{DR}$ 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. This is feasible as the DLOGs are very small. The key is computed as $\lambda := \lambda'' + \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$. The DR also checks if the provided share of the TSP λ'' matches the decrypted version Λ'' and if the decrypted user key $\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}$ equals the expected key $\mathsf{pk}_{\mathcal{U}}^{DR}$. If the check fails, the DR aborts.

Using λ, the fraud detection IDs belonging to the previous and upcoming versions 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 precomputed and blacklisted.

4.4.7 Debt Accumulation. In this protocol (Fig. 3) 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 toll p may depend on the user's attributes, the current and the previous reader's attributes, as well as other factors like the current time, etc. The user stays anonymous, i.e., the RSU does not receive the user's key or see the previous balance. Only $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ (the attributes of the RSU who signed the previous state of the wallet) are revealed to the RSU, which are assumed not to contain identifying information.

The user's main input is the state token

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

containing the state of his previous protocol run and wallet. To update his wallet state, the user computes a fresh commitment $c'_{\mathcal{R}}$ 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 randomness 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 his new wallet state certified by the RSU, the user needs to invalidate his previous state. To do so, he needs to reveal the fraud detection ID and double-spending tag of his previous state. For the latter, the RSU sends a random challenge u_2 along with a commitment $c''_{\text{ser}} = \mathsf{Com}(s'')$ on his share of the serial number of this transaction (which is part of the Blum coin toss). Upon receiving these values, the user computes the double-spending tag $t := u_2 \cdot \mathsf{sk}_{\mathcal{U}} + u_1^{\text{next}} \bmod \mathfrak{p}$ (a linear equation in the unknowns u_1 and $\mathsf{sk}_{\mathcal{U}}$), the fraud detection ID $\phi := \mathsf{PRF}(\lambda, x)$, and a hidden user ID $c_{\mathsf{hid}} := \mathsf{Com}(\mathsf{pk}_{\mathcal{U}})$. The latter is used in Prove Participation to associate this interaction with the user.[14] As response, the user sends over c_{hid}, $c'_{\mathcal{R}}$, 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 $c_{\mathcal{T}}$ and $c_{\mathcal{R}}^{\text{prev}}$ such that $c_{\mathcal{R}}^{\text{prev}}$ and $c'_{\mathcal{R}}$ are commitments on the same messages except for the double-spending randomness, that the (hidden) signature on $c_{\mathcal{R}}^{\text{prev}}$ verifies under some (hidden) RSU key $\mathsf{pk}_{\mathcal{R}}^{\text{prev}}$ certified by the TSP, and that t, ϕ, and c_{hid} have been computed using the values contained in $c_{\mathcal{T}}$ and $c_{\mathcal{R}}^{\text{prev}}$.

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 $c'_{\mathcal{R}}$. The resulting commitment $c_{\mathcal{R}}$ is signed along with the serial number $s := s' \cdot s''$ using $\mathsf{sk}_{\mathcal{R}}$. Then the

[14] We are aware of alternatives realizing this mechanism without introducing c_{hid}, which are, however, less efficient.

$\mathcal{U}(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}})$		$\mathcal{R}(\mathsf{cert}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, bl_{\mathcal{R}})$
$s' \xleftarrow{R} G_1$		$s'' \xleftarrow{R} G_1$
$u_1^{\mathrm{next}} \xleftarrow{R} \mathbb{Z}_p$		$u_2 \xleftarrow{R} \mathbb{Z}_p$
$(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{Com}(\lambda, b^{\mathrm{prev}}, u_1^{\mathrm{next}}, x)$		$(c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) \leftarrow \mathsf{Com}(s'')$
	$\xleftarrow{u_2, c''_{\mathrm{ser}}}$	
$t := \mathsf{sk}_{\mathcal{U}} u_2 + u_1^{\mathrm{next}}$		
$\phi := \mathsf{PRF}(\lambda, x)$		
$(c_{\mathrm{hid}}, d_{\mathrm{hid}}) \leftarrow \mathsf{Com}(\mathsf{pk}_{\mathcal{U}})$		
Compute proof π		
	$\xrightarrow{c_{\mathrm{hid}}, c'_{\mathcal{R}}, \phi, t, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, \pi, s'}$	
		if $\phi \in bl_{\mathcal{R}} \vee \pi$ *does not verify*
		return $\perp$
		Calculate price p
		$s := s' \cdot s''$
		$(c''_{\mathcal{R}}, d''_{\mathcal{R}}) \leftarrow \mathsf{Com}(0, p, 0, 1)$
		$c_{\mathcal{R}} := c'_{\mathcal{R}} \cdot c''_{\mathcal{R}}$
		$\sigma_{\mathcal{R}} \leftarrow \mathsf{Sgn}(\mathsf{sk}_{\mathcal{R}}, (c_{\mathcal{R}}, s))$
	$\xleftarrow{c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p, \mathsf{cert}_{\mathcal{R}}, s'', d''_{\mathrm{ser}}}$	
$d_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$		
if $\mathsf{Open}(s'', c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) = 0 \ \vee$		
$\mathsf{Vfy}(\mathsf{pk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (c_{\mathcal{R}}, s)) = 0$		
return $\perp$		
$\tau := (\mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, \lambda,$		
$b := b^{\mathrm{prev}} + p, u_1^{\mathrm{next}}, x + 1, s := s' \cdot s'')$		
return $(\tau, c_{\mathrm{hid}}, d_{\mathrm{hid}})$		return $(\phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, p,$ $t, u_2, c_{\mathrm{hid}}, s)$

Fig. 3. A Simplified Version of Debt Accumulation

RSU sends $c_{\mathcal{R}}$ to the user, along with its decommitment information $d_{\mathcal{R}}$, its signature $\sigma_{\mathcal{R}}$, the added price p, the certificate for $\mathsf{pk}_{\mathcal{R}}$, RSU's share s'' of the serial number and the decommitment value d''_{ser} for c''_{ser}.

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

$$\tau := (\mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, \lambda, b^{\mathrm{prev}} + p, u_1^{\mathrm{next}}, x + 1, s)$$

containing his new wallet state $c_{\mathcal{T}}, c_{\mathcal{R}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}}$. +He additionally stores $(c_{\mathrm{hid}}, d_{\mathrm{hid}})$, where d_{hid} allows to open the hidden ID. The RSU stores $(\phi, p, t, u_2, c_{\mathrm{hid}}, s)$ as transaction information.

4.4.8 Prove Participation. In this protocol executed between a user and the SA, the user proves that he participated in one of the Debt Accumulation transactions under audit by the SA. This protocol is identifying, as the SA retrieved the user's physical ID from his license plate number and, thus, is aware of his public key $\mathsf{pk}_{\mathcal{U}}$. +First, it sends the list $\Omega_{\mathcal{R}}^{\text{pp}}$ of hidden ID commitments observed by the RSU during the considered interactions. +If the user owns some c_{hid} contained in $\Omega_{\mathcal{R}}^{\text{pp}}$, he simply sends over c_{hid} along with the corresponding opening d_{hid} he stored and the serial number s of the transaction.[15] The SA accepts if the commitment d_{hid} indeed opens to $\text{pk}_{\mathcal{U}}$ and s is part of the corresponding transaction information the RSU stored. No other user may claim to have sent c_{hid}, as this would imply to open c_{hid} with a different public key.

4.4.9 Debt Clearance. In this protocol executed between a user and the TSP, the user identifies himself using $\text{sk}_{\mathcal{U}}$ and reveals the balance for his current wallet state. The wallet is invalidated by not creating a new state. Upon receiving a challenge u_2 from the TSP, the user computes the double-spending tag t and fraud detection ID ϕ for his wallet. This is the same as in Debt Accumulation. Then the user sends over $\text{pk}_{\mathcal{U}}$, b^{prev}, ϕ, t and a NIZK π. This NIZK essentially shows 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 $\text{pk}_{\mathcal{U}}$. Note that if the user does not make use of his latest wallet state, double-spending detection will reveal this. If the proof verifies, the balance and the double-spending information, i.e., $(b^{\text{prev}}, \phi, t, u_2)$, is stored by the TSP.

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

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

5 SECURITY THEOREM

Assuming co-CDH is hard and our building blocks are secure, we prove that the protocols from Section 4 (combined as one comprehensive toll collection protocol π_{P4TC}) GUC-realize the ideal model $\mathcal{F}_{\text{P4TC}}$ from Section 3 (in the $(\mathcal{F}_{\text{CRS}}, \overline{\mathcal{G}}_{\text{bb}})$-hybrid model), i.e.,

$$\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \overline{\mathcal{G}}_{\text{bb}}} \geq_{\text{UC}} \mathcal{F}^{\overline{\mathcal{G}}_{\text{bb}}}.$$

Informally, this means the ideal model and our protocol are indistinguishable and therefore provide the same guarantees regarding security and privacy. The statement holds given 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. In Appendix E.1, more details on corruption possibilities are presented. Other details of our adversarial model, i.e. the underlying channel model and handling of aborts, can be found in Appendix E as well. Section 5 gives an outline of the proof, the full proof is given in Appendix F.

Proof Outline

As mentioned above, we separately prove correctness, operator security and user security and privacy.

[15] By additionally providing a time interval out-of-band this search can be accelerated.

The first step does not consider the real protocol, but only examines the ideal functionality. Showing correctness separately, serves two purposes: The ideal functionality is rather complex and virtually constitutes a protocol on its own. Hence it is not immediately obvious whether the ideal functionality actually does what it is expected to do. From this perspective, the proof of correctness is closely coupled to the properties we want to achieve (cp. Sections 1.5.3 and 3.5). In some way, the proof of correctness provides a more formal backup why the desired properties are fulfilled. Secondly, for didactic reasons the proof of correctness is convenient to introduce notation and to present the graph-theoretical approach that is common to all our proofs.

To show correctness of the ideal functionality (cp. Appendix F.1), we start by asserting several structural properties of the ideal transaction database *TRDB*. 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 define a graph structure with vertices being represented by serial numbers s and edges by pairs (s^{prev}, s). Assigning a label $(\phi, \lambda, pid_{\mathcal{U}}, b)$ to the vertex s and a label $(pid_{\mathcal{R}}, p)$ to the edge (s^{prev}, 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.

The proofs of operator security and user security are conducted by explicitly specifying a simulator for the ideal experiment. In addition a sequence of hybrid experiments defines a stepwise transformation from the real experiment to the ideal experiment. The indistinguishability between the real and ideal experiment follows from the pairwise indistinguishable of consecutive hybrids. The latter is proven by individual reductions to the security of our cryptographic building blocks and hardness assumptions. However, the spirit of the hybrids for operator security at the one hand side and of the hybrids for user security at the other hand side is quite different.

For the proof of operator security (cp. Appendix F.2) input privacy does not significantly matter. There is nearly no input of the operator that needs to be kept secret. Consequently, the user learns nearly everything about the operator even in the ideal model as part of its prescribed output and thus simulation of mostly all messages is perfectly enabled. The crucial point is to prove that no user can deviate from the protocol and thereby cheat the operator. To this end, the simulator maintains a separate graph of transactions $\overline{TRDB}$. This *Simulated Transaction Graph* resembles the Ideal Transaction Graph but augments each node by the in- and out-commitments $(c^{\text{in}}_{\mathcal{R}}, c^{\text{in}}_{\mathcal{T}})$ and $(c^{\text{out}}_{\mathcal{R}}, c^{\text{out}}_{\mathcal{T}})$ from the real protocols. These commitments are the fixed and updateable part of the wallet before and after the transaction (cp. Section 4). This information gives an alternative set of edges where two transactions $\overline{trdb}$ and $\overline{trdb}'$ are connected if $(c^{\text{out}}_{\mathcal{R}}, c^{\text{out}}_{\mathcal{T}})$ corresponds to $(c^{\text{in}}_{\mathcal{R}}{}', c^{\text{in}}_{\mathcal{T}}{}')$. Most hybrids introduce an additional check on this alternative graph structure. Together, these checks assert that the alternative graph structure of the Simulated Transaction Graph coincides with the Ideal Transaction Graph and thus a malicious user cannot cheat with overwhelming probability.

Contrarily, the proof of user security (cp. Appendix F.3) follows a different spirit. In this case, input privacy of the user is the crucial point while deviation from the protocol by the operator is a minor concern. For this reasons, most hybrids replace messages from the user by information-theoretically "empty" messages that are independent from any user secret.

6 PERFORMANCE EVALUATION

In order to evaluate the practicality of P4TC, we implemented our system for a realistic target platform. We performed our measurements for the user side on an i.MX6 Dual-Core processor running at 800MHz with 1GB DDR3 RAM and 4GB eMMC Flash, the same processor as used in the Savari MobiWAVE-1000 OBU [63]. The processor runs an embedded Linux, is ARM Cortex-A9 based (32-bit), and also exists in a more powerful Quad-Core variant. For the RSU hardware we take the ECONOLITE Connected Vehicle CoProcessor Module

Table 1. Averaged performance results for execution time t and transmitted data n. Byte values are rounded.

Protocol	$\lvert\mathbf{a}_{\mathcal{U}}\rvert = \lvert\mathbf{a}_{\mathcal{R}}\rvert = 1$				$\lvert\mathbf{a}_{\mathcal{U}}\rvert = \lvert\mathbf{a}_{\mathcal{R}}\rvert = 4$			
	t_{user}	$t_{\text{RSU/TSP}}$	n_{user}	$n_{\text{RSU/TSP}}$	t_{user}	$t_{\text{RSU/TSP}}$	n_{user}	$n_{\text{RSU/TSP}}$
	[ms]	[ms]	[byte]	[byte]	[ms]	[ms]	[byte]	[byte]
Session Key Generation	15.01	0.03	131	–	15.00	0.01	131	–
Wallet Issuing	27 063.61	(8 490.00)	87 951	944	27 182.82	(8 544.74)	88 107	1 152
Debt Accumulation								
– Precomp (offline)	2 715.03	–	–	–	2 715.66	–	–	–
– Online	348.14	474.64	8 128	976	455.52	525.83	8 336	1 088
– Online (cached certificate)	41.01	474.64	8 128	976	40.23	525.83	8 336	1 088
– Postprocessing (offline)	33.84	–	–	–	33.78	–	–	–
Debt Clearance	2 435.47	(745.30)	7 071	96	2 435.78	(766.90)	7 280	96

as a reference system, which was specifically "designed to enable third-party-developed or processor-intensive applications" [40] and measured on comparable hardware. For simplicity, we measured TSP runtime on a standard laptop featuring an i7-6600U processor, although we can expect the TSP to be equipped with much more powerful hardware.

As the CPU of the OBU and the RSU are comparable, we measured both sides using the OBU processor.

6.1 Building Block Instantiation

We implemented P4TC in C++14 using the RELIC toolkit v.0.4.1, an open source cryptography and arithmetic library written in C, with support for pairing-friendly elliptic curves [5]. We developed our own library for Groth-Sahai NIZK proofs [36, 39] and employed the method in [15] to realize the range proofs required in Wallet Issuing. We make use of two types of commitments: the shrinking commitment scheme from [3], as well as a (dual-mode) extractable commitment scheme from [39]. Moreover, we implemented the structure-preserving signature scheme from [1]. We use an adopted variant of the IND-CCA-secure encryption scheme from [16] (cp. Appendix C.2.5) for encrypting PRF key shares in the Wallet Issuing protocol, as well as the IND-CCA-secure encryption scheme from [25] (in combination with AES-CBC and HMAC-SHA256) to establish secure channels for exchanging protocol messages. The PRF is instantiated with the Dodis-Yampolskiy construction introduced in [35].

6.2 Parameter Choice

As for the bilinear group setting, we use the Barreto-Naehrig curves Fp254BNb and Fp254n2BNb presented by Aranha et al. [11, 50]. For the pairing function, we use the optimal Ate pairing since it results in the shortest execution times [59]. This yields a security level of about 100 bit [9].

We evaluated P4TC for two sizes of attribute vectors: $\lvert\mathbf{a}_{\mathcal{U}}\rvert = \lvert\mathbf{a}_{\mathcal{R}}\rvert = 1$ and $\lvert\mathbf{a}_{\mathcal{U}}\rvert = \lvert\mathbf{a}_{\mathcal{R}}\rvert = 4$. With curves of 254-bit order, each vector component can encode up to 253 bits of arbitrary information. In practice, it should be possible to encode multiple attributes into one such component. Since the actual encoding of data depends on the concrete scenario, we only focus here on evaluating the performance penalties when increasing the size of the vectors.

6.3 Implementation Results

Table 1 shows the results of our measurements on the OBU and on the RSU/TSP side in terms of execution time and transmitted data. All values are averaged over 1000 independent protocol executions. Note that the processor is running an embedded Linux, hence execution times can vary by tens of milliseconds due to internal processes and scheduling. We only considered the main protocols which include the expensive NIZKs. The row entitled "Session Key Generation" in Table 1 includes the execution time and size of data to setup a session key for the secure channel which is established prior to any protocol run.[16] In order to utilize the capabilities of our OBU hardware, the user side algorithms were optimized for two CPU cores. Note that the given TSP times (the gray entries in Table 1) are likely an upper bound, since the TSP is equipped with more powerful hardware than a standard laptop.

Debt Accumulation. While the protocols Wallet Issuing and Debt Clearance can be regarded as non-time-critical, Debt Accumulation is performed while driving (possibly at high speed). Thus, execution has to be as efficient as possible. Fortunately, all parts of the expensive NIZK proof which do not involve the challenge value u_2 (provided by the RSU) can be precomputed (*offline* phase) at the OBU which takes approximately 2700ms. Regarding $|\mathbf{a}_{\mathcal{U}}| = |\mathbf{a}_{\mathcal{R}}| = 1$, computations in this online phase take only approximately 350ms on the OBU, mostly due to the verification of the RSU certificate. When caching valid certificates, the runtime can be reduced to approximately 40ms. After the OBU received a response from the RSU, which computes for about 475ms, the internal wallet has to be updated. Since this step can be done offline again, we measure the execution time separately and call the phase *postprocessing*. We also optimized the computations performed by the RSU, taking advantage of the 4 CPU cores and the batching techniques for Groth-Sahai verification by Herold et al. [43]. In summary, all computations in the online phase of Debt Accumulation can be performed in about 825ms or just 515ms when the certificate has already been cached.

The WAVE data transmission standard on DSRC guarantees a transmission rate of 24 Mbit/s [56]. At this rate, all data of the Debt Accumulation protocol are transmitted in approximately 27ms. While the standard claims communication ranges of up to 1km, we assume a toll collection zone of 50m. Moreover, we may assume one RSU per lane [53] such that the workload can be easily spread among the available units. Going at 120km/h, it takes a car 1.5s to travel this distance, leaving us with a time buffer of about 700ms for uncached certificates or 1s for cached certificates. Considering the mandated safety distance at this speed, there should only be a single car inside the 50m zone on each lane. However, since computations take less time than it takes a car to cross the toll collection zone, the system could theoretically handle cars with a distance of only 28m (uncached certificate) or 17m (cached certificate) at 120km/h. We therefore conclude that the performance is sufficient for real-world scenarios.

Storage Requirements. During Debt Accumulation, the RSU and the OBU collect data in order to, e.g., prevent double-spending or to prove participation in a protocol run. In Debt Accumulation, the OBU has to store 137 bytes of transaction information and (optionally) 268 bytes to cache the RSU certificate. Assuming that in one billing period 10000 transactions are performed by the OBU, it only has to store 1.37MB of transaction information and, even if all visited RSUs were different, 2.68MB of cached certificates. The wallet itself consumes 1kB of memory and is fixed in size. The RSU has to store 246 bytes of transaction information after each run of Debt Accumulation (for 32-bit toll values). All this information is eventually aggregated at the TSP's database. The US-based toll collection system E-ZPass reported about 252.4 million transactions per month in 2016 [41], which would result in a database of size 62GB. In case a wallet is blacklisted, an RSU is updated with a list of future fraud detection IDs. Each detection ID consumes 35 bytes of memory. Using appropriate data structures such as

[16] All communication between the participants is encrypted with AES-CBC and HMAC-SHA256 to realize a secure channel. Therefore, each individual protocol is preceded by a symmetric key exchange via the KEM by Cash et al. [25].

hashsets or hashmaps, a lookup is performed in less than 1ms in sets of size 10^6. Hence, blacklisting does not affect the execution time on the RSU. Such a set consumes 35MB of memory, neglecting the overhead induced by the data structure.

Computing DLOGs. To blacklist a user, the DR has to compute a number of discrete logarithms to recover λ. With our choice of parameters, λ is split into 32-bit values, thus resulting in the computation of eight 32-bit DLOGs. While DLOGs of this size can be brute-forced naively, the technique of Bernstein et al. [14] can be used to speed up this process. Using their algorithm, computing a discrete logarithm in an interval of order 2^{32} takes around 1.5 seconds on a single core of a standard desktop using a 55kB table of precomputed elements. These precomputations need to be done only *once* by the DR when setting up the system and take one hour on a desktop computer. Thus, the required DLOGs can be computed in reasonable time by the DR.

REFERENCES

[1] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. 2011. Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups. In *Advances in Cryptology – CRYPTO 2011 (Lecture Notes in Computer Science)*, Phillip Rogaway (Ed.), Vol. 6841. Springer, Heidelberg, 649–666.

[2] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. 2014. Converting Cryptographic Schemes from Symmetric to Asymmetric Bilinear Groups. In *Advances in Cryptology – CRYPTO 2014, Part I (Lecture Notes in Computer Science)*, Juan A. Garay and Rosario Gennaro (Eds.), Vol. 8616. Springer, Heidelberg, 241–260. https://doi.org/10.1007/978-3-662-44371-2_14

[3] Masayuki Abe, Markulf Kohlweiss, Miyako Ohkubo, and Mehdi Tibouchi. 2015. Fully Structure-Preserving Signatures and Shrinking Commitments. In *Advances in Cryptology – EUROCRYPT 2015, Part II (Lecture Notes in Computer Science)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 35–65. https://doi.org/10.1007/978-3-662-46803-6_2

[4] Carolin Albrecht, Frank Gurski, Jochen Rethmann, and Eda Yilmaz. 2018. Knapsack Problems: A Parameterized Point of View. *Theoretical Computer Science* (2018).

[5] D. F. Aranha and C. P. L. Gouvêa. 2016. RELIC is an Efficient Library for Cryptography. Online Resource. (2016). https://github.com/relic-toolkit/relic.

[6] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. 2010. PrETP: Privacy-Preserving Electronic Toll Pricing (extended version). In *Proceedings of the 19th USENIX Security Symposium*. 63–78.

[7] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. 2015. Anonymous Transferable E-Cash. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings (Lecture Notes in Computer Science)*, Jonathan Katz (Ed.), Vol. 9020. Springer, 101–124. https://doi.org/10.1007/978-3-662-46447-2_5

[8] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. 2004. Universally Composable Protocols with Relaxed Set-Up Assumptions. In *45th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 186–195.

[9] Razvan Barbulescu and Sylvain Duquesne. 2017. Updating key size estimations for pairings. Cryptology ePrint Archive, Report 2017/334. (2017). http://eprint.iacr.org/2017/334.

[10] Amira Barki, Solenn Brunet, Nicolas Desmoulins, Sébastien Gambs, Saïd Gharout, and Jacques Traoré. 2016. Private eCash in Practice (Short Paper). In *International Conference on Financial Cryptography and Data Security (FC 2016)*. Springer, 99–109.

[11] Paulo S. L. M. Barreto and Michael Naehrig. 2006. Pairing-Friendly Elliptic Curves of Prime Order. In *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography (Lecture Notes in Computer Science)*, Bart Preneel and Stafford Tavares (Eds.), Vol. 3897. Springer, Heidelberg, 319–331.

[12] Mihir Bellare. 2015. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. *Journal of Cryptology* 28, 4 (2015), 844–878.

[13] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. 1998. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Advances in Cryptology – CRYPTO'98 (Lecture Notes in Computer Science)*, Hugo Krawczyk (Ed.), Vol. 1462. Springer, Heidelberg, 26–45.

[14] Daniel J. Bernstein and Tanja Lange. 2012. Computing small discrete logarithms faster. Cryptology ePrint Archive, Report 2012/458. (2012). http://eprint.iacr.org/2012/458.

[15] Jan Camenisch, Rafik Chaabouni, and abhi shelat. 2008. Efficient Protocols for Set Membership and Range Proofs. In *Advances in Cryptology – ASIACRYPT 2008 (Lecture Notes in Computer Science)*, Josef Pieprzyk (Ed.), Vol. 5350. Springer, Heidelberg, 234–252.

[16] Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. 2011. Structure Preserving CCA Secure Encryption and Applications. In *Advances in Cryptology – ASIACRYPT 2011 (Lecture Notes in Computer Science)*, Dong Hoon Lee and Xiaoyun Wang (Eds.), Vol. 7073. Springer, Heidelberg, 89–106.

[17] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 3494. Springer, 302–321. https://doi.org/10.1007/11426639_18

[18] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *Advances in Cryptology – EUROCRYPT 2005 (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 3494. Springer, Heidelberg, 302–321.

[19] Sébastien Canard and Aline Gouget. 2008. Anonymity in Transferable E-cash. In *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings (Lecture Notes in Computer Science)*, Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung (Eds.), Vol. 5037. 207–223. https://doi.org/10.1007/978-3-540-68914-0_13

[20] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 136–145.

[21] Ran Canetti. 2007. Obtaining Universally Composable Security: Towards the Bare Bones of Trust. Cryptology ePrint Archive, Report 2007/475. (2007). http://eprint.iacr.org/2007/475.

[22] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. In *TCC 2007: 4th Theory of Cryptography Conference (Lecture Notes in Computer Science)*, Salil P. Vadhan (Ed.), Vol. 4392. Springer, Heidelberg, 61–85.

[23] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. 2002. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*. ACM Press, 494–503.

[24] Ran Canetti, Daniel Shahaf, and Margarita Vald. 2016. Universally Composable Authentication and Key-Exchange with Global PKI. In *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II (Lecture Notes in Computer Science)*, Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang (Eds.), Vol. 9615. Springer, Heidelberg, 265–296. https://doi.org/10.1007/978-3-662-49387-8_11

[25] David Cash, Eike Kiltz, and Victor Shoup. 2008. The Twin Diffie-Hellman Problem and Applications. In *Advances in Cryptology – EUROCRYPT 2008 (Lecture Notes in Computer Science)*, Nigel P. Smart (Ed.), Vol. 4965. Springer, Heidelberg, 127–145.

[26] Xihui Chen, Gabriele Lenzini, Souke Mauw, and Jun Pang. 2012. A Group Signature Based Electronic Toll Pricing System. In *2012 Seventh International Conference on Availability, Reliability and Security (ARES 2012)*. 85–93.

[27] Xihui Chen, Gabriele Lenzini, Sjouke Mauw, and Jun Pang. 2013. Design and Formal Analysis of A Group Signature Based Electronic Toll Pricing System. *JoWUA* 4, 1 (2013), 55–75. http://isyou.info/jowua/papers/jowua-v4n1-3.pdf

[28] European Commission. 2015. Study on State of the Art of Electronic Road Tolling. https://ec.europa.eu/transport/sites/transport/files/modes/road/road_charging/doc/study-electronic-road-tolling.pdf. (2015). [Online; accessed April-19-2018].

[29] European Commission. 2017. 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). https://ec.europa.eu/transport/sites/transport/files/com20170280-eets-directive.pdf. (2017). [Online; accessed April-19-2018].

[30] Morten Dahl, Stéphanie Delaune, and Graham Steel. 2012. Formal Analysis of Privacy for Anonymous Location Based Services. *Theory of Security and Applications* (2012), 98–112.

[31] Datatilsynet. 2007. Statens Vegvesen Holdt Tilbake Viktig AutoPASS-Informasjon (Press release). http://www.datatilsynet.no/. (2007). [Online; accessed April-19-2018].

[32] Jeremy Day, Yizhou Huang, Edward Knapp, and Ian Goldberg. 2011. SPEcTRe: Spot-checked Private Ecash Tolling at Roadside. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society (WPES '11)*. 61–68.

[33] Wiebren de Jonge and Bart Jacobs. 2008. Privacy-Friendly Electronic Traffic Pricing via Commits. In *Formal Aspects in Security and Trust – FAST 2008 (Lecture Notes in Computer Science)*, Vol. 5491. 143–161.

[34] Deutsches Bundesamt für Güterverkehr. 2018. Monatliche Mautstatistik für Januar 2018. https://www.bag.bund.de/SharedDocs/Downloads/DE/Statistik/Lkw-Maut/18_Monatstab_01.html (in German). (2018). [Online; accessed January-09-2019].

[35] Yevgeniy Dodis and Aleksandr Yampolskiy. 2004. A Verifiable Random Function With Short Proofs and Keys. Cryptology ePrint Archive, Report 2004/310. (2004). http://eprint.iacr.org/2004/310.

[36] Alex Escala and Jens Groth. 2014. Fine-Tuning Groth-Sahai Proofs. In *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography (Lecture Notes in Computer Science)*, Hugo Krawczyk (Ed.), Vol. 8383. Springer, Heidelberg, 630–649. https://doi.org/10.1007/978-3-642-54631-0_36

[37] eugdpr.org. 2018. The EU General Data Protection Regulation (GDPR). https://www.eugdpr.org/. (2018). [Online; accessed April-19-2018].

[38] Flavio D Garcia, Eric R Verheul, and Bart Jacobs. 2011. Cell-Based Roadpricing. In *European Public Key Infrastructure Workshop – EuroPKI 2011 (Lecture Notes in Computer Science)*, Vol. 7163. 106–122.

[39] Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *Advances in Cryptology – EUROCRYPT 2008 (Lecture Notes in Computer Science)*, Nigel P. Smart (Ed.), Vol. 4965. Springer, Heidelberg, 415–432.

[40] ECONOLITE Group. 2018. Connected Vehicle CoProcessor Module. http://www.econolitegroup.com/wp-content/uploads/2017/05/controllers-connectedvehicle-datasheet.pdf. (2018). [Online; accessed 07-April-2018].

[41] E-ZPass Group. 2017. E-ZPass Statistics: 2005 - 2016. https://e-zpassiag.com/about-us/statistics. (2017). [Online; accessed May-08-2018].

[42] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. 2017. BBA+: Improving the Security and Applicability of Privacy-Preserving Point Collection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1925–1942. https://doi.org/10.1145/3133956.3134071

[43] Gottfried Herold, Max Hoffmann, Michael Klooß, Carla Ràfols, and Andy Rupp. 2017. New Techniques for Structural Batch Verification in Bilinear Groups with Applications to Groth-Sahai Proofs. In *ACM CCS 17: 24th Conference on Computer and Communications Security*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1547–1564.

[44] Jaap-Henk Hoepman and George Huitema. 2010. Privacy Enhanced Fraud Resistant Road Pricing. *What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience* (2010), 202–213.

[45] Kapsch (Toll Collection System Integrator and Supplier). 2018. Personal Communication. (2018).

[46] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2008. Founding Cryptography on Oblivious Transfer - Efficiently. In *Advances in Cryptology – CRYPTO 2008 (Lecture Notes in Computer Science)*, David Wagner (Ed.), Vol. 5157. Springer, Heidelberg, 572–591.

[47] Roger Jardí-Cedó, Jordi Castellà-Roca, and Alexandre Viejo. 2014. Privacy-Preserving Electronic Toll System with Dynamic Pricing for Low Emission Zones. In *Data Privacy Management, Autonomous Spontaneous Security and Security Assurance – DPM 2014, SETOP 2014 and QASA 2014. Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 8872. 327–334.

[48] Roger Jardí-Cedó, Macià Mut-Puigserver, M. Magdalena Payeras-Capellà, Jordi Castellà-Roca, and Alexandre Viejo. 2014. Electronic Road Pricing System for Low Emission Zones to Preserve Driver Privacy. In *Modeling Decisions for Artificial Intelligence – MDAI 2014. Proceedings*. 1–13.

[49] Roger Jardí-Cedó, Macià Mut-Puigserver, M. Magdalena Payeras-Capellà, Jordi Castellà-Roca, and Alexandre Viejo. 2016. 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)*. 158–165.

[50] Yuto Kawahara, Tetsutaro Kobayashi, Michael Scott, and Akihiro Kato. 2016. *Barreto-Naehrig Curves*. Internet Draft. Internet Engineering Task Force. Work in Progress.

[51] Florian Kerschbaum and Hoon Wei Lim. 2015. Privacy-Preserving Observation in Public Spaces. In *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part II (Lecture Notes in Computer Science)*, Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl (Eds.), Vol. 9327. Springer, Heidelberg, 81–100. https://doi.org/10.1007/978-3-319-24177-7_5

[52] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. 2015. Structure-Preserving Signatures from Standard Assumptions, Revisited. In *Advances in Cryptology – CRYPTO 2015, Part II (Lecture Notes in Computer Science)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.), Vol. 9216. Springer, Heidelberg, 275–295. https://doi.org/10.1007/978-3-662-48000-7_14

[53] Hamish Koelmeyer and Sithamparanathan Kandeepan. 2017. Tagless Tolling using DSRC for Intelligent Transport System: An Interference Study. In *Asia Modelling Symposium*. IEEE.

[54] Sebastian Kummer, Maria Dieplinger, and Mario Dobrovnik. 2015. Endbericht der Studie "Flächendeckende Schwerverkehrs-Maut in Österreich". https://www.wko.at/branchen/stmk/transport-verkehr/gueterbefoerderungsgewerbe/Endbericht_FlaechendeckendeMaut_final.pdf (in German). (2015). [Online; accessed January-09-2019].

[55] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. 2010. 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 (IFIP Advances in Information and Communication Technology)*, Kai Rannenberg, Vijay Varadharajan, and Christian Weber (Eds.), Vol. 330. Springer, 289–301. https://doi.org/10.1007/978-3-642-15257-3_26

[56] Yunxin Jeff Li. 2010. An overview of the DSRC/WAVE technology. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Springer, 544–558.

[57] Markets and Markets. 2017. Electronic Toll Collection Market Study. https://www.marketsandmarkets.com/Market-Reports/electronic-toll-collection-system-market-224492059.html. (2017). [Online; accessed April-19-2018].

[58] Sarah Meiklejohn, Keaton Mowery, Stephen Checkoway, and Hovav Shacham. 2011. The Phantom Tollbooth: PrivacyPreserving Electronic Toll Collection in the Presence of Driver Collusion. In *Proceedings of the 20th USENIX Security Symposium*.

[59] Dustin Moody, Rene C. Peralta, Ray A. Perlner, Andrew R. Regenscheid, Allen L. Roginsky, and Lidong Chen. 2015. Report on Pairing-based Cryptography. In *Journal of Research of the National Institute of Standards and Technology*, Vol. 120. National Insititute of Standards and Technology, Gaithersburg, MD, USA, 11–27.

[60] ABC News. 2008. Toll Records Catch Unfaithful Spouses. https://abcnews.go.com/Technology/story?id=3468712&page=1. (2008). [Online; accessed April-19-2018].

[61] European Parliament. 2014. Technology Options for the European Electronic Toll Service. http://www.europarl.europa.eu/RegData/etudes/STUD/2014/529058/IPOL_STUD(2014)529058_EN.pdf. (2014). [Online; accessed April-19-2018].

[62] Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. 2009. VPriv: Protecting Privacy in Location-Based Vehicular Services. In *Proceedings of the 18th USENIX Security Symposium*. 335–350.

Table 2. Information an adversary learns about honest users.

Protocol	**Leakage**					
	$\mathsf{pk}_{\mathcal{U}}$	$\mathbf{a}_{\mathcal{U}}$	$\mathbf{a}_{\mathcal{R}}/\mathbf{a}_{\mathcal{T}}$	$\mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}$	p	b^{bill}
User Registration	●					
Wallet Issuing	●	●	●			
Debt Accumulation		●	●	●	●	
Debt Clearance	●	(●)		●		●
Prove Participation	●					

[63] Savari.net. 2017. MobiWAVE On-Board-Unit (OBU). http://savari.net/wp-content/uploads/2017/05/MW-1000_April2017.pdf. (2017). [Online; accessed 05-February-2018].

[64] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 5 (2002), 557–570. https://doi.org/10.1142/S0218488502001648

[65] American Civil Liberties Union. 2015. Toll Records Catch Unfaithful Spouses. https://www.aclu.org/blog/privacy-technology/location-tracking/newly-obtained-records-reveal-extensive-monitoring-e-zpass. (2015). [Online; accessed April-19-2018].

A INFORMATION LEAKAGE AND DISCUSSION ON PRIVACY IMPLICATIONS

As briefly discussed in Section 1.5.2, 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 $\mathcal{F}_{\mathrm{P4TC}}$, 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 2 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. In all tasks except Debt Accumulation the user's public key $\mathsf{pk}_{\mathcal{U}}$ is leaked. The variables $\mathbf{a}_{\mathcal{U}}$, $\mathbf{a}_{\mathcal{R}}$, $\mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}$ 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 tasks (cp. Table 2).

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 track, 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 would like to elaborate a bit on the general aspects of the question, how a user can be linked to a full track. 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 a users. Terminal nodes represent Debt Clearance transactions and are also linked to users and final balances b^{bill}.

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 equal $\mathbf{a}_{\mathcal{R}}$ 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: Let $\{u_i\}_{i\in\{1,\dots,n\}}$ be a finite set of knapsacks (users) with volume $b_i := b(u_i)$ (total balance). Further let $\{t_j\}_{j\in\{1,\dots,m\}}$ be a finite set of items (transactions) with volumes $p_j := p(t_j)$ (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

$$\sum_{j=1}^{m}\sum_{i=1}^{n} x_{ij} v(t_j) \tag{1}$$

under the constraints

$$\sum_{j=1}^{m} x_{ij} p_j \leq b_i \text{ for } i \in \{1, \dots, m\} \tag{2}$$

and

$$\sum_{i=1}^{n} x_{ij} \leq 1 \text{ for } j \in \{1, \dots, m\} \tag{3}$$

Here, v denotes a value function that assigns a benefit to each item. In our case, we set $v = \mathit{const}$, i.e., each item (transaction) is equally valuable. Informally, this means to pack as many items (transactions) into knapsacks (invoices) without exceeding the volume (balance) of each knapsack (invoice) or assigning an item to more than one knapsack. Since we know that all knapsacks can be filled exactly without any item being left over, we can conclude that any solution that does not completely use a knapsack up to its limit must omit an item. Hence, such a solution is never optimal and equality holds for all optimal solutions in Eq. (2).

Two remarks are in order. With the above explanation, we map an instance of our track-finding problem onto an instance of the KNAPSACK problem, i.e., this only shows that our problem at hand is not harder than 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 track-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 track-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 [4]. 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. We take the German Toll Collect[17] for trucks and analyze the statistics from January, 2018 [34], as a more concrete example. This system uses 24 distinct[18] 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.[19] Within this timeslot each RSU is passed by 28.4 other trucks. Picking a fixed RSU and a fixed timeslot, this implies that the probability that at least one other truck with identical attributes passes by equals

$$p = 1 - \left(\frac{23}{24}\right)^{28.4} \approx 0.7 \tag{4}$$

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 $0.7 \cdot 613 = 429$ 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 time slot. 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 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 A.1. 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

[17] https://www.toll-collect.de/en/

[18] The actual system uses more attribute values (i.e., 720) for statistical purposes. However, we only counted attribute values that the price depends on.

[19] 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 [54]. Additionally, we assumed 80 $\frac{\text{km}}{\text{h}}$ travel speed for trucks.

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 [64] 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.

B FULL SYSTEM DEFINITION

In this appendix we give a detailed description and explanation of our ideal privacy-preserving electronic toll collection functionality $\mathcal{F}_{\text{P4TC}}$. As explained before we define this 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 $\mathcal{F}_{\text{P4TC}}$ before we go on to describe its behavior in a modular way by explaining each task[20] it provides.

The main feature of $\mathcal{F}_{\text{P4TC}}$ is that it keeps track of all conducted transactions in a global transaction database *TRDB* (see Fig. 4). 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

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

It contains the identities $pid_{\mathcal{U}}$ and $pid_{\mathcal{R}}$ of the involved user and RSU (or TSP 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 $trdb^{\text{prev}}$ (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 $x = (x^{\text{prev}} + 1)$ 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 $\mathcal{F}_{\text{P4TC}}$. Full transaction entries *trdb* are only created by instances of Debt Accumulation. Both Wallet Issuing and Debt Clearance create stubs of the form

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

respectively. 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, $\mathcal{F}_{\text{P4TC}}$ stores three more partially defined mappings: $f_{\mathcal{A}_{\mathcal{U}}} : \mathcal{L} \rightarrow \mathcal{A}_{\mathcal{U}}$ and $f_{\mathcal{A}_{\mathcal{R}}} : \mathcal{PID}_{\mathcal{R}} \rightarrow \mathcal{A}_{\mathcal{R}}$ of user and RSU attribute vectors as well as f_{Π} of proofs of guilt that have been issued or queried in the context of double-spending detection.

The ideal function $\mathcal{F}_{\text{P4TC}}$ provides twelve different tasks in total which we divide up into three categories: "System Setup Tasks" (comprising all Registrations and RSU Certification), "Basic Tasks" (Wallet Issuing, Debt Accumulation and Debt Clearance), and "Feature Tasks" (Prove Participation, Double-Spending Detection, Guilt Verification and User Blacklisting).

[20] 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.

Functionality $\mathcal{F}_{\text{P4TC}}$

I. State

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

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

- 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:

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

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

$$f_{\mathcal{A}_{\mathcal{R}}} : \mathcal{PID}_{\mathcal{R}} \to \mathcal{A}_{\mathcal{R}}, \mathit{pid}_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}$$

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

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

II. Behavior

- *DR Registration* (Fig. 5)
- *TSP Registration* (Fig. 6)
- *RSU Registration* (Fig. 7)
- *User Registration* (Fig. 8)
- *RSU Certification* (Fig. 9)
- *Wallet Issuing* (Fig. 10)
- *Debt Accumulation* (Fig. 11)
- *Debt Clearance* (Fig. 12)
- *Prove Participation* (Fig. 13)
- *Double-Spending Detection* (Fig. 14)
- *Guilt Verification* (Fig. 15)
- *User Blacklisting* (Fig. 16)

Fig. 4. The functionality $\mathcal{F}_{\text{P4TC}}$

Table 3. Notation that only occurs in the ideal functionality

Identifier	**Description**
$\mathcal{PID}_{\text{corrupt}}$	set of corrupted party identifiers
f_Φ	(partial) mapping giving the fraud detection ID ϕ corresponding to given wallet ID λ and counter x
$f_{\mathcal{A}_{\mathcal{U}}}$	(partial) mapping assigning user attributes to a given wallet ID λ
$f_{\mathcal{A}_{\mathcal{R}}}$	(partial) mapping $f_{\mathcal{A}_{\mathcal{R}}}$ assigning RSU attributes to a given RSU PID $\mathit{pid}_{\mathcal{R}}$

For better clarity of the following task descriptions, an overview of the variables used can be found in Tables 3 and 4.

B.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 $\overline{\mathcal{G}}_{\text{bb}}$ to be able to participate in the toll collection system. As all of these registration

Table 4. Notation that occurs in the ideal functionality and in the real protocol

Identifier	**Abstract Domain**	**Instantiation**	**Description**
pid_{DR}	$\mathcal{PID}_{DR}$	$\{0,1\}^*$	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}	$\mathcal{PK}_{DR}$	$G_1^3 \times G_2^3 \times (G_1^2)^{\ell+2} \times (G_2^2)^4 \times (G_2^2)^{\ell+2}$	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
$\mathsf{pk}_{\mathcal{R}}$	$\mathcal{PK}_{\mathcal{R}}$	$G_1^3 \times G_2$	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	$\mathbb{Z}_\mathfrak{p}$	$\mathbb{Z}_\mathfrak{p}$	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	$\mathcal{L}$	$\mathbb{Z}_\mathfrak{p}$	wallet ID; is used as PRF seed
x	$\mathbb{N}_0$	$\{0,\ldots,n_{\mathrm{PRF}}\}$	(PRF) counter
$bl_{\mathcal{R}}$	list of Φ elements	list of G_1 elements	RSU blacklist
$x_{\mathrm{bl}_{\mathcal{R}}}$	$\mathbb{N}$	$\mathbb{N}$	RSU blacklist parameter
$bl_{\mathcal{T}}$	list of $\mathcal{PK}_{\mathcal{U}}$ elements	list of G_1 elements	TSP blacklist

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.

B.1.1 Registrations. The tasks of DR, RSU and User Registration (cp. Figs. 5 to 8) 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 $\overline{\mathcal{G}}_{\mathrm{bb}}$ and output to the newly registered party.

The registration of the TSP is slightly different (cp. Fig. 6). 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 leaked to the adversary together with $pid_{\mathcal{T}}$ when the public key $\mathsf{pk}_{\mathcal{T}}$ 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 $\mathsf{pk}_{\mathcal{T}}$ is output to the TSP.

B.1.2 RSU Certification. RSU certification (cp. Fig. 9) is a two-party task between the TSP and an RSU in which

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *DR Registration*

DR input: (`register`)

(1) If this task has been run before, output $\perp$ and abort.
(2) Send $(\texttt{registering\_dr}, pid_{DR})$ to the adversary and obtain the key (pk_{DR}).[a]
(3) Call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{DR})$.

DR output: (pk_{DR})

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

Fig. 5. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *TSP Registration*

TSP input: $(\texttt{register}, \mathsf{a}_{\mathcal{T}})$

(1) If this task has been run before, output $\perp$ and abort.
(2) $TRDB := \emptyset$
(3) Send $(\texttt{registering\_tsp}, pid_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}})$ to the adversary and obtain the key $(\mathsf{pk}_{\mathcal{T}})$.[a]
(4) Call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{T}})$.

TSP output: $(\mathsf{pk}_{\mathcal{T}})$

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

Fig. 6. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *RSU Registration*

RSU input: (`register`)

(1) If this task has been run before, output $\perp$ and abort.
(2) Send $(\texttt{registering\_rsu}, pid_{\mathcal{R}})$ to the adversary and obtain the key $(\mathsf{pk}_{\mathcal{R}})$.[a]
(3) Call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{R}})$.

RSU output: $(\mathsf{pk}_{\mathcal{R}})$

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

Fig. 7. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

the RSU is assigned an attribute vector $\mathbf{a}_{\mathcal{R}}$.[21] The content of attribute vectors is in no way restricted by $\mathcal{F}_{\text{P4TC}}$ 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. This $\mathbf{a}_{\mathcal{R}}$ is input by the TSP, while the RSU only inputs

[21] 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 $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *User Registration*

User input: $(\mathtt{register})$

(1) If this task has been run before, output $\perp$ and abort.
(2) Send $(\mathtt{registering\_user}, \mathit{pid}_{\mathcal{U}})$ to the adversary and obtain the key $(\mathrm{pk}_{\mathcal{U}})$.[a]
(3) Call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\mathtt{register}, \mathrm{pk}_{\mathcal{U}})$.

User output: $(\mathrm{pk}_{\mathcal{U}})$

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

Fig. 8. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *RSU Certification*

RSU input: $(\mathtt{certify})$

TSP input: $(\mathtt{certify}, \mathbf{a}_{\mathcal{R}})$

(1) If $f_{\mathcal{A}_{\mathcal{R}}}(\mathit{pid}_{\mathcal{R}})$ is already defined, output $\perp$ to both parties and abort; else append $f_{\mathcal{A}_{\mathcal{R}}}(\mathit{pid}_{\mathcal{R}}) := \mathbf{a}_{\mathcal{R}}$ to $f_{\mathcal{A}_{\mathcal{R}}}$.
(2) Leak $(\mathtt{certifying\_rsu}, \mathit{pid}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$ to the adversary.

RSU output: $(\mathbf{a}_{\mathcal{R}})$

TSP output: $(\mathtt{OK})$

Fig. 9. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

its desire to be certified. $\mathcal{F}_{\text{P4TC}}$ checks if there have already been attributes assigned to the RSU previously (in which case it aborts) and otherwise appends $f_{\mathcal{A}_{\mathcal{R}}}(\mathit{pid}_{\mathcal{R}}) := \mathbf{a}_{\mathcal{R}}$ to the partial mapping $f_{\mathcal{A}_{\mathcal{R}}}$ which internally stores all RSU attributes already assigned. The identity $\mathit{pid}_{\mathcal{R}}$ and attributes $\mathbf{a}_{\mathcal{R}}$ are leaked to the adversary before the attributes are output to the RSU.

B.2 Basic Tasks

In this section 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.

B.2.1 Wallet Issuing. Wallet Issuing (cp. Fig. 10) 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 $\mathit{bl}_{\mathcal{T}}$ of user public keys that are not allowed to obtain any new wallets. 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 $\mathit{pid}_{\mathcal{U}}$ 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 $\mathit{pid}_{\mathcal{U}}$ is obtained from the bulletin board $\overline{\mathcal{G}}_{\text{bb}}$ and checked against the TSP's blacklist $\mathit{bl}_{\mathcal{T}}$. 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

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *Wallet Issuing*

User input: $(\texttt{issue})$

TSP input: $(\texttt{issue}, \mathbf{a}_{\mathcal{U}}, bl_{\mathcal{T}})$

(1) Pick serial number $s \overset{\text{R}}{\leftarrow} S$ that has not previously been used.
(2) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ 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 $\mathrm{pk}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.[⊥]
(4) If $\mathrm{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$, output $\texttt{blacklisted}$ to both parties and abort.
(5) Pick wallet ID $\lambda \overset{\text{R}}{\leftarrow} \mathcal{L}$ that has not previously been used.
(6) If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\phi \overset{\text{R}}{\leftarrow} \Phi$ 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 $trdb := (\bot, s, \phi, 0, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, 0, 0)$ to $TRDB$
(8) Append $f_{\mathcal{A}_{\mathcal{U}}}(\lambda) := \mathbf{a}_{\mathcal{U}}$ to $f_{\mathcal{A}_{\mathcal{U}}}$.

User output: $(s, \mathbf{a}_{\mathcal{U}})$

TSP output: (s)

[⊥] If this does not exist, output $\bot$ 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.

Fig. 10. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

transaction entry

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

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.

B.2.2 Debt Accumulation. This two-party task (cp. Fig. 11) 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 he wishes to use 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 him. Multiple corrupted users may choose to swap wallets between them but are not able to use an honest user's wallet. From the previous wallet state

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

the ideal function gets a fraud detection ID ϕ^{prev} that is checked against $bl_{\mathcal{R}}$. The other information in $trdb^{\text{prev}}$ is then used 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

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *Debt Accumulation*

User input: $(\texttt{pay\_toll}, s^{\text{prev}})$
RSU input: $(\texttt{pay\_toll}, bl_{\mathcal{R}})$

(1) Pick serial number $s \overset{\text{R}}{\leftarrow} S$ that has not previously been used.
(2) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ ask the adversary, if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
(3) Select $(\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in \mathit{TRDB}$ (with s^{prev}, $pid_{\mathcal{U}}$ being the uniqe key)$^{\perp}$.
(4) If $\phi^{\text{prev}} \in bl_{\mathcal{R}}$, output `blacklisted` to both parties and abort.
(5) Set $\lambda := \lambda^{\text{prev}}$ and $x := x^{\text{prev}} + 1$.
(6) If $f_{\Phi}(\lambda, x)$ is already defined, set $\phi := f_{\Phi}(\lambda, x)$.
Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\phi \overset{\text{R}}{\leftarrow} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID ϕ that has not previously been used.[b] Append $f_{\Phi}(\lambda, x) := \phi$ to f_{Φ}.
(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.

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

$^{\perp}$If this does not exist, 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] The ideal model only guarantees privacy for honest users. For corrupted users the fraud detection ID might be chosen adversarially (cp. text body).

Fig. 11. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

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 attributes $\mathbf{a}_{\mathcal{U}}$, $\mathbf{a}_{\mathcal{R}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ are again looked up internally and leaked 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 before *trdb* is 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 he may check he only paid the amount he expected. 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.

B.2.3 Debt Clearance. As Debt Clearance (cp. Fig. 12) 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 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

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *Debt Clearance*

User input: $(\texttt{clear\_debt}, s^{\text{prev}})$
TSP input: $(\texttt{clear\_debt})$

(1) Pick serial number $s \xleftarrow{\text{R}} S$ that has not previously been used.
(2) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ ask the adversary, if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
(3) Receive $\mathsf{pk}_{\mathcal{U}}$ from the bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.$^{\perp}$
(4) Select $(\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in \mathit{TRDB}$ (with $s^{\text{prev}}, pid_{\mathcal{U}}$ 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, set $\phi := f_{\Phi}(\lambda, x)$.
Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\phi \xleftarrow{\text{R}} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID ϕ that has not previously been used.[b] Append $f_{\Phi}(\lambda, x) := \phi$ to f_{Φ}.
(7) If $pid_{\mathcal{T}} \in \mathcal{PID}_{\text{corrupt}}$, 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*.

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

$^{\perp}$If this does not exist, 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] The ideal model only guarantees privacy for honest users. For corrupted users the fraud detection ID might be chosen adversarially (cp. text body).

Fig. 12. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

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

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

is added to *TRDB* and the bill 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.

B.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 User Blacklisting. All of those tasks deal with different aspects arising from fraudulent user behavior.

B.3.1 Prove Participation. This is a two-party task involving a user and the SA (cp. Fig. 13) 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 his successful participation in a transaction with the RSU where the photo was taken, while the fraudulent user will not be able to do so. The SA inputs the public key $\mathsf{pk}_{\mathcal{U}}$ of the user which the SA wishes to prove its participation and a set $S_{\mathcal{R}}^{\text{pp}}$ of serial numbers in question. The user has no explicit input but simply expresses its consent by running the protocol.

Functionality $\mathcal{F}_{\mathrm{P4TC}}$ (cont.) – Task *Prove Participation*

User input: $(\texttt{prove\_participation})$
SA input: $(\texttt{prove\_participation}, \mathsf{pk}_{\mathcal{U}}, S^{\mathrm{pp}}_{\mathcal{R}})$

(1) Obtain $pid'_{\mathcal{U}}$ for $\mathsf{pk}_{\mathcal{U}}$ from $\bar{\mathcal{G}}_{\mathrm{bb}}$.[a]
(2) If $(pid'_{\mathcal{U}} \notin \mathcal{PID}_{\mathrm{corrupt}} \lor pid_{\mathcal{U}} \notin \mathcal{PID}_{\mathrm{corrupt}}) \land pid'_{\mathcal{U}} \neq pid_{\mathcal{U}}$, abort.
(3) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\mathrm{corrupt}}$ leak $S^{\mathrm{pp}}_{\mathcal{R}}$ to the adversary.
(4) If $\exists\ (\cdot, s, \cdot, \cdot, \cdot, pid'_{\mathcal{U}}, \cdot, \cdot, \cdot) \in \mathit{TRDB}$ such that $s \in S^{\mathrm{pp}}_{\mathcal{R}}$,
then $\mathsf{out}_{\mathcal{U}} := \mathsf{out}_{SA} := \mathsf{OK}$
else $\mathsf{out}_{\mathcal{U}} := \mathsf{out}_{SA} := \mathsf{NOK}$.

User output: $(\mathsf{out}_{\mathcal{U}})$
SA output: (out_{SA})

[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.

Fig. 13. The functionality $\mathcal{F}_{\mathrm{P4TC}}$ (cont. from Fig. 4)

First note, there is no guarantee that the user which participates in the protocol (with PID $pid_{\mathcal{U}}$) is the user the SA wants to prove its 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 $S^{\mathrm{pp}}_{\mathcal{R}}$. 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 its transaction information to another corrupted user (with $pid_{\mathcal{U}}$) which 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 itself.

If the user is corrupted, the set of serial numbers in question is leaked 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 $S^{\mathrm{pp}}_{\mathcal{R}}$ or not, this single bit of information is sufficient to restore the complete serial 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.

B.3.2 Double-Spending Detection and Guilt Verification. Due to our requirement to allow offline RSUs, a user is able to fraudulently collect debt on outdated states of his wallet. 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. 14) and Guilt Verification (cp. Fig. 15).

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

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *Double-Spending Detection*

TSP input: $(\texttt{scan\_for\_fraud}, \phi)$

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

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

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

Fig. 14. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

Functionality $\mathcal{F}_{\text{P4TC}}$ (cont.) – Task *Guilt Verification*

Party input: $(\texttt{verify\_guilt}, \mathsf{pk}_{\mathcal{U}}, \pi)$

(1) Receive $pid_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for key $\mathsf{pk}_{\mathcal{U}}$.$^{\perp}$
(2) If $f_{\Pi}(pid_{\mathcal{U}}, \pi)$ is defined, then set $\texttt{out} := f_{\Pi}(pid_{\mathcal{U}}, \pi)$ and output $(\texttt{out})$.
(3) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$, then leak $(pid_{\mathcal{U}}, \pi)$ to the advesary and obtain result $\texttt{out}$, else set $\texttt{out} := \texttt{NOK}$.
(4) Append $(pid_{\mathcal{U}}, \pi) \mapsto \texttt{out}$ to f_{Π}.

Party output: $(\texttt{out})$

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

Fig. 15. The functionality $\mathcal{F}_{\text{P4TC}}$ (cont. from Fig. 4)

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.

B.3.3 User Blacklisting. User Blacklisting (cp. Fig. 16) 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 $\mathcal{F}_{\text{P4TC}}$.

Functionality $\mathcal{F}_{\mathrm{P4TC}}$ (cont.) – Task *User Blacklisting*

DR input: $(\texttt{blacklist\_user}, \mathrm{pk}_{\mathcal{U}}^{\mathcal{DR}})$

TSP input: $(\texttt{blacklist\_user}, \mathrm{pk}_{\mathcal{U}}^{\mathcal{T}})$

(1) If $\mathrm{pk}_{\mathcal{U}}^{\mathcal{DR}} \neq \mathrm{pk}_{\mathcal{U}}^{\mathcal{T}}$, abort.

(2) Receive $\mathit{pid}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\mathrm{bb}}$ for $\mathrm{pk}_{\mathcal{U}}^{\mathcal{T}}$.$^{\perp}$

(3) Distinguish 3 cases:

  (a) $\mathit{pid}_{\mathcal{T}} \notin \mathcal{PID}_{\mathrm{corrupt}}$: Set $\mathcal{L}_{\mathrm{bl}} := \{\lambda \mid (\cdot,\cdot,\cdot,\cdot,\lambda,\mathit{pid}_{\mathcal{U}},\cdot,\cdot,\cdot) \in \mathit{TRDB}\}$.

  (b) $\mathit{pid}_{\mathcal{T}} \in \mathcal{PID}_{\mathrm{corrupt}}$, $\mathit{pid}_{\mathcal{U}} \notin \mathcal{PID}_{\mathrm{corrupt}}$: Obtain a set of serial numbers S_{root} from the adversary and set $\mathcal{L}_{\mathrm{bl}} := \{\lambda \mid (\perp, s, \cdot,\cdot,\lambda,\mathit{pid}_{\mathcal{U}},\cdot,\cdot,\cdot) \in \mathit{TRDB}$ with $s \in S_{\mathrm{root}}\}$.

  (c) $\mathit{pid}_{\mathcal{T}} \in \mathcal{PID}_{\mathrm{corrupt}}$, $\mathit{pid}_{\mathcal{U}} \in \mathcal{PID}_{\mathrm{corrupt}}$: Let the adversary decide on the output for DR and TSP and stop.

(4) $\mathit{TRDB}_{\mathrm{bl}} := \{\mathit{trdb} \in \mathit{TRDB} \mid \mathit{trdb} = (\cdot,\cdot,\cdot,\cdot,\lambda,\cdot,\cdot,p,\cdot) \text{ s. t. } \lambda \in \mathcal{L}_{\mathrm{bl}}\}$

(5) $b^{\mathrm{bill}} := \sum_{\mathit{trdb} \in \mathit{TRDB}_{\mathrm{bl}}} p$.

(6) For each $\lambda \in \mathcal{L}_{\mathrm{bl}}$:

  (a) $x_\lambda := \max\{\, x \mid f_\Phi(\lambda, x) \text{ is already defined}\}$.

  (b) For $x \in \{x_\lambda + 1, \ldots, x_{\mathrm{bl}_{\mathcal{R}}}\}$:

    (i) If $\mathit{pid}_{\mathcal{U}} \notin \mathcal{PID}_{\mathrm{corrupt}}$, pick $\phi \overset{\mathrm{R}}{\leftarrow} \Phi$ 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) $\Phi_{\mathrm{bl}} := \{f_\Phi(\lambda, x) \mid \lambda \in \mathcal{L}_{\mathrm{bl}}, 0 \leq x \leq x_{\mathrm{bl}_{\mathcal{R}}}\}$.

DR output: $(\texttt{OK})$

TSP output: $(b^{\mathrm{bill}}, \Phi_{\mathrm{bl}})$

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

Fig. 16. The functionality $\mathcal{F}_{\mathrm{P4TC}}$ (cont. from Fig. 4)

Both parties—the TSP and the DR—input the public key ($\mathrm{pk}_{\mathcal{U}}^{\mathcal{DR}}$ 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 (cp. Fig. 16, Step 3a) for an honest TSP. (N.b.: The DR is assumed to be always honest.) To calculate the user's outstanding debt, all transaction entries in *TRDB* containing $\mathit{pid}_{\mathcal{U}}$ 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 $\mathit{pid}_{\mathcal{U}}$ 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 (cp. Fig. 16, Steps 3b and 3c). 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.

C SETTING AND BUILDING BLOCKS

In this appendix we introduce the algebraic setting and building blocks we make use of. In particular, the latter includes non-interactive zero-knowledge proofs, commitments, signatures, encryption and pseudo-random functions. We also describe possible instantiations for these building blocks and explain how these primitives are used in our system.

C.1 Algebraic Setting and Assumptions

Our protocol instantiations are based on an asymmetric bilinear group setting $\mathsf{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2)$. We adopt the following definition from [42].

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

$$\mathsf{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^n) \tag{5}$$

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

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

The setting is called *asymmetric*, if no efficiently computable homomorphisms between G_1 and G_2 are known. In the remainder of this paper, we consider the asymmetric kind.

Our construction relies on the co-CDH assumption for identification, and the security of our building blocks (cp. Appendix C.2) in asymmetric bilinear groups. For our special instantiation of the building blocks (see there), 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 C.2 (DDH and SXDH Assumption).

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

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^n) \\ x, y, z \leftarrow \mathbb{Z}_{\mathfrak{p}}; h_0 := g_i^{xy}; h_1 := g_i^z \\ b \overset{\mathrm{R}}{\leftarrow} \{0, 1\} \\ b' \leftarrow \mathcal{A}(1^n, \mathsf{gp}, g_i^x, g_i^y, h_b) \end{array} \right] - \frac{1}{2} \right|$$

is a negligible function in n 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 C.3 (Co-CDH assumption). We say that the *co-CDH assumption* holds with respect to SetupGrp if the advantage $\mathrm{Adv}^{\text{CO-CDH}}_{\text{SetupGrp},\mathcal{A}}(1^n)$ defined by

$$\Pr\left[\, a = g_2^x \;\middle|\; \begin{array}{c} \mathrm{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathrm{SetupGrp}(1^n) \\ x \leftarrow \mathbb{Z}_{\mathfrak{p}} \\ a \leftarrow \mathcal{A}(1^n, \mathrm{gp}, g_1^x) \end{array} \right]$$

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

The co-DLIN assumption is defined as follows:

Definition C.4. We say that the *co-DLIN assumption* holds with respect to SetupGrp if the advantage $\mathrm{Adv}^{\text{CO-DLIN}}_{\text{SetupGrp},\mathcal{A}}(1^n)$ defined by

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} \mathrm{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathrm{SetupGrp}(1^n) \\ \alpha, \beta, \gamma \overset{R}{\leftarrow} \mathbb{Z}_{\mathfrak{p}} \\ b \overset{R}{\leftarrow} \{0, 1\} \\ \check{h}_1 := g_1^{\alpha},\, \check{h}_2 := g_1^{\beta},\, \check{h}_3 := g_1^{\alpha+\beta+b\gamma} \\ \hat{h}_1 := g_2^{\alpha},\, \hat{h}_2 := g_2^{\beta},\, \hat{h}_3 := g_2^{\alpha+\beta+b\gamma} \\ b' \leftarrow \mathcal{A}(1^n, \mathrm{gp}, \check{h}_1, \check{h}_2, \check{h}_3, \hat{h}_1, \hat{h}_2, \hat{h}_3) \end{array} \right]$$

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

C.2 Cryptographic Building Blocks

Our semi-generic construction makes use of various cryptographic primitives including (F_{gp}-extractable) NIZK proofs, equivocal and extractable homomorphic commitments, digital signatures, public-key encryption, symmetric encryption and pseudo-random functions. The latter building blocks need to be efficiently and securely combinable with the chosen NIZK proof system, which is Groth-Sahai (GS) in our case. In the following, we give an introduction to the formal definition of these building blocks.

C.2.1 Group setup. Let SetupGrp be a bilinear group generator (cp. Definition C.1) that outputs descriptions of asymmetric bilinear groups gp ← SetupGrp(1^n). The following building blocks all make use of SetupGrp as their common group setup algorithm.

C.2.2 NIZKs. Let R be a witness relation for some NP language $L = \{stmnt \mid \exists\, wit \text{ s.t. } (stmnt, wit) \in R\}$. A zero-knowledge proof system allows a prover P to convince a verifier V that some *stmnt* is contained in L without V learning anything beyond that fact. In a non-interactive zero-knowledge (NIZK) proof, only one message, the proof π, is sent from P to V for that purpose.

More precisely, a (group-based) NIZK proof system is defined as:

Definition C.5 (Group-based NIZK proof system). Let R be an efficiently verifiable relation containing triples (gp, x, w). We call gp the group setup, x the statement, and w the witness. Given some gp, let L_{gp} be the language containing all statements x such that $(\mathrm{gp}, x, w) \in R$. Let POK := (SetupGrp, SetupPoK, Prove, Vfy) be a tuple of PPT algorithms such that

- SetupGrp takes as input a security parameter 1^n and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms.
- SetupPoK takes as input gp and outputs a (public) common reference string $\mathrm{CRS}_{\mathrm{pok}}$.
- Prove takes as input the common reference string $\mathrm{CRS}_{\mathrm{pok}}$, a statement x, and a witness w with $(\mathrm{gp}, x, w) \in R$ and outputs a proof π.

- Vfy takes as input the common reference string $\mathsf{CRS}_{\mathsf{pok}}$, a statement x, and a proof π and outputs 1 or 0.

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

(1) *Perfect completeness:* For all $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n)$, $\mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}(\mathsf{gp})$, $(\mathsf{gp}, x, w) \in R$, and $\pi \leftarrow \mathsf{Prove}(\mathsf{CRS}_{\mathsf{pok}}, x, w)$ we have that $\mathsf{Vfy}(\mathsf{CRS}_{\mathsf{pok}}, x, \pi) = 1$.

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

$$\Pr\left[\ \mathsf{Vfy}(\mathsf{CRS}_{\mathsf{pok}}, x, \pi) = 0 \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ \mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}(\mathsf{gp}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{pok}}) \\ x \notin L_{\mathsf{gp}} \end{array} \right]$$

is 1.

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

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

$$\left| \begin{array}{l} \Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{pok}}) \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ \mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}(\mathsf{gp}) \end{array} \right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{CRS}'_{\mathsf{pok}}) \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ (\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{epok}}) \leftarrow \mathsf{SetupEPoK}(\mathsf{gp}) \end{array} \right] \end{array} \right|$$

is zero.

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

$$\Pr\left[\ \exists\, w :\ F_{\mathsf{gp}}(w) = W \wedge\ (\mathsf{gp}, x, w) \in R \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ (\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{epok}}) \leftarrow \mathsf{SetupEPoK}(\mathsf{gp}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{epok}}) \\ 1 \leftarrow \mathsf{Vfy}(\mathsf{CRS}'_{\mathsf{pok}}, x, \pi) \\ W \leftarrow \mathsf{ExtractW}(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{epok}}, x, \pi) \end{array} \right]$$

is 1.

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

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

$$\left| \begin{array}{l} \Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{pok}}) \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ \mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}(\mathsf{gp}) \end{array} \right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}(\mathsf{CRS}'_{\mathsf{pok}}) \ \middle|\ \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ \mathit{hint} \leftarrow \mathsf{GenHint}(\mathsf{gp}), \\ (\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}) \leftarrow \mathsf{SetupSPoK}(\mathsf{gp}, \mathit{hint}), \end{array} \right] \end{array} \right|$$

is negligible in n.

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

$$\left| \begin{array}{l} \Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathsf{SimProof}'(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}, \cdot, \cdot)}(1^n, \mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}})\ \right] \\ -\Pr\left[\ 1 \leftarrow \mathcal{A}^{\mathsf{Prove}(\mathsf{CRS}'_{\mathsf{pok}}, \cdot, \cdot)}(1^n, \mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}})\ \right] \end{array} \right|$$

is negligible in n, where $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n)$, $(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}) \leftarrow \mathsf{SetupSPoK}(\mathsf{gp})$, and $\mathsf{SimProof}'(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}, \cdot, \cdot)$ is an oracle which on input $(x, z) \in R$ returns $\mathsf{SimProof}(\mathsf{CRS}'_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}, x)$. Both $\mathsf{SimProof}'$ and Prove return $\perp$ on input $(x, z) \notin R$.

We wish to point out some remarks.

Remark C.6. (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 x which verifies, violating soundness, then obviously there is no witness w for x, which violates extractability.

(3) Extractability essentially means that ExtractW—given a trapdoor $\mathsf{td}_{\mathsf{epok}}$—is able to extract $F_{\mathsf{gp}}(\mathit{wit})$ for an NP-witness wit for $\mathit{stmnt} \in L_{\mathsf{gp}}$ 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*.

Our Instantiation. We choose the SXDH-based Groth-Sahai proof system [36, 39] 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, it is not always zero-knowledge, and F_{gp} is not always the identity function. For the sake of completeness Appendix C.3 contains a description what types of equations are supported by GS. 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 $F^{(1)}_{\mathsf{gp}}$, $F^{(2)}_{\mathsf{gp}}$ and $F^{(3)}_{\mathsf{gp}}$ depend on the considered languages L_1, L_2 and L_3 (defined in Appendices D.6 to D.8) but they have the following in common: They behave as the identity function with respect to group elements and map elements from $\mathbb{Z}_\mathfrak{p}$ 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.

All proof systems will share a common reference string. More precisely, we demand that there is a shared extraction setup algorithm which generates the CRS and also a single extraction trapdoor for P1, P2 and P3. Let us denote this algorithm by SetupEPoK and its output by $(\mathsf{CRS}_{\mathsf{pok}}, \mathsf{td}_{\mathsf{epok}}) \leftarrow \mathsf{SetupEPoK}(\mathsf{gp})$ in the following. Furthermore, let us denote the prove and verify algorithms of these proof systems by $\mathsf{P}X.\mathsf{Prove}$ and $\mathsf{P}X.\mathsf{Vfy}$, for $1 \leq X \leq 3$.

Range Proofs. For one particular task[22] we need range proofs in order to show that some $\mathbb{Z}_\mathfrak{p}$-element λ'_i is "smaller" than some fixed system parameter $\mathcal{B}$ with both elements being regarded as elements from $\{0, \ldots, \mathfrak{p} - 1\}$ and the normal $\leq$-relation from the integers. We realize these range proofs using Groth-Sahai by applying the signature-based technique in [15]. 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 in [1].

C.2.3 *Commitments.*

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

[22] More precisely: The task Wallet Issuing

Definition C.7. A *commitment scheme* COM := (SetupGrp, Gen, Com, Open) consists of four algorithms:

- SetupGrp takes as input a security parameter 1^n and outputs public parameters gp. These parameters also define a message space $\mathcal{M}$, an implicit message space $\mathcal{M}'$ and a function $F_{\mathrm{gp}} : \mathcal{M} \to \mathcal{M}'$ mapping a message to its implicit representation. We assume that gp is given as implicit input to all algorithms.
- Gen is a PPT algorithm, which takes gp as input and outputs public parameters $\mathsf{CRS}_{\mathsf{com}}$.
- Com is a PPT algorithm, which takes as input parameters $\mathsf{CRS}_{\mathsf{com}}$ and a message $m \in \mathcal{M}$ and outputs a commitment c to m and some decommitment value d.
- Open is a deterministic polynomial-time algorithm, which takes as input parameters $\mathsf{CRS}_{\mathsf{com}}$, commitment c, an implicit message $M \in \mathcal{M}'$, and opening d. It returns either 0 or 1.

COM is *correct* if for all $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n)$, $\mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{Gen}(\mathsf{gp})$, $m \in \mathcal{M}$, and $(c, d) \leftarrow \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, m)$ it holds that $1 = \mathsf{Open}(\mathsf{CRS}_{\mathsf{com}}, F_{\mathrm{gp}}(m), c, d)$.

We say that COM 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 $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(1^n)$ defined by

$$\left|\Pr\left[\; b = b' \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ \mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (m_0, m_1, \mathit{state}) \leftarrow \mathcal{A}(1^n, \mathsf{CRS}_{\mathsf{com}}) \\ b \overset{\mathrm{R}}{\leftarrow} \{0,1\} \\ (c, d) \leftarrow \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, m_b) \\ b' \leftarrow \mathcal{A}(c, \mathit{state}) \end{array} \right] - \frac{1}{2}\right|$$

is negligible in n. The scheme is called *statistically hiding* if $\mathsf{Adv}^{\mathsf{Hiding}}_{\mathsf{COM},\mathcal{A}}(1^n)$ is negligible even for an unbounded adversary $\mathcal{A}$.

(2) F_{gp}*-Binding:* For all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{F_{\mathrm{gp}}\text{-}\mathsf{Binding}}_{\mathcal{A}}(1^n)$ defined by

$$\Pr\left[\begin{array}{c} \mathsf{Open}(\mathsf{CRS}_{\mathsf{com}}, M, c, d) = 1 \\ \wedge \\ \mathsf{Open}(\mathsf{CRS}_{\mathsf{com}}, M', c, d') = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ \mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (c, M, d, M', d') \leftarrow \mathcal{A}(1^n, \mathsf{CRS}_{\mathsf{com}}) \\ M \neq M' \end{array}\right]$$

is negligible in n.

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

(a) we have that the advantage $\mathsf{Adv}^{\mathsf{SimGen}}_{\mathsf{COM},\mathcal{A}}(n)$ defined by

$$\left|\begin{array}{r} \Pr\left[\; 1 \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{com}}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ \mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{Gen}(\mathsf{gp}) \end{array}\right] \\ - \Pr\left[\; 1 \leftarrow \mathcal{A}(\mathsf{CRS}'_{\mathsf{com}}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ (\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}) \leftarrow \mathsf{SimGen}(\mathsf{gp}) \end{array}\right] \end{array}\right|$$

is negligible in n.

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

$$\left| \Pr\left[1 \leftarrow \mathcal{A}\left(\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}, m, c, d\right) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ (\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}) \leftarrow \mathsf{SimGen}(\mathsf{gp}), \\ m \leftarrow \mathcal{M}, \\ (c, d) \leftarrow \mathsf{Com}(\mathsf{CRS}'_{\mathsf{com}}, m) \end{array} \right] - \Pr\left[1 \leftarrow \mathcal{A}\left(\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}, m, c', d'\right) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ (\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}) \leftarrow \mathsf{SimGen}(\mathsf{gp}), \\ (c', r) \leftarrow \mathsf{SimCom}(\mathsf{gp}), \\ m \leftarrow \mathcal{M}, \\ d' \leftarrow \mathsf{Equiv}(\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}, m, r) \end{array} \right] \right|$$

is zero.

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

(a) we have that the advantage $\mathsf{Adv}^{\mathsf{ExtGen}}_{\mathsf{COM},\mathcal{A}}(n)$ defined by

$$\left| \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{CRS}_{\mathsf{com}}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ \mathsf{CRS}_{\mathsf{com}} \leftarrow \mathsf{Gen}(\mathsf{gp}) \end{array} \right] - \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{CRS}'_{\mathsf{com}}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n), \\ (\mathsf{CRS}'_{\mathsf{com}}, \mathsf{td}_{\mathsf{extcom}}) \leftarrow \mathsf{ExtGen}(\mathsf{gp}) \end{array} \right] \right|$$

is negligible in n.

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

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

is zero.

Furthermore, assume that the message space of COM is an additive group. Then COM is called *additively homomorphic*, if there exist additional PPT algorithms $c \leftarrow \mathsf{CAdd}(\mathsf{CRS}_{\mathsf{com}}, c_1, c_2)$ and $d \leftarrow \mathsf{DAdd}(\mathsf{CRS}_{\mathsf{com}}, d_1, d_2)$ which on input of two commitments and corresponding decommitment values $(c_1, d_1) \leftarrow \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, m_1)$ and $(c_2, d_2) \leftarrow \mathsf{Com}(\mathsf{CRS}_{\mathsf{com}}, m_2)$, output a commitment c and decommitment d, respectively, such that $\mathsf{Open}(\mathsf{CRS}_{\mathsf{com}}, c, F_{\mathsf{gp}}(m_1 + m_2), d) = 1$.

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

Our Instantiation. We will 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. [3]. This commitment scheme has message space $\mathbb{Z}_p^\alpha$, commitment space G_2 and opening value space G_1. It is statistically hiding, additively homomorphic, equivocal, and F'_{gp}-Binding, for $F'_{\mathsf{gp}}(m_1, \ldots, m_\alpha) := (g_1^{m_1}, \ldots, g_1^{m_\alpha})$. We use this commitment scheme as C1 with CRS $\mathsf{CRS}^1_{\mathsf{com}}$ in the following ways in our system:

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

We also use the (dual-mode) equivocal and extractable commitment scheme from Groth and Sahai [39]. This commitment scheme has message space G_1, commitment space G_1^2 and opening value space $\mathbb{Z}_p^2$. It is equivocal, extractable, hiding and F'_{gp}-Binding for $F'_{\mathrm{gp}}(m) := m$. In our system, we use this commitment scheme as C2 with CRS $\mathrm{CRS}^2_{\mathrm{com}}$ in the Wallet Issuing and Debt Accumulation tasks.

C.2.4 Digital signatures. A signature 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 pk of the signer but nobody can feasibly forge a signature without knowing sk.

Definition C.8. A *digital signature scheme* $\mathsf{S} := (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Sgn}, \mathsf{Vfy})$ consists of four PPT algorithms:

- SetupGrp takes as input a security parameter 1^n and outputs public parameters gp. We assume that gp is given as implicit input to all algorithms.
- Gen takes gp as input and outputs a key pair (pk, sk). The public key and gp define a message space $\mathcal{M}$.
- Sgn takes as input the secret key sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- Vfy takes as input the public key pk, a message $m \in \mathcal{M}$, and a purported signature σ, and outputs a bit.

We call S correct if for all $n \in \mathbb{N}$, $\mathrm{gp} \leftarrow \mathsf{SetupGrp}(1^n)$, $m \in \mathcal{M}$, $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{Gen}(\mathrm{gp})$, $\sigma \leftarrow \mathsf{Sgn}(\mathrm{sk}, m)$ we have $1 \leftarrow \mathsf{Vfy}(pk, \sigma, m)$.

We say that S is EUF-CMA secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{S},\mathcal{A}}(1^n)$ defined by

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

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

Our Instantiation. As we need to prove statements about signatures, the signature scheme has to be algebraic. For our construction, we use the structure-preserving signature scheme of Abe et al. [1], 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 [52]. For the scheme in [1], 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 $\mathrm{sk} \in \mathbb{Z}_\mathrm{p}^{\mu+\nu+2}$, $\mathrm{pk} \in G_1^{\mu+2} \times G_2^\nu$ and $\sigma \in G_2^2 \times G_1$.

We use the signature scheme S from Abe et al. [1] in the following ways in our system:

- 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 $\mu = 0$).

C.2.5 Asymmetric Encryption. We use the standard definitions for asymmetric encryption schemes and corresponding security notions, except that we enhance them with a SetupGrp algorithm to fit our algebraic setting.

Definition C.9 (Asymmetric Encryption). An *asymmetric encryption scheme* $\mathsf{E} := (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of four PPT algorithms:

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

- $\mathsf{Dec}(\mathsf{sk}, c)$ takes a key sk and a ciphertext c and outputs a plaintext message m or $\bot$. We assume that Dec is deterministic.

Correctness is defined in the usual sense.

An asymmetric encryption scheme E is *IND-CCA2*-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\text{IND-CCA-asym}}_{\mathsf{E},\mathcal{A}}(1^n)$ defined by

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

is negligible in n, 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^*.

An asymmetric encryption scheme E is *NM-CCA2*-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}^{\text{NM-CCA}}_{\mathsf{E},\mathcal{A}}(1^n)$ defined by

$$\left| \mathsf{Succs}^{\text{NM-CCA}}_{\mathsf{E},\mathcal{A},\text{real}}(1^n) - \mathsf{Succs}^{\text{NM-CCA}}_{\mathsf{E},\mathcal{A},\text{random}}(1^n) \right|$$

is negligible with

$$\mathsf{Succs}^{\text{NM-CCA}}_{\mathsf{E},\mathcal{A},\text{real}}(1^n) := \Pr\left[\begin{array}{c} c \notin \mathbf{c} \wedge \\ \bot \notin \mathbf{m} \wedge \\ R(m, \mathbf{m}) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (M, \mathit{state}) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot)}(1^n, \mathsf{pk}) \\ m \overset{\mathrm{R}}{\leftarrow} M \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ (R, \mathbf{c}) \leftarrow \mathcal{A}^{\mathsf{Dec}'(\mathsf{sk},\cdot)}(1^n, \mathit{state}, c) \\ \mathbf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathbf{c}) \end{array} \right]$$

and

$$\mathsf{Succs}^{\text{NM-CCA}}_{\mathsf{E},\mathcal{A},\text{random}}(1^n) := \Pr\left[\begin{array}{c} c \notin \mathbf{c} \wedge \\ \bot \notin \mathbf{m} \wedge \\ R(\widetilde{m}, \mathbf{m}) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n) \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp}) \\ (M, \mathit{state}) \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot)}(1^n, \mathsf{pk}) \\ m, \widetilde{m} \overset{\mathrm{R}}{\leftarrow} M \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ (R, \mathbf{c}) \leftarrow \mathcal{A}^{\mathsf{Dec}'(\mathsf{sk},\cdot)}(1^n, \mathit{state}, c) \\ \mathbf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathbf{c}) \end{array} \right],$$

where M denotes a space of valid, equally long messages, $R \subseteq M \times M^*$ denotes an relation, $\mathsf{Dec}(\mathsf{sk}, \cdot)$ 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.

An encryption is IND-CCA2 secure if and only if it is NM-CCA2 secure [13].

Our Instantiation. We will make use of two different IND-CCA2-secure encryption schemes:

- We implicitly use the encryption scheme by Cash et al. [25] to realize the secure channels underlying our model.
- We use a variant of Camenisch et al. [16] to instantiate the explicit encryption scheme E1 for the deposit Wallet IDs.

The former scheme is based on the TWIN-DH assumption and is used to setup a session key for a symmetric encryption of all protocol messages (cp. Appendix C.2.6) in the usual way.

The latter scheme is an adapted variant of the structure-preserving, IND-CCA2 secure encryption scheme by Camenisch et al. [16]. Thus, some remarks are in order. 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. [2] with some additional, manual optimizations. The transformed scheme encrypts vectors of G_1-elements and is secure under the co-DLIN assumption (cp. Definition C.4) which holds in the generic group model. This follows automatically from [2] (or can also be easily seen by inspecting the original proof in [16]). We present the modified scheme in full detail.

Definition C.10 (Type-3 variant of Camenisch et al. [16]). Let $\mathsf{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^n)$ be as in Definition C.1. Let $\wp$ be the dimension of the message space $G_1^{\wp}$. The algorithms Gen, Enc and Dec are depicted in Fig. 17.

We instantiate this scheme with $\wp = \ell + 2$.

C.2.6 Symmetric Encryption. We use standard definitions for symmetric encryption schemes and corresponding security notions.

Definition C.11 (Symmetric Encryption). A *symmetric encryption scheme* $\mathsf{E} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three PPT algorithms:

- $\mathsf{Gen}(1^n)$ outputs a (random) key k.
- $\mathsf{Enc}(k, m)$ takes a key k and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext c.
- $\mathsf{Dec}(k, c)$ takes a key k and a ciphertext c and outputs a plaintext message m or $\perp$. We assume that Dec is deterministic.

As for asymmetric encryption, we require correctness in the usual sense.

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 C.12 (IND-CCA2-Security for Symmetric Encryption). A symmetric encryption scheme E is *IND-CCA2*-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}_{\mathsf{E},\mathcal{A}}^{\mathsf{IND\text{-}CCA\text{-}sym}}(1^n)$ defined by

$$\left| \Pr\left[b = b' \;\middle|\; \begin{array}{c} k \leftarrow \mathsf{Gen}(1^n) \\ (state, j, \mathbf{m_0}, \mathbf{m_1}) \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot),\mathsf{Dec}(k,\cdot)}(1^n) \\ b \leftarrow \{0,1\} \\ \mathbf{c}^* \leftarrow (\mathsf{Enc}(k, m_{b,1}), \ldots, \mathsf{Enc}(k, m_{b,j})) \\ b' \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot),\mathsf{Dec}'(k,\cdot)}(state, \mathbf{c}^*) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in n, where $\mathbf{m_0}, \mathbf{m_1}$ are two vectors of $j \in \mathbb{N}$ bitstrings each such that for all $1 \le i \le j$: $|m_{0,i}| = |m_{1,i}|$, $\mathsf{Enc}(k,\cdot)$ and $\mathsf{Dec}(k,\cdot)$ denote oracles that return $\mathsf{Enc}(k,m)$ and $\mathsf{Dec}(k,c)$ for a m or c chosen by the adversary, and $\mathsf{Dec}'(k,\cdot)$ is the same as $\mathsf{Dec}(k,\cdot)$, except that it returns $\perp$ on input of any c_i^* that is contained in $\mathbf{c}^*$.

Our Instantiation. We use an IND-CCA2-secure symmetric encryption scheme in our protocol to encrypt the exchanged protocol messages. To this end, we combine an IND-CCA2-secure asymmetric encryption (see Appendix C.2.5) with an IND-CCA2-secure symmetric encryption in the usual KEM/DEM approach. The symmetric encryption 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 [12].

```
$\mathsf{Gen}(\mathsf{gp}, \wp)$
parse $(G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) := \mathsf{gp}$
$\alpha_1, \ldots, \alpha_\wp, \beta_0, \ldots, \beta_3, \gamma_1, \ldots, \gamma_\wp \overset{R}{\leftarrow} \mathbb{Z}_\mathfrak{p}^3$
$\mathsf{sk} := (\{\alpha_i\}_{i=1,\ldots,\wp}, \{\beta_i\}_{i=0,\ldots,3}, \{\gamma_i\}_{i=1,\ldots,\wp})$
$\xi_1, \ldots, \xi_3 \overset{R}{\leftarrow} \mathbb{Z}_\mathfrak{p}^*$
$\check{h}_1 := g_1^{\xi_1}, \qquad \check{h}_2 := g_1^{\xi_2}, \qquad \check{h}_3 := g_1^{\xi_3}$
$\hat{h}_1 := g_2^{\xi_1}, \qquad \hat{h}_2 := g_2^{\xi_2}, \qquad \hat{h}_3 := g_2^{\xi_3}$
$x_{i,1} := \check{h}_1^{\alpha_{i,1}} \check{h}_3^{\alpha_{i,3}}, \qquad x_{i,2} := \check{h}_2^{\alpha_{i,2}} \check{h}_3^{\alpha_{i,3}}, \qquad$ for $i = 1, \ldots, \wp$
$y_{i,1} := \hat{h}_1^{\beta_{i,1}} \hat{h}_3^{\beta_{i,3}}, \qquad y_{i,2} := \hat{h}_2^{\beta_{i,2}} \hat{h}_3^{\beta_{i,3}}, \qquad$ for $i = 0, \ldots, 3$
$z_{i,1} := \hat{h}_1^{\gamma_{i,1}} \hat{h}_3^{\gamma_{i,3}}, \qquad z_{i,2} := \hat{h}_2^{\gamma_{i,2}} \hat{h}_3^{\gamma_{i,3}}, \qquad$ for $i = 1, \ldots, \wp$
$\mathsf{pk} := \Big(\check{h}_1, \check{h}_2, \check{h}_3, \hat{h}_1, \hat{h}_2, \hat{h}_3,$
    $\{x_{i,1}, x_{i,2}\}_{i=1,\ldots,\wp}, \{y_{i,1}, y_{i,2}\}_{i=0,\ldots,3}, \{z_{i,1}, z_{i,2}\}_{i=1,\ldots,\wp}\Big)$
**return** $(\mathsf{pk}, \mathsf{sk})$
```

(a) Key generation algorithm Gen

```
$\mathsf{Enc}(\mathsf{pk}, m)$
parse $\Big(\check{h}_1, \check{h}_2, \check{h}_3, \hat{h}_1, \hat{h}_2, \hat{h}_3, \{x_{i,1}, x_{i,2}\}_{i=1,\ldots,\wp},$
    $\{y_{i,1}, y_{i,2}\}_{i=0,\ldots,3}, \{z_{i,1}, z_{i,2}\}_{i=1,\ldots,\wp}\Big) := \mathsf{pk}$
$r, s \overset{R}{\leftarrow} \mathbb{Z}_\mathfrak{p}$
$\check{u}_1 := \check{h}_1^r \qquad \check{u}_2 := \check{h}_2^s \qquad \check{u}_3 := \check{h}_3^{r+s}$
$\hat{u}_1 := \hat{h}_1^r \qquad \hat{u}_2 := \hat{h}_2^s \qquad \hat{u}_3 := \hat{h}_3^{r+s}$
$c_i = m_i x_{i,1}^r x_{i,2}^s$ for $i = 1, \ldots, \wp$
$v = \prod_{i=0}^{3} e\left(\check{u}_i, y_{i,1}^r y_{i,2}^s\right) \prod_{i=1}^{\wp} e\left(c_i, z_{i,1}^r z_{i,2}^s\right)$ with $\check{u}_0 := g_1$
$\mathfrak{c} := (u, c, v)$ with $u := (\check{u}_1, \check{u}_2, \check{u}_3, \hat{u}_1, \hat{u}_2, \hat{u}_3)$ and $c := (c_1, \ldots, c_\wp)$
**return** $(\mathfrak{c})$
```

(b) Encryption algorithm Enc

```
$\mathsf{Dec}(\mathsf{sk}, \mathfrak{c})$
parse $(\{\alpha_i\}_{i=1,\ldots,\wp}, \{\beta_i\}_{i=0,\ldots,3}, \{\gamma_i\}_{i=1,\ldots,\wp}) := \mathsf{sk}$
parse $(u, c, v) := \mathfrak{c}$, $(\check{u}_1, \check{u}_2, \check{u}_3, \hat{u}_1, \hat{u}_2, \hat{u}_3) := u$ and $(c_1, \ldots, c_\wp) := c$
$\check{u}_0 := g_1$
**if** $v \neq \prod_{i=0}^{3} e\left(\check{u}_i, \hat{u}_1^{\beta_{i,1}} \hat{u}_2^{\beta_{i,2}} \hat{u}_3^{\beta_{i,3}}\right) \prod_{i=1}^{\wp} e\left(c_i, \hat{u}_1^{\gamma_{i,1}} \hat{u}_2^{\gamma_{i,2}} \hat{u}_3^{\gamma_{i,3}}\right)$ abort
**if** $e(\check{u}_i, g_2) \neq e(g_1, \hat{u}_i)$ for any $i \in \{1, 2, 3\}$ abort
$m_i := c_i \check{u}_1^{-\alpha_{i,1}} \check{u}_2^{-\alpha_{i,2}} \check{u}_3^{-\alpha_{i,3}}$ for $i \in \{1, \ldots, \wp\}$
$m := (m_1, \ldots, m_\wp)$
**return** $(m)$
```

(c) Decryption algorithm Dec

Fig. 17. The adapted CCA-secure encryption scheme by Camenisch et al. [16] with parameter $\wp$ and message space $G_1^\wp$

C.2.7 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. More precisely, a PRF—more precisely a family of PRF's in the security parameter 1^n—is defined as follows.

Definition C.13. A (group-based) *pseudo-random function* (PRF) PRF := (SetupGrp, Gen, Eval) consists of three PPT algorithms:

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

We say that PRF is secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathrm{Adv}^{\mathrm{prf}}_{\mathcal{A}}(1^n)$ defined by

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

is negligible in n.

Our Instantiation. As we want to efficiently prove statements about PRF outputs, we use an efficient algebraic construction, namely the Dodis-Yampolskiy PRF [35]. This function is defined by $F(k, x) : \mathbb{Z}_{\mathrm{p}}^2 \rightarrow G_1$, $(k, x) \mapsto g_1^{\frac{1}{x+k}}$, where $k \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathrm{p}}$ is the random PRF key. It is secure for inputs $\{0, \ldots, n_{\mathrm{PRF}}\} \subset \mathbb{Z}_{\mathrm{p}}$ under the n_{PRF}-DDHI assumption. This is a family of increasingly stronger assumptions which is assumed to hold for asymmetric bilinear groups.

C.3 Types of Equations Supported by GS-NIZKs

Let SetupGrp be a bilinear group generator (cf. Definition C.1) for which the SXDH assumption holds and gp := $(G_1, G_2, G_T, e, \mathrm{p}, g_1, g_2, g_T) \leftarrow \mathsf{SetupGrp}(1^n)$ denotes the output of SetupGrp. Furthermore, let $X_1, \ldots, X_{m_1} \in G_1$, $x_1, \ldots, x_{m_2} \in \mathbb{Z}_{\mathrm{p}}$, $Y_1, \ldots, Y_{m_3} \in G_2$, and $y_1, \ldots, y_{m_4} \in \mathbb{Z}_{\mathrm{p}}$ denote variables in the following types of equations:

- *Pairing-Product Equation (PPE):*

$$\prod_{i=1}^{m_3} e(A_i, Y_i) \prod_{i=1}^{m_1} e(X_i, B_i) \prod_{i=1}^{m_1} \prod_{j=1}^{m_3} e(X_i, Y_i)^{\gamma_{i,j}} = t_T$$

  for constants $A_i \in G_1$, $B_i \in G_2$, $t_T \in G_T$, $\gamma_{i,j} \in \mathbb{Z}_{\mathrm{p}}$.
- *Multi-Scalar Equation (MSE) over G_1:*

$$\prod_{i=1}^{m_4} A_i^{y_i} \prod_{i=1}^{m_1} X_i^{b_i} \prod_{i=1}^{m_1} \prod_{j=1}^{m_4} X_i^{\gamma_{i,j} y_j} = t_1$$

  for constants $A_i, t_1 \in G_1$, $b_i, \gamma_{i,j} \in \mathbb{Z}_{\mathrm{p}}$.
- *Multi-Scalar Equation (MSE) over G_2:*

$$\prod_{i=1}^{m_2} B_i^{x_i} \prod_{i=1}^{m_3} Y_i^{a_i} \prod_{i=1}^{m_2} \prod_{j=1}^{m_3} Y_j^{\gamma_{i,j} x_i} = t_2$$

  for constants $B_i, t_2 \in G_2$, $a_i, \gamma_{i,j} \in \mathbb{Z}_{\mathrm{p}}$.
- *Quadratic Equation (QE) over $\mathbb{Z}_{\mathrm{p}}$:*

$$\sum_{i=1}^{m_4} a_i y_i + \sum_{i=1}^{m_2} x_i b_i + \sum_{i=1}^{m_2} \sum_{j=1}^{m_4} \gamma_{i,j} x_i y_j = t$$

for constants $a_i, b_i, \gamma_{i,j}, t \in \mathbb{Z}_p$.

Let L_{gp} be a language containing statements described by the conjunction of n_1 pairing-product equations over gp, n_2 multi-scalar equations over G_1, n_3 multi-scalar equations over G_2, and n_4 quadratic equations over $\mathbb{Z}_p$, where $n_i \in \mathbb{N}_0$ are constants, as well as by witnesses

$$w = (X_1, \ldots, X_{m_1}, x_1, \ldots, x_{m_2}, Y_1, \ldots, Y_{m_3}, y_1, \ldots, y_{m_4}),$$

where $m_i \in \mathbb{N}_0$. Then the Groth-Sahai proof system for L_{gp}, as introduced by [39], is perfectly correct, perfectly sound, and satisfies F_{gp}-extractability [36, 39] for

$$F_{\text{gp}} : G_1^{m_1} \times \mathbb{Z}_p^{m_2} \times G_2^{m_3} \times Z_p^{m_4} \rightarrow G_1^{m_1} \times G_1^{m_2} \times G_2^{m_3} \times G_2^{m_4}$$

with

$$F_{\text{gp}}(w) := ((X_i)_{i\in[m_1]}, (g_1^{x_i})_{i\in[m_2]}, (Y_i)_{i\in[m_3]}, (g_2^{y_i})_{i\in[m_4]}).^{23}$$

It is also known to be composable zero-knowledge [36, 39] as long as for all PPEs in L_{gp} holds that either

- $t_T = 1$ or
- the right-hand side of the PPE can be written as $\prod_{i=1}^{k} e(A_i, B_i)$ for constants $A_i \in G_1$, $B_i \in G_2$, such that for each i DLOG(A_i) or DLOG(B_i) is known.

In the latter case, *hint* from Definition C.5 would contain these discrete logarithms which would simply be put (as additional elements) into the simulation trapdoor td_{spok}. Also note that if these discrete logarithms are not known, there is a workaround which consists of adding new helper variables to L_{gp} [39].

D FULL PROTOCOL DESCRIPTION

In this appendix we describe and define a real protocol π_{P4TC} that implements our toll collection system $\mathcal{F}_{\text{P4TC}}$. We say

Definition D.1 (P4TC Scheme). A protocol π is called a *privacy-preserving electronic toll collection* scheme, if it GUC-realizes $\mathcal{F}_{\text{P4TC}}$.

The proof that π_{P4TC} is a GUC-realization of $\mathcal{F}_{\text{P4TC}}$ is postponed to Appendix F. The style of the presentation follows the same lines as the presentation of the ideal model $\mathcal{F}_{\text{P4TC}}$ in Appendix B: 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 tags” as well as “proof of participation challenges”, which are periodically sent to the TSP, and the TSP creates and keeps “hidden user trapdoors” for each wallet issued. A precise definition what is stored by which party is depicted in Fig. 18. For typographic reasons we additionally split the presentation of most tasks into a *wrapper protocol* and a *core protocol*. Except for a few cases, there is a one-to-one correspondence between wrapper and core protocols. The wrapper protocols have the same input/output interfaces as their ideal counterparts and describe steps that are executed by each party locally before and after the respective core protocol. These steps include loading keys, parsing the previously stored state, persisting the new state after the core protocol has returned, etc. The core protocols describe the actual interaction between parties and what messages are exchanged.

This dichotomy between wrapper and core protocols is lifted for four exceptions:

[23] F acts as identity function on group elements $a \in G_1$ and $b \in G_2$ but returns $g_1^s \in G_1$ or $g_2^s \in G_2$ for exponents $s \in \mathbb{Z}_p$.

UC-Protocol π_{P4TC}

I. Local State

(1) The TSP internally records:
- It's public and private key $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$.
- A self-signed certificate $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$.
- A set *HTD* of hidden user trapdoors.
- A (partial) mapping $\{\mathsf{pk}_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}\}$ of RSU attributes.

(2) Each RSU internally records:
- It's public and private key $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$.
- A certificate $\mathsf{cert}_{\mathcal{R}}$ signed by the TSP.
- Sets Ω^{dsp}, Ω^{bl} and $\Omega^{\text{pp}}_{\mathcal{R}}$ of transaction information for double-spending detection, blacklisting and prove participation respectively.

(3) Each user internally records:
- His public and private key $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$.
- A set $\{\tau\}$ of all past tokens issued to him.
- A set $\Omega^{\text{pp}}_{\mathcal{U}}$ of transaction information for prove participation.

II. Behavior

- *DR Registration* (Fig. 20)
- *TSP Registration* (Fig. 22)
- *RSU Registration* (Fig. 24)
- *User Registration* (Fig. 26)
- *RSU Certification* (Fig. 28)
- *Wallet Issuing* (Fig. 30)
- *Debt Accumulation* (Fig. 32)
- *Debt Clearance* (Fig. 34)
- *Prove Participation* (Fig. 36)
- *Double-Spending Detection* (Fig. 38)
- *Guilt Verification* (Fig. 39)
- *User Blacklisting* (Fig. 40)

Fig. 18. The UC-protocol π_{P4TC}

(1) We give an algorithm for the setup of the system (cf. Fig. 19) which explains how the CRS is generated. Of course, there is no wrapper protocol because setup of the CRS is not even part of our protocol but part of the setup assumption and provided by $\mathcal{F}_{\text{CRS}}$.

(2) We describe a "utility algorithm" WalletVerification (cf. Fig. 42). This algorithm has no purpose on its own, but simple collects some shared code of multiple tasks.

(3)+(4) We only have "wrapper protocols" for the tasks Double-Spending Detection and Guilt Verification (cf. Figs. 38 and 39) because they are so simple that splitting it each into two yields no advantage.

D.1 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 protocols.

```
$\mathsf{Setup}(1^n, \mathcal{B})$
$\mathsf{gp} := (G_1, G_2, G_T, e, \mathfrak{p}, g_1, g_2) \leftarrow \mathsf{SetupGrp}(1^n)$
$\mathsf{CRS}^1_{\mathsf{com}} \leftarrow \mathsf{C1.Gen}(\mathsf{gp})$
$\mathsf{CRS}^2_{\mathsf{com}} \leftarrow \mathsf{C2.Gen}(\mathsf{gp})$
$\mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}(\mathsf{gp})$
$\mathsf{CRS} := (\mathsf{gp}, \mathcal{B}, \mathsf{CRS}^1_{\mathsf{com}}, \mathsf{CRS}^2_{\mathsf{com}}, \mathsf{CRS}_{\mathsf{pok}})$
return $\mathsf{CRS}$
```

Fig. 19. System Setup Algorithm

D.2 Wallets

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

$$\tau := (s, \phi, x^{\text{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\text{next}}).$$

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* $c_{\mathcal{T}}$ (a commitment on λ and the secret user key $\mathsf{sk}_{\mathcal{U}}$), its corresponding opening $d_{\mathcal{T}}$ and a signature $\sigma_{\mathcal{T}}$ on $c_{\mathcal{T}}$ and $\mathbf{a}_{\mathcal{U}}$ created by the TSP.

The alterable components consist of the *RSU commitment* $c_{\mathcal{R}}$ (a commitment on λ, b, u_1^{next} and x^{next}), its corresponding opening $d_{\mathcal{R}}$, a signature $\sigma_{\mathcal{R}}$ on $c_{\mathcal{R}}$ 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* $\mathsf{cert}_{\mathcal{R}}$ and the *serial number* s and *fraud detection ID* $\phi := \mathsf{PRF}(\lambda, x^{\text{next}} - 1)$ for the current transaction. These components change after each interaction with an RSU via the Debt Accumulation task.

In the following, a protocol or algorithm for each task is presented. For a better overview, it is depicted in Fig. 1 which parties are involved in each task (except for some registration tasks). Also, the used variables are summarized in Tables 4 and 5.

D.3 System Setup

To setup the system once (see Fig. 19), 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. We assume that the CRS is implicitly available to all protocols and algorithms. Either a number of mutually distrusting parties 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.

D.4 Registration

The registration algorithms of *DR*, $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{U}$ are all presented with a wrapper protocol π_{P4TC} (see Figs. 20, 22, 24 and 26) and a core protocol (see Figs. 21, 23, 25 and 27). The wrapper protocol interacts with other UC entities, pre-processes the inputs, post-processes the outputs and internally invokes the core protocols. In the following, only the mechanics of the core protocols are explicitly described.

Table 5. Notation that only occurs in the real protocol

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

UC-Protocol π_{P4TC} (cont.) – Task *DR Registration*

DR input: $(\texttt{register})$

(1) If a key pair $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})$ has already been recorded, output $\perp$ and abort.
(2) Obtain CRS CRS from $\mathcal{F}_{\text{CRS}}$.
(3) Run $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}}) \leftarrow \mathsf{DRRegistration}(\mathsf{CRS})$ (see Fig. 21).
(4) Record $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{DR}})$.

DR output: $(\mathsf{pk}_{\mathcal{DR}})$

Fig. 20. The UC-protocol π_{P4TC} (cont. from Fig. 18)

```
$\mathsf{DRRegistration}(\mathsf{CRS})$
parse $(\mathsf{gp}, \mathcal{B}, \mathsf{CRS}^1_{\mathsf{com}}, \mathsf{CRS}^2_{\mathsf{com}}, \mathsf{CRS}_{\mathsf{pok}}) := \mathsf{CRS}$
$(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}}) \leftarrow \mathsf{E.Gen}(\mathsf{gp})$
return $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})$
```

Fig. 21. DR Registration Core Protocol

UC-Protocol π_{P4TC} (cont.) – Task *TSP Registration*

TSP input: $(\texttt{register}, \mathbf{a}_{\mathcal{T}})$

(1) If a key pair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ has already been recorded, output $\perp$ and abort.
(2) Obtain CRS CRS from $\mathcal{F}_{\text{CRS}}$.
(3) Run $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{TSPRegistration}(\mathsf{CRS}, \mathbf{a}_{\mathcal{T}})$ (see Fig. 23).
(4) Record $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ and $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{T}})$.

TSP output: $(\mathsf{pk}_{\mathcal{T}})$

Fig. 22. The UC-protocol π_{P4TC} (cont. from Fig. 18)

```
$\mathsf{TSPRegistration}(\mathsf{CRS}, \mathbf{a}_{\mathcal{T}})$
parse $(\mathsf{gp}, \mathcal{B}, \mathsf{CRS}^1_{\mathsf{com}}, \mathsf{CRS}^2_{\mathsf{com}}, \mathsf{CRS}_{\mathsf{pok}}) := \mathsf{CRS}$
$(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
$(\mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
$(\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
$(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) := \big((\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}), (\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}})\big)$
$\sigma^{\mathsf{cert}}_{\mathcal{T}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}}, (\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}}))$
$\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}} := (\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}}, \sigma^{\mathsf{cert}}_{\mathcal{T}})$
return $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$
```

Fig. 23. TSP Registration Core Protocol

UC-Protocol π_{P4TC} (cont.) – Task *RSU Registration*

RSU input: (`register`)

(1) If a key pair $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ has already been stored, output $\perp$ and abort.
(2) Obtain CRS CRS from $\mathcal{F}_{\text{CRS}}$.
(3) Run $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{RSURegistration}(\mathsf{CRS})$ (see Fig. 25).
(4) Store $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input (`register`, $\mathsf{pk}_{\mathcal{R}}$).

RSU output: $(\mathsf{pk}_{\mathcal{R}})$

Fig. 24. The UC-protocol π_{P4TC} (cont. from Fig. 18)

```
RSURegistration(CRS)
parse $(\mathsf{gp}, \mathcal{B}, \mathsf{CRS}^1_{\mathsf{com}}, \mathsf{CRS}^2_{\mathsf{com}}, \mathsf{CRS}_{\mathsf{pok}}) := \mathsf{CRS}$
$(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{S.Gen}(\mathsf{gp})$
return $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$
```

Fig. 25. RSU Registration Core Protocol

UC-Protocol π_{P4TC} (cont.) – Task *User Registration*

User input: (`register`)

(1) If a key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ has already been stored, output $\perp$ and abort.
(2) Obtain CRS CRS from $\mathcal{F}_{\text{CRS}}$.
(3) Run $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \leftarrow \mathsf{UserRegistration}(\mathsf{CRS})$ (see Fig. 27).
(4) Store $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input (`register`, $\mathsf{pk}_{\mathcal{U}}$).

User output: $(\mathsf{pk}_{\mathcal{U}})$

Fig. 26. The UC-protocol π_{P4TC} (cont. from Fig. 18)

```
UserRegistration(CRS)
parse $(\mathsf{gp}, \mathcal{B}, \mathsf{CRS}^1_{\mathsf{com}}, \mathsf{CRS}^2_{\mathsf{com}}, \mathsf{CRS}_{\mathsf{pok}}) := \mathsf{CRS}$
$y \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$
$(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) := (g_1^y, y)$
return $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$
```

Fig. 27. User Registration Core Protocol

The DR computes a key pair $(\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$ which can be used to remove the unlinkability of user transactions in case of a dispute between the user and the TSP (see Figs. 20 and 21). The DR could be a (non-governmental) organization trusted by both, users to protect their privacy and the TSP to protect operator security.

UC-Protocol π_{P4TC} (cont.) – Task *RSU Certification*

RSU input: $(\texttt{certify})$

TSP input: $(\texttt{certify}, \mathbf{a}_{\mathcal{R}})$

(1) At the RSU side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$.$^{\perp}$
- Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $\mathit{pid}_{\mathcal{T}}$.$^{\perp}$

(2) At the TSP side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$.$^{\perp}$
- Receive $\mathsf{pk}_{\mathcal{R}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $\mathit{pid}_{\mathcal{R}}$.$^{\perp}$
- Check that no mapping $\mathsf{pk}_{\mathcal{R}} \mapsto \mathbf{a}'_{\mathcal{R}}$ has been registered before, else output $\perp$ and abort.

(3) Both sides: Run the code of RSUCertification between the RSU and the TSP (see Fig. 29)

$$((\mathsf{cert}_{\mathcal{R}}), (\mathsf{OK})) \leftarrow \mathsf{RSUCertification}\left\langle \mathcal{R}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}), \mathcal{T}(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}) \right\rangle .$$

(4) At the RSU side:
- Parse $\mathbf{a}_{\mathcal{R}}$ from $\mathsf{cert}_{\mathcal{R}}$.
- Record $\mathsf{cert}_{\mathcal{R}}$ internally.

(5) At the TSP side:
- Record $\mathsf{pk}_{\mathcal{R}} \mapsto \mathbf{a}_{\mathcal{R}}$ internally.

RSU output: $(\mathbf{a}_{\mathcal{R}})$

TSP output: (OK)

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

Fig. 28. The UC-protocol π_{P4TC} (cont. from Fig. 18)

The TSP must also generate a key pair (see Figs. 22 and 23). Therefore, the TSP generates several signature key pairs $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{T}})$, $(\mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \mathsf{sk}_{\mathcal{T}}^{\text{cert}})$, $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}})$, where $\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$ is used in the Wallet Issuing task to sign the TSP commitment $c_{\mathcal{T}}$ 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 $c_{\mathcal{R}}$ 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 $\mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}$.

Each RSU must generate a key pair as well (see Figs. 24 and 25). For that purpose, each RSU generates a signature key pair $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ that is used in the Debt Accumulation task to sign the RSU commitment $c_{\mathcal{R}}$.

Each user also has to generate a key pair (see Figs. 26 and 27). 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.

D.5 RSU Certification

The RSU Certification task is executed between $\mathcal{R}$ and $\mathcal{T}$ when a new RSU is deployed into the field. The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 28) and a core protocol RSUCertification (see Fig. 29). The wrapper protocol interacts with other UC entities, pre-processes the input and post-processes the output.

$\mathcal{R}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}})$		$\mathcal{T}(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$
parse $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$		parse $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$
		parse $(\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathsf{cert}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{sk}_{\mathcal{T}}$
		$\sigma_{\mathcal{R}}^{\mathsf{cert}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}_{\mathcal{T}}^{\mathsf{cert}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$
		$\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\mathsf{cert}})$
	$\xleftarrow{\mathsf{cert}_{\mathcal{R}}}$	
parse $(\mathsf{pk}'_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\mathsf{cert}}) := \mathsf{cert}_{\mathcal{R}}$		
if $\mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{T}}^{\mathsf{cert}}, \sigma_{\mathcal{R}}^{\mathsf{cert}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$		
return $\perp$		
return $(\mathsf{cert}_{\mathcal{R}})$		return (OK)

Fig. 29. RSU Certification Core Protocol

The wrapper protocol internally invokes the core protocol

$$((\mathsf{cert}_{\mathcal{R}}), (\mathsf{OK})) \leftarrow \mathsf{RSUCertification}\left\langle \mathcal{R}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}), \mathcal{T}(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}) \right\rangle.$$

In the core protocol, 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 $\mathsf{pk}_{\mathcal{R}}$ 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.

D.6 Wallet Issuing

The Wallet Issuing task is executed between $\mathcal{U}$ and $\mathcal{T}$. It is executed at the beginning of each billing period to generate a fresh wallet for the user. The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 30) and a core protocol WalletIssuing (see Fig. 31). The wrapper protocol interacts with other UC entities, pre-processes the input, post-processes the output and checks the validity of the created wallet by executing the WalletVerification algorithm (see Fig. 42) after the core protocol. The wrapper protocol internally invokes the core protocol

$$((\tau), (s, htd)) \leftarrow \mathsf{WalletIssuing}\left\langle \mathcal{U}(\mathsf{pk}_{DR}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}), \mathcal{T}(\mathsf{pk}_{DR}, \mathsf{sk}_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}, bl_{\mathcal{T}}) \right\rangle.$$

The joint input of the core protocol is the public key of the DR pk_{DR}. The user additionally obtains its public and secret key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$. The TSP also gets his own secret key $\mathsf{sk}_{\mathcal{T}}$, the attribute vector $\mathbf{a}_{\mathcal{U}}$ for the user, its own certificate $\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}$ and the TSP blacklist $bl_{\mathcal{T}}$ as input.

The protocol 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 $\lambda := \lambda' + \lambda''$. To this end, the parties engage in the first two message of a Blum coin

UC-Protocol π_{P4TC} (cont.) – Task *Wallet Issuing*

User input: $(\texttt{issue})$
TSP input: $(\texttt{issue}, \mathbf{a}_{\mathcal{U}}, bl_{\mathcal{T}})$

(1) At the user side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$.$^\perp$
- Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{T}}$.$^\perp$
- Receive pk_{DR} from the bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$ for PID pid_{DR}.$^\perp$

(2) At the TSP side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$.$^\perp$
- Load the internally recorded $\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}$.$^\perp$
- Receive pk_{DR} from the bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$ for PID pid_{DR}.$^\perp$

(3) Both sides: Run the code of WalletIssuing between the user and the TSP (see Fig. 31)

$$((\tau), (s, htd)) \leftarrow \mathsf{WalletIssuing}\left\langle \mathcal{U}(\mathsf{pk}_{DR}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}), \mathcal{T}(\mathsf{pk}_{DR}, \mathsf{sk}_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}, bl_{\mathcal{T}})\right\rangle.$$

(4) At the user side:
- Run the code of $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$ (see Fig. 42).
- If WalletVerification returns 0, output $\perp$ and abort.
- Record τ internally.
- Parse s and $\mathbf{a}_{\mathcal{U}}$ from τ.

(5) At the TSP side:
- Insert htd into HTD.

User output: $(s, \mathbf{a}_{\mathcal{U}})$
TSP output: (s)

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

Fig. 30. The UC-protocol π_{P4TC} (cont. from Fig. 18)

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 pk_{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., $\mathbb{Z}_p$) 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 $\lambda'_0, \ldots, \lambda'_{\ell-1} \in \{0, \ldots, \mathcal{B}-1\}$ s.t. $\lambda' = \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$ for some base $\mathcal{B}$. The 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 its own public key $\mathsf{pk}_{\mathcal{U}}$ under the public key pk_{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 $htd := (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$. 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 User Blacklisting task in Fig. 41.

$\mathcal{U}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$		$\mathcal{T}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, bl_{\mathcal{T}})$
		parse $(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{sk}_{\mathcal{T}}$
$s' \overset{\mathrm{R}}{\leftarrow} G_1$		$s'' \overset{\mathrm{R}}{\leftarrow} G_1$
$\lambda'_i \overset{\mathrm{R}}{\leftarrow} \{0, \ldots, \mathcal{B} - 1\}$ for $i \in \{0, \ldots, \ell - 1\}$		
$\lambda' := \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$		$\lambda'' \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$
$(c'_{\mathrm{seed}}, d'_{\mathrm{seed}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, \lambda')$		$(c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) \leftarrow \mathsf{C2.Com}(\mathsf{CRS}^2_{\mathrm{com}}, s'')$
	$\xrightarrow{\mathsf{pk}_{\mathcal{U}}, c'_{\mathrm{seed}}}$	

		if $\mathsf{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$
		return `blacklisted`

	$\xleftarrow{\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}, c''_{\mathrm{ser}}, \lambda''}$	

parse $(\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}) := \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$
if $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}})) = 0$
return $\perp$
$\lambda := \lambda' + \lambda''$		

$\Lambda := g_1^{\lambda}, \Lambda' := g_1^{\lambda'}, \Lambda'' := g_1^{\lambda''}, \Lambda'_i := g_1^{\lambda'_i}$ for $i \in \{0, \ldots, \ell - 1\}$		$\Lambda'' := g_1^{\lambda''}$
$r_1, r_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ $e^* \leftarrow \mathsf{E1.Enc}(\mathsf{pk}_{\mathcal{DR}}, (\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}_{\mathcal{U}}); r_1, r_2)$ $u_1^{\mathrm{next}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ $(c_{\mathcal{T}}, d_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda, \mathsf{sk}_{\mathcal{U}}))$ $(c_{\mathcal{R}}, d_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda, 0, u_1^{\mathrm{next}}, 1))$ $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{DR}}, e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, c'_{\mathrm{seed}}, \Lambda'', \lambda'')$ $wit := (\lambda, \lambda', \lambda'_0, \ldots, \lambda'_{\ell-1}, r_1, r_2, \Lambda, \Lambda',$ $\Lambda'_0, \ldots, \Lambda'_{\ell-1}, g_1^{u_1^{\mathrm{next}}}, d_{\mathcal{T}}, d_{\mathcal{R}}, d'_{\mathrm{seed}}, g_2^{\mathsf{sk}_{\mathcal{U}}})$ $\pi \leftarrow \mathsf{P1.Prove}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, wit)$		
	$\xrightarrow{s', e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, \pi}$	
		$s := s' \cdot s''$ $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{DR}}, e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, c'_{\mathrm{seed}}, \Lambda'', \lambda'')$ if $\mathsf{P1.Vfy}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, \pi) = 0$ return $\perp$ $\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (c_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}))$ $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (c_{\mathcal{R}}, s))$
	$\xleftarrow{s'', d''_{\mathrm{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}}}$	

if $\mathsf{C2.Open}(\mathsf{CRS}^2_{\mathrm{com}}, s'', c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) = 0$
return $\perp$
$s := s' \cdot s''$
$\tau := (s, \mathsf{PRF}(\lambda, 0), 1, \lambda, \mathsf{a}_{\mathcal{U}},$		

$c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, 0, u_1^{\mathrm{next}})$		$htd := (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$
return (τ)		return (s, htd)

Fig. 31. Wallet Issuing Core Protocol

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 and 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. He commits to the wallet ID λ and his secret user key $\mathsf{sk}_{\mathcal{U}}$ for the TSP commitment $c_{\mathcal{T}}$. For the RSU commitment $c_{\mathcal{R}}$, he commits to the wallet ID λ, the balance $b := 0$, a fresh double-spending mask u_1^{next} and the PRF counter $x^{\text{next}} := 1$. He then 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 *stmnt* from the language $L_{\text{gp}}^{(1)}$ defined by

$$L_{\text{gp}}^{(1)} := \left\{ \begin{pmatrix} \mathsf{pk}_{\mathcal{U}} \\ \mathsf{pk}_{DR} \\ e^* \\ c_{\mathcal{T}} \\ c_{\mathcal{R}} \\ c'_{\text{seed}} \\ \Lambda'' \\ \lambda'' \end{pmatrix}^{\top} \left| \begin{array}{l} \exists\, \lambda, \lambda', \lambda'_0, \ldots, \lambda'_{\ell-1}, r_1, r_2 \in \mathbb{Z}_p; \\ \Lambda, \Lambda', \Lambda'_0, \ldots, \Lambda'_{\ell-1}, U_1^{\text{next}}, d_{\mathcal{T}}, d_{\mathcal{R}}, d'_{\text{seed}} \in G_1; \\ \mathsf{SK}_{\mathcal{U}} \in G_2 : \\[4pt] e\left(\mathsf{pk}_{\mathcal{U}}, g_2\right) = e\left(g_1, \mathsf{SK}_{\mathcal{U}}\right) \\ \mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, \mathsf{pk}_{\mathcal{U}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, \mathbb{1}, U_1^{\text{next}}, g_1), c_{\mathcal{R}}, d_{\mathcal{R}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, \Lambda', c'_{\text{seed}}, d'_{\text{seed}}) = 1 \\ e^* = \mathsf{E1.Enc}(\mathsf{pk}_{DR}, (\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}_{\mathcal{U}}); r_1, r_2) \\ \lambda = \lambda' + \lambda'' \\ \Lambda = g_1^{\lambda},\ \ \Lambda' = g_1^{\lambda'} \\ \lambda' = \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i \\ \forall i \in \{0, \ldots, \ell-1\} : \\ \qquad \lambda'_i \in \{0, \ldots, \mathcal{B}-1\} \\ \qquad \Lambda'_i = g_1^{\lambda'_i} \end{array} \right. \right\} \tag{6}$$

Note that the first equation in Eq. (6) actually proves the knowledge of $g_2^{\mathsf{sk}_{\mathcal{U}}}$ (rather than $\mathsf{sk}_{\mathcal{U}}$ itself).[24] However, computing $g_2^{\mathsf{sk}_{\mathcal{U}}}$ without knowing $\mathsf{sk}_{\mathcal{U}}$ (only given $\mathsf{pk}_{\mathcal{U}}$) is assumed to be a hard problem (Co-CDH).

In temporal order, the protocol proceeds as follows: In the first message (from user to TSP) the user sends its own public key $\mathsf{pk}_{\mathcal{U}}$ and starts the Blum coin toss for the wallet ID by sending c'_{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 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 c''_{ser}. Moreover, the TSP sends its own certificate $\mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}$ 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 $c_{\mathcal{T}}$ and $c_{\mathcal{R}}$ 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^*, $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ 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 $c_{\mathcal{T}}$ together with $\mathbf{a}_{\mathcal{U}}$ for the fixed part of the wallet, and $\sigma_{\mathcal{R}}$ on $c_{\mathcal{R}}$ together with s on the updatable part of the wallet. Please note that at this point the serial number $s := s' \cdot s''$ is fixed and known to the TSP. In the forth 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 d''_{ser}. At this point the user assembles all part to obtain a fully functional wallet

$$\tau := (s, \phi := \mathsf{PRF}(\lambda, 0), x^{\text{next}} := 1, \lambda := \lambda' + \lambda'', \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{T}}^{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, b := 0, u_1^{\text{next}}).$$

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

UC-Protocol π_{P4TC} (cont.) – Task *Debt Accumulation*

User input: $(\mathtt{pay\_toll}, s^{\mathrm{prev}})$
RSU input: $(\mathtt{pay\_toll}, bl_{\mathcal{R}})$

(1) At the user side:
- Load the internally recorded $(\mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}})$.$^{\perp}$
- Receive $\mathrm{pk}_{\mathcal{T}}$ from the bulletin-board $\overline{\mathcal{G}}_{\mathrm{bb}}$ for PID $pid_{\mathcal{T}}$.$^{\perp}$
- Load the internally recorded token τ^{prev} for serial number s^{prev}.$^{\perp}$

(2) At the RSU side:
- Load the internally recorded $(\mathrm{pk}_{\mathcal{R}}, \mathrm{sk}_{\mathcal{R}})$.$^{\perp}$
- Load the internally recorded $\mathrm{cert}_{\mathcal{R}}$.$^{\perp}$
- Receive $\mathrm{pk}_{\mathcal{T}}$ from the bulletin-board $\overline{\mathcal{G}}_{\mathrm{bb}}$ for PID $pid_{\mathcal{T}}$.$^{\perp}$

(3) Both sides: Run the code of DebtAccumulation between the user and the RSU (see Fig. 33)

$$\begin{pmatrix} \left(\tau, \omega^{\mathrm{pp}}_{\mathcal{U}}\right), \\ \left(\mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, \omega^{\mathrm{dsp}}, \omega^{\mathrm{bl}}, \omega^{\mathrm{pp}}_{\mathcal{R}}\right) \end{pmatrix} \leftarrow \mathsf{DebtAccumulation}\left\langle \begin{array}{c} \mathcal{U}(\mathrm{pk}_{\mathcal{T}}, \mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}}), \\ \mathcal{R}^{\mathrm{O}_{\mathrm{pricing}}(\cdot,\cdot,\cdot)}(\mathrm{pk}_{\mathcal{T}}, \mathrm{cert}_{\mathcal{R}}, \mathrm{sk}_{\mathcal{R}}, bl_{\mathcal{R}}) \end{array} \right\rangle$$

and forward calls to the pricing oracle $\mathrm{O}_{\mathrm{pricing}}$ of the form $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}})$ to the adversary and pass the result p back.

(4) At the user side:
- Run the code of $\mathsf{WalletVerification}(\mathrm{pk}_{\mathcal{T}}, \mathrm{pk}_{\mathcal{U}}, \tau)$ (see Fig. 42).
- If WalletVerification returns 0, output $\perp$ and abort.
- Record τ and $\omega^{\mathrm{pp}}_{\mathcal{U}}$ internally.
- Parse s, $\mathrm{cert}_{\mathcal{R}}$, p and b from τ.
- Parse $\mathbf{a}_{\mathcal{R}}$ from $\mathrm{cert}_{\mathcal{R}}$.

(5) At the RSU side:
- Record ω^{dsp}, ω^{bl} and $\omega^{\mathrm{pp}}_{\mathcal{R}}$ internally.
- Parse s from $\omega^{\mathrm{pp}}_{\mathcal{R}}$.
- Parse ϕ from ω^{bl}.

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

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

Fig. 32. The UC-protocol π_{P4TC} (cont. from Fig. 18)

At the end of the protocol, the user returns the wallet τ. The TSP returns the serial number of this transaction and the hidden user trapdoor *htd*.

D.7 Debt Accumulation

When a driving car passes an RSU, the Debt Accumulation task is executed between $\mathcal{U}$ and $\mathcal{R}$. The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 32) and a core protocol DebtAccumulation (see Fig. 33). The wrapper protocol interacts with other UC entities, pre-processes the input, post-processes the output and lets the user execute the WalletVerification algorithm (see Fig. 42) after the core protocol has terminated. The

```
$\mathcal{U}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}})$
    parse $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$
    parse $(s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, c^{\mathrm{prev}}_{\mathcal{R}}, d^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}},$
        $c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathrm{prev}}, u_1) := \tau^{\mathrm{prev}}$
    parse $(\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{cert prev}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}{}^{\mathrm{prev}}$
    $\phi := \mathsf{PRF}(\lambda, x)$
    $s' \overset{R}{\leftarrow} G_1$
    $u^{\mathrm{next}}_1 \overset{R}{\leftarrow} \mathbb{Z}_p$
    $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda, b^{\mathrm{prev}}, u^{\mathrm{next}}_1, x))$

$\mathcal{R}^{\mathsf{O}_{\mathrm{pricing}}(\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, bl_{\mathcal{R}})$
    parse $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$
    parse $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$
    $s'' \overset{R}{\leftarrow} G_1$
    $u_2 \overset{R}{\leftarrow} \mathbb{Z}_p$
    $(c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) \leftarrow \mathsf{C2.Com}(\mathsf{CRS}^2_{\mathrm{com}}, s'')$

$\xleftarrow{u_2, c''_{\mathrm{ser}}, \mathsf{cert}_{\mathcal{R}}}$

$\mathcal{U}$:
    parse $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$
    if $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$
        return $\perp$
    $t := \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \bmod p$
    $(c_{\mathrm{hid}}, d_{\mathrm{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, \mathsf{sk}_{\mathcal{U}})$
    $stmnt := (\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t, u_2)$
    $wit := (x, \lambda, \mathsf{sk}_{\mathcal{U}}, u_1, s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, g^x_1, g^{\lambda}_1, \mathsf{pk}_{\mathcal{U}}, g^{b^{\mathrm{prev}}}_1, g^{u_1}_1, g^{u^{\mathrm{next}}_1}_1,$
        $d_{\mathrm{hid}}, d^{\mathrm{prev}}_{\mathcal{R}}, d'_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}, c^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{cert prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}})$
    $\pi \leftarrow \mathsf{P2.Prove}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, wit)$

$\xrightarrow{s', \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t}$

$\mathcal{R}$:
    $stmnt := (\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t, u_2)$
    if $\mathsf{P2.Vfy}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, \pi) = 0$
        return $\perp$
    if $\phi \in bl_{\mathcal{R}}$
        return blacklisted
    // obtains the price from the pricing oracle
    // based on $\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}$ and possibly
    // other public-verifiable, environmental information
    $p \leftarrow \mathsf{O}_{\mathrm{pricing}}(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}})$
    $s := s' \cdot s''$
    $(c''_{\mathcal{R}}, d''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (0, p, 0, 1))$
    $c_{\mathcal{R}} := c'_{\mathcal{R}} \cdot c''_{\mathcal{R}}$
    $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}_{\mathcal{R}}, (c_{\mathcal{R}}, s))$

$\xleftarrow{s'', d''_{\mathrm{ser}}, c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p}$

$\mathcal{U}$:
    if $\mathsf{C2.Open}(\mathsf{CRS}^2_{\mathrm{com}}, s'', c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) = 0$
        return $\perp$
    $s := s' \cdot s''$
    $d_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$
    $b := b^{\mathrm{prev}} + p$
    $x^{\mathrm{next}} := x + 1$
    $\tau := (s, \phi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u^{\mathrm{next}}_1)$
    $\omega^{\mathrm{pp}}_{\mathcal{U}} := (s, c_{\mathrm{hid}}, d_{\mathrm{hid}})$
    return $(\tau, \omega^{\mathrm{pp}}_{\mathcal{U}})$

$\mathcal{R}$:
    $\omega^{\mathrm{dsp}} := (\phi, t, u_2)$
    $\omega^{\mathrm{bl}} := (\phi, p)$
    $\omega^{\mathrm{pp}}_{\mathcal{R}} := (s, c_{\mathrm{hid}})$
    return $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, \omega^{\mathrm{dsp}}, \omega^{\mathrm{bl}}, \omega^{\mathrm{pp}}_{\mathcal{R}})$
```

Fig. 33. Debt Accumulation Core Protocol

wrapper protocol internally invokes the core protocol

$$\left(\begin{array}{c}\left(\tau, \omega_{\mathcal{U}}^{\mathrm{pp}}\right), \\ \left(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, \omega^{\mathrm{dsp}}, \omega^{\mathrm{bl}}, \omega_{\mathcal{R}}^{\mathrm{pp}}\right)\end{array}\right) \leftarrow \mathsf{DebtAccumulation}\left\langle\begin{array}{c}\mathcal{U}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}}), \\ \mathcal{R}^{\mathrm{O}_{\mathrm{pricing}}(\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, bl_{\mathcal{R}})\end{array}\right\rangle.$$

The user gets his public and private key, the public key of the TSP and his current wallet

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

as input. The RSU gets its own secret key and certificate, the public key of the TSP and the RSU blacklist $bl_{\mathcal{R}}$ as input. 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 WalletIssuing, 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 following protocol description.

The RSU starts the protocol 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 the randomness to calculate the double-spending tag $t := \mathsf{sk}_{\mathcal{U}} \cdot u_2 + u_1 \bmod \mathfrak{p}$. He then calculates the fraud detection ID for the current transaction as $\phi := \mathsf{PRF}(\lambda, x)$. The user then proceeds by preparing the updated wallet. Therefore, he first chooses a fresh double-spending mask u_1^{next} and executes $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}_{\mathrm{com}}^1, (\lambda, b^{\mathrm{prev}}, u_1^{\mathrm{next}}, x))$ to commit to his wallet ID, the current balance, the fresh double-spending mask and the current counter. He also executes $(c_{\mathrm{hid}}, d_{\mathrm{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}_{\mathrm{com}}^1, \mathsf{sk}_{\mathcal{U}})$ to create a fresh commitment on his secret user key. This commitment can be used at a later point in the Prove Participation task (cf. Figs. 36 and 37) 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 *stmnt* from the language $L_{\mathrm{gp}}^{(2)}$ defined by

$$L_{\mathrm{gp}}^{(2)} := \left\{ \begin{pmatrix} \mathsf{pk}_{\mathcal{T}}^{\mathcal{T}} \\ \mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}} \\ \phi \\ \mathbf{a}_{\mathcal{U}} \\ \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}} \\ c_{\mathrm{hid}} \\ c'_{\mathcal{R}} \\ t \\ u_2 \end{pmatrix}^{\top} \;\middle|\; \begin{array}{l} \exists\, x, \lambda, \mathsf{sk}_{\mathcal{U}}, u_1 \in \mathbb{Z}_{\mathfrak{p}};\; s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, X, \Lambda, \mathsf{pk}_{\mathcal{U}}, \\ B^{\mathrm{prev}}, U_1, U_1^{\mathrm{next}}, d_{\mathrm{hid}}, d_{\mathcal{R}}^{\mathrm{prev}}, d'_{\mathcal{R}}, d_{\mathcal{T}} \in G_1; \\ \mathsf{pk}_{\mathcal{R}}^{\mathrm{prev}} \in G_1^3;\; c_{\mathcal{R}}^{\mathrm{prev}}, c_{\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}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\mathrm{com}}^1, \mathsf{pk}_{\mathcal{U}}, c_{\mathrm{hid}}, d_{\mathrm{hid}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\mathrm{com}}^1, (\Lambda, B^{\mathrm{prev}}, U_1, X), c_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{prev}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\mathrm{com}}^1, (\Lambda, B^{\mathrm{prev}}, U_1^{\mathrm{next}}, X), c'_{\mathcal{R}}, d'_{\mathcal{R}}) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \sigma_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, (c_{\mathcal{R}}^{\mathrm{prev}}, s^{\mathrm{prev}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}}, \sigma_{\mathcal{R}}^{\mathrm{cert\,prev}}, (\mathsf{pk}_{\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{7}$$

This proof essentially shows that the wallet τ is valid, i.e., that the commitments $c_{\mathcal{T}}$ and $c_{\mathcal{R}}^{\mathrm{prev}}$ 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 c_{hid} is valid and contains the secret user key, that $c_{\mathcal{R}}^{\mathrm{prev}}$ and $c'_{\mathcal{R}}$ 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 $(\pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t)$ 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 $(c''_{\mathcal{R}}, d''_{\mathcal{R}}) := \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (0, p, 0, 1))$. Then it computes the new RSU commitment $c_{\mathcal{R}}$ by adding $c'_{\mathcal{R}}$ and $c''_{\mathcal{R}}$ (remember that C1 is homomorphic) and also signs it along with the serial number s. The RSU then sends $(c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to the user. It also stores several transaction information: The blacklisting transaction information $\omega^{\mathrm{bl}} := (\phi, p)$ can be used at a later point in the User Blacklisting task (cf. Fig. 40) to calculate the total fee of a fraudulent user. The double-spending transaction information $\omega^{\mathrm{dsp}} := (\phi, t, u_2)$ enables the TSP to identify the user if he uses the old state of the wallet (with unchanged balance) in another transaction (cf. Fig. 38). The RSU prove participation transaction information $\omega^{\mathrm{pp}}_{\mathcal{R}} := (s, c_{\mathrm{hid}})$ can be used later in the Prove Participation task (cf. Fig. 36).

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

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

The user's output is the updated wallet along with the user prove participation transaction information $\omega^{\mathrm{pp}}_{\mathcal{U}} := (s, c_{\mathrm{hid}}, d_{\mathrm{hid}})$. The RSU's output are the user's attributes $\mathbf{a}_{\mathcal{U}}$, the attributes $\mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}$ of the RSU from the previous transaction of the user and the three transaction information ω^{bl}, ω^{dsp} and $\omega^{\mathrm{pp}}_{\mathcal{R}}$.

D.8 Debt Clearance

After the end of a billing period, the Debt Clearance task is executed between $\mathcal{U}$ and $\mathcal{T}$. The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 34) and a core protocol DebtClearance (see Fig. 35). The wrapper protocol interacts with other UC entities, pre-processes the input and post-processes the output. The wrapper protocol internally invokes the core protocol

$$\left(\left(b^{\mathrm{bill}}\right), \left(\mathsf{pk}_{\mathcal{U}}, \omega^{\mathrm{bl}}, \omega^{\mathrm{dsp}}\right)\right) \leftarrow \mathsf{DebtClearance}\left\langle \mathcal{U}\left(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}}\right), \mathcal{T}\left(\mathsf{pk}_{\mathcal{T}}\right)\right\rangle.$$

The user gets the public key $\mathsf{pk}_{\mathcal{T}}$ of the TSP, his own public and private key $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ and his current wallet

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

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

The protocol is similar to the DebtAccumulation protocol, 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 DebtAccumulation, the TSP first sends a fresh double-spending randomness u_2 to the user and the user calculates the double-spending tag $t := \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \bmod \mathfrak{p}$ and the fraud detection ID ϕ for this transaction. He then continues by preparing a proof of knowledge. More precisely, P3 is used to compute a proof π for a statement

UC-Protocol π_{P4TC} (cont.) – Task *Debt Clearance*

User input: $(\texttt{clear\_debt}, s^{\text{prev}})$
TSP input: $(\texttt{clear\_debt})$

(1) At the user side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$.$^{\perp}$
- Receive $\mathsf{pk}_{\mathcal{T}}$ from the bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{T}}$.$^{\perp}$
- Load the internally recorded token τ^{prev} for serial number s^{prev}.$^{\perp}$

(2) At the TSP side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$.$^{\perp}$

(3) Both sides: Run the code of DebtClearance between the user and the TSP (see Fig. 35)

$$\Big((b^{\text{bill}}), (\mathsf{pk}_{\mathcal{U}}, \omega^{\text{bl}}, \omega^{\text{dsp}})\Big) \leftarrow \mathsf{DebtClearance}\Big\langle \mathcal{U}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\text{prev}}), \mathcal{T}(\mathsf{pk}_{\mathcal{T}}) \Big\rangle .$$

(4) At the TSP side:
- Record ω^{bl} and ω^{dsp} internally.
- Parse ϕ and $-b^{\text{bill}}$ from ω^{bl}.

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

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

Fig. 34. The UC-protocol π_{P4TC} (cont. from Fig. 18)

stmnt from the language $L^{(3)}_{\text{gp}}$ defined by

$$L^{(3)}_{\text{gp}} := \left\{ \begin{pmatrix} \mathsf{pk}_{\mathcal{U}} \\ \mathsf{pk}^{\mathcal{T}}_{\mathcal{T}} \\ \mathsf{pk}^{\text{cert}}_{\mathcal{T}} \\ \phi \\ \mathbf{a}_{\mathcal{U}} \\ \mathbf{a}^{\text{prev}}_{\mathcal{R}} \\ B^{\text{prev}} \\ t \\ u_2 \end{pmatrix}^{\top} \left| \begin{array}{l} \exists\, \lambda, x, u_1, \mathsf{sk}_{\mathcal{U}} \in \mathbb{Z}_p;\ \phi^{\text{prev}}, s^{\text{prev}}, X, \Lambda, U_1, \\ d^{\text{prev}}_{\mathcal{R}}, d_{\mathcal{T}} \in G_1;\ \mathsf{pk}^{\text{prev}}_{\mathcal{R}} \in G_1^3;\ c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}} \in G_2; \\ \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{cert}\,\text{prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}} \in G_2^2 \times G_1 : \\[4pt] \mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, \mathsf{pk}_{\mathcal{U}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, B^{\text{prev}}, U_1, X), c^{\text{prev}}_{\mathcal{R}}, d^{\text{prev}}_{\mathcal{R}}) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{prev}}_{\mathcal{R}}, (c^{\text{prev}}_{\mathcal{R}}, s^{\text{prev}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}\,\text{prev}}_{\mathcal{R}}, (\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}})) = 1 \\ \phi^{\text{prev}} = \mathsf{PRF}(\lambda, x-1),\ \phi = \mathsf{PRF}(\lambda, x), \\ t = \mathsf{sk}_{\mathcal{U}} u_2 + u_1 \\ \mathsf{pk}_{\mathcal{U}} = g_1^{\mathsf{sk}_{\mathcal{U}}},\ U_1 = g_1^{u_1},\ X = g_1^{x},\ \Lambda = g_1^{\lambda} \end{array} \right. \right\} \tag{8}$$

The proof is a simplified version of the one in the DebtAccumulation protocol. 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 $c'_{\mathcal{R}}$ and c_{hid}.

The user then sends $(\mathsf{pk}_{\mathcal{U}}, \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{prev}}, t)$ to the TSP. Unlike in DebtAccumulation, 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 protocol, the TSP outputs the user's public key $\mathsf{pk}_{\mathcal{U}}$, the blacklisting transaction information $\omega^{\text{bl}} := (\phi, -b^{\text{prev}})$ and the double-spending transaction information $\omega^{\text{dsp}} := (\phi, t, u_2)$. The user just outputs his final debt b^{prev} for this billing period.

$\mathcal{U}(\mathrm{pk}_{\mathcal{T}}, \mathrm{pk}_{\mathcal{U}}, \mathrm{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}})$		$\mathcal{T}(\mathrm{pk}_{\mathcal{T}})$
parse $(\mathrm{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathrm{pk}_{\mathcal{T}}^{\mathrm{cert}}, \mathrm{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathrm{pk}_{\mathcal{T}}$		parse $(\mathrm{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathrm{pk}_{\mathcal{T}}^{\mathrm{cert}}, \mathrm{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathrm{pk}_{\mathcal{T}}$
parse $(s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, \mathrm{cert}_{\mathcal{R}}{}^{\mathrm{prev}},$ $c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\mathrm{prev}}, u_1) := \tau^{\mathrm{prev}}$		
parse $(\mathrm{pk}_{\mathcal{R}}^{\mathrm{prev}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{cert}\,\mathrm{prev}}) := \mathrm{cert}_{\mathcal{R}}{}^{\mathrm{prev}}$ $\phi := \mathrm{PRF}(\lambda, x)$		
		$u_2 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathfrak{p}}$
	$\xleftarrow{u_2}$	
$t := \mathrm{sk}_{\mathcal{U}} u_2 + u_1 \bmod \mathfrak{p}$ $stmnt := (\mathrm{pk}_{\mathcal{U}}, \mathrm{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathrm{pk}_{\mathcal{T}}^{\mathrm{cert}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, g_1^{b^{\mathrm{prev}}}, t, u_2)$ $wit := (x, \lambda, \mathrm{sk}_{\mathcal{U}}, u_1, s^{\mathrm{prev}}, \phi^{\mathrm{prev}}, g_1^x, g_1^{\lambda}, g_1^{u_1}, d_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{T}},$ $\mathrm{pk}_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathcal{T}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{cert}\,\mathrm{prev}}, \sigma_{\mathcal{T}})$ $\pi \leftarrow \mathrm{P3.Prove}(\mathrm{CRS}_{\mathrm{pok}}, stmnt, wit)$		
	$\xrightarrow{\mathrm{pk}_{\mathcal{U}}, \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, b^{\mathrm{prev}}, t}$	

		$stmnt := (\mathrm{pk}_{\mathcal{U}}, \mathrm{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathrm{pk}_{\mathcal{T}}^{\mathrm{cert}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, g_1^{b^{\mathrm{prev}}}, t, u_2)$
		if $\mathrm{P3.Vfy}(\mathrm{CRS}_{\mathrm{pok}}, stmnt, \pi) = 0$
		    return $\perp$

	$\xleftarrow{\texttt{OK}}$	
$b^{\mathrm{bill}} := b^{\mathrm{prev}}$		$b^{\mathrm{bill}} := b^{\mathrm{prev}}$ $\omega^{\mathrm{bl}} := (\phi, -b^{\mathrm{bill}})$ $\omega^{\mathrm{dsp}} := (\phi, t, u_2)$
return (b^{bill})		return $(\mathrm{pk}_{\mathcal{U}}, \omega^{\mathrm{bl}}, \omega^{\mathrm{dsp}})$

Fig. 35. Debt Clearance Core Protocol

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 protocol 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.

D.9 Prove Participation

The Prove Participation task is used by a user to prove to the SA that he 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 protocol can be used to identify the fraudulent driver.[25] 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 he honestly participated in a corresponding RSU transaction is found guilty.

[25] 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.

UC-Protocol π_{P4TC} (cont.) – Task *Prove Participation*

User input: $(\texttt{prove\_participation})$
SA input: $(\texttt{prove\_participation}, \mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}})$

(1) At the SA side:
- Load the internally recorded set $\Omega^{\text{pp}}_{\mathcal{R}}$ of all prove participation transaction information with serial numbers in $S^{\text{pp}}_{\mathcal{R}}$.

(2) At the user side:
- Load the internally recorded set $\Omega^{\text{pp}}_{\mathcal{U}}$ of all prove participation transaction information.

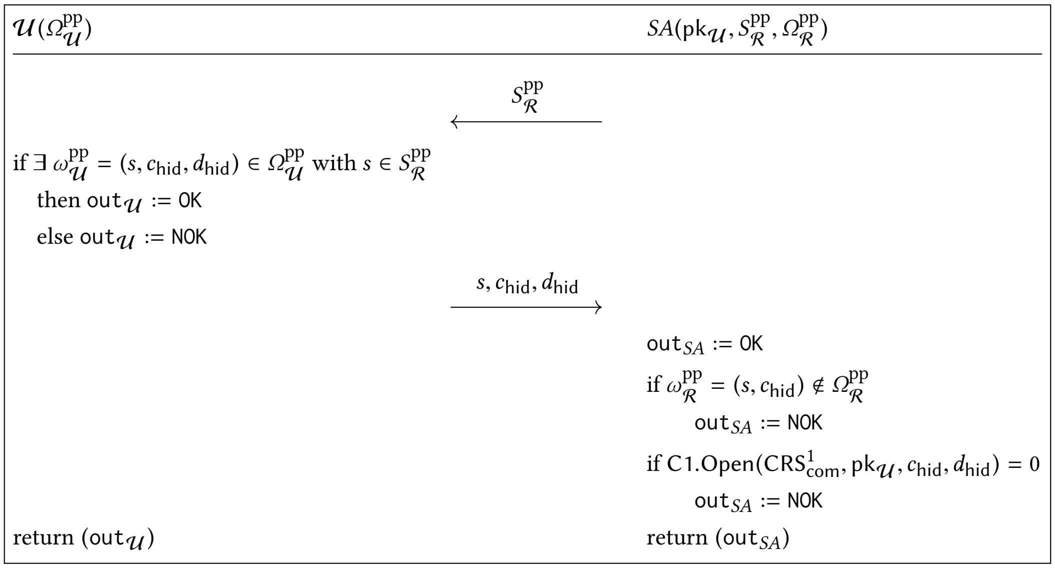
(3) Both sides: Run the code of ProveParticipation between the User and the SA (see Fig. 37)

$$((\mathsf{out}_{\mathcal{U}}), (\mathsf{out}_{SA})) \leftarrow \text{ProveParticipation}\left\langle \mathcal{U}(\Omega^{\text{pp}}_{\mathcal{U}}), SA(\mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}, \Omega^{\text{pp}}_{\mathcal{R}})\right\rangle.$$

User output: $(\mathsf{out}_{\mathcal{U}})$
SA output: (out_{SA})

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

Fig. 36. The UC-protocol π_{P4TC} (cont. from Fig. 18)



```
$\mathcal{U}(\Omega^{\text{pp}}_{\mathcal{U}})$                                        $SA(\mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}, \Omega^{\text{pp}}_{\mathcal{R}})$

                                   $\xleftarrow{S^{\text{pp}}_{\mathcal{R}}}$
if $\exists\, \omega^{\text{pp}}_{\mathcal{U}} = (s, c_{\text{hid}}, d_{\text{hid}}) \in \Omega^{\text{pp}}_{\mathcal{U}}$ with $s \in S^{\text{pp}}_{\mathcal{R}}$
    then $\mathsf{out}_{\mathcal{U}} := \mathsf{OK}$
    else $\mathsf{out}_{\mathcal{U}} := \mathsf{NOK}$
                                   $\xrightarrow{s, c_{\text{hid}}, d_{\text{hid}}}$
                                                          $\mathsf{out}_{SA} := \mathsf{OK}$
                                                          if $\omega^{\text{pp}}_{\mathcal{R}} = (s, c_{\text{hid}}) \notin \Omega^{\text{pp}}_{\mathcal{R}}$
                                                              $\mathsf{out}_{SA} := \mathsf{NOK}$
                                                          if $\text{C1.Open}(\text{CRS}^{1}_{\text{com}}, \mathsf{pk}_{\mathcal{U}}, c_{\text{hid}}, d_{\text{hid}}) = 0$
                                                              $\mathsf{out}_{SA} := \mathsf{NOK}$
return $(\mathsf{out}_{\mathcal{U}})$                                       return $(\mathsf{out}_{SA})$
```

Fig. 37. Prove Participation Core Protocol

The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 36) and a core protocol ProveParticipation (see Fig. 37). In the wrapper protocol, the SA interacts with other UC entities and processes the set $S^{\text{pp}}_{\mathcal{R}}$ of all serial numbers that were recorded by the RSU that took the photo at roughly the time the photo was taken. In particular, the SA loads the internally recorded set $\Omega^{\text{pp}}_{\mathcal{R}}$ of all RSU prove participation transaction information that contain serial numbers that are also in $S^{\text{pp}}_{\mathcal{R}}$. Afterwards, the wrapper protocol invokes the core protocol

$$((\mathsf{out}_{\mathcal{U}}), (\mathsf{out}_{SA})) \leftarrow \text{ProveParticipation}\left\langle \mathcal{U}(\Omega^{\text{pp}}_{\mathcal{U}}), SA(\mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}, \Omega^{\text{pp}}_{\mathcal{R}})\right\rangle.$$

UC-Protocol π_{P4TC} (cont.) – Task *Double-Spending Detection*

TSP input: $(\texttt{scan\_for\_fraud}, \phi)$

(1) Load the internally recorded set Ω^{dsp} of all double-spending transaction information.
(2) Pick transaction information $\omega^{\mathrm{dsp}} = (\phi, t, u_2)$ and $\omega^{\mathrm{dsp}\prime} = (\phi', t', u_2')$ from the database Ω^{dsp}, such that $\phi = \phi'$ and $u_2 \neq u_2'$.[⊥]
(3) $\mathsf{sk}_{\mathcal{U}} := (t - t') \cdot (u_2 - u_2')^{-1} \bmod \mathfrak{p}$.
(4) $\mathsf{pk}_{\mathcal{U}} := g_1^{\mathsf{sk}_{\mathcal{U}}}$.
(5) $\pi := \mathsf{sk}_{\mathcal{U}}$.

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

[⊥] If this does not exist, output $\perp$ and abort.

Fig. 38. The UC-protocol π_{P4TC} (cont. from Fig. 18)

The SA gets the user's public key $\mathsf{pk}_{\mathcal{U}}$, the set $S_{\mathcal{R}}^{\mathrm{pp}}$ of serial numbers that were observed roughly at the time the photograph was taken and the set of corresponding RSU prove participation transaction information $\Omega_{\mathcal{R}}^{\mathrm{pp}}$ as input. The user gets his internally recorded set $\Omega_{\mathcal{U}}^{\mathrm{pp}}$ of all prove participation transaction information as input.

The protocol itself is simple. The SA first sends $S_{\mathcal{R}}^{\mathrm{pp}}$ to the user who searches $\Omega_{\mathcal{U}}^{\mathrm{pp}}$ for a user prove participation transaction information $\omega_{\mathcal{U}}^{\mathrm{pp}}$ for which the serial number s in $\omega_{\mathcal{U}}^{\mathrm{pp}}$ is also in $S_{\mathcal{R}}^{\mathrm{pp}}$. If one is found, the user's output bit $\mathtt{out}_{\mathcal{U}}$ is set to $\mathtt{OK}$, if none is found, $\mathtt{out}_{\mathcal{U}}$ is set to $\mathtt{NOK}$ ("not ok"). The user then sends $\omega_{\mathcal{U}}^{\mathrm{pp}} := (s, c_{\mathsf{hid}}, d_{\mathsf{hid}})$ to the SA and the SA checks if $(s, c_{\mathsf{hid}}) \in \Omega_{\mathcal{R}}^{\mathrm{pp}}$ and if d_{hid} is the opening of c_{hid} under $\mathsf{pk}_{\mathcal{U}}$. If both checks succeed, the SA's output bit $\mathtt{out}_{\mathcal{SA}}$ is set to $\mathtt{OK}$, if at least one check fails, $\mathtt{out}_{\mathcal{SA}}$ is set to $\mathtt{NOK}$. At the end of the protocol both parties output their bits $\mathtt{out}_{\mathcal{U}}$ and $\mathtt{out}_{\mathcal{SA}}$, respectively.[26]

If the SA's output equals $\mathtt{NOK}$, the user is found guilty and appropriate measures are taken (e.g., the user gets blacklisted).

D.10 Double-Spending Detection

The double-spending transaction information ω^{dsp} collected by the RSUs are periodically transmitted to the TSP's database, which is regularly checked for two double-spending transaction information associated with the same serial number. If the database contains two such records, the Double-Spending Detection task (see Fig. 38) can be used by the TSP to extract the public key of the user these double-spending transaction information belong to as well as a proof (such as his secret user key) that the user is guilty.

In particular, the task gets a serial number s as input and searches the internal database for two double-spending transaction information $\omega^{\mathrm{dsp}} = (\phi, t, u_2)$ and $\omega^{\mathrm{dsp}\prime} = (\phi', t', u_2')$ that contain the same serial number $s = s'$ but not the same double-spending randomness $u_2 \neq u_2'$. Then the fraudulent user's secret key can be recovered as $\mathsf{sk}_{\mathcal{U}} := (t - t') \cdot (u_2 - u_2)^{-1} \bmod \mathfrak{p}$. His public key is then $\mathsf{pk}_{\mathcal{U}} := g_1^{\mathsf{sk}_{\mathcal{U}}}$. The secret key $\mathsf{sk}_{\mathcal{U}}$ can be used as a proof of guilt in the Guilt Verification task (cf. Fig. 39).

Every user that is convicted of double-spending is added to the blacklist via the User Blacklisting task (cf. Figs. 40 and 41) and additional measures are taken (these are out of scope).

[26] Note that after a successful (both parties output $\mathtt{OK}$) execution of the Prove Participation protocol a single transaction can be linked to the user. But as long as the user does not get guiltlessly photographed at every RSU he passes, tracking is not possible.

UC-Protocol π_{P4TC} (cont.) – Task *Guilt Verification*

Party input: $(\texttt{verify\_guilt}, \mathsf{pk}_{\mathcal{U}}, \pi)$

(1) Receive $pid_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for key $\mathsf{pk}_{\mathcal{U}}$.$^{\perp}$
(2) If $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$, then $\texttt{out} := \texttt{OK}$, else $\texttt{out} := \texttt{NOK}$.

Party output: $(\texttt{out})$

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

Fig. 39. The UC-protocol π_{P4TC} (cont. from Fig. 18)

D.11 Guilt Verification

Whether a user is indeed guilty of double-spending can be verified using the Guilt Verification task depicted in Fig. 39. 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 (cf. Fig. 38).

D.12 User Blacklisting

The User Blacklisting task executed between the DR and TSP is used to put a user on the blacklist. There are several reasons why a user is entered on the blacklist:

(1) A user did not submit his balance at the end of the billing period.
(2) A user did not physically pay his debt after submitting his 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 his 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).

The User Blacklisting task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 40) and a core protocol UserBlacklisting (see Fig. 41). The wrapper protocol interacts with other UC entities, pre-processes the input and afterwards internally invokes the core protocol

$$((\texttt{OK}), (\Phi_{\mathcal{U}})) \leftarrow \mathsf{UserBlacklisting}\left\langle DR(\mathsf{pk}_{DR}, \mathsf{sk}_{DR}, \mathsf{pk}_{\mathcal{U}}^{DR}), \mathcal{T}(HTD_{\mathcal{U}})\right\rangle .$$

The DR gets as input its public and private key $(\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$ together with the public key $\mathsf{pk}_{\mathcal{U}}^{DR}$ of the user to be blacklisted. The TSP gets a set $HTD_{\mathcal{U}}$ containing all his hidden user trapdoors from the current billing period as input.[27] We assume that the DR and TSP agreed upon which user is going to be blacklisted before the protocol out-of-band.

At the beginning of the protocol the TSP sends its input HTD to the DR. The DR then recovers the corresponding wallet ID λ for every $htd := (\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}, s, \lambda'', e^*) \in HTD$. To this end, the DR decrypts e^* to get $(\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}_{\mathcal{U}}^{\mathcal{T}})$. Firstly, the DR checks if the decrypted public key $\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}$ equals the expected public $\mathsf{pk}_{\mathcal{U}}^{DR}$ of the correct user in

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

UC-Protocol π_{P4TC} (cont.) – Task *User Blacklisting*

DR input: $(\texttt{blacklist\_user}, \mathsf{pk}_{\mathcal{U}}^{\mathcal{DR}})$

TSP input: $(\texttt{blacklist\_user}, \mathsf{pk}_{\mathcal{U}}^{\mathcal{T}})$

(1) At the DR side:
- Load the internally recorded $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})^{\perp}$.
- Receive $\mathsf{pk}_{\mathcal{U}}^{\mathcal{DR}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}^{\mathcal{DR}}$.$^{\perp}$

(2) At the TSP side:
- Load internally recorded set HTD of all hidden user trapdoors and set $HTD_{\mathcal{U}} := \{htd \mid (\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}, \cdot, \cdot, \cdot) \in HTD\}$.

(3) Both sides: Run the code of UserBlacklisting between the DR and the TSP (see Fig. 41)

$$((\mathtt{OK}), (\Phi_{\mathcal{U}})) \leftarrow \mathsf{UserBlacklisting}\left\langle \mathcal{DR}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}}, \mathsf{pk}_{\mathcal{U}}^{\mathcal{DR}}), \mathcal{T}(HTD_{\mathcal{U}}) \right\rangle .$$

(4) At the TSP side:
- Load the internally recorded set Ω^{bl} of all blacklisting transaction information.
- Let $\Omega_{\mathcal{U}}^{\text{bl}}$ be the subset of transaction entries $\omega^{\text{bl}} = (\phi, p)$ with fraud detection IDs $\phi \in \Phi_{\mathcal{U}}$.
- $b^{\text{bill}} := \sum_{\omega^{\text{bl}} \in \Omega_{\mathcal{U}}^{\text{bl}}} p$.

DR output: $(\mathtt{OK})$

TSP output: $(b^{\text{bill}}, \Phi_{\mathcal{U}})$

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

Fig. 40. The UC-protocol π_{P4TC} (cont. from Fig. 18)

```
$\mathcal{DR}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}}, \mathsf{pk}_{\mathcal{U}}^{\mathcal{DR}})$                                        $\mathcal{T}(HTD_{\mathcal{U}})$

                                    $\xleftarrow{\quad HTD_{\mathcal{U}} \quad}$
$\Phi_{\mathcal{U}} := \emptyset$
for all $htd \in HTD_{\mathcal{U}}$
    parse $(\mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}, s, \lambda'', e^*) := htd$
    $(\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}) \leftarrow \mathsf{E1.Dec}(\mathsf{sk}_{\mathcal{DR}}, e^*)$
    if decryption fails $\vee\ \Lambda'' \neq g_1^{\lambda''} \vee \mathsf{pk}_{\mathcal{U}}^{\mathcal{DR}} \neq \mathsf{pk}_{\mathcal{U}}^{\mathcal{T}}$
        return $\perp$
    $\lambda := \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$
    $\Phi_{\mathcal{U}} \leftarrow \Phi_{\mathcal{U}} \cup \{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\text{bl}_{\mathcal{R}}})\}$
                                    $\xrightarrow{\quad \Phi_{\mathcal{U}} \quad}$
return $(\mathtt{OK})$                                                   return $(\Phi_{\mathcal{U}})$
```

Fig. 41. User Blacklisting Core Protocol

question.[28] 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 $\lambda = \lambda'' + \sum_{i=0}^{\ell-1} \lambda'_i \cdot \mathcal{B}^i$. Finally, the DR sends the union of all sets $\{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\mathrm{bl}_\mathcal{R}})\}$ of fraud detection IDs for every $htd \in HTD$ to the TSP. 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 DR's output of the core protocol is simply $\mathtt{OK}$, while the TSP's output is a set of fraud detection IDs.

After the core protocol has terminated, the TSP calculates the total toll amount the user owes for the billing period in question in the wrapper protocol. To this end, the TSP calculates the subset $\Omega^{\mathrm{bl}}_{\mathcal{U}} \subseteq \Omega^{\mathrm{bl}}$ of all blacklisting transaction information that correspond to the unveiled fraud detection IDs in $\Phi_{\mathcal{U}}$. By summing up all the prices in $\Omega^{\mathrm{bl}}_{\mathcal{U}}$, the TSP can calculate the total fee b^{bill} the user owes.

After the wrapper protocol has terminated, the fraud detection IDs $\Phi_{\mathcal{U}}$ are added to the RSU blacklist $bl_\mathcal{R}$ and the user's public key $\mathrm{pk}_\mathcal{U}$ is added on the TSP blacklist $bl_\mathcal{T}$.

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 $bl_\mathcal{R}$.

D.13 Wallet Verification

The WalletVerification algorithm is depicted in Fig. 42. A user can verify with this algorithm that the wallet he stores at the end of a transaction is valid. In particular, the algorithm verifies that the commitments $c_\mathcal{T}$ and $c_\mathcal{R}$ are valid and contain the values they are supposed to contain, that $\sigma_\mathcal{T}$ is a valid signature under $\mathrm{sk}^\mathcal{T}_\mathcal{T}$ of $c_\mathcal{T}$ and $\mathbf{a}_\mathcal{U}$, that $\sigma_\mathcal{R}$ is a valid signature under $\mathrm{sk}_\mathcal{R}$ of $c_\mathcal{R}$ and s, that the certificate $\mathrm{cert}_\mathcal{R}$ containing $\mathrm{pk}_\mathcal{R}$ 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).

E ADVERSARIAL MODEL

For our security analysis to hold we consider a restricted class of adversarial environments $\mathcal{Z}$ and will argue why these restrictions are reasonable.

E.1 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 only comes into play with deniability which is not part of our desired properties.

[28] N.b.: The public key $\mathrm{pk}^\mathcal{T}_\mathcal{U}$ is stored redundantly: in clear as part of $htd := (\mathrm{pk}^\mathcal{T}_\mathcal{U}, s, \lambda'', e^*)$ and encrypted as part of the cipher text $e^* = \mathsf{Enc}(\Lambda'_0, \ldots, \Lambda'_{\ell-1}, \Lambda'', \mathrm{pk}^\mathcal{T}_\mathcal{U})$. 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 $\mathrm{pk}^\mathcal{T}_\mathcal{U}$.

```
$\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$
parse $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$
parse $(s, \phi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathrm{next}}) := \tau$
parse $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$
if
    $\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, (g_1^{\lambda}, \mathsf{pk}_{\mathcal{U}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}})) = 0 \vee$
    $\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, (g_1^{\lambda}, g_1^{b}, g_1^{u_1^{\mathrm{next}}}, g_1^{x^{\mathrm{next}}}), c_{\mathcal{R}}, d_{\mathcal{R}}) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (c_{\mathcal{R}}, s)) = 0 \vee$
    $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0 \vee$
    $\mathsf{PRF}(\lambda, x^{\mathrm{next}} - 1) \neq \phi$
then return 0
else return 1
```

Fig. 42. Algorithm for Wallet Verification

Secondly, we only consider adversaries $\mathcal{Z}$ that corrupt one of the following sets[29]:

(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.

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

[29] Note that "subset" also includes the empty or full set.

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 counter measures 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.

E.2 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.

E.3 Handling of Aborts

Lastly, we assume our functionality also uses the implicit writing conventions for ideal functionalities [20]. 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.

Another important aspect with respect to aborts is that privacy may be partially lost for an honest user if a task aborts prematurely. In this case the user's identity can be unveiled if he chooses to take part in another transaction with the same wallet. The reason for this is the double-spending detection mechanism. If the (honest) user has not correctly updated his previous state, the user must start the next interaction from the same state as before. Thus an (honest) user can be tricked into committing a double-spending without any fault of his own. We explicitly model this artifact into the simulator.

Again, as in Appendix E.1 this kind of "attack" does not require to break any cryptographic assumptions. Hence, the UC notion as well as any other cryptographic security notion does not take aborts into account—the more so as aborts can occur at any time due to technical problems. To mitigate the effect of aborts we propose that each user has one or more backup wallets and switches over to another wallet if the principal wallet becomes unusable and the user does not accept to become linkable for a single transaction. Of course, at the end of a billing period a user must always clear all of his wallets. If aborts occur more frequently than one would reasonably expect due to technical problems and the RSU/TSP is suspected to purposely abort in order to lift privacy, the user (or some NGO) is expected to file a claim. However, these are non-technical countermeasures.

F SECURITY PROOF

In this appendix we show that π_{P4TC} UC-realizes $\mathcal{F}_{\mathrm{P4TC}}$ in the $(\mathcal{F}_{\mathrm{CRS}}, \overline{\mathcal{G}}_{\mathrm{bb}})$-hybrid model for static corruption. More precisely, we show the following theorem:

Theorem F.1 (Security Statement). *Assume that the SXDH-problem is hard for* gp $:= (G_1, G_2, G_T, e, p, g_1, g_2)$, *the Co-CDH problem is hard for* (G_1, G_2), *the* n_{PRF}*-DDHI problem is hard for* G_1, *and the DLOG-problem is hard for* G_1 *and our building blocks (NIZK, commitment schemes, signature scheme, encryption schemes and PRF) are instantiated as described in Appendix C.2. Then*

$$\pi_{P4TC}^{\mathcal{F}_{CRS}, \overline{\mathcal{G}}_{bb}} \geq_{UC} \mathcal{F}^{\overline{\mathcal{G}}_{bb}},$$

holds under static corruption of

(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.

Please note, that the hardness of the Co-CDH problem and DLOG-problem is already implied by the SXDH-assumption. For a discussion of the reasons and why this limited corruption model is not a severe restriction from a practical vantage point see Appendix E.1.

We prove Theorem F.1 in three steps:

- In Appendix F.1 we first show some structural properties of $\mathcal{F}^{\overline{\mathcal{G}}_{\text{bb}}}$.
- In Appendix F.2, Theorem F.10 proves Theorem F.1 for the corruption scenario (1) and (2). We call this case "Operator Security".
- In Appendix F.3, Theorem F.25 proves Theorem F.1 for the corruption scenario (3) and (4). We call this case "User Security and Privacy".

Proof of Theorem F.1. The theorem is immediately implied by Theorems F.10 and F.25. □

Before giving the proof in full detail and complexity in Appendices F.1 to F.3, Section 5 explains our course of action and outlines the proof on a high level.

F.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 constrast, 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 section 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 a closely associated to the desired properties of a toll collection scheme (cp. Sections 1.5.3 and 3.5).

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 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. 43 for a depiction. Transaction entries or nodes that are inserted by Wallet Issuing do not have a predecessor, therefore $s^{\text{prev}} = \perp$ 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

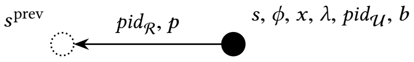


Fig. 43. An entry $trdb \in TRDB$ visualized as an element of a directed graph

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 F.2 (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 F.3 (Forest Structure of the Ideal Transaction Graph). *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 F.4 (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 " $\Longleftarrow$ " it 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 F.5. *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 F.4. □

In other words, Lemma F.5 states that a wallet (a tree in *TRDB*) is always owned by a distinct user. But a user can own multiple wallets.

Lemma F.6. *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 $x := 0$ and an unused ϕ is chosen. In the scope of Debt Accumulation or Debt Clearance, x is calculated as $x := x^{\text{prev}} + 1$, 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 User Blacklisting) it is chosen from the set of *unused* fraud detection IDs and therefore unique for given λ and x. □

Lemma F.7 (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 proof of Lemma F.6. □

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 F.8 (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. Fig. 14). 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 Fig. 15). Guilt Verification only returns `OK`, if $f_{\Pi}(pid_{\mathcal{U}}, \pi) = \texttt{OK}$ 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 $f_{\Pi}(pid_{\mathcal{U}}, \pi) = \texttt{OK}$ (cp. Step 2, in Fig. 14). □

Lemma F.9 (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 $\mathcal{T}$ by User Blacklisting 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. (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 $x_{\lambda} := \max\{x \mid f_{\Phi}(\lambda, x) \neq \perp\}$. If $\mathcal{U}$ with $pid_{\mathcal{U}}$ participated in less than $x_{\text{bl}_{\mathcal{R}}}$ transaction, then the maximum depth $x_{\max} = \max_{\lambda \in \mathcal{L}_{\mathcal{U}}} x_{\lambda}$ is smaller than $x_{\text{bl}_{\mathcal{R}}}$. 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 $\Phi_{\mathcal{U}} := \{f_{\Phi}(\lambda, x) \mid \lambda \in \mathcal{L}_{\mathcal{U}}, 0 \leq x \leq x_{\text{bl}_{\mathcal{R}}}\}$.
(2) Let s^{prev} denote the serial number for which Debt Accumulation is invoked and let $trdb^{\text{prev}} = (\cdot, s^{\text{prev}}, \phi^{\text{prev}}, x^{\text{prev}}, \lambda, \ldots)$ be the corresponding transaction entry. By assumption $x^{\text{prev}} < x_{\text{bl}_{\mathcal{R}}}$ holds. As User Blacklisting has previously been called, $\phi = f_{\Phi}(\lambda, x^{\text{prev}} + 1)$ is already fixed. Moreover, $\phi \in \Phi_{\mathcal{U}} = bl_{\mathcal{R}}$ holds and thus Debt Accumulation aborts.

□

Simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$

Setup:

(1) Run a modified version of the algorithm $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^n)$ with
   (a) $\mathsf{CRS}^2_{\mathrm{com}} \leftarrow \mathsf{C2.Gen}$ being replaced by $(\mathsf{CRS}^2_{\mathrm{com}}, \mathsf{td}_{\mathrm{eqcom}}) \leftarrow \mathsf{C2.SimGen}$, and
   (b) $\mathsf{CRS}_{\mathrm{pok}} \leftarrow \mathsf{SetupPoK}$ being replaced by $(\mathsf{CRS}_{\mathrm{pok}}, \mathsf{td}_{\mathrm{epok}}) \leftarrow \mathsf{SetupEPoK}$.
(2) Record CRS, $\mathsf{td}_{\mathrm{eqcom}}$ and $\mathsf{td}_{\mathrm{epok}}$.
(3) Set $\overline{TRDB} := \emptyset$.
(4) Set $\overline{\Omega}^{\mathrm{dsp}} := \emptyset$.
(5) Set $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}} := \emptyset$.

DR Registration: Upon receiving $(\texttt{registering\_dr}, pid_{DR})$ run $(\mathsf{pk}_{DR}, \mathsf{sk}_{DR}) \leftarrow \mathsf{DRRegistration}(\mathsf{CRS})$, record $pid_{DR} \mapsto (\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$, and return pk_{DR} to $\mathcal{F}_{\text{P4TC}}$.

TSP Registration: Upon receiving $(\texttt{registering\_tsp}, pid_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}})$ run $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{TSPRegistration}(\mathsf{CRS}, \mathbf{a}_{\mathcal{T}})$, record $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and return $\mathsf{pk}_{\mathcal{T}}$ to $\mathcal{F}_{\text{P4TC}}$.

RSU Registration: Upon receiving $(\texttt{registering\_rsu}, pid_{\mathcal{R}})$ run $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{RSURegistration}(\mathsf{CRS})$, and record $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$, and return $\mathsf{pk}_{\mathcal{R}}$ to $\mathcal{F}_{\text{P4TC}}$.

User Registration: If user is corrupted, then nothing to do, as this is a local algorithm.
If user is honest: Upon receiving $(\texttt{registering\_user}, pid_{\mathcal{U}})$ run $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \leftarrow \mathsf{UserRegistration}(\mathsf{CRS})$, record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and return $\mathsf{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.

RSU Certification: Upon receiving $(\texttt{certifying\_rsu}, pid_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}) \ldots$

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
(2) Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}})$ with $\sigma^{\mathrm{cert}}_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$ faithfully.
(3) Update record $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$.

Fig. 44. The simulator for Operator Security

F.2 Proof of Operator Security

In this section we show the following theorem.

Theorem F.10 (Operator Security). *Under the assumptions of Theorem F.1*

$$\pi_{P4TC}^{\mathcal{F}_{CRS}, \overline{\mathcal{G}}_{bb}} \geq_{UC} \mathcal{F}^{\overline{\mathcal{G}}_{bb}}$$

holds under static corruption of

(1) a subset of users, or
(2) all users and a subset of RSUs, TSP and SA.

The definition of the UC-simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ for Theorem F.10 can be found in Figs. 44 to 47. Please note that while the real protocol π_{P4TC} lives in the $(\mathcal{F}_{\text{CRS}}, \overline{\mathcal{G}}_{\text{bb}})$-model the ideal functionality $\mathcal{F}_{\text{P4TC}}$ has no CRS. Hence, the CRS (but not the bulletin board) is likewise simulated by $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$, 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 $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ records certain information similar to what the parties or the ideal functionality internally record, namely the set of simulated transaction information for double-spending detection $\overline{\Omega}^{\mathrm{dsp}}$, the set of simulated transaction information for prove participation $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}}$, and the simulated

Simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$

```
Wallet Issuing:
  (1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, $pid_{DR} \mapsto (\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
  (2) Upon receiving $(\mathsf{pk}_{\mathcal{U}}, \underline{c'_{\text{seed}}})$ from $\mathcal{Z}^{\text{sys-sec}}$ in the name of $\mathcal{U}$ with $pid^*_{\mathcal{U}}$ ...
    (a) Look up $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for which $\mathsf{pk}_{\mathcal{U}}$ has been recorded; if no $pid_{\mathcal{U}}$ exists abort.
    (b) If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ give up simulation.
    (c) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{issue})$
  (3) Upon receiving leaked $(s, \mathbf{a}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ ...
    (a) $(c''_{\text{ser}}, \overline{d}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{CRS}^2_{\text{com}})$.
    (b) $\lambda'' \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_{\mathrm{p}}$.
    (c) Send $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}, c''_{\text{ser}}, \lambda'')$ to $\mathcal{Z}^{\text{sys-sec}}$ as the 1st message from $\mathcal{T}$ to $\mathcal{U}$.
  (4) Upon receiving $(s', e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, \pi)$ from $\mathcal{Z}^{\text{sys-sec}}$ in the name of $\mathcal{U}$ with $pid^*_{\mathcal{U}}$ ...
    (a) $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{DR}, e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, \pi)$.
    (b) If $\mathsf{P1.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.
    (c) Extract $Wit = (\Lambda, \Lambda', \Lambda'_0, \dots, \Lambda'_{\ell-1}, \dots, U^{\text{next}}_1, d_{\mathcal{T}}, d_{\mathcal{R}}, d'_{\text{seed}}, \mathsf{SK}_{\mathcal{U}}) \leftarrow \mathsf{P1.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.
    (d) Assert that $(stmnt, Wit)$ fullfills the projected equations from $L^{(1)}_{\text{gp}}$, else abort (event $E1$)
    (e) $\lambda := \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$
    (f) Provide alternative user PID $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.
  (5) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide $\phi$ ...
    (a) $\phi := \mathsf{PRF}(\lambda, x)$ with $x := 0$
    (b) Provide $\phi$ to $\mathcal{F}_{\text{P4TC}}$.
  (6) Upon receiving output $(s, \mathbf{a}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ for $\mathcal{U}$ ...
    (a) $s'' := s \cdot s'^{-1}$
    (b) Equivoke $d''_{\text{ser}} \leftarrow \mathsf{C2.Equiv}(\mathsf{CRS}^2_{\text{com}}, \mathsf{td}_{\text{eqcom}}, s'', c''_{\text{ser}}, \overline{d}_{\text{ser}})$.
    (c) $\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}))$
    (d) $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (c_{\mathcal{R}}, s))$
    (e) Set $s^{\text{prev}} := \bot$, $p := 0$, $b := 0$, $c^{\text{in}}_{\mathcal{T}} := \bot$, $d^{\text{in}}_{\mathcal{T}} := \bot$, $M^{\text{in}}_{\mathcal{T}} := \bot$, $c^{\text{in}}_{\mathcal{R}} := \bot$, $d^{\text{in}}_{\mathcal{R}} := \bot$, $M^{\text{in}}_{\mathcal{R}} := \bot$, $c^{\text{out}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{out}}_{\mathcal{T}} := d_{\mathcal{T}}$, $M^{\text{out}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$, $c^{\text{out}}_{\mathcal{R}} := c_{\mathcal{R}}$, $d^{\text{out}}_{\mathcal{R}} := d_{\mathcal{R}}$, and $M^{\text{out}}_{\mathcal{R}} := (\Lambda, g_1^b, U_1, g_1^{x+1})$.
    (f) Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, p, b, c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}, c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})$ to $\overline{TRDB}$.
    (g) Send $(s'', d''_{\text{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}})$ to $\mathcal{Z}^{\text{sys-sec}}$ as the 2nd message from $\mathcal{T}$ to $\mathcal{U}$.
```

Fig. 45. The simulator for Operator Security (cont. from Fig. 44)

transaction graph $\overline{TRDB}$. Basically, $\overline{\Omega}^{\text{dsp}}$, $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ and $\overline{TRDB}$ correspond to Ω^{dsp}, $\Omega^{\text{pp}}_{\mathcal{U}}$ 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 $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ as $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ and $\overline{TRDB}$ and inside the ideal functionality $\mathcal{F}_{\text{P4TC}}$ (in case of $TRDB$)

Simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Debt Accumulation:

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Pick $u_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\mathfrak{p}}$.

(3) $(c''_{\text{ser}}, \overline{d}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{CRS}^2_{\text{com}})$.

(4) Send $(u_2, c''_{\text{ser}}, \mathsf{cert}_{\mathcal{R}})$ to $\mathcal{Z}$ as the 1st message from $\mathcal{R}$ to $\mathcal{U}$.

(5) Upon receiving $(s', \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ from $\mathcal{Z}$ as the 2nd message from $\mathcal{U}$ to $\mathcal{R}$ …

(a) $stmnt := (\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t, u_2)$.

(b) If $\mathsf{P2.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.

(c) Extract $Wit = (X, \Lambda, \mathsf{pk}_{\mathcal{U}}, U_1, s^{\text{prev}}, \phi^{\text{prev}}, X, \Lambda, \mathsf{pk}_{\mathcal{U}}, B^{\text{prev}}, U_1, U_1^{\text{next}}, d_{\text{hid}}, d^{\text{prev}}_{\mathcal{R}}, d'_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\text{prev}}_{\mathcal{R}}, c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{certprev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.

(d) Assert that $(stmnt, Wit)$ fullfills the projected equations from $L^{(2)}_{\text{gp}}$, else abort (event *E1*)

(e) Look up $\overline{trdb}^* := (s^{\text{prev},*}, s^*, \phi^*, x^*, \lambda^*, pid^*_{\mathcal{U}}, pid^*_{\mathcal{T}}, p^*, b^*, c^{\text{in}\,*}_{\mathcal{T}}, d^{\text{in}\,*}_{\mathcal{T}}, M^{\text{in}\,*}_{\mathcal{T}}, c^{\text{in}\,*}_{\mathcal{R}}, d^{\text{in}\,*}_{\mathcal{R}}, M^{\text{in}\,*}_{\mathcal{R}}, c^{\text{out}*}_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, c^{\text{out}*}_{\mathcal{R}}, d^{\text{out}*}_{\mathcal{R}}, M^{\text{out}*}_{\mathcal{R}})$ 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: $c^{\text{out}*}_{\mathcal{R}} \neq c^{\text{prev}}_{\mathcal{R}}$ (event *E3*), $\Lambda \neq g_1^{\lambda^*}$ (event *E4*), $c^{\text{out}*}_{\mathcal{T}} \neq c_{\mathcal{T}}$ (event *E5*), $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}^*_{\mathcal{U}}$ with $(\Lambda^*, \mathsf{pk}^*_{\mathcal{U}}) := M^{\text{out}*}_{\mathcal{T}}$ (event *E6*), $B^{\text{prev}} \neq g_1^{b^*}$ (event *E7*), or $X \neq g_1^{x^*+1}$ (event *E8*).

(g) Retrieve $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathsf{pk}_{\mathcal{U}}$

(h) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{pay\_toll}, s^{\text{prev}})$ in the name of $\mathcal{U}$

(6) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide an alternative PID, return $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.

(7) If being asked by $\mathcal{F}_{\text{P4TC}}$ to provide ϕ, return $\phi := \mathsf{PRF}(\lambda, x)$ with $x := x^* + 1$ to $\mathcal{F}_{\text{P4TC}}$.[a]

(8) Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide a price for $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}})$, return a price p as the dummy adversary would do.

(9) Upon receiving output $(s, \mathbf{a}_{\mathcal{R}}, p, b)$ from $\mathcal{F}_{\text{P4TC}}$ for $\mathcal{U}$ …

(a) Set $\overline{\omega}^{\text{pp}}_{\mathcal{U}} := (s, c_{\text{hid}}, d_{\text{hid}}, \mathsf{pk}_{\mathcal{U}})$ and append $\overline{\omega}^{\text{pp}}_{\mathcal{U}}$ to $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$.

(b) Run $(c''_{\mathcal{R}}, d''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, p, 0, 1))$, $c_{\mathcal{R}} := c'_{\mathcal{R}} \cdot c''_{\mathcal{R}}$, and $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}_{\mathcal{R}}, (c_{\mathcal{R}}, s))$ honestly as the real protocol would do.

(c) Set $s'' := s \cdot s'^{-1}$ and equivoke $d''_{\text{ser}} \leftarrow \mathsf{C2.Equiv}(\mathsf{CRS}^2_{\text{com}}, s'', c''_{\text{ser}}, \overline{d}_{\text{ser}})$.

(d) Set $c^{\text{in}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{in}}_{\mathcal{T}} := d_{\mathcal{T}}$, $M^{\text{in}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$, $c^{\text{in}}_{\mathcal{R}} := c^{\text{prev}}_{\mathcal{R}}$, $d^{\text{in}}_{\mathcal{R}} := d^{\text{prev}}_{\mathcal{R}}$, $M^{\text{in}}_{\mathcal{R}} := (\Lambda, B^{\text{prev}}, U_1, X)$, $c^{\text{out}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{out}}_{\mathcal{T}} := d'_{\mathcal{T}} \cdot d''_{\mathcal{T}}$, $M^{\text{out}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$, $c^{\text{out}}_{\mathcal{R}} := c_{\mathcal{R}}$, $d^{\text{out}}_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$, and $M^{\text{out}}_{\mathcal{R}} := (\Lambda, g_1^b, U_1, g_1^{x+1})$.

(e) Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b, c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}, c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})$ to $\overline{TRDB}$.

(f) Append $\overline{\omega}^{\text{dsp}} = (\phi, t, u_2)$ to $\overline{\Omega}^{\text{dsp}}$

(g) Check if $\overline{\omega}^{\text{dsp}\,\ddagger} = (\phi^\ddagger, t^\ddagger, u_2^\ddagger) \in \overline{\Omega}^{\text{dsp}}$ exists with $\phi = \phi^\ddagger$ and $u_2 \neq u_2^\ddagger$; in this case

(i) $\mathsf{sk}_{\mathcal{U}} := (t - t^\ddagger) \cdot (u_2 - u_2^\ddagger)^{-1} \bmod \mathfrak{p}$

(ii) Record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ internally

(h) Send $(s'', d''_{\text{ser}}, c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to $\mathcal{Z}$ as the 3rd message from $\mathcal{R}$ to $\mathcal{U}$.

[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.

Fig. 46. The simulator for Operator Security (cont. from Fig. 44)

Simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Debt Clearance:

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$ if this does not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Pick $u_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\mathfrak{p}}$.

(3) Send u_2 to $\mathcal{Z}$ as the 1st message from $\mathcal{T}$ to $\mathcal{U}$.

(4) Upon receiving $(\mathsf{pk}_{\mathcal{U}}, \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{prev}}, t)$ from $\mathcal{Z}$ as the 2nd message from $\mathcal{U}$ to $\mathcal{T}$ …

- (a) $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{prev}}}, t, u_2)$.
- (b) If $\mathsf{P3.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.
- (c) Extract $Wit = (X, \Lambda, \mathsf{pk}_{\mathcal{U}}, U_1, s^{\text{prev}}, \phi^{\text{prev}}, X, \Lambda, U_1, d^{\text{prev}}_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\text{prev}}_{\mathcal{R}}, c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{cert}\,\text{prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) \leftarrow \mathsf{P3.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.
- (d) Assert that $(stmnt, Wit)$ fullfills the projected equations from $L^{(3)}_{\text{gp}}$, else abort (event *E1*)
- (e) Lookup $\overline{trdb}^{*} := (s^{\text{prev},*}, s^*, \phi^*, x^*, \lambda^*, pid^*_{\mathcal{U}}, pid^*_{\mathcal{T}}, p^*, b^*, c^{\text{in}\,*}_{\mathcal{T}}, d^{\text{in}\,*}_{\mathcal{T}}, M^{\text{in}\,*}_{\mathcal{T}}, c^{\text{in}*}_{\mathcal{R}}, d^{\text{in}*}_{\mathcal{R}}, M^{\text{in}*}_{\mathcal{R}}, c^{\text{out}*}_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, c^{\text{out}*}_{\mathcal{R}}, d^{\text{out}*}_{\mathcal{R}}, M^{\text{out}*}_{\mathcal{R}})$ 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: $c^{\text{out}*}_{\mathcal{R}} \neq c^{\text{prev}}_{\mathcal{R}}$ (event *E3*), $\Lambda \neq g_1^{\lambda^*}$ (event *E4*), $c^{\text{out}*}_{\mathcal{T}} \neq c_{\mathcal{T}}$ (event *E5*), $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}^*_{\mathcal{U}}$ with $(\Lambda^*, \mathsf{pk}^*_{\mathcal{U}}) := M^{\text{out}*}_{\mathcal{T}}$ (event *E6*), or $B^{\text{prev}} \neq g_1^{b^*}$ (event *E7*).
- (g) Retrieve $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathsf{pk}_{\mathcal{U}}$
- (h) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{clear\_debt}, s^{\text{prev}})$ in the name of $\mathcal{U}$

(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 $\phi := \mathsf{PRF}(\lambda, x)$ with $x := x^{\text{prev}} + 1$ to $\mathcal{F}_{\text{P4TC}}$.[a]

(7) Upon receiving output (b^{bill}) from $\mathcal{F}_{\text{P4TC}}$ for $\mathcal{U}$ …

- (a) Set $c^{\text{in}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{in}}_{\mathcal{T}} := d_{\mathcal{T}}$, $M^{\text{in}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$, $c^{\text{in}}_{\mathcal{R}} := c^{\text{prev}}_{\mathcal{R}}$, $d^{\text{in}}_{\mathcal{R}} := d^{\text{prev}}_{\mathcal{R}}$, $M^{\text{in}}_{\mathcal{R}} := (\Lambda, g_1^{b^{\text{prev}}}, U_1, X)$, $c^{\text{out}}_{\mathcal{T}} := \bot$, $d^{\text{out}}_{\mathcal{T}} := \bot$, $M^{\text{out}}_{\mathcal{T}} := \bot$, $c^{\text{out}}_{\mathcal{R}} := \bot$, $d^{\text{out}}_{\mathcal{R}} := \bot$, and $M^{\text{out}}_{\mathcal{R}} := \bot$.
- (b) Append $(s^{\text{prev}}, s, \phi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b, c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}, c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})$ 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) $\mathsf{sk}_{\mathcal{U}} := (t - t^{\ddagger}) \cdot (u_2 - u_2^{\ddagger})^{-1} \bmod \mathfrak{p}$
  - (ii) Record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ internally
- (e) Send (OK) to $\mathcal{Z}$ as the 3rd message from $\mathcal{T}$ to $\mathcal{U}$.

[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.

Fig. 47. The simulator for Operator Security (cont. from Fig. 44)

or the environment (in case of $\Omega^{\text{pp}}_{\mathcal{U}}$ for a corrupted user).[30] A crucial part of the security proof is to show that these sets stay in sync.

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 F.2) in more details. The *Ideal Transaction Graph*

[30] Note, that we get rid of Ω^{dsp} during the proof, because honest user are only dummy parties and only $\overline{\Omega}^{\text{dsp}}$ remains.

Simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Prove Participation:

(1) Call $\mathcal{F}_{\text{P4TC}}$ with input (`prove_participation`) in the name of $\mathcal{U}$
(2) Obtain leaked set $S^{\text{pp}}_{\mathcal{R}}$ of serial numbers from $\mathcal{F}_{\text{P4TC}}$.
(3) Upon receiving output $(\mathtt{out}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ for $\mathcal{U}$ …
  (a) Delay the output of $\mathcal{F}_{\text{P4TC}}$ to SA.
  (b) Send $S^{\text{pp}}_{\mathcal{R}}$ to $\mathcal{Z}$ as the 1st message from SA to $\mathcal{U}$.
(4) Upon receiving $(s, c_{\text{hid}}, d_{\text{hid}})$ from $\mathcal{Z}$ as the 2nd message from $\mathcal{U}$ to SA …
  (a) If $(s, c_{\text{hid}}, \cdot, \cdot) \notin \overline{\Omega}^{\text{pp}}_{\mathcal{U}}$, let $\mathcal{F}_{\text{P4TC}}$ abort.
  (b) Look up $\text{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for PID $\mathit{pid}_{\mathcal{U}}$; if no $\text{pk}_{\mathcal{U}}$ has been recorded abort.
  (c) If $\text{C1.Open}(\text{CRS}^1_{\text{com}}, \text{pk}_{\mathcal{U}}, c_{\text{hid}}, d_{\text{hid}}) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.
  (d) If $\mathtt{out}_{\mathcal{U}} = \mathsf{NOK}$ give up simulation.
  (e) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to SA.

Double-Spending Detection: Upon being ask to provide a proof for $\mathit{pid}_{\mathcal{U}}$, look up $\mathit{pid}_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$, and return $\text{sk}_{\mathcal{U}}$.

Guilt Verification: Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide out for $(\mathit{pid}_{\mathcal{U}}, \pi)$ …

(1) Receive $\text{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathit{pid}_{\mathcal{U}}$.
(2) If $g_1^{\pi} = \text{pk}_{\mathcal{U}}$, then return $\mathtt{out} := \mathsf{OK}$, else $\mathtt{out} := \mathsf{NOK}$ to $\mathcal{F}_{\text{P4TC}}$.

User Blacklisting: Upon receiving (λ, x) and being ask to provide a serial number, return $\phi := \text{PRF}(\lambda, x)$.

Fig. 48. The simulator for Operator Security (cont. from Fig. 44)

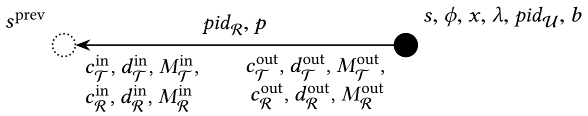

s^{prev}
$\mathit{pid}_{\mathcal{R}}, p$
$s, \phi, x, \lambda, \mathit{pid}_{\mathcal{U}}, b$
$c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}},$
$c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}$
$c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}},$
$c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}}$

Fig. 49. An entry $\overline{\mathit{trdb}} \in \overline{\mathit{TRDB}}$ visualized as an element of a directed 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* $\overline{\mathit{trdb}}$ has the form

$$\begin{aligned}\overline{\mathit{trdb}} = (&s^{\text{prev}}, s, \phi, x, \lambda, \mathit{pid}_{\mathcal{U}}, \mathit{pid}_{\mathcal{R}}, p, b,\\ &c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}},\\ &c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})\end{aligned} \tag{9}$$

with c, d and M with equal suffixes denoting a commitment, its decommitment information and the opening in the implicit message space (see Fig. 49). At the beginning of a transaction in the scope of Debt Accumulation or Debt Clearance the user loads his token τ^{prev} which contains two commitments $c_{\mathcal{T}}$ and $c^{\text{prev}}_{\mathcal{R}}$, randomizes the commitments and at the end the user possesses two updated commitments $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ 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 F.11 (Simulated Transaction Graph (informal)). The set $\overline{TRDB} = \{\overline{trdb_i}\}$ with $\overline{trdb_i}$ defined as in Eq. (9) 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 $c^{\text{in}}_{\mathcal{R}} = c^{\text{prev}}_{\mathcal{R}}$ (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

$$\mathsf{H}_i := \mathsf{EXEC}_{\pi_i, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_i, \mathcal{Z}^{\text{sys-sec}}}(1^n).$$

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 $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$.

We proceed by giving concrete (incremental) definitions of all hybrids H_i. Please note that *input privacy* for the 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 its prescribed output anyway. In other words, the simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ mimicking the role of an honest operator can perfectly simulate most messages towards the (malicious) user after is 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

$$\mathsf{H}_0 := \mathsf{EXEC}_{\pi_0, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_0, \mathcal{Z}^{\text{sys-sec}}}(1^n)$$

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 $\mathsf{CRS}_{\mathsf{pok}}$ is generated by SetupEPoK, and $\mathsf{CRS}^2_{\mathsf{com}}$ is generated by C2.SimGen. Additionally, $\mathcal{S}_1$ initializes the internal sets $\overline{TRDB}$, $\overline{\Omega}^{\text{dsp}}$, and $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ 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 $(\texttt{certifying\_rsu}, \mathit{pid}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}})$, creates the certificate $\sigma_{\mathcal{R}}^{\text{cert}}$ and records it.

Whenever the honest TSP or honest RSU running π_3 would send $\sigma_{\mathcal{R}}^{\text{cert}}$ as part of its messages in the scope of Wallet Issuing or Debt Accumulation, they omit $\sigma_{\mathcal{R}}^{\text{cert}}$. Instead, the simulator $\mathcal{S}_3$ injects $\sigma_{\mathcal{R}}^{\text{cert}}$ 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 $c_{\mathcal{R}}$ and $d''_{\mathcal{R}}$ in its final message, but reports the price p to $\mathcal{S}_4$, $\mathcal{S}_4$ creates $c_{\mathcal{R}}$ and $d''_{\mathcal{R}}$ honestly and injects them into the message.

Hybrid H_5. 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 c''_{ser} in the scope of Wallet Issueinng or Debt Accumulation. Instead $\mathcal{S}_5$ runs $(c''_{\text{ser}}, \overline{d}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{CRS}^2_{\text{com}})$ and injects c''_{ser} into the message. Moreover, π_5 for the TSP/RSU is modified such that it uniformly and independently picks $s \overset{\text{R}}{\leftarrow} \mathbb{Z}_{\text{p}}$ and passes s to $\mathcal{S}_5$ as part of the final message. $\mathcal{S}_5$ calculates $s'' := s \cdot (s')^{-1}$, executes $d''_{\text{ser}} \leftarrow \mathsf{C2.Equiv}(\mathsf{CRS}^2_{\text{com}}, \mathsf{td}_{\text{eqcom}}, s'', c''_{\text{ser}}, \overline{d}_{\text{ser}})$ and injects s'' together with d''_{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 $\lambda := \lambda'' + \sum_{i=0}^{\ell-1} \mathsf{DLOG}(\Lambda'_i) \cdot \mathcal{B}^i$, assembles $\overline{\mathit{trdb}}$ and appends it to $\overline{\mathit{TRDB}}$. Additionally, $\mathcal{S}_6$ also assembles $\overline{\omega}^{\text{pp}}_{\mathcal{U}}$, $\overline{\omega}^{\text{dsp}}$ in the scope of Debt Accumulation and appends entries to $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$, $\overline{\Omega}^{\text{dsp}}$ resp. If $\overline{\Omega}^{\text{dsp}}$ already contains an entry $\overline{\omega}^{\text{dsp}\ddagger}$ with matching fraud detection ID, the secret key $\mathsf{sk}_{\mathcal{U}}$ is immediately reconstructed and the pair $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ 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 $L^{(1)}_{\text{gp}}$, $L^{(2)}_{\text{gp}}$, and $L^{(3)}_{\text{gp}}$ resp. If not, $\mathcal{S}_6$ abort with failure event (*E1*).

Hybrid H_7. This hybrid modifies the code π_7 for $\mathcal{T}$, *SA* and *DR* in the scope of the tasks Prove Participation, Double-Spending Detection, Guilt Verification and User Blacklisting. The honest parties become dummy parties, the code is moved to the simulator and $\mathcal{S}_7$ resorts to its "lookup tables" $\overline{\mathit{TRDB}}$, $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$, and $\overline{\Omega}^{\text{dsp}}$ that have been introduced by the previous hybrid.

More precisely, in Prove Participation the party *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 *SA*.

In the task Double-Spending Detection the honest $\mathcal{T}$ 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 $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ has been recorded and returns $\mathsf{sk}_{\mathcal{U}}$.

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 User Blacklisting is modified accordingly. The dispute resolver *DR* becomes a dummy party and simply sends it input $(\texttt{blacklist\_user}, \mathsf{pk}^{\mathit{DR}}_{\mathcal{U}})$ 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 $\overline{\mathit{TRDB}}$ 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 $c_{\mathcal{R}}^{\text{out}*} \neq c_{\mathcal{R}}^{\text{prev}}$ and gives up the simulation with event *E3*, if the check suceeds.

Hybrid H_{11}. The simulator $\mathcal{S}_{11}$ additionally checks for $\Lambda \neq g_1^{\lambda^*}$ and gives up the simulation with event *E4*, if the check suceeds.

Hybrid H_{12}. The simulator $\mathcal{S}_{12}$ additionally checks for $c_{\mathcal{T}}^{\text{out}*} \neq c_{\mathcal{T}}$ and gives up the simulation with event *E5*, if the check suceeds.

Hybrid H_{13}. The simulator $\mathcal{S}_{13}$ parses $(\Lambda^*, \mathsf{pk}_{\mathcal{U}}^*) := M_{\mathcal{T}}^{\text{out}*}$ and checks for $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}_{\mathcal{U}}^*$. If the check suceeds, it gives up the simulation with event *E6*.

Hybrid H_{14}. The simulator $\mathcal{S}_{14}$ additionally checks for $B^{\text{prev}} \neq g_1^{b^*}$ and gives up the simulation with event *E7*, if the check suceeds.

Hybrid H_{15}. The simulator $\mathcal{S}_{15}$ additionally checks for $X \neq g_1^{x^*+1}$ and gives up the simulation with event *E8*, if the check suceeds.

For the proof of Theorem F.10 we show the indistinguishability of subsequent hybrids by a series of lemmas. The Lemmas F.12 to F.14 are rather trivial and thus Lemma F.13 handles various hybrids at once.

Lemma F.12 (Indistinguishability between H_0 and H_1). *Under the assumptions of Theorem F.10,* $\mathsf{H}_0 \overset{c}{\equiv} \mathsf{H}_1$ *holds.*

Proof. This hop solely changes how the CRS is created during the setup phase. This is indistinguishable for $\mathsf{CRS}_{\mathsf{pok}}$, and $\mathsf{CRS}_{\mathsf{com}}^2$ (see the extractability property of Definition C.5 and the equivocality property of Definition C.7, resp., condition (a) each). □

Lemma F.13 (Indistinguishability between its predecessors and H_2, H_3, H_4, H_7, or H_8, resp.). *Under the assumptions of Theorem F.10,* $\mathsf{H}_1 \overset{c}{\equiv} \mathsf{H}_2$, $\mathsf{H}_2 \overset{c}{\equiv} \mathsf{H}_3$, $\mathsf{H}_3 \overset{c}{\equiv} \mathsf{H}_4$, $\mathsf{H}_6 \overset{c}{\equiv} \mathsf{H}_7$, *and* $\mathsf{H}_7 \overset{c}{\equiv} \mathsf{H}_8$ *holds.*

Proof. The hops are all indistinguishable as the 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 F.14 (Indistinguishability between H_4 and H_5). *Under the assumptions of Theorem F.10,* $\mathsf{H}_4 \overset{c}{\equiv} \mathsf{H}_5$ *holds.*

Proof. This hop is indistinguishable as the equivoced decommitment information is perfectly indistinguishable from a decommitment that has originally been created with the correct message (cp. Definition C.7, 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 F.15 (Indistinguishability between H_5 and H_6). *Under the assumptions of Theorem F.10,* $\mathsf{H}_5 \stackrel{c}{\equiv} \mathsf{H}_6$ *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 $L^{(1)}_{\mathrm{gp}}$, $L^{(2)}_{\mathrm{gp}}$ and $L^{(3)}_{\mathrm{gp}}$, resp. In the following only the sub-hybrid for the language $L^{(1)}_{\mathrm{gp}}$ 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 trigger 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 C.5, Item (3b). If the event *E1* occurs internally, $\mathcal{B}$ outputs the corresponding pair (stmnt, π). In the second and third sub-hybrid $\mathcal{B}$ internally extracts the witness for the previous sub-hybrid using the extraction trapdoor $\mathsf{td}_{\mathrm{epok}}$ which $\mathcal{B}$ obtains as part of its input. □

Remark F.16. We observe that Lemma F.15 implies that the equations

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, m, c_{\mathcal{T}}, d_{\mathcal{T}}) = 1 \qquad \text{with} \qquad m = (\Lambda, \mathsf{pk}_{\mathcal{U}}) \tag{10}$$

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, m, c_{\mathcal{R}}, d_{\mathcal{R}}) = 1 \qquad \text{with} \qquad m = (\Lambda, \mathbb{1}, U_1^{\mathrm{next}}, g_1) \tag{11}$$

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, m, c_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{prev}}) = 1 \qquad \text{with} \qquad m = (\Lambda, B^{\mathrm{prev}}, U_1, X) \tag{12}$$

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, m, c'_{\mathcal{R}}, d'_{\mathcal{R}}) = 1 \qquad \text{with} \qquad m = (\Lambda, B^{\mathrm{prev}}, U_1^{\mathrm{next}}, X) \tag{13}$$

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, \Lambda', c'_{\mathrm{seed}}, d'_{\mathrm{seed}}) = 1 \tag{14}$$

$$\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, \mathsf{pk}_{\mathcal{U}}, c_{\mathrm{hid}}, d_{\mathrm{hid}}) = 1 \tag{15}$$

and

$$\mathsf{S.Vfy}(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, m) = 1 \qquad \text{with} \qquad m = (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}) \tag{16}$$

$$\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, m) = 1 \qquad \text{with} \qquad m = (c^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}}) \tag{17}$$

$$\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}\,\mathrm{prev}}_{\mathcal{R}}, m) = 1 \qquad \text{with} \qquad m = (\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}) \tag{18}$$

resp., hold 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 $\mathbb{Z}_p$ 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 $\mathsf{H}_0 \stackrel{c}{\equiv} \mathsf{H}_8$ 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 $\mathcal{Z}^{\text{sys-sec}}$ 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 $\overline{\mathit{TRDB}}$, but $\overline{\mathit{TRDB}}$ 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 F.3 and F.4. Intuitively, the commitments $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ induce a graph structure onto $\overline{TRDB}$ comparable to the wallet ID λ and serial number s.

Lemma F.17 (Forest Structure of the Simulated Transaction Graph). *(1) Every $\overline{trdb} = (s^{prev}, s, \ldots) \in \overline{TRDB}$ 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. (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 F.18 (Indistinguishability between H_8 and H_9). *Under the assumptions of Theorem F.10, $\mathsf{H}_8 \overset{c}{\equiv} \mathsf{H}_9$ holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that trigger 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 F.17, 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 $(c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ is a forgery. Or alternatively, the allegedly previous RSU does not exits but has been imagined by the user. Then $(c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ may have a honest, valid signature (because the user feigned the RSU), but the certificate $\text{cert}_{\mathcal{R}}{}^{\text{prev}}$ 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 $pid_{\mathcal{R}}^{prev} \mapsto (\mathsf{pk}_{\mathcal{R}}^{prev}, \mathsf{sk}_{\mathcal{R}}^{prev})$ has been recorded:* In other words, $(c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ has never been legitimately issued by a the the allegedly previous RSU.[31] 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 $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. $\mathcal{B}$ 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 $\mathsf{pk}_{\mathcal{R}}^{\text{prev}}$ it embeds the challenge as $\mathsf{pk}_{\mathcal{R}}^{\text{prev}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}_9$ needs to issue a signature with respect to $\mathsf{pk}_{\mathcal{R}}^{\text{prev}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event $(E2)$ occurs, $\mathcal{B}$ extracts $(c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ and $\sigma_{\mathcal{R}}^{\text{prev}}$ from the proof and outputs the forgery. N.b., $(c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})$ has never been signed with respect to $\mathsf{pk}_{\mathcal{R}}^{\text{prev}} = \mathsf{pk}_C$ by assumption.

(2) *A record $pid_{\mathcal{R}}^{prev} \mapsto (\mathsf{pk}_{\mathcal{R}}^{prev}, \mathsf{sk}_{\mathcal{R}}^{prev}, \text{cert}_{\mathcal{R}})$ 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 $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the adversary $\mathcal{B}$ has to internally provide $\mathsf{pk}_{\mathcal{T}} = (\mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{T}})$ playing the role of $\mathcal{S}_9$ in the scope of the TSP Registration, $\mathcal{B}$ embeds the external

[31] N.b.: 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 an RSU. For brevity, we only consider RSUs here.

challenge as $\mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}} := \mathsf{pk}_{C}$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}_9$ in the scope of RSU Certification needs to issue signatures with respect to $\mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event (*E2*) occurs, $\mathcal{B}$ extracts $\mathsf{cert}_{\mathcal{R}}{}^{\mathsf{prev}} = (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathsf{cert}}_{\mathcal{R}})$ from the proof and outputs $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$ together with $\sigma^{\mathsf{cert}}_{\mathcal{R}}$ as the forgery. N.b.: $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})$ has never been signed by the TSP with respect to $\mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}} = \mathsf{pk}_{C}$ as otherwise a mapping $pid^{\mathsf{prev}}_{\mathcal{R}} \mapsto (\mathsf{pk}^{\mathsf{prev}}_{\mathcal{R}}, \mathsf{sk}^{\mathsf{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$ would have been recorded.

The forgeries are indeed valid due to Remark F.16. □

Remark F.19. Without Lemma F.18 it is unclear in Lemma F.17, 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 F.17, 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 F.17, Item (2) remains correct. Anyway, this oddity is ruled out by Lemma F.18.

Lemma F.20 (Indistinguishability between H_9 and H_{10}). *Under the assumptions of Theorem F.10,* $\mathsf{H}_9 \overset{c}{\equiv} \mathsf{H}_{10}$ *holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that trigger 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 F.18 as $(c^{\mathsf{prev}}_{\mathcal{R}}, s^{\mathsf{prev}})$ are jointly signed by the same signature $\sigma_{\mathcal{R}}$. □

Lemma F.21 (Indistinguishability between H_{10} and H_{11}). *Under the assumptions of Theorem F.10,* $\mathsf{H}_{10} \overset{c}{\equiv} \mathsf{H}_{11}$ *holds.*

Proof. Assume there is an environment $\mathcal{Z}^{\text{sys-sec}}$ that trigger 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 C.7, Item (2). When the event (*E3*) occurs, $\mathcal{B}$ sets

$$M^{\mathsf{prev}}_{\mathcal{R}} := (\Lambda, B^{\mathsf{prev}}, U_1, X)$$

from the extracted witness and obtains

$$M^{\mathsf{out}*}_{\mathcal{R}} = (\Lambda^*, B^*, U^*_1, X^*)$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(c^{\mathsf{out}*}_{\mathcal{R}}, M^{\mathsf{prev}}_{\mathcal{R}}, d^{\mathsf{prev}}_{\mathcal{R}}, M^{\mathsf{out}*}_{\mathcal{R}}, d^{\mathsf{out}*}_{\mathcal{R}})$ to the external game. By assumption $\Lambda \neq \Lambda^*$ holds and Remark F.16 asserts that both openings are valid. □

Lemma F.22 (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 F.4. □

Lemma F.23 (Indistinguishability between H_{11} and H_{12}). *Under the assumptions of Theorem F.10,* $\mathsf{H}_{11} \overset{c}{\equiv} \mathsf{H}_{12}$ *holds.*

Proof. We introduce a sub-hybrid that splits between two cases why event *E5* is triggered: (1) $c^{\mathsf{out}*}_{\mathcal{T}} \neq c_{\mathcal{T}}$ and $c_{\mathcal{T}}$ is not recorded in any $\overline{trdb} \in \overline{TRDB}$. (2) $c^{\mathsf{out}*}_{\mathcal{T}} \neq c_{\mathcal{T}}$ and $c_{\mathcal{T}}$ 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 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 $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When $\mathcal{B}$ must internally provide $\mathsf{pk}_{\mathcal{T}} = (\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{T}}_{\mathcal{T}})$ playing the role of $\mathcal{S}_{12}$ in the scope of the TSP Registration, $\mathcal{B}$ embeds the external challenge as $\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}} := \mathsf{pk}_C$. 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 $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event $(E5)$ occurs, $\mathcal{B}$ extracts $c_{\mathcal{T}}$ and $\sigma_{\mathcal{T}}$ from the proof and outputs the forgery.

(2) We construct an efficient adversary $\mathcal{B}$ against the binding property of 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 C.7, Item (2). As $(E5)$ has not been raised earlier, $c^{\text{out}(i)}_{\mathcal{T}} = c^{\text{out}*}_{\mathcal{T}} \neq c_{\mathcal{T}}$ holds for all $c^{\text{out}(i)}_{\mathcal{T}}$ in the same tree. Consequently, $\overline{trdb}^{\ddagger}$ with $c^{\text{out}\ddagger}_{\mathcal{T}} = c_{\mathcal{T}}$ is part of a different tree in $\overline{TRDB}$ and thus $\Lambda^{\ddagger} \neq \Lambda^{*} = \Lambda$ follows by Lemma F.22. $\mathcal{B}$ sets

$$M_{\mathcal{T}} := (\Lambda, \mathsf{pk}_{\mathcal{U}})$$

from the extracted witness and obtains

$$M^{\text{out}\ddagger}_{\mathcal{T}} = (\Lambda^{\ddagger}, \mathsf{pk}^{\ddagger}_{\mathcal{U}})$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(c_{\mathcal{T}}, M_{\mathcal{T}}, d_{\mathcal{T}}, M^{\text{out}\ddagger}_{\mathcal{T}}, d^{\text{out}\ddagger}_{\mathcal{T}})$ to the external game.

Remark F.16 asserts that the forgery in (1) and both openings in (2) are indeed valid. □

Lemma F.24 (Indistinguishability between H_{12}, H_{13}, H_{14} and H_{15}). *Under the assumptions of Theorem F.10, $\mathsf{H}_{12} \overset{c}{\equiv} \mathsf{H}_{13} \overset{c}{\equiv} \mathsf{H}_{14} \overset{c}{\equiv} \mathsf{H}_{15}$ 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 against the binding property of 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 C.7, Item (2). If event $(E7)$ or $(E8)$ occurs, $\mathcal{B}$ sets

$$M^{\text{prev}}_{\mathcal{R}} = (\Lambda, B^{\text{prev}}, U_1, g_1 X)$$

from the extracted witness and obtains

$$M^{\text{out}*}_{\mathcal{R}} := (\Lambda^{*}, B^{*}, U^{*}_1, X^{*})$$

from $\overline{TRDB}$. $\mathcal{B}$ outputs $(c_{\mathcal{R}}, M^{\text{prev}}_{\mathcal{R}}, d^{\text{prev}}_{\mathcal{R}}, M^{\text{out}*}_{\mathcal{R}}, d^{\text{out}*}_{\mathcal{R}})$ to the external game. If the event $(E6)$ is triggered, $\mathcal{B}$ proceeds analogous but for the fixed part of wallet $c_{\mathcal{T}}$. □

Taking all the aforementioned statements together, Theorem F.10 from the beginning of this section follows. For the sake of formal completeness we recall it again.

Theorem F.10 (Operator Security). *Under the assumptions of Theorem F.1*

$$\pi^{\mathcal{F}_{CRS}, \bar{\mathcal{G}}_{bb}}_{P4TC} \geq_{UC} \mathcal{F}^{\bar{\mathcal{G}}_{bb}}$$

holds under static corruption of

(1) a subset of users, or
(2) all users and a subset of RSUs, TSP and SA.

Proof. A direct consequence of Lemmas F.12 to F.15, F.18, F.20, F.21, F.23 and F.24. □

F.3 Proof of User Security and Privacy

In this section we show the following theorem.

Theorem F.25 (User Security and Privacy). *Under the assumptions of Theorem F.1*

$$\pi_{P4TC}^{\mathcal{F}_{CRS},\overline{\mathcal{G}}_{bb}} \geq_{UC} \mathcal{F}^{\overline{\mathcal{G}}_{bb}}$$

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.

The definition of the UC-simulator $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$ for Theorem F.25 can be found in Figs. 50 to 54. Please note that while the real protocol π_{P4TC} lives in the $(\mathcal{F}_{\text{CRS}}, \overline{\mathcal{G}}_{\text{bb}})$-model, the ideal functionality $\mathcal{F}_{\text{P4TC}}$ has no CRS. Hence, the CRS (but not the bulletin board) is likewise simulated, providing $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$ with a lever to simulate the ZK proofs P1, P2, and P3, to equivoke C1, and to extract C2.

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

$$\mathsf{H}_i := \mathsf{EXEC}_{\pi_i, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_i, \mathcal{Z}^{\text{user-sec}}}(1^n).$$

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 $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$. We proceed by giving concrete (incremental) definitions of all hybrids H_i.

Hybrid H_0. The hybrid H_0 is defined as

$$\mathsf{H}_0 := \mathsf{EXEC}_{\pi_0, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_0, \mathcal{Z}^{\text{user-sec}}}(1^n)$$

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 SetupSPoK, $\text{CRS}_{\text{com}}^1$ is generated by C1.SimGen and $\text{CRS}_{\text{com}}^2$ is generated by C2.ExtGen. Additionally, $\mathcal{S}_1$ initializes the internal sets $\overline{\Omega}^{\text{dsp}}$, $\overline{\Omega}_{\mathcal{U}}^{\text{pp}}$ and $\overline{HTD}$ as empty sets and records the respective entries as the final simulator $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$ 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 $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$ 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 $s'' \leftarrow \text{C2.Extract}(\text{CRS}_{\text{com}}^2, c''_{\text{ser}})$, calculates $s' := s \cdot (s'')^{-1}$ and inserts s' into the message from the user to the TSP or RSU respectively.

Simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$

Setup:

(1) Run a modified version of the algorithm $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^n)$ with
  (a) $\mathsf{CRS}^1_{\mathsf{com}} \leftarrow \mathsf{C1.Gen}$ being replaced by $(\mathsf{CRS}^1_{\mathsf{com}}, \mathsf{td}_{\mathsf{eqcom}}) \leftarrow \mathsf{C1.SimGen}$,
  (b) $\mathsf{CRS}^2_{\mathsf{com}} \leftarrow \mathsf{C2.Gen}$ being replaced by $(\mathsf{CRS}^2_{\mathsf{com}}, \mathsf{td}_{\mathsf{extcom}}) \leftarrow \mathsf{C2.ExtGen}$, and
  (c) $\mathsf{CRS}_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}$ being replaced by $(\mathsf{CRS}_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}) \leftarrow \mathsf{SetupSPoK}$.
(2) Record CRS, $\mathsf{td}_{\mathsf{eqcom}}$, $\mathsf{td}_{\mathsf{extcom}}$, and $\mathsf{td}_{\mathsf{spok}}$.
(3) Set $\overline{\Omega}^{\mathrm{dsp}} := \emptyset$.
(4) Set $\overline{\Omega}^{\mathrm{pp}}_{\mathcal{U}} := \emptyset$.
(5) Set $\overline{HTD} := \emptyset$.

DR Registration: Upon receiving $(\texttt{registering\_dr}, pid_{\mathcal{DR}})$ run $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}}) \leftarrow \mathsf{DRRegistration}(\mathsf{CRS})$, return $\mathsf{pk}_{\mathcal{DR}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $pid_{\mathcal{DR}} \mapsto (\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})$.

TSP Registration: Distinguish two cases:

TSP is corrupted: (nothing to do as this is a local algorithm for a corrupted TSP)

TSP honest: Upon receiving $(\texttt{registering\_tsp}, pid_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}})$ run $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{TSPRegistration}(\mathsf{CRS}, \mathbf{a}_{\mathcal{T}})$, return $\mathsf{pk}_{\mathcal{T}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$.

RSU Registration: Distinguish two cases:

RSU is corrupted: (nothing to do as this is a local algorithm for a corrupted RSU)

RSU honest: Upon receiving $(\texttt{registering\_rsu}, pid_{\mathcal{R}})$ run $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathsf{RSURegistration}(\mathsf{CRS})$, return $\mathsf{pk}_{\mathcal{R}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$.

User Registration: Upon receiving $(\texttt{registering\_user}, pid_{\mathcal{U}})$ run $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \leftarrow \mathsf{UserRegistration}(\mathsf{CRS})$, return $\mathsf{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$.

RSU Certification: Distinguish four cases:

TSP and RSU honest: Upon receiving $(\texttt{certifying\_rsu}, pid_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}) \ldots$

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
(2) Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}})$ with $\sigma^{\mathrm{cert}}_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$ faithfully.
(3) Update record $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$.

TSP honest, RSU corrupted: Upon receiving $(\texttt{certifying\_rsu}, pid_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}) \ldots$

(1) Load the recorded $pid_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and obtain $\mathsf{pk}_{\mathcal{R}}$ from $\overline{\mathcal{G}}_{\mathrm{bb}}$ for $pid_{\mathcal{R}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
(2) Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}})$ with $\sigma^{\mathrm{cert}}_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$ faithfully.
(3) Record $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \bot, \mathsf{cert}_{\mathcal{R}})$.
(4) Output cert to $\mathcal{Z}^{\text{user-sec}}$.

TSP corrupted, RSU honest: Upon receiving $(\mathsf{cert}_{\mathcal{R}})$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}} \ldots$

(1) Load the recorded $pid_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\overline{\mathcal{G}}_{\mathrm{bb}}$ for $pid_{\mathcal{T}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
(2) Parse $\mathbf{a}_{\mathcal{R}}$ and $\sigma^{\mathrm{cert}}_{\mathcal{R}}$ from $\mathsf{cert}_{\mathcal{R}}$.
(3) If $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.
(4) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{certify}, \mathbf{a}_{\mathcal{R}})$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$.

TSP and RSU corrupted: (nothing to do as $\mathcal{Z}^{\text{user-sec}}$ plays both parties)

Fig. 50. The simulator for User Security and Privacy

Simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Wallet Issuing: Distinguish two cases:

TSP is honest: (nothing to do)

TSP is corrupted:

(1) Load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{T}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) $(c'_{\text{seed}}, d'^{\text{sim}}_{\text{seed}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{CRS}^1_{\text{com}})$

(3) Send $(\mathsf{pk}_{\mathcal{U}}, c'_{\text{seed}})$ to $\mathcal{Z}^{\text{user-sec}}$ as the 1st message from $\mathcal{U}$ to $\mathcal{T}$

(4) Upon receiving $(\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}, c''_{\text{ser}}, \lambda'')$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$ …[a]

 (a) Parse $(\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{T}}) := \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$.

 (b) If $\mathsf{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{T}}, (\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}})) = 0$ abort.

 (c) $\Lambda'' := g_1^{\lambda''}$

 (d) $s'' \leftarrow \mathsf{C2.Extract}(\mathsf{CRS}^2_{\text{com}}, \mathsf{td}_{\text{extcom}}, c''_{\text{ser}})$.

 (e) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{issue}, \mathbf{a}_{\mathcal{U}}, \emptyset)$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$.[b]

(5) Upon receiving output (s) from $\mathcal{F}_{\text{P4TC}}$ for $\mathcal{T}$ …

 (a) $s' := s \cdot s''^{-1}$.

 (b) $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$.

 (c) $e^* \leftarrow \mathsf{E1.Enc}(\mathsf{pk}_{DR}, (\underbrace{\mathbb{1}, \dots, \mathbb{1}}_{\ell+2}); r_1, r_2)$

 (d) $(c_{\mathcal{T}}, d_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, 0))$.

 (e) $(c_{\mathcal{R}}, d_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, 0, 0, 0))$.

 (f) $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{DR}, e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, c'_{\text{seed}}, \Lambda'', \lambda'')$.

 (g) $\pi \leftarrow \mathsf{P1.SimProof}(\mathsf{CRS}_{\text{pok}}, \mathsf{td}_{\text{spok}}, stmnt)$.

 (h) Send $(s', e^*, c_{\mathcal{T}}, c_{\mathcal{R}}, \pi)$ to $\mathcal{Z}^{\text{user-sec}}$ as the 2nd message from $\mathcal{U}$ to $\mathcal{T}$

(6) Upon receiving $(s'', d''_{\text{ser}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}})$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$ …

 (a) If $\mathsf{C2.Open}(\mathsf{CRS}^2_{\text{com}}, s'', c''_{\text{ser}}, d''_{\text{ser}}) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.

 (b) Set $\overline{htd} := (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ and insert $\overline{htd}$ into $\overline{HTD}$.

 (c) Create real token τ faithfully.

 (d) If $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.

 (e) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the User.

[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.

Fig. 51. The simulator for User Security and Privacy (cont. from Fig. 50)

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 c'_{seed}, $c_{\mathcal{T}}$ and $c_{\mathcal{R}}$ in the Wallet Issuing task and $c'_{\mathcal{R}}$ in the Debt Accumulation task. Instead, $\mathcal{S}_6$ injects suitable commitments to vectors of zeros. This equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.

Simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Debt Accumulation: Distinguish two cases:

RSU is honest: (nothing to do)

RSU is corrupted:

(1) Obtain $\mathsf{pk}_{\mathcal{T}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{T}}$; if it does not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Upon receiving $u_2, c''_{\text{ser}}, \mathsf{cert}_{\mathcal{R}}$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{R}$ with $pid_{\mathcal{R}}$, do …

(a) Parse $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\text{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$.

(b) If $\mathsf{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$ abort.

(c) $s'' \leftarrow \mathsf{C2.Extract}(\mathsf{CRS}^2_{\text{com}}, c''_{\text{ser}})$.

(d) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{pay\_toll}, \emptyset)$[a] in the name of $\mathcal{R}$ with $pid_{\mathcal{R}}$.

(e) Obtain RSU's output $(s, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}})$ from $\mathcal{F}_{\text{P4TC}}$, 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 $(c_{\text{hid}}, \overline{d}_{\text{hid}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{CRS}^1_{\text{com}})$ and append $(s, c_{\text{hid}}, \overline{d}_{\text{hid}})$ to $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$.

(b) $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, 0, 0, 0))$.

(c) Check if any $(\phi, t', u'_2) \in \overline{\Omega}^{\text{dsp}}$ has been recorded previously with (ϕ) being used as key. If no, pick $t \xleftarrow{\text{R}} \mathbb{Z}_p$. If yes, load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ and set $t := t' + \mathsf{sk}_{\mathcal{U}}(u_2 - u'_2)$. Insert (ϕ, t, u_2) into $\overline{\Omega}^{\text{dsp}}$.

(d) $stmnt := (\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t, u_2)$.

(e) $\pi \leftarrow \mathsf{P2.SimProof}(\mathsf{CRS}_{\text{pok}}, \mathsf{td}_{\text{spok}}, stmnt)$.

(f) Output $(s', \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ to $\mathcal{Z}^{\text{user-sec}}$.

(4) Upon being ask to provide a price for $(\mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}})$ return a price p as the dummy adversary would do.

(5) Upon receiving $(s'', d''_{\text{ser}}, c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ from $\mathcal{Z}^{\text{user-sec}}$ …[b]

(a) $d_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$.

(b) If $\mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (1, g_1^p, 1, g_1), c_{\mathcal{R}}, d_{\mathcal{R}}) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.

(c) If $\mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{R}}, \sigma_{\mathcal{R}}, (c_{\mathcal{R}}, s)) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.

(d) Let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the User.

[a] Use empty set as blacklist.

[b] 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`.

Fig. 52. The simulator for User Security and Privacy (cont. from Fig. 50)

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 $\overline{htd} = (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ in $\overline{HTD}$. Please note that $\mathcal{S}_7$ knows s due to the change in H_4.

Moreover, if the task User Blacklisting is invoked, $\mathcal{S}_7$ partitions the set of hidden user trapdoors $HTD_{\mathcal{U}}$ provided by the environment into two "subsets"[32] $HTD^{\text{genuine}}_{\mathcal{U}}$ and $HTD^{\text{fake}}_{\mathcal{U}}$ (cp. Fig. 54, Steps 2a to 2c). If for a hidden user trapdoor $htd = (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*)$ a corresponding entry $(\cdot, \cdot, \lambda'', e^*)$ is recorded in $\overline{HTD}$, then we call it a *genuine*

[32] 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 $(\mathsf{pk}_{\mathcal{U}}, s, \cdot, \cdot)$ are sanitized.

Simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

Debt Clearance: Distinguish two cases:

TSP is honest: (nothing to do)

TSP is corrupted:

(1) Load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{T}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.

(2) Upon receiving u_2 from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$, do …

(a) Call $\mathcal{F}_{\text{P4TC}}$ with input (`clear_debt`) in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$.

(b) Obtain leaked $\mathbf{a}^{\text{prev}}_{\mathcal{R}}$.

(c) Obtain TSP's output $(pid_{\mathcal{U}}, \phi, b^{\text{bill}})$ 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, pick $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$. If yes, load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ and set $t := t' + \mathsf{sk}_{\mathcal{U}}(u_2 - u'_2)$. Insert (ϕ, t, u_2) into $\overline{\Omega}^{\text{dsp}}$.

(b) $stmnt := (\mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{bill}}}, t, u_2)$.

(c) $\pi \leftarrow \mathsf{P3.SimProof}(\mathsf{CRS}_{\mathsf{pok}}, \mathsf{td}_{\mathsf{spok}}, stmnt)$.

(d) Output $(\mathsf{pk}_{\mathcal{U}}, \pi, \phi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{bill}}, t)$ to $\mathcal{Z}^{\text{user-sec}}$.

(4) Upon receiving (OK) from $\mathcal{Z}^{\text{user-sec}}$,[a] let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the User.

[a] If no message is received, let $\mathcal{F}_{\text{P4TC}}$ abort.

Fig. 53. The simulator for User Security and Privacy (cont. from Fig. 50)

hidden user trapdoor and the public key $\mathsf{pk}_{\mathcal{U}}$ and the serial number s are set to the originally recorded value. 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^*.[33] $\mathcal{S}_7$ additionally checks if the hidden user trapdoors of both sets $HTD^{\text{genuine}}_{\mathcal{U}}$ and $HTD^{\text{fake}}_{\mathcal{U}}$ belong to the same user public key $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}}$ or else aborts. This equals the behavior of the final simulator.

$\mathcal{S}_7$ runs the code of an honest *DR* on $HTD^{\text{genuine}}_{\mathcal{U}} \cup HTD^{\text{fake}}_{\mathcal{U}}$ 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 $\mathsf{pk}^{DR}_{\mathcal{U}}$ 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 $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ 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.[34] The final

[33] We assume, that Dec returns $\perp$ if e^* cannot be decrypted.

[34] 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.

Simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ (cont.)

```
Prove Participation:
    (1) Load the recorded $pid_{\mathcal{U}} \mapsto (\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$; if this does not exist let $\mathcal{F}_{\text{P4TC}}$ abort.
    (2) Upon receiving $S^{\text{pp}}_{\mathcal{R}}$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $SA$ ...
      (a) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{prove\_participation}, pid_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}})$.
      (b) Obtain the $SA$'s output $(\mathtt{out})$ from $\mathcal{F}_{\text{P4TC}}$.
      (c) If $\mathtt{out} = \mathtt{NOK}$, abort.
      (d) Pick $\overline{\omega}^{\text{pp}}_{\mathcal{U}} = (s, c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}})$ from $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ such that $s \in S^{\text{pp}}_{\mathcal{R}}$; if this does not exist abort.
      (e) Equivoke $d_{\mathsf{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{CRS}^{1}_{\mathsf{com}}, \mathsf{sk}_{\mathcal{U}}, c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}})$.
      (f) Output $(s, c_{\mathsf{hid}}, d_{\mathsf{hid}})$ to $\mathcal{Z}^{\text{user-sec}}$ as message from $\mathcal{U}$ to $SA$.
Double-Spending Detection: Upon being ask to provide a proof for $pid_{\mathcal{U}}$, look up $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, and return
    $\mathsf{sk}_{\mathcal{U}}$.
Guilt Verification: Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide $\mathtt{out}$ for $(pid_{\mathcal{U}}, \pi)$ ...
    (1) Receive $\mathsf{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{U}}$.
    (2) If $g_1^{\pi} = \mathsf{pk}_{\mathcal{U}}$, then return $\mathtt{out} := \mathtt{OK}$, else $\mathtt{out} := \mathtt{NOK}$ to $\mathcal{F}_{\text{P4TC}}$.
User Blacklisting: Distinguish two cases:
      TSP honest: (nothing to do)
      TSP corrupted:
      (1) Load the recorded $pid_{DR} \mapsto (\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$.
      (2) Upon receiving $HTD_{\mathcal{U}}$ from $\mathcal{Z}^{\text{user-sec}}$ ...
        (a) $HTD^{\text{genuine}}_{\mathcal{U}} = \left\{(\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \,\middle|\, (\cdot, \cdot, \lambda'', e^*) \in HTD_{\mathcal{U}} \wedge (\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \in \overline{HTD}\right\}$
        (b) $HTD^{\text{fake}}_{\mathcal{U}} = \left\{(\mathsf{pk}_{\mathcal{U}}, s, \lambda'', e^*) \,\middle|\, (\cdot, s, \lambda'', e^*) \in HTD_{\mathcal{U}} \wedge (\cdot, \cdot, \lambda'', e^*) \notin \overline{HTD} \wedge (\ldots, \mathsf{pk}_{\mathcal{U}}) \leftarrow \mathsf{E1.Dec}(\mathsf{sk}_{DR}, e^*)\right\}$
        (c) Assert $\mathsf{pk}^{(1)}_{\mathcal{U}} = \mathsf{pk}^{(2)}_{\mathcal{U}}$ for all $(\mathsf{pk}^{(1)}_{\mathcal{U}}, \cdot, \cdot, \cdot), (\mathsf{pk}^{(2)}_{\mathcal{U}}, \cdot, \cdot, \cdot) \in HTD^{\text{genuine}}_{\mathcal{U}} \cup HTD^{\text{fake}}_{\mathcal{U}}$ and set $\mathsf{pk}^{\mathcal{T}}_{\mathcal{U}} := \mathsf{pk}^{(1)}_{\mathcal{U}}$,
            else abort
        (d) Call $\mathcal{F}_{\text{P4TC}}$ with $(\texttt{blacklist\_user}, \mathsf{pk}^{\mathcal{T}}_{\mathcal{U}})$.
      (3) Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide $S_{\text{root}}$ ...
        (a) $S_{\text{root}} := \left\{s \,\middle|\, (\cdot, s, \cdot, \cdot) \in HTD^{\text{genuine}}_{\mathcal{U}}\right\}$
        (b) Provide $S_{\text{root}}$ to $\mathcal{F}_{\text{P4TC}}$
      (4) Upon receiving TSP's output $(b^{\text{bill}}, \Phi_{\text{bl}})$ from $\mathcal{F}_{\text{P4TC}}$ ...
        (a) For $HTD^{\text{fake}}_{\mathcal{U}}$ recover $\Phi^{\text{fake}}$ 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 $DR$ to $\mathcal{T}$
```

Fig. 54. The simulator for User Security and Privacy (cont. from Fig. 50)

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.[35]

[35] I.e., communication is confidential, reliable and trustworthy. One might think of this entity as a preliminary version of the eventual ideal functionality.

$\mathcal{F}_{\phi\text{-rand}}$ 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 $f_\Phi(\lambda, x) := \mathsf{PRF}(\lambda, x)$. 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 $\{f_\Phi(\lambda, 0), \dots, f_\Phi(\lambda, x_{\mathrm{bl}_\mathcal{R}})\}$.

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 User Blacklisting is invoked, the simulator $\mathcal{S}_8$ proceeds as follows: For hidden user trapdoors in the set $\mathit{HTD}_\mathcal{U}^{\text{fake}}$ it still decrypts the seed and evaluates the PRF. However, for hidden user trapdoors $(\mathsf{pk}_\mathcal{U}, s, \lambda'', e^*)$ in the set $\mathit{HTD}_\mathcal{U}^{\text{genuine}}$, $\mathcal{S}_8$ 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^*.

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 $f_\Phi(\lambda, x) := \phi$.

Hybrid H_{11}. In 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 $(\phi, t', u_2') \in \overline{\Omega}^{\text{dsp}}$ has been recorded previously, $\mathcal{S}_{11}$ picks $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$, else $\mathcal{S}_{11}$ sets $t := t' + \mathsf{sk}_\mathcal{U}(u_2 - u_2')$. Finally, $\mathcal{S}_{11}$ inserts (ϕ, t, u_2) into $\overline{\Omega}^{\text{dsp}}$. This equals the behavior of the final simulator $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$.

Hybrid H_{12}. The hybrid H_{12} modifies Debt Accumulation and Prove Participation. In Debt Accumulation the simulator $\mathcal{S}_{12}$ runs $(c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{CRS}_{\mathsf{com}}^1)$ and appends $(s, c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}})$ to $\overline{\Omega}_\mathcal{U}^{\text{pp}}$. 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 $(s^*, c_{\mathsf{hid}}^*, d_{\mathsf{hid}}^*)$ and respectively sends `OK` or `NOK` to $\mathcal{S}_{12}$. If $\mathcal{S}_{12}$ receives `OK` from the user, then it picks $\omega_\mathcal{U}^{\text{pp}} = (s, c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}})$ from $\overline{\Omega}_\mathcal{U}^{\text{pp}}$ such that $s \in S_\mathcal{R}^{\text{pp}}$. Furthermore, it runs $d_{\mathsf{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{CRS}_{\mathsf{com}}^1, \mathsf{sk}_\mathcal{U}, c_{\mathsf{hid}}, \overline{d}_{\mathsf{hid}})$ and sends $(s, c_{\mathsf{hid}}, d_{\mathsf{hid}})$ to $\mathcal{T}$. Again, this equals the behavior of the final simulator $\mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}$.

The combinations of all modifications from H_0 to H_{12} yields

$$\begin{aligned} \mathsf{H}_{12} &= \mathsf{EXEC}_{\pi_{12}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_{12}, \mathcal{Z}^{\text{user-sec}}}(1^n) \\ &= \mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_{\pi_{\text{P4TC}}}^{\text{user-sec}}, \mathcal{Z}^{\text{user-sec}}}(1^n). \end{aligned}$$

With these hybrids we can now give the proof of Theorem F.25. We do not spell out all steps of the proofs in full detail, but rather sketch the necessary reductions.

Proof of Theorem F.25.

From H_0 *to* H_1*:* This hop solely changes how the CRS is created during the setup phase. This is indistinguishable for $\mathsf{CRS}_{\mathsf{pok}}$, $\mathsf{CRS}_{\mathsf{com}}^1$, and $\mathsf{CRS}_{\mathsf{com}}^2$ (see the composable zero-knowledge property of Definition C.5, the equivocality property and the extractability property of Definition C.7, 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 $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.

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 $\mathsf{Adv}^{\text{pok-zk}}_{\mathsf{POK},\mathcal{B}}(n)$. 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.SimProof. $\mathcal{B}$ 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 c'_{seed}, $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ and $c'_{\mathcal{R}}$ 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 E1 the ciphertext e^* determines a unique message (for a fix key pair pk_{DR}, sk_{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 $\{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\text{bl}_{\mathcal{R}}})\}$, if required. Again, this is possible due to the correctness of E1, 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 $\mathbb{1}$-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 E1. 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-CCA 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 $\mathsf{pk}_{DR} := \mathsf{pk}^{\mathcal{C}}$. $\mathcal{B}$ 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 $\mathbb{1}$-vector for the remaining invocations of Wallet Issuing. If $\mathcal{Z}^{\text{user-sec}}$ invokes the task Blacklist User and $\mathcal{B}$ needs to restore the wallet ID, the following two cases may occur: a) The presented hidden user trapdoor is a genuine trapdoor. In this case $\mathcal{B}$ uses its lookup table to recover the correct set of fraud detection IDs. b) The presented hidden user trapdoor is a fake trapdoor. In this case $\mathcal{B}$ uses its decryption oracle of the external CCA-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 against the pseudo-random game as defined in Definition C.13. 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, 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 $\mathbb{1}$-vectors in the previous hybrid H_9. This enables the external challenger to pick any seed $\hat{\lambda}$.

From H_{10} *to* H_{11}*:* This hop is 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 c_{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. □