PrETP: Privacy-Preserving Electronic Toll Pricing * (extended version)

Josep Balasch, Alfredo Rial, Carmela Troncoso,
Bart Preneel, Ingrid Verbauwhede
IBBT-K.U.Leuven, ESAT/COSIC,
Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium.
firstname.lastname@esat.kuleuven.be

Christophe Geuens
K.U.Leuven, ICRI,
Sint-Michielsstraat 6,
B-3000 Leuven, Belgium,
christophe.geuens@law.kuleuven.be

Abstract

Current Electronic Toll Pricing (ETP) implementations rely on on-board units sending fine-grained location data to the service provider. We present PrETP, a privacy-preserving ETP system in which on-board units can prove that they use genuine data and perform correct operations while disclosing the minimum amount of location data. PrETP employs a cryptographic protocol, Optimistic Payment, which we define in the ideal-world/real-world paradigm, construct, and prove secure under standard assumptions. We provide an efficient implementation of this construction and build an on-board unit on an embedded microcontroller which is, to the best of our knowledge, the first self-contained prototype that supports remote auditing. We thoroughly analyze our system from a security, legal and performance perspective and demonstrate that PrETP is suitable for low-cost commercial applications.

1 Introduction

Vehicular location-based technologies [36, 43] are viewed by governments as a perfect tool to support applications such as electronic toll collection, automated law enforcement, or collection of traffic statistics. In October 2009, the European Commission announced that the current flat road tax systems existing in the Member States will be substituted by an European Electronic Toll Service (EETS) [12, 19]. In the United States, there are also ongoing initiatives to introduce *Electronic Toll Pricing* (ETP), as for instance the Regional High Occupancy Toll Network of the California Metropolitan Transportation Commission [1].

ETP allows road taxes to be calculated depending on parameters such as the distance covered by a driver, the kind of road used, or the time of usage. This is beneficial both for citizens and governments. The former pay only for their actual road use, while the latter can improve road mobility by applying "congestion pricing". This strategy assigns prices to roads depending on their traffic density such that driving in congested roads implies a higher cost. This in turn will encourage users to change their route (or even avoid using their vehicles) thus reducing congestion. Moreover, ETP has also environmental benefits as it discourages driving hence reduces pollution.

ETP architectures proposed so far [1, 12, 19] require that vehicles are equipped with an on-board unit necessary for collecting location data. At the end of each tax period, the fee corresponding to those data

*This work extends the contents of the paper published at USENIX Security 2010 to include the security proof of Optimistic Payment. Please cite the original version of this paper: J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede. PrETP: Privacy-Preserving Electronic Toll Pricing. In *Proceedings of the 19th USENIX Security Symposium*, USENIX, pp. 63-78, 2010.

is computed either remotely [36, 43] or locally [45], and relayed to the service provider. In both cases the service provider needs to be convinced that the fees correspond to the actual road usage of the driver, and that they have been correctly calculated. The verification is straightforward in implementations in which all the location data is sent to the service provider, but this constitutes an inherent threat to users' privacy.

We propose PrETP, a privacy-preserving ETP system in which, without making impractical assumptions, on-board units i) compute the fee locally, and ii) prove to the service provider that they carry out correct computations while revealing the minimum amount of location data. PrETP employs a cryptographic protocol, Optimistic Payment (OP), in which on-board units send along with the final fee commitments to the locations and prices used in the fee computation. These commitments do not reveal information on the locations or prices to the service provider. Moreover, they ensure that drivers cannot claim that they were at any other position, nor used different prices, from the ones used to create the commitments. In order to check the veracity of the committed values, we rely on the service provider having access to a proof (e.g., a photograph taken by a road-side radar or a toll gate) that a car was at a specific point at a particular time, as previously suggested in [16, 39]. Upon being challenged with this proof, the on-board unit must respond with some information proving that the location point where it was spotted was correctly used in the calculation of the final fee. To this end, it opens the commitment containing this location, thus revealing *only* the location data and the price at the instant specified in the proof. This information suffices for the provider to verify that correct input data (location and price) was used to calculate the fee.

We formally define Optimistic Payment and propose a construction based on homomorphic commitments and signature schemes that allow for efficient zero-knowledge proofs of signature possession. We prove our construction secure under standard assumptions. Finally, we present a prototype implementation on an embedded platform, and demonstrate that the cryptographic overhead of Optimistic Payment is efficient enough to be practically deployed in commercial in-car devices. Further, the fact that on-board units carry out all operations without interaction with the driver makes our system ideal in terms of usability.

The rest of the paper is organized as follows: we describe our system models and the security properties we seek in Sect. 2. Sect. 3 presents a high level description of our construction. Our prototype implementation and its evaluation are presented in Sect. 4, and we discuss some practical issues in Sect. 5. We situate our work within the landscape of proposals for privacy-friendly vehicular applications in Sect. 6, and we conclude in Sect. 7. Finally, we define the concept of Optimistic Payment in Appendix A, and describe in detail our cryptographic construction in Appendix B.

2 System model

PrETP employs the architecture and technologies recommended at European level [12, 19], although it could be adapted to other systems, such as [1]. The system model, illustrated in Fig. 1 (left), comprises three entities: an On-Board Unit (OBU), a Toll Service Provider (TSP), and a Toll Charger (TC). The OBU is an electronic device installed in vehicles subscribed to an ETP service, and it is in charge of collecting GPS data and calculating the fee at the end of each tax period. The TSP is the entity that offers the ETP service. It is responsible for providing vehicles with OBUs and monitor their performance and integrity. Finally, the TC is the organization (either public or private) that levies tolls for the use of roads and defines the correct use of the system. In agreement with the TC, the TSP establishes prices for driving on each of the roads. Such pricing policy can depend on the type of road (e.g., highways vs. secondary roads), its traffic density, or the time of the day (e.g., rush hours vs. the middle of the night). Additionally, prices can also depend on attributes of the vehicle or of the driver (e.g., low-pollution vehicles, or discounts for retired people). For the sake of clarity, in this work we focus on the core functionality of PrETP, and defer the discussion of practical issues to Sect. 5.

When the vehicle is driving, the OBU calculates the subfees corresponding to the trajectories according

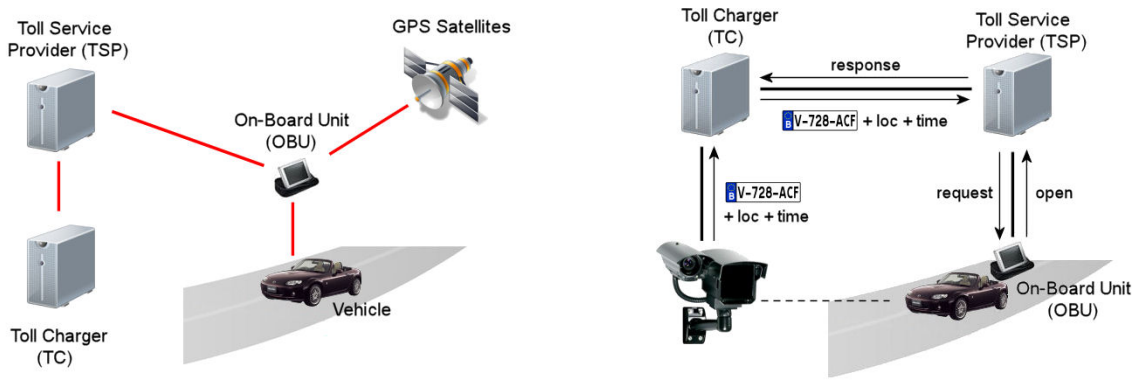

Figure 1: Entities in our Electronic Toll Pricing architecture (left.) Enforcement spot-check model (right.)

to the TSP pricing policy. At the end of each tax period, the OBU aggregates all the subfees to obtain a total fee and sends it to the TSP. This process safeguards the privacy of the driver from the TSP, the TC, or any other third party eavesdropping the communications, as no location data leaves the OBU. The privacy objectives of PrETP focus on the limitation of deliberate surveillance by any external party with limited access to the vehicle. We note that for an adversary with physical access to the vehicle it would be trivial to track it, e.g. by installing a tracking device. In order to further protect the privacy of users from adversaries that have occasional access to OBUs (e.g., mechanic, valet), all location data stored in the OBU is securely encrypted as specified in [45].

Besides preserving users' privacy, the system has to protect the interests of both TC and TSP and provide means to prevent users from committing fraud. Our threat model considers malicious drivers capable of tampering with the internal functionality of the OBU, as well as with any of its interfaces. Under these considerations, we define the security goals of our system as the detection of:

Vehicles with inactive OBUs. Drivers should not be able to shut down their OBUs at will to simulate they drove less.

OBUs reporting false GPS location data. Drivers should not be able to spoof the GPS signal and simulate a cheaper route than the actual roads on which they are driving.

OBUs using incorrect road prices. Drivers should not be able to assign arbitrary prices to the roads on which they are driving.

OBUs reporting false final fees. Drivers should not be able to report an arbitrary fee, but only the result from the correct calculations in the OBU.

Focusing on the detection of tampering rather that at its prevention allows us to consider a very simple OBU with no trusted components, reducing the production costs of the device.

In order to perform this detection, reliable information about the vehicle's whereabouts is required. We consider that the TC can perform random "spot checks" that are recorded as proof of the time and location where a vehicle has been seen. Such spot checks can be carried out by using an automatic license plate reader, a police control, or even challenging the OBUs using Dedicated Short-Range Communications (DSRC) [12]. Without loss of generality in this work we assume that the proof is gathered using an automatic license plate reader. This proof can be used to challenge the vehicle's OBU to verify its functioning. In order to be able to respond to this challenge, the OBU slices the trajectories recorded in segments, and computes the subfees corresponding to them, such that these subfees add up to the final fee transmitted to the TSP. For each segment, the TSP receives a payment tuple that consists of a commitment to location data and time, a homomorphic commitment to the subfee, and a proof that the committed subfee is computed according to the policy. These payment tuples, explained in detail in the next section, bind the reported final fee to the

committed values such that the OBU cannot claim having used other locations or prices in its computations. Furthermore, they are signed by the OBU to prevent a malicious TSP from framing an honest driver.

The verification process, depicted in Fig. 1 (right), is initiated when the TC gathers a proof of location of a vehicle. Then it forwards this information to the TSP, along with a request to check the correct functioning of the vehicle's OBU. To this end, the TSP challenges the OBU to open a commitment containing the location and time appearing in the proof. The TSP verifies that both challenge and response match, for instance as explained in [39], and reports to the TC whether or not the functioning of the OBU is correct. We assume that the TC (e.g., the government in the EETS architecture) is honest and does not use fake proofs to challenge OBUs.

3 Optimistic Payment

In this section we sketch the technical concepts necessary to understand the construction of Optimistic Payment, and we outline our efficient implementation of the protocol. For a comprehensive and more formal description of OP, we refer the reader to Appendix B.

3.1 Technical Preliminaries

Signature Schemes. A signature scheme consists of the algorithms $\mathsf{SigKeygen}$, $\mathsf{SigSign}$ and $\mathsf{SigVerify}$. $\mathsf{SigKeygen}$ outputs a secret key sk and a public key pk. $\mathsf{SigSign}(sk, x)$ outputs a signature s_x of message x. $\mathsf{SigVerify}(pk, x, s_x)$ outputs accept if s_x is a valid signature of x and reject otherwise. A signature scheme must be correct and unforgeable [25]. Informally speaking, correctness implies that the $\mathsf{SigVerify}$ algorithm always accepts an honestly generated signature. Unforgeability means that no p.p.t adversary should be able to output a message-signature pair (x, s_x) unless he has previously obtained a signature on x.

Commitment schemes. A non-interactive commitment scheme consists of the algorithms $\mathsf{ComSetup}$, Commit and Open. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme $params_{Com}$. $\mathsf{Commit}(params_{Com}, x)$ outputs a commitment c_x to x and auxiliary information $open_x$. A commitment is opened by revealing $(x, open_x)$ and checking whether $\mathsf{Open}(params_{Com}, c_x, x, open_x)$ is true. A commitment scheme has a hiding property and a binding property. Informally speaking, the hiding property ensures that a commitment c_x to x does not reveal any information about x, whereas the binding property ensures that c_x cannot be opened to another value x'. Given two commitments c_{x_1} and c_{x_2} with openings $(x_1, open_{x_1})$ and $(x_2, open_{x_2})$ respectively, the additively homomorphic property ensures that, if $c = c_{x_1} \cdot c_{x_2}$, then $\mathsf{Open}(params_{Com}, c, x_1 + x_2, open_{x_1} + open_{x_2})$.

Proofs of Knowledge. A zero-knowledge proof of knowledge is a two-party protocol between a prover and a verifier. The prover proves to the verifier knowledge of some secret values that fulfill some statement without disclosing the secret values to the verifier. For instance, let x be the secret key of a public key $y = g^x$, and let the prover know (x, g, y), while the verifier only knows (g, y). By means of a proof of knowledge, the prover can convince the verifier that he knows x such that $y = g^x$, without revealing any information about x.

3.2 Intuition Behind Our Construction

We consider a setting with the entities presented in Sect. 2. During each tax period tag, the OBU slices the trajectories of the driver in segments formed by a structure containing GPS location data and time. Additionally, this data structure can contain information about any other parameter that influences the price to be paid for driving on the segment. We represent this data structure as a tuple $(loc, time)$. The TSP establishes a function $f : (loc, time) \rightarrow \Upsilon$ that maps every possible tuple $(loc, time)$ to a price $p \in \Upsilon$. For

each segment, the OBU calculates f on input $(loc, time)$ to get a price p, and computes a payment tuple that consists of a randomized hash h on the data structure $(loc, time)$, a homomorphic commitment c_p to its price, and a proof π that the committed price belongs to Υ. The randomization of the hash is needed in order to prevent dictionary attacks to recover $(loc, time)$.

At the end of the tax period, the OBU and the TSP engage in a two-party protocol. The OBU adds the fees of all the segments to obtain a total fee fee. The OBU adds all the openings $open_p$ to obtain an opening $open_{fee}$. Next, the OBU composes a payment message m that consists of $(tag, fee, open_{fee})$ and all the payment tuples (h, c_p, π). The OBU signs m and sends both the message m and its signature s_m to the TSP. The TSP verifies the signature and, for each payment tuple, verifies the proof π. Then the TSP, by using the homomorphic property of the commitment scheme, adds the commitments c_p of all the payment tuples to obtain a commitment c'_{fee}, and checks that $(fee, open_{fee})$ is a valid opening for c'_{fee}.

When the TC sends the TSP a proof ϕ that a car was at some position at a given time, the TSP relays ϕ to the OBU. The OBU first verifies that the request is signed by the TC, and then it searches for a payment tuple (h, c_p, π) for which $\mu(\phi, (loc, time))$ outputs accept. Here, $\mu : (\phi, (loc, time)) \rightarrow \{\mathrm{accept}, \mathrm{reject}\}$ is a function established by the TSP that outputs accept when the information in ϕ and in $(loc, time)$ are similar in accordance with some metric, such as the one proposed in [39]. Once the payment tuple is found, the OBU sends the number of the tuple to the TSP together with the preimage $(loc, time)$ of h and the opening $(p, open_p)$ of c_p. The TSP checks that $(p, open_p)$ is the valid opening of c_p, that $(loc, time)$ is the preimage of h and that $\mu(\phi, (loc, time))$ outputs accept.

Intuitively, this protocol ensures the four security properties enunciated in the previous section. Drivers cannot shut down their OBUs, nor report false GPS data as they run the risk of not having committed to a segment containing the $(loc, time)$ in the challenge ϕ. We note that after sending (m, s_m) to the TSP, OBUs cannot claim that they were at any position $(loc', time')$ different from the ones used to compute the message m. Similarly, OBUs cannot use incorrect road prices without being detected, as the TSP can check whether the correct price for a segment $(loc, time)$ was used once the commitments are opened. The homomorphic property ensures that the reported final fee is not arbitrary, but the sum of all the committed subfees. Moreover, by making the OBU prove that the committed prices belong to the image of f, we avoid that a malicious OBU could decrease the final fee by sending only one wrong commitment to a negative price in the payment message, which would give it an overwhelming probability of not being detected by the spot checks. Additionally, the fact that the OBU signs the payment message m ensures that no malicious TSP can frame an OBU by modifying the received commitments, and that a malicious OBU cannot plead innocent by invoking the possibility of being framed by a malicious TSP. Similarly, the fact that the TC signs the challenge ϕ prevents a malicious TSP sending fake proofs to the OBU, e.g. with the aim of learning its location. Finally, the privacy of the drivers is preserved as the OBU does not need to disclose more location information than that in the payment tuple that matches the proof ϕ (already known to TSP).

3.3 Efficient Instantiation: High Level Specification

We now outline at high level our efficient instantiation of Optimistic Payment. We employ the integer commitment scheme due to Damgård and Fujisaki [14] and the CL-RSA signature scheme proposed by Camenisch and Lysyanskaya [8]. Both schemes use cryptographic keys based on special RSA modulus n of length l_n. A commitment c_x to a value x is computed as $c_x = {g_0}^x {g_1}^{open_x} \pmod n$, where the opening $open_x$ is a random number of length l_n and the bases (g_0, g_1) correspond to the commitment public parameters. Given a public key $pk = (n, R, S, Z)$, a CL-RSA signature has the form (A, e, v), with lengths l_n, l_e, and l_v respectively, such that $Z \equiv A^e R^x S^v (\mathrm{mod}\ n)$. To prove that a price belongs to Υ, we use a non-interactive proof of possession of a CL-RSA signature on the price. We also employ a collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{l_c}$.

Initialization. The pricing policy $f : (loc, time) \rightarrow \Upsilon$, where each price $p \in \Upsilon$ has associated a valid

	OBU Pay() algorithm		**TSP** VerifyPayment() algorithm	
1	**// Main loop**			1
2	*For all* $1 \leq k \leq N$ *tuples do:*			2
3	$p_k = f(loc_k, time_k)$			3
4	**// Hash computation**			4
5	$h_k = H((loc_k, time_k))$			5
6	**// Commitment computation**			6
7	$open_{p_k} \leftarrow \{0,1\}^{l_n}$			7
8	$c_{p_k} = {g_0}^{p_k} {g_1}^{open_{p_k}} \pmod n$			8
9	**// Proof computation**			9
10	$open_w, w \leftarrow \{0,1\}^{l_n}$			10
11	$\tilde{A} = A{g_0}^{w} \pmod n$			11
12	$c_w = {g_0}^{w} {g_1}^{open_w} \pmod n$		$\text{OBUverify}(pk_{\text{OBU}}, m, s_m)$	12
13	$r_\alpha \leftarrow \{0,1\}^{l_\alpha}$		**// Main loop**	13
14	$t_{c_{p_k}} = {g_0}^{r_{p_k}} {g_1}^{r_{open_{p_k}}}$	$\xrightarrow{(m, s_m)}$	*For all* $1 \leq k \leq N$ *tuples do:*	14
15	$t_Z = \tilde{A}^{r_e} R^{r_{p_k}} S^{r_v} (g_0^{-1})^{r_{w \cdot e}}$		$t'_{c_{p_k}} = c_{p_k}^{ch} {g_0}^{s_{p_k}} {g_1}^{s_{open_x}}$	15
16	$t_{c_w} = g_0^{r_w} g_1^{r_{open_w}}$		$t'_Z = Z^{ch} \tilde{A}^{s_e} R^{s_{p_k}} S^{s_v} (1/g_0)^{s_{w \cdot e}}$	16
17	$t = c_w^{r_e} (g_0^{-1})^{r_{w \cdot e}} (g_1^{-1})^{r_{open_w \cdot e}}$		$t'_{c_w} = c_w^{ch} {g_0}^{s_w} {g_1}^{s_{open_w}}$	17
18	$ch = H(\beta \vert\vert t_{c_{p_k}} \vert\vert t_Z \vert\vert t_{c_w} \vert\vert t)$		$t' = C_w^{s_e} (1/g_0)^{s_{w \cdot e}} (1/g_1)^{s_{open_{w \cdot e}}}$	18
19	$s_\alpha = r_\alpha - ch \cdot \alpha$		$ch' = H(\beta \vert\vert t'_{c_{p_k}} \vert\vert t'_Z \vert\vert t'_{c_w} \vert\vert t')? = ch$	19
20	$\pi_k = (\tilde{A}, c_w, ch, s_\alpha)$		$s_e \in \{0,1\}^{l_e + l_c + l_z}$	20
21	*End for*		$s_{p_k} \in \{0,1\}^{l_p + l_c + l_z}$	21
22	**// Fee reporting**		*End for*	22
23	$fee = \sum_{k=1}^{N} p_k$		**// Commitment validation**	23
24	$open_{fee} = \sum_{k=1}^{N} open_{p_k}$		$c'_{fee} = \prod_{k=1}^{N} c_{p_k}$	24
25	$m = [tag, fee, open_{fee}, (h_k, c_{p_k}, \pi_k)_{k=1}^{N}]$		$c_{fee} = {g_0}^{fee} {g_1}^{open_{fee}} \pmod n$	25
26	$s_m = \text{OBUsign}(sk_{\text{OBU}}, m)$		$c_{fee}? = c'_{fee}$	26
$\alpha \in \{p_k, open_{p_k}, e, v, w, open_w, w \cdot e, open_{w \cdot e}\}$ $\beta = (n \vert\vert g_0 \vert\vert g_1 \vert\vert \tilde{A} \vert\vert R \vert\vert S \vert\vert g_0^{-1} \vert\vert g_1^{-1} \vert\vert c_{p_k} \vert\vert Z \vert\vert c_w \vert\vert 1)$				

Protocol 1: Protocol between OBU and TSP during taxing phase

CL-RSA signature (A, e, v) generated by the TSP, the cryptographic key pair $(pk_{\text{OBU}}, sk_{\text{OBU}})$, the public key of the TSP (n, R, S, Z), the public key of TC, and the public parameters (g_0, g_1) of the commitment scheme are stored on the OBU. Similarly, the TSP possesses its own secret key (sk_{TSP}) and knows all the public keys in the system.

Tax period. Protocol 1 illustrates the calculations and interactions between the OBU and the TSP under normal functioning during the tax period. We denote the operations carried out by the OBU as Pay(), and the operations executed by the TSP as VerifyPayment(). While driving, the OBU collects location data and slices it in segments $(loc, time)$ according to the policy. For each of the N collected segments, the OBU generates a payment tuple (h_k, c_{p_k}, π_k). This iterative step is broken down in lines 1 to 21 in Protocol 1. The most resource consuming operation is the computation of π_k, which proves the possession of a valid CL-RSA signature on the price p_k (lines 9 to 20). The length of the random values used in this step is specified in Appendix B.2. At the end of the tax period the OBU generates and signs the payment message m including the tag tag, the total fee, the opening $open_{fee}$, and all the payment tuples (h_k, c_{p_k}, π_k), lines

22 to 26. Finally it sends (m, s_m) to the TSP.

Upon reception of a payment message, the TSP executes the VerifyPayment() algorithm. First the TSP verifies the signature s_m using the OBU's public key pk_{OBU}. Next, it proceeds to the verification of the proof π_k included in each of the N payment tuples contained in m, lines 13 to 22. In each iteration it performs a series of modular exponentiations, and uses the intermediate results to compute the hash ch'. Then, it checks whether ch' is the same as the value ch contained in π_k. If this verification, together with the two range proofs in lines 20 and 21, is successful, the TSP is convinced that all the prices p_k used by the OBU are indeed a valid image of f. Finally, the TSP validates the commitments c_{p_k} to ensure that the aggregation of all subfees add up to the final fee (lines 24 to 26). For this, it calculates c'_{fee} as the product of all commitments c_{p_k}, and computes the commitment c_{fee} using the values fee and $open_{fee}$ provided by the OBU. If both values are the same, the TSP is convinced that the final fee reported by the OBU adds up to the sum of all subfees reported in the payment tuples.

Proof Challenge. We denote as OBUopen() and Check() the algorithms carried out by the OBU and the TSP, respectively, when the former is challenged with ϕ. When running the OBUopen() algorithm, the OBU searches for the pre-image $(loc_k, time_k)$ of a hash h_k containing the location and time satisfying ϕ, and sends this information to the service provider along with the price p_k and the opening $open_{p_k}$.

Upon reception of this message, the TSP executes the Check() algorithm. First, it verifies whether the segment $(loc_k, time_k)$ actually contains the location in ϕ. Then, it computes the value $h'_k = H(loc_k, time_k)$ and checks whether the OBU had committed to this value in one of the payment tuples reported during the tax period. Lastly, the TSP uses $open_{p_k}$ to open the commitment c_{p_k} and verifies whether $p'_k = f(loc_k, time_k)$ equals the price p_k reported by the OBU during the OBUopen() algorithm. If all verifications succeed, the TSP is convinced that the location data used by the OBU in the fee calculation and the price assigned by the OBU to the segment $(loc_k, time_k)$ are correct.

4 PrETP Evaluation

In this section we evaluate the performance of PrETP. We start by describing the test scenario and both our OBU and TSP prototypes. Next, we analyze the performance of the prototypes for different configuration parameters. Finally, we study the communication overhead in PrETP, and compare it to existing ETP systems.

4.1 Test Scenario

Policy model. The first step in the implementation of PrETP consists in specifying a policy model in the form of the mapping function $f : (loc, time) \rightarrow \Upsilon$. We decide to follow the same criteria as currently existing ETP schemes [36], i.e., road prices are determined by two parameters: type of road and time of the day. More specifically, we define three categories of roads ('highway', 'primary', and 'others') and three time slots during the day. For each of the possible nine combinations we assign a price per kilometer p and we create a valid signature (A, e, v) using the TSP's secret key. We note that the choice of this policy is arbitrary and that PrETP, as well as OP, can accommodate other price strategies.

Location data. We provide the OBU with a set of location data describing a real trajectory of a vehicle . These data are obtained by driving with our prototype for one hour in an urban area, covering a total distance of 24 kilometers. We note that such dataset is sufficient to validate the performance of PrETP, since results for different driving scenarios (e.g., faster or slower) can easily be extrapolated from the results presented in this section.

Parameters of the instantiation. The performance of OP depends directly on the length of the protocol

instantiation parameters, and in particular, on the size of the cryptographic keys of the entities (l_n). In our experiments we consider three case studies: medium security (l_n = 1024 bits), high security (l_n = 1536 bits), and very high security (l_n = 2048 bits). The value l_p is determined by the length of the prices p, which in turn determines the value of l_e. Therefore, both lengths are constant for all security cases. The value of l_v varies depending on the value of l_n. Finally, the rest of parameters (l_h, l_r, l_z, and l_c) are set as the output length of the chosen hash function primitive (see Sect. 4.2). These lengths determine the size of the random numbers generated in line 13 in Protocol 1 (see Appendix B for a detailed explanation). Table 1 summarizes the parameter lengths considered for each security level.

Table 1: Length of the parameters (in bits)

Parameter	l_n	l_e	l_v	l_p	l_r,l_h,l_z,l_c
Normal Sec.	1 024	128	1 216	32	160
High Sec.	1 536	128	1 728	32	160
Very high Sec.	2 048	128	2 240	32	160

OBU Platform. In order to make our prototype as realistic as possible, we implement PrETP using as starting point the embedded design described in [3], which performs the conversion of raw GPS data into a final fee internally. We extend and adapt this prototype with the functionalities of OP to make it compatible with PrETP.

At high-level, the elements of our OBU prototype [3] are: a processing unit, a GPS receiver, a GSM modem, and an external memory module. We use as benchmark the Keil MCB2388 evaluation board [30], which contains an NXP LPC2388 [34] 32-bit ARM7TDMI [2] microcontroller. This microcontroller implements a RISC architecture, it runs at 72 MHz, and it offers 512 Kbytes of on-chip program memory and 98 Kbytes of internal SRAM. As external memory, we use an off-the-shelf 1 GByte SD Card connected to the microcontroller. Finally, we use the Telit GM862-GPS [44] as both GPS receiver and GSM modem.

As our platform does not contain any cryptographic coprocessors, we implement all functionalities exclusively in software. Note that although we could easily add a hardware coprocessor (e.g., [35]) to the prototype in order to carry out the most expensive cryptographic computations, we choose the option that minimizes the production costs of the OBU. Besides, this approach allows us to identify the bottlenecks in the protocol implementation, leaving the door open to hardware-based improvements if needed.

We have constructed a cryptographic library with the primitives required by our instantiation of the OP protocol, namely: i) a modular exponentiation technique, ii) a one-way hash function, and iii) a random number generator. For the first primitive we use the ACL [4] library, a collection of arithmetic and modular routines specially designed for ARM microcontrollers. As hash function we choose RIPEMD-160 [21], with an output length l_h of 160 bits. As our platform does not provide any physical random number generator, we use the Salsa20 [5] stream cipher in keystream mode as third primitive. We note that a commercial OBU should include a source of true randomness.

In order to keep the OBU flexible and easily scalable, we arrange data in different memory areas depending on their lifespan. Long-term parameters (pk_{OBU}, sk_{OBU}, pk_{TSP}, commitment parameters) are directly embedded into the microcontroller's program memory, while short-term parameters (payment tuples, $(loc, time)$ segments) and updatable parameters (digital road map, policy f) are stored separately on the SD Card. We note that our library provides a byte-oriented interface with the SD Card, resulting in a considerable overhead when reading/writing values.

TSP Platform. We implement our TSP prototype on a commodity computer equipped with an Intel Core2 Duo E8400 processor at 3 GHz, and 4 Gbyte of RAM. We use C as programming language, and the GMP [24] library for large-integer cryptographic operations.

4.2 Performance Evaluation

Table 2: Execution times (in seconds) for an hour journey of 24 km, for all possible security scenarios.

		Medium Security		High Security		Very high Security	
Algorithm		**Segment**	**Full trip**	**Segment**	**Full trip**	**Segment**	**Full trip**
Mapping ()		76.10 s	839.11 s	76.10 s	839.11 s	76.10 s	839.11 s
Pay ()		7.88 s	183.91 s	22.13 s	528.47 s	47.79 s	1 143.30 s
	h_k	0.08 s	1.08 s	0.08 s	1.08 s	0.08 s	1.08 s
	E_k	0.43 s	6.35 s	0.43 s	6.35 s	0.43 s	6.35 s
	c_{p_k}	0.76 s	18.19 s	2.25 s	54.08 s	5.69 s	136.82 s
	π_k	6.20 s	158.09 s	19.45 s	466.96 s	41.64 s	999.05 s

OBU performance. The most time-consuming operations carried out by the OBU during the taxing phase are the Mapping() algorithm and the Pay() algorithm. The Mapping() algorithm is executed every time a new GPS string is available in the microcontroller. Its function is to search in the digital road map the type of road given the GPS coordinates. When the vehicle drives for a kilometer, the OBU maps the segment to the adequate price p_k as specified in the policy. At this point, the Pay() algorithm is executed in order to create the payment tuple. For each segment, the OBU generates: i) a hash value h_k of the location data, ii) a commitment c_{p_k} to the price p_k, and iii) a proof π_k proving that the price p_k is genuinely signed by the TSP (and thus belongs to the image of f). To protect users' privacy we also require that no sensitive data is stored in the SD Card in plaintext form. For this purpose we use the AES [33] block cipher in CCM mode [22] with a key length of 128 bits. We denote this operation as E_k. At the end of the taxing phase, the OBU adds all the prices p_k mapped to each segment to obtain the fee, and all the openings $open_k$ to obtain $open_{fee}$. Finally, the OBU constructs and signs the payment message m and sends it to the TSP.

As it does not involve the key, the computing time of the Mapping() algorithm is independent of the security scenario. Further, this time only depends on the duration of the trip and is independent of the speed of the vehicle: the Mapping() algorithm is always executed 3 600 times per hour, taking a total of 839.11 seconds in our prototype. However, for each of the segments this time can vary depending on the number of points that have to be processed, i.e., depending on the speed of the vehicle. In our experiments it requires 76.10 seconds for the longest segment, i.e., the one where the vehicle spent more time to drive one kilometer and thus $(loc_k, time_k)$ contains the larger number of points.

Similarly, the execution time for h_k and E_k depends exclusively on the length of the segments $(loc_k, time_k)$, as it is proportional to the number of GPS points in the segments. The amount of points per segment varies not only with the average speed of the car but also depending on the length of the segments defined in the pricing policy. In our experiments, computing h_k and E_k take 0.08 seconds and 0.43 seconds, respectively, for the shortest and the longest segments. For the Mapping() algorithm and both h_k and E_k operations, more than 90% of the time is spent in the communication with the SD card.

On the other hand, the execution time for c_{p_k} and π_k is constant for all segments, as it does not depend on the length of a particular slice (see lines 6 to 20 in Protocol 1). In order to calculate c_{p_k}, the OBU needs to generate a random opening $open_{p_k}$ and perform two modular exponentiations and a modular multiplication. The computation of π_k involves the generation of ten random numbers and a hash value, and the execution of fourteen modular exponentiations, nine modular multiplications, eight additions, and eight multiplications. The bottleneck of both operations is determined by the modular operations. Although we could take advantage of fixed-base modular exponentiation techniques, we choose to use multi-exponentiations algorithms [17], which have less storage requirements. Multi-exponentiation based algorithms, which compute values of the form $a^b c^d (\text{mod } n)$ in one step, allow us to considerably speed up the process. The average execution times for computing c_{p_k} are 0.76 seconds, 2.25 seconds, and 5.69 seconds for medium, high, and

very high security respectively. For π_k, these times are 6.20 seconds, 19.45 seconds, and 41.64 seconds, respectively.

Table 2 summarizes the timings for all OBU operations and routines for a journey of one hour. We note that, even when 2048-bit RSA keys are used, the OBU can perform all operations needed to create the payment tuples in real time. While the trip lasted one hour, the Mapping() and Pay() algorithms only required $1\,982.41$ seconds. The computation time is dominated by the Pay() algorithm, which depends on the number of GPS strings in each segment $(loc, time)$. This number varies with the speed of the vehicle and the pricing policy. If a vehicle is driving at a constant speed, policies that establish prices for small distances result in segments containing less GPS points than policies that consider long distances. Similarly, given a policy fixing the size of the segments, driving faster produces segments with less points than driving slower. In both cases, π_k has to be computed fewer times and the Pay() algorithm runs faster. Thus, the policy can be used as tuning parameter to guarantee the real-time operation of the OBU.

Using the values in Table 2, for each of the levels of security we can calculate the time our OBU is idle – in our case $(3\,600 - 839.11)$ seconds, with 839.11 seconds being the time required by the Mapping() algorithm. Then, considering our current policy, we can estimate the number of times the Pay() algorithm could be executed, which in turn represents the number of kilometers that could have been driven by a car in one hour, i.e., the average speed of the car. For normal security, our OBU could operate in real time even if a vehicle was driving at 350 km/h. This speed decreases to 124 km/h when 1536-bit keys are used, and to 57 km/h if the keys have length 2048 bits. Only when using high security parameters our OBU would have problems to operate in the field. However, as mentioned before, including a cryptographic coprocessor in the platform would suffice to solve this problem whenever high security is required. Moreover, in our tests we consider a worst-case scenario in which all GPS strings are processed upon reception. In fact, processing fewer strings would suffice to determine the location of the vehicle. As the execution time required by the Mapping() algorithm would decrease linearly, OBUs would be able to support higher vehicle speeds.

In the OBUopen() algorithm, only executed upon request from TC, the OBU searches its memory for a segment $(loc, time)$ in accordance to the proof sent by the TSP. Here, the time accuracy provided by the GPS system is used to ensure synchronization between the data in ϕ and the segment $(loc, time)$. The main bottleneck of this operation is the decryption of the location data corresponding to the correct segment. On average, our prototype can decrypt such a segment in 0.27 seconds.

TSP performance. The most consuming task the TSP must perform corresponds to the VerifyPayment() algorithm, which has to be executed each time the TSP receives a payment message. This algorithm involves three operations: the verification of the proof π_k for each segment, the multiplication of all commitments c_{p_k} to obtain c_{fee}, and the opening of c_{fee} in order to check whether it corresponds to the reported final fee. The most costly operation is the verification of π_k, in particular the calculation of the parameters (t'_{c_m}, t'_Z, t'_{c_w}, t') which requires a total of eleven modular exponentiations (lines 14 to 22 in Protocol 1).

Table 4.2 (left) shows the performance of the VerifyPayment() algorithm for each of the considered security levels when segments have length one kilometer. We also provide an estimation of the time required to process all the proofs sent by OBU during a month, assuming that a vehicle drives an average of $18\,000$ km per year ($1\,500$ km per month).

These results allow us to extrapolate the number of OBUs that can be supported by a single TSP in each security scenario for different segment lengths. Intuitively, the capacity of TSP increases when segments are larger, as the payment messages contain fewer proofs π_k. The number of OBUs supported by a single TSP is presented in Table 4.2 (right). For a segment length of 1 km, the TSP is able to support $164\,000$, $58\,000$, and $29\,000$ vehicles depending on the chosen security level. Even when l_n is 2048 bits, only 36 servers are needed to accommodate one million OBUs. This number can be reduced by parallelizing tasks at the server side, or by using fast cryptographic hardware for the modular exponentiations.

Table 3: Timings (in seconds) for the execution of VerifyPayment() in TSP (left). Number of OBUs supported by a single TSP (right).

VerifyPayment()	Segment	One Month
Medium Sec.	0.0105 s	15.750 s
High Sec.	0.0295 s	44.250 s
Very high Sec.	0.0587 s	88.050 s

Segment size	Medium Sec.	High Sec.	Very high Sec.
0.5 km	82 000	29 000	14 000
0.75 km	123 000	43 000	22 000
1 km	164 000	58 000	29 000
2 km	329 000	117 000	58 000
3 km	493 000	175 000	88 000

4.3 Communication overhead

We now compare the communication overhead of PrETP with respect to straightforward ETP implementations and VPriv [39]. Both in straightforward ETP implementations and in VPriv the OBU sends all GPS strings to the TSP. Let us consider that vehicles drive 1 500 km per month at an average speed of 80 km/h. Then, transmitting the full GPS information to the the TSP requires 2.05 Mbyte (considering a shortened GPS string of 32 bytes containing only latitude, longitude, date and time). VPriv requires more bandwidth than straightforward ETP systems, as extra communications are necessary to carry out the interactive verification protocol (see Sect. 6). Using PrETP, the communication overhead comes from the payment tuples that must be sent along with the fee. For each segment, the OBU sends the payment tuple (h, c_p, π) to the TSP. When sent uncompressed, this implies an overhead of approximately 1.5 Kbyte per segment, i.e., less than 2 Mbyte per month, for medium security (l_n=1024 bits). Additionally, less than 50 Kbyte have to be sent occasionally to respond a verification challenge after a vehicle has been seen at a spot check. We believe this overhead is not excessive for the additional security and privacy properties offered by PrETP.

The communication overhead in PrETP is dominated by the payment message m sent by the OBU to the TSP, the length of which depends on the number of segments covered by the driver. Therefore, the segment length can be seen as a parameter of the system that tunes the tradeoff between privacy and communication overhead. The smaller the segments, the larger the communication overhead, because more tuples (h_k, c_{p_k}, π_k) need to be sent. Allowing larger segments reduces the communication cost but also reduces privacy because the OBU must disclose a bigger segment when responding a verification challenge.

Further, the communication overhead can be almost eliminated by having the OBU sending only the hash of the payment message at the end of each tax period and leave the correct operation verification subject to random checks. Following the spirit of the random "spot checks" used for checking the input and prices, the OBUs could occasionally be challenged to prove its correct functioning by sending the payment message corresponding to the preimage of the hash sent at the end of a random tax period.

5 Discussion

Practical issues. Our OP scheme allows the OBU to prove its correct operation to the TSP while revealing a minimum amount of information. Nevertheless, we note that fee calculation is not flexible. The reason is that the OBU should store signatures created by the TSP on all the prices that belong to $\mathrm{Im}(f)$, and thus, for the sake of efficiency, we need to keep $\mathrm{Im}(f)$ small. For this purpose, in our evaluation f is only defined for trajectory segments of a fixed length (one kilometer) and of a fixed road type. There are two obvious cases in which this feature is problematic: when a vehicle has driven a non-integer amount of kilometers, and when one of the segments contains pieces of roads with different cost (e.g., when a driver leaves the highway entering a secondary road). In both cases the OBU cannot produce a payment tuple because it does not have the signature by the TSP on the price of the segment.

There are two possible solutions to these issues. A first option would be to solve them at contractual level. The policy designed by the TSP could include clauses that indicate how to proceed when these conflicts arise. For instance, in the first case the TSP could dictate that the driver must pay for the whole kilometer, and in the second case the policy could be that the price corresponds to the cheapest of the roads, or to the most expensive. We note that these decisions do not conflict with the general purpose of the system: congestion control, as in all cases, on average, drivers will pay proportionally to their use of the roads. The second option would be to change the way the OBU proves that the committed prices belong to $\mathrm{Im}(f)$. In the construction proposed in Sect. 3, the OBU employs a set membership proof, based on proving signature possession, to prove that the committed prices belong to the finite set $\mathrm{Im}(f)$. Alternatively, we can define $\mathrm{Im}(f)$ as a range of (positive) prices, and let the OBU use a range proof to prove that the committed prices belong to $\mathrm{Im}(f)$. Since now $\mathrm{Im}(f)$ is much bigger, f can be defined for segments of arbitrary length that include several types of road. We outline a construction that employs range proofs in Appendix D.

Another issue is that our OP scheme does not offer protection against OBUs that do not reply upon receiving a verification challenge. In this case, the TSP should be able to demonstrate to the TC that the OBU is misbehaving. To permit this, the TSP can delegate to the TC the verification of the "spot-check", i.e, the TSP sends the payment message m and the signature s_m to the TC, and the TC interacts with the OBU (electronically, or by contacting the driver through some other means) to verify that m is valid.

Although in Sect. 2 we mentioned that the cost associated with roads could depend on attributes of the driver (e.g., retired users may get discounts) or on attributes of the car (e.g., ecological cars may have reduced fees), the pricing policy used by our prototype is rather inflexible. We note that this is a limitation of our prototype and that PrETP can support more flexible policies. For instance, the TSP can apply discounts to the total fee reported by the OBU, without the knowledge of fine grained location data. Further, the system model in this work considers only one service provider. However, the European legislation [12, 19] points out that several TSPs may provide services in a given Toll Charger domain. PrETP can be trivially extended to this setting.

Production cost. Our OBU prototype, constructed with off-the-shelf components, demonstrates that a system like PrETP can be built at a reasonable cost [1]. Although the security of our Optimistic Payment scheme does not rely on any countermeasure against physical attacks by drivers, for liability reasons it is desirable to use OBUs with a certain level of tamper resistance. Nevertheless, we note that on-board units in the market [36, 43] already rely on tamper resistance. Further, secure remote firmware updates are also required in privacy invasive designs, and additional updates in PrETP containing new maps and policies can be considered occasional.

Privacy. Although we protect the privacy of the users by keeping the location data in the client domain and exploiting the hiding property of cryptographic commitments, there exist a few sources of information available to the TSP. First, as in many other services, users in PrETP must subscribe to the service by revealing their identity, and most likely their home address, to the TSP. Second, the final fee and all the commitments (which indicate the number of kilometers driven), must be sent to the TSP at the end of each tax period. Decoding techniques (e.g., [15]) using these data could be employed by the TSP to infer the trajectories followed by a vehicle by inspecting the possible combination of prices per kilometers that could have generated the total fee. A possible solution to this problem consists in giving users the possibility to send data associated to dummy segments. For this, a price p zero should be included in the pricing policy so that it does not imply any cost for the drivers when aggregating the homomorphic commitments, and that the proofs π_k are still accepted by the TSP. The downside of this approach is that it introduces an overhead in both the processing of the OBU and the communication link with the TSP. Apart from this, subliminal channels in the communication or the encryption schemes must be avoided, e.g., by proving a true physical

[1] The cost of our prototype amounts to $500; such a number would be drastically reduced in a mass-production scenario.

randomness source in the OBU (see [45] for further discussion on the topic).

Legal Compliance. We build on the analysis presented in [45] and discuss the compliance of PrETP with European Legislation. With regard to data processing, the data controller (Art.6.2. in [12]) has to abide by principles found in the Data Protection Directive 95/46/EC [20] (DPD) in Art. 6.1, 16 and 17. We use these principles to assess compliance of the proposed architecture since these principles have been further specified in the other provisions of the DPD. We only look at the principles of direct interest for this paper which are that i) the data must be adequate, relevant and not excessive, ii) kept accurate and up to date, iii) the data should be processed in a secure and confidential manner and iv) data should not be kept longer than necessary. Firstly, data must be kept accurate and up-to-date (Art. 6.1(d) in [20]). In PrETP the OBU commits to location data and to its price when reporting the final fee. These commitments do not reveal any details on the location or the price calculation. Given that the controller is only allowed to process the data adequate, relevant and not excessive for the provision of the service (Art. 6.1(c) in [20]), this seems a good solution to the problem. The TC and the TSP should know that the information given by the user is correct but the information that the commitment covers is not needed for PrETP [28, 38]. The commitments implemented in PrETP are designed to guarantee that the OBU sends out the correct data without putting all the user's data in the hands of the TSP or the TC. The TC might want to execute checks at certain points in time to verify the veracity of these commitments and sends "spot-checks" to the TSP, which interacts with the OBU for the sake of verification. Only at those times will more data be disclosed because then it is required to know the information the commitment is based on to know whether the commitment is reliable. Data used for verification will however only be kept when an infringement is found. If there is no infringement, the data will not be kept in accordance with data protection principles (Art. 6.1(E) in [28, 38]). Secondly, the processing must be secure and confidential as stated by Art. 16-17 in [20]. A positive step of PrETP in this regard is keeping all the data inside the OBU and the applied algorithms to protect these data [28, 38]. The algorithms presented in this work are designed to reconcile the conflicting interest of the users and the TSP, while protecting the user from excessive data processing (note that the data set in road tolling could be potentially quite comprehensive – Art. 7, Annex VI in [12])). This criterion may be the most important in a road tolling setting.

6 Related work

A privacy-friendly architecture for ETP in which location data is not revealed to the service provider was presented in [45], and its viability was shown in [3]. However, the design by [45] does not take into account that the TSP and the TC need to check the correctness of the operations carried out in the on-board unit jeopardizing its applicability to real world scenarios.

Another line of research has focused on the design of secure multi-party protocols between the TSP and the OBUs that allow TSPs to compute the total fee and detect malicious OBUs while protecting location privacy. Solutions proposed in [7, 6, 41] resort to general reductions for secure multi-party computation and are very inefficient. A more efficient protocol, VPriv, was proposed in [39]. The basic idea consists in sending the location data generated by a driver sliced into segments to the TSP, in such a way that it remains hidden among segments from multiple drivers. Then the TSP calculates the subfees (fees of small time periods that add to the final fee) of all segments and returns them to all OBUs. Each OBU uses this information to compute its total fee and, without disclosing any location data, proves to the TSP that the total fee is computed correctly, i.e., by only using the subfees that correspond to the location data input by this particular OBU. Moreover, in order to prevent malicious users from spoofing the GPS signal to simulate cheaper trips, VPriv has an out-of-band enforcement mechanism. This mechanism is based on the use of random spot checks that demonstrate that a vehicle has been at a location at a time (e.g., a photograph taken by a road-side radar). Given this proof, the TSP challenges the OBU to prove that its fee calculation

includes the location where the vehicle was spotted.

The protocol proposed in [39] has several practical drawbacks. First, it requires vehicles to send anonymous messages to the server (e.g., by using Tor [18]) imposing high additional costs to the system. Second, their protocol only avoids leaking any additional information beyond what can be deduced from the anonymized database. As the database contains path segments, the TSP could use tracking algorithms to recover paths followed by the drivers [29, 27, 32] and infer further information about them. Third, the scalability of the system is limited by the complexity of the protocol on the client side, as it depends on the number of drivers in the system. Practical implementations require simplifications such as partitioning the set of vehicles into smaller groups, thus reducing the anonymity set of the drivers. Fourth, VPriv only uses spot checks to verify correctness of the location, and thus needs an extra protocol to verify the correct pricing of segments. This extra protocol produces an overhead both in terms of computation and communication complexity.

Our solution, similar to PriPAYD [45], does not require messages between the OBU and the TSP to be anonymous as the computation of the fee is made locally and no personal data is sent to the provider. Thus, no database of personal data is created and we do not need to rely on database anonymization techniques to ensure users' privacy. Further, the OBU's operations depend only on the data it collects, independently of the number of vehicles in the system. Finally, our protocol can be integrated into a stand-alone OBU without the need of external devices to carry out the cryptographic protocols.

To the best of our knowledge, the only protocol that so far employs spot checks to verify both correctness of the location and of the fee calculation is due to Jonge and Jacobs [16]. In this solution, OBUs commit to segments of location data and its corresponding subfees when reporting the total fee to the TSP. They employ hash functions as commitments. Upon being challenged to ratify the information in the spot check, OBUs must provide the hash pre-image of the corresponding segment, and demonstrate that indeed the location was used to compute the final fee.

Jonge and Jacobs' protocol is limited by the fact that using hash-based commitments one cannot prove that the commitments to the subfees add to the total fee. As solution, they propose that the OBU also commits to the subfees corresponding to bigger time intervals following a tree structure. Each tax period is divided into months, each month is divided into weeks, and so forth, and subfees for each month, week, day,... are calculated and committed. Then, instead of asking the OBU to open only one commitment containing the instant specified in TC's proof, the TSP asks the OBU to open all the commitments in the tree that include that instant. This indeed proves that the sum is correct at the cost of revealing much more information to the TSP.

PrETP avoids this information leakage. The reason is that, in our OP scheme, commitments are homomorphic and thus allow TSP to check that the commitments to the subfees add to the total fee without additional data. The use of homomorphic commitments was also proposed and briefly sketched in [16]. However, their scheme does not prevent the OBU from committing to a "negative" price, which would give a malicious OBU the possibility of reducing the final fee by sending only one wrong commitment, thus with an overwhelming probability of not being detected by the spot checks.

7 Conclusion

The revelation of location data in Electronic Toll Pricing (ETP) systems, besides conflicting with the users' right to privacy, can also pose inconveniences and extra investments to service providers as the law demands that personal data is stored and processed under strong security guarantees [20]. Furthermore, it has been shown [31] that security and privacy concerns are among the main reasons that discourage the use of electronic communication services. Recent research [46] demonstrates that users confronted to a prominent display of private information not only prefer service providers that offer better privacy guarantees but also

are willing to pay higher prices to utilize more privacy protective systems. Consequently, it is of interest for service providers to deploy systems where the amount of location information that users need to disclose is minimized.

As ETP systems are becoming increasingly important [12, 1], it is a challenge to implement them respecting both the users' privacy and the interest of the service provider. Previous work relied on too expensive solutions, or on unrealistic requirements, to fulfill both properties. In this work we have presented PrETP, an ETP system that allows on-board units to prove that they operate correctly leaking the minimum amount of information. Namely, upon request of the service provider, on-board units can attest that the input location data for the calculation of the fee is authentic and has not been tampered with. For this purpose we proposed a new cryptographic protocol, Optimistic Payment, that we define, construct and prove secure under standard assumptions. For this protocol, we also provide an efficient instantiation based on known secure cryptographic primitives.

We have performed a holistic analysis of PrETP. Besides the security analysis, we have built an on-board unit prototype on an embedded platform, as well as a service provider prototype on a commodity computer, and we have thoroughly tested the performance of both using real world collected data. The result of our experiments confirms that our protocol can be executed in real time in an on-board unit constructed with off-the-shelf components. Finally, we have analyzed the legal compliance of PrETP under the European Law framework and conclude that it fully supports the Data Protection Directive principles.

8 Acknowledgements

The authors want to thank M. Peeters and S. Motte for early valuable discussions, and G. Danezis and C. Diaz for their editorial suggestions that greatly improved the readability of the paper. We thank B. Gierlichs for driving us around to collect the data used in our experiments. C. Troncoso and A. Rial are research assistants of the Fund for Scientific Research in Flanders (FWO). This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by the Flemish IBBT NextGenITS project, by the European Commission under grant agreement ICT-2007-216676 ECRYPT NoE phase II, and by K.U. Leuven-BOF (OT/06/40). The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

[1] AB 744 (Torrico) Authorize a BayArea Express Lane Network to Deliver Congestion Relief and PublicTransit Funding with No NewTaxes, August 2009.

[2] ARM. ARM7TDMI technical reference manual, revision: r4p3. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0234b/DDI0234.pdf`, 2004.

[3] J. Balasch, I. Verbauwhede, and B. Preneel. An embedded platform for privacy-friendly road charging applications. In *Design, Automation and Test in Europe (DATE 2010)*, page 6, Dresden, 2010. IEEE.

[4] J. Ban. Cryptographic library for ARM7TDMI processors. Master's thesis, Technical University of Kosice, 2007.

[5] D. Bernstein. Salsa20. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/025, 2005.

[6] A. Blumberg and R. Chase. Congestion privacy that respects "driver privacy". In *Intelligent Transportation Systems Conference, ITSC*, pages 725 – 732, 2006.

[7] A. Blumberg, L. Keeler, and a. shelat. Automated traffic enforcement which respects driver privacy. In *Intelligent Transportation Systems Conference, ITSC*, pages 941– 946, 2004.

[8] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *In SCN 2002, volume 2576 of LNCS*, pages 268–289. Springer, 2002.

[9] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zürich, March 1997.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[11] D. Chaum and T. Pedersen. Wallet databases with observers. In *CRYPTO '92*, volume 740 of *LNCS*, pages 89–105, 1993.

[12] Comission Decission of 6 October 2009 on the definition of the European Electronic Toll Service and its technical elements, 2009.

[13] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[14] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Y. Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.

[15] G. Danezis and C. Diaz. Space-efficient private search with applications to rateless codes. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2007.

[16] W. de Jonge and B. Jacobs. Privacy-friendly electronic traffic pricing via commits. In P. Degano, J. Guttman, and F. Martinelli, editors, *Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2008.

[17] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and fast algorithms for multiexponentiations. *IEEE Transactions on Computers*, 49(2), 2000.

[18] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[19] Directive 2004/52/EC of the European Parliament and of the Council of 29 April 2004 on the interoperability of electronic road toll systems in the Community, 2004.

[20] Directive 95/46/EC of the European parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, 1995.

[21] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *FSE*, volume 1039 of *LNCS*, pages 71–82. Springer, 1996.

[22] Morris Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. NIST special publication 800-38c, National Institute for Standards and Technology, 2004.

[23] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[24] GMP. The GNU Multi-precision Library. `http://gmplib.org/`.

[25] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[26] J. Groth. Non-interactive zero-knowledge arguments for voting. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 467–482, 2005.

[27] M. Gruteser and B. Hoh. On the anonymity of periodic location samples. In D. Hutter and M. Ullmann, editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2005.

[28] J. H. Hoepman. Follow that car! over de mogelijke privacy gevolgen van rekeningrijden, en hoe die te vermijden. *Privacy & Informatie*, 5(11):225–230, 2008.

[29] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing security and privacy in traffic-monitoring systems. *IEEE Pervasive Computing*, 5(4):38–46, 2006.

[30] Keil. MCB2300 Evaluation Board Family.

[31] P. Koargonkar and L. Wolin. A multivariate analysis of web usage. *Journal of Advertising Research*, pages 53–68, March/April 1999.

[32] J. Krumm. Inference attacks on location tracks. In A. LaMarca, M. Langheinrich, and K. Truong, editors, *Pervasive*, volume 4480 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2007.

[33] NIST. *Advanced Encryption Standard (AES) (FIPS PUB 197)*. National Institute of Standards and Technology, November 2001.

[34] NXP Semiconductors. LPC23xx User Manual.

[35] NXP Semiconductors. SmartMX P5xC012/020/024/037/052 family. Secure contact PKI smart card controller.

[36] Octo Telematics S.p.A. `http://www.octotelematics.com/`.

[37] T. Okamoto. An efficient divisible electronic cash scheme. In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 1995.

[38] International Working Group on Data Protection in Telecommunications. Report and Guidance on Road Pricing, "Sofia Memorandum".

[39] R. Popa, H. Balakrishnan, and A. Blumberg. VPriv: Protecting privacy in location-based vehicular services. In *Proceedings of the 18th Usenix Security Symposium*, August 2009.

[40] M. O. Rabin and J. O. Shallit. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics*, 39(S1):239–256, 1986.

[41] S. Rass, S. Fuchs, M. Schaffer, and K. Kyamakya. How to protect privacy in floating car data systems. In V. Sadekar, P. Santi, Y. Hu, and M. Mauve, editors, *Vehicular Ad Hoc Networks*, pages 17–22. ACM, 2008.

[42] C. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[43] STOK Nederland BV. `http://www.stok-nederland.nl/`.

[44] Telit. GM862-GPS Hardware User Guide.

[45] C. Troncoso, G. Danezis, E. Kosta, and B. Preneel. PriPAYD: privacy friendly pay-as-you-drive insurance. In Peng Ning and Ting Yu, editors, *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007*, pages 99–107. ACM, 2007.

[46] J. Tsai, S. Egelman, L. Cranor, and A. Acquisti. The effect of online privacy information on purchasing behavior: An experimental study, working paper. In *The 6th Workshop on the Economics of Information Security*, 2007.

A Security Definition of Optimistic Payment

Ideal-world/real-world paradigm. We use the ideal-world/real-world paradigm to prove our construction secure. In this paradigm, parties are modeled as probabilistic polynomial time interactive Turing machines. A protocol ψ is secure if there exists no environment $\mathcal{Z}$ that can distinguish whether it is interacting with adversary $\mathcal{A}$ and parties running protocol ψ or with the ideal process for carrying out the desired task, where ideal adversary $\mathcal{S}$ and dummy parties interact with an ideal functionality $\mathcal{F}_\psi$. More formally, we say that protocol ψ emulates the ideal process if, for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the ensembles $\mathrm{IDEAL}_{\mathcal{F}_\psi,\mathcal{S},\mathcal{Z}}$ and $\mathrm{REAL}_{\psi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. We refer to [10] for a description of these ensembles.

Our construction operates in the $\mathcal{F}_{\mathrm{REG}}$-hybrid model, where parties register their public keys at a trusted registration entity and obtain from it a common reference string. Below we depict the ideal functionality

$\mathcal{F}_{\text{REG}}$, which is parameterized with a set of participants $\mathcal{P}$ that is restricted to contain OBU, TSP and TC only. We also describe an ideal functionality $\mathcal{F}_{\text{OP}}$ for Optimistic Payment. Every functionality and every protocol invocation should be instantiated with a unique session-ID that distinguishes it from other instantiations. For the sake of ease of notation, we omit session-IDs from our description.

Functionality $\mathcal{F}_{\text{REG}}$. Parameterized with a set of parties $\mathcal{P}$, $\mathcal{F}_{\text{REG}}$ works as follows:

- On input (crs) from party P, if $P \notin \mathcal{P}$ it aborts. Otherwise, if there is no value r recorded, it picks $r \leftarrow D$ and records r. It sends (crs, r) to P.
- Upon receiving (register, v) from party $P \in \mathcal{P}$, it records the value (P, v).
- Upon receiving (retrieve, P) from party $P' \in \mathcal{P}$, if (P, v) is recorded then return (retrieve, P, v) to P'. Otherwise send (retrieve, $P, \perp$) to P'.

Functionality $\mathcal{F}_{\text{OP}}$. Running with OBU, TSP and TC, $\mathcal{F}_{\text{OP}}$ works as follows:

- On input a message (initialize, f, μ) from TSP, where f is a mapping $f : (loc, time) \rightarrow \Upsilon$ and $\mu : (\phi, (loc, time)) \rightarrow \{\text{accept}, \text{reject}\}$, $\mathcal{F}_{\text{OP}}$ stores (f, μ) and sends (initialize, f, μ) to OBU.
- On input a message (payment, $tag, fee, (k, (loc_k, time_k), p_k)_{k=1}^{N}$) from OBU, where tag identifies the tax period, $\mathcal{F}_{\text{OP}}$ checks that a message (payment, $tag, \ldots$) was not received before, that for $k = 1$ to N, $p_k \in \Upsilon$, and that $fee = \sum_{k=1}^{N} p_k$. If these checks succeed, $\mathcal{F}_{\text{OP}}$ sends (payment, tag, fee, N) to TSP and stores the tuple $(tag, fee, (k, (loc_k, time_k), p_k)_{k=1}^{N})$. Otherwise $\mathcal{F}_{\text{OP}}$ sends (payment, $tag, \perp$) and stores $(tag, \perp)$.
- On input a message (proof, tag, ϕ) from TC, $\mathcal{F}_{\text{OP}}$ stores (tag, ϕ) and sends (proof, tag, ϕ) to TSP.
- On input a message (verify, tag, ϕ) from TSP, $\mathcal{F}_{\text{OP}}$ checks that it stores messages (payment, $tag, \ldots$) and (proof, tag, ϕ). If it is the case, $\mathcal{F}_{\text{OP}}$ sends (verifyreq, tag, ϕ) to OBU. Upon receiving (verifyresp, $tag, (\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$), $\mathcal{F}_{\text{OP}}$ checks whether the stored payment tuple $(k, (loc_k, time_k), p_k)$ equals $(\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$ for $k = \sigma$, whether $\mu(\phi, (loc'_\sigma, time'_\sigma))$ outputs accept, and whether $p'_\sigma = f(loc'_\sigma, time'_\sigma)$. If these checks are correct, $\mathcal{F}_{\text{OP}}$ sends (verifyresul, *not guilty*, $(\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$) to TSP. Otherwise it sends (verifyresul, *guilty*, $(\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$).
- On input a message (blame, tag) from TSP, $\mathcal{F}_{\text{OP}}$ checks that messages (payment, $tag, \ldots$), (proof, tag, ϕ) and (verifyresp, $tag, \ldots$) were previously received, and in this case it proceeds with the same checks done for (verify, $\ldots$). It sends to TC either (*guilty*) or (*not guilty*).

B Construction of an Optimistic Payment Scheme

We use several existing results to prove statements about discrete logarithms: (1) proof of knowledge of a discrete logarithm modulo a prime [42]; (2) proof of knowledge of the equality of some element in different representations [11]; (3) proof with interval checks [37] and (4) proof of the disjunction or conjunction of any two of the previous [13]. These results are often given in the form of Σ-protocols but they can be turned into non-interactive zero-knowledge arguments in the random oracle model via the Fiat-Shamir heuristic [23].

When referring to the proofs above, we follow the notation introduced by Camenisch and Stadler [9] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. $\mathsf{NIPK}\{(\alpha, \beta, \delta) : y = {g_0}^{\alpha} {g_1}^{\beta} \wedge \tilde{y} = {\tilde{g_0}}^{\alpha} {\tilde{g_1}}^{\delta} \wedge A \leq \alpha \leq B\}$ denotes a "*zero-knowledge Proof of Knowledge of integers α, β, and δ such that $y = {g_0}^{\alpha} {g_1}^{\beta}$, $\tilde{y} = {\tilde{g_0}}^{\alpha} {\tilde{g_1}}^{\delta}$ and $A \leq \alpha \leq B$ holds*", where $y, g_0, g_1, \tilde{y}, \tilde{g_0}$, and $\tilde{g_1}$ are elements of some groups $G = \langle g_0 \rangle = \langle g_1 \rangle$ and $\tilde{G} = \langle \tilde{g_0} \rangle = \langle \tilde{g_1} \rangle$ that have the same order. (Note that some elements in the representation of y and $\tilde{y}$ are equal.) The convention is that letters in the parenthesis, in this example α, β, and δ, denote quantities whose knowledge is being proven, while all other values are known to the verifier. We denote a non-interactive proof of signature possession as $\mathsf{NIPK}\{(x, s_x) : \mathsf{SigVerify}(pk, x, s_x) = \text{accept}\}$.

B.1 Construction

We begin with a high level description of the optimistic payment scheme. We assume that each party registers its public key at $\mathcal{F}_{\text{REG}}$, and retrieves public keys from other parties by querying $\mathcal{F}_{\text{REG}}$. They also retrieve the common reference string $params_{Com}$, which is computed by algorithm $\mathsf{SetupOP}$.

Optimistic Payment

When TSP is activated with (initialize, f, μ), TSP runs $\mathsf{TSPkg}(1^k)$ to obtain $(sk_{\text{TSP}}, pk_{\text{TSP}})$, and obtains a setup $params$ with $\mathsf{TSPinit}(f, sk_{\text{TSP}})$. TSP stores $\text{TSP}_0 = (f, \mu, sk_{\text{TSP}}, pk_{\text{TSP}}, params_{Com}, params)$ and sends $(f, \mu, params)$ to OBU. OBU runs $\mathsf{OBUkg}(1^k)$ to get $(sk_{\text{OBU}}, pk_{\text{OBU}})$ and executes $\mathsf{OBUinit}(params, pk_{\text{TSP}})$ to get a bit b. If $b = 0$, OBU rejects $params$. Otherwise OBU stores the tuple $\text{OBU}_0 = (f, \mu, sk_{\text{OBU}}, pk_{\text{OBU}}, pk_{\text{TSP}}, params_{Com}, params)$.

When OBU is activated with (payment, $tag, fee, (k, (loc_k, time_k), p_k)_{k=1}^N$) and OBU has previously received $(f, \mu, params)$, OBU runs algorithm $\mathsf{Pay}(params_{Com}, params, pk_{\text{OBU}}, sk_{\text{OBU}}, pk_{\text{TSP}}, tag, fee, (k, (loc_k, time_k), p_k)_{k=1}^N)$ to obtain a payment message m along with a signature s_m, and auxiliary information aux. OBU sets $aux = (aux, (k, (loc_k, time_k), p_k)_{k=1}^N)$, stores $\text{OBU}_{tag} = (\text{OBU}_0, m, s_m, aux)$ and sends (m, s_m) to TSP. TSP runs $\mathsf{VerifyPayment}(params_{Com}, pk_{\text{OBU}}, pk_{\text{TSP}}, m, s_m)$ to obtain a bit b. If $b = 0$, TSP rejects (m, s_m). Otherwise TSP stores $\text{TSP}_{tag} = (\text{TSP}_0, m, s_m, pk_{\text{OBU}})$.

When TC is activated with (proof, tag, ϕ), TC runs $\mathsf{TCkg}(1^k)$ to get $(pk_{\text{TC}}, sk_{\text{TC}})$, runs $\mathsf{Prove}(sk_{\text{TC}}, tag, \phi)$ to obtain a proof Q and sends (Q) to TSP. TSP runs $\mathsf{VerifyProof}(pk_{\text{TC}}, Q)$ and aborts if $b = 0$. Otherwise TSP stores $\text{TSP}_{tag} = (\text{TSP}_{tag}, Q)$.

When TSP is activated with (verify, tag, ϕ), and TSP has previously obtained (m, s_m) and (Q), TSP sends (Q) to OBU. OBU executes $\mathsf{VerifyProof}(pk_{\text{TC}}, Q)$ and aborts if $b = 0$. Otherwise OBU runs $\mathsf{OBUopen}(sk_{\text{OBU}}, Q, aux)$ to get a response R and sends (R) to TSP. TSP runs $\mathsf{Check}(params_{Com}, pk_{\text{OBU}}, pk_{\text{TSP}}, m, s_m, Q, R)$ to obtain either (*not guilty*, $(k, (loc_k, time_k), p_k)$ or (*guilty*, $(k, (loc_k, time_k), p_k)$).

When TSP is activated with (blame, tag), and messages (m, s_m), (Q) and (R) were previously received, TSP sends $((m, s_m), R)$ to TC. TC runs $\mathsf{Check}(params_{Com}, pk_{\text{OBU}}, pk_{\text{TSP}}, m, s_m, Q, R)$ to obtain (*not guilty*, $(k, (loc_k, time_k), p_k)$) or (*guilty*, $(k, (loc_k, time_k), p_k)$).

In the following, we denote the signature algorithms used by TSP, OBU and TC as (TSPkeygen, TSPsign, TSPverify), (OBUkeygen, OBUsign, OBUverify) and (TCkeygen, TCsign, TCverify). H stands for a collision-resistant hash function, which is modeled as a random oracle.

$\mathsf{SetupOP}(1^k)$**.** Run $\mathsf{ComSetup}(1^k)$ and output $params_{Com}$.

$\mathsf{TSPkg}(1^k)$**.** Run $\text{TSPkeygen}(1^k)$ to get a key pair $(pk_{\text{TSP}}, sk_{\text{TSP}})$. Output $(pk_{\text{TSP}}, sk_{\text{TSP}})$.

$\mathsf{OBUkg}(1^k)$**.** Run $\text{OBUkeygen}(1^k)$ to get a key pair $(pk_{\text{OBU}}, sk_{\text{OBU}})$. Output $(pk_{\text{OBU}}, sk_{\text{OBU}})$.

$\mathsf{TCkg}(1^k)$**.** Run $\text{TCkeygen}(1^k)$ to obtain a key pair $(pk_{\text{TC}}, sk_{\text{TC}})$. Output $(pk_{\text{TC}}, sk_{\text{TC}})$.

$\mathsf{TSPinit}(f, sk_{\text{TSP}})$**.** For all possible prices $p \in \Upsilon$, run $s = \text{TSPsign}(sk_{\text{TSP}}, p)$ and output the set $params = (p, s)$.

$\mathsf{OBUinit}(params, pk_{\text{TSP}})$**.** Parse $params$ as (p, s) and run $\text{TSPverify}(pk_{\text{TSP}}, p, s)$ for all $p \in \Upsilon$. If all the signatures are correct, output $b = 1$ else $b = 0$.

$\mathsf{Pay}(params_{Com}, params, pk_{\text{OBU}}, sk_{\text{OBU}}, pk_{\text{TSP}}, tag\ \ fee, (k, (loc_k, time_k), p_k)_{k=1}^N)$. For $k = 1$ to N, execute $h_k = H(loc_k, time_k)$, calculate a commitment to the price $(c_k, open_k) = \mathsf{Commit}(params_{Com}, p_k)$ and compute a proof of possession of a signature on the price $\pi_k = \mathsf{NIPK}\{(p_k, open_k, s_k) : \text{TSPverify}(pk_{\text{TSP}}, p_k, s_k) = \text{accept} \wedge (c_k, open_k) = \mathsf{Commit}(params_{Com}, p_k)\}$. Add all the prices to obtain the total fee fee and all the openings $open_k$ to get an opening $open_{fee}$ to the commitment to the fee. Set payment message $m = (tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$ and run $s_m = \text{OBUsign}(sk_{\text{OBU}}, m)$. Output (m, s_m) and $aux = (open_k)_{k=1}^N$.

VerifyPayment$(params_{Com}, pk_{\mathrm{OBU}}, pk_{\mathrm{TSP}}, m, s_m)$**.** Parse m as $(tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$. For $k = 1$ to N, verify π_k. Add all the commitments to obtain a commitment to the total fee c_{fee}, and run Open$(params_{Com}, c_{fee}, fee, open_{fee})$. If the opening is correct, output $b = 1$. Otherwise output $b = 0$.

Prove$(sk_{\mathrm{TC}}, tag, \phi)$**.** Set $q = (tag, \phi)$ and run $s_q = \mathrm{TCsign}(sk_{\mathrm{TC}}, q)$. Output $Q = (q, s_q)$.

VerifyProof(pk_{TC}, Q)**.** Parse Q as (q, s_q) and run TCverify$(pk_{\mathrm{TC}}, q, s_q)$. Output $b = 1$ if the signature is correct and $b = 0$ otherwise.

OBUopen$(sk_{\mathrm{OBU}}, Q, aux)$**.** Parse proof Q as (q, s_q), q as (tag, ϕ) and aux as $(open_k, (k, (loc_k, time_k), p_k))_{k=1}^N$. Find the data structure $(loc_k, time_k)$ such that $\mu(\phi, (loc_k, time_k))$ outputs accept. Set $r = (tag, (k, (loc_k, time_k), p_k), open_k)$ and run $s_r = \mathrm{OBUsign}(sk_{\mathrm{OBU}}, r)$. Output $R = (r, s_r)$.

Check$(params_{Com}, pk_{\mathrm{OBU}}, pk_{\mathrm{TSP}}, m, s_m, Q, R)$**.** Parse R as (r, s_r) and run OBUverify$(pk_{\mathrm{OBU}}, r, s_r)$. If the signature is correct, parse r as $(tag, (\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma), open_\sigma)$, Q as $((tag, \phi), s_q)$ and m as $(tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$. Check that $open_{fee}$ was picked from the adequate interval. Compute $h'_\sigma = H(loc'_\sigma, time'_\sigma)$, check if $h'_\sigma = h_\sigma$ and if $\mu(\phi, (loc'_\sigma, time'_\sigma))$ outputs accept. If it is the case, set $reasonpos = 0$ and otherwise $reasonpos = 1$. Compute $p_\sigma = f(loc'_\sigma, time'_\sigma)$ and check if $p_\sigma = p'_\sigma$. Run Open$(params_{Com}, c_k, p_k, open_k)$. If it opens correctly set $reasonprice = 0$ and otherwise $reasonprice = 1$. If $reasonpos = reasonprice = 0$, output (***not guilty***$, (k, (loc_k, time_k), p_k))$. If not, output (***guilty***$, (k, (loc_k, time_k), p_k))$.

Theorem 1 *This OP scheme securely realizes $\mathcal{F}_{OP}$.*

We prove Theorem 2 in Appendix C.

B.2 Efficient Instantiation

We propose an efficient instantiation for the commitment scheme, TSP's signature scheme and the non-interactive proof of signature possession that are used in the construction described in the previous section. The signature schemes of TC and OBU can be instantiated with any existentially unforgeable signature scheme.

Signature Scheme. We select the signature scheme proposed by Camenisch and Lysyanskaya [8].

- SigKeygen. On input 1^k, generate two safe primes p, q of length k such that $p = 2p' + 1$ and $q = 2q' + 1$. The special RSA modulus of length l_n is defined as $n = pq$. Output secret key $sk = (p, q)$. Choose uniformly at random $S \in_R QR_n$, and $R, Z \in_R \langle S \rangle$. Output public key $pk = (n, R, S, Z)$.
- SigSign. On input message x of length l_x, choose a random prime number e of length $l_e \geq l_x + 3$, and a random number v of length $l_v = l_n + l_x + l_r$, where l_r is a security parameter [8]. Compute the value A such that $Z \equiv A^e R^x S^v (\mathrm{mod}\ n)$. Output the signature (A, e, v).
- SigVerify. On inputs message x and signature (A, e, v), check that $Z = A^e R^x S^v (\mathrm{mod}\ n)$ and $2^{l_e} \leq e \leq 2^{l_e - 1}$.

Commitment Scheme. We select the integer commitment scheme due to Damgard and Fujisaki [14].

- ComSetup. Given a special RSA modulus, pick a random generator $g_1 \in_R QR_n$. Pick random $\alpha \leftarrow \{0,1\}^{l_n + l_z}$ and compute $g_0 = g_1^\alpha$. Output parameters (g_0, g_1, n).
- Commit. On input message x of length l_x, choose a random number $open_x \in \{0,1\}^{l_n + l_z}$, and compute $c_x = {g_0}^x {g_1}^{open_x} (\mathrm{mod}\ n)$. Output the commitment c_x and the opening $open_x$.
- Open. On inputs message x and opening $open_x$, compute $c'_x = {g_0}^x {g_1}^{open_x} (\mathrm{mod}\ n)$ and check whether $c_x = c'_x$.

Non-Interactive Zero-Knowledge Argument. We employ the proof of possession of a signature in [8]. Given a signature (A, e, v) on message x and a commitment to the message $c_x = {g_0}^x {g_1}^{open_x}$, the prover

computes $\tilde{A} = Ag^w$, a commitment $c_w = g^w h^{open_w}$ and a proof that:

$$\mathsf{NIPK}\{\ (x, open_x, e, v, w, open_w, w \cdot e, open_w \cdot e) : c_x = {g_0}^x {g_1}^{open_x} \ \wedge\ Z = \tilde{A}^e R^x S^v (1/g_0)^{w \cdot e} \wedge$$
$$c_w = {g_0}^w {g_1}^{open_w} \ \wedge\ 1 = c_w^e (1/g_0)^{w \cdot e} (1/g_1)^{open_w \cdot e} \ \wedge\ e \in \{0,1\}^{l_e + l_c + l_z} \ \wedge\ x \in \{0,1\}^{l_x + l_c + l_z}\}$$

We turn it into a non-interactive zero-knowledge argument via the Fiat-Shamir heuristic. The prover picks random values:

$r_x \leftarrow \{0,1\}^{l_x+l_c+l_z}$, $r_{open_x} \leftarrow \{0,1\}^{l_n+l_c+l_z}$, $r_w \leftarrow \{0,1\}^{l_n+l_c+l_z}$, $r_{open_w} \leftarrow \{0,1\}^{l_n+l_c+l_z}$
$r_e \leftarrow \{0,1\}^{l_e+l_c+l_z}$, $r_{w \cdot e} \leftarrow \{0,1\}^{l_n+l_e+l_c+l_z}$, $r_v \leftarrow \{0,1\}^{l_v+l_c+l_z}$, $r_{open_w \cdot e} \leftarrow \{0,1\}^{l_n+l_e+l_c+l_z}$

and computes commitments:

$$t_{c_x} = {g_0}^{r_x} {g_1}^{r_{open_x}} \ ,\ t_{c_w} = g^{r_w} h^{r_{open_w}} \ ,\ t'_Z = \tilde{A}^{r_e} R^{r_x} S^{r_v} (1/g_0)^{r_{w \cdot e}} \ ,\ t' = c_w^{r_e} (1/g_0)^{r_{w \cdot e}} (1/g_1)^{r_{open_w \cdot e}}$$

Let the challenge computed by the prover be:

$$ch = H(n||g_0||g_1||\tilde{A}||R||S||1/g_0||1/g_1||c_x||Z||c_w||1||t_{c_x}||t_Z||t_{c_w}||t\ ).$$

The prover computes responses:

$s_x = r_x - ch \cdot x$, $s_{open_x} = r_{open_x} - ch \cdot open_x$, $s_w = r_w - ch \cdot w$
$s_{open_w} = r_{open_w} - ch \cdot open_w$, $s_e = r_e - ch \cdot e$, $s_{w \cdot e} = r_{w \cdot e} - ch \cdot (w \cdot e)$
$s_v = r_v - ch \cdot v$, $s_{open_w \cdot e} = r_{open_w \cdot e} - ch \cdot (open_w \cdot e)$

and sends to the verifier:

$$\pi = (\tilde{A}, c_w, ch, s_x, s_{open_x}, s_e, s_v, s_w, s_{open_w}, s_{w \cdot e}, s_{open_w \cdot e})\ .$$

The verifier computes:

$$t'_{c_x} = c_x^{ch} {g_0}^{s_m} {g_1}^{s_{open_x}} \ ,\ t'_{c_w} = c_w^{ch} {g_0}^{s_w} {g_1}^{s_{open_w}} \ ,\ t'_Z = Z^{ch} \tilde{A}^{s_e} R^{s_x} S^{s_v} (1/g_0)^{s_{w \cdot e}}$$
$$t' = C_w^{s_e} (1/g_0)^{s_{w \cdot e}} (1/g_1)^{s_{open_w \cdot e}}$$

and checks whether:

$$s_e \in \{0,1\}^{l_e+l_c+l_z} \ , \quad s_x \in \{0,1\}^{l_x+l_c+l_z}$$

and finally:

$$ch = H(n||g_0||g_1||\tilde{A}||R||S||1/g_0||1/g_1||c_x||Z||c_w||1||t'_{c_x}||t'_Z||t'_{c_w}||t'\ ).$$

C Security Proof

Theorem 2 *This OP scheme securely realizes $\mathcal{F}_{OP}$.*

In order to prove this theorem, we need to build a simulator $\mathcal{S}$ that invokes a copy of adversary $\mathcal{A}$ and interacts with $\mathcal{F}_{\text{OP}}$ and environment $\mathcal{Z}$ in such a way that ensembles $\text{IDEAL}_{\mathcal{F}_{\text{OP}},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\text{OP},\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. In our setting TC is trusted: it always follows the protocol specification, it inputs correct proofs ϕ and it colludes neither with TSP nor with OBU.

Simulation of TSP **security.** In this case only OBU is corrupted.

$\mathcal{S}$ runs $\mathsf{SetupOP}(1^k)$ to get $params_{Com}$, $\mathsf{TSPkg}(1^k)$ to obtain $(pk_{\text{TSP}}, sk_{\text{TSP}})$, and $\mathsf{TCkg}(1^k)$ to obtain $(pk_{\text{TC}}, sk_{\text{TC}})$.

Upon receiving (crs) from $\mathcal{A}$, $\mathcal{S}$ returns (crs, $params_{Com}$). Upon receiving (retrieve, TSP) (resp. (retrieve, TC)), $\mathcal{S}$ returns $(\text{TSP}, pk_{\text{TSP}})$ (resp. $(\text{TC}, pk_{\text{TC}})$). Upon receiving (register, pk_{OBU}), $\mathcal{S}$ stores $(\text{OBU}, pk_{\text{OBU}})$.

Upon receiving (initialize, f, μ) from $\mathcal{F}_{\text{OP}}$, $\mathcal{S}$ executes $\mathsf{TSPinit}(f, sk_{\text{TSP}})$ to obtain a setup $params$, stores $(f, \mu, sk_{\text{TSP}}, pk_{\text{TSP}}, params_{Com}, params)$ and sends $(f, \mu, params)$ to $\mathcal{A}$.

$\mathcal{S}$ records each random oracle query $H(loc, time)$ performed by $\mathcal{A}$ and responds by providing consistent random values (i.e., if $\mathcal{A}$ queries a value twice, $\mathcal{S}$ returns the same result).

Upon receiving (m, s_m) from $\mathcal{A}$, $\mathcal{S}$ verifies the signature s_m by using pk_{OBU} and ignores the message if it is not correct. $\mathcal{S}$ parses m as $(tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$. For $k = 1$ to N, $\mathcal{S}$ verifies π_k and then extracts the witness $(p_k, open_k, s_k)$ from π_k. $\mathcal{S}$ aborts if p_k is not included in $params$. $\mathcal{S}$ maps to each value h_k output by the random oracle the corresponding query $(loc, time)$. For values h_k that were not output by the random oracle, $\mathcal{S}$ assigns a random pair $(loc, time)$. $\mathcal{S}$ computes $c_{fee} = \prod_{k=1}^N c_k$ to obtain a commitment to the fee and runs $\mathsf{Open}(params_{Com}, c_{fee}, fee, open_{fee})$. If the output is $b = 1$, $\mathcal{S}$ stores (m, s_m) and sends (payment, $tag, fee, (k, (loc_k, time_k), p_k)_{k=1}^N$) to $\mathcal{F}_{\text{OP}}$.

Upon receiving (verifyreq, tag, ϕ) from $\mathcal{F}_{\text{OP}}$, $\mathcal{S}$ executes $\mathsf{Prove}(sk_{\text{TC}}, tag, \phi)$ to obtain a proof Q and sends (Q) to $\mathcal{A}$.

Upon receiving (R) from $\mathcal{A}$, $\mathcal{S}$ parses R as $(tag, (\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma), open_\sigma, s_r)$ and verifies the signature s_r. If it is correct, $\mathcal{S}$ parses m as $(tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$ and checks if $h_\sigma = H(loc'_\sigma, time'_\sigma)$. If it is not the case, $\mathcal{S}$ ignores the message. Otherwise $\mathcal{S}$ aborts if h_σ was not output by the random oracle on input $(loc'_\sigma, time'_\sigma)$. $\mathcal{S}$ runs $\mathsf{Open}(params_{Com}, c_\sigma, p'_\sigma, open_\sigma)$ and aborts if the opening is correct but p'_σ is different from the price extracted from π_σ. If the price is the same, $\mathcal{S}$ sends (verifyresp, $tag, (\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$) to $\mathcal{F}_{\text{OP}}$.

Claim 1 *When only* OBU *is corrupted, the distribution ensembles* $\text{IDEAL}_{\mathcal{F}_{OP},\mathcal{S},\mathcal{Z}}$ *and* $\text{REAL}_{OP,\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable under the existential unforgeability of* TSP*'s signature scheme, the binding property of the commitment scheme, the extractability of the proof system and the collision resistance property of H in the random oracle model.*

Proof. We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\text{REAL}_{\text{OP},\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $\text{IDEAL}_{\mathcal{F}_{\text{OP}},\mathcal{S},\mathcal{Z}}$ with non-negligible probability. We denote by Pr [**Game** i] the probability that $\mathcal{Z}$ distinguishes between the ensemble of **Game** i and that of the real execution.

Game 0**:** This game corresponds to the execution of the real-world protocol with an honest TSP. Therefore, Pr [**Game** 0] $= 0$.

Game 1**:** This game proceeds as **Game** 0, except that $params_{Com}$ and the public keys pk_{TC} and pk_{TSP} are replaced by other $params_{Com}{}'$, $pk_{\text{TC}}{}'$ and $pk_{\text{TSP}}{}'$ that are obtained by running $\mathsf{SetupOP}$, TCkg and TSPkg respectively. Moreover, $params$ is replaced by another setup $params'$, which is generated by running $\mathsf{TSPinit}$ on input the same function f. Since the new values are taken from the same distribution as $(params_{Com}, pk_{\text{TC}}, pk_{\text{TSP}}, params)$, then |Pr [**Game** 1] − Pr [**Game** 0]| $= 0$.

Game 2**:** This game proceeds as **Game** 1, except that, for $k = 1$ to N, we extract the witness $(p_k, open_k, s_k)$ of each proof π_k in message m. Since extraction fails with negligible probability, |Pr [**Game** 2] − Pr [**Game** 1]| $= \nu_1(k)$.

Game 3**:** This game proceeds as **Game** 2, except that **Game** 3 aborts if the extracted p_k is not a price signed in $params'$. The probability that $\mathcal{Z}$ distinguishes between **Game** 3 and **Game** 2 is bounded by the following lemma:

Lemma 1 *If* TSP*'s signature scheme is existentially unforgeable, |Pr [**Game** 3] − Pr [**Game** 2]| $= \nu_2(k)$.*

Proof. Given an adversary $\mathcal{A}$ that makes **Game** 3 abort with non-negligible probability, we construct an algorithm $\mathcal{B}$ that breaks the existential unforgeability of TSP's signature scheme with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}$ of the existential unforgeability game a public

key pk and assigns $pk_{\mathrm{TSP}} = pk$. $\mathcal{B}$ uses the signing oracle provided by $\mathcal{E}$ to compute $params$. Eventually, $\mathcal{A}$ sends a payment tuple (h_k, c_k, π_k) such that the extracted witness $(p_k, open_k, s_k)$ contains a price p_k that was not queried to the signing oracle. $\mathcal{B}$ sends (p_k, s_k) to $\mathcal{E}$ as its forgery.

Game 4**:** This game proceeds as **Game** 3, except that the proof P is replaced by another valid proof P', which is generated by running algorithm Prove on input the correct values (tag, ϕ). Therefore, since P and P' are identically distributed, $|\Pr[\textbf{Game } 4] - \Pr[\textbf{Game } 3]| = 0$.

Game 5**:** This game proceeds as **Game** 4, except that **Game** 5 aborts if the pair $(p'_\sigma, open_\sigma)$ included in R opens correctly the commitment c_σ included in m, but p'_σ is different from the price extracted from proof π_σ. The probability that $\mathcal{Z}$ distinguishes between **Game** 5 and **Game** 4 is bounded by the following lemma:

Lemma 2 *Under the binding property of the commitment scheme, $|Pr[$**Game** $5] - Pr[$**Game** $4]| = \nu_3(k)$.*

Proof. Given an adversary $\mathcal{A}$ that makes **Game** 5 abort with non-negligible probability, we construct an algorithm $\mathcal{B}$ that breaks the binding property of the commitment scheme with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}$ of the binding property game parameters of the commitment scheme and uses them to set $params_{Com}$. Upon receiving a payment tuple (h_k, c_k, π_k), $\mathcal{B}$ extracts the witness $(p_k, open_k, s_k)$ and stores $(c_k, p_k, open_k)$. Eventually, $\mathcal{A}$ outputs a response R with a pair $(p'_\sigma, open'_\sigma)$ such that $p_\sigma \neq p'_\sigma$. $\mathcal{B}$ sends $(c_\sigma, p_\sigma, open_\sigma, p'_\sigma, open'_\sigma)$ to $\mathcal{E}$ in order to break the binding property.

Game 6**:** This game proceeds as **Game** 5, except that **Game** 6 aborts if the pair $(loc'_\sigma, time'_\sigma)$ is a valid preimage of the value h_σ included in m, but h_σ was output on input a different pair $(loc_\sigma, time_\sigma)$. The probability that $\mathcal{Z}$ distinguishes between **Game** 6 and **Game** 5 is bounded by the following lemma:

Lemma 3 *Under the collision resistance property of H, $|Pr[$**Game** $6] - Pr[$**Game** $5]| = \nu_4(k)$.*

Proof. Given an adversary $\mathcal{A}$ that makes **Game** 6 abort with non-negligible probability, we construct an algorithm $\mathcal{B}$ that breaks the collision resistance property of the hash function with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}$ a hash function H. Upon receiving a random oracle query $(loc_k, time_k)$, $\mathcal{E}$ stores $(loc_k, time_k)$ and outputs h_k. When receiving a payment tuple (h_k, c_k, π_k), $\mathcal{B}$ records $(h_k, loc_k, time_k)$. Eventually, $\mathcal{A}$ outputs a response R where the pair $(loc'_\sigma, time'_\sigma)$ is a preimage of h_k, but $(loc'_\sigma, time'_\sigma) \neq (loc_k, time_k)$. $\mathcal{B}$ sends $(h_k, (loc_k, time_k), (loc'_\sigma, time'_\sigma))$ to $\mathcal{E}$ to break the collision resistance property of H.

$\mathcal{S}$ performs all the changes described in **Game** 6, and forwards and receives messages from $\mathcal{F}_{\mathrm{OP}}$ as described in our simulation. The distribution produced in **Game** 6 is identical to that of our simulation. Therefore, by summation we have that $|\Pr[\textbf{Game } 6] \leq \nu_5(k)$.

Simulation of OBU **security.** In this case only TSP is corrupted.

$\mathcal{S}$ runs $\mathsf{SetupOP}(1^k)$ to get $params_{Com}$ and a trapdoor t, $\mathsf{OBUkg}(1^k)$ to obtain $(pk_{\mathrm{OBU}}, sk_{\mathrm{OBU}})$ and $\mathsf{TCkg}(1^k)$ to obtain $(pk_{\mathrm{TC}}, sk_{\mathrm{TC}})$. Trapdoor t allows $\mathcal{S}$ to open a commitment to any value.

Upon receiving (crs) from $\mathcal{A}$, $\mathcal{S}$ returns $(\mathrm{crs}, params_{Com})$. Upon receiving $(\mathrm{register}, pk_{\mathrm{TSP}})$, $\mathcal{S}$ stores $(\mathrm{TSP}, pk_{\mathrm{TSP}})$. Upon receiving (retrieve, OBU) (resp. (retrieve, TC)) from $\mathcal{A}$, $\mathcal{S}$ returns $(\mathrm{OBU}, pk_{\mathrm{OBU}})$ (resp. $(\mathrm{TC}, pk_{\mathrm{TC}})$).

Upon receiving $(f, \mu, params)$ from $\mathcal{A}$, $\mathcal{S}$ runs $\mathsf{OBUinit}(params, pk_{\mathrm{TSP}})$. If $b = 0$, $\mathcal{S}$ ignores the message. Otherwise $\mathcal{S}$ stores $params$ and sends $(\mathrm{initialize}, f, \mu)$ to $\mathcal{F}_{\mathrm{OP}}$.

Upon receiving (payment, tag, fee, N) from $\mathcal{F}_{\text{OP}}$, $\mathcal{S}$ chooses N prices such that $p_1 + \ldots + p_N = fee$. Then $\mathcal{S}$ computes $m = (tag, fee, open_{fee}, (h_k, c_k, \pi_k)_{k=1}^N)$ and the signature s_m by following algorithm Pay, except that h_k is set to a random string of the required length. $\mathcal{S}$ sends (m, s_m) to $\mathcal{A}$.

Upon receiving (proof, tag, ϕ) from $\mathcal{F}_{\text{OP}}$, $\mathcal{S}$ runs $\mathsf{Prove}(sk_{\text{TC}}, tag, \phi)$ to obtain a proof Q and sends (Q) to $\mathcal{A}$.

Upon receiving (Q') from $\mathcal{A}$, $\mathcal{S}$ runs $\mathsf{VerifyProof}(pk_{\text{TC}}, Q')$ and ignores the message if the result is not correct. Otherwise, $\mathcal{S}$ parses Q' as $(q', s_{q'})$ and Q as (q, s_q) and, if $q \neq q'$, $\mathcal{S}$ aborts. Otherwise $\mathcal{S}$ sends (verify, tag, ϕ) to $\mathcal{F}_{\text{OP}}$.

Upon receiving the message (verifyresul, *not guilty*, $(\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$) or the message (verifyresul, *guilty*, $(\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma)$) from $\mathcal{F}_{\text{OP}}$, $\mathcal{S}$ parses m to obtain $(h_\sigma, c_\sigma, \pi_\sigma)$ and uses the trapdoor t to compute an opening $open$ such that the output of $\mathsf{Open}(params_{Com}, c_\sigma, p'_\sigma, open)$ is correct. $\mathcal{S}$ sets $R = ((tag, (\sigma, (loc'_\sigma, time'_\sigma), p'_\sigma), open_\sigma), s_r)$ and sends (R) to $\mathcal{A}$. When $\mathcal{A}$ submits an oracle query of the form $(loc, time)$, if $(loc, time) = (loc'_\sigma, time'_\sigma)$, $\mathcal{S}$ returns h_σ. Otherwise $\mathcal{S}$ returns a consistent random value.

Upon receiving $((m', s_{m'}), R')$ from $\mathcal{A}$, $\mathcal{S}$ executes $\mathsf{VerifyPayment}(params_{Com}, pk_{\text{OBU}}, pk_{\text{TSP}}, m', s_{m'})$, parses R' as (r, s_r) and runs $\text{OBU}\mathsf{verify}(pk_{\text{OBU}}, r, s_r)$. If all these checks verify but $m' \neq m$ or $r' \neq r$ then $\mathcal{S}$ aborts. Otherwise $\mathcal{S}$ sends (blame, tag) to $\mathcal{F}_{\text{OP}}$.

Claim 2 *When only* TSP *is corrupted, the distribution ensembles* $\text{IDEAL}_{\mathcal{F}_{OP},\mathcal{S},\mathcal{Z}}$ *and* $\text{REAL}_{OP,\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable under the hiding property of the commitment scheme, the zero-knowledge property of the proof system, and the existential unforgeability of the signatures schemes of* TC *and* OBU.

Proof.

Game 0**:** This game corresponds to the execution of the real-world protocol with an honest OBU. Therefore, Pr [**Game** 0] $= 0$.

Game 1**:** This game proceeds as **Game** 0, except that $params_{Com}$ and the public keys pk_{TC} and pk_{OBU} are replaced by other other $params_{Com}{}'$ and other public keys $pk_{\text{TC}}{}'$ and $pk_{\text{OBU}}{}'$ that are obtained by running SetupOP, TCkg and OBUkg respectively. Since these public keys have the same distribution as pk_{TC} and pk_{OBU}, then $|\text{Pr [Game 1]} - \text{Pr [Game 0]}| = 0$.

Game 2**:** This game proceeds as **Game** 1, except that, for $k = 1$ to N, the values h_k are replaced by random strings. Under the assumption that H behaves as a random oracle, values h_k and random strings have the same distribution. Therefore, $|\text{Pr [Game 2]} - \text{Pr [Game 1]}| = 0$.

Game 3**:** This game proceeds as **Game** 2, except that, for $k = 1$ to N, the commitment c_k and the proof π_k are replaced by another valid commitment c_k to a price p'_k such that $\sum_{i=1}^N p'_k = fee$, and by a proof π'_k that uses as witness $(p'_k, open'_k)$ and the signature on p'_k included in $params$. The probability that $\mathcal{Z}$ distinguishes between **Game** 3 and **Game** 2 is bounded by the following lemma:

Lemma 4 *Under the assumption that the commitment scheme is hiding and the non-interactive proof system is zero-knowledge,* $|\text{Pr [Game 3]} - \text{Pr [Game 2]}| = \nu_1(k)$.

Proof. We employ a sequence of hybrid games. Let game-i be the game in which the payment tuples $(h_k, c_k, \pi_k)_{k=1}^i$ consist of a commitment c_k and a proof π_k that are computed by using an (incorrect) price p'_k, while the tuples $(h_k, c_k, \pi_k)_{k=i+1}^N$ remain unchanged, i.e., (c_k, π_k) are computed on input the valid price p_k. Clearly, game-0 corresponds to **Game** 2, and game-N corresponds to **Game** 3. If an environment $\mathcal{Z}$ distinguishes **Game** 3 from **Game** 2 with non-negligible probability ϵ, there must be an index i such that $\mathcal{Z}$ distinguishes game-$(i+1)$ from game-i with non-negligible probability at least ϵ/N. Given such a $\mathcal{Z}$, we construct an algorithm $\mathcal{B}$ that breaks the hiding property of the commitment scheme with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}_{com}$ of the hiding property

game the parameters of the commitment scheme and uses them to set up $params_{Com}$. $\mathcal{B}$ computes a payment message as follows. The first i payment tuples are replaced by tuples computed on input random prices in $\mathrm{Im}(f)$, while, from $k = i + 2$ to N, payment tuples remain unchanged. (The sum of all prices should be *fee*.) To compute the $(i + 1)$-tuple, $\mathcal{B}$ submits a challenge (p_0, p_1), where p_0 is the original price of tuple $i + 1$ and p_1 is the new random price, to the challenger $\mathcal{E}_{com}$. $\mathcal{E}$ flips a coin b, computes $(c_b, open_b) = \mathsf{Commit}(params_{Com}, p_b)$ and returns c_b. $\mathcal{B}$ sets $c_{i+1} = c_b$ and uses the zero-knowledge property of the proof system to compute a simulated proof π_{i+1}. The $(i+1)$-payment tuple is $(h_{i+1}, c_{i+1}, \pi_{i+1})$. Clearly, if $b = 0$ the distribution corresponds to that of game-i, while if $b = 1$ the distribution corresponds to that of game-$(i + 1)$. $\mathcal{Z}$ outputs a bit b', which is forwarded by $\mathcal{B}$ to $\mathcal{E}_{com}$ as its guess for the hiding property game.

Game 4**:** This game proceeds as **Game** 3, except that the proof Q that is sent to $\mathcal{A}$ is replaced by another valid proof Q', which is generated by running algorithm Prove on input the correct values (tag, ϕ). Therefore, since Q and Q' are identically distributed, $|\Pr[\textbf{Game } 5] - \Pr[\textbf{Game } 4]| = 0$.

Game 5**:** This game proceeds as **Game** 4, except that $\mathcal{S}$ parses Q' as (q', s'_q) and Q as (q, s_q) and aborts if $q' \neq q$. The probability that $\mathcal{Z}$ distinguishes between **Game** 5 and **Game** 4 is bounded by the following lemma:

Lemma 5 *Under the unforgeability of* TC*'s signature scheme ,* $|Pr[\textbf{\textit{Game }} 5] - Pr[\textbf{\textit{Game }} 4]| = \nu_2(k)$.

Proof. Given an adversary $\mathcal{A}$ that makes **Game** 5 abort with non-negligible probability, we construct an algorithm $\mathcal{B}$ that breaks the existential unforgeability of TC's signature scheme with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}$ of the existential unforgeability game a public key pk and assigns $pk_{\mathrm{TC}} = pk$. $\mathcal{B}$ uses the signing oracle provided by $\mathcal{E}$ to obtain a signature s_q on q. Eventually, $\mathcal{A}$ sends a proof $Q' = (q', s_{q'})$ where $s_{q'}$ is valid signature on q', but q' was not submitted to the signing oracle. $\mathcal{B}$ sends $(q', s_{q'})$ to $\mathcal{E}$ as its forgery.

Game 6**:** This game proceeds as **Game** 5, except that the opening $open_\sigma$ included in R is replaced by other opening $open$, such that the commitment c_σ can be opened to p_σ in R. Under the assumption that we use a trapdoor commitment scheme, trapdoor t always allows finding such an opening. Moreover, since both openings are equally distributed, $|\Pr[\textbf{Game } 6] - \Pr[\textbf{Game } 5]| = 0$.

Game 7**:** This game proceeds as **Game** 6, except that **Game** 7 aborts if $(m', s_{m'})$ or $(r', s_{r'})$ received from $\mathcal{A}$ are correct but $m' \neq m$ or $r \neq r'$, where (m, r) were previously sent to $\mathcal{A}$. The probability that $\mathcal{Z}$ distinguishes between **Game** 7 and **Game** 6 is bounded by the following lemma:

Lemma 6 *Under the unforgeability of* OBU*'s signature scheme,* $|Pr[\textbf{\textit{Game }} 7] - Pr[\textbf{\textit{Game }} 6]| = \nu_3(k)$.

Proof. Given an adversary $\mathcal{A}$ that makes **Game** 7 abort with non-negligible probability, we construct an algorithm $\mathcal{B}$ that breaks the existential unforgeability of TC's signature scheme with non-negligible probability. $\mathcal{B}$ receives from the challenger $\mathcal{E}$ of the existential unforgeability game a public key pk and assigns $pk_{\mathrm{OBU}} = pk$. $\mathcal{B}$ uses the signing oracle provided by $\mathcal{E}$ to obtain a signature s_m on each payment message m and a signature s_r on each response r. Eventually, $\mathcal{A}$ sends a pair $(m', s_{m'})$ or a pair $(r', s_{r'})$ with valid signatures, but such that m' or r' were not submitted to the signing oracle. $\mathcal{B}$ sends $(m', s_{m'})$ or $(r', s_{r'})$ to $\mathcal{E}$ as its forgery.

$\mathcal{S}$ performs all the changes described in **Game** 7, and forwards and receives messages from $\mathcal{F}_{\mathrm{OP}}$ as described in our simulation. The distribution produced in **Game** 7 is identical to that of our simulation. Therefore, by summation we have that $|\Pr[\textbf{Game } 7] \leq \nu_4(k)$.

D Efficient Instantiation Based on Range Proofs

We describe another possible instantiation of the construction presented in Sect. B.1. It differs from the instantiation described in Sect. B.2 in the way OBU proves that it uses prices that belong to $\mathrm{Im}(f)$. In Sect. B.2, we employ a set membership proof where OBU proves possession of a signature by TSP on the price. Now, we define $\mathrm{Im}(f)$ to be any non-negative value, and OBU employs a range proof to prove that it uses non-negative prices.

The proof computed in algorithm Pay is now $\pi_k = \mathsf{NIPK}\{(p_k, open_k) : (c_k, open_k) = \mathsf{Commit}(params_{Com}, p_k) \wedge p_k \geq 0\}$. Moreover, TSP does not employ any signature scheme, and thus algorithms TSPkg, TSPinit and OBUinit are unnecessary.

Commitment Scheme. We select the integer commitment scheme due to Groth [26].

- ComSetup. Given a special RSA modulus, pick a random generator $h \in QR_n$. Pick random $\alpha, \alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \{0,1\}^{l_n+l_z}$ and compute $g = h^{\alpha}, g_1 = h^{\alpha_1}, g_2 = h^{\alpha_2}, g_3 = h^{\alpha_3}, g_4 = h^{\alpha_4}$. Output public commitment parameters $(g, g_1, g_2, g_3, g_4, h, n)$.
- Commit. On input integers $\langle m, m_1, m_2, m_3, m_4\rangle$ of length l_m, choose a random $open \in \{0,1\}^{l_n+l_z}$, and compute $C = g^m g_1^{m_1} g_2^{m_2} g_3^{m_3} g_4^{m_4} h^{open} (\mathrm{mod}\ n)$. Output the commitment C and the opening $open$.
- Open. On inputs integers $\langle m', m'_1, m'_2, m'_3, m'_4\rangle$ and $open'$, compute $C' = g^{m'} g_1^{m'_1} g_2^{m'_2} g_3^{m'_3} g_4^{m'_4} h^{open'} (\mathrm{mod}\ n)$ and check whether $C = C'$.

Non-Interactive Zero-Knowledge Argument. We employ the non-interactive zero-knowledge argument due to Groth [26] to prove that an integer $m \geq 0$. The proof is based on the fact that any positive integer m of the form $4m+1$ can be written as a sum of three squares $a^2 + b^2 + d^2$. Therefore, to prove that $m \geq 0$, Groth proposes to prove that $4m + 1 = a^2 + b^2 + d^2$. Values (a, b, c) can be computed via the Rabin-Shallit algorithm [40]. The proof is:

$$\mathsf{NIPK}\{\ (m, open_m, a, b, d) : C_m = g^m h^{open_m} \ \wedge\ 4m + 1 = a^2 + b^2 + d^2\}$$

The prover picks random $r_m, r_a, r_b, r_d \in \{0,1\}^{l_n+l_c+l_z}$ and computes $\Delta = 4r_m - 2ar_a - 2br_b - 2dr_d$. Then the prover picks random $open_m, open_{sq} \in \{0,1\}^{l_n+l_z}$ and computes a commitment to the message $C_m = g^m h^{open_m}$ and a commitment $C_{sq} = g^m g_1^a g_2^b g_3^d g_4^{\Delta} h^{open_{sq}}$. The prover picks $r_{open_m}, r_{open_{sq}} \in \{0,1\}^{l_n+l_z+l_c+l_z}$ and computes $t_{C_m} = g^{r_m} h^{r_{open_m}}$ and $t_{C_{sq}} = g^{r_m} g_1^{r_a} g_2^{r_b} g_3^{r_d} g_4^{-r_a^2 - r_b^2 - r_d^2} h^{r_{open_{sq}}}$. Then the prover computes the challenge $c = \mathcal{H}(n||g||g_1||g_2||g_3||g_4||h||C_m||C_{sq}||t_{C_m}||t_{C_{sq}})$ and the responses $s_m = r_m - cm$, $s_a = r_a - ca$, $s_b = r_b - cb$, $s_d = r_d - cd$, $s_{open_m} = r_{open_m} - copen_m$ and $s_{open_{sq}} = r_{open_{sq}} - copen_{sq}$. The prover sends $(C_m, C_{sq}, c, s_m, s_a, s_b, s_d, s_{open_m}, s_{open_{sq}})$.

The verifier computes $s_{\Delta} = -c(4s_m + c) - s_a^2 - s_b^2 - s_d^2$. Then the verifier computes $t'_{C_m} = C_m^c g^{s_m} h^{s_{open_m}}$ and $t'_{C_{sq}} = C_{sq}^c g^{s_m} g_1^{s_a} g_2^{s_b} g_3^{s_d} g_4^{s_{\Delta}} h^{s_{open_{sq}}}$ and checks whether $c = \mathcal{H}(n||g||g_1||g_2||g_3||g_4||h||C_m||C_{sq}||t'_{C_m}||t'_{C_{sq}})$.