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 cryptographic model and 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), and do not come with a thorough security model and proof. In fact, to the best of our knowledge, our system is the first in the DSRC setting with a (rigorous) 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.

For our ETC system, we make use of and significantly extend a payment protocol building block, called Black-Box Accumulators, introduced at ACM CCS 2017. Additionally, we provide a prototypical implementation of our system 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 [50] 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 [23]. Hence, ETC has been and will be an important technology deserving a careful analysis of security and privacy issues as well as dedicated 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 [53], EZ-pass transponders are abused to track drivers throughout New York City [58], and the Norwegian AutoPASS system allowed anyone to obtain a transcript of which toll booths a car visited [25].

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) [31] 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 [22, 54].

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: Open road (aka free-flow) tolling is done without the use of toll booths, separated lanes, or barriers. Instead, 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 enforcing honesty by cameras.

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 [54]. 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, existing systems 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 [4, 21, 24]—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 privacy-preserving ETC 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 [26, 41–43] 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 scenario 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, the user may not overpay since he cannot get any change from the RSU (in a privacy-preserving way). So he would not only need enough money but the right denomination of e-coins. Moreover, *none* of the previous proposals come with a formal security model and proof.

In [41–43] multiple electronic road pricing systems specifically tailored for Low Emission Zones (LEZ) are proposed. In [41] a user drives by an RSU and directly pays some price depending on this RSU. For [42, 43] 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 [26], 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 [7], 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.

1.4.2 GNSS (GPS) Setting. A variety of GNSS-based toll collection systems can be found in the literature. Here, the OBU equipped with GPS and GSM typically collects location-time data or road segment prices and sends this data to the 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 [27]. 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 [55], 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 [24] ProVerif is used to show that VPriv is privacy-preserving for honest-but-curious adversaries.

In [38] 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 [4] 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 [51], 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 as an extension of PrETP. 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 [20], 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 [21], the system's security and privacy properties are verified using ProVerif.

Another cell-based road pricing scheme is presented in [32]. 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. Because of the cell structure, the accuracy requirements for the on-board GPS-devices are fairly relaxed in comparison to other systems. 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. If they were visible or predictable, users could easily collude and cheat. On the other hand, spot checks reveal a user's location and thus prevent users from cheating. Hence, fixing the number of spot checks is a trade-off between ensuring honesty and preserving privacy. Clearly, the penalty a user faces when cheating influences the number of spot checks required to ensure a certain security level.

In [45], 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. This protocol is based on oblivious transfer, a fair coin flip, and an identification protocol. 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 ensure that a user actually interacts with the device, a violation 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 and we could also surrender the GNSS technology.

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.

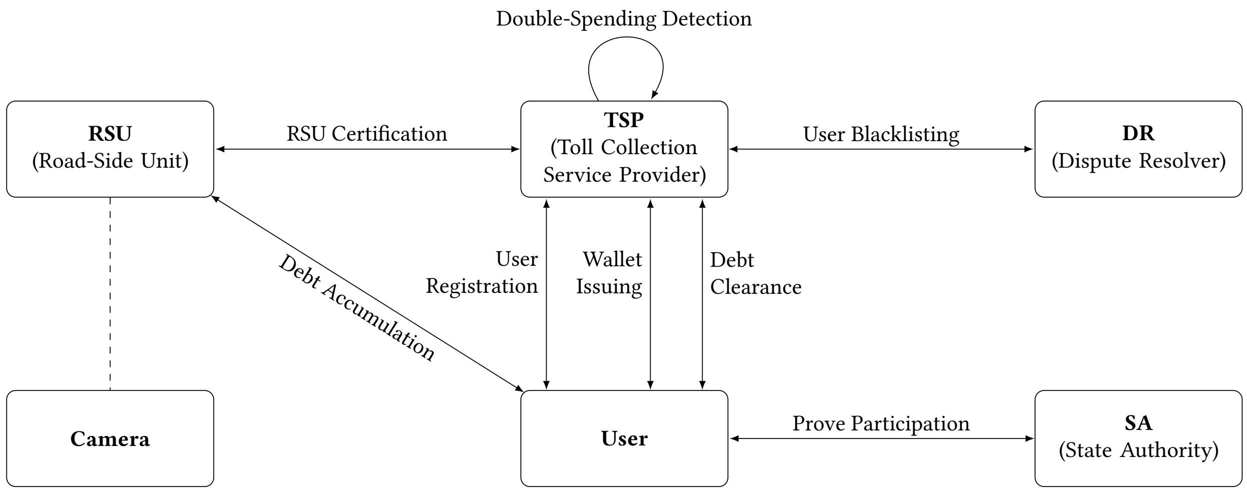

Fig. 1. The P4TC System Model

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.

- *Cameras* triggered by the RSUs which are typically owned by the SA and used to make photos of license plates (and possibly drivers) in case anything goes wrong. Alternatively or additionally, 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: In the scope of this protocol, a wallet is issued to a user by the TSP which is bound to the user's ID and a set of user attributes (the role of attributes is discussed later on). The wallet is used to accumulate debt.

Debt Accumulation: Every time the user passes an RSU with his car, the user (by means of his OBU) and the (offline) 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 [39]. In these cases, photos of more than one car being in the range of

the RSU are taken. These photos are transmitted to the SA along with transcripts of the protocols having be run at the time in question.

Prove Participation: After the SA has identified the users behind the license plates involved in an incident, the transactions need to be matched with the users in order to determine 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, respectively. This prevents honest users who successfully ran Debt Accumulation from being penalized.

Debt Clearance: At the end of a billing period, all users who obtained a wallet for this period are requested to participate in a clearance phase with the TSP. As a 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 provided by the system. It provides provable privacy up to what can be possibly be deduced by an operator (TSP and RSUs) who *explicitly* learns those values as part of its input needed to compute the pricing function. Additionally, at the end of a billing period the TSP learns the total balance owed by a user, but no individual transactions. Our framework guarantees that protocol executions of honest users do not leak anything (useful) beyond that.

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. For realistic pricing functions the chance to correctly identify an individual trip at the end of a billing period given only its total balance is negligible [56].

[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 for RSUs and/or users need to be avoided.

1.5.3 Desired Properties. The following list summarizes the desired security properties in an informal manner. Having them in mind, we propose an ideal world model formally covering what we would like to achieve in Section 2.

(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.
(P4) If a user re-uses 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 of our construction might appear quite intuitive and we partly draw from known techniques, a major challenge was to twist and combine all these techniques in order to achieve simulation-based security and practicality *at the same time*.

To this end, we started from a payment system building-block called black-box accumulation (BBA+), recently introduced in [36]. 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 subsection 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 [12]. 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

[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 it to the TSP, it could track each and every user in the system.

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, e.g., 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 RSU can be limited to enable fast transactions. IDs that belong to a previous billing period can be removed from the blacklist. Similarly, the transaction database needed to recalculate balances can also 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, we 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 [48] 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 [16].

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 [17, 40].

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-commitments are non-transferable, i.e., the commitment message cannot be passed to a different party, but we exploit this property heavily. Abstract 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 [57]. 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 [16], 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 [13, 16].

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 position of the corrupted party, reports its internal state to $\mathcal{Z}$ and from then on may arbitrarily deviate from the protocol 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 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 $id = (pid, sid)$.

[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.

- 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 subsection 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 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, $\overline{\mathcal{G}}_{\text{bb}}$ [18, Fig. 3], which is sometimes also referred to as key registration service in the literature [5, 15].

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 [16] 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 subsection 1.5.3. As already mentioned, 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 thus correctness and security can be defined more easily.

An excerpt of $\mathcal{F}_{\text{P4TC}}$ is depicted in Fig. 2. For better understanding we refrain from giving a complete descrip-

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

I. State

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

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

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

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

- 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_Π assigning a user PID $pid_\mathcal{U}$ and a proof of guilt π to a validity bit:

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

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

II. Behavior (Task Debt Accumulation only)

User input: ($\texttt{pay\_toll}$, s^{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) *User corrupted:* Ask the adversary if the PID $pid_\mathcal{U}$ of another corrupted user should be used.[a]
(3) Look up $(\cdot, s^{\text{prev}}, \varphi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_\mathcal{U}, pid_\mathcal{R}^{\text{prev}}, \cdot, b^{\text{prev}}) \in$ *TRDB*, with $(s^{\text{prev}}, pid_\mathcal{U})$ being the unique key.
(4) If $\varphi^{\text{prev}} \in bl_\mathcal{R}$, output $\texttt{blacklisted}$ 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 $\varphi := f_\Phi(\lambda, x)$ has already been defined; continue with (7).
Pick fraud detection ID $\varphi \overset{\text{R}}{\leftarrow} \Phi$ that has not previously been used.
User corrupted: Allow the adversary to choose a previously unused fraud detection ID φ.[b]
Append $(\lambda, x) \mapsto \varphi$ 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 $p := \text{O}_{\text{pricing}}(\mathbf{a}_\mathcal{U}, \mathbf{a}_\mathcal{R}, \mathbf{a}_\mathcal{R}^{\text{prev}})$.
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 $(s^{\text{prev}}, s, \varphi, x, \lambda, pid_\mathcal{U}, pid_\mathcal{R}, p, b)$ to *TRDB*.

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

[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 (cp. text body).

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

tion 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 C.

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 or misbehaving[6] parties and weaken the security and privacy guarantees in those cases. The key idea behind $\mathcal{F}_{\text{P4TC}}$ is to keep track of all conducted transactions in a pervasive transaction database *TRDB* (cp. Fig. 2). Each transaction entry $trdb \in TRDB$ is of the form

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

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 our system to have 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 $(\varphi, x, \lambda, pid_{\mathcal{U}}, b)$. 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 re-uses 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.

[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.
[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.

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 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 ($(\lambda, x) \mapsto \varphi$) 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 ($\mathbf{a}_{\mathcal{R}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}})$, $\mathbf{a}_{\mathcal{R}}^{\text{prev}} := f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}^{\text{prev}})$). Finally, the ideal functionality queries the pricing oracle $\mathrm{O}_{\text{pricing}}$ for the price p of this transaction, calculates the new balance of the wallet ($b := b^{\text{prev}} + p$) 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 $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ 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

[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.

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

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

The information leakage that needs to be considered for an eventual privacy assessment directly follows from the in- and output of the ideal functionality. We stress that we only care about privacy for honest and well-behaving users. Hence, the grayed out steps can be ignored. Moreover, we ignore "trivial attacks" like a conspicuously chosen price (see Fig. 2, Step 8 and cp. subsection 1.5.2) and the premature abort in case of a blacklisted user.[10] 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 information leakage of an honest and well-behaving user in this task is determined by his and the previous RSU's attributes $\mathbf{a}_{\mathcal{U}}$ and $\mathbf{a}_{\mathcal{R}}^{\text{prev}}$ which are both learned by the current RSU at the end of Debt Accumulation.

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 $(\lambda, x) \mapsto \varphi$ 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. Please note that these sketches omit some details regarding special cases if one of the involved parties is corrupted. The full-fledged ideal functionality is described in Appendix C.

Registration and Certification: There are some auxiliary tasks for registration and certification of parties. They are modeled in the obvious manner and append the appropriate mappings (e.g., $f_{\mathcal{A}_{\mathcal{R}}}$) with the given attributes for the PIDs at hand.

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

[10]These privacy losses are inherent to any such scheme and need to be explicitly modeled in the ideal functionality in order for the indistinguishability proof to hold.

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. Additionally, the proof is internally recorded as being issued as a valid proof for this particular 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.

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. subsection 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.

(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.

(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 $S^{\text{pp}}_{\mathcal{R}}$ 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 $(s, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}})$ 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 version the user wishes to use for the transaction is contained in this list.

(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): Is handled outside the scope of $\mathcal{F}_{\text{P4TC}}$ by encoding a limited time of validity into the RSU's attributes (cp. subsection 1.5.2).

4 PROTOCOL OVERVIEW

For an easier comprehension of the system, this section gives only a *simplified* description of the main protocols of our system. We believe that this description illustrates the main ideas behind P4TC. The description of the full protocols is postponed to 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 in our case). For readers not familiar with these building blocks and their security properties, we refer to Appendix A for a detailed description.

Our system instantiation is based on an asymmetric bilinear group setting $(G_1, G_2, G_T, e, p, g_1, g_2)$. Here, G_1, G_2, G_T are cyclic groups of prime order 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 $e\colon\ G_1 \times G_2 \to G_T$ is a (non-degenerated) bilinear map. We rely on the SXDH assumption [36] asserting that the well-known DDH assumption holds in both G_1 and G_2.

In this section we use a simpler notation than the one introduced in Appendix A 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 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.

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 $\text{sk}_{\mathcal{T}}^{\mathcal{T}}$ 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 $\text{sk}_{\mathcal{R}}$. The fixed part $c_{\mathcal{T}} = \text{Com}(\lambda, \text{pk}_{\mathcal{U}}^{\text{id}})$[11] is a commitment on the PRF key λ

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

(which is used as wallet ID) and the user's secret identification key $\mathsf{sk}_{\mathcal{U}}^{\mathrm{id}}$. The updatable part $c_{\mathcal{R}} = \mathsf{Com}(\lambda, b, u_1, x)$ also contains λ (to link both parts), the current balance b (debt), some user-chosen randomness u_1 to generate double-spending tags for the current version of the wallet, and a counter value x being the input to the PRF. The value $\mathsf{PRF}(\lambda, x)$ 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.

4.4.2 DR/TSP/RSU Key Generation. The DR generates a key pair $(\mathsf{pk}_{DR}, \mathsf{sk}_{DR})$ for an IND-CPA 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 $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ is generated for each RSU to sign the updatable part of a user's wallet. Moreover, the TSP generates several key pairs $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{T}})$, $(\mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}}, \mathsf{sk}_{\mathcal{T}}^{\mathrm{cert}})$, $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}, \mathsf{sk}_{\mathcal{T}}^{\mathcal{R}})$ for an EUF-CMA secure signature scheme. The key $\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$ is used to sign fixed user-specific information when a new wallet is issued. Using $\mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}$, 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 $\mathsf{cert}_{\mathcal{R}}$. It contains the RSU's public key $\mathsf{pk}_{\mathcal{R}}$, 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 $\mathsf{sk}_{\mathcal{T}}^{\mathrm{cert}}$.

4.4.4 User Registration. Before a user can participate in the system, he needs to generate a key-pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$. We assume that binding $\mathsf{pk}_{\mathcal{U}}$ 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.

The keys $\mathsf{pk}_{\mathcal{U}} = (\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}}, \mathsf{pk}_{\mathcal{U}}^{\mathrm{auth}})$ and $\mathsf{sk}_{\mathcal{U}} = (\mathsf{sk}_{\mathcal{U}}^{\mathrm{id}}, \mathsf{sk}_{\mathcal{U}}^{\mathrm{auth}})$ consist of two parts each: $(\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}}, \mathsf{sk}_{\mathcal{U}}^{\mathrm{id}})$ is used throughout the protocols to bind a wallet to a user, whereas $(\mathsf{pk}_{\mathcal{U}}^{\mathrm{auth}}, \mathsf{sk}_{\mathcal{U}}^{\mathrm{auth}})$ is used only when a new wallet is issued to authenticate the deposited PRF key. We use $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}} = g_1^{\mathsf{sk}_{\mathcal{U}}^{\mathrm{id}}} \in G_1$ and $\mathsf{sk}_{\mathcal{U}}^{\mathrm{id}} \in \mathbb{Z}_p$.

4.4.5 Wallet Issuing. In this protocol executed between a user and the TSP, a new wallet with balance 0 is created and a new PRF key is chosen and deposited using the DR's public encryption key. The PRF key needs to be chosen jointly and must only be known to the user. 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 generate the wallet, the user encodes his secrets into the wallet. That means, he randomly chooses his share λ' of the PRF key and some randomness u_1 to generate a double-spending tag (later on). Then he computes the commitments $c_{\mathcal{T}} = \mathsf{Com}(\lambda', \mathsf{sk}_{\mathcal{U}}^{\mathrm{id}})$ and $c_{\mathcal{R}} := \mathsf{Com}(\lambda', b := 0, u_1, x := 0)$. 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, which has key space $\mathbb{Z}_p$. To work with Groth-Sahai, we would need an algebraic encryption scheme with message space $\mathbb{Z}_p$.

Unfortunately, we are not aware of any such scheme.[12] We therefore use a workaround: The user splits up λ' into small pieces $\lambda'_i < \mathcal{B}$ (e.g., $\mathcal{B} = 2^{32}$) and encrypts $g_1^{\lambda'_i}$ under pk_{DR} resulting in ciphertexts e_i.

The user then sends over $\mathrm{pk}_{\mathcal{U}}$, $c_{\mathcal{T}}$, $c_{\mathcal{R}}$, e_i along with a NIZK π proving that everything has been generated honestly and the wallet is bound to the user owning $\mathrm{sk}_{\mathcal{U}}$. This NIZK, in particular, includes range proofs (cf. Appendix A.2.2) for $\lambda'_i < \mathcal{B}$. Besides this, the user also sends over an encryption e^* of $g_1^{\lambda'}$ and a signature σ^* on the ciphertexts e_i and e^* under $\mathrm{sk}_{\mathcal{U}}^{\mathrm{auth}}$. This information is used to bind a PRF key to a user, so the DR cannot be tricked into recovering the trapdoor for a different user.

When the TSP receives this data, it verifies the signature and the NIZK first. If these checks pass, the TSP chooses its share λ'' of the PRF key and adds it to the user's share in the commitments $c_{\mathcal{T}}$, $c_{\mathcal{R}}$ by using the homomorphic property of the commitment scheme. Finally, $c_{\mathcal{T}}$ is signed using $\mathrm{sk}_{\mathcal{T}}^{\mathcal{T}}$ along with attributes $\mathbf{a}_{\mathcal{U}}$ the TSP assigned to the user, and $c_{\mathcal{R}}$ is signed along with s using $\mathrm{sk}_{\mathcal{T}}^{\mathcal{R}}$.[13] The resulting signatures $\sigma_{\mathcal{T}}$, $\sigma_{\mathcal{R}}$, the commitments $c_{\mathcal{T}}$, $c_{\mathcal{R}}$, the share λ'', and the attributes $\mathbf{a}_{\mathcal{U}}$ are sent to the user, who checks their correctness. The user finally stores his freshly generated state token

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

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 $htd := (\lambda'', e_i, e^*, \sigma^*)$ as hidden trapdoor to recover λ with help of DR.

4.4.6 User Blacklisting. This is a protocol executed by DR upon request of the TSP, who provides inputs $\mathrm{pk}_{\mathcal{U}}^{\mathrm{auth}}$, htd. It recovers the PRF key of the user owning $\mathrm{pk}_{\mathcal{U}}^{\mathrm{auth}}$ or aborts.

First, DR decrypts each e_i using sk_{DR} to obtain $a_i := g_1^{\lambda'_i}$, where λ'_i are the small pieces of the user's share λ' of the PRF key. Then it computes the discrete logarithms of a_i to the base g_1 to recover the pieces itself. This is feasible as the DLOGs are very small. The key λ can be computed as $\lambda := \lambda' + \lambda''$, where $\lambda' := \sum_i \lambda'_i \cdot \mathcal{B}^i$. The DR also checks if e^* decrypts to $g_1^{\lambda'}$ and verifies the signature σ^*. If a check fails, the DR aborts as λ' has not been deposited by the user owning $\mathrm{pk}_{\mathcal{U}}^{\mathrm{auth}}$.

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. Here, we need to precompute enough IDs to cover all Debt Accumulation transactions the user may do until the end of the billing period.

4.4.7 Debt Accumulation. In this protocol executed between a user and an RSU, the toll p is determined and a new version of the user's wallet with a debt increased by p is created. See Fig. 3 for a depiction of the protocol. 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 $\mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}$ (the attributes of the RSU who signed the previous version 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

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

containing the state of his previous protocol run including the old version of his wallet. To prepare a new version of his wallet, the user computes a fresh commitment $c'_{\mathcal{R}}$ on $(\lambda, b^{\mathrm{prev}}, u_1^{\mathrm{next}}, x)$, where except for u_1^{next} the same values are contained in the previous version of his wallet. The randomness u_1^{next} is freshly chosen by the user

[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).

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

Fig. 3. A Simplified Version of the Debt Accumulation Protocol

and is used to generate a double-spending tag for the new wallet version. In order to get his new wallet version certified by the RSU, the user needs to invalidate his previous version. To do so, he needs to reveal the fraud detection ID and double-spending tag of his previous version. 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 $t := u_2 \cdot \mathsf{sk}_{\mathcal{U}}^{\text{id}} + u_1^{\text{next}} \bmod \mathfrak{p}$ (a linear equation in the unknowns u_1 and $\mathsf{sk}_{\mathcal{U}}^{\text{id}}$), the fraud detection ID $\varphi := \mathsf{PRF}(\lambda, x)$, and a hidden user ID $c_{\text{hid}} := \mathsf{Com}(\mathsf{pk}_{\mathcal{U}}^{\text{id}})$. 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

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

a certified wallet version 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 $\text{pk}_{\mathcal{R}}^{\text{prev}}$ 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 $\text{sk}_{\mathcal{R}}$. Then the 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 $\text{pk}_{\mathcal{R}}$, 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

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

containing his new wallet version $c_{\mathcal{T}}, c_{\mathcal{R}}, \sigma_{\mathcal{T}}, \sigma_{\mathcal{R}}$. He additionally stores $(c_{\text{hid}}, d_{\text{hid}})$, where d_{hid} allows to open the hidden ID. The RSU stores $(\varphi, p, t, u_2, c_{\text{hid}}, s)$ 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 $\text{pk}_{\mathcal{U}}^{\text{id}}$. It first 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 $\text{pk}_{\mathcal{U}}^{\text{id}}$ 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 $\text{pk}_{\mathcal{U}}^{\text{id}}$ and reveals the balance for his current wallet version. The wallet is invalidated by not creating a new version.

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 $\text{pk}_{\mathcal{U}}^{\text{id}}$, b^{prev}, φ, t and a NIZK π. This NIZK essentially shows that the user knows a certified wallet version with balance b^{prev}, fraud detection ID φ, and double-spending tag t which is bound to $\text{pk}_{\mathcal{U}}^{\text{id}}$. Note that if the user does not make use of his latest wallet, double-spending detection will reveal this. If the proof verifies, the balance and the double-spending information, i.e., $(b^{\text{prev}}, \varphi, t, u_2)$, is stored by the TSP.

4.4.10 Double-Spending Detection. This algorithm is applied by the TSP to its database containing double-spending tags. The fraud detection ID φ is used as the database index associated with the double-spending tag for a wallet version. 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 entries (φ, t, u_2), (φ, t', u'_2) in the TSP's database. In this case, $\text{sk}_{\mathcal{U}}^{\text{id}}$ can be recovered as $\text{sk}_{\mathcal{U}}^{\text{id}} = (t - t')(u_2 - u'_2)^{-1} \bmod \mathfrak{p}$ with overwhelming probability. The cheater's public identification key $\text{pk}_{\mathcal{U}}^{\text{id}}$ can be computed from $\text{sk}_{\mathcal{U}}^{\text{id}}$. As a consequence, the wallet bound to $\text{pk}_{\mathcal{U}}^{\text{id}}$ could be blacklisted. The output of the protocol is the fraudulent user's identity along with a proof-of-guilt $\pi := \text{sk}_{\mathcal{U}}^{\text{id}}$.

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 $\text{pk}_{\mathcal{U}}^{\text{id}}$ and a proof-of-guilt π, it checks if π equals $\text{sk}_{\mathcal{U}}^{\text{id}}$ (which can be done by evaluating $g_1^{\pi} = \text{pk}_{\mathcal{U}}^{\text{id}}$).

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

5 SECURITY THEOREM

For our toll collection scheme we formally prove that under the Co-CDH assumption and security of our building blocks 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 2 (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 with regard to their 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 of RSUs, TSP and SA as well as a subset of users.

Be reminded that we assume the DR to be honest. In Appendix B.1 more details regarding the 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 B as well.

We now briefly describe the outline of the proof. The detailed formal statement including underlying assumptions and the full proof is postponed to Appendix E.

Proof Outline

For our security statement, we separately prove correctness, system security and user security and privacy.

Although proofs of correctness are often neglected for smaller UC-secure protocols, they are highly nontrivial for extensive ideal models as are required for a complex real-world task like toll collection. Since it is not only instructive to understand the underlying functionality but also a helpful basis for our proofs of system and user security, we briefly sketch the idea behind our proof of correctness here. The detailed formal proof can be found in Appendix E.1.

First, note that the entries $trdb = (s^{\text{prev}}, s, \varphi, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ of the ideal transaction database *TRDB* define a graph structure where serial numbers s are considered vertices and predecessors s^{prev} define edges (s^{prev}, s). Assigning a label $(pid_{\mathcal{R}}, p)$ to this edge and $(\varphi, \lambda, pid_{\mathcal{U}}, b)$ to the vertex s results in a graph where every vertex represents the state of a wallet and the incoming edge represents the transaction that led to this wallet state. We call this perception of *TRDB* the *Ideal Transaction Graph* and give graph-theoretic proofs of its structural properties. These properties include that the graph as a whole is a directed forest where each tree corresponds to a wallet ID λ, double-spending corresponds to branching and different wallet states have the same fraud detection ID φ if and only if they have the same depth in the same tree.

Secondly we add in- and out-commitments $(c_{\mathcal{R}}^{\text{in}}, c_{\mathcal{T}}^{\text{in}})$ and $(c_{\mathcal{R}}^{\text{out}}, c_{\mathcal{T}}^{\text{out}})$ from the real protocols to each transaction node in the Ideal Transaction Graph. These commitments are the fixed and updateable part of the wallet before and after the transaction (cp. Section 4). This information gives a second set of edges where two transactions *trdb* and *trdb** are connected if $(c_{\mathcal{R}}^{\text{out}}, c_{\mathcal{T}}^{\text{out}})$ corresponds to $(c_{\mathcal{R}}^{\text{in}\,*}, c_{\mathcal{T}}^{\text{in}\,*})$. We call the resulting graph the *Augmented Transaction Graph*. Showing that in case of an honest execution of the toll collection protocol both of those graph structures coincide with overwhelming probability yields correctness.

The proofs of system security and user security and privacy are conducted by explicitly specifying a simulator and reducing the indistinguishability of real and ideal world to the security of our building blocks. During an execution of the protocol our simulator generates the Augmented Transaction Graph as explained above. After showing that all messages sent by the simulator are statistically close to the real messages, i.e., simulated perfectly, that leaves only two kinds of reasons the environment $\mathcal{Z}$ could be able to distinguish both worlds. The first kind are *failure events* where the two graph structures in the augmented transaction graph diverge and the second

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} [ms]	$t_{\text{RSU/TSP}}$ [ms]	n_{user} [byte]	$n_{\text{RSU/TSP}}$ [byte]	t_{user} [ms]	$t_{\text{RSU/TSP}}$ [ms]	n_{user} [byte]	$n_{\text{RSU/TSP}}$ [byte]
Session Key Generation	14.97	2.30	130	–	14.98	2.42	130	–
Wallet Issuing	21205.86	30675.71	78975	1264	21306.07	30698.55	78995	1456
Debt Accumulation								
– Precomp (offline)	2677.31	–	–	–	2676.53	–	–	–
– Online	346.80	462.27	7968	976	453.72	479.34	8160	1072
– Online (cached certificate)	41.01	462.27	7968	976	40.23	479.34	8160	1072
– Postprocessing (offline)	34.22	–	–	–	36.35	–	–	–
Debt Clearance	2398.88	3062.49	7135	96	2399.32	3140.02	7328	96

are *discrepancy events* where some party's outputs could be distinguished. We show that both of those cases only occur with negligible probability by various reductions to our cryptographic building blocks and hardness assumptions. The full versions of those proofs can be found in Appendices E.2 and E.3.

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 [57]. 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 as a reference system, which was specifically "designed to enable third-party-developed or processor-intensive applications" [34] and measured on comparable hardware.

6.1 Building Block Instantiation

We implemented P4TC in C++14 using our own library for Groth-Sahai NIZK proofs [30, 33] and employed the method in [11] to realize the range proofs required in Wallet Issuing. We make use of two types of commitments: the shrinking commitment scheme from [2], as well as the (dual-mode) extractable commitment scheme from [33]. Moreover, we implemented the structure-preserving signature scheme from [1]. We use the ElGamal encryption scheme [29] for encrypting PRF key shares in the Wallet Issuing protocol, as well as the IND-CCA-secure encryption scheme from [19] (in combination with AES-CBC and HMAC-SHA256) to establish secure channels for exchanging protocol messages. The required PRF is instantiated with the Dodis-Yampolskiy construction introduced in [28]. For the underlying math we used the RELIC toolkit v.0.4.1, an open source cryptography and arithmetic library written in C under LGPL 2.1 license, with support for pairing-friendly elliptic curves [3].

6.2 Parameter Choice

As for the bilinear group setting, we use the Barreto-Naehrig curves Fp254BNb and Fp254n2BNb presented by Aranha et al. [8, 44]. For the pairing function, we use the optimal Ate pairing since it results in the shortest execution times [52]. This yields a security level of about 100 bit [6].

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. So 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 attribute 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 runtime 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 target platform, the user side algorithms were optimized for two CPU cores. Note that for simplicity, we assumed the same hardware for the TSP as for the RSU, although we can expect the TSP side to be equipped with much more powerful hardware.

6.3.1 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 2670ms. During the actual interaction with the RSU (*online* phase), the remaining part of the NIZK is computed and all data is transmitted. Computations in this online phase take only approximately 350ms (for $|\mathbf{a}_{\mathcal{U}}| = |\mathbf{a}_{\mathcal{R}}| = 1$) on the OBU, mostly due to the verification of the RSU certificate. When caching valid certificates, computations during the online phase can be reduced to approximately 40ms. After the OBU received a response from the RSU, the internal wallet has to be updated. Since this step can be done offline again, we measured the execution time separately and called 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. [37]. In this way, we obtain an online runtime of about 460ms. In summary, all computations in the online phase of Debt Accumulation can be performed in about 810ms or just 500ms when the certificate has already been cached.

The WAVE data transmission standard on DSRC guarantees a transmission rate of 24 Mbit/s [49]. 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 [47] 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 more than 600ms in case of an uncached certificate or one full second with a cached certificate. 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 24m at 120km/h. We therefore conclude that the performance is sufficient for real-world scenarios.

6.3.2 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 134 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 1.34MB 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 242 bytes of transaction information after each run of

[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. [19].

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 [35], which would result in a database of size 61GB. In case a wallet is blacklisted, an RSU is updated with a list of future fraud detection IDs. Each detection ID consumes 34 bytes of memory. Using appropriate data structures such as 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 34MB of memory, neglecting the overhead induced by the data structure.

6.3.3 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. [10] can be used to speed up this process. Using their algorithm, computing a discrete logarithm in an interval of order 2^{32} takes $1.93 \cdot 2^{32/3} \approx 3138$ multiplications with a table of 1626 precomputed elements, which can be generated with $1.24 \cdot 2^{64/3} \approx 3197118$ multiplications. However, this 55kB table only has to be generated once for the entire system. With this table, computing all 8 DLOGs requires approximately 25104 multiplications on average. Using only a single core on a standard desktop, this translates to 12 seconds of computation time. 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, 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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Ran Canetti. 2006. Security and Composition of Cryptographic Protocols: A Tutorial. Cryptology ePrint Archive, Report 2006/465. (2006). http://eprint.iacr.org/2006/465.

[15] 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.

[16] 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.

[17] 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.

[18] 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

[19] 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.

[20] 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.

[21] 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

[22] 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].

[23] 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].

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] Taher ElGamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology – CRYPTO'84 (Lecture Notes in Computer Science)*, G. R. Blakley and David Chaum (Eds.), Vol. 196. Springer, Heidelberg, 10–18.

[30] 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

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

[32] 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.

[33] 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.

[34] 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].

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

[36] 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

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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

[46] 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

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

[48] 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

[49] 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.

[50] 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].

[51] 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.*

[52] 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.

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

[54] 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].

[55] 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.

[56] Andy Rupp, Foteini Baldimtsi, Gesine Hinterwälder, and Christof Paar. 2015. Cryptographic Theory Meets Practice: Efficient and Privacy-Preserving Payments for Public Transport. *ACM Trans. Inf. Syst. Secur.* 17, 3 (2015), 10:1–10:31. https://doi.org/10.1145/2699904

[57] 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].

[58] 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 PRELIMINARIES

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.

A.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 [36].

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

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

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 $e\colon G_1 \times G_2 \to G_T$ 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 A.2) in asymmetric bilinear groups. For our special instantiation of the building blocks (see there), security holds under the SXDH assumption, which again 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 A.2 (DDH and SXDH Assumption).

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

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

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

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

The Co-CDH assumption is defined as follows:

Definition A.3 (Co-CDH assumption). We say that the Co-CDH assumption holds with respect to $\mathsf{SetupGrp}$ if the advantage $\mathsf{Adv}^{\mathsf{CO\text{-}CDH}}_{\mathsf{SetupGrp},\mathcal{A}}(1^n)$ defined by

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

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

A.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.

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

A.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 A.4 (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 $(\mathsf{gp}, x, w) \in R$. Let $\mathsf{POK} := (\mathsf{SetupGrp}, \mathsf{SetupPoK}, \mathsf{Prove}, \mathsf{Vfy})$ 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 $\mathsf{CRS}_{\mathsf{pok}}$.
- Prove takes as input the common reference string $\mathsf{CRS}_{\mathsf{pok}}$, a statement x, and a witness w with $(\mathsf{gp}, x, w) \in R$ 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 (SetupEPoK, 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 $\mathsf{Adv}^{\text{pok-ext}}_{\text{POK},\mathcal{A}}(n)$ defined by

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

is 1.

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

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

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

is negligible in n.

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

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

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

We wish to point out some remarks.

Remark A.5. (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 $\mathsf{td}_{\text{epok}}$—is able to extract $F_{\text{gp}}(\mathit{wit})$ for an NP-witness wit for $\mathit{stmnt} \in L_{\text{gp}}$ 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 [30, 33] 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 A.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 $F^{(1)}_{\text{gp}}$, $F^{(2)}_{\text{gp}}$ and $F^{(3)}_{\text{gp}}$ 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[17] 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 [11]. 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].

A.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 $F_\mathrm{gp}(m)$. We call the codomain of F_gp the implicit message space.

Definition A.6. A *commitment scheme* $\mathsf{COM} := (\mathsf{SetupGrp}, \mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ 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

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

[17] More precisely: The task Wallet Issuing

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

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

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

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

is negligible in n.

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

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

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

We also use the (dual-mode) equivocal and extractable commitment scheme from Groth and Sahai [33]. 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 $F'_{\mathsf{gp}}(m) := m$. In our system, we use this commitment scheme as C2 with CRS $\mathsf{CRS}^2_{\mathsf{com}}$ in the Wallet Issuing and Debt Accumulation tasks.

A.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 A.7. 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 $n \in \mathbb{N}$, $\mathsf{gp} \leftarrow \mathsf{SetupGrp}(1^n)$, $m \in \mathcal{M}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{gp})$, $\sigma \leftarrow \mathsf{Sgn}(\mathsf{sk}, m)$ 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 $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(1^n)$ defined by

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

is negligible in n, where $\mathsf{Sgn}(\mathsf{sk}, \cdot)$ is an oracle that, on input m, returns $\mathsf{Sgn}(\mathsf{sk}, m)$, 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 [46]. 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 $\mathsf{sk} \in \mathbb{Z}_{\mathrm{p}}^{\mu+\nu+2}$, $\mathsf{pk} \in G_1^{\mu+2} \times G_2^{\nu}$ 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 tasks Wallet Issuing and Debt Accumulation 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 $G_1^{2\ell+4}$ ($\nu = 2\ell + 4$ and $\mu = 0$).
- Also in the Wallet Issuing task we use S for messages from G_2^{1+j} ($\nu = 0$ and $\mu = 1 + j$).
- In the RSU Certification task we use S for messages from G_1^{3+y} ($\nu = 3 + y$ and $\mu = 0$).
- In the TSP Registration task we use S for messages from G_1^{3+y} ($\nu = 3 + y$ and $\mu = 0$).

A.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 A.8 (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.
- $\mathsf{Enc}(\mathsf{pk}, m)$ 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 $\perp$. We assume that Dec is deterministic.

Correctness is defined in the usual sense. We will make use of an IND-CPA-secure as well as an IND-CCA-secure encryption scheme.

Definition A.9 (IND-CPA-Security for Asymmetric Encryption). An asymmetric encryption scheme E is *IND-CPA*-secure if for all PPT adversaries $\mathcal{A}$ it holds that the advantage $\mathsf{Adv}_{\mathsf{E},\mathcal{A}}^{\mathsf{IND\text{-}CPA\text{-}sym}}(1^n)$ defined by

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

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

Definition A.10 (IND-CCA2-Security for Asymmetric Encryption). An asymmetric 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

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

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 $\perp$ on input c^*.

Our Instantiation. The encryption scheme in the construction of our base protocol requires IND-CPA-security and needs to be algebraic since we want to use it with GS proofs. This building block can be instantiated with the ElGamal encryption scheme [29] which is IND-CPA-secure under the DDH assumption.[18] We use this encryption scheme E for messages in G_1 in the Wallet Issuing task.

In order to establish secure channels for exchanging protocol messages we use the Twin-DH-based encryption scheme from Cash et al. [19].

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

Definition A.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 A.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 $\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}sym}}_{\mathsf{E},\mathcal{A}}(1^n)$ 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 section above) with an IND-CCA2-secure symmetric encryption in the usual KEM/DEM approach. The symmetric encryption can for example 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 [9].

A.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.

[18] The DDH assumption is implied by the SXDH assumption.

Definition A.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

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

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 [28]. This function is defined by $F(k, x) : \mathbb{Z}_p^2 \to G_1, (k, x) \mapsto g_1^{\frac{1}{x+k}}$, where $k \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$ is the random PRF key. It is secure for inputs $\{0, \dots, n_{\mathrm{PRF}}\} \subset \mathbb{Z}_p$ under the n_{PRF}-DDHI assumption. This is a family of increasingly stronger assumptions which is assumed to hold for asymmetric bilinear groups.

A.3 Types of Equations Supported by GS-NIZKs

Let SetupGrp be a bilinear group generator (cf. Definition A.1) for which the SXDH assumption holds and gp := $(G_1, G_2, G_T, e, p, g_1, g_2, g_T) \leftarrow \mathsf{SetupGrp}(1^n)$ denotes the output of SetupGrp. Furthermore, let $X_1, \dots, X_{m_1} \in G_1$, $x_1, \dots, x_{m_2} \in \mathbb{Z}_p$, $Y_1, \dots, Y_{m_3} \in G_2$, and $y_1, \dots, y_{m_4} \in \mathbb{Z}_p$ 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 $A_i \in G_1$, $B_i \in G_2$, $t_T \in G_T$, $\gamma_{i,j} \in \mathbb{Z}_p$.
- *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 $A_i, t_1 \in G_1$, $b_i, \gamma_{i,j} \in \mathbb{Z}_p$.
- *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 $B_i, t_2 \in G_2$, $a_i, \gamma_{i,j} \in \mathbb{Z}_p$.
- *Quadratic Equation (QE) over $\mathbb{Z}_p$:*
$$\sum_{i=1}^{m_4} a_i y_i + \sum_{i=1}^{m_2} x_i b_i + \sum_{i=1}^{m_2} \sum_{j=1}^{m_4} \gamma_{i,j} x_i y_j = t$$

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

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

where $m_i \in \mathbb{N}_0$. Then the Groth-Sahai proof system for L_{gp}, as introduced by [33], is perfectly correct, perfectly sound, and satisfies F_{gp}-extractability [30, 33] for

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

with

$$F_{gp}(w) := ((X_i)_{i\in[m_1]}, (g_1^{x_i})_{i\in[m_2]}, (Y_i)_{i\in[m_3]}, (g_2^{y_i})_{i\in[m_4]}) .$$[19]

It is also known to be composable zero-knowledge [30, 33] 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 $\mathrm{DLOG}(A_i)$ or $\mathrm{DLOG}(B_i)$ is known.

In the latter case, *hint* from Definition A.4 would contain these discrete logarithms which would simply be put (as additional elements) into the simulation trapdoor $\mathrm{td}_{\mathrm{spok}}$. Also note that if these discrete logarithms are not known there is a workaround which consists of adding new helper variables to L_{gp} [33].

B 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.

B.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.

Secondly, we only consider adversaries $\mathcal{Z}$ that corrupt one of the following sets[20]:

(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.

[19] 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 $s \in \mathbb{Z}_p$.
[20] Note that "subset" also includes the empty or full set.

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 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.

B.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.

B.3 Handling of Aborts

Lastly, we assume our functionality also uses the implicit writing conventions for ideal functionalities [13]. 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 B.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.

C 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 firstly 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[21] 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

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

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 $f_\Phi : (\mathcal{L} \times \mathbb{N}_0) \to \Phi$ 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

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

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: $f_{\mathcal{A}_{\mathcal{U}}} : \mathcal{L} \to \mathcal{A}_{\mathcal{U}}$ and $f_{\mathcal{A}_{\mathcal{R}}} : \mathcal{PID}_{\mathcal{R}} \to \mathcal{A}_{\mathcal{R}}$ 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).

[21]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

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

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

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

- 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:

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

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 2. 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 2 and 3.

C.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 3. Notation that occurs in the ideal functionality and in the real protocol

Identifier	Ideal Type	Real Type	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	public DR key
$\mathrm{pk}_{\mathcal{T}}$	$\mathcal{PK}_{\mathcal{T}}$	$G_1^3 \times (G_1^2 \times G_2^{3+y}) \times G_1^3$	public TSP key
$\mathrm{pk}_{\mathcal{R}}$	$\mathcal{PK}_{\mathcal{R}}$	G_1^3	public RSU key
$\mathrm{pk}_{\mathcal{U}}$	$\mathcal{PK}_{\mathcal{U}}$	$G_1 \times (G_1^2 \times G_2^2)$	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	$\mathbb{Z}_p$	$\mathbb{Z}_p$	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	$\mathcal{L}$	$\mathbb{Z}_p$	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 $G_1 \times (G_1^2 \times G_2^2)$ 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.

C.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 $\mathrm{pk}_{\mathcal{T}}$ is obtained. In addition all data structures of $\mathcal{F}_{\mathrm{P4TC}}$ are initialized as empty sets and empty (partial) mappings respectively. Again, the public key $\mathrm{pk}_{\mathcal{T}}$ is output to the TSP.

C.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: $(\texttt{register})$

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

(1) If this task has been run before, output $\perp$ and abort.
(2) $TRDB := \emptyset$
(3) Send $(\texttt{registering\_tsp}, pid_{\mathcal{T}}, \mathbf{a}_{\mathcal{T}})$ 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: $(\texttt{register})$

(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 $\mathbf{a}_{\mathcal{R}}$.[22] The content of attribute vectors is in no way restricted by $\mathcal{F}_{\text{P4TC}}$

[22]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: $(\texttt{register})$

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

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

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

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

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

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

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 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 $f_{\mathcal{A}_{\mathcal{R}}}(pid_{\mathcal{R}}) := \mathbf{a}_{\mathcal{R}}$ to the partial mapping $f_{\mathcal{A}_{\mathcal{R}}}$ which internally stores all RSU attributes already assigned. The identity $pid_{\mathcal{R}}$ and attributes $\mathbf{a}_{\mathcal{R}}$ are leaked to the adversary before the attributes are output to the RSU.

C.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.

C.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 $bl_{\mathcal{T}}$ of user public keys that are not allowed to obtain any new wallets. Firstly $\mathcal{F}_{\text{P4TC}}$ 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 wallet. Multiple corrupted users are allowed to have wallets issued for one another but are not able to request a new wallet for an honest user. The corresponding public key for the user ID $pid_{\mathcal{U}}$ is obtained from the bulletin board $\overline{\mathcal{G}}_{\text{bb}}$ and checked against the TSP's blacklist $bl_{\mathcal{T}}$. If this does not lead to an abort a new wallet ID λ and fraud detection ID φ are uniquely and randomly picked,

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 $s \overset{R}{\leftarrow} S$ that has not previously been used.
(2) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ leak $\mathbf{a}_{\mathcal{U}}$ to the adversary, and ask if another PID $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ should be used instead.[a]
(3) Receive $\text{pk}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.[⊥]
(4) If $\text{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$, output $\texttt{blacklisted}$ to both parties and abort.
(5) Pick wallet ID $\lambda \overset{R}{\leftarrow} \mathcal{L}$ that has not previously been used.
(6) If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\varphi \overset{R}{\leftarrow} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID φ that has not previously been used.[b] Append $f_{\Phi}(\lambda, 0) := \varphi$ to f_{Φ}.
(7) Append $trdb := (\perp, s, \varphi, 0, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, 0, 0)$ 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 $\perp$ and abort.

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

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

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

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. Lastly a transaction entry

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

corresponding to the new and empty wallet is stored in *TRDB* and the wallet's attributes $f_{\mathcal{A}_{\mathcal{U}}}(\lambda) := \mathbf{a}_{\mathcal{U}}$ 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.

C.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. Firstly, $\mathcal{F}_{\text{P4TC}}$ 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

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

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 $(\cdot, s^{\text{prev}}, \varphi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in \mathit{TRDB}$ (with s^{prev}, $pid_{\mathcal{U}}$ being the uniqe key)$^{\perp}$.
(4) If $\varphi^{\text{prev}} \in bl_{\mathcal{R}}$, 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 $\varphi := f_{\Phi}(\lambda, x)$.
Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\varphi \overset{\text{R}}{\leftarrow} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID φ that has not previously been used.[b] Append $f_{\Phi}(\lambda, x) := \varphi$ 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 $(s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ to *TRDB*.

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

$^{\perp}$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 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} increased by one to obtain x. $\mathcal{F}_{\text{P4TC}}$ checks if there is already a fraud detection ID $\varphi := f_{\Phi}(\lambda, x)$ assigned to the pair (λ, x) (either because the user committed double spending or because it has been precalculated for blacklisting purposes). If not and the user is honest it picks a new φ uniquely at random. If the user is corrupted the fraud detection ID is not randomly drawn but picked by the adversary. This may infringe upon the unlinkability of the user's transactions but as mentioned before, we do not give any privacy guarantees for corrupted users. The 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. Lastly 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

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 $s \overset{R}{\leftarrow} S$ 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 $(\cdot, s^{\text{prev}}, \varphi^{\text{prev}}, x^{\text{prev}}, \lambda^{\text{prev}}, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, \cdot, b^{\text{prev}}) \in TRDB$ (with s^{prev}, $pid_{\mathcal{U}}$ being the unique key).$^{\perp}$
(4) Set $\lambda := \lambda^{\text{prev}}$ and $x := x^{\text{prev}} + 1$.
(5) If $f_{\Phi}(\lambda, x)$ is already defined, set $\varphi := f_{\Phi}(\lambda, x)$.
Else, if $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ pick $\varphi \overset{R}{\leftarrow} \Phi$ that has not previously been used, otherwise ask the adversary for a fraud detection ID φ that has not previously been used.[b] Append $f_{\Phi}(\lambda, x) := \varphi$ to f_{Φ}.
(6) 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.
(7) $b^{\text{bill}} := b^{\text{prev}}$.
(8) Append $(s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{T}}, -b^{\text{bill}}, 0)$ to *TRDB*.

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

$^{\perp}$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)

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.

C.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 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

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

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.

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

User input: (`prove_participation`)

SA input: (`prove_participation`, $pid_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}$)

(1) If $pid_{\mathcal{U}} \in \mathcal{PID}_{\text{corrupt}}$ leak $S^{\text{pp}}_{\mathcal{R}}$ to the adversary.

(2) If $\exists\, (\cdot, s, \cdot, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, \cdot, \cdot) \in \mathit{TRDB}$ such that $s \in S^{\text{pp}}_{\mathcal{R}}$,
then $\mathtt{out}_{\mathcal{U}} := \mathtt{out}_{SA} := \mathtt{OK}$
else $\mathtt{out}_{\mathcal{U}} := \mathtt{out}_{SA} := \mathtt{NOK}$

User output: ($\mathtt{out}_{\mathcal{U}}$)

SA output: ($\mathtt{out}_{SA}$)

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

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

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

TSP input: (`scan_for_fraud`, φ)

(1) Pick $trdb \neq trdb'$ in TRDB such that $trdb = (\cdot, \cdot, \varphi, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, \cdot, \cdot)$ and $trdb' = (\cdot, \cdot, \varphi, \cdot, \cdot, pid_{\mathcal{U}}, \cdot, \cdot, \cdot)$.$^{\perp}$

(2) Ask the adversary for a proof $\pi \in \Pi$ corresponding to $pid_{\mathcal{U}}$ and append $(pid_{\mathcal{U}}, \pi) \mapsto \mathtt{OK}$ to f_{Π}.

TSP output: ($pid_{\mathcal{U}}, \pi$)

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

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

C.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.

C.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 physically identified by one of the SA's cameras. It is meant to allow every 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. This deanonymizes the one transaction proven by the user but does not effect the anonymity or unlinkability of any other transactions. As input the task requires a user ID $pid_{\mathcal{U}}$ and a set $S^{\text{pp}}_{\mathcal{R}}$ of serial numbers in question from the SA, but only consent from the user. If the user is corrupted, the serial numbers are leaked to the adversary. $\mathcal{F}_{\text{P4TC}}$ then checks if there is a transaction recorded in TRDB with the user ID $pid_{\mathcal{U}}$ and a serial number s contained in $S^{\text{pp}}_{\mathcal{R}}$. The result of the check (`OK` or `NOK`) is output to both parties.

C.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, $\mathcal{F}_{\text{P4TC}}$ provides the tasks Double-Spending Detection (cp. Fig. 14) and Guilt Verification (cp. Fig. 15).

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

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

1. If $f_{\Pi}(pid_{\mathcal{U}}, \pi)$ is defined, then set $\texttt{out} := f_{\Pi}(pid_{\mathcal{U}}, \pi)$ and output $(\texttt{out})$.
2. 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 $\texttt{out} := \texttt{NOK}$.
3. Append $(pid_{\mathcal{U}}, \pi) \mapsto \texttt{out}$ to f_{Π},

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

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

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

Guilt Verification is a one-party task as well but can be performed by any party. It takes a user ID $pid_{\mathcal{U}}$ and a double-spending proof π as input. First, it checks if this particular pair $(pid_{\mathcal{U}}, \pi)$ has already been defined and outputs whatever has been output before. This is necessary to ensure consistency across different invocations. If $(pid_{\mathcal{U}}, \pi)$ has neither been issued nor queried before *and* the affected user is corrupted, the adversary is allowed to decide, if this proof is should be accepted. This means, 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 user 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.

C.3.3 User Blacklisting. The task of User Blacklisting (cp. Fig. 16) is a two-party task between the DR and TSP and serves two purposes: Firstly, the debt b^{bill} owed by the user that is to be blacklisted is calculated. Secondly, 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 blacklists $bl_{\mathcal{R}}$. 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 $\mathcal{F}_{\text{P4TC}}$.

The TSP inputs the ID $pid_{\mathcal{U}}$ of the user in question while the DR only needs to consent. To calculate the user's outstanding debt, all transaction entries in *TRDB* containing $pid_{\mathcal{U}}$ are taken and their respective prices p summed up to obtain b^{bill}. Note that although this sum may contain the prices of transactions and wallets that have already been cleared, this does not falsify the value of b^{bill} as every successful execution of Debt Clearance creates an entry with the amount that was cleared as negative price. For the actual blacklisting the set of all wallet IDs belonging to $pid_{\mathcal{U}}$ is looked up and the remainder of the task is conducted for every wallet λ separately. $\mathcal{F}_{\text{P4TC}}$ checks how many values of $f_{\Phi}(\lambda, \cdot)$ are already defined and extends them to the first $x_{\text{bl}_{\mathcal{R}}}$ fraud detection IDs, where $x_{\text{bl}_{\mathcal{R}}}$ 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 $x \le x_{\text{bl}_{\mathcal{R}}}$ are uniquely and randomly drawn or—in case of a corrupted user—obtained from the adversary. Finally, all fraud detection IDs $\varphi = f_{\Phi}(\lambda, x)$ for $x \le x_{\text{bl}_{\mathcal{R}}}$ and all wallets λ of the user are output to the TSP together with the outstanding debt b^{bill}.

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

DR input: (`blacklist_user`)
TSP input: (`blacklist_user`, $pid_{\mathcal{U}}$)

(1) If $pid_{\mathcal{T}} \notin \mathcal{PID}_{\text{corrupt}}$, set $\mathcal{L}_{\text{bl}} := \{\lambda \mid (\cdot,\cdot,\cdot,\cdot,\lambda, pid_{\mathcal{U}},\cdot,\cdot,\cdot) \in \mathit{TRDB}\}$, otherwise obtain a set of serial numbers S_{root} from the adversary and set $\mathcal{L}_{\text{bl}} := \{\lambda \mid (\bot, s,\cdot,\cdot,\lambda, pid_{\mathcal{U}},\cdot,\cdot,\cdot) \in \mathit{TRDB}$ with $s \in S_{\text{root}}\}$.
(2) $\mathit{TRDB}_{\text{bl}} := \{\mathit{trdb} \in \mathit{TRDB} \mid \mathit{trdb} = (\cdot,\cdot,\cdot,\cdot,\lambda,\cdot,\cdot,p,\cdot) \text{ s.t. } \lambda \in \mathcal{L}_{\text{bl}}\}$
(3) $b^{\text{bill}} := \sum_{\mathit{trdb} \in \mathit{TRDB}_{\text{bl}}} p$.
(4) For each $\lambda \in \mathcal{L}_{\text{bl}}$:
  (a) $x_\lambda := \max\{\, x \mid f_\Phi(\lambda, x) \text{ is already defined}\}$.
  (b) For $x \in \{x_\lambda + 1, \ldots, x_{\text{bl}_{\mathcal{R}}}\}$:
    (i) If $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$, pick $\varphi \overset{\text{R}}{\leftarrow} \Phi$ 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 $(\lambda, x) \mapsto \varphi$ to f_Φ.
(5) $\Phi_{\text{bl}} := \{f_\Phi(\lambda, x) \mid \lambda \in \mathcal{L}_{\text{bl}}, 0 \le x \le x_{\text{bl}_{\mathcal{R}}}\}$.

DR output: (`OK`)
TSP output: $(b^{\text{bill}}, \Phi_{\text{bl}})$

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

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 E. The style of the presentation follows the same lines as the presentation of the ideal model $\mathcal{F}_{\text{P4TC}}$ in Appendix C: 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. 17. 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:

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 *AHTD* of augmented hidden 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. 19)
- *TSP Registration* (Fig. 21)
- *RSU Registration* (Fig. 23)
- *User Registration* (Fig. 25)
- *RSU Certification* (Fig. 27)
- *Wallet Issuing* (Fig. 29)
- *Debt Accumulation* (Fig. 31)
- *Debt Clearance* (Fig. 33)
- *Prove Participation* (Fig. 35)
- *Double-Spending Detection* (Fig. 37)
- *Guilt Verification* (Fig. 38)
- *User Blacklisting* (Fig. 39)

Fig. 17. The UC-protocol π_{P4TC}

(1) We give an algorithm for the setup of the system (cf. Fig. 18) 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. 41). 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. 37 and 38) because they are so simple that splitting it each into two yields no advantage.

D.1 Secure 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.

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

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

```
Setup(1^n, 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 CRS
```

Fig. 18. System Setup Algorithm

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 identification key $\mathsf{sk}^{\mathsf{id}}_{\mathcal{U}}$), 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* $\varphi := \mathsf{PRF}(\lambda, x^{\mathsf{next}} - 1)$ for the current transaction. These components change after each interaction with a 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 3 and 4.

D.3 System Setup

To setup the system once (see Fig. 18), 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. 19, 21, 23 and 25), and a core protocol (see Figs. 20, 22, 24 and 26). 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.

The Dispute Resolver (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. 19 and 20). The DR could be a (non-governmental) organization trusted by both, users to protect their privacy and the TSP to protect system security.

The TSP must also generate a key pair (see Figs. 21 and 22). Therefore, the TSP generates several signature key pairs $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{T}}_{\mathcal{T}})$, $(\mathsf{pk}^{\mathsf{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{cert}}_{\mathcal{T}})$, $(\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}})$, where $\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}$ is used in the Wallet Issuing task to sign the TSP

Table 4. Notation that only occurs in the real protocol

Identifier	**Type**	**Description**
sk_{DR}	$\mathbb{Z}_p$	secret DR key
$\mathsf{sk}_{\mathcal{T}}$	$\mathbb{Z}_p^3 \times \mathbb{Z}_p^{5+y} \times \mathbb{Z}_p^3$	secret TSP key
$\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}$	G_1^3	public TSP commitment signing key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathcal{T}}$	$\mathbb{Z}_p^3$	secret TSP commitment signing key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}}$	$G_1^2 \times G_2^{3+y}$	public certification key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathrm{cert}}$	$\mathbb{Z}_p^{5+y}$	secret certification key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}$	G_1^3	public TSP's RSU commitment signing key (part of $\mathsf{pk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{T}}^{\mathcal{R}}$	$\mathbb{Z}_p^3$	secret TSP's RSU commitment signing key (part of $\mathsf{sk}_{\mathcal{T}}$)
$\mathsf{sk}_{\mathcal{R}}$	$\mathbb{Z}_p^3$	secret RSU key
$\mathsf{sk}_{\mathcal{U}}$	$\mathbb{Z}_p \times \mathbb{Z}_p^4$	secret user key
$\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}}$	G_1	public user identification key (part of $\mathsf{pk}_{\mathcal{U}}$)
$\mathsf{sk}_{\mathcal{U}}^{\mathrm{id}}$	$\mathbb{Z}_p$	secret user identification key (part of $\mathsf{sk}_{\mathcal{U}}$)
$\mathsf{pk}_{\mathcal{U}}^{\mathrm{auth}}$	$G_1^2 \times G_2^2$	public user authentication key (part of $\mathsf{pk}_{\mathcal{U}}$)
$\mathsf{sk}_{\mathcal{U}}^{\mathrm{auth}}$	$\mathbb{Z}_p^4$	secret user authentication key (part of $\mathsf{sk}_{\mathcal{U}}$)
$c_{\mathcal{T}}$	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''_{ser}	G_1^2	commitment on the RSU half of the serial number
d''_{ser}	$\mathbb{Z}_p^2$	opening of c''_{ser}
ω^{dsp}	$G_1 \times \mathbb{Z}_p \times \mathbb{Z}_p$	transaction information for double-spending
ω^{bl}	$G_1 \times \mathbb{Z}_p$	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	$\mathbb{Z}_p$	double-spending mask
u_2	$\mathbb{Z}_p$	double-spending randomness
t	$\mathbb{Z}_p$	double-spending tag
htd	$\mathbb{Z}_p \times G_2 \times G_1 \times G_1^{2\ell+4} \times (G_2^2 \times G_1)$	hidden user trapdoor
HTD	set of $\mathbb{Z}_p \times G_2 \times G_1 \times G_1^{2\ell+4} \times (G_2^2 \times G_1)$ elements	set of hidden trapdoors
$ahtd$	$\mathcal{PID}_{\mathcal{U}} \times S \times HTD$	augmented hidden user trapdoor
$AHTD$	set of $\mathcal{PID}_{\mathcal{U}} \times S \times HTD$ elements	set of augmented hidden trapdoors
n_{PRF}	$\mathbb{N}$	maximum value of the PRF counter

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

DR input: (`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. 20).
(4) Record $(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{DR}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input (`register`, $\mathsf{pk}_{\mathcal{DR}}$).

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

Fig. 19. The UC-protocol π_{P4TC} (cont. from Fig. 17)

```
DRRegistration(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. 20. DR Registration Core Protocol

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

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

(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 $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathsf{TSPRegistration}(\mathsf{CRS}, \mathsf{a}_{\mathcal{T}})$ (see Fig. 22).
(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 (`register`, $\mathsf{pk}_{\mathcal{T}}$).

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

Fig. 21. The UC-protocol π_{P4TC} (cont. from Fig. 17)

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

Fig. 22. TSP Registration Core Protocol

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

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

(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. 24).
(4) Store $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{R}})$.

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

Fig. 23. The UC-protocol π_{P4TC} (cont. from Fig. 17)

```
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. 24. RSU Registration Core Protocol

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

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

(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. 26).
(4) Store $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ internally and call $\overline{\mathcal{G}}_{\text{bb}}$ with input $(\texttt{register}, \mathsf{pk}_{\mathcal{U}})$.

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

Fig. 25. The UC-protocol π_{P4TC} (cont. from Fig. 17)

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

Fig. 26. User Registration Core Protocol

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 $pid_{\mathcal{T}}$.$^{\perp}$

(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 $pid_{\mathcal{R}}$.$^{\perp}$
- 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. 28)

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

(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. 27. The UC-protocol π_{P4TC} (cont. from Fig. 17)

commitment $c_{\mathcal{T}}$ and the user attributes $\mathbf{a}_{\mathcal{U}}$, $\mathsf{sk}^{\text{cert}}_{\mathcal{T}}$ is used to sign RSU public keys in the RSU Certification task and $\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}$ is used in the Wallet Issuing task to sign the RSU commitment $c_{\mathcal{R}}$ and the serial number s in place of a RSU. The TSP also generates a certificate $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$ for its own key $\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}$.

Each RSU must generate a key pair as well (see Figs. 23 and 24). 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. 25 and 26). The public key of the *user identification key pair* $(\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{sk}^{\text{id}}_{\mathcal{U}})$ 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. Each user also creates a *user authentication key pair* $(\mathsf{pk}^{\text{auth}}_{\mathcal{U}}, \mathsf{sk}^{\text{auth}}_{\mathcal{U}})$ that is used to bind the *hidden user trapdoor htd* in Wallet Issuing and User Blacklisting to the user.

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. 27) and a core protocol RSUCertification (see Fig. 28). 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 $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$		parse $(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{pk}_{\mathcal{T}}$
		parse $(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{sk}_{\mathcal{T}}$
		$\sigma^{\mathrm{cert}}_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}))$
		$\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma^{\mathrm{cert}}_{\mathcal{R}})$
	$\xleftarrow{\mathsf{cert}_{\mathcal{R}}}$	
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 $\perp$		
return $(\mathsf{cert}_{\mathcal{R}})$		return (OK)

Fig. 28. 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. 29) and a core protocol WalletIssuing (see Fig. 30). 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. 41) after the core protocol. The wrapper protocol internally invokes the core protocol

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

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 $\mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$ 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 hidden fashion 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 a preliminary wallet ID λ' and λ'', respectively, that together form the wallet ID $\lambda := \lambda' + \lambda''$. Note that the TSP does not learn the wallet ID.

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

User input: (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 $\overline{\mathcal{G}}_{\mathrm{bb}}$ for PID $pid_{\mathcal{T}}$.$^{\perp}$
- Receive pk_{DR} from the bulletin-board $\overline{\mathcal{G}}_{\mathrm{bb}}$ 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 $\overline{\mathcal{G}}_{\mathrm{bb}}$ for PID pid_{DR}.$^{\perp}$

(3) Both sides: Run the code of WalletIssuing between the user and the TSP (see Fig. 30)

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

(4) At the user side:
- Run the code of $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$ (see Fig. 41).
- 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:
- Receive $pid_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\mathrm{bb}}$ for public key $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}}$.
- Insert $(pid_{\mathcal{U}}, s, htd)$ into *AHTD*.

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

TSP output: (s)

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

Fig. 29. The UC-protocol π_{P4TC} (cont. from Fig. 17)

For the second objective it would not be sufficient for the user to simply commit to λ', since then one could only prove that $g_1^{\lambda'}$ is contained. Considering the DLog assumption in G_1, recovering $\lambda' \in \mathbb{Z}_p$ from $g_1^{\lambda'}$ is infeasible. Therefore, the user splits λ' into $\lambda'_0, \ldots, \lambda'_\ell \in \{0, \ldots, \mathcal{B} - 1\}$ s.t. $\lambda' = \sum_{i=0}^{l} \lambda'_i \cdot \mathcal{B}^i$ for some splitting 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 each $g_1^{\lambda'_i}$ under the public key of the DR and transmits these encryptions e_i to the TSP. To bind these encryptions to the user and to prevent misuse from the TSP, the user also encrypts $g_1^{\lambda'}$ and signs the resulting ciphertext e^* along with the ciphertexts $e_0, \ldots, e_\ell$ with S under his secret user authentication key $\mathsf{sk}_{\mathcal{U}}^{\mathrm{auth}}$. The ciphertext e^* and signature σ^* are also sent to the TSP. The TSP creates the hidden user trapdoor as $htd := (\lambda'', e_0, \ldots, e_\ell, e^*, \sigma^*)$. In the case that the user commits a fraud, the TSP can send all htd of this user along with $\mathsf{pk}_{\mathcal{U}}$ to the DR and the DR can recover the wallet ID λ. The DR in this case also checks that htd and $\mathsf{pk}_{\mathcal{U}}$ match (by verifying σ^*). For more details see the User Blacklisting task in Fig. 40.

The goal of the third objective is to create a truly random serial number $s \in S$ for this transaction (we use $S := G_1$). To ensure that the serial number is indeed random (and not maliciously chosen by the user or the TSP),

$\mathcal{U}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$		$\mathcal{T}(\mathsf{pk}_{\mathcal{DR}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, bl_{\mathcal{T}})$
parse $\left((\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{pk}^{\mathrm{auth}}_{\mathcal{U}}), (\mathsf{sk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{sk}^{\mathrm{auth}}_{\mathcal{U}})\right) := (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$		parse $(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{sk}^{\mathrm{cert}}_{\mathcal{T}}, \mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}) := \mathsf{sk}_{\mathcal{T}}$
$s' \overset{\mathrm{R}}{\leftarrow} S$		$s'' \overset{\mathrm{R}}{\leftarrow} S$
$\lambda' \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$		$\lambda'' \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$
$u_1^{\mathrm{next}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$		
Split λ' into $\lambda'_0, \dots, \lambda'_\ell \in \{0, \dots, \mathcal{B} - 1\}$		
s.t. $\lambda' = \sum_{i=0}^{\ell} \lambda'_i \cdot \mathcal{B}^i$		
$\forall i \in \{0, \dots, \ell\}$:		
$\quad r_i \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$		
$\quad e_i \leftarrow \mathsf{E.Enc}(\mathsf{pk}_{\mathcal{DR}}, g_1^{\lambda'_i}; r_i)$		
$r^* \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_\mathrm{p}$		
$e^* \leftarrow \mathsf{E.Enc}(\mathsf{pk}_{\mathcal{DR}}, g_1^{\lambda'}; r^*)$		
$(c'_{\mathcal{T}}, d'_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda', \mathsf{sk}^{\mathrm{id}}_{\mathcal{U}}))$		$(c''_{\mathcal{T}}, d''_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda'', 0))$
$(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda', 0, u_1^{\mathrm{next}}, 0))$		$(c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) \leftarrow \mathsf{C2.Com}(\mathsf{CRS}^2_{\mathrm{com}}, s'')$
$stmnt := (\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{DR}}, e^*, e_0, \dots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$		
$wit := (\lambda', \lambda'_0, \dots, \lambda'_\ell, r^*, r_1, \dots, r_\ell,$		
$g_1^{\lambda'}, g_1^{\lambda'_0}, \dots, g_1^{\lambda'_\ell}, g_1^{u_1^{\mathrm{next}}}, d'_{\mathcal{R}}, d'_{\mathcal{T}}, g_2^{\mathsf{sk}^{\mathrm{id}}_{\mathcal{U}}})$		
$\pi \leftarrow \mathsf{P1.Prove}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, wit)$		
	$\xleftarrow{\mathsf{a}_{\mathcal{U}}, c''_{\mathrm{ser}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, c''_{\mathcal{T}}}$	
parse $(\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}) := \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}$		
if $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{cert}}_{\mathcal{T}}, \sigma^{\mathrm{cert}}_{\mathcal{T}}, (\mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}})) = 0$		
$\quad$ return $\perp$		
$\sigma^* \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathrm{auth}}_{\mathcal{U}}, (c''_{\mathcal{T}}, e_0, \dots, e_\ell, e^*))$		
	$\xrightarrow{\mathsf{pk}_{\mathcal{U}}, s', e^*, \sigma^*, \pi, e_0, \dots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}}}$	
		parse $(\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{pk}^{\mathrm{auth}}_{\mathcal{U}}) := \mathsf{pk}_{\mathcal{U}}$
		if $\mathsf{pk}_{\mathcal{U}} \in bl_{\mathcal{T}}$
		$\quad$ return `blacklisted`
		if $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{auth}}_{\mathcal{U}}, \sigma^*, (c''_{\mathcal{T}}, e_0, \dots, e_\ell, e^*)) = 0$
		$\quad$ return $\perp$
		$stmnt := (\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{DR}}, e^*, e_0, \dots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$
		if $\mathsf{P1.Vfy}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, \pi) = 0$
		$\quad$ return $\perp$
		$c_{\mathcal{T}} := c'_{\mathcal{T}} \cdot c''_{\mathcal{T}}$
		$\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (c_{\mathcal{T}}, \mathsf{a}_{\mathcal{U}}))$
		$s := s' \cdot s''$
		$(c''_{\mathcal{R}}, d''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathrm{com}}, (\lambda'', 0, 0, 1))$
		$c_{\mathcal{R}} := c'_{\mathcal{R}} \cdot c''_{\mathcal{R}}$
		$\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (c_{\mathcal{R}}, s))$
	$\xleftarrow{s'', d''_{\mathrm{ser}}, \lambda'', c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, c_{\mathcal{T}}, d''_{\mathcal{T}}, \sigma_{\mathcal{T}}}$	
if $\mathsf{C2.Open}(\mathsf{CRS}^2_{\mathrm{com}}, s'', c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) = 0$		
$\quad$ return $\perp$		
$\tau := ((s' \cdot s''), \mathsf{PRF}(\lambda, 0), 1, (\lambda' + \lambda''), \mathsf{a}_{\mathcal{U}},$		
$\quad c_{\mathcal{R}}, (d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}), \sigma_{\mathcal{R}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, c_{\mathcal{T}}, (d'_{\mathcal{T}} \cdot d''_{\mathcal{T}}), \sigma_{\mathcal{T}}, 0, u_1^{\mathrm{next}})$		$htd := (\lambda'', c''_{\mathcal{T}}, d''_{\mathcal{T}}, e_0, \dots, e_\ell, e^*, \sigma^*)$
return (τ)		return $(s, \mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, htd)$

Fig. 30. Wallet Issuing Core Protocol

the user and the TSP engage in a Blum coin toss. First they randomly choose a $s' \overset{R}{\leftarrow} S$, respectively $s'' \overset{R}{\leftarrow} S$. Then the TSP creates a commitment c''_{ser} on s'' using the equivocable and extractable commitment scheme C2. The TSP then transmits c''_{ser} to the user, the user transmits c'_{ser} to the TSP, the TSP transmits c''_{ser} and the opening of c''_{ser} to the user and finally the user checks if the commitment c''_{ser} is valid. If yes, the serial number is defined as $s := s' + s''$ and is known to both the user and the TSP.

For the last objective, the user generates preliminary TSP and RSU commitments. He commits to his preliminary wallet ID λ' and his secret user identification key $\text{sk}^{\text{id}}_{\mathcal{U}}$ for the preliminary TSP commitment $c'_{\mathcal{T}}$. For the preliminary RSU commitment $c'_{\mathcal{R}}$, he commits to his preliminary wallet ID λ', the balance $b := 0$, a fresh double-spending mask u_1^{next} and the PRF counter $x := 0$. He then computes a proof showing that these commitments are formed correctly and bound to his public identification key. In the proof it is also shown that the encryptions $e_0, \ldots, e_\ell, e^*$ are formed correctly 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 $L^{(1)}_{\text{gp}}$ defined by

$$L^{(1)}_{\text{gp}} := \left\{ \begin{pmatrix} \text{pk}^{\text{id}}_{\mathcal{U}} \\ \text{pk}_{\mathcal{DR}} \\ e^* \\ e_0 \\ \vdots \\ e_\ell \\ c'_{\mathcal{R}} \\ c'_{\mathcal{T}} \end{pmatrix}^{\top} \middle| \begin{array}{l} \exists\, \lambda', \lambda'_0, \ldots, \lambda'_\ell, r^*, r_1, \ldots, r_\ell \in \mathbb{Z}_p; \\ \Lambda', \Lambda'_0, \ldots, \Lambda'_\ell, U_1^{\text{next}}, d'_{\mathcal{R}}, d'_{\mathcal{T}} \in G_1; \text{SK}^{\text{id}}_{\mathcal{U}} \in G_2 : \\ e(\text{pk}^{\text{id}}_{\mathcal{U}}, g_2) = e(g_1, \text{SK}^{\text{id}}_{\mathcal{U}}) \\ \text{C1.Open}(\text{CRS}^1_{\text{com}}, (\Lambda', \text{pk}^{\text{id}}_{\mathcal{U}}), c'_{\mathcal{T}}, d'_{\mathcal{T}}) = 1 \\ \text{C1.Open}(\text{CRS}^1_{\text{com}}, (\Lambda', 1, U_1^{\text{next}}, 1), c'_{\mathcal{R}}, d'_{\mathcal{R}}) = 1 \\ \Lambda' = g_1^{\lambda'},\ \lambda' = \sum_{i=0}^{\ell} \lambda'_i \cdot \mathcal{B}^i \\ e^* = \text{E.Enc}(\text{pk}_{\mathcal{DR}}, \Lambda'; r^*) \\ \forall i \in \{0, \ldots, \ell\} : \\ \qquad \lambda'_i \in \{0, \ldots, \mathcal{B} - 1\} \\ \qquad e_i = \text{E.Enc}(\text{pk}_{\mathcal{DR}}, \Lambda'_i; r_i) \\ \qquad \Lambda'_i = g_1^{\lambda'_i} \end{array} \right\} \tag{2}$$

Note that the first equation in Eq. (2) actually proves the knowledge of $g_2^{\text{sk}^{\text{id}}_{\mathcal{U}}}$ (rather than $\text{sk}^{\text{id}}_{\mathcal{U}}$ itself).[23] However, computing $g_2^{\text{sk}^{\text{id}}_{\mathcal{U}}}$ without knowing $\text{sk}^{\text{id}}_{\mathcal{U}}$ (only given $\text{pk}^{\text{id}}_{\mathcal{U}}$) is assumed to be a hard problem (Co-CDH). The user then transmits his public key, the proof and the two preliminary commitments to the TSP. The TSP first checks if the user's public key is contained in the TSP blacklist $bl_{\mathcal{T}}$. If yes, the protocol is aborted and the user does not get a fresh wallet. Else, the protocol continues. Then the TSP verifies the signature σ^* and the proof π. The TSP then uses the homomorphic property of the commitment scheme to create the final TSP and RSU commitments. It adds λ'' to the preliminary TSP commitment to create a TSP commitment $c_{\mathcal{T}}$ on $(\lambda, \text{sk}^{\text{id}}_{\mathcal{U}})$. It adds λ'' to the preliminary RSU commitment and sets the PRF counter to 1 to create a RSU commitment $c_{\mathcal{R}}$ on $(\lambda, 0, u_1^{\text{next}}, 1)$. The TSP also creates a signature $\sigma_{\mathcal{T}} \leftarrow \text{S.Sgn}(\text{sk}^{\mathcal{T}}_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}))$ and a signature $\sigma_{\mathcal{R}} \leftarrow \text{S.Sgn}(\text{sk}^{\mathcal{R}}_{\mathcal{T}}, (c_{\mathcal{R}}, s))$. The TSP then sends its preliminary wallet ID λ'', the TSP and RSU commitments along with their openings and signatures to the user. The user then assembles his wallet

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

At the end of the protocol, the user returns the wallet τ. The TSP returns the user's public identification key $\text{pk}^{\text{id}}_{\mathcal{U}}$ and the hidden user trapdoor *htd*.

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

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

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

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

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

(3) Both sides: Run the code of DebtAccumulation between the user and the RSU (see Fig. 32)

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

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

(4) At the user side:
- Run the code of $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau)$ (see Fig. 41).
- If WalletVerification returns 0, output $\perp$ and abort.
- Record τ and $\omega^{\text{pp}}_{\mathcal{U}}$ internally.
- Parse s, $\mathsf{cert}_{\mathcal{R}}$, p and b from τ.
- Parse $\mathbf{a}_{\mathcal{R}}$ from $\mathsf{cert}_{\mathcal{R}}$.

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

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

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

Fig. 31. The UC-protocol π_{P4TC} (cont. from Fig. 17)

D.7 Debt Accumulation

When a driving car passes a 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. 31) and a core protocol DebtAccumulation (see Fig. 32). 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. 41) after the core protocol has terminated. The wrapper protocol internally invokes the core protocol

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

```
$\mathcal{U}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\mathrm{prev}})$

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

$\mathcal{R}^{\mathsf{O}_{\mathrm{pricing}}(\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}, bl_{\mathcal{R}})$

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

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

parse $(\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}}, \sigma_{\mathcal{R}}^{\mathrm{cert}}) := \mathsf{cert}_{\mathcal{R}}$
if $\mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}}, \sigma_{\mathcal{R}}^{\mathrm{cert}}, (\mathsf{pk}_{\mathcal{R}}, \mathbf{a}_{\mathcal{R}})) = 0$
    return $\perp$
$t := \mathsf{sk}_{\mathcal{U}}^{\mathrm{id}} u_2 + u_1 \mod p$
$(c_{\mathrm{hid}}, d_{\mathrm{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}_{\mathrm{com}}^1, \mathsf{sk}_{\mathcal{U}}^{\mathrm{id}})$
$stmnt := (\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\mathrm{cert}}, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t, u_2)$
$wit := (x, \lambda, \mathsf{sk}_{\mathcal{U}}^{\mathrm{id}}, u_1, s^{\mathrm{prev}}, \varphi^{\mathrm{prev}}, g_1^x, g_1^{\lambda}, \mathsf{pk}_{\mathcal{U}}^{\mathrm{id}}, g_1^{b^{\mathrm{prev}}}, g_1^{u_1}, g_1^{u_1^{\mathrm{next}}},$
    $d_{\mathrm{hid}}, d_{\mathcal{R}}^{\mathrm{prev}}, d'_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathcal{T}}, \sigma_{\mathcal{R}}^{\mathrm{prev}}, \sigma_{\mathcal{R}}^{\mathrm{cert prev}}, \sigma_{\mathcal{T}})$
$\pi \leftarrow \mathsf{P2.Prove}(\mathsf{CRS}_{\mathrm{pok}}, stmnt, wit)$

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

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

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

if $\mathsf{C2.Open}(\mathsf{CRS}_{\mathrm{com}}^2, s'', c''_{\mathrm{ser}}, d''_{\mathrm{ser}}) = 0$
    return $\perp$
$s := s' \cdot s''$
$d_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$
$b := b^{\mathrm{prev}} + p$
$x^{\mathrm{next}} := x + 1$
$\tau := (s, \varphi, x^{\mathrm{next}}, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}, d_{\mathcal{R}}, \sigma_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}}, c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b, u_1^{\mathrm{next}})$
$\omega_{\mathcal{U}}^{\mathrm{pp}} := (s, c_{\mathrm{hid}}, d_{\mathrm{hid}})$
return $(\tau, \omega_{\mathcal{U}}^{\mathrm{pp}})$

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

Fig. 32. Debt Accumulation Core Protocol

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

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

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 $\mathrm{O}_{\text{pricing}}(\cdot,\cdot,\cdot)$, which helps it to determine the price the user has to pay.

In the protocol the RSU and the user utilize a Blum coin toss to jointly compute a fresh and random serial number s. The coin toss is analogous to WalletIssuing and therefore omitted in this 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 $t := \mathsf{sk}_{\mathcal{U}}^{\text{id}} \cdot u_2 + u_1 \bmod p$. He then calculates the fraud detection ID for the current transaction as $\varphi := \mathsf{PRF}(\lambda, x)$. The user then proceeds by preparing the updated wallet. Therefore he first chooses a fresh double-spending mask u_1^{next} and executes $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}_{\text{com}}^1, (\lambda, b^{\text{prev}}, u_1^{\text{next}}, x))$ to commit to his wallet ID, the current balance, the fresh double-spending mask and the current counter. He also executes $(c_{\text{hid}}, d_{\text{hid}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}_{\text{com}}^1, \mathsf{sk}_{\mathcal{U}}^{\text{id}})$ to create a fresh commitment on his secret identification key. This commitment can be used at a later point in the Prove Participation task (cf. Figs. 35 and 36) 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 $L_{\text{gp}}^{(2)}$ defined by

$$L_{\text{gp}}^{(2)} := \left\{ \begin{pmatrix} \mathsf{pk}_{\mathcal{T}}^{\mathcal{T}} \\ \mathsf{pk}_{\mathcal{T}}^{\text{cert}} \\ \varphi \\ \mathbf{a}_{\mathcal{U}} \\ \mathbf{a}_{\mathcal{R}}^{\text{prev}} \\ c_{\text{hid}} \\ c'_{\mathcal{R}} \\ t \\ u_2 \end{pmatrix}^{\top} \left| \begin{array}{l} \exists\, x, \lambda, \mathsf{sk}_{\mathcal{U}}^{\text{id}}, u_1 \in \mathbb{Z}_p;\; s^{\text{prev}}, \varphi^{\text{prev}}, X, \Lambda, \mathsf{pk}_{\mathcal{U}}^{\text{id}}, \\ B^{\text{prev}}, U_1, U_1^{\text{next}}, d_{\text{hid}}, d_{\mathcal{R}}^{\text{prev}}, d'_{\mathcal{R}}, d_{\mathcal{T}} \in G_1; \\ \mathsf{pk}_{\mathcal{R}}^{\text{prev}} \in G_1^3;\; c_{\mathcal{R}}^{\text{prev}}, c_{\mathcal{T}} \in G_2;\; \sigma_{\mathcal{R}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{cert prev}}, \\ \sigma_{\mathcal{T}} \in G_2^2 \times G_1 : \\ \\ \mathsf{C1.Open}(\mathsf{CRS}_{\text{com}}^1, (\Lambda, \mathsf{pk}_{\mathcal{U}}^{\text{id}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\text{com}}^1, \mathsf{pk}_{\mathcal{U}}^{\text{id}}, c_{\text{hid}}, d_{\text{hid}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\text{com}}^1, (\Lambda, B^{\text{prev}}, U_1, X), c_{\mathcal{R}}^{\text{prev}}, d_{\mathcal{R}}^{\text{prev}}) = 1 \\ \mathsf{C1.Open}(\mathsf{CRS}_{\text{com}}^1, (\Lambda, B^{\text{prev}}, U_1^{\text{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}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{prev}}, (c_{\mathcal{R}}^{\text{prev}}, s^{\text{prev}})) = 1 \\ \mathsf{S.Vfy}(\mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \sigma_{\mathcal{R}}^{\text{cert prev}}, (\mathsf{pk}_{\mathcal{R}}^{\text{prev}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}})) = 1 \\ \varphi^{\text{prev}} = \mathsf{PRF}(\lambda, x-1),\;\; \varphi = \mathsf{PRF}(\lambda, x), \\ t = \mathsf{sk}_{\mathcal{U}}^{\text{id}} u_2 + u_1 \\ \mathsf{pk}_{\mathcal{U}}^{\text{id}} = g_1^{\mathsf{sk}_{\mathcal{U}}^{\text{id}}},\;\; U_1 = g_1^{u_1},\;\; X = g_1^{x},\;\; \Lambda = g_1^{\lambda} \end{array} \right. \right\} \tag{3}$$

This proof essentially shows that the wallet τ is valid, i.e., that the commitments $c_{\mathcal{T}}$ and $c_{\mathcal{R}}^{\text{prev}}$ are valid, bound to the user and have valid signatures, that the certificate $\mathsf{cert}_{\mathcal{R}}^{\text{prev}}$ 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 identification key, that $c_{\mathcal{R}}^{\text{prev}}$ 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 $(\pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ to the RSU, who first checks if the proof π verifies and if 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 time of the day, the current traffic volume and the attributes of the current and previous RSU. 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 $(c''_{\mathcal{R}}, d''_{\mathcal{R}}) := \mathsf{C1.Com}(\mathsf{CRS}_{\text{com}}^1, (0, p, 0, 1))$. 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

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 $\overline{\mathcal{G}}_{\text{bb}}$ 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. 34)

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

(4) At the TSP side:
- Receive $pid_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for public key $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$.
- Record ω^{bl} and ω^{dsp} internally.
- Parse φ and $-b^{\text{bill}}$ from ω^{bl}.

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

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

Fig. 33. The UC-protocol π_{P4TC} (cont. from Fig. 17)

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 $\omega^{\text{bl}} := (\varphi, p)$ can be used at a later point in the User Blacklisting task (cf. Fig. 39) to calculate the total fee of a fraudulent user. The double-spending transaction information $\omega^{\text{dsp}} := (\varphi, t, u_2)$ enables the TSP to identify the user if he uses the old version of the wallet (with unchanged balance) in another transaction (cf. Fig. 37). The RSU prove participation transaction information $\omega^{\text{pp}}_{\mathcal{R}} := (s, c_{\text{hid}})$ can be used later in the Prove Participation task (cf. Figs. 35 and 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

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

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

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. 33) and a core protocol DebtClearance (see Fig. 34). 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

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

$\mathcal{U}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}, \tau^{\text{prev}})$		$\mathcal{T}(\mathsf{pk}_{\mathcal{T}})$
parse $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$		parse $(\mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{R}}) := \mathsf{pk}_{\mathcal{T}}$
parse $\left((\mathsf{pk}_{\mathcal{U}}^{\text{id}}, \mathsf{pk}_{\mathcal{U}}^{\text{auth}}), (\mathsf{sk}_{\mathcal{U}}^{\text{id}}, \mathsf{sk}_{\mathcal{U}}^{\text{auth}})\right) := (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$		
parse $(s^{\text{prev}}, \varphi^{\text{prev}}, x, \lambda, \mathbf{a}_{\mathcal{U}}, c_{\mathcal{R}}^{\text{prev}}, d_{\mathcal{R}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{prev}}, \mathsf{cert}_{\mathcal{R}}{}^{\text{prev}},$ $c_{\mathcal{T}}, d_{\mathcal{T}}, \sigma_{\mathcal{T}}, b^{\text{prev}}, u_1) := \tau^{\text{prev}}$		
parse $(\mathsf{pk}_{\mathcal{R}}^{\text{prev}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{certprev}}) := \mathsf{cert}_{\mathcal{R}}{}^{\text{prev}}$		
$\varphi := \mathsf{PRF}(\lambda, x)$		
		$u_2 \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$
	$\xleftarrow{\quad u_2 \quad}$	
$t := \mathsf{sk}_{\mathcal{U}}^{\text{id}} u_2 + u_1 \mod p$		
stmnt $:= (\mathsf{pk}_{\mathcal{U}}^{\text{id}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, g_1^{b^{\text{prev}}}, t, u_2)$		
wit $:= (x, \lambda, \mathsf{sk}_{\mathcal{U}}^{\text{id}}, u_1, s^{\text{prev}}, \varphi^{\text{prev}}, g_1^x, g_1^\lambda, g_1^{u_1}, d_{\mathcal{R}}^{\text{prev}}, d_{\mathcal{T}},$ $\mathsf{pk}_{\mathcal{R}}^{\text{prev}}, c_{\mathcal{R}}^{\text{prev}}, c_{\mathcal{T}}, \sigma_{\mathcal{R}}^{\text{prev}}, \sigma_{\mathcal{R}}^{\text{certprev}}, \sigma_{\mathcal{T}})$		
$\pi \leftarrow \mathsf{P3.Prove}(\mathsf{CRS}_{\mathsf{pok}}, \textit{stmnt}, \textit{wit})$		
	$\xrightarrow{\mathsf{pk}_{\mathcal{U}}, \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, b^{\text{prev}}, t}$	
		parse $(\mathsf{pk}_{\mathcal{U}}^{\text{id}}, \mathsf{pk}_{\mathcal{U}}^{\text{auth}}) := \mathsf{pk}_{\mathcal{U}}$
		stmnt $:= (\mathsf{pk}_{\mathcal{U}}^{\text{id}}, \mathsf{pk}_{\mathcal{T}}^{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}^{\text{cert}}, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, g_1^{b^{\text{prev}}}, t, u_2)$
		if $\mathsf{P3.Vfy}(\mathsf{CRS}_{\mathsf{pok}}, \textit{stmnt}, \pi) = 0$
		return $\bot$
	$\xleftarrow{\quad \mathsf{OK} \quad}$	
$b^{\text{bill}} := b^{\text{prev}}$		$b^{\text{bill}} := b^{\text{prev}}$
		$\omega^{\text{bl}} := (\varphi, -b^{\text{bill}})$
		$\omega^{\text{dsp}} := (\varphi, t, u_2)$
return (b^{bill})		return $(\mathsf{pk}_{\mathcal{U}}^{\text{id}}, \omega^{\text{bl}}, \omega^{\text{dsp}})$

Fig. 34. Debt Clearance Core Protocol

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

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

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 $t := \mathsf{sk}_{\mathcal{U}}^{\text{id}} u_2 + u_1 \bmod p$ 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 *stmnt* from the language $L_{\text{gp}}^{(3)}$ defined by

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

The proof is a simplified version of the one in the DebtAccumulation protocol. The balance and the public user identification 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 $(\mathsf{pk}_{\mathcal{U}}, \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\text{prev}}, b^{\text{prev}}, t)$ 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 identification key $\mathsf{pk}_{\mathcal{U}}^{\text{id}}$, the blacklisting transaction information $\omega^{\text{bl}} := (\varphi, -b^{\text{prev}})$ and the double-spending transaction information $\omega^{\text{dsp}} := (\varphi, t, u_2)$. The user just outputs his final debt b^{prev} for this billing period.

Note that the wallet itself is discarded. It is expected that the user and the TSP execute the Wallet Issuing task next, to give the user a fresh wallet. After the 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.[24] 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.

The task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 35) and a core protocol ProveParticipation (see Fig. 36). In the wrapper protocol, the SA interacts with other UC entities and processes the set $S_{\mathcal{R}}^{\text{pp}}$ of all serial numbers that were recorded by the RSU that took the photo at roughly the time the photo was taken. In particular, the SA loads the internally recorded set $\Omega_{\mathcal{R}}^{\text{pp}}$ of all RSU prove participation transaction information that contain serial numbers that are also in $S_{\mathcal{R}}^{\text{pp}}$. Afterwards, the wrapper protocol invokes the core protocol

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

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

[24] 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: $(\texttt{prove\_participation}, pid_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}})$

(1) At the SA side:
- Receive $\mathsf{pk}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.$^{\perp}$
- 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. 36)

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

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

SA output: (out_{SA})

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

Fig. 35. The UC-protocol π_{P4TC} (cont. from Fig. 17)

$\mathcal{U}(\Omega^{\text{pp}}_{\mathcal{U}})$		$SA(\mathsf{pk}_{\mathcal{U}}, S^{\text{pp}}_{\mathcal{R}}, \Omega^{\text{pp}}_{\mathcal{R}})$
		parse $(\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}^{\text{auth}}_{\mathcal{U}}) := \mathsf{pk}_{\mathcal{U}}$
	$\xleftarrow{\quad S^{\text{pp}}_{\mathcal{R}} \quad}$	
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 $\mathsf{out}_{\mathcal{U}} := \mathsf{NOK}$		
	$\xrightarrow{\quad s, c_{\text{hid}}, d_{\text{hid}} \quad}$	
		$\mathsf{out}_{SA} := \mathsf{OK}$
		if $\omega^{\text{pp}}_{\mathcal{R}} = (s, c_{\text{hid}}) \notin \Omega^{\text{pp}}_{\mathcal{R}}$
		$\quad \mathsf{out}_{SA} := \mathsf{NOK}$
		if $\mathsf{C1.Open}(\mathsf{CRS}^{1}_{\text{com}}, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, c_{\text{hid}}, d_{\text{hid}}) = 0$
		$\quad \mathsf{out}_{SA} := \mathsf{NOK}$
return $(\mathsf{out}_{\mathcal{U}})$		return (out_{SA})

Fig. 36. Prove Participation Core Protocol

The protocol itself is simple. The SA first sends $S^{\text{pp}}_{\mathcal{R}}$ to the user who searches $\Omega^{\text{pp}}_{\mathcal{U}}$ for a user prove participation transaction information $\omega^{\text{pp}}_{\mathcal{U}}$ for which the serial number s in $\omega^{\text{pp}}_{\mathcal{U}}$ is also in $S^{\text{pp}}_{\mathcal{R}}$. If one is found the user's output bit $\mathsf{out}_{\mathcal{U}}$ is set to OK, if none is found $\mathsf{out}_{\mathcal{U}}$ is set to NOK ("not ok"). The user then sends $\omega^{\text{pp}}_{\mathcal{U}} := (s, c_{\text{hid}}, d_{\text{hid}})$ to

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

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

(1) Load the internally recorded set Ω^{dsp} of all double spending transaction information.

(2) Pick transaction information $\omega^{\text{dsp}} = (\varphi, t, u_2)$ and $\omega^{\text{dsp}\prime} = (\varphi', t', u_2')$ from the database Ω^{dsp}, such that $\varphi = \varphi'$ and $u_2 \neq u_2'$.$^\perp$

(3) $\text{sk}_{\mathcal{U}}^{\text{id}} := (t - t') \cdot (u_2 - u_2')^{-1} \bmod p$.

(4) $\text{pk}_{\mathcal{U}}^{\text{id}} := g_1^{\text{sk}_{\mathcal{U}}^{\text{id}}}$.

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

(6) $\pi := \text{sk}_{\mathcal{U}}^{\text{id}}$.

TSP output: $(pid_{\mathcal{U}}, \pi)$

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

Fig. 37. The UC-protocol π_{P4TC} (cont. from Fig. 17)

the SA and the SA checks if $(s, c_{\text{hid}}) \in \Omega_{\mathcal{R}}^{\text{pp}}$ and if d_{hid} is the opening of c_{hid} under $\text{pk}_{\mathcal{U}}^{\text{id}}$. If both checks succeed the SA's output bit $\texttt{out}_{SA}$ is set to $\texttt{OK}$, if at least one check fails $\texttt{out}_{SA}$ is set to $\texttt{NOK}$. At the end of the protocol both parties output their bits $\texttt{out}_{\mathcal{U}}$ and $\texttt{out}_{SA}$, respectively[25]

If the SA's output equals 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. 37) can be used by the TSP to extract the public identification key of the user these double-spending transactions information belong to as well as a proof (such as his secret identification 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 $\omega^{\text{dsp}} = (\varphi, t, u_2)$ and $\omega^{\text{dsp}\prime} = (\varphi', t', u_2')$ 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 identification key can be recovered as $\text{sk}_{\mathcal{U}}^{\text{id}} := (t - t') \cdot (u_2 - u_2)^{-1} \bmod p$. His public identification key is then $\text{pk}_{\mathcal{U}}^{\text{id}} := g_1^{\text{sk}_{\mathcal{U}}^{\text{id}}}$. The secret identification key $\text{sk}_{\mathcal{U}}^{\text{id}}$ can be used as a proof of guilt in the Guilt Verification task (cf. Fig. 38).

Every user that is convicted of double-spending is added to the blacklist via the User Blacklisting task (cf. Figs. 39 and 40) and additional measures are taken (these are out of scope).

D.11 Guilt Verification

If a user is indeed guilty of double-spending can be verified using the Guilt Verification task depicted in Fig. 38. This algorithm may be run by anyone, in particular by justice. Essentially, the algorithm checks if a given public user identification key $\text{pk}_{\mathcal{U}}^{\text{id}}$ and a proof of guilt π match. This is easily accomplished because they match if and

[25]Note that after a successful (both parties output 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: $(\texttt{verify\_guilt}, pid_{\mathcal{U}}, \pi)$

(1) Receive $\text{pk}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$.$^{\perp}$
(2) Parse $\text{pk}_{\mathcal{U}}^{\text{id}}$ from $\text{pk}_{\mathcal{U}}$.
(3) If $g_1^{\pi} = \text{pk}_{\mathcal{U}}^{\text{id}}$, 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. 38. The UC-protocol π_{P4TC} (cont. from Fig. 17)

only if $g_1^{\pi} = \text{pk}_{\mathcal{U}}^{\text{id}}$ holds. This equation holds if and only if π equals the user's secret identification key $\text{sk}_{\mathcal{U}}^{\text{id}}$ that was recovered using the Double-Spending Detection task (cf. Fig. 37).

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.

Users that are on the blacklist are unable to get issued a new wallet and they also get photographed at every RSU they pass. They can also be punished by some other means (which are out of scope).

The User Blacklisting task is presented in two parts: a wrapper protocol π_{P4TC} (see Fig. 39) and a core protocol UserBlacklisting (see Fig. 40). The wrapper protocol interacts with other UC entities, pre-processes the input and afterwards internally invokes the core protocol

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

The DR gets its public and private key $(\text{pk}_{DR}, \text{sk}_{DR})$ as input and the TSP gets the public key $\text{pk}_{\mathcal{U}}$ of the user to be blacklisted along with a set $HTD_{\mathcal{U}}$ containing all his hidden user trapdoors from the current billing period as input.[26]

At the beginning of the protocol, the TSP sends its input $\text{pk}_{\mathcal{U}}, HTD$ to the DR. The DR then recovers for every $htd := (\lambda'', e_0, \ldots, e_{\ell}, e^*, \sigma^*) \in HTD$ the corresponding wallet id λ. For that purpose, the DR first checks if the signature σ^* of $(e_0, \ldots, e_{\ell}, e^*)$ under the user's authentication key verifies. This ensures that the DR can not be tricked by the TSP into recovering the wallet ID for a different user.

The DR proceeds by decrypting every e_i to get $(g_1^{\lambda'_0}, \ldots, g_1^{\lambda'_{\ell}})$. Since each λ'_i is small $(\lambda'_i < \mathcal{B})$, the DR can compute the discrete logarithms in a reasonable amount of time to recover $(\lambda'_0, \ldots, \lambda'_{\ell})$. 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. The DR then calculates the user-chosen part of the wallet ID as

[26]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: (`blacklist_user`)
TSP input: (`blacklist_user`, $pid_{\mathcal{U}}$)

(1) At the DR side:
   - Load the internally recorded $(\mathsf{pk}_{DR}, \mathsf{sk}_{DR})^{\perp}$.

(2) At the TSP side:
   - Receive $\mathsf{pk}_{\mathcal{U}}$ from the bulletin-board $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}{}^{\perp}$.
   - Load internally recorded set $AHTD$ of all augmented hidden trapdoors and set $HTD := \{htd \mid (pid_{\mathcal{U}}, \cdot, htd) \in AHTD\}$

(3) Both sides: Run the code of UserBlacklisting between the DR and the TSP (see Fig. 40)

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

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

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

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

Fig. 39. The UC-protocol π_{P4TC} (cont. from Fig. 17)

$\lambda' := \sum_{i=0}^{\ell} \lambda'_i \cdot \mathcal{B}^i$ and checks if e^* is indeed an encryption of $g_1^{\lambda'}$. If the check succeeds, the wallet ID can be recovered as $\lambda := \lambda' + \lambda''$ (remember that λ'' is directly stored in htd).

The set $\Phi_\lambda := \{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\text{bl}_{\mathcal{R}}})\}$ of fraud detection IDs is then sent to the TSP (for every $htd \in HTD$). The blacklist parameter $x_{\text{bl}_{\mathcal{R}}}$ is chosen in such a way that a user is expected to perform at most $x_{\text{bl}_{\mathcal{R}}}$ executions of the Debt Accumulation task in a single billing period. The DR's output of the core protocol is simply `OK`, while the TSP's output is $\Phi_{\mathcal{U}} := \bigcup_{HTD} \Phi_\lambda$.

After the termination of the core protocol, the TSP calculates the total toll amount the user owes for the billing period in question in the wrapper protocol. Therefore, the TSP uses the internally recorded set $\Omega^{\text{bl}}_{\mathcal{U}}$ of all blacklisting transaction information for this user (the TSP can calculate this set with its knowledge of $\Phi_{\mathcal{U}}$). By summing up all the prices inside the blacklisting transaction information, the TSP can calculate the total fee b^{bill} the user owes.

After the termination of the wrapper protocol, the fraud detection IDs in $\Phi_{\mathcal{U}}$ are added to the RSU blacklist $bl_{\mathcal{R}}$ and the user's public identification key $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ is added on the TSP blacklist $bl_{\mathcal{T}}$. The TSP blacklist $bl_{\mathcal{T}}$ is used by the TSP in the Wallet Issuing task to ensure that no user that did not pay his invoice receives 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. 41. 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 $\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}$ of $c_{\mathcal{T}}$ and

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

                                   $\xleftarrow{\mathsf{pk}_{\mathcal{U}}, HTD}$

for all $htd \in HTD$
    parse $(\lambda'', c''_{\mathcal{T}}, d''_{\mathcal{T}} e_0, \ldots, e_\ell, e^*, \sigma^*) := htd$
    parse $(\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \mathsf{pk}^{\mathrm{auth}}_{\mathcal{U}}) := \mathsf{pk}_{\mathcal{U}}$
    if $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{auth}}_{\mathcal{U}}, \sigma^*, (c''_{\mathcal{T}}, e_0, \ldots, e_\ell, e^*)) = 0$
        return $\perp$
    if $\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathrm{com}}, (g_1^{\lambda''}, 0), c''_{\mathcal{T}}, d''_{\mathcal{T}}) = 0$
        return $\perp$
    $\lambda' := 0$
    for $i$ from 0 to $\ell$ :
        $\lambda'_i := \mathsf{DLOG}(\mathsf{E.Dec}(\mathsf{sk}_{DR}, e_i))$
        $\lambda' := \lambda' + \lambda'_i \cdot \mathcal{B}^i$
    if $g_1^{\lambda'} \neq \mathsf{E.Dec}(\mathsf{sk}_{DR}, e^*)$
        return $\perp$
    $\lambda := \lambda' + \lambda''$
    $\Phi_\lambda := \{\mathsf{PRF}(\lambda, 0), \ldots, \mathsf{PRF}(\lambda, x_{\mathrm{bl}_{\mathcal{R}}})\}$
$\Phi_{\mathcal{U}} := \bigcup_{HTD} \Phi_\lambda$

                                   $\xrightarrow{\Phi_{\mathcal{U}}}$

return $(\mathsf{OK})$                                                 return $(\Phi_{\mathcal{U}})$
```

Fig. 40. User Blacklisting Core Protocol

$\mathbf{a}_{\mathcal{U}}$, that $\sigma_{\mathcal{R}}$ is a valid signature under $\mathsf{sk}_{\mathcal{R}}$ of $c_{\mathcal{R}}$ and s, that the certificate $\mathsf{cert}_{\mathcal{R}}$ containing $\mathsf{pk}_{\mathcal{R}}$ 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 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 E.1 (Security Statement). *Assume that the SXDH-problem is hard for* $\mathsf{gp} := (G_1, G_2, G_T, e, p, g_1, g_2)$*, 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 A.2. Then*

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

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

Fig. 41. Algorithm for Wallet Verification

if either only a subset of the users or only a subset of the RSUs, the TSP and the SA is statically corrupted.

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 B.1.

We prove Theorem E.1 in three steps:

- In Appendix E.1 we first show in Theorem E.21 that $\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}, \overline{\mathcal{G}}_{\mathrm{bb}}}$ correctly implements $\mathcal{F}^{\overline{\mathcal{G}}_{\mathrm{bb}}}$, if all parties follow the protocol honestly.
- In the next step, Theorem E.22 proves $\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}, \overline{\mathcal{G}}_{\mathrm{bb}}} \geq_{\mathrm{UC}} \mathcal{F}^{\overline{\mathcal{G}}_{\mathrm{bb}}}$, if only a subset of the users is statically corrupted. We call this case “System Security”. Theorem E.22 is presented in Appendix E.2.
- In the last step, Theorem E.36 proves $\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}, \overline{\mathcal{G}}_{\mathrm{bb}}} \geq_{\mathrm{UC}} \mathcal{F}^{\overline{\mathcal{G}}_{\mathrm{bb}}}$, if only a subset of the RSUs is statically corrupted. We call this case “User Security and Privacy” and present it in Appendix E.3.

Proof of Theorem E.1. The theorem is immediately implied by Theorems E.21, E.22 and E.36. □

Please note that showing correctness separately is technically not necessary. Both Theorems E.22 and E.36 individually imply correctness, as the case in which all parties are honest is a special case of a scenario in which a subset of parties is corrupted. Showing correctness separately serves two purposes: For didactic reasons a restricted setting (i.e., all parties are assumed to be honest) is convenient to introduce some notation and to present the underlying idea that is common to all following proofs. Secondly, having shown correctness as a special case before helps during the course of the proof of system security to keep matters simple, because we can revert to the special case.

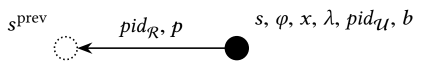

Fig. 42. An entry *trdb* ∈ *TRDB* visualized as an element of a directed graph

E.1 Proof of Correctness

In many papers where something is shown to be UC-secure the considered protocols are rather simple (e.g., a commitment, an oblivious transfer, a coin toss) and correctness is mostly obvious. Our protocol in contrast is a complex system with polynomially many parties that can reactively interact forever, i.e., the protocol itself has no inherent exit point except that at some point the polynomially bounded runtime of the environment is exhausted. In order to prove the security of our system later (see Appendices E.2 and E.3) we make use of a theoretical construct we call the *Augmented Transaction Graph*. This graph helps us to link the interactions of the parties across the various tasks our protocol provides. However, the Augmented Transaction Graph is also useful to prove correctness and hence we use this proof to introduce the concept.

In this section no security reduction occurs, because we assume all parties to be honest and solely show that the subroutine input/output of the main parties is indistinguishable between the ideal and the real model.

We start with a series of simple lemmas about the ideal functionality $\mathcal{F}_{\text{P4TC}}$ which also help to develop a good conception about how the individual tasks/transactions are connected.

Internally, $\mathcal{F}_{\text{P4TC}}$ stores a pervasive database *TRDB* whose entries *trdb* are of the form

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

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 $(s, \varphi, x, \lambda, pid_{\mathcal{U}}, b)$ and identified by s. The edge to the predecessor is identified by (s^{prev}, s) and labeled with $(pid_{\mathcal{R}}, p)$. See Fig. 42 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 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 E.2 (Ideal Transaction Graph (informal)). The transaction database $TRDB = \{trdb_i\}$ with $trdb = (s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ is a directed, labeled graph as defined above. This graph is called the *Ideal Transaction Graph.*

LEMMA E.3. *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 $\perp$), each vertex has in-degree at most one. □

Lemma E.4. *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 E.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 E.4. □

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

Lemma E.6. *Within a tree of TRDB, every node $trdb = (s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ 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 E.7. *Let $trdb = (s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b)$ be an arbitrary but fixed node. If trdb is not a root let $trdb^{\text{prev}} = (s^{\text{prev,prev}}, s^{\text{prev}}, \varphi^{\text{prev}}, x^{\text{prev}}, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}^{\text{prev}}, p^{\text{prev}}, b^{\text{prev}})$ be its predecessor. Then $b = b^{\text{prev}} + p$ holds for non-root nodes and $p = \bot$, $b = 0$ for root nodes.*

Proof. Same induction argument as in proof of Lemma E.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 E.8 (Protection Against False Accusation). *(1) The task Double-Spending Detection returns a proof $\pi \neq \bot$ 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 2 and 3 in Fig. 15). Guilt Verification only returns `OK`, if $f_{\Pi}(pid_{\mathcal{U}}, \pi) =$ `OK` has already been defined (cp. Step 1). As (in case of honest users) Guilt Verification extends f_{Π} by nothing but invalid proofs, Double-Spending Detection exclusively sets $f_{\Pi}(pid_{\mathcal{U}}, \pi) =$ `OK` (cp. Step 2, in Fig. 14). □

Lemma E.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 $x_{\mathrm{bl}_{\mathcal{R}}}$ transactions, i.e., in less then $x_{\mathrm{bl}_{\mathcal{R}}}$ invocations of Wallet Issuing, Debt Accumulation and Debt Clearance, the following two statements hold:*

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

□

After these preliminaries that only dealt with the ideal functionality on its own, we are now ready to argue why π_{P4TC} implements $\mathcal{F}_{\mathrm{P4TC}}$ correctly. Please note that to this extent all parties are assumed to be honest. The simulator $\mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}$ is depicted in Fig. 43. As before in Appendix B we assume encrypted and one-sidedly authenticated channels. Hence, the simulator is not required to simulate any messages, but only needs to provide the ideal function with implementation-specific values if being asked to do so. This also implies that the only option for the environment $\mathcal{Z}$ to distinguish between the real and the ideal model is by means of different outputs from the different tasks. Luckily, either these tasks run independent from each other (e.g., registration) or their output is information-theoretically independent from any shared state (e.g., various `OK`-messages). Hence, only a handful of options remain and those can be handled individually.

Lemma E.10 (Registration Correctness). *Let $\mathcal{Z}^{\mathrm{correct}}$ be an environment that does not corrupt any parties. Then the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\mathrm{P4TC}}$ resp. in the scope of the tasks DR/TSP/RSU/User Registration is perfectly indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\mathrm{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\mathrm{correct}}}(1^n).$$

Proof. Obvious. $\mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}$ runs the real algorithms DRRegistration, TSPRegistration, RSURegistration, and UserRegistration in the ideal experiment. □

Lemma E.11 (RSU Certification Correctness). *Let $\mathcal{Z}^{\mathrm{correct}}$ be an environment that does not corrupt any parties. Under the assumption that* S *is correct, the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\mathrm{P4TC}}$ resp. in the scope of the task RSU Certification is perfectly indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\mathrm{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\mathrm{correct}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\mathrm{correct}}}(1^n).$$

Proof. Obvious. In the ideal model the TSP inputs $\mathbf{a}_{\mathcal{R}}$ and the RSU outputs $\mathbf{a}_{\mathcal{R}}$ again. In the real model, the TSP sends $\mathbf{a}_{\mathcal{R}}$ to the RSU together with a signature. As S is correct and all parties honest, the RSU does not abort. □

Simulator $\mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}$

Setup:

(1) Run the algorithm $\text{CRS} \leftarrow \text{Setup}(1^n)$.

(2) Record CRS.

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

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

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

User Registration: Upon receiving $(\texttt{registering\_user}, pid_{\mathcal{U}})$ run $(\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}}) \leftarrow \text{UserRegistration}(\text{CRS})$, return $\text{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $pid_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$.

RSU Certification: (nothing to do)

Wallet Issuing: (nothing to do)

Debt Accumulation: Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide $\omega^{\text{pp}}_{\mathcal{U}}$ for s, look up $pid_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$, run $(c_{\text{hid}}, d_{\text{hid}}) \leftarrow \text{C1.Com}(\text{CRS}^1_{\text{com}}, \text{sk}^{\text{id}}_{\mathcal{U}})$, and return $\omega^{\text{pp}}_{\mathcal{U}} := (s, c_{\text{hid}}, d_{\text{hid}})$.

Debt Clearance: (nothing to do)

Prove Participation: (nothing to do)

Double-Spending Detection: Upon being ask to provide a proof for $pid_{\mathcal{U}}$, look up $pid_{\mathcal{U}} \mapsto (\text{pk}_{\mathcal{U}}, \text{sk}_{\mathcal{U}})$, parse $\text{sk}^{\text{id}}_{\mathcal{U}}$ from $\text{sk}_{\mathcal{U}}$ and return $\text{sk}^{\text{id}}_{\mathcal{U}}$.

Guilt Verification: (nothing to do)

User Blacklisting: (nothing to do)

Fig. 43. The simulator for System Correctness

Lemma E.12 (Wallet Issuing Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties and does not conduct blacklisting.*[27] *Under the assumption that all building blocks are correct and* PRF *is a pseudo-random function, the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Wallet Issuing is perfectly indistinguishable between the experiments*

$$\text{EXEC}_{\pi^{\mathcal{F}_{\text{CRS}}}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \quad \text{and}$$
$$\text{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. In the ideal model the TSP inputs $\mathbf{a}_{\mathcal{U}}$ which is output by the user. In the real model the TSP sends $\mathbf{a}_{\mathcal{U}}$ together with a signature. As S is correct and all parties honest, the user does not abort. In the ideal model the user outputs s which is uniformly drawn. In the real model s is drawn by means of a Blum coin toss. □

Until now, all previous lemmas have been rather trivial which is likely one of the reasons why correctness is usually not considered separately in UC proofs. Generally, it is quite obvious that a real protocol does what it is expected to do. The following lemma is slightly more involved and the main reason why we decided to show correctness first.

[27]The case that Wallet Issuing aborts due to blacklisting is postponed to Lemma E.20.

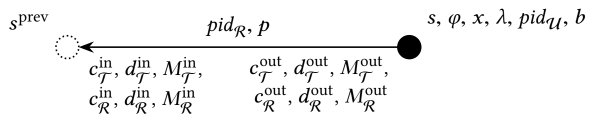
s^{prev}
$pid_{\mathcal{R}}, p$
$s, \varphi, x, \lambda, pid_{\mathcal{U}}, b$
$c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}},$
$c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}}$
$c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}},$
$c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}}$

Fig. 44. An entry $\overline{trdb} \in \overline{TRDB}$ visualized as an element of a directed graph

We start with more notation and augment the entries $trdb$ of the Ideal Transaction Graph $TRDB$ with additional information. An *Augmented Transaction Entry* $\overline{trdb}$ has the form

$$\begin{aligned}\overline{trdb} = (&s^{\text{prev}}, s, \varphi, x, \lambda, pid_{\mathcal{U}}, pid_{\mathcal{R}}, p, b,\\ &c^{\text{in}}_{\mathcal{T}}, d^{\text{in}}_{\mathcal{T}}, M^{\text{in}}_{\mathcal{T}}, c^{\text{in}}_{\mathcal{R}}, d^{\text{in}}_{\mathcal{R}}, M^{\text{in}}_{\mathcal{R}},\\ &c^{\text{out}}_{\mathcal{T}}, d^{\text{out}}_{\mathcal{T}}, M^{\text{out}}_{\mathcal{T}}, c^{\text{out}}_{\mathcal{R}}, d^{\text{out}}_{\mathcal{R}}, M^{\text{out}}_{\mathcal{R}})\end{aligned} \tag{5}$$

with c, d and M with equal suffixes denoting a commitment, its decommitment information and the opening in the implicit message space. 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 E.13 (Augmented Transaction Graph (informal)). The set $\overline{TRDB} = \{\overline{trdb}_i\}$ with $\overline{trdb}_i$ defined as in Eq. (5) is called the *Augmented 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 not 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. While this is rather easy to show in the scope of the proof of correctness if all parties are honest, these equalities are not obviously true as soon as corrupted parties are considered.

The augmented transaction information is depicted in Fig. 44.

The next proposition will be used several times below and maps an execution of the real protocol one-to-one and onto an execution of the ideal functionality. In order to be formally correct we do not only claim that an Augmented Transaction Graph exists mathematically, but also show how it can be obtained computationally. To this end we describe a UC-like experiment $\text{EXEC}_{\mathcal{F}_{\text{P4TC}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^*, \mathcal{Z}^*}(1^n)$. We consider a special environment $\mathcal{Z}^*$ that corrupts all parties but plays them honestly, i.e., $\mathcal{Z}^*$ is an honest adversary. This is a technical requirement such that the simulator $\mathcal{S}^*$ is able to "see" the messages. $\mathcal{S}^*$ simulates a CRS such that the ZK-proofs become extractable and otherwise behaves like the dummy adversary. Please note that $\mathcal{S}^*$ does not need to do anything with respect to $\mathcal{Z}^*$ except message delivery, as $\mathcal{Z}^*$ plays with itself. In addition to the dummy adversary $\mathcal{S}^*$ performs some bookkeeping (not required for simulation). It keeps track of all interactions and maintains a shadow copy of the Transaction Graph perfectly in sync with the Ideal Transaction Graph that $\mathcal{F}_{\text{P4TC}}$ maintains internally. Moreover, $\mathcal{S}^*$ augments its own copy of the Transaction Graph with the commitments it sees. $\mathcal{S}^*$ can obtain the in-commitments via extraction of the ZK-proofs, and the out-commitments are sent by the TSP/RSU in the clear.

Proposition E.14. *Consider the experiment*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}},\overline{\mathcal{G}}_{\mathrm{bb}},\mathcal{S}^*,\mathcal{Z}^*}(1^n)$$

as defined earlier, and the Augmented Transaction Graph $\overline{TRDB}$ corresponding to this experiment.

(1) Let $\overline{trdb}^{\mathrm{prev}} = (s^{\mathrm{prev,prev}}, s^{\mathrm{prev}}, \ldots)$ and $\overline{trdb} = (s^{\mathrm{prev}}, s, \ldots)$ be two transaction entries that are connected by an edge. Then $c_{\mathcal{T}}^{\mathrm{out\,prev}} = c_{\mathcal{T}}^{\mathrm{in}}$ and $c_{\mathcal{R}}^{\mathrm{out\,prev}} = c_{\mathcal{R}}^{\mathrm{in}}$ hold.

(2) Let $\overline{trdb}^* = (s^{\mathrm{prev},*}, s^*, \ldots)$ and $\overline{trdb} = (s^{\mathrm{prev}}, s, \ldots)$ be two transaction entries with $c_{\mathcal{T}}^{\mathrm{out}*} = c_{\mathcal{T}}^{\mathrm{in}}$ and $c_{\mathcal{R}}^{\mathrm{out}*} = c_{\mathcal{R}}^{\mathrm{in}}$. Then $s^* = s^{\mathrm{prev}}$ holds with overwhelming probability.

In other words, the in-/out-commitments also induce a graph structure on the Augmented Transaction Graph and for an honest execution of the protocol this structure coincides with the graph structure inherited from the Ideal Transaction Graph by means of serial numbers s.

Proof. (1) Follows from the fact that the users are honest. At the end of some transaction with serial number s^{prev} the user obtains the out-commitments $c_{\mathcal{T}}^{\mathrm{out\,prev}}$, $c_{\mathcal{R}}^{\mathrm{out\,prev}}$ and records them locally. If the user invokes a new transaction with serial s and s^{prev} as its predecessor, he loads the recorded commitments $c_{\mathcal{T}}^{\mathrm{out\,prev}}$, $c_{\mathcal{R}}^{\mathrm{out\,prev}}$ and uses them as in-commitments $c_{\mathcal{T}}^{\mathrm{in}}$, $c_{\mathcal{R}}^{\mathrm{in}}$.

(2) Follows from the fact that the out-commitments are honestly generated and therefore unique with overwhelming probability. Assume the contrary, i.e., $c_{\mathcal{T}}^{\mathrm{out}*} = c_{\mathcal{T}}^{\mathrm{in}}$ and $c_{\mathcal{R}}^{\mathrm{out}*} = c_{\mathcal{R}}^{\mathrm{in}}$ hold but $s^* \neq s^{\mathrm{prev}}$. This can only happen due to one of two reasons:
   (a) At the beginning of a transaction the environment uses an in-commitment that is not the out-commitment of the preceding transaction. This contradicts the assumption that the user behaves honestly.
   (b) At the end of some transaction an out-commitment is accidentally generated that equals an in-commitment elsewhere. Again, as the out-commitments are honestly generated, this happens only with negligible probability.

□

With Proposition E.14 we can show the next lemma on our way to proving correctness.

Lemma E.15 (Debt Accumulation Correctness). *Let $\mathcal{Z}^{\mathrm{correct}}$ be an environment that does not corrupt any parties and does not conduct blacklisting.*[28] *Under the assumption that all building blocks are correct,* C1 *is homomorphic, and* PRF *is a pseudo-random function the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\mathrm{P4TC}}$ resp. in the scope of the task Debt Accumulation is computationally indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\mathrm{P4TC}}^{\mathcal{F}_{\mathrm{CRS}}},\overline{\mathcal{G}}_{\mathrm{bb}},\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\mathrm{correct}},\mathcal{Z}^{\mathrm{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}},\overline{\mathcal{G}}_{\mathrm{bb}},\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\mathrm{correct}},\mathcal{Z}^{\mathrm{correct}}}(1^n).$$

Proof. With respect to the output of s, $\mathbf{a}_{\mathcal{R}}$ and $\mathbf{a}_{\mathcal{U}}$ we refer to the proofs of Lemmas E.11 and E.12. Here, the same arguments hold. It remains to show that the output with respect to b, p is indistinguishable. Consider the path in the Augmented Transaction Graph $\overline{TRDB}$ from the root to this invocation of the Debt Accumulation task. Let $\overline{trdb}^{\mathrm{prev}}$ and $\overline{trdb}$ be two arbitrary but fixed transaction entries with $\overline{trdb}^{\mathrm{prev}}$ being the predecessor of $\overline{trdb}$. Due to Proposition E.14 $c_{\mathcal{T}}^{\mathrm{out\,prev}} = c_{\mathcal{T}}^{\mathrm{in}}$ and $c_{\mathcal{R}}^{\mathrm{out\,prev}} = c_{\mathcal{R}}^{\mathrm{in}}$ holds. Moreover, $c_{\mathcal{R}}^{\mathrm{in}}$ is updated to an out-commitment $c_{\mathcal{R}}^{\mathrm{out}}$ using the homomorphism of the commitment scheme C1 that is consistent with the modification of the semantic information in the ideal model, i.e., $b = b^{\mathrm{prev}} + p$. □

[28]The case that Debt Accumulation aborts due to blacklisting is postponed to Lemma E.20.

Lemma E.16 (Debt Clearance Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties. Under the assumption that all building blocks are correct,* C1 *is homomorphic, and* PRF *is a pseudo-random function the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Debt Clearance is computationally indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. Uses the same arguments as the proof of Lemma E.15. □

The remaining Lemmas E.17 to E.20 deal with correctness of the auxiliary tasks Prove Participation, Double-Spending Detection, Guilt Verification, and User Blacklisting. The proofs are actually neither complicated nor give any technical insight, because correctness simply follows from comparing the code of the real implementation with the pseudo-code of the ideal functionality under the assumption that the building blocks are correct. We merely include those proofs for the sake of completeness.

Lemma E.17 (Prove Participation Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties. Under the assumption that all building blocks are correct, the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Prove Participation is indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. The statement obviously follows by comparison of what the ideal functionality and the real implementation do. If Prove Participation returns OK in the ideal experiment, then there exists s s.t. $(\cdot, s, \cdot, x, \lambda, pid_{\mathcal{U}}, \cdot, \cdot, \cdot) \in$ *TRDB*. This means the respective user has successfully participated in the xth transaction for a wallet with ID λ in the scope of Debt Accumulation. In the real experiment the user sends an honestly generated commitment c_{hid} which is recorded for s by both parties. During Prove Participation this commitment is re-sent and successfully opened. The reverse direction is likewise obvious. □

Lemma E.18 (Double-Spending Detection Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties. Under the assumption that all building blocks are correct, the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Double-Spending Detection is indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. In the ideal model Double-Spending Detection returns the correct $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$ for $pid_{\mathcal{U}}$ in case of a fraud by definition of $\mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}$. In the real model Double-Spending Detection returns $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$ with overwhelming probability. As the honest user is committed to the same u_1, the RSU picks two different values u_2, u_2' which result into two different points (u_2, t), (u_2', t') with overwhelming probability. This means the linear equation system has a unique solution which equals $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$. □

Lemma E.19 (Guilt Verification Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties. Under the assumption that all building blocks are correct, and DLOG is a hard problem for G_1, the subroutine* *input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Guilt Verification is indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. In the ideal model Guilt Verification returns OK if and only if π has been an output of Double-Spending Detection (by Lemma E.8) and thus $g_1^\pi = \mathsf{pk}^{\text{id}}_{\mathcal{U}}$ holds (by definition of $\mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}$, cp. Fig. 43). In the real model Guilt Verification returns OK if and only if $g_1^\pi = \mathsf{pk}^{\text{id}}_{\mathcal{U}}$. Hence, the only case in which the ideal and the real model differ is if $g_1^\pi = \mathsf{pk}^{\text{id}}_{\mathcal{U}}$ holds but π is not an output of Double-Spending Detection. This event is negligible, as DLOG is assumed to be a hard problem for G_1. □

Lemma E.20 (User Blacklisting Correctness). *Let $\mathcal{Z}^{\text{correct}}$ be an environment that does not corrupt any parties. Assume that all building blocks are correct and* PRF *is a pseudo-random function.*

(1) *If no user uses his wallet more than $x_{\text{bl}_\mathcal{R}}$ times, the subroutine input/output of the main parties of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task User Blacklisting is indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

(2) *The abort behavior of π_{P4TC} and $\mathcal{F}_{\text{P4TC}}$ resp. in the scope of the task Debt Accumulation is computationally indistinguishable between the experiments*

$$\mathsf{EXEC}_{\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n) \qquad \text{and}$$
$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{correct}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{correct}}}(1^n).$$

Proof. (1) In the ideal model, the output of User Blacklisting depends on the transaction graph *TRDB*. For this reason, we proceed by a proof of induction for a fixed but arbitrary PID $pid_\mathcal{U}$. There are only three tasks, namely Wallet Issuing, Debt Accumulation and Debt Clearance, that modify *TRDB*. Assume that $\mathcal{Z}^{\text{correct}}$ has not triggered any of these tasks yet for $pid_\mathcal{U}$. Then, in the ideal model $TRDB_\mathcal{U}$ is empty and User Blacklisting returns $b^{\text{bill}} = 0$ and $\Phi_\mathcal{U} = \emptyset$. Likewise in the real model the set of hidden trapdoors *HTD* is empty and thus $b^{\text{bill}} = 0$ and $\Phi_\mathcal{U} = \emptyset$ is returned.
Assume that correctness holds for n transactions before User Blacklisting is called and we consider an execution with $n + 1$ previous transactions for the fixed $pid_\mathcal{U}$. We look at the three cases of Wallet Issuing, Debt Accumulation and Debt Clearance separately.

(a) *The $(n+1)^{\text{th}}$ transaction is Wallet Issuing:* In the ideal model a new entry $trdb^*$ with a unique λ^* and $p^* = 0$ is inserted into *TRDB*, i.e., b^{bill} does not change. In the real model b^{bill} does not change either, because Wallet Issuing does not insert a ω^{bl} into Ω^{bl}. In the ideal model the partial mapping f_Φ is extended onto the domain $\{(\lambda^*, 0), \ldots, (\lambda^*, x_{\text{bl}_\mathcal{R}})\}$ and mapped to uniformly drawn fraud detection IDs. The output $\Phi_\mathcal{U}$ increases by $x_{\text{bl}_\mathcal{R}} + 1$ elements as the probability to pick the same element twice is negligible. In the real model Wallet Issuing inserts a new element *htd* into *HTD* which is sent to *DR* and—by assumption that every party is honest—verifies correctly and is extracted to a unique λ^*.[29] Consequently, *DR* inserts $x_{\text{bl}_\mathcal{R}} + 1$ elements $\{\mathsf{PRF}(\lambda^*, 0), \ldots, \mathsf{PRF}(\lambda^*, x_{\text{bl}_\mathcal{R}})\}$ into $\Phi_\mathcal{U}$. Under the assumption that PRF is a pseudo-random function, these elements are indistinguishable from those drawn in the ideal experiment.

[29]N.b.: By abuse of notation we use the same variable here as in the ideal experiment, but of course they are not necessarily equal—only computationally indistinguishable over the randomness of the experiment.

(b) *The* $(n+1)^{\text{th}}$ *transaction is Debt Accumulation:* In the ideal model a new entry $trdb^*$ with fraud detection ID φ^* and price p^* is inserted into *TRDB*, i.e., b^{bill} increases by p^*. In the real model $\mathcal{R}$ outputs $\omega^{\text{bl}^*} = (\varphi^*, p^*)$ with $\varphi^* = \mathsf{PRF}(\lambda^*, x^*)$ at the end of Debt Accumulation. By assumption that the wallet has been generated honestly, a hidden trapdoor htd^* that can be extracted to λ^* is contained in *HTD*. By assumption that the user does not use his wallet more than $x_{\text{bl}_{\mathcal{R}}}$ times, $x^* \leq x_{\text{bl}_{\mathcal{R}}}$ holds. Hence, $\varphi^* \in \Phi_{\mathcal{U}}$ and $\omega^{\text{bl}} \in \Omega^{\text{bl}}_{\mathcal{U}}$ follows whereby b^{bill} increases by p^*.

(c) *The* $(n+1)^{\text{th}}$ *transaction is Debt Clearance:* Same argument as for Debt Accumulation.

(2) Let s^{prev} denote the serial number for which Debt Accumulation is invoked and let

$$\overline{trdb}^{\text{prev}} = (\cdot, s^{\text{prev}}, \varphi^{\text{prev}}, x^{\text{prev}}, \lambda, \ldots)$$

be the corresponding transaction entry. We distinguish two cases:

(a) *In the ideal model* $f_\Phi(\lambda, x^{\text{prev}} + 1)$ *has already been defined:* This can happen due to two cases; either there is a $\overline{trdb}^*$ with $\lambda^* = \lambda$ and $x^* = x^{\text{prev}} + 1$ or User Blacklisting has been invoked and $x^{\text{prev}} < x_{\text{bl}_{\mathcal{R}}}$ holds. In either case the fraud detection ID $\varphi = f_\Phi(\lambda, x^{\text{prev}} + 1)$ associated with the considered transaction has already been fixed and given as part of some output to $\mathcal{Z}^{\text{correct}}$. In the real model $\varphi = \mathsf{PRF}(\lambda, x^{\text{prev}} + 1)$ has likewise been output to $\mathcal{Z}^{\text{correct}}$. Debt Accumulation aborts with `blacklisted`, if and only if φ[30] is contained in $bl_{\mathcal{R}}$.

(b) *In the ideal model* $f_\Phi(\lambda, x^{\text{prev}} + 1)$ *has not yet been defined:* In the ideal model a fresh φ is uniformly drawn, the probability that $\varphi \in bl_{\mathcal{R}}$ holds is negligible and thus Debt Accumulation negligibly aborts with `blacklisted`. In the real model, φ is set as $\mathsf{PRF}(\lambda, x + 1)$ and φ has never been output before. If $\mathcal{Z}^{\text{correct}}$ inputs a $bl_{\mathcal{R}}$ such that `blacklisted` occurs with non-negligible probability this immediately yields an efficient adversary $\mathcal{B}$ against the unpredictability of PRF. Assume such a $\mathcal{Z}^{\text{correct}}$ exists, then $\mathcal{B}$ can be constructed as follows: $\mathcal{B}$ internally executes the real experiment, i.e., it runs $\mathcal{Z}^{\text{correct}}$ and plays all parties and the adversary for $\mathcal{Z}^{\text{correct}}$ honestly in its head. Externally $\mathcal{B}$ plays the unpredictability game. $\mathcal{B}$ needs to guess for which wallet $\mathcal{Z}^{\text{correct}}$ eventually causes an abort. In the scope of Wallet Issuing $\mathcal{B}$ picks an unused seed $\tilde{\lambda}$ but queries its external PRF-oracle for $x = 0$. N.b., $\mathcal{B}$ never uses $\tilde{\lambda}$, it is only a placeholder for the unknown challenge λ^C used by the external oracle. Whenever $\mathcal{B}$ needs to evaluate $\mathsf{PRF}(\tilde{\lambda}, x)$ it queries its external oracle instead. Note, that $\mathcal{Z}^{\text{correct}}$ cannot distinguish, if it is executed within a reduction, because $\tilde{\lambda}$ and λ^C are both randomly chosen. $\mathcal{B}$ guesses the interaction in which $\mathcal{Z}^{\text{correct}}$ causes the abort, takes the blacklist $bl_{\mathcal{R}}$ from $\mathcal{Z}^{\text{correct}}$, guesses what element of the blacklist is the next image of PRF and outputs this externally.

□

Theorem E.21 (System Correctness). *Let* $\mathcal{Z}^{\text{correct}}$ *be an environment that does not corrupt any parties. Under the assumption that all building blocks are correct,* $\mathsf{C1}$ *is homomorphic,* PRF *is a pseudo-random function, and DLOG is a hard problem for* G_1

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

holds. In other words, $\pi_{\text{P4TC}}^{\mathcal{F}_{\text{CRS}}, \overline{\mathcal{G}}_{\text{bb}}}$ *is a correct implementation of* $\mathcal{F}^{\overline{\mathcal{G}}_{\text{bb}}}$.

Proof. A direct consequence of Lemmas E.10 to E.12 and E.15 to E.20. □

[30]N.b.: By abuse of notation we use the same variable for the real and the ideal experiment, but of course they are not necessarily equal—only computationally indistinguishable over the randomness of the experiment

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

Setup:

(1) Run a modified version of the algorithm $\text{CRS} \leftarrow \text{Setup}(1^n)$ with SetupPoK being replaced by SetupEPoK and C2.Gen being replaced by C2.SimGen.
(2) Record CRS, td_{spok} and td_{eqcom}.
(3) Set $\Omega^{\text{dsp}} := \emptyset$.
(4) Set $\overline{\Omega}^{\text{pp}}_{\mathcal{U}} := \emptyset$.

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

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

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

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

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

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

Fig. 45. The simulator for System Security

E.2 Proof of System Security

In this section we show the following theorem.

Theorem E.22 (System Security). *Under the assumptions of Theorem E.1 and static corruption of a subset of the users*

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

holds.

The definition of the UC-simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ for Theorem E.22 can be found in Figs. 45 to 48. 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. Please note that the simulator records more information during its execution—namely the Augmented Transaction Graph (cp. Definition E.13) and the set of augmented proof-of-participation information $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$—than is technically required by the simulator for an indistinguishable simulation from the viewpoint of the environment. This additional information only simplifies the presentation of the reduction argument within the security proof.

By skipping ahead we shortly explain how this additional tracking of information supports the proof by giving a concrete example for $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ and the binding property of the commitment scheme. Each of the upcoming security proofs roughly follows the same lines of argument. If an environment $\mathcal{Z}$ can efficiently distinguish between the real and the ideal experiment, then we can construct an efficient adversary $\mathcal{B}$ against one of the underlying

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) $\lambda'' \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$.

(3) Run $(c''_{\mathcal{T}}, d''_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (\lambda'', 0))$, $(c''_{\text{ser}}, d^{\text{sim}}_{\text{ser}}) \leftarrow \mathsf{C2.SimCom}(\mathsf{CRS}^2_{\text{com}})$.

(4) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{issue})$ and obtain leaked $\mathbf{a}_{\mathcal{U}}$.[a]

(5) Output $(\mathbf{a}_{\mathcal{U}}, c''_{\text{ser}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$ to $\mathcal{Z}^{\text{sys-sec}}$ as the message from $\mathcal{T}$ to $\mathcal{U}$.

(6) Upon receiving $(\mathsf{pk}_{\mathcal{U}}, s', e^*, \sigma^*, \pi, e_0, \ldots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$ from $\mathcal{Z}^{\text{sys-sec}}$ in the name of $\mathcal{U}$ with $pid^*_{\mathcal{U}}$ …

(a) $stmnt := (\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}_{DR}, e^*, e_0, \ldots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$.

(b) If $\mathsf{S.Vfy}(\mathsf{pk}^{\text{auth}}_{\mathcal{U}}, \sigma^*, (e_0, \ldots, e_\ell, e^*)) = 0$ or $\mathsf{P1.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.

(c) Look up $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for which $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ has been recorded; if no $pid_{\mathcal{U}}$ exists abort.

(d) If $pid_{\mathcal{U}} \neq pid^*_{\mathcal{U}}$ and $pid_{\mathcal{U}} \notin \mathcal{PID}_{\text{corrupt}}$ give up simulation.

(e) Extract $Wit = (\Lambda', \Lambda'_0, \ldots, \Lambda'_\ell, R^*, R_1, \ldots, R_\ell, \Lambda', \Lambda'_0, \ldots, \Lambda'_\ell, \Lambda', \Lambda'_0, \ldots, \Lambda'_\ell, U_1, d'_{\mathcal{R}}, d'_{\mathcal{T}}, \mathsf{SK}^{\text{id}}_{\mathcal{U}}) \leftarrow \mathsf{P1.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.

(f) Set $htd := (\lambda'', e_0, \ldots, e_\ell, e^*, \sigma^*)$ and extract λ from htd.

(g) Provide alternative user PID $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.

(h) Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide φ, return $\varphi := \mathsf{PRF}(\lambda, x)$ with $x := 0$ to $\mathcal{F}_{\text{P4TC}}$.

(i) Obtain the user's output $(s, \mathbf{a}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ and set $b := 0$.

(j) Run $c_{\mathcal{T}} := c'_{\mathcal{T}} \cdot c''_{\mathcal{T}}$, $\sigma_{\mathcal{T}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{T}}_{\mathcal{T}}, (c_{\mathcal{T}}, \mathbf{a}_{\mathcal{U}}))$, $(c''_{\mathcal{R}}, d''_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (\lambda'', 0, 0, 1))$, $c_{\mathcal{R}} := c'_{\mathcal{R}} \cdot c''_{\mathcal{R}}$ and $\sigma_{\mathcal{R}} \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\mathcal{R}}_{\mathcal{T}}, (c_{\mathcal{R}}, s))$ honestly as the real protocol would do.

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

(l) Set $s^{\text{prev}} := \bot$, $p := 0$, $b := 0$, $c^{\text{in}}_{\mathcal{T}} := \bot$, $d^{\text{in}}_{\mathcal{T}} := \bot$, $M^{\text{in}}_{\mathcal{T}} := \bot$, $c^{\text{in}}_{\mathcal{R}} := \bot$, $d^{\text{in}}_{\mathcal{R}} := \bot$, $M^{\text{in}}_{\mathcal{R}} := \bot$, $c^{\text{out}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{out}}_{\mathcal{T}} := d'_{\mathcal{T}} \cdot d''_{\mathcal{T}}$, $M^{\text{out}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}})$, $c^{\text{out}}_{\mathcal{R}} := c_{\mathcal{R}}$, $d^{\text{out}}_{\mathcal{R}} := d'_{\mathcal{R}} \cdot d''_{\mathcal{R}}$, and $M^{\text{out}}_{\mathcal{R}} := (\Lambda, g_1^b, U_1, g_1^{x+1}, s)$.

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

(n) Output $(s'', d''_{\text{ser}}, \lambda'', c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, c_{\mathcal{T}}, d''_{\mathcal{T}}, \sigma_{\mathcal{T}})$ to $\mathcal{Z}^{\text{sys-sec}}$ as the message from $\mathcal{T}$ to $\mathcal{U}$.

[a]Do not provide an alternative $pid_{\mathcal{U}}$ immediately, but let $\mathcal{F}_{\text{P4TC}}$ sleep.

Fig. 46. The simulator for System Security (cont. from Fig. 45)

cryptographic building blocks, i.e., the binding property of the commitment scheme for the example at hand. 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 and the ideal functionality internally, it only has *black-box access* to them. In other words, although everything happens inside "the head of $\mathcal{B}$" it cannot somehow magically extract $\mathcal{Z}$'s attack strategy. Moreover, in order for $\mathcal{B}$ to win the external game against the binding property the mere existence of a break does not suffice, but $\mathcal{B}$ actually has to present two different openings for the same commitment. In order to do so, $\mathcal{B}$ observes all of $\mathcal{Z}$'s communication and systematically records parts of the observed transcript such that it can efficiently look up a commitment and its (former) opening if the same commitment with a different opening occurs a second time. This is the only reason why the simulator additionally records the observed

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 $u_2 \xleftarrow{\text{R}} \mathbb{Z}_p$.

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

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

(5) Upon receiving $(s', \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ from $\mathcal{Z}$ in the name of $\mathcal{U}$ ...

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

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

(c) Extract $Wit = (X, \Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, U_1, s^{\text{prev}}, \varphi^{\text{prev}}, X, \Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, B^{\text{prev}}, U_1, U^{\text{next}}_1, d_{\text{hid}}, d^{\text{prev}}_{\mathcal{R}}, d'_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\text{prev}}_{\mathcal{R}}, c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{certprev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.

(d) Look up $\overline{trdb}^* := (s^{\text{prev},*}, s^*, \varphi^*, x^*, \lambda^*, pid^*_{\mathcal{U}}, pid^*_{\mathcal{T}}, p^*, b^*, c^{\text{in}*}_{\mathcal{T}}, d^{\text{in}*}_{\mathcal{T}}, M^{\text{in}*}_{\mathcal{T}}, c^{\text{in}*}_{\mathcal{R}}, d^{\text{in}*}_{\mathcal{R}}, M^{\text{in}*}_{\mathcal{R}}, c^{\text{out}*}_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, c^{\text{out}*}_{\mathcal{R}}, d^{\text{out}*}_{\mathcal{R}}, M^{\text{out}*}_{\mathcal{R}})$ with $c^{\text{out}*}_{\mathcal{R}} = c^{\text{prev}}_{\mathcal{R}}$ and $c^{\text{out}*}_{\mathcal{T}} = c_{\mathcal{T}}$ being used as keys; if no unique entry exists, give up simulation (event *F1*).

(e) Give up simulation if any of these conditions meet: $\Lambda \neq g_1^{\lambda^*}$ (event *F2*), $\mathsf{pk}^{\text{id}}_{\mathcal{U}} \neq \mathsf{pk}^{\text{id}*}_{\mathcal{U}}$ for $M^{\text{out}*}_{\mathcal{T}} = (\Lambda^*, \mathsf{pk}^{\text{id}*}_{\mathcal{U}})$ (event *F3*), $s^{\text{prev}} \neq s^*$ (event *F4*), $B^{\text{prev}} \neq g_1^{b^*}$ (event *D1*), or $X \neq g_1^{x^*+1}$ (event *D2*).

(f) Retrieve $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ and assert $pid^*_{\mathcal{U}} = pid_{\mathcal{U}}$ else abort (event *F5*).

(g) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{pay\_toll}, s^{\text{prev}})$; upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide an alternative PID, return $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.

(h) If $\mathcal{F}_{\text{P4TC}}$ aborts for some other reason than `blacklisted`, give up simulation (event *F6*).

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

(j) 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.

(k) Obtain the user's output $(s, \mathbf{a}_{\mathcal{R}}, p, b)$ from $\mathcal{F}_{\text{P4TC}}$.

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

(m) 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.

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

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

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

(q) Append $\omega^{\text{dsp}} = (\varphi, t, u_2)$ to Ω^{dsp}

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

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

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

(s) Output $(s'', d''_{\text{ser}}, c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, p)$ to $\mathcal{Z}$ as the second 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. 47. The simulator for System Security (cont. from Fig. 45)

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 $u_2 \xleftarrow{\text{R}} \mathbb{Z}_p$.

(3) Output u_2 to $\mathcal{Z}$ as the first message from $\mathcal{T}$ to $\mathcal{U}$.

(4) Upon receiving $(\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{prev}}, t)$ from $\mathcal{Z}$ in the name of $\mathcal{U}$ …

(a) $stmnt := (\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{prev}}}, t, u_2)$.

(b) If $\text{P3.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 0$ let $\mathcal{F}_{\text{P4TC}}$ abort.

(c) Extract $Wit = (X, \Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, U_1, s^{\text{prev}}, \varphi^{\text{prev}}, X, \Lambda, U_1, d^{\text{prev}}_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\text{prev}}_{\mathcal{R}}, c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{cert}\,\text{prev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) \leftarrow \text{P3.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$.

(d) Lookup $\overline{trdb}^{*} := (s^{\text{prev},*}, s^*, \varphi^*, x^*, \lambda^*, pid^*_{\mathcal{U}}, pid^*_{\mathcal{T}}, p^*, b^*, c^{\text{in}\,*}_{\mathcal{T}}, d^{\text{in}\,*}_{\mathcal{T}}, M^{\text{in}\,*}_{\mathcal{T}}, c^{\text{in}*}_{\mathcal{R}}, d^{\text{in}*}_{\mathcal{R}}, M^{\text{in}*}_{\mathcal{R}}, c^{\text{out}*}_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, c^{\text{out}*}_{\mathcal{R}}, d^{\text{out}*}_{\mathcal{R}}, M^{\text{out}*}_{\mathcal{R}})$ with $c^{\text{out}*}_{\mathcal{R}} = c^{\text{prev}}_{\mathcal{R}}$ and $c^{\text{out}*}_{\mathcal{T}} = c_{\mathcal{T}}$ being used as keys; if no unique entry exists, give up simulation (event *F1*).

(e) Give up simulation if any of these conditions meet: $\Lambda \neq g_1^{\lambda^*}$ (event *F2*), $\mathsf{pk}^{\text{id}}_{\mathcal{U}} \neq \mathsf{pk}^{\text{id}\,*}_{\mathcal{U}}$ with $M^{\text{out}*}_{\mathcal{T}} = (\Lambda^*, \mathsf{pk}^{\text{id}\,*}_{\mathcal{U}})$ (event *F3*), $s^{\text{prev}} \neq s^*$ (event *F4*), or $B^{\text{prev}} \neq g_1^{b^*}$ (event *D1*).

(f) Retrieve $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ and assert $pid^*_{\mathcal{U}} = pid_{\mathcal{U}}$ else abort (event *F5*).

(g) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\texttt{clear\_debt}, s^{\text{prev}})$; upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide an alternative PID, return $pid_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$.

(h) If $\mathcal{F}_{\text{P4TC}}$ aborts, give up simulation (event *F6*).

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

(j) Obtain the user's output (b^{bill}) from $\mathcal{F}_{\text{P4TC}}$.

(k) Set $c^{\text{in}}_{\mathcal{T}} := c_{\mathcal{T}}$, $d^{\text{in}}_{\mathcal{T}} := d_{\mathcal{T}}$, $M^{\text{in}}_{\mathcal{T}} := (\Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}})$, $c^{\text{in}}_{\mathcal{R}} := c^{\text{prev}}_{\mathcal{R}}$, $d^{\text{in}}_{\mathcal{R}} := d^{\text{prev}}_{\mathcal{R}}$, $M^{\text{in}}_{\mathcal{R}} := (\Lambda, g_1^{b^{\text{prev}}}, U_1, X, s^{\text{prev}})$, $c^{\text{out}}_{\mathcal{T}} := \bot$, $d^{\text{out}}_{\mathcal{T}} := \bot$, $M^{\text{out}}_{\mathcal{T}} := \bot$, $c^{\text{out}}_{\mathcal{R}} := \bot$, $d^{\text{out}}_{\mathcal{R}} := \bot$, and $M^{\text{out}}_{\mathcal{R}} := \bot$.

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

(m) Append $\omega^{\text{dsp}} = (\varphi, t, u_2)$ to Ω^{dsp}

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

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

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

(o) Output $(\texttt{OK})$ to $\mathcal{Z}$ as the second 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. 48. The simulator for System Security (cont. from Fig. 45)

opening messages in $\overline{\Omega}^{\text{pp}}_{\mathcal{U}}$ but never uses them for its own simulation. These opening are only required in the outer game during the reduction argument.

As already stated in Appendix B we assume half-authenticated secure channels. In this model the adversary only learns the length of the messages being sent between honest parties. Secure channels can be realized by

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

Prove Participation:

(1) Call $\mathcal{F}_{\text{P4TC}}$ with input (`prove_participation`).
(2) Obtain leaked set $S^{\text{pp}}_{\mathcal{R}}$ of serial numbers from $\mathcal{F}_{\text{P4TC}}$.
(3) Obtain the user's output $(\mathsf{out}_{\mathcal{U}})$ from $\mathcal{F}_{\text{P4TC}}$ and delay the output of the *SA*.
(4) Output $S^{\text{pp}}_{\mathcal{R}}$ to $\mathcal{Z}$ as message from *SA* to $\mathcal{U}$.
(5) Upon receiving $(s, c_{\text{hid}}, d_{\text{hid}})$ from $\mathcal{Z}$ in the name of $\mathcal{U}$ …
  (a) If $(s, c_{\text{hid}}, \cdot, \cdot) \notin \overline{\Omega}^{\text{pp}}_{\mathcal{U}}$, let $\mathcal{F}_{\text{P4TC}}$ abort.
  (b) Look up $\mathsf{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for PID $pid_{\mathcal{U}}$; if no $\mathsf{pk}_{\mathcal{U}}$ has been recorded abort.
  (c) Parse $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ from $\mathsf{pk}_{\mathcal{U}}$.
  (d) If $\mathsf{C1.Open}(\mathsf{CRS}^{1}_{\text{com}}, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, c_{\text{hid}}, d_{\text{hid}}) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.
  (e) If $\mathsf{out}_{\mathcal{U}} = \mathsf{NOK}$ give up simulation.
  (f) Else let $\mathcal{F}_{\text{P4TC}}$ return the delayed output to the *SA*.

Double-Spending Detection: Upon being ask to provide a proof for $pid_{\mathcal{U}}$, look up $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, parse $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$ from $\mathsf{sk}_{\mathcal{U}}$ and return $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$.

Guilt Verification: Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide out for $(pid_{\mathcal{U}}, \pi)$ …

(1) Receive $\mathsf{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{U}}$ and parse $(\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}^{\text{auth}}_{\mathcal{U}}) := \mathsf{pk}_{\mathcal{U}}$
(2) If $g_1^{\pi} = \mathsf{pk}^{\text{id}}_{\mathcal{U}}$, then return $\mathsf{out} := \mathsf{OK}$, else $\mathsf{out} := \mathsf{NOK}$ to $\mathcal{F}_{\text{P4TC}}$.

User Blacklisting: Upon receiving (λ, x) and being ask to provide a serial number, return $\varphi := \mathsf{PRF}(\lambda, x)$.

Fig. 49. The simulator for System Security (cont. from Fig. 45)

any IND-CCA secure encryption scheme (cp. [14]). Secure channels release us from the burden to spell out a simulator for messages between honest parties.

First, note the following simple observation.

Lemma E.23. *The messages simulated by $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ in the name of honest parties and sent to $\mathcal{Z}^{\text{sys-sec}}$ are perfect, i.e., statistically identical to real messages.*

Proof. Follows from the definition of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ (cp. Figs. 45 to 48). □

The messages generated by $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ do not depend on any unknown secrets. Moreover, another simple observation is

Lemma E.24. *Whenever any $\mathcal{Z}^{\text{sys-sec}}$ triggers an abort of one of the tasks in the real experiment $\mathsf{EXEC}_{\pi^{\mathcal{F}_{\text{CRS}}}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$, the simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ lets $\mathcal{F}_{\text{P4TC}}$ also abort in the ideal experiment $\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$.*

Proof. Follows from the definition of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ (cp. Figs. 45 to 48). □

An abort in the real model only happens if one of the messages is invalid, e.g., a signature does not verify or a ZK proof is not valid. If $\mathcal{Z}^{\text{sys-sec}}$ playing the corrupted parties sends a message in the ideal experiment, the simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ playing the honest receiver performs the same checks and lets $\mathcal{F}_{\text{P4TC}}$ abort, if necessary.

Due to Lemmas E.23 and E.24 there is only a limited number of ways how $\mathcal{Z}^{\text{sys-sec}}$ can distinguish between the real and the ideal experiment. All options fall into one out of two categories:

- $\mathcal{Z}^{\text{sys-sec}}$ triggers an abort in the ideal experiment that does not exist in the real experiment. In other words $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ is not able to continue the simulation and gives up.

- $\mathcal{Z}^{\text{sys-sec}}$ makes the output of an honest party differ between the real and ideal experiment. As this events are directly observable by the simulator $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ as well, $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ proceeds as above and gives up simulation.

We call events of the first type *failure events* and events of the second type *discrepancy events*. Naturally, the proof of Theorem E.22 proceeds in two parts according to these categories. Please note that discrepancy events do not immediately require $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ to give up, we could also decide to let $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ ignore this kind of event. But this way, the proof is simpler. Both parts heavily rely on the Augmented Transaction Graph.

In the first part we exploit that its graph structure is redundantly defined: Edges are defined by linking serial numbers to predecessor serial numbers and a second set of edges is defined by means of in- and out-commitments. As already shown in Appendix E.1, both of these graph structures coincide if all parties are honest. We show that this still holds with overwhelming probability if the users are corrupted. More precisely, a *failure event* is a successful attempt of $\mathcal{Z}^{\text{sys-sec}}$ to make both graph structures fall apart. In this case $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ immediately gives up the simulation. We show that a failure event implies a successful attack on one of our cryptographic building blocks.

The second part of the proof considers executions of the UC-experiment in which $\mathcal{Z}^{\text{sys-sec}}$ does not force $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ to give up simulation prematurely, but in which $\mathcal{Z}^{\text{sys-sec}}$ can distinguish due to different input/output distributions of honest parties.

The category of *failure events* comprises the following individual events:

Definition E.25 (Failure Events). $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ gives up simulation because. . .

(*F1*) The extracted $c_{\mathcal{T}}$ and $c^{\text{prev}}_{\mathcal{R}}$ do not point to a unique predecessor (cp. Fig. 46, Step 5d and Fig. 47, Step 4d).
(*F2*) The extracted Λ does not correspond to the previously recorded λ^* in the Augmented Transaction Graph (cp. Fig. 46, Step 5e and Fig. 47, Step 4e).
(*F3*) The extracted $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ does not match the previously recorded $\mathsf{pk}^{\text{id}\,*}_{\mathcal{U}}$ in the Augmented Transaction Graph (cp. Fig. 46, Step 5e and Fig. 47, Step 4e).
(*F4*) The extracted s^{prev} does not match the previously recorded s^* in the Augmented Transaction Graph (cp. Fig. 46, Step 5e and Fig. 47, Step 4e).
(*F5*) The extracted $\mathsf{pk}^{\text{id}}_{\mathcal{U}}$ has not been registered in $\overline{\mathcal{G}}_{\text{bb}}$ (cp. Fig. 46, Step 5f) and Fig. 47, Step 4f).
(*F6*) $\mathcal{F}_{\text{P4TC}}$ aborts for some other reason than `blacklisted` (cp. Fig. 46, Step 5h and Fig. 47, Step 4h).

The category of *discrepancy events* comprises the following individual events:

Definition E.26 (Discrepancy Events).

(*D1*) The dummy $\mathcal{T}$ outputs a different b^{bill} to $\mathcal{Z}^{\text{sys-sec}}$ in the scope of Debt Clearance than a real $\mathcal{T}$ running π_{P4TC} (cp. Fig. 46, Step 5e and Fig. 47, Step 4e).
(*D2*) $\mathcal{Z}^{\text{sys-sec}}$ triggers a non-negligible discrepancy between the abort behavior of the dummy $\mathcal{T}$ and the real $\mathcal{T}$ within the scope of Debt Accumulation (cp. Fig. 46, Step 5e).
(*D3*) The dummy SA outputs a different $\mathtt{out}_{SA}$ to $\mathcal{Z}^{\text{sys-sec}}$ in the scope of Prove Participation than the real *SA* (cp. Fig. 48, Step 5e).

We start by ruling out failure events. As a preliminary step we need some kind of “loop invariant” that holds as long as no failure event has occurred.

Lemma E.27. *For every execution of*

$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}},\overline{\mathcal{G}}_{\text{bb}},\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}},\mathcal{Z}^{\text{sys-sec}}}(1^n)$$

that does not trigger a failure event the two graph structures in the Augmented Transaction Graph $\overline{TRDB}$ maintained by $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ are identical to the Ideal Transaction Graph TRDB maintained by $\mathcal{F}_{\text{P4TC}}$ immediately after the first *message from the user (cp. Fig. 45 Step 6, Fig. 46, Step 5 and Fig. 47, Step 4) and immediately before the last message to the user (cp. Fig. 45 Step 6n, Fig. 46, Step 5s and Fig. 47, Step 4o) within the scope of the tasks Wallet Issuing, Debt Accumulation and Debt Clearance.*

Proof. (Informal) The statement is true for every execution of $\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}},\overline{\mathcal{G}}_{\text{bb}},\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}},\mathcal{Z}^{\text{sys-sec}}}(1^n)$ that never triggered Wallet Issuing, Debt Accumulation or Debt Clearance. Assume the statement holds at the end of n such interactions when the $(n+1)^{\text{th}}$ interaction starts. By assumption the Ideal Transaction Graph of $\mathcal{F}_{\text{P4TC}}$ and the two graph structures of the Augmented Transaction graph of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ are in sync before the first message. After the the first message of the $(n+1)^{\text{th}}$ interaction has been received, the $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ conducts a collection of consistency checks, that by assumption do not fail (otherwise a failure event is triggered). The ideal function is invoked and terminates without abort (again by assumption). During its execution $\mathcal{F}_{\text{P4TC}}$ updates its internal Ideal Transaction Graph *TRDB*. After $\mathcal{F}_{\text{P4TC}}$ returned its output to the $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ and before $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ sends its last message $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ updates the Augmented Transaction Graph $\overline{TRDB}$. By definition of $\mathcal{F}_{\text{P4TC}}$ and $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ both updates are consistent. □

An immediate consequence of Lemma E.27 is the following

Lemma E.28 (No unexpected abort). *The failure event (F6) does not occur.*

Proof. Assume for contradiction that (*F6*) has occurred. This implies that no other failure event has occurred before, in particular none of the events (*F3*) or (*F4*). Due to Lemma E.27 both transaction graphs *TRDB* and $\overline{TRDB}$ are in sync. But $\mathcal{F}_{\text{P4TC}}$ only aborts unexpectedly and raises (*F6*) if the predecessor serial number s^{prev} and the PID $pid_{\mathcal{U}}$ are not recorded in *TRDB*. In this case (*F3*) or (*F4*) must have occurred before. □

Lemma E.29. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}},\overline{\mathcal{G}}_{\text{bb}},\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}},\mathcal{Z}^{\text{sys-sec}}}(1^n)$$

the event (F1) (cp. Definition E.25) only occurs with negligible probability for any PPT environment $\mathcal{Z}^{\text{sys-sec}}$ that is restricted to static corruption of users.

Proof. Assume a $\mathcal{Z}^{\text{sys-sec}}$ exists that triggers (*F1*) with non-negligible probability. Without loss of generality we only consider that (*F1*) occurs in Debt Accumulation; the case of Debt Clearance is analogous. Let $(s', \pi, \varphi, \mathrm{a}_{\mathcal{U}}, \mathrm{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ denote the message that was sent by $\mathcal{Z}^{\text{sys-sec}}$ and raised (*F1*). Let $Wit \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$ denote the extracted witness and parse it as $(\cdot, \Lambda, \ldots, s^{\text{prev}}, \ldots, d^{\text{prev}}_{\mathcal{R}}, d_{\mathcal{T}}, \mathsf{pk}^{\text{prev}}_{\mathcal{R}}, c^{\text{prev}}_{\mathcal{R}}, c_{\mathcal{T}}, \sigma^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{certprev}}_{\mathcal{R}}, \sigma_{\mathcal{T}}) := Wit$.
We observe the following facts:

- $\mathsf{P2.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 1$ holds, otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ would have let $\mathcal{F}_{\text{P4TC}}$ aborted earlier.
- As P2 is perfectly sound, the equations

$$\begin{aligned}
\mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, \ldots), c_{\mathcal{T}}, d_{\mathcal{T}}) &= 1\\
\mathsf{C1.Open}(\mathsf{CRS}^1_{\text{com}}, (\Lambda, \ldots), c^{\text{prev}}_{\mathcal{R}}, d^{\text{prev}}_{\mathcal{R}}) &= 1\\
\mathsf{S.Vfy}(\mathsf{pk}^{\mathcal{T}}_{\mathcal{T}}, \sigma_{\mathcal{T}}, (c_{\mathcal{T}}, \mathrm{a}_{\mathcal{U}})) &= 1\\
\mathsf{S.Vfy}(\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \sigma^{\text{prev}}_{\mathcal{R}}, (c^{\text{prev}}_{\mathcal{R}}, s^{\text{prev}})) &= 1\\
\mathsf{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{certprev}}_{\mathcal{R}}, (\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \ldots)) &= 1
\end{aligned}$$

  hold.

First we deal with the cases that at least one of $c_{\mathcal{T}}$ or $c^{\text{prev}}_{\mathcal{R}}$ does not exist as an out-commitment in $\overline{TRDB}$.

(1) *The commitment $c_{\mathcal{T}}$ does not exist as an out-commitment:* In other words, the commitment $c_{\mathcal{T}}$ has never been issued by the TSP. 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 simulates $\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}},\overline{\mathcal{G}}_{\text{bb}},\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}},\mathcal{Z}^{\text{sys-sec}}}(1^n)$ for $\mathcal{Z}^{\text{sys-sec}}$, i.e., $\mathcal{B}$ internally plays the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ and $\mathcal{F}_{\text{P4TC}}$. 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 $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ 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 $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ 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 (*F1*) occurs, $\mathcal{B}$ extracts $c_{\mathcal{T}}$ and $\sigma_{\mathcal{T}}$ from the proof and outputs the forgery.

(2) *The commitment $c^{\text{prev}}_{\mathcal{R}}$ does not exist as an out-commitment:* We need to distinguish two sub-cases. On an abstract level these sub-cases correspond to the following scenarios: Either the previous RSU exists. Then the signature on $c^{\text{prev}}_{\mathcal{R}}$ is a forgery. Or alternatively, the allegedly previous RSU does not exits but has been imagined by the user. Then $c^{\text{prev}}_{\mathcal{R}}$ may have a honest, valid signature (because the user feigned the RSU), but the certificate for the fake RSU is a forgery.

(a) *A record $pid^{\text{prev}}_{\mathcal{R}} \mapsto (\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \mathsf{sk}^{\text{prev}}_{\mathcal{R}})$ has been recorded:* Again, we construct an efficient adversary $\mathcal{B}$ against the EUF-CMA security of S along the same lines as above. But in this case, $\mathcal{B}$ needs to guess for which $pid^{\text{prev}}_{\mathcal{R}}$ the event (*F1*) eventually occurs. When the RSU with $pid^{\text{prev}}_{\mathcal{R}}$ registers itself, and $\mathcal{B}$ playing $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ needs to provide $\mathsf{pk}^{\text{prev}}_{\mathcal{R}}$ it embeds the challenge as $\mathsf{pk}^{\text{prev}}_{\mathcal{R}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ needs to issue a signature with respect to $\mathsf{pk}^{\text{prev}}_{\mathcal{R}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event (*F1*) occurs, $\mathcal{B}$ extracts $(c^{\text{prev}}_{\mathcal{R}}, s^{\text{prev}})$ and $\sigma^{\text{prev}}_{\mathcal{R}}$ from the proof and outputs the forgery. N.b., has never been signed wrpt. $\mathsf{pk}^{\text{prev}}_{\mathcal{R}} = \mathsf{pk}_C$ by assumption.

(b) *No $pid^{\text{prev}}_{\mathcal{R}} \mapsto (\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \mathsf{sk}^{\text{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$ has been recorded:* 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 simulates $\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}},\overline{\mathcal{G}}_{\text{bb}},\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}},\mathcal{Z}^{\text{sys-sec}}}(1^n)$ for $\mathcal{Z}^{\text{sys-sec}}$, i.e., $\mathcal{B}$ internally plays the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ and $\mathcal{F}_{\text{P4TC}}$. 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 $\mathsf{pk}_{\mathcal{T}} = (\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{R}}_{\mathcal{T}}, \mathsf{pk}^{\mathcal{T}}_{\mathcal{T}})$ playing the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ in the scope of the TSP Registration, $\mathcal{B}$ embeds the external challenge as $\mathsf{pk}^{\text{cert}}_{\mathcal{T}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ in the scope of RSU Certification needs to issue signatures with respect to $\mathsf{pk}^{\text{cert}}_{\mathcal{T}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event (*F1*) occurs, $\mathcal{B}$ extracts $\mathsf{cert}_{\mathcal{R}}{}^{\text{prev}}$ and $\sigma^{\text{certprev}}_{\mathcal{R}}$ from the proof and outputs the forgery. N.b.: $\mathsf{cert}_{\mathcal{R}}{}^{\text{prev}}$ has never been signed by the TSP with respect to $\mathsf{pk}^{\text{cert}}_{\mathcal{T}} = \mathsf{pk}_C$ as otherwise a mapping $pid^{\text{prev}}_{\mathcal{R}} \mapsto (\mathsf{pk}^{\text{prev}}_{\mathcal{R}}, \mathsf{sk}^{\text{prev}}_{\mathcal{R}}, \mathsf{cert}_{\mathcal{R}})$ would have been recorded.

At this point we know that both commitments $c_{\mathcal{T}}$ and $c^{\text{prev}}_{\mathcal{R}}$ exist as an out-commitment somewhere in $\overline{TRDB}$. By assumption there is no single $\overline{trdb}^*$ with both commitments as out-commitments. Let $\overline{trdb}^*$ denote the transaction entry with $c^{\text{out}*}_{\mathcal{T}} = c_{\mathcal{T}}$ and Let $\overline{trdb}^{**}$ denote the transaction entry with $c^{\text{out}**}_{\mathcal{R}} = c^{\text{prev}}_{\mathcal{R}}$. We have

$$\overline{trdb}^* = (\ldots, \lambda^*, \ldots, \underbrace{c^{\text{out}*}_{\mathcal{T}}}_{=c_{\mathcal{T}}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, \ldots)$$

$$\overline{trdb}^{**} = (\ldots, \lambda^{**}, \ldots, \underbrace{c^{\text{out}**}_{\mathcal{R}}}_{=c^{\text{prev}}_{\mathcal{R}}}, d^{\text{out}**}_{\mathcal{R}}, M^{\text{out}**}_{\mathcal{R}})$$

and $\overline{trdb}^{*} \neq \overline{trdb}^{**}$. Because no failure event occurred before, *TRDB* and $\overline{TRDB}$ are in sync by Lemma E.27, *TRDB* is correct by definition and hence all $c^{\text{in}}_{\mathcal{T}\,i}{}^{*}, c^{\text{out}*}_{\mathcal{T}\,i}$ are constant and equal to $c_{\mathcal{T}}$ within the tree to that the node $\overline{trdb}^{*}$ belongs. This implies the nodes $\overline{trdb}^{*}$ and $\overline{trdb}^{**}$ and therefore the commitments $c_{\mathcal{T}}$ and $c^{\text{prev}}_{\mathcal{R}}$ belong to different trees. From Lemma E.4 we conclude that $\lambda^{*} \neq \lambda^{**}$ holds with overwhelming probability. On the one hand it follows that

$$\begin{aligned} M^{\text{out}*}_{\mathcal{T}} &= (\Lambda^{*}, \mathsf{pk}^{\text{id}}_{\mathcal{U}}{}^{*}) \qquad \text{and} \\ M^{\text{out}**}_{\mathcal{R}} &= (\Lambda^{**}, \ldots) \end{aligned}$$

are implicit openings with different wallet IDs. On the other hand the perfect correctness and extractability of the proof system yield

$$\begin{aligned} M_{\mathcal{T}} &= (\Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}) \qquad \text{and} \\ M^{\text{prev}}_{\mathcal{R}} &= (\Lambda, \ldots) \end{aligned}$$

to be valid implicit openings to the same wallet ID. In summary, at least one of the commitments $c^{\text{out}*}_{\mathcal{T}} = c_{\mathcal{T}}$ or $c^{\text{out}**}_{\mathcal{R}} = c^{\text{prev}}_{\mathcal{R}}$ has two different implicit openings. 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 simulates $\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^{n})$ for $\mathcal{Z}^{\text{sys-sec}}$, i.e., $\mathcal{B}$ internally plays the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ and $\mathcal{F}_{\text{P4TC}}$. When the event (*F1*) occurs, $\mathcal{B}$ searches the forest $\overline{TRDB}$ for the duplicate of the commitment and outputs both openings. □

The above lemma requires a remark. If the lemma is used to construct an efficient adversary $\mathcal{B}$ out of an environment $\mathcal{Z}^{\text{sys-sec}}$ that triggers the failure event (*F1*), the adversary $\mathcal{B}$ is fixated on the security game it plays externally. As $\mathcal{B}$ uses $\mathcal{Z}^{\text{sys-sec}}$ in a black-box fashion it does not know how $\mathcal{Z}^{\text{sys-sec}}$ eventually causes the event (*F1*). But there are only finite many options and if the probability that $\mathcal{Z}^{\text{sys-sec}}$ distinguishes correctly is non-negligible, then at least one of the options is non-negligible, too. In other words, $\mathcal{B}$ needs to push its luck that $\mathcal{Z}^{\text{sys-sec}}$ breaks the same underlying building block which $\mathcal{B}$ attacks externally. If $\mathcal{B}$ guesses wrong, $\mathcal{B}$ loses its external game, but the advantage is still non-negligible.

Lemma E.30. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^{n})$$

the events (*F2*) *or* (*F3*) *(cp. Definition E.25) only occur with negligible probability for any PPT environment* $\mathcal{Z}^{\text{sys-sec}}$ *that is restricted to static corruption of users.*

Proof. Assume a $\mathcal{Z}^{\text{sys-sec}}$ exists that triggers (*F2*) or (*F3*) with non-negligible probability but not (*F1*) as otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ would have aborted earlier. Without loss of generality we only consider Debt Accumulation; the case of Debt Clearance is analogous. Let $(s', \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ denote the message that was sent by $\mathcal{Z}^{\text{sys-sec}}$ and raised (*F2*) or (*F3*). Let $Wit = (\ldots, \Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}, \ldots, d_{\mathcal{T}}, \ldots, c_{\mathcal{T}}, \ldots) \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\text{epok}}, stmnt, \pi)$ denote the extracted witness. We observe the following facts:

- $\mathsf{P2.Vfy}(\mathsf{CRS}_{\text{pok}}, stmnt, \pi) = 1$ holds, otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}$ would have let $\mathcal{F}_{\text{P4TC}}$ aborted earlier.
- As P2 is perfectly sound, the equation $\mathsf{C1.Open}(\mathsf{CRS}^{1}_{\text{com}}, (\Lambda, \mathsf{pk}^{\text{id}}_{\mathcal{U}}), c_{\mathcal{T}}, d_{\mathcal{T}}) = 1$, holds.

As (*F1*) has not occurred we know by Lemma E.29 that there is a unique $\overline{trdb}^{*} = (\ldots, \lambda^{*}, \ldots, c^{\text{out}*}_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}, M^{\text{out}*}_{\mathcal{T}}, \ldots)$ with $c^{\text{out}*}_{\mathcal{T}} = c_{\mathcal{T}}$ and $M^{\text{out}*}_{\mathcal{T}} = (\Lambda^{*}, \mathsf{pk}^{\text{id}}_{\mathcal{U}}{}^{*})$. Hence

$$\mathsf{C1.Open}(\mathsf{CRS}^{1}_{\text{com}}, (\Lambda^{*}, \mathsf{pk}^{\text{id}}_{\mathcal{U}}{}^{*}), c_{\mathcal{T}}, d^{\text{out}*}_{\mathcal{T}}) = 1$$

holds, too. By assumption at least one of $\Lambda^* \neq \Lambda$ or ${\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}}^* \neq \mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}$ holds. This immediately yields an efficient adversary $\mathcal{B}$ against the binding property of C1: $\mathcal{B}$ internally executes the experiment $\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \bar{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$ and plays $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ and $\mathcal{F}_{\mathrm{P4TC}}$ for $\mathcal{Z}^{\text{sys-sec}}$ until $\mathcal{B}$ observes event (*F2*) or (*F3*). Then $\mathcal{B}$ outputs $c_{\mathcal{T}}$ together with two different openings $M := (\Lambda, \mathsf{pk}^{\mathrm{id}}_{\mathcal{U}})$, $d_{\mathcal{T}}$ $M^{\mathrm{out}*}_{\mathcal{T}} = (\Lambda^*, {\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}}^*)$, and $d^{\mathrm{out}*}_{\mathcal{T}}$. □

Lemma E.31. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \bar{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$$

the event (*F4*) *(cp. Definition E.25) only occurs with negligible probability for any PPT environment* $\mathcal{Z}^{\text{sys-sec}}$ *that is restricted to static corruption of users.*

Proof. Assume a $\mathcal{Z}^{\text{sys-sec}}$ exists that triggers (*F4*) with non-negligible probability but not (*F1*) as otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ would have aborted earlier. Without loss of generality we only consider Debt Accumulation; the case of Debt Clearance is analogous. Let $(s', \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t)$ denote the message that was sent by $\mathcal{Z}^{\text{sys-sec}}$ and raised (*F4*). Let $\mathit{Wit} = (\ldots, s^{\mathrm{prev}}, \ldots, c^{\mathrm{prev}}_{\mathcal{R}}, \ldots, \sigma^{\mathrm{prev}}_{\mathcal{R}}, \ldots) \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\mathrm{epok}}, \mathit{stmnt}, \pi)$ denote the extracted witness. We observe the following facts:

- $\mathsf{P2.Vfy}(\mathsf{CRS}_{\mathrm{pok}}, \mathit{stmnt}, \pi) = 1$ holds, otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ would have let $\mathcal{F}_{\mathrm{P4TC}}$ aborted earlier.
- As P2 is perfectly sound, the equation $\mathsf{S.Vfy}(\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}, \sigma^{\mathrm{prev}}_{\mathcal{R}}, (c^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}})) = 1$ holds.

As (*F1*) has not occurred we know by Lemma E.29 that there exists a unique $\overline{\mathit{trdb}}^* = (\ldots, s^*, \ldots, \mathit{pid}^{\mathrm{prev}}_{\mathcal{R}}, \ldots, c^{\mathrm{out}*}_{\mathcal{R}})$ with $c^{\mathrm{out}*}_{\mathcal{R}} = c^{\mathrm{prev}}_{\mathcal{R}}$, but $s^* \neq s^{\mathrm{prev}}$. Due to the uniqueness of $c^{\mathrm{prev}}_{\mathcal{R}}$ the message $(c^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}})$ has never been signed and thus $\sigma^{\mathrm{prev}}_{\mathcal{R}}$ is a forgery. We show how to construct an efficient adversary $\mathcal{B}$ against the EUF-CMA security of S. Externally, $\mathcal{B}$ plays the EUF-CMA security experiment with a challenger C and a signing oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. Internally, $\mathcal{B}$ executes the experiment $\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \bar{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$ in its head and plays $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ and $\mathcal{F}_{\mathrm{P4TC}}$ for $\mathcal{Z}^{\text{sys-sec}}$. $\mathcal{B}$ needs to guess for which $\mathit{pid}^{\mathrm{prev}}_{\mathcal{R}}$ the event (*F1*) eventually occurs. When the RSU with $\mathit{pid}^{\mathrm{prev}}_{\mathcal{R}}$ registers itself, and $\mathcal{B}$ playing $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ needs to provide $\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}$ it embeds the challenge as $\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}} := \mathsf{pk}_C$. Whenever $\mathcal{B}$ playing the role of $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ needs to issue a signature with respect to $\mathsf{pk}^{\mathrm{prev}}_{\mathcal{R}}$, it does so using its external EUF-CMA oracle $\mathsf{O}^{\mathsf{S}}_{\mathsf{pk},\mathsf{sk}}$. When the event (*F4*) occurs, $\mathcal{B}$ extracts $(c^{\mathrm{prev}}_{\mathcal{R}}, s^{\mathrm{prev}})$ and $\sigma^{\mathrm{prev}}_{\mathcal{R}}$ from the proof and outputs the forgery. □

Lemma E.32. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \bar{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$$

the event (*F5*) *(cp. Definition E.25) does not occur for any PPT environment* $\mathcal{Z}^{\text{sys-sec}}$ *that is restricted to static corruption of users.*

Proof. Without loss of generality we only consider Debt Accumulation; the case of Debt Clearance is analogous. Let $(s', \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\mathrm{prev}}_{\mathcal{R}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t)$ denote the message that was sent by $\mathcal{Z}^{\text{sys-sec}}$ and raised (*F5*). Let $\mathit{Wit} \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\mathrm{epok}}, \mathit{stmnt}, \pi)$ denote the extracted witness and parse $(\ldots, \mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}, \ldots, c^{\mathrm{prev}}_{\mathcal{R}}, \ldots) := \mathit{Wit}$. As (*F1*) has not occurred we know by Lemma E.29 that there is a unique $\overline{\mathit{trdb}}^* = (\ldots, \mathit{pid}^*_{\mathcal{U}}, \ldots, c^{\mathrm{out}*}_{\mathcal{R}}, \ldots, M^{\mathrm{out}*}_{\mathcal{T}}, \ldots)$ with $c^{\mathrm{out}*}_{\mathcal{R}} = c^{\mathrm{prev}}_{\mathcal{R}}$. We distinguish two cases:

(1) $\overline{\mathit{trdb}}^*$ *is the result of a run of Debt Accumulation or Debt Clearance:* This immediately leads to a contradiction. The event (*F3*) has not occurred, as otherwise $\mathcal{S}^{\text{sys-sec}}_{\pi_{\mathrm{P4TC}}}$ would have aborted earlier. Hence $\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}} = {\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}}^*$ holds and this implies that ${\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}}^*$ does not exist in the bulletin board neither. However, $M^{\mathrm{out}*}_{\mathcal{T}} = (\Lambda^*, {\mathsf{pk}^{\mathrm{id}}_{\mathcal{U}}}^*)$

is the result of a successful previous run of Debt Accumulation or Debt Clearance and hence $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}\,*}$ must exist in the bulletin board. Contradiction!

(2) *$\overline{trdb}^{*}$ is the result of a run of Wallet Issuing:* The event ($F3$) has not occurred, as otherwise $\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}$ would have aborted earlier. Hence $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}} = \mathsf{pk}_{\mathcal{U}}^{\mathrm{id}\,*}$ holds and this implies that $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}\,*}$ does not exist in the bulletin board neither. However, $M_{\mathcal{T}}^{\mathrm{out}*} = (\Lambda^{*}, \mathsf{pk}_{\mathcal{U}}^{\mathrm{id}\,*})$ is the result of Wallet Issuing. But Wallet Issuing asserts that $\mathsf{pk}_{\mathcal{U}}^{\mathrm{id}\,*}$ exists or aborts (cp. Fig. 45, Step 6c). Contradiction!

□

Lemma E.33. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}, \mathcal{Z}^{\text{sys-sec}}}(1^{n})$$

the event ($D1$) *(cp. Definition E.26) only occurs with negligible probability for any PPT environment $\mathcal{Z}^{\text{sys-sec}}$ that is restricted to static corruption of users.*

Proof. Assume a $\mathcal{Z}^{\text{sys-sec}}$ exists that triggers ($D1$) with non-negligible probability but not ($F1$) as otherwise $\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}$ would have aborted earlier. Unveiling a different, final balance b^{bill} in the scope of Debt Clearance implies that the balance b^{*} of the (real) wallet diverges from the expected balance of the (ideal) wallet at some point along the path of transactions from the root node with $b^{*} = 0$ (Wallet Issue) to the leaf node with $b^{*} = b^{\mathrm{bill}}$ (Debt Clearance). Without loss of generality we only consider a transition between inner nodes (i.e., Debt Accumulation); the transition from the next-to-last to the leaf node (i.e., Debt Clearance) is analogous.

Let $(s', \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathrm{hid}}, c'_{\mathcal{R}}, t)$ denote the message that was sent by $\mathcal{Z}^{\text{sys-sec}}$ and raised ($D1$). Let $Wit = (X, \ldots, \Lambda, \ldots, B^{\mathrm{prev}}, \ldots, U_{1}, \ldots, d_{\mathcal{R}}^{\mathrm{prev}}, \ldots, c_{\mathcal{R}}^{\mathrm{prev}}, \ldots) \leftarrow \mathsf{P2.ExtractW}(\mathsf{CRS}, \mathsf{td}_{\mathrm{epok}}, \mathit{stmnt}, \pi)$ denote the extracted witness. We observe the following facts:

- $\mathsf{P2.Vfy}(\mathsf{CRS}_{\mathrm{pok}}, \mathit{stmnt}, \pi) = 1$ holds, otherwise $\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}$ would have let $\mathcal{F}_{\mathrm{P4TC}}$ aborted earlier.
- As P2 is perfectly sound, the equation $\mathsf{C1.Open}(\mathsf{CRS}_{\mathrm{com}}^{1}, M_{\mathcal{R}}^{\mathrm{prev}}, c_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{prev}}) = 1$ with $M_{\mathcal{R}}^{\mathrm{prev}} := (\Lambda, B^{\mathrm{prev}}, U_{1}, X)$, holds.

As ($F1$) has not occurred we know by Lemma E.29 that there is a unique $\overline{trdb}^{*} = (\ldots, \lambda^{*}, \ldots, c_{\mathcal{R}}^{\mathrm{out}*}, d_{\mathcal{R}}^{\mathrm{out}*}, M_{\mathcal{R}}^{\mathrm{out}*})$ with $c_{\mathcal{R}}^{\mathrm{out}*} = c_{\mathcal{R}}^{\mathrm{prev}}$ and hence $\mathsf{C1.Open}(\mathsf{CRS}_{\mathrm{com}}^{1}, M_{\mathcal{R}}^{\mathrm{out}*}, c_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{out}*}) = 1$ holds, too. Let be $M_{\mathcal{R}}^{\mathrm{out}*} = (\Lambda^{*}, g_{1}^{b^{*}}, U_{1}, g_{1}^{x^{*}+1})$. By assumption $g_{1}^{b^{*}} \neq B^{\mathrm{prev}}$ holds (raise condition for event ($D1$)) and so does $M_{\mathcal{R}}^{\mathrm{prev}} \neq M_{\mathcal{R}}^{\mathrm{out}*}$. This immediately yields an efficient adversary $\mathcal{B}$ against the binding property of C1: $\mathcal{B}$ internally executes the experiment $\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}, \mathcal{Z}^{\text{sys-sec}}}(1^{n})$ in its head and plays $\mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}$ and $\mathcal{F}_{\mathrm{P4TC}}$ for $\mathcal{Z}^{\text{sys-sec}}$ until $\mathcal{B}$ observes event ($D1$). Then $\mathcal{B}$ outputs $c_{\mathcal{R}}^{\mathrm{prev}}$ together with two different openings $M_{\mathcal{R}}^{\mathrm{prev}}, d_{\mathcal{R}}^{\mathrm{prev}}$ and $M_{\mathcal{R}}^{\mathrm{out}*}, d_{\mathcal{R}}^{\mathrm{out}*}$. □

Please note that for an honest RSU $\mathcal{R}$ and and a malicious user $\mathcal{U}$ the blacklist $bl_{\mathcal{R}}$ (input of $\mathcal{R}$) and fraud detection ID φ (part of the message from $\mathcal{U}$ to $\mathcal{R}$) are both given by $\mathcal{Z}^{\text{sys-sec}}$. Moreover, $\mathcal{Z}^{\text{sys-sec}}$ provides a proof that $\varphi = \mathsf{PRF}(\lambda, x)$ has been evaluated correctly. Hence, the only way how $\mathcal{Z}^{\text{sys-sec}}$ can achieve a discrepancy with respect to the abort behavior between the real and ideal experiment is to manipulate x.

Lemma E.34. *If* P2 *and* P3 *are perfectly sound,* C1 *is computationally binding and* S *is EUF-CMA secure, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathrm{P4TC}}, \overline{\mathcal{G}}_{\mathrm{bb}}, \mathcal{S}_{\pi_{\mathrm{P4TC}}}^{\text{sys-sec}}, \mathcal{Z}^{\text{sys-sec}}}(1^{n})$$

the event ($D2$) *(cp. Definition E.26) only occurs with negligible probability for any PPT environment $\mathcal{Z}^{\text{sys-sec}}$ that is restricted to static corruption of users.*

Proof. Identical to the proof of Lemma E.33 except for a different raise condition. □

LEMMA E.35. *If* P2 *is perfectly sound and* C1 *is computationally binding, then in a run of*

$$\mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{sys-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{sys-sec}}}(1^n)$$

the event (*D3*) *(cp. Definition E.26) only occurs with negligible probability for any PPT environment* $\mathcal{Z}^{\text{sys-sec}}$ *that is restricted to static corruption of users.*

PROOF. Please note that in the event of (*D3*), we already know that $\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathsf{com}}, \mathsf{pk}^{\mathsf{id}}_{\mathcal{U}}, c_{\mathsf{hid}}, d_{\mathsf{hid}}) = 1$ holds as otherwise $\mathcal{F}_{\text{P4TC}}$ would have aborted earlier. Let $\overline{\omega}^{\mathsf{pp}*}_{\mathcal{U}} = (s^*, c^*_{\mathsf{hid}}, d^*_{\mathsf{hid}}, \mathsf{pk}^{\mathsf{id}\,*}_{\mathcal{U}}) \in \overline{\Omega}^{\mathsf{pp}}_{\mathcal{U}}$ denote the recorded entry with $s^* = s$ and $c^*_{\mathsf{hid}} = c_{\mathsf{hid}}$. Obviously, $\mathsf{C1.Open}(\mathsf{CRS}^1_{\mathsf{com}}, \mathsf{pk}^{\mathsf{id}\,*}_{\mathcal{U}}, c^*_{\mathsf{hid}}, d^*_{\mathsf{hid}}) = 1$ holds, as otherwise the entry never would have been recorded. On the other hand, the user with $pid_{\mathcal{U}}$ and $\mathsf{pk}^{\mathsf{id}}_{\mathcal{U}}$ has never participated in a transaction with serial number s, because $\mathcal{F}_{\text{P4TC}}$ returned $\mathsf{out}_{\mathcal{U}} = 0$ to the simulator. Hence, $\mathsf{pk}^{\mathsf{id}}_{\mathcal{U}} \neq \mathsf{pk}^{\mathsf{id}\,*}_{\mathcal{U}}$ follows which is a contradiction to the binding property of C1. An reduction to an adversary $\mathcal{B}$ can be constructed the usual way. □

Taking all the aforementioned statements together, Theorem E.22 from the beginning of this section follows. For the sake of formal completeness we recall it again.

THEOREM E.22 (SYSTEM SECURITY). *Under the assumptions of Theorem E.1 and static corruption of a subset of the users*

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

holds.

PROOF. A direct consequence of Lemmas E.23, E.24 and E.27 to E.35. □

E.3 Proof of User Security and Privacy

In this section we show the following theorem.

THEOREM E.36 (USER SECURITY AND PRIVACY). *Under the assumptions of Theorem E.1 and static corruption of a subset of the RSUs, TSP and SA*

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

holds.

The definition of the UC-simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ for Theorem E.36 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 by $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$, giving it the 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_{11} is identical to the ideal experiment. Each hybrid has the form

$$\mathsf{H}_i := \mathsf{EXEC}_{\pi_i, \overline{\mathcal{G}}_{\text{bb}}, \mathcal{S}_i, \mathcal{Z}^{\text{user-sec}}}(1^n).$$

We show 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.

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.

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

Setup:

1. Run a modified version of the algorithm $\text{CRS} \leftarrow \mathtt{Setup}(1^n)$ with SetupPoK being replaced by SetupSPoK, C1.Gen being replaced by C1.SimGen, and C2.Gen being replaced by C2.ExtGen.
2. Record CRS, $\mathsf{td}_{\text{spok}}$, $\mathsf{td}_{\text{eqcom}}$ and $\mathsf{td}_{\text{extcom}}$.
3. Set $\Omega^{\text{dsp}} := \emptyset$.
4. Set $\Omega^{\text{pp}}_{\mathcal{U}} := \emptyset$.
5. Set $\mathit{AHTD} := \emptyset$.

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

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 $(\mathtt{registering\_tsp}, \mathit{pid}_{\mathcal{T}}, \mathsf{a}_{\mathcal{T}})$ run $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}) \leftarrow \mathtt{TSPRegistration}(\text{CRS}, \mathsf{a}_{\mathcal{T}})$, return $\mathsf{pk}_{\mathcal{T}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $\mathit{pid}_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$.

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 $(\mathtt{registering\_rsu}, \mathit{pid}_{\mathcal{R}})$ run $(\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}}) \leftarrow \mathtt{RSURegistration}(\text{CRS})$, return $\mathsf{pk}_{\mathcal{R}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $\mathit{pid}_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$.

User Registration: Upon receiving $(\mathtt{registering\_user}, \mathit{pid}_{\mathcal{U}})$ run $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \leftarrow \mathtt{UserRegistration}(\text{CRS})$, return $\mathsf{pk}_{\mathcal{U}}$ to $\mathcal{F}_{\text{P4TC}}$ and record $\mathit{pid}_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$.

RSU Certification: Distinguish four cases:

TSP and RSU honest: Upon receiving $(\mathtt{certifying\_rsu}, \mathit{pid}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}) \ldots$

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

TSP honest, RSU corrupted: Upon receiving $(\mathtt{certifying\_rsu}, \mathit{pid}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}) \ldots$

1. Load the recorded $\mathit{pid}_{\mathcal{T}} \mapsto (\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}})$, and obtain $\mathsf{pk}_{\mathcal{R}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathit{pid}_{\mathcal{R}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
2. Generate $\mathsf{cert}_{\mathcal{R}} := (\mathsf{pk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma^{\text{cert}}_{\mathcal{R}})$ with $\sigma^{\text{cert}}_{\mathcal{R}} \leftarrow \text{S.Sgn}(\mathsf{sk}^{\text{cert}}_{\mathcal{T}}, (\mathsf{pk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}))$ faithfully.
3. Record $\mathit{pid}_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \bot, \mathsf{cert}_{\mathcal{R}})$.
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 $\mathit{pid}_{\mathcal{T}} \ldots$

1. Load the recorded $\mathit{pid}_{\mathcal{R}} \mapsto (\mathsf{pk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$, and obtain $\mathsf{pk}_{\mathcal{T}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathit{pid}_{\mathcal{T}}$; if any of these do not exist, let $\mathcal{F}_{\text{P4TC}}$ abort.
2. Parse $\mathsf{a}_{\mathcal{R}}$ and $\sigma^{\text{cert}}_{\mathcal{R}}$ from $\mathsf{cert}_{\mathcal{R}}$.
3. If $\text{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}})) = 0$, let $\mathcal{F}_{\text{P4TC}}$ abort.
4. Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{certify}, \mathsf{a}_{\mathcal{R}})$ in the name of $\mathcal{T}$ with $\mathit{pid}_{\mathcal{T}}$.

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) Upon receiving $\mathbf{a}_{\mathcal{U}}, c''_{\text{ser}}, \mathsf{cert}^{\mathcal{R}}_{\mathcal{T}}, c''_{\mathcal{T}}$ …

  (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) $s'' \leftarrow \mathsf{C2.Extract}(\mathsf{CRS}^2_{\text{com}}, c''_{\text{ser}})$.

  (d) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{issue}, \mathbf{a}_{\mathcal{U}}, \emptyset)$[a] in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$.

  (e) Obtain TSP's output (s) 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) $r_0, \ldots, r_\ell, r^* \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$.

  (b) $e_i \leftarrow \mathsf{E.Enc}(\mathsf{pk}_{\mathcal{DR}}, 1; r_i)$ for $i = 0, \ldots, \ell$.

  (c) $e^* \leftarrow \mathsf{E.Enc}(\mathsf{pk}_{\mathcal{DR}}, 1; r^*)$.

  (d) $\sigma^* \leftarrow \mathsf{S.Sgn}(\mathsf{sk}^{\text{auth}}_{\mathcal{U}}, (c''_{\mathcal{T}}, e_0, \ldots, e_\ell, e^*))$.

  (e) $(c'_{\mathcal{T}}, d'_{\mathcal{T}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, 0)$.

  (f) $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\text{com}}, (0, 0, 0, 0))$.

  (g) $stmnt := (\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{DR}}, e^*, e_0, \ldots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$.

  (h) $\pi \leftarrow \mathsf{P1.SimProof}(\mathsf{CRS}_{\text{pok}}, \mathsf{td}_{\text{spok}}, stmnt)$.

  (i) Output $(\mathsf{pk}_{\mathcal{U}}, s', e^*, \sigma^*, \pi, e_0, \ldots, e_\ell, c'_{\mathcal{R}}, c'_{\mathcal{T}})$ to $\mathcal{Z}^{\text{user-sec}}$.

(4) Upon receiving $(s'', d''_{\text{ser}}, \lambda'', c_{\mathcal{R}}, d''_{\mathcal{R}}, \sigma_{\mathcal{R}}, c_{\mathcal{T}}, d''_{\mathcal{T}}, \sigma_{\mathcal{T}})$ from $\mathcal{Z}^{\text{user-sec}}$ in the name of $\mathcal{T}$ with $pid_{\mathcal{T}}$ …[b]

  (a) Set $htd := (\lambda'', c''_{\mathcal{T}}, d''_{\mathcal{T}}, e_0, \ldots, e_\ell, e, \sigma)$ and insert $(pid_{\mathcal{U}}, s, htd)$ into $AHTD$.

  (b) Create real token τ faithfully.

  (c) If $\mathsf{WalletVerification}(\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{U}}, \tau) = 0$, 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. 51. The simulator for User Security and Privacy (cont. from Fig. 50)

Hybrid H_1. In hybrid H_1 we modify $\mathcal{S}_1$ such that $\mathsf{CRS}_{\text{pok}}$ is generated by SetupSPoK, $\mathsf{CRS}^1_{\text{com}}$ is generated by C1.SimGen and $\mathsf{CRS}^2_{\text{com}}$ is generated by C2.ExtGen. Additionally, $\mathcal{S}_1$ initializes the internal sets $\Omega^{\text{dsp}} := \emptyset$, $\Omega^{\text{pp}}_{\mathcal{U}} := \emptyset$ and $AHTD := \emptyset$ and records the respective entries as the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ does.

Hybrid H_2. In hybrid H_2 we replace 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. This equals the method in which the keys are generated in the final ideal experiment.

Hybrid H_3. In hybrid H_3 the task RSU Certification is modified. For an honest $\mathcal{T}$ or an honest $\mathcal{R}$ the code of π_3 is replaced by the code of a dummy party. The simulator $\mathcal{S}_3$ proceeds as the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$ would do.

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 $(\mathsf{pk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}}, \sigma^{\text{cert}}_{\mathcal{R}}) := \mathsf{cert}_{\mathcal{R}}$.
    (b) If $\mathsf{S.Vfy}(\mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \sigma^{\text{cert}}_{\mathcal{R}}, (\mathsf{pk}_{\mathcal{R}}, \mathsf{a}_{\mathcal{R}})) = 0$ abort.
    (c) $s'' \leftarrow \mathsf{C2.Extract}(\mathsf{CRS}^{2}_{\text{com}}, c''_{\text{ser}})$.
    (d) Call $\mathcal{F}_{\text{P4TC}}$ with input $(\mathtt{pay\_toll}, \emptyset)$[a] in the name of $\mathcal{R}$ with $pid_{\mathcal{R}}$.
    (e) Obtain RSU's output $(s, \varphi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}_{\mathcal{R}})$ 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 $(c_{\text{hid}}, d^{\text{sim}}_{\text{hid}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{CRS}^{1}_{\text{com}})$ and append $(s, c_{\text{hid}}, d^{\text{sim}}_{\text{hid}})$ to $\Omega^{\text{pp}}_{\mathcal{U}}$.
    (b) $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^{1}_{\text{com}}, (0,0,0,0))$.
    (c) Check if any $(\varphi, t', u'_2) \in \Omega^{\text{dsp}}$ has been recorded previously with $(\varphi)$ being used as key. If no, pick
        $t \xleftarrow{R} \mathbb{Z}_p$. 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 $(\varphi, t, u_2)$
        into $\Omega^{\text{dsp}}$.
    (d) $stmnt := (\mathsf{pk}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \varphi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t, u_2)$.
    (e) $\pi \leftarrow \mathsf{P2.SimProof}(\mathsf{CRS}_{\text{pok}}, \mathsf{td}_{\text{spok}}, stmnt)$.
    (f) Output $(s', \pi, \varphi, \mathsf{a}_{\mathcal{U}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}}, c_{\text{hid}}, c'_{\mathcal{R}}, t)$ to $\mathcal{Z}^{\text{user-sec}}$.
(4) Upon being ask to provide a price for $(\mathsf{a}_{\mathcal{U}}, \mathsf{a}_{\mathcal{R}}, \mathsf{a}^{\text{prev}}_{\mathcal{R}})$ 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_4. Hybrid H_4 mostly modifies the task User Blacklisting. If the task Wallet Issuing is executed, π_4 still runs the real protocol as in π_{P4TC}, but the simulator $\mathcal{S}_4$ already records *htd* and inserts $(pid_{\mathcal{U}}, s, htd)$ into *AHTD*. Moreover, $\mathcal{S}_4$ additionally checks if the set of hidden trapdoors *HTD* provided by the environment is a subset of the hidden trapdoors that have actually been generated in the scope of Wallet Issuing for this particular user and thus have been recorded by the simulator (cp. Fig. 54, Steps 2b to 2d). This equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.[31]

[31]Note that these modifications only have an effect if $\mathcal{T}$ is malicious.

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 $(pid_{\mathcal{U}}, \varphi, b^{\text{bill}})$ 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 $(\varphi, t', u'_2) \in \Omega^{\text{dsp}}$ 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 Ω^{dsp}.
  (b) $stmnt := (\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{T}}, \mathsf{pk}^{\text{cert}}_{\mathcal{T}}, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, g_1^{b^{\text{bill}}}, t, u_2)$.
  (c) $\pi \leftarrow \mathsf{P3.SimProof}(\mathsf{CRS}_{\text{pok}}, \mathsf{td}_{\text{spok}}, stmnt)$.
  (d) Output $(\mathsf{pk}_{\mathcal{U}}, \pi, \varphi, \mathbf{a}_{\mathcal{U}}, \mathbf{a}^{\text{prev}}_{\mathcal{R}}, b^{\text{bill}}, t)$ 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)

Hybrid H_5. In H_5 the simulator $\mathcal{S}_5$ replaces all proofs in a message from a user to a TSP or RSU by a simulated proof.[32]

Hybrid H_6. In H_6 the simulator $\mathcal{S}_6$ prepares a pool of independently and uniformly drawn fraud detection IDs $\Phi_{pid_{\mathcal{U}}, s_{\text{root}}} := \{\varphi_1, \ldots, \varphi_{x_{\text{bl}_{\mathcal{R}}}}\}$ every time a new wallet with s_{root} being its root's serial number for a user with $pid_{\mathcal{U}}$ is issued. Moreover, $\mathcal{S}_6$ keeps a partial mapping f^{pseudo}_{Φ} that maps $(pid_{\mathcal{U}}, \varphi^{\text{real}})$-pairs to φ^{ideal}. In all messages that the simulator delivers in the name of a user with $pid_{\mathcal{U}}$ as the pretended sender the simulator substitutes "real" fraud detection IDs φ^{real} with "ideal" fraud detection IDs φ^{ideal} consistently. To this end, whenever the simulator forwards a message and encounters a fraud detection ID $\varphi = \varphi^{\text{real}}$, the simulator $\mathcal{S}_6$ first checks if $f^{\text{pseudo}}_{\Phi}(pid_{\mathcal{U}}, \varphi^{\text{real}})$ is already defined. If so, $\varphi^{\text{ideal}} = f^{\text{pseudo}}_{\Phi}(pid_{\mathcal{U}}, \varphi^{\text{real}})$ is taken as the substitute. If not, $\mathcal{S}_6$ picks an element $\varphi^{\text{ideal}} \in \Phi_{pid_{\mathcal{U}}}$ that is not yet used as an image for $f^{\text{pseudo}}_{\Phi}(pid_{\mathcal{U}}, \cdot)$ and sets $f^{\text{pseudo}}_{\Phi}(pid_{\mathcal{U}}, \varphi^{\text{real}}) := \varphi^{\text{ideal}}$. If User Blacklisting is invoked for the user with $pid_{\mathcal{U}}$ and the consistency check introduced in H_4 succeeds, the simulator $\mathcal{S}_6$ does not deliver the message *HTD* to *DR* but instead replies to $\mathcal{T}$ with $\Phi_{pid_{\mathcal{U}}}$ in the name of *DR* for every wallet whose root serial number s is an element of *AHTD*$'$.

Some remarks about H_6 are in order. In H_6 an honest user still runs almost the real protocol, i.e., an honest user sends fraud detection IDs that are generated by PRF for a seed that $\mathcal{S}_6$ does not know. In the step from H_5 to H_6 these IDs are replaced by uniformly drawn IDs as in the ideal model. However, $\mathcal{S}_6$ must replace the IDs consistently in case $\mathcal{Z}^{\text{user-sec}}$ lets a user re-use an old wallet. If $\mathcal{Z}^{\text{user-sec}}$ invokes the task Blacklist User the simulator $\mathcal{S}_6$ simply returns the prepared pool of fraud detection IDs. In particular, this implies that *DR* does not

[32]This modification only has an effect if $\mathcal{T}$ or $\mathcal{R}$ is malicious.

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 $(\texttt{out})$ from $\mathcal{F}_{\text{P4TC}}$.

  (c) If $\texttt{out} = \mathsf{NOK}$, abort.

  (d) Pick $\omega^{\text{pp}}_{\mathcal{U}} = (s, c_{\text{hid}}, d^{\text{sim}}_{\text{hid}})$ from $\Omega^{\text{pp}}_{\mathcal{U}}$ such that $s \in S^{\text{pp}}_{\mathcal{R}}$; if this does not exist abort.

  (e) Parse $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$ from $\mathsf{sk}_{\mathcal{U}}$.

  (f) Equivoke $d_{\text{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{CRS}^1_{\text{com}}, \mathsf{sk}^{\text{id}}_{\mathcal{U}}, c_{\text{hid}}, d^{\text{sim}}_{\text{hid}})$.

  (g) Output $(s, c_{\text{hid}}, d_{\text{hid}})$ 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 $pid_{\mathcal{U}} \mapsto (\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$, parse $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$ from $\mathsf{sk}_{\mathcal{U}}$ and return $\mathsf{sk}^{\text{id}}_{\mathcal{U}}$.

Guilt Verification: Upon being ask by $\mathcal{F}_{\text{P4TC}}$ to provide out for $(pid_{\mathcal{U}}, \pi)$ …

(1) Receive $\mathsf{pk}_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $pid_{\mathcal{U}}$ and parse $(\mathsf{pk}^{\text{id}}_{\mathcal{U}}, \mathsf{pk}^{\text{auth}}_{\mathcal{U}}) := \mathsf{pk}_{\mathcal{U}}$

(2) If $g_1^{\pi} = \mathsf{pk}^{\text{id}}_{\mathcal{U}}$, then return $\texttt{out} := \mathsf{OK}$, else $\texttt{out} := \mathsf{NOK}$ 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 $(\mathsf{pk}_{\mathcal{U}}, HTD)$ from $\mathcal{Z}^{\text{user-sec}}$ …

  (a) Obtain $pid_{\mathcal{U}}$ from $\overline{\mathcal{G}}_{\text{bb}}$ for $\mathsf{pk}_{\mathcal{U}}$.

  (b) $AHTD' := \{ahtd' \in AHTD \mid ahtd' = (pid'_{\mathcal{U}}, s', htd') \text{ with } pid'_{\mathcal{U}} = pid_{\mathcal{U}} \wedge htd' \in HTD\}$.

  (c) $HTD' := \{htd' \mid (\cdot, \cdot, htd') \in AHTD'\}$.

  (d) Assert that $HTD = HTD'$, else abort.

  (e) Call $\mathcal{F}_{\text{P4TC}}$ with $(\texttt{blacklist\_user}, pid_{\mathcal{U}})$.

  (f) Upon being asked by $\mathcal{F}_{\text{P4TC}}$ to provide S_{root}, return $S_{\text{root}} := \{s \mid (\cdot, s, \cdot) \in AHTD'\}$

  (g) Forward output of $\mathcal{F}_{\text{P4TC}}$ to $\mathcal{Z}^{\text{user-sec}}$.

Fig. 54. The simulator for User Security and Privacy (cont. from Fig. 50)

need to decrypt any hidden trapdoor *htd* which is a crucial point for hybrid H_8 in which the encryption of the Wallet ID is replaced by an encryption of zero.

Hybrid H_7. H_7 modifies the tasks of Wallet Issuing and Debt Accumulation. The code of π_7 for the user is modified such that it does not send s' but randomly picks s and sends it to $\mathcal{S}_7$. Then $\mathcal{S}_7$ extracts $s'' \leftarrow \mathsf{C2.Extract}(\mathsf{CRS}^2_{\text{com}}, c''_{\text{ser}})$, calculates $s' := s \cdot (s'')^{-1}$ and $\mathcal{S}_7$ inserts s' into the message from the user to the TSP or RSU respectively.

Hybrid H_8. In the scope of Wallet Issuing the simulator $\mathcal{S}_8$ replaces e^*, e_i for $i = 0, \ldots, \ell$ by encryption of zero and $c'_{\mathcal{T}}$, $c'_{\mathcal{R}}$ by commitments to zero. This equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.

Hybrid H_9. In H_9 in the scope of Debt Accumulation and Debt Clearance, the simulator $\mathcal{S}_9$ replaces t in the message from the user. If no $(\varphi, t', u'_2) \in \Omega^{\text{dsp}}$ has been recorded previously, $\mathcal{S}_9$ picks $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_p$, else $\mathcal{S}_9$ sets

$t := t' + \mathsf{sk}_{\mathcal{U}}(u_2 - u'_2)$. Finally, $\mathcal{S}_9$ inserts (φ, t, u_2) into Ω^{dsp}. This equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.

Hybrid H_{10}. Within the scope of Debt Accumulation the simulator $\mathcal{S}_{10}$ runs $(c'_{\mathcal{R}}, d'_{\mathcal{R}}) \leftarrow \mathsf{C1.Com}(\mathsf{CRS}^1_{\mathsf{com}}, (0,0,0,0))$ and replaces the original $c'_{\mathcal{R}}$ in the message from the user to the TSP.

Hybrid H_{11}. The hybrid H_{11} modifies Debt Accumulation and Prove Participation. In Debt Accumulation $\mathcal{S}_{11}$ runs $(c_{\mathsf{hid}}, d^{\mathsf{sim}}_{\mathsf{hid}}) \leftarrow \mathsf{C1.SimCom}(\mathsf{CRS}^1_{\mathsf{com}})$ and appends $(s, c_{\mathsf{hid}}, d^{\mathsf{sim}}_{\mathsf{hid}})$ to $\Omega^{\text{pp}}_{\mathcal{U}}$. In Prove Participation the code of $\mathcal{S}_{11}$ 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 sends $\mathtt{OK}$ or $\mathtt{NOK}$ resp. to $\mathcal{S}_{11}$. If $\mathcal{S}_{11}$ receives $\mathtt{OK}$ from the user, then it picks $\omega^{\text{pp}}_{\mathcal{U}} = (s, c_{\mathsf{hid}}, d^{\mathsf{sim}}_{\mathsf{hid}})$ from $\Omega^{\text{pp}}_{\mathcal{U}}$ such that $s \in S^{\text{pp}}_{\mathcal{R}}$. Furthermore it parses $(\mathsf{sk}^{\mathsf{id}}_{\mathcal{U}}, \mathsf{sk}^{\mathsf{auth}}_{\mathcal{U}}) := \mathsf{sk}_{\mathcal{U}}$, runs $d_{\mathsf{hid}} \leftarrow \mathsf{C1.Equiv}(\mathsf{CRS}^1_{\mathsf{com}}, \mathsf{sk}^{\mathsf{id}}_{\mathcal{U}}, c_{\mathsf{hid}}, d^{\mathsf{sim}}_{\mathsf{hid}})$ and sends $(s, c_{\mathsf{hid}}, d_{\mathsf{hid}})$ to $\mathcal{T}$. Again, this equals the behavior of the final simulator $\mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}$.

Please note, that the combinations of all modifications from H_0 to H_{11} yields

$$\begin{aligned}\mathsf{H}_{11} &= \mathsf{EXEC}_{\pi_{11}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}_{11}, \mathcal{Z}^{\text{user-sec}}}(1^n) \\ &= \mathsf{EXEC}_{\mathcal{F}_{\text{P4TC}}, \bar{\mathcal{G}}_{\text{bb}}, \mathcal{S}^{\text{user-sec}}_{\pi_{\text{P4TC}}}, \mathcal{Z}^{\text{user-sec}}}(1^n).\end{aligned}$$

We are now prepared to give the proof of Theorem E.36. We do not spell out the reductions in detail, but only give hints.

Proof of Theorem E.36.

From H_0 *to* H_1*:* This hop only changes how the CRS is created during the setup phase. This is indistinguishable for $\mathsf{CRS}_{\mathsf{pok}}$, $\mathsf{CRS}^1_{\mathsf{com}}$, and $\mathsf{CRS}^2_{\mathsf{com}}$ (see the composable zero-knowledge property of Definition A.4, the equivocality property and the extractability property of Definition A.6, resp., condition (a) each).

From H_1 *to* H_2*:* This hop does not change anything 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 code, but this has no effect on the view of $\mathcal{Z}^{\text{user-sec}}$.

From H_3 *to* H_4*:* This hop introduces an additional check. Assume for the sake of contraction that there is a $\mathcal{Z}^{\text{user-sec}}$ that can distinguish between H_3 and H_4 with non-negligible advantage. This can only happen if $\mathcal{Z}^{\text{user-sec}}$ triggers an abort in H_4 due to a failing check $HTD = \{htd \mid (pid_{\mathcal{U}}, s, htd) \in AHTD\}$ but DR does not abort in H_3. However, this immediately yields an efficient adversary against the EUF-CMA security of the signature scheme S or against the binding property of the commitment scheme $\mathsf{C2}$. The adversary can only trigger a difference if it gets the DR to evaluate the pseudo-random function on a seed λ in the real experiment that has not been issued as a legitimate wallet ID for the particular user and thus has no correspondent mapping in the ideal experiment.[33] As $\lambda = \lambda' + \lambda''$ holds, tampering with λ implies that λ' or λ'' must have been modified, too. The former is the user's share of the wallet ID and has been encrypted under the DR's public key and then signed by the user. The latter is the TSP's share of the wallet ID to which the TSP has committed itself as $c''_{\mathcal{T}}$ during Wallet Issuing and then has been signed by the user.

From H_4 *to* H_5*:* This game hop replaces the real proofs by simulated proofs. To this end, we have to consider a sequence of sub-hybrids—one for each of the different ZK proof systems $\mathsf{P1}$, $\mathsf{P2}$ and $\mathsf{P3}$. In the first sub-hybrid all proofs for $\mathsf{P1}$ are replaced by simulated proofs, in the second sub-hybrid all proofs for $\mathsf{P2}$ are replaced and finally all proofs for $\mathsf{P3}$. 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 $\mathcal{Z}^{\text{user-sec}}$

[33] Please further note that this "trivial" attack does not yield any practical benefit for a corrupted TSP in the real world. This attack implies that the TSP avoids to completely blacklist a user, but blacklists the user only on a subset of its wallet versions (or none). However, this would be a noticeable difference from what happens in the ideal experiment.

and plays the protocol and simulator for $\mathcal{Z}^{\text{user-sec}}$. All calls of the simulator to $\mathsf{P1.Prove}$ are forwarded by $\mathcal{B}$ to its own oracle in the external challenge game which is either $\mathsf{P1.Prove}$ or $\mathsf{P1.SimProof}$. $\mathcal{B}$ 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 pseudo-random fraud detection IDs are replaced by uniformly drawn IDs. Again, we proceed by a sequence of sub-hybrids. In each sub-hybrid the fraud detection IDs of one particular wallet 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 A.13. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the protocol and simulator for $\mathcal{Z}^{\text{user-sec}}$. Externally, $\mathcal{B}$ either interacts with an oracle that is either a true random oracle or a PRF for an unknown seed. Whenever $\mathcal{B}$ playing the simulator for $\mathcal{Z}^{\text{user-sec}}$ internally needs to draw a fraud detection ID for the particular wallet, $\mathcal{B}$ uses its external oracle. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs.

From H_6 *to* H_7*:* This hop does not change anything from the perspective of $\mathcal{Z}^{\text{user-sec}}$ as $\mathsf{C2}$ is perfectly extractable. The change is just syntactical to push the simulator closer to the final one.

From H_7 *to* H_8*:* In this hop the commitments $c'_{\mathcal{T}}$, $c'_{\mathcal{R}}$ and the encryptions of the user's share of the seed e^*, e_i for $i = 0, \dots, \ell$ are replaced with zero-messages for every user that participates in the system. To this end, the hop from H_7 to H_8 is further split into a sequence of sub-hybrids with each sub-hybrid replacing all the messages of only one particular (user) wallet. The sub-hybrids are ordered according to the invocations of Wallet Issuing. Assume $\mathcal{Z}^{\text{user-sec}}$ can distinguish between H_7 and H_8 with non-negligible advantage. This either yields an efficient adversary $\mathcal{B}$ against the IND-CPA security of the encryption E or one against the hiding property of $\mathsf{C1}$. We only sketch the proof for the IND-CPA security of E. Internally, $\mathcal{B}$ runs $\mathcal{Z}^{\text{user-sec}}$ and plays the role of all parties and the simulator for $\mathcal{Z}^{\text{user-sec}}$. Externally, $\mathcal{B}$ plays the IND-CPA 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 $\mathsf{pk}_{DR} := \mathsf{pk}^{C}$. $\mathcal{B}$ 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 "makes the difference". 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 zero for the remaining invocations of Wallet Issuing. $\mathcal{B}$ outputs whatever $\mathcal{Z}^{\text{user-sec}}$ outputs.

From H_8 *to* H_9*:* This hop is statistically identical. As long as no double-spending has occurred, 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_9 *to* H_{10}*:* This hop is indistinguishable by the same argument as from H_7 to H_8.

From H_{10} *to* H_{11}*:* In this hop the simulator $\mathcal{S}_{11}$ sends simulated commitments c_{hid} for the hidden user ID instead of commitments to the true values and later $\mathcal{S}_{11}$ equivokes these commitments to the correct $\mathsf{pk}_{\mathcal{U}}$ on demand, 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_{10} and H_{11}, then an efficient adversary $\mathcal{B}$ can be constructed against the hiding property and equivocability of $\mathsf{C1}$. □