Privacy-Preserving Smart Metering

Alfredo Rial
ESAT-COSIC / IBBT
KU Leuven, Belgium
alfredo.rial@esat.kuleuven.be

George Danezis
Microsoft Research
Cambridge, UK
gdane@microsoft.com

***Abstract*—Current smart grid proposals threaten users' privacy by disclosing fine-grained consumption data to utility providers, primarily for billing. We propose a privacy-preserving protocol for billing and for performing general calculations on fine grained meter readings. Our protocol combines a signed tariff policy from the utility with signed readings output by a tamper-resistant meter to compute an aggregate bill. It uses zero-knowledge proofs to ensure that the fee is correct without disclosing any consumption data. A wide variety of practical tariff policies can be applied and easily changed. Tariff structures currently proposed for smart-meters are particularly simple and efficient. Our implementation shows that the schemes are generally practical in terms of both communication and computational cost. They can be integrated in a smart metering system in various ways, and we analyse their suitability for different deployments. Furthermore our constructions are generic enough to be used in other billing settings such as pay-as-you-drive car insurance, electronic traffic pricing and on-line service provision.**

***Keywords*-Privacy; Security; Smart metering; Smart grid; Billing; Pay-as-you drive; tolling; tariffs.**

I. Introduction

The concept of smart grid refers to the modernization of the existing electrical grid, including bidirectional communication between meters and utilities, more accurate meter readings and flexible tariffs [1]. Expected electricity savings depend on matching generation and demand, which is done through feedback on consumers' electricity consumption, as well as on billing using flexible tariffs with higher rates during peak consumption periods. Both the United States and the European Union currently promote the deployment of smart grids.[1]

Currently, most smart grid deployment projects lean towards an architecture with severe privacy problems [2]: meters send all fine-grain measurements to the utilities or a centralised database. Yet, it is recognised that meter readings are personal information. For example, it is possible through load monitoring [3], [4] to identify which electrical appliances are used. As a result, detailed consumption data would facilitate the creation of users' lifestyle profiles, with information such as when they are at home, when they eat, whether they arrive late to work, etc.

[1]US Energy Independence and Security Act of 2007 and EU directive 2009/72/EC

Because of such concerns, privacy impact assessments (PIA) are included in ongoing standardization processes [5]. NIST has identified a number of key personally identifiable items of information, including fine-grained readings that are used for purposes of load monitoring, forecasting, demand-response, efficiency analysis, and billing. All other functions aside billing can be performed by pushing data to the customer or on sampled, incomplete, anonymizimed or aggregated meter readings and do not require the fine-grained readings to be made available to providers. Thus privacy friendly billing is the key challenge, and one that has never been addressed appropriately.

As a result of privacy concerns the mandatory deployment of smart meters has been prohibited in The Netherlands [6]. The current deadlock between mandated deployment and consumer reaction stems from the assumption that smart metering is necessarily privacy invasive. Our protocols demonstrate that smart grid billing does not have to be any more privacy invasive than electricity billing today.

Our Contribution: We propose a set of protocols amongst a provider, a user and a tamper-evident meter. The meter outputs certified readings of measurements and gives them to the user. The user combines those readings with a certified tariff policy to produce a final bill. The bill is then transmitted to the provider alongside a zero-knowledge proof that shows the calculation is correct, without leaking any additional information. Complex, non-linear tariff policies can be applied over individual meter readings or arbitrary periods of time (i.e. per day, per week). While no other information is leaked for billing purposes, we allow individual or aggregate meter readings to be disclosed according to a privacy policy to facilitate abuse prevention, or load prediction.

The provider ensures meter readings are signed correctly through tamper-evident hardware and counters as for conventional metering security. Users can delegate the calculation of their bill to any device or service they wish, without compromising the security properties of the scheme. This greatly facilitates deployment and their free choice strengthens privacy.

Variants of the scheme ensure privacy even if the meter actively attempts to leak information to the provider by eliminating any covert channel in the protocols. On the other

hand, when the user trusts the meter not to leak information, a very efficient protocol for billing that employs simple tariff policies can be used. It requires no zero-knowledge proofs and relies instead on homomorphic properties of commitments.

Outline of the Paper: First we provide an overview of current work on smart-grid privacy in Sect. II. Sect. III provides a high level overview of our scheme and system model. The formal definitions of security for smart metering are in Sect. IV, and the properties we expect from our cryptographic primitives in Sect. V. The cryptographic construction is detailed in Sect. VI, and its real-world performance is evaluated in Sect. VII. We discuss how best to deploy our technology in the current environment in Sect. VIII and conclude in Sect. IX.

II. Related Work.

The damage that smart meters can cause to users' privacy has previously been studied both from a technical perspective [7], [8] and from a legal perspective [1], [9]. These works propose enforcement of privacy properties based on organizational means, codes of conduct and regulations, subject to current legislations. In those works there is an assumption that leaking detailed measurements is inevitable as they are required for billing.

So far, little work has been done on the design of technical solutions to protect users' privacy specifically for smart grids. Wagner et al. [10] propose a privacy-aware framework for the smart grid based on semantic web technologies. Garcia and Jacobs [11] design a multiparty computation protocol that allows a set of users (those living in the same neighbourhood) to compute the sum of their consumption without disclosing their individual consumption. The result is obtained by data concentrators at neighbourhood level, which can compare it with their own measurement of the total consumption in order to detect fraud. The NIST privacy subgroup [5] suggests anonymizing traces of readings, as proposed by Efthymiou et al. [**?**], but also warns of the ease of re-identification. No solution is provided for computing privately individual bills without leaking information.

Other work focuses on further aspects of smart grids. Anderson and Fuloria [2] analyze the security economics of electricity metering. In addition to privacy issues, they discuss behavioural economics aspects needed to understand how smart meters can help to save electricity, pricing policies, and the conflict of interests among the different entities. McLaughlin et al. [12] analyze security of smart grids and conclude that they introduce new vulnerabilities that ease electricity theft. The design of algorithms that schedule energy consumption to reduce costs has also been addressed [13]. Proposals to enhance the security of the smart grid infrastructure include Fatemieh et al. [14].

Smart-metering is a special case of metering. LeMay et al. propose an architecture for attested metering [15] based on calculation performed on trusted hardware. Our protocol follows an approach similar to the one described in [16], [17] for the design of a privacy-friendly electronic toll pricing system. Balasch et al. use 'spot checks', whereas we use small tamper evident meters instead to protect the integrity of raw readings. We extend the paradigm of proving some aspects of a metering system using cryptography, to providing full end-to-end verifiability for billing.

Troncoso et al. [18] propose an architecture in which secure meters (or black boxes in cars in their setting) are used to calculate final bills for pay-as-you-drive insurance. While this architecture could be applied to our secure metering settings it has some drawbacks. Meters are larger and more complex making their independent certification by metrological authorities harder. Changing complex tariff structures would require remote upgrade facilities or physical inspection which is not desirable for electricity meters (the aim of smart-grids is to limit physical inspections). End-to-end verifiability is limited as the integrity of the bill depends on the correct functioning of the software performing the calculation. Different parties cannot rely on the same set of certified readings to bill customers for different usages (for example car insurance and road-taxation both rely on the same position of the car). Most importantly the black box model might misalign incentives [19]: meters are provided by utilities that have no incentive to invest in high quality privacy for their customers. They are regulated by metrological authorities that have no established competence in mandating privacy features[2].

The aim of our protocols is to keep meters extremely simple, and rely on cryptographic calculations outside the meter for integrity. Our approach provides flexibility about where calculations are performed, future-proofing smart-gird deployments while preserving high integrity. We enforce privacy by simply prohibiting information flows from the simple metrological unit to service providers that are not mediated through a device or service under the control of the customer that calculates the bill. This is a simple certification goal that can be implemented through physical separation of the metrological unit from the rest of the smart meter. This is already a design principle to prevent software attacks from disabling the meter.

III. System Model

We propose a protocol to preserve users' privacy in smart metering applications that is flexible enough to be applied in a number of settings, such as electricity, water or gas metering, etc. Our schemes can also be extended to automotive applications such as pay-as-you-drive insurance, tolling, road taxation and congestion charging. It can handle

[2]For example, the UK SI 2006 No. 1679 "The Measuring Instruments (Active Electrical Energy Meters) Regulations 2006" sets out the regulatory framework for certifying meters. The UK National Measurement Office certifies metrological units.

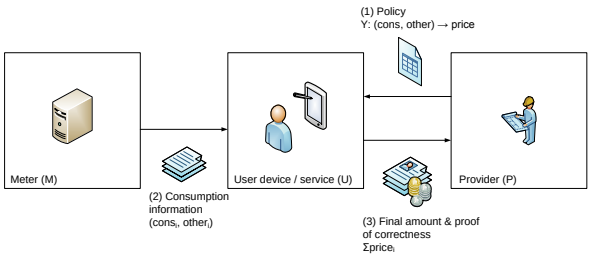

Figure 1. Interactions between parties.

a wide variety of pricing policies and it does not impose restrictions on the quantities to be measured, their number, or on the frequency of measurements. Our main requirement is the use of simple tamper-resistant meters that output signed readings.

We describe our protocol in an abstract setting that comprises three parties, as illustrated in Figure 1: a meter M that outputs consumption data $cons$ and related information $other$; a service provider P that establishes a pricing policy Υ and that, at each billing period, requests the user to pay the fee fee corresponding to her total consumption; and a user U that receives consumption readings from meter M and pays a fee to provider P. The pricing policy $\Upsilon : (cons, other) \rightarrow price$ is a public function that takes in consumption data $cons$ along with other information $other$ (e.g., the time of consumption, ...) and outputs a price. The fee is computed by adding the prices corresponding to the total consumption in a billing period, i.e., if n is the number of readings, $fee = \sum_{i=1}^{n} price_i$. Pricing policies can also be applied to aggregates of readings (e.g. per day or per week) leading to further prices that contribute to the final fee.

The basic operation of the system is as follows: the provider P sends the user U a pricing policy Υ. During a billing period, the meter M outputs consumption readings $cons$ along with other information $other$ that influences the cost. This output is collected by the user U who computes the total fee and sends it to the provider P. The user U also produces and sends a proof that the fee is correctly computed, namely by applying the pricing policy Υ to all the consumption measurements output by the meter M.

Obviously, users U perform all necessary calculations on devices they trust, that can include parts of the smart-meter itself, home computers, smart portable devices or even on-line services. We discuss trade-offs between those options in the last section.

We present some example pricing policies that are fairly generic as well as efficient: a *Linear Policy* sets a cost per unit of consumption (for example how much each unit of electricity costs at different times); a *Cumulative Policy* determines the price to be paid as a set of different linear functions determined by the amount consumed. The latter mechanism allows the expression of complex, non-linear pricing policies, such as imposing different rates per unit of electricity before and after a certain consumption threshold. Any policy can be applied for any time interval, i.e. per day, week, month, and policies can be chained and combined at will without leaking additional information.

Our protocol guarantees the following security properties. First *integrity:* the provider P is assured that the user U reports the correct fee. Second *privacy*, the provider P does not learn any information but the total fee, i.e., the provider P does not get any information on the detailed readings or on other information used to compute the fee (aside the number of readings). Finally, the provider P cannot claim that a user U must pay an incorrect fee.

Additionally, our protocol provides the possibility that the user U discloses selectively some consumption readings to the provider P, or some function of the readings (like the total consumption for a billing period). This only happens with the user's U consent, and the provider P is assured that the reported consumption measurements are correct, i.e., they are computed based on the outputs of the meter M.

To achieve our integrity properties we require simple tamper-resistant meters, as it is the case today and proposed for all forms of smart-metering. In our scheme each meter stores a signature key, and signs the readings it outputs. This is a low-value key, unique to the meter, and compromising it does not affect the integrity of other meters. No other computation is performed within the tamper-resistant enclosure, no mobile code needs to be executed within, and no updates are necessary, making it cheap to manufacture and easy to formally verify. As argued by Garcia and Jacobs [11], independently verified tamper-resistant meters are also required for consumer protection.

Our system model differs from ongoing smart grid projects as we restrict direct communication between the metering core M and the provider P to protect privacy. A multi-level policy [20] is defined to protect confidentiality: raw readings in the meter M are classified as "high" and the provider's P systems are cleared only for "low" information, i.e. the final billing information. It is safe for information to flow "up" from the provider P to the meter M, for example to push updates or commands. Any flow of information from the meter M to the provider P has to be mediated and declassified through the user U running our protocols to ensure it leaks only the final bill and optionally other information according to the policy.

As a result our architecture does not hinder other possible functionalities of smart meters, such as the ability of the provider P to switch users off remotely or to switch them to a prepayment mode when the user U does not pay. Deciding whether those features are desirable from a consumer protection perspective is a different matter [2]. Our architecture allows the user U, the devices they control or services they authorise to access consumption information, but such access is not necessary for billing.

IV. Definitions

A. Security Model

We define security following the ideal-world/real-world paradigm [21]. In the real world, a set of parties interact according to the protocol description in the presence of a real adversary $\mathcal{A}$, while in the ideal world dummy parties interact with an ideal functionality that carries out the desired task in the presence of an ideal adversary $\mathcal{E}$. 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 $\mathcal{E}$ and dummy parties interact with an ideal functionality $\mathcal{F}_\psi$. More formally, we say that protocol ψ emulates the ideal process when, for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{E}$ such that for all environments $\mathcal{Z}$, the ensembles $\mathrm{IDEAL}_{\mathcal{F}_\psi,\mathcal{E},\mathcal{Z}}$ and $\mathrm{REAL}_{\psi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. We refer to [21] for a description of how these ensembles are constructed. 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 the description of our ideal functionalities.

B. Privacy-Preserving Smart Metering

We first define an ideal functionality $\mathcal{F}_{\mathrm{PSM}}$ for privacy-preserving smart metering (see Figure 2). Any construction that realizes $\mathcal{F}_{\mathrm{PSM}}$ ensures that U pays the right fee for the consumption data output by M, i.e., that the fee is computed following Υ. It also ensures that, if M and P do not collude, P only learns the fee paid, i.e., it learns neither the consumption data *cons* nor the other information *other* used to compute *fee*. Additionally, it ensures that a malicious provider cannot prove that the fee that U must pay is different from the one computed following Υ.

Our construction operates in the $\mathcal{F}_{\mathrm{REG}}$-hybrid model [21] (Figure 3), where parties register their public keys at a trusted registration entity. Below we depict the ideal functionality $\mathcal{F}_{\mathrm{REG}}$, which is parameterized with a set of participants $\mathcal{P}$ that is restricted to contain M, U and P only. This functionality abstracts key management, which is a separate concern from privacy.

V. Preliminaries

Signature Schemes: A signature scheme consists of the algorithms $(\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$. $\mathsf{Keygen}(1^k)$ outputs a key pair (sk, pk). $\mathsf{Sign}(sk, m)$ outputs a signature s on message m. $\mathsf{Verify}(pk, s, m)$ outputs accept if s is a valid signature on m and reject otherwise. This definition can be extended to support multi-block messages $\vec{m} = \{m_1, \ldots, m_n\}$. Existential unforgeability [22] requires that no p.p.t. adversary should be able to output a message-signature pair (s, m) unless he has previously obtained a signature on m.

Functionality $\mathcal{F}_{\mathrm{REG}}$

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

- Upon receiving $(\mathsf{register}, v)$ from party $P \in \mathcal{P}$, it records the value (P, v).
- Upon receiving $(\mathsf{retrieve}, P)$ from party $P' \in \mathcal{P}$, if (P, v) is recorded then return $(\mathsf{retrieve}, P, v)$ to P'. Otherwise send $(\mathsf{retrieve}, P, \perp)$ to P'.

Figure 3. The $\mathcal{F}_{\mathrm{REG}}$ functionality abstracting key management.

Commitment schemes: A non-interactive commitment scheme consists of the algorithms ComSetup, Commit and Open. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme par_c. $\mathsf{Commit}(par_c, x)$ outputs a commitment c_x to x and auxiliary information $open_x$. A commitment is opened by revealing $(x, open_x)$ and checking whether $\mathsf{Open}(par_c, c_x, x, open_x)$ outputs accept. 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'.

Our fast protocols make heavy use of homomorphic commitment schemes. A commitment scheme is said to be additively homomorphic if, given two commitments c_{x_1} and c_{x_2} with openings $(x_1, open_{x_1})$ and $(x_2, open_{x_2})$ respectively, there exists an operation $\otimes$ such that, if $c = c_{x_1} \otimes c_{x_2}$, then $\mathsf{Open}(par_c, c, x_1 + x_2, open_{x_1} + open_{x_2})$ outputs accept. Additionally, we require a commitment scheme that also provides an operation $\odot$ between a commitment c_{x_1} and a value x_2 such that, if $c = c_{x_1} \odot x_2$, then $\mathsf{Open}(par_c, c, x_1 \times x_2, open_{x_1} \times x_2)$ outputs accept.

For the purposes of proving security, we employ a trapdoor commitment scheme, in which algorithm $\mathsf{ComSetup}(1^k)$ generates par_c and a trapdoor td. Given a commitment c with opening $(x_1, open_{x_1})$ and a value x_2, the trapdoor td allows finding $open_{x_2}$ such that algorithm $\mathsf{Open}(par_c, c, x_2, open_{x_2})$ outputs accept.

Proofs of Knowledge: A zero-knowledge proof of knowledge [23] is a two-party protocol between a prover and a verifier. The prover proves to the verifier knowledge of some secret input (witness) that fulfills some statement without disclosing this input to the verifier. The protocol should fulfill two properties. First, it should be a proof of knowledge, i.e., a prover without knowledge of the secret input convinces the verifier with negligible probability. More technically, there exists a knowledge extractor that extracts the secret input from a successful prover with all but negligible probability. Second, it should be zero-knowledge,

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

Running with a meter M, a service provider P, and a user U, $\mathcal{F}_{\text{PSM}}$ works as follows:

- On input $(\mathsf{policy}, \Upsilon)$ from P, $\mathcal{F}_{\text{PSM}}$ stores Υ and sends $(\mathsf{policy}, \Upsilon)$ to U.
- On input $(\mathsf{consume}, cons, other)$ from M, $\mathcal{F}_{\text{PSM}}$ increments a counter d and appends $(d, cons, other)$ to a table T that stores all the consumptions. $\mathcal{F}_{\text{PSM}}$ sends $(\mathsf{consume}, cons, other)$ to U.
- On input $(\mathsf{payment})$ from P, $\mathcal{F}_{\text{PSM}}$ computes the total fee fee as follows. Let N be the number of entries stored in T after the previous $(\mathsf{payment})$ message was received. For $i = d - N$ to d, $\mathcal{F}_{\text{PSM}}$ calculates $price_i = \Upsilon(cons_i, other_i)$. The fee is $fee = \sum_{i=d-N}^{d} price_i$. $\mathcal{F}_{\text{PSM}}$ sends $(\mathsf{payment}, fee, N)$ to U and, if U is corrupted, $\mathcal{F}_{\text{PSM}}$ receives (pay, fee', N'). If $fee \neq fee'$ or $N \neq N'$, $\mathcal{F}_{\text{PSM}}$ sets $fee = fee'$, $N = N'$ and $b = 0$, and otherwise it sets $b = 1$. $\mathcal{F}_{\text{PSM}}$ sends (pay, fee, N, b) to P.
- On input (reveal, i) from P, $\mathcal{F}_{\text{PSM}}$ checks that $i \in [0, d]$, sends (reveal, i) to U and picks the entry $(i, cons, other) \in T$. If U is corrupted, $\mathcal{F}_{\text{PSM}}$ receives $(\mathsf{revresp}, cons', other')$ and, if $(cons', other')$ does not equal those in $(i, cons, other) \in T$, then $\mathcal{F}_{\text{PSM}}$ sets $cons = cons'$, $other = other'$ and $b = 0$. Otherwise it sets $b = 1$. $\mathcal{F}_{\text{PSM}}$ sends $(\mathsf{revresp}, cons, other, b)$ to P.

Figure 2. The $\mathcal{F}_{\text{PSM}}$ functionality defining the security properties of our scheme.

i.e., the verifier learns nothing but the truth of the statement. More technically, for all possible verifiers there exists a simulator that, without knowledge of the secret input, yields a distribution that cannot be distinguished from the interaction with a real prover. Witness indistinguishability is a weaker property that requires that the proof does not reveal which witness (among all possible witnesses) was used by the prover.

We use several existing results to prove statements about discrete logarithms: proof of knowledge of a discrete logarithm [24]; proof of knowledge of the equality of some element in different representations [25]; proof with interval checks [26], range proof [27] and proof of the disjunction or conjunction of any two of the previous [28]. 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 [29].

When referring to the proofs above, we follow the notation introduced by Camenisch and Stadler [30] 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 $y, g_0, g_1, \tilde{y}, \tilde{g_0}, \tilde{g_1}$ 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 $\mathsf{NIPK}\{(x, s_x) : \mathsf{Verify}(pk, x, s_x) = \text{accept}\}$.

VI. Construction

A. Intuition Behind Our Construction

We consider a setting with the entities presented in Section III, i.e., a meter M, a user U and a provider P. After M is installed at U's side, no communication between M and P is possible.[3] Consequently, P communicates with U to bill U's consumption, and, if permitted by U, to learn consumption data.

Every entity computes a key pair of a signature scheme, stores the secret key and reveals the public key to the other entities. P also computes the parameters of a commitment scheme and reveals them to U and to M.

At the initialization phase, P chooses a pricing policy $\Upsilon : (cons, other) \rightarrow price$ that maps consumption values to prices. The variable $other$ denotes any other parameter that influences the price to be paid, e.g. time of day. The policy Υ is signed and sent to U. We note that P can update the policy later on by sending a new signed policy to U.

During a billing period, M obtains consumption values $cons$ and outputs tuples $(d, cons, other)$, where d is a counter initialized at 0 that is incremented each time M outputs a new tuple. These tuples are signed as follows. First, M commits to $cons$ and to $other$, and then computes signatures sc on the commitments and on d. U is given the message-signature pairs and the openings of the commitments.

At the end of a billing period, U stores the signed policy given by P and a set of tuples $(d, cons, other)$ signed by M. Armed with these signatures, U is able to reveal the total fee fee to P and prove that fee is correct without disclosing any information about the tuples $(cons, other)$. Basically, U

[3]The meter here abstracts the metrological unit producing readings. In practice functions of the meter that have no access to readings can communicate with the Provider freely.

reveals to P the signatures sc by M on the commitments to $cons$ and $other$. For each signature, U computes:

1) a commitment to the price $price$ to be paid according to Υ;
2) a non-interactive zero-knowledge proof π that she knows (a) the openings of the signed commitments, (b) the opening of the commitment to the price, and (c) that she possesses a signature on a policy $(cons, other, price)$ that states that $price$ is the price to be paid for $(cons, other)$.

Additionally, U aggregates all the openings of the commitments to the prices to obtain an opening $open_{fee}$ to the total fee. U sets a payment message that contains fee, $open_{fee}$ and, for each signature sc, a commitment to the price and a proof π, and signs the payment message.

Upon receiving the payment message, P verifies the signature by U on the payment message, and the signatures by M on the commitments to $(cons, other)$ and on d. P also verifies the proofs π. Then P, using the homomorphic property of the commitment scheme, aggregates all the commitments to the prices to get a commitment to fee and checks if $(fee, open_{fee})$ is a valid opening for it. The counter d is used by P to check that U reports all the signatures output by M.

P can also ask U to reveal some $(cons, other)$ tuples. If U allows P to learn this information, U reveals P the openings of some commitments to $(cons, other)$.

Intuitively, this protocol provides all the security properties required in $\mathcal{F}_{\text{PSM}}$. P is only given the total fee, but he is assured that the fee is correct in accordance with the pricing policy Υ and the consumption values output by M. U's privacy relies on the hiding property of commitments and on the zero-knowledge property of proofs, while P's security on the binding property of the commitment scheme and on the unforgeability of the signature schemes employed by P and M. Additionally, P cannot claim that U must pay a fee other than the one reported thanks to the unforgeability property of U's signature scheme. Finally, the binding property ensures that U cannot reveal to P $(cons, other)$ tuples different from the ones committed to and signed by M.

B. Construction

In the following, we denote the signature schemes used by M, U and P as $(\text{Mkeygen}, \text{Msign}, \text{Mverify})$, $(\text{Ukeygen}, \text{Usign}, \text{Uverify})$ and $(\text{Pkeygen}, \text{Psign}, \text{Pverify})$ respectively. H stands for a collision-resistant hash function.

In the setup phase, M runs $\text{Mkeygen}(1^k)$ to obtain a key pair $(sk_{\text{M}}, pk_{\text{M}})$, U runs $\text{Ukeygen}(1^k)$ to get a key pair $(sk_{\text{U}}, pk_{\text{U}})$ and P runs $\text{Pkeygen}(1^k)$ to get a key pair $(sk_{\text{P}}, pk_{\text{P}})$. Each party registers its public key with $\mathcal{F}_{\text{REG}}$ and retrieves public keys from other parties by querying $\mathcal{F}_{\text{REG}}$. P runs $\mathsf{ComSetup}(1^k)$ to get par_c and a trapdoor td, computes a proof $\pi = \mathsf{NIPK}\{(td) : (par_c, td) \leftarrow \mathsf{ComSetup}(1^k)\}$ and sends (par_c, π) to U and (par_c) to M. U verifies π.

The outline of our concrete construction is presented in Figure 4, and the functions called are detailed below.

- $\mathsf{SignPolicy}(sk_{\text{P}}, \Upsilon)$. For each tuple $(cons, other, price) \in \Upsilon$, compute $\text{sp} = \text{Psign}(sk_{\text{P}}, \langle cons, other, price\rangle)$. (The way the tuples $(cons, other, price)$ are signed depends on the particular policy Υ to be signed. Examples are given in Section VI-C.) Let $\Upsilon_s = (cons_i, other_i, price_i, \text{sp}_i)_{i=1}^n$ be the set of message-signature tuples. Output Υ_s.
- $\mathsf{VerifyPolicy}(pk_{\text{P}}, \Upsilon_s)$. For $i = 1$ to n, parse Υ_s as $(cons_i, other_i, price_i, \text{sp}_i)_{i=1}^n$, and, for $i = 1$ to n, run $\text{Pverify}(pk_{\text{P}}, \text{sp}_i, \langle cons_i, other_i, price_i\rangle)$. If any of the outputs is reject, output $b = 0$, and otherwise output $b = 1$.
- $\mathsf{SignConsumption}(sk_{\text{M}}, par_c, cons, other, d_{\text{M}})$. Execute both $(c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons)$ and $(c_{other}, open_{other}) = \mathsf{Commit}(par_c, other)$. Run $\text{sc} = \text{Msign}(sk_{\text{M}}, \langle d_{\text{M}}, c_{cons}, c_{other}\rangle)$ and output $\text{SC} = (d_{\text{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \text{sc})$.
- $\mathsf{VerifyConsumption}(pk_{\text{M}}, par_c, \text{SC}, d_{\text{U}})$. Parse message SC as $(d_{\text{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \text{sc})$. Compute $\mathsf{Open}(par_c, c_{cons}, cons, open_{cons})$ and $\mathsf{Open}(par_c, c_{other}, other, open_{other})$ and output $b = 0$ if any of them outputs reject. Run $\text{Mverify}(pk_{\text{M}}, \text{sc}, \langle d_{\text{U}}, c_{cons}, c_{other}\rangle)$ and output $b = 0$ if the output is reject. Otherwise output $b = 1$.
- $\mathsf{Pay}(sk_{\text{U}}, par_c, \Upsilon_s, T)$. For each entry $(d_{\text{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \text{sc}) \in T$, calculate $price = \Upsilon(cons, other)$, run $(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price)$ and calculate a non-interactive witness-indistinguishable proof π:[4]

$$\begin{aligned}\mathsf{NIPK}\{\ &(price, open_{price}, cons, open_{cons}, other,\\ &open_{other}, \text{sp}) :\\ &(c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge\\ &(c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge\\ &(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price) \wedge\\ &\text{Pverify}(pk_{\text{P}}, \text{sp}, \langle cons, other, price\rangle) =\\ &\mathsf{accept}\}.\end{aligned}$$

Let N be the number of entries in T. Compute the total fee $fee = \sum_{i=1}^{N} price_i$ and add all the openings $open_{fee} = \sum_{i=1}^{N} open_{price_i}$ to get an opening to the commitment to the fee. Set a payment message $p = (fee, open_{fee}, \{\text{sc}_i, d_{\text{M}_i}, c_{cons_i},$

[4]The proof π also depends on the policy Υ. See Section VI-C.

Protocol PSM

- **Initialization.** When P is activated with $(\mathsf{policy}, \Upsilon)$, P runs $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$ to get a signed policy Υ_s. P sends Υ_s to U. U runs $\mathsf{VerifyPolicy}(pk_{\mathrm{P}}, \Upsilon_s)$ to get a bit b. If $b = 0$, U rejects the policy. Otherwise U stores Υ_s.
- **Consumption.** When M is activated with $(\mathsf{consume}, cons, other)$, M increments a counter d_{M} (initialized at zero) and runs $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, cons, other, d_{\mathrm{M}})$ to obtain a signed consumption SC. M sends (SC) to U. U increments a counter d_{U} and runs $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, \mathrm{SC}, d_{\mathrm{U}})$ to obtain a bit b. If $b = 0$, U rejects SC and sends P a message indicating malfunctioning meter. Otherwise U appends SC to a table T that stores all the consumptions.
- **Payment.** When P is activated with $(\mathsf{payment})$, P sends $(\mathsf{payment})$ to U. Let N be the number of $(\mathsf{consume}, \ldots)$ messages received by U since the previous message $(\mathsf{payment})$ was received. U runs $\mathsf{Pay}(sk_{\mathrm{U}}, par_c, \Upsilon_s, T[d_{\mathrm{U}} - N : d_{\mathrm{U}}])$ to obtain a payment message Q and sends (Q) to P. P runs $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, \mathrm{Q}, d_{\mathrm{P}})$ to obtain (b, d'_{P}). If $b = 0$, P rejects the payment, and otherwise accepts it and sets $d_{\mathrm{P}} = d'_{\mathrm{P}}$.
- **Reveal.** When P is activated with (reveal, i), P checks that $i \in [0, d_{\mathrm{P}}]$ and sends (i) to U. U runs $\mathsf{Reveal}(sk_{\mathrm{U}}, T, i)$ to get an opening message R and sends (R) to P. P picks the payment message Q that contains i and runs $\mathsf{VerifyReveal}(pk_{\mathrm{U}}, par_c, \mathrm{Q}, R, i)$ to get a bit b. If $b = 0$, P sends $(\mathsf{reject}, \mathrm{Q}, R)$ to U, and otherwise it sends (accept) to U.

Figure 4. The concrete implementation of privacy-friendly metering.

$c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^{N}$). Compute a signature[5] $s_p = \mathsf{Usign}(sk_{\mathrm{U}}, p)$ and set a payment message $\mathrm{Q} = (p, s_p)$.

- $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, \mathrm{Q}, d_{\mathrm{P}})$. Parse Q as (p, s_p) and run $\mathsf{Uverify}(pk_{\mathrm{U}}, s_p, p)$. Output $b = 0$ if it rejects. Otherwise parse p as $(fee, open_{fee}, \{\mathrm{sc}_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^{N})$ and, for $i = 1$ to N, increment d_{P}, run $\mathrm{Mverify}(pk_{\mathrm{M}}, \mathrm{sc}_i, \langle d_{\mathrm{P}}, c_{cons_i}, c_{other_i}\rangle)$ and verify π_i. Output $b = 0$ if any of the signatures or the proofs is not correct. Add the commitments to the prices $c'_{fee} = \otimes_{i=1}^{N} c_{price_i}$ and execute $\mathsf{Open}(par_c, c'_{fee}, fee, open_{fee})$. If the output is accept, set $b = 1$ and otherwise $b = 0$. Output (b, d_{P}).
- $\mathsf{Reveal}(sk_{\mathrm{U}}, T, i)$. Pick the tuple $r = (i, cons, open_{cons}, other, open_{other})$ in the entry $(i, \ldots) \in T$, sign $s_r = \mathsf{Usign}(sk_{\mathrm{U}}, r)$ and output $R = (r, s_r)$.
- $\mathsf{VerifyReveal}(pk_{\mathrm{U}}, par_c, \mathrm{Q}, R, j)$. Parse Q as (p, s_p) and p as $(fee, open_{fee}, \{\mathrm{sc}_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^{N})$. Pick the tuple $(\mathrm{sc}_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i)$ such that $d_i = j$. Parse R as (r, s_r) and r as $(i, cons, open_{cons}, other, open_{other})$. Run algorithms $\mathsf{Open}(par_c, c_{cons_i}, cons, open_{cons})$ and $\mathsf{Open}(par_c, c_{other_i}, other, open_{other})$. If both algorithms output accept, output $b = 1$ and otherwise $b = 0$.

C. Policies

We detail here the computation of the signed policy Υ_s and the proof π, which proves that the price for a tuple $(cons, other)$ is computed in accordance with the right tuple $(cons, other, price)$ specified in the policy Υ. They are carried out depending on different types of policies Υ. We provide details of two policies: a linear policy that can be used to apply a different rate to each measurement and a "cumulative" policy that allows the application of a non-linear function to measurements to calculate their contribution to the bill. The linear policy implements the tariff policy for smart-metering, where a different rate is applied per half-hour according to the policy a customer has subscribed to. The cumulative policy illustrates the generality of the scheme.

Three further policies we considered are the discrete policy, that looks up a tariff in a table, and the interval policy that charges a fixed rate per different ranges of consumption. These are special cases of the cumulative policy or linear policy, which can be implemented more efficiently, and we do not discuss them in detail. A more generic construction for building and proving the application of non-linear functions using splines is also described. It is worth noting that all these pricing policies can be combined or chained together to engineer complex composite policies, e.g. applying a different non-linear policy according to aggregate consumption of each day, and subtracting from the final bill a rebate of 10% if the total units of consumption exceeds a threshold.

1) Linear Policy: A linear policy specifies the price per unit consumption for different contexts. For instance, if the policy says that the price per unit is 3 and your consumption is 6, the price due is 18. Therefore, since a linear policy specifies the price per unit of consumption, it is given by $\Upsilon : other \rightarrow price$. The parameter $other$ denotes any variable that influences the price per unit, e.g., the time interval in which the consumption takes place.

[5] If p does not belong to the message space of the signature scheme, sign $H(p)$, where H is a collision resistant hash function whose range is the message space of the signature scheme.

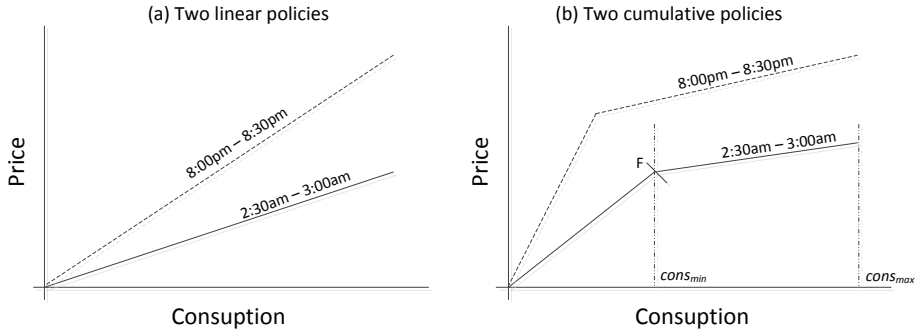

Figure 5. (a) A linear policy specifies the rate per unit consumption that is applied to determine the price to be paid for each measurement. The rate can be selected through information associated with the reading, like the time of the day or the location of a vehicle (without revealing this information). (b) A cumulative policy specifies a rate per unit that is determined as a function of the hidden consumption – allowing non linear functions to be applied for pricing. Higher order polynomials can be used to express pricing functions within intervals allowing pricing functions that can be expressed as arbitrary splines.

To sign this policy, for $i = 1$ to n, P runs $\mathrm{sp}_i = \mathsf{Psign}(sk_\mathrm{P}, \langle other_i, price_i \rangle)$, and sets $\Upsilon_s = (other_i, price_i, \mathrm{sp}_i)_{i=1}^n$. To compute a proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the total price $price_t$ ($(c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t)$). (The total price equals $price_t = price \cdot cons$, where $price = \Upsilon(other)$.) Then U computes a proof of possession of a signature $\mathrm{sp} \in \Upsilon_s$ on $(other, price)$, a proof of equality between $other$ and the values committed to in c_{other}, and a proof that $price_t$ committed to in c_{price_t} equals $price \cdot cons$:

$$
\begin{aligned}
\mathsf{NIPK}\{\ & (price_t, open_{price_t}, price, cons, open_{cons}, other, \\
& open_{other}, \mathrm{sp}) : \\
& (c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge \\
& (c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge \\
& (c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t) \wedge \\
& \mathrm{Pverify}(pk_\mathrm{P}, \mathrm{sp}, \langle other, price \rangle) = \mathsf{accept} \wedge \\
& price_t = price \cdot cons\}.
\end{aligned}
$$

2) Cumulative Policy: A cumulative policy allows the computation and proof in zero-knowledge of non linear functions. It can be used to apply different rates according to the hidden consumption, expressing rates getting cheaper or more expensive as consumption rises.

To apply the cumulative policy, the consumption values domain is divided into intervals and each interval is mapped to a rate per unit of consumption. The price due is the definite integral of the policy Υ over the interval $[0, cons]$. For instance, let Υ be a policy as follows[6]: $[0, 3] \to 2$, $(3, 7] \to 5$, $(7, \infty) \to 8$, and let your consumption be 9. The price due is $3 \times 2 + 4 \times 5 + 2 \times 8 = 42$. Therefore, a cumulative policy is given by $\Upsilon : (cons_{min}, cons_{max}, F, other) \to price$, where it is required that intervals defined by $[cons_{min}, cons_{max}]$ be disjoint. F is the definite integral of Υ over the interval $[0, cons_{min}]$.

To sign this policy, for $i = 1$ to n, P runs $\mathrm{sp}_i = \mathsf{Psign}(sk_\mathrm{P}, \langle cons_{min_i}, cons_{max_i}, F_i, other_i, price_i \rangle)$, and sets $\Upsilon_s = (cons_{min_i}, cons_{max_i}, F_i, other_i, price_i, \mathrm{sp}_i)_{i=1}^n$. In the previous example, the tuples to be signed are $(0, 3, 0, \perp, 2)$, $(3, 7, 6, \perp, 5)$ and $(7, \max, 26, \perp, 8)$ (max represents the maximum consumption). To compute a proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the price $price_t$ ($(c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t)$) to be paid, which equals $price_t = (cons - cons_{min}) \times price + F$). Then U computes a proof of possession of a signature $\mathrm{sp} \in \Upsilon_s$ on $(cons_{min}, cons_{max}, F, other, price)$, a proof of equality between $(other)$ and the value committed to in c_{other}, a proof that $cons \in [cons_{min}, cons_{max}]$ and a proof that $price_t = (cons - cons_{min}) \times price + F$:

$$
\begin{aligned}
\mathsf{NIPK}\{\ & (price_t, open_{price_t}, cons, open_{cons}, other, open_{other}, \\
& price, cons_{min}, cons_{max}, F, \mathrm{sp}) : \\
& (c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge \\
& (c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge \\
& (c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t) \wedge \\
& \mathrm{Pverify}(pk_\mathrm{P}, \mathrm{sp}, \langle cons_{min}, cons_{max}, F, other, \\
& price \rangle) = \mathsf{accept} \wedge \\
& cons \in [cons_{min}, cons_{max}] \wedge \\
& price_t = (cons - cons_{min}) \times price + F\}.
\end{aligned}
$$

[6]The parameter $other$ is left unused, in this example, but can in general be used to select the rate.

3) Other Policies: Another possible policy Υ is that defined by a polynomial function $\sum_{i=0}^{N} a_i x^i$ over a commutative ring R, which in our implementation is given by the integers modulo a composite (see Section A). The price due is the evaluation of Υ on input the consumption $x = cons$.

Let n be the number of polynomials that define the policy (e.g., each of them associated with a different parameter $other$). To sign this policy, for $i = 1$ to n, P runs $\mathrm{sp}_i = \mathsf{Psign}(sk_\mathrm{P}, \langle a_{Ni}, \ldots, a_{0i}, other_i \rangle)$, and sets $\Upsilon_s = (a_{Ni}, \ldots, a_{0i}, other, \mathrm{sp}_i)_{i=1}^{n}$. To compute a proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the price $price_t$ $((c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t))$ to be paid, which equals $price_t = \sum_{i=0}^{N} a_i cons^i$. Then U computes a proof of possession of a signature $\mathrm{sp} \in \Upsilon_s$ on $(a_N, \ldots, a_0, other)$, a proof of equality between $(other)$ and the value committed to in c_{other}, and a proof that $price_t = \sum_{i=0}^{N} a_i cons^i$:

$$
\begin{aligned}
\mathsf{NIPK}\{\ & (price_t, open_{price_t}, cons, open_{cons}, other, \\
& open_{other}, \mathrm{sp}) : \\
& (c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge \\
& (c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge \\
& (c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t) \wedge \\
& \mathsf{Pverify}(pk_\mathrm{P}, \mathrm{sp}, \langle a_N, \ldots, a_0, other \rangle) = \mathsf{accept} \wedge \\
& price_t = \sum_{i=0}^{N} a_i cons^i \}.
\end{aligned}
$$

The combination of the polynomial policy and the cumulative policy allows the evaluation and proof of arbitrary polynomial segments. Therefore complex policies expressed as polynomial splines can be used for pricing or any other calculation in zero-knowledge.

D. Efficient Construction for public linear policies

In the previous construction, the policy Υ consists of several formulas that map consumption values to prices. The formula that should be applied to a particular tuple $(cons, other)$ depends on the consumption $cons$, on the other parameters $other$, or on both. Therefore, the formula used to compute the fee needs to be hidden from P, because otherwise P can learn some information on $(cons, other)$ (consider the example policy "if consumption > 10 then fee = consumption * 5". The rate (5) has to be hidden to hide the fact that the consumption exceed the threshold).

However, if the choice of formula depends on parameters already known by P, then the formula used does not need to be hidden. In this case, when Υ consists of linear formulas of the form $price = a_1 \cdot cons + a_0$, we provide an efficient construction that avoids the use of non-interactive zero-knowledge proofs. This construction is based on the use of a commitment scheme provided with two operations $\otimes$ and $\odot$ (see Section V) that allow to compute a commitment to the price given a commitment to the consumption value.

The protocol is described in Figure 6, and the functions called are detailed below.

- $\mathsf{EffPay}(sk_\mathrm{U}, par_c, id_\Upsilon, \Upsilon, T)$. For each table entry $(d_\mathrm{M}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \mathrm{sc}) \in T$, calculate $price = a_1 \cdot cons + a_0$ and $open_{price} = open_{cons} \cdot a_1$. Let N be the number of entries in T. Compute the total fee $fee = \sum_{i=1}^{N} price_i$ and add all the openings $open_{fee} = \sum_{i=1}^{N} open_{price_i}$ to get an opening to the commitment to the fee. Set a payment message $p = (id_\Upsilon, fee, open_{fee}, \{\mathrm{sc}_i, d_{\mathrm{M}_i}, c_{cons_i}, c_{other_i}\}_{i=1}^{N})$. Compute a signature[7] $s_p = \mathsf{Usign}(sk_\mathrm{U}, p)$ and set a payment message $\mathrm{Q} = (p, s_p)$.
- $\mathsf{EffVerifyPayment}(pk_\mathrm{M}, pk_\mathrm{U}, par_c, id_\Upsilon, \mathrm{Q}, d_\mathrm{P})$. Parse Q as (p, s_p) and run $\mathrm{Uverify}(pk_\mathrm{U}, s_p, p)$. Output $b = 0$ if it rejects. Otherwise parse p as $(id_\Upsilon{}', fee, open_{fee}, \{\mathrm{sc}_i, d_i, c_{cons_i}, c_{other_i}\}_{i=1}^{N})$, check that $id_\Upsilon = id_\Upsilon{}'$ and, for $i = 1$ to N, increment d_P and run $\mathrm{Mverify}(pk_\mathrm{M}, \mathrm{sc}_i, \langle d_\mathrm{P}, c_{cons_i}, c_{other_i} \rangle)$. Output $b = 0$ if any of the signatures or the proofs is not correct. Compute commitments to the prices $c_{price_i} = (c_{cons_i} \odot a_1) \otimes \mathsf{Commit}(par_c, a_0, 0)$, add them $c_{fee} = \otimes_{i=1}^{N} c_{price_i}$ and execute $\mathsf{Open}(par_c, c_{fee}, fee, open_{fee})$. If the output is accept, set $b = 1$ and otherwise $b = 0$. Output (b, d_P).

Security of this scheme relies on the unforgeability of the signature schemes and on the binding and hiding properties of the commitment schemes. The policy identifier id_Υ is introduced to ensure that U and P employ the policy published previously by P to compute and verify the payment message.

E. Discussion

We discuss here possible optimizations of the scheme, as well as modifications needed when it is applied to certain application scenarios and when the adversary corrupts more than one party. We also propose possible extensions.

1) Optimizations: In the construction depicted above, M commits separately to $cons$ and to $other$. This is done in order to allow U to selectively disclose either one value or the other to P in the reveal phase. However, in applications where both parameters are always disclosed together or where the reveal phase never takes place, M can commit to both values in a single commitment (see the commitment scheme we employ in Section VII-B) in order to improve efficiency.

Additionally, in the construction above, U, for each tuple $(cons, other)$ output by M, computes a commitment to the price to be paid and a proof that this price is correct. To prove that the total fee is the sum of all the committed

[7] If p does not belong to the message space of the signature scheme, sign $H(p)$, where H is a collision resistant hash function whose range is the message space of the signature scheme.

Protocol Fast-PSM

- **Initialization.** When P is activated with $(\mathsf{policy}, \Upsilon)$, where Υ is a linear policy, P publishes a unique policy identifier id_Υ and sends (id_Υ, Υ) to U.
- **Consumption.** It works as the previous construction.
- **Payment.** When P is activated with $(\mathsf{payment})$, P sends $(\mathsf{payment})$ to U. Let N be the number of $(\mathsf{consume}, \ldots)$ messages received by U since the previous message $(\mathsf{payment})$ was received. U runs $\mathsf{EffPay}(sk_\mathrm{U}, par_c, id_\Upsilon, \Upsilon, T[d_\mathrm{U} - N : d_\mathrm{U}])$ to obtain a payment message Q and sends (Q) to P. P runs $\mathsf{EffVerifyPayment}(pk_\mathrm{M}, pk_\mathrm{U}, par_c, id_\Upsilon, \mathrm{Q}, d_\mathrm{P})$ to obtain (b, d'_P). If $b = 0$, P rejects the payment, and otherwise accepts it and sets $d_\mathrm{P} = d'_\mathrm{P}$.
- **Reveal.** It works as the previous construction.

Figure 6. The fast protocol for billing in smart-grids.

prices, U provides P with the sum of the openings of all the commitments. Computing a commitment and a proof for each tuple $(cons, other)$ is done mainly to allow U to start the computation of the payment from the beginning of the billing period, when the total fee is unknown. Nevertheless, in applications in which the computation of the payment message can be delayed until all the tuples $(cons, other)$ are known by U, it is possible to avoid the computation of the commitments to prices and of one proof of knowledge per tuple. Instead, it suffices to compute only one zero-knowledge proof of knowledge per payment message. This proof should prove that the sum of the prices to be paid for each $(cons, other)$ tuple equals the total fee.

2) Modifications: We note that the scheme described above only works when P knows the amount of tuples that M outputs at each billing period. Otherwise, U can report less tuples in order to pay less. That is suitable in applications like electricity metering, where commonly M outputs $(cons, other)$ tuples periodically. However, this may not be the case in other applications. To solve this problem, one possibility is to require M to output, at the end of the billing period, a signature on the amount of tuples that were output at that period. This signature must be reported by U to P.

We also note that we prove the construction secure when only one party is corrupted (see Section A), and thus not when two parties collude against the remaining one. A collusion between U and P against M is meaningless, and a collusion between M and U against P is avoided by the fact that we assume meters to be tamper-resistant. A collusion between M and P against U is possible if both parties where already corrupted at the setup phase (later on no communication between them is allowed). Such collusion makes sense in practical applications, in which P is likely to provide the meters and thus can manipulate them at the setup phase. Our construction fails to protect U against such collusion. For example, P can choose the seed of the pseudorandom number generator of M in order to know later the openings of the commitments computed by M.

Nevertheless, the construction can be modified in order to protect U against such a collusion. In the modified construction, M outputs signatures on the tuples $(d, cons, other)$ and does not compute any commitment. Then, U, instead of revealing the signature to P, commits to $(cons, other)$ and computes a non-interactive zero-knowledge proof of possession of a signature by M on messages $(d, cons, other)$ (d is disclosed to P). This proof is conjuncted with the proof of possession of a signature on $(cons, other, price)$ given in the signed policy Υ_s. In this modified construction, thanks to the zero-knowledge property of proofs and to the hiding property of commitments (computed with randomness chosen by U), no information is revealed to P.

Finally, we note that in the construction described above, the pricing policy Υ is sent by P to U at the initialization phase and later it is not updated. To update the policy and ensure that U only employs the new policy to compute the payment message, one possibility is that P generates a new key pair each time a new policy needs to be signed. Therefore, since a different public key of P will be used to verify the payment message, U must use the new policy.

3) Extensions: In our construction, U reveals the total fee to P and is assumed to pay the amount *fee* through an arbitrary payment channel. While revealing the total fee to P is not a privacy threat in most cases, there can be applications in which U may also want to hide the total fee. To this end, a prepaid mechanism like the one used in priced oblivious transfer can be applied [31]. The basic idea is as follows. First, U pays an initial deposit to P through an arbitrary payment channel. To compute a payment message, U commits to the new value of the deposit, i.e., the old one minus the total fee of that billing period, and proves in zero-knowledge that the committed value is a correct update of the deposit and that it is non-negative, so that P can check that U still has enough funds.

VII. Evaluation

A. Security

The security of protocol PSM is analyzed by proving indistinguishability between the view of the environment $\mathcal{Z}$ in the real world, where parties interact following the protocol description in the presence of a real adversary $\mathcal{A}$, and in the ideal world, which is secure by definition since an ideal functionality carries out the task. In order to prove indistinguishability, for all real world adversaries $\mathcal{A}$, we construct an ideal world adversary $\mathcal{E}$ such that no environment can distinguish whether it is interacting with $\mathcal{A}$ or with $\mathcal{E}$.

We divide the proof of Theorem **??** into several claims. Each of the claims proves indistinguishability when a different subset of parties is corrupted. (We prove security under static corruptions.) We do not consider the cases where all the parties are honest, where all the parties are dishonest, where the user U and the provider P are dishonest or when only the meter M is dishonest because they do not have practical interest. The case in which both U and M are dishonest is not possible because we assume tamper-resistant meters. To prove security when P and M are dishonest, we need to modify protocol PSM as described in Section VI-E.

For each of the claims, we prove indistinguishability between real and ideal worlds by defining a sequence of hybrid games **Game** $0, \ldots,$ **Game** n, where **Game** 0 corresponds to the real word and **Game** n to the ideal world. We denote by Pr [**Game** i] the probability that $\mathcal{Z}$ distinguishes between the distribution ensemble of **Game** i and that of the real execution. Therefore, |Pr [**Game** $i+1$] − Pr [**Game** i]| denotes the probability that $\mathcal{Z}$ distinguishes between the distribution ensembles of two consecutive games. By summing up all those probabilities, we obtain an upper bound for the probability that $\mathcal{Z}$ distinguishes between the real execution ensemble $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ and the ideal ensemble $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$. Such upper bound should be negligible in the security parameter for the claim to hold.

When P is dishonest, we claim and prove indistinguishability between real and ideal world under the unforgeability of the signature schemes $(\mathrm{Mkeygen}, \mathrm{Msign}, \mathrm{Mverify})$ and $(\mathrm{Ukeygen}, \mathrm{Usign}, \mathrm{Uverify})$, under the hiding property of the commitment scheme and the extractability and witness-indistinguishability of proofs of knowledge. Intuitively, the proof of this claim ensures that P is not able to get any information from U except the total fee and the number of consumption readings, and that P is not able to claim that U must pay a fee different from the one calculated on input the consumption readings and the pricing policy.

When U is dishonest, we prove indistinguishability under the unforgeability of the signature schemes $(\mathrm{Mkeygen}, \mathrm{Msign}, \mathrm{Mverify})$ and $(\mathrm{Pkeygen}, \mathrm{Psign}, \mathrm{Pverify})$, under the binding property of the commitment scheme and under the extractability and zero-knowledge property of proofs of knowledge. Intuitively, this proof ensures that the total fee calculated by U is correct, i.e., it is computed following the pricing policy on input the consumption readings output by M.

B. Efficient Instantiation

We propose an efficient instantiation for the commitment scheme, for the signature schemes used by M, U and P, and for the non-interactive proofs of knowledge that are used in the construction described in Section VI.

1) Commitment Scheme: We choose the integer commitment scheme due to Groth [32].

2) Signature Schemes: The signature schemes of M and U can be instantiated with any existentially unforgeable signature scheme. For P's signature scheme, we choose the signature scheme proposed by Camenisch and Lysyanskaya [33].

3) Non-Interactive Zero-Knowledge Proof: We describe the basic buildings block that compose the non-interactive zero-knowledge proofs utilized in our construction in Section VI. Such non-interactive zero-knowledge proofs consist of the conjunction of some of those building blocks. The basic building blocks are a non-interactive zero-knowledge proof of possession of a Camenisch-Lysyanskaya signature, a proof that a committed value is the product of two committed values and a proof that a committed value lies in an interval.

To prove possession of a Camenisch-Lysyanskaya signature, we employ the proof described in [34]. To prove that a message m_3 committed to in $c_{m_3} = g_1^{m_3} h^{open_{m_3}}$ is the product of two messages m_1 and m_2 committed to in $c_{m_1} = g_1^{m_1} h^{open_{m_1}}$ and $c_{m_2} = g_1^{m_2} h^{open_{m_2}}$ respectively, the following proof can be used:

$$\begin{aligned}\mathsf{NIPK}\{\ & (m_1, open_{m_1}, m_2, open_{m_2}, m_3, open_{m_3}, \\ & m_2 \cdot open_{m_1}) : c_{m_1} = g_1^{m_1} h^{open_{m_1}} \wedge \\ & c_{m_2} = g_1^{m_2} h^{open_{m_2}} \wedge c_{m_3} = g_1^{m_3} h^{open_{m_3}} \wedge \\ & 1 = c_{m_1}^{m_2} (1/g_1)^{m_3} (1/h)^{m_2 \cdot open_{m_1}}\}.\end{aligned}$$

To prove that a committed value x lies in an interval $[a, b]$, it is necessary to prove that $x - a \geq 0$ and $b - x \geq 0$. We employ the non-interactive zero-knowledge proof due to Groth [32] to prove that an integer $m \geq 0$.

C. Performance

To assess the performance of the proposed protocols we implemented all the functionality required to generate keys, policies, prove bills and verify bills. The C++ implementation of the system spans 8200 lines of code, including 1000 lines for interfacing with big number libraries and 800 lines to implement fast exponentiation using pre-computation tables. The proof libraries provide generic support for expressing computations on certified inputs, generate and verify proofs of the correctness of their results. The smart-metering

Generic Protocol	1024 bits		2048 bits	
(per reading)	ticks	sec^{-1}	ticks	sec^{-1}
Gen. policy	147522	(97.0579/s)	489754	(29.2355/s)
Prove bill	162816	(87.9409/s)	586586	(24.4093/s)
Verify bill	344703	(41.5377/s)	1270456	(11.2701/s)

Table I
GENERIC PROTOCOL TIMINGS. POLICY GENERATION PER LINE OF POLICY (AMORTISED OVER 100 LINES). PROOF AND VERIFICATION PER READING (AMORTISED AND AVERAGED OVER 1000 READINGS).

Fast Protocol	1024 bits		2048 bits	
(per reading)	ticks	sec^{-1}	ticks	sec^{-1}
Prove bill	48 ticks	(298295/s)	59 ticks	(242681/s)
Verify bill	158 ticks	(90621.4/s)	504 ticks	(28409.1/s)

Table II
FAST PROTOCOL TIMINGS. PROOF AND VERIFICATION PER READING (AMORTISED AND AVERAGED OVER 1000 READINGS). POLICY PACKAGING REQUIRES NO CRYPTOGRAPHY.

Proof size	1024 bits	2048 bits
(for 1000 readings)	KBytes	KBytes
Generic Protocol	$\sim$ 6586 Kb	$\sim$ 10586 Kb
Fast Protocol	$\sim$ 125 Kb	$\sim$ 250 Kb
Fast ECC Protocol (Estimate)	($\sim$ 20 Kb)	

Table III
SIZE IN KILO BYTES REQUIRED TO TRANSMIT THE PROOF ASSOCIATED WITH 1000 METER READINGS IN (A) THE GENERIC PROTOCOL (B) THE FAST PROTOCOL AND (C) AN ELLIPTIC CURVE IMPLEMENTATION OF THE FAST PROTOCOL (ESTIMATE FOR 160 BIT CURVE).

specific code spans about 250 lines of code – including the fast protocol and performance measurement code.

The reference platform for our measurements is a Intel Xeon E5440 running at 2.83GHz (8 cores split over 2 processors) with 32GB Ram running a 64 Bit Windows Server Enterprise operating system. All our experiments were performed on a single core executing 14318180 ticks/s. The reference platform is typical of the systems we expect verifiers to use, but our tick count should be stable over any 64 bit amd64 platform that could be used to prove bills.

Two reference billing problems were considered:

- **Generic Protocol.** A user needs to certify 1000 meter readings, corresponding to approximately 3 weeks of electricity measurements, using a complex cumulative pricing policy. A 100 line policy is certified and the verifier needs to verify the resulting bill. This illustrates a complex billing scenario where a non-linear policy is applied at the finest granularity of readings.
- **Fast Protocol.** A user needs to certify 1000 meter readings, corresponding to approximately 3 weeks of electricity measurements, by applying a linear public policy. This corresponds to the smart-grid billing problem, and we apply our fast protocol for the proof and verification.

For both settings all timings are calculated over 1000 readings and averaged. Two reference security parameters are used, namely a 1024 bit and 2048 bit RSA modulus (all other security parameters are the same as recommended in the implementation section.)

Table I illustrates the time required to create policies, prove and verify bills for the generic protocol setting, while table II illustrates the fast protocol setting. Table III describes the sizes of the bill for different settings.

As expected verifying bills is about twice as slow as generating bills in the generic protocol, due to aggressive pre-computations that are not available to the verifier (the cost of pre-computations is folded into the timing measurements). In real terms it takes from a few seconds to a few minutes to calculate and verify 3 months of billing data depending on the security parameter.

The fast protocol is extremely efficient compared with the generic protocols. Generating bill proofs requires a few tens of ticks since no cryptographic operations are required, only big integer arithmetic. Verifying bills is also extremely fast as the exponentiations only involve very small exponents. In real terms our reference platform could verify the 3 weekly bills of 120 households in just one second if all 8 cores were involved in the verification. A single core could verify 3 weeks of readings from every household equipped with a smart meter in the UK (27 million) in about 12 days, even using the slowest, highest security 2048 bit parameter.

In the generic protocol setting, over 90% of the time is spent in the modular multiplication libraries, which are in turn called by the modular exponentiation libraries. Any performance improvement in those will have a dramatic impact on the performance of the protocols. The measurements shown here were performed using a hand optimised amd64 SSE2 instruction set Montgomery multiplication routine. Faster implementation could be achieved by using hardware co-processors, GPU or FPGA implementations. Such acceleration would be particularly beneficial to the verifier, whereas even our generic protocols are fast enough for provers to process their own bills in real-time (3 weeks of electricity consumption would take 100 seconds to bill under the complex non-linear policy).

Similarly, if proof size is a crucial factor, an elliptic curve could be used to implement the fast protocol to reduce bill sizes for 3 months to about 20 KBytes.

We did not explicitly evaluate the cryptographic load of the meter. It consists of performing one commitment (one short and one long modular exponentiation with fixed generators) to a reading per 15 or 30 minutes, and one signature on a set of commitments per billing period. This is well within the capabilities of cheap, off-the-shelf, somehow tamper resistant smart-cards or microcontrollers.

VIII. Deployment & integration

Smart metering provides a portfolio of functionality, one of which is fine-grained billing of energy usage. A fully fledged smart meter is a complex device, with a full CPU, a display, local or wide area network telecommunications and remote upgrade capabilities. According to the PIA performed by NIST, billing seems to be the key function requiring all detailed readings to be made available to third parties. Other functions, such as load forecasting, efficiency analysis, demand-response can be performed by pushing data to customers or using sampled, anonymized or aggregated readings volunteered by users (or possibly from the aggregate supply head-end meter rather than the customers' meters).

Our scheme requires only a small sub-system, the metrological unit, to ensure the correctness of billing. Current metrological units have to be augmented with the ability to certify readings through commitments and signatures. In practice off-the-shelf smart cards are capable of delivering this functionality at very low cost, and could readily be integrated in new generation smart meters.

Functions of the smart meter not related to consumption measurements and billing can be performed outside the tamper-evident metrological unit. Proposals for modern meters include provisions for updating policies, measuring tamper resistance parameters, testing whether a meter is on-line, switching electricity supply to pre-paid mode, and offering a rich user interface displaying current energy prices. All these functions can be performed as long as they do not interfere with the integrity of the metrological unit that certifies reading, and cannot access raw meter readings directly.

Bills derived from certified readings and policies are guaranteed to be correct through our cryptographic scheme. This allows calculations to be done outside the tamper-evident enclosure, on commodity hardware, mobile devices or even on-line services. The software and platform used to extract certified readings from the meter, and process them into bills has to be trusted by the user to maintain the privacy of raw measurements, and to output only the final bill and its proofs of correctness.

A variety of tariff policies can be applied to the certified readings to calculate the final bill. As the bills are calculated and proved correct outside the meter these changes do not affect the code of the metrology unit in the meter. The tariffs can vary over time, even at a high frequency, and their structure can change as long as the software producing the bill can be updated. In any case the integrity of the billing is guaranteed by the verification process. This property is similar to "software independence" [36] that is sought in electronic election protocols.

Privacy on the other hand relies on the platform and software processing the certified reading to produce the bill without leaking information. The user is free to choose any software package to produce bills on any platform of their choice. They can delegate the calculation to any third party entity they trust with their data, use any vendors product, or even write their own if they wish. Users can switch platform at will. This is in stark contrast with a privacy-invasive architecture where the user has to trust the utility provider with the safekeeping of their consumption data, or even a privacy-friendly architecture that performs calculations in a complex smart meter provided by a fixed third party (often the provider). The flexibility the user has to freely choose any platform to process the bill and protect their own privacy provides the right alignment of incentives to maintain privacy.

Given these issues for privacy there are a few options concerning where the calculation of the final bill and zero-knowledge proofs are done, providing different trade-offs:

- **Smart-meter**: The simple tamper-resistant metrology unit that measures certifies readings is part of a larger smart meter that also computes the final bill and proofs of correctness, and transmits those to the provider. This deployment strategy is closest to the current vision of "smart meters". It is "utility robust" as the full platform is under the control of the utility. This is a key requirement of the UK energy retail industry [37]. The dedicated processors in the smart meter can calculate bills in real-time as measurements are certified by the metrology unity, as we show in our evaluation. On the downside, customers must trust smart meter manufacturers to transmit only the privacy-preserving billing information back to the utility. This is particularly challenging as the full smart meter maybe otherwise fully under the control of the utility. A mixture of privacy regulation and audits could maintain such customer confidence.
- **Home-server**: In case customers are reluctant to trust the smart-meter (beyond the certified metrology unity) with their privacy, the bill calculation can be done on their own hardware, possibly a home server or desktop computer. This is particularly relevant to smart grid deployments that rely on home equipment to provide wide-area network connectivity [37]. While this deployment model is optimal for privacy, as it offers maximal flexibility to the user, it might hinder reliability as it relies on equipment that is not maintained by the utility. Economic incentives to provide timely bills can be provided by overcharging or overestimating projected consumption and rectifying it when certified bills are submitted. Metrology units in the EU are required to store 3 months of detailed reading, and recovering them in case a bill is not submitted on time may be required. Alternatively utility companies may rely on predicting bills and overcharging as the sole means of encouraging

timely delivery of bills.

- **Third-party service**: To improve robustness to failures or denial-of-service, the metrology unit could transmit the signed readings to a third-party provider of the user's choice. The third-party service calculates bills and proofs before transmitting them to the energy provider. Large IT service providers have already launched services to help customers visualise and manage their energy consumption such as Microsoft Hohm[8] and Google Powermeter[9]. Smaller or local services can also be used since the integrity of the final bill does not depend on the provider being honest. Using a third party provider may improve reliability, but requires the user to entrust them with her private data. In this case the third party signs the payment message, including a user identifier in the message for authentication purposes.
- **Smart-devices**: The metrology unit can use a handheld device with mobile connectivity (such as a smart phone) to compute the bill and send it to the provider. This solves the open issues of privacy as well as wide-area connectivity for smart metering infrastructures. A fall back strategy to extract bills, or over prediction, should be considered in case the providers do not receive bills in a timely fashion.

A key consideration when it comes to rolling out smart metering infrastructure is interoperability of meters between providers. Our cryptographic approach to billing has distinct advantages over the traditional approach that requires the full smart meter to be trusted by the provider. The small metrology unit is certified according to existing national and international standards [**?**] to produce readings all providers trust. The rest of the smart meter can act in arbitrary ways without compromising the integrity of the bill computation. Different charging policies can be applied to the output of the simple metering core by different providers without changing anything within the metrology unit. Such a scheme future-proofs the infrastructure: even if new developments occur on how to transmit, display, or bill consumption the same trusted core can be used to certify readings. Functions of the metering infrastructure can migrate to different technologies, and novel business models can be used to perform billing on the same certified readings.

A. Other metering and billing applications

The privacy-preserving metering and billing protocols are generic in so far as they accommodate any situation in which a meter certifies readings that have to be billed according to a policy, without revealing any other information. Many applications aside utility provision require similar functionality.

[8]http://www.microsoft-hohm.com/

[9]http://www.google.com/powermeter/about/

A number of automotive applications can be satisfied:

- **Pay-as-you-drive (PAYD) insurance**: A meter is fixed to a car that records its GPS coordinates, speed, distance travelled, time and date, and provides those as certified readings. A policy maps regions of GPS coordinates to a charging regime per distance travelled to calculate an insurance premium. This provides a full cryptographic solutions to the PAYD problem, with a smaller TCB than the one considered in PriPAYD [18].
- **Road charging and tolling**: As for the PAYD application, a black box in the car records its position and distance travelled. The car position can be used as key to a look-up a table mapping rectangles to the tax premium or toll charge to be paid. Tariffs can change according to the time of day or predicted road congestion incentivising drivers to avoid certain times or roads respectively. Our scheme avoids the need for spot-checks, necessary in [16], and relies instead on the tamper-resistance of the meter for integrity, as long as the user cannot switch it off.

As for utility metering, a number of tweaks can benefit automotive applications: certified readings from an integrated 3D-accelerometer can be used to determine whether the GPS might be jammed or working improperly, to prevent or quickly detect abuse or compromised meters; a more complex metrology core mapping GPS coordinates traces to certified road segments would allow a greater flexibility of efficient charging policies in both cases.

A privacy-preserving infrastructure for metering can also be used for charging use of on-line services, cloud computing resources or software applications. A software meter, implemented using the trusted computing module available on modern PCs, can provide certified measurements of a computing platform. These measurements are made available to the user that computes the bill for particular resources or services used.

A server software provider, for example, can run a hypervisor with an attested software meter that certifies readings of CPU usage, network usage, I/O throughput or any other information. The certified readings are provided to the virtual machines alongside a charging policy, to produce a certified bill.

A similar scheme can be used for charging for virtualised services running on on-premise private clouds, not directly under the control of a cloud provider. Privacy preserving software license billing, as well as digital rights management applications are also within the scope of our algorithms.

In all these cases, infrastructure, software or digital goods consumption can be billed in a very detailed manner without revealing any information about the activities of the users aside the final bill. Furthermore billing is universally verifiable end-to-end, and only depends on the correctness of the software meter. Certifying such meters is unchartered territory for current national metrology authorities.

IX. Conclusion

Privacy is a serious concern raised by smart-metering, and failure to protect it has delayed or even jeopardised smart-grid deployments, as it was the case in The Netherlands. Naively, there seems to be a balance to be struck between the intrusion necessary for billing, and the claimed social benefits of smart-grids. We show this intuition to be false, and present practical privacy-friendly metering systems that do not necessarily leak any information to third parties while providing unforgeable bills based on complex dynamic tariff policies.

Our schemes use simple cryptography on the meters to certify readings, and then perform all other operations outside the metrological unit. This allows further calculations of the bill to be done on any device under the control of the user as well as a smart-meter, if it is trusted. The integrity of the bill is software independent, and the utilities can ensure its correctness through cryptographic verification. Our evaluation ensures the security of the scheme through rigorous security analysis, and its practicality through an implementation of the proof and verification process for common electricity billing as well as more complex settings. It is striking that for the common smart-grid tariff structure the fast version of the protocols could verify 3 weeks of bills of 27 million UK homes in a few days on a single core of a modern PC.

All four proposed deployment strategies include the full benefits of smart metering: fine-grained metering and tariffs, a rich interactive understanding of the current electricity consumption and cost and the ability of devices to access local consumption information. All these can be done through a fancy meter, a local computer, a mobile device or an on-line service. The provider on the other hand can check the integrity of billing, and is able to manage the smart metering functions not related to billing (such as switching meters between debit and credit).

An advantage of the proposed schemes is their flexibility: they offer different options on how to certify meter readings depending on the kind of trust the user is ready to place on the privacy of the meter itself. Furthermore, complex tariff policies can be applied to extract bills from measurements and other associated information without revealing information. Since we allow bill calculations to be performed on any device, the schemes proposed can keep up with changes of tariff structure or policy, as well as changes in technologies for processing or transmission of the readings and bills.

Beyond smart-grids the proposed protocols are applicable to any setting where a bill has to be produced from a set of certified readings and a policy. We have discussed automotive location-based applications as well as applications relating to on-line services. Unlike what the common wisdom dictates, none of them require privacy-invasion for the purpose of billing.

Acknowledgment

...

References

[1] A. Cavoukian, J. Polonetsky, and C. Wolf, "Smartprivacy for the smart grid: embedding privacy into the design of electricity conservation," in *Identity in the Information Society*, 2009, http://www.springerlink.com/content/L4674P732571465H.

[2] R. Anderson and S. Fuloria, "On the security economics of electricity metering," in *The Ninth Workshop on the Economics of Information Security*, 2010.

[3] G. W. Hart, "Nonintrusive appliance load monitoring," in *Proceedings of the IEEE*, December 1992, pp. 1870–1891.

[4] C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong, "Power signature analysis," *Power and Energy Magazine, IEEE*, vol. 1, no. 2, pp. 56–63, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1192027

[5] T. S. G. I. Panel, "Smart grid cyber security strategy and requirements," National Institute of Standards and Technology, Tech. Rep. [Online]. Available: http://csrc.nist.gov/publications/drafts/nistir-7628/draft-nistir-7628_2nd-public-draft.pdf

[6] C. Cuijpers. [Online]. Available: http://vortex.uvt.nl/TILTblog/?p=54

[7] M. Lisovich and S. Wicker, "Privacy concerns in upcoming residential and commercial demand-response systems," in *2008 Clemson University Power Systems Conference*. Clemson University, March 2008. [Online]. Available: http://www.truststc.org/pubs/332.html

[8] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security and Privacy*, vol. 7, pp. 75–77, 2009.

[9] E. L. Quinn, "Privacy and the new energy infrastructure," *SSRN eLibrary*, 2009.

[10] A. Wagner, S. Speiser, O. Raabe, and A. Harth, "Linked data for a privacy-aware smart grid," in *INFORMATIK 2010 Workshop - Informatik fr die Energiesysteme der Zukunft*, 2010.

[11] F. D. Garcia and B. Jacobs, "Privacy-friendly energy-metering via homomorphic encryption," Radboud Universiteit Nijmegen, Tech. Rep., February 2010.

[12] S. McLaughlin, P. McDaniel, and D. Podkuiko, "Energy theft in the advanced metering infrastructure," in *4th International Workshop on Critical Information Infraestructures Security*, 2009.

[13] A. hamed Mohsenian-rad, V. W. S. Wong, J. Jatskevich, and R. Schober, "1 optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid."

[14] O. Fatemieh, R. Chandra, and C. A. Gunter, "Low cost and secure smart meter communications using the tv white spaces," *ISRCS '10: IEEE International Symposium on Resilient Control Systems*, August. 2010.

[15] M. LeMay, G. Gross, C. A. Gunter, and S. Garg, "Unified architecture for large-scale attested metering," in *Hawaii International Conference on System Sciences*. Big Island, Hawaii: ACM, January 2007.

[16] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "Pretp: Privacy-preserving electronic toll pricing," in *19th Usenix Security Symposium*, August 2010.

[17] W. de Jonge and B. Jacobs, "Privacy-friendly electronic traffic pricing via commits," in *Formal Aspects in Security and Trust*, ser. Lecture Notes in Computer Science, P. Degano, J. Guttman, and F. Martinelli, Eds., vol. 5491. Springer, 2008, pp. 143–161.

[18] C. Troncoso, G. Danezis, E. Kosta, and B. Preneel, "PriPAYD: privacy friendly pay-as-you-drive insurance," in *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007*, P. Ning and T. Yu, Eds. ACM, 2007, pp. 99–107.

[19] R. J. Anderson, "Why information security is hard-an economic perspective," in *ACSAC*. IEEE Computer Society, 2001, pp. 358–365.

[20] D. Bell, "Looking back at the bell-la padula model," dec. 2005, pp. 15 pp. –351.

[21] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, 2001, pp. 136–145.

[22] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.

[23] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *CRYPTO '92*, E. F. Brickell, Ed., vol. 740. Springer-Verlag, 1992, pp. 390–420.

[24] C. Schnorr, "Efficient signature generation for smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 239–252, 1991.

[25] D. Chaum and T. Pedersen, "Wallet databases with observers," in *CRYPTO '92*, ser. LNCS, vol. 740, 1993, pp. 89–105.

[26] T. Okamoto, "An efficient divisible electronic cash scheme," in *CRYPTO*, ser. Lecture Notes in Computer Science, D. Coppersmith, Ed., vol. 963. Springer, 1995, pp. 438–451.

[27] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 1807. Springer, 2000, pp. 431–444.

[28] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 839. Springer, 1994, pp. 174–187.

[29] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, ser. LNCS, A. Odlyzko, Ed., vol. 263. Springer, 1986, pp. 186–194.

[30] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Institute for Theoretical Computer Science, ETH Zürich, Tech. Rep. TR 260, Mar. 1997.

[31] A. Rial, M. Kohlweiss, and B. Preneel, "Universally composable adaptive priced oblivious transfer," in *Pairing*, ser. Lecture Notes in Computer Science, H. Shacham and B. Waters, Eds., vol. 5671. Springer, 2009, pp. 231–247.

[32] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *ACNS*, ser. Lecture Notes in Computer Science, J. Ioannidis, A. Keromytis, and M. Yung, Eds., vol. 3531, 2005, pp. 467–482.

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

[34] M. Dworkin, "Cryptographic protocols of the identity mixer library, v. 2.3.0." IBM Research, IBM Research Report RZ3730, 2010, http://domino.research.ibm.com/library/cyberdig.nsf/index.html.

[35] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM Conference on Computer and Communications Security*, 1993, pp. 62–73.

[36] R. Rivest, "On the notion of software independencein voting systems," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, p. 3759, 2008.

[37] A. Manson, "Smart metering wan communications requirements," Draft Report of the energy retail association, November 2008.

[38] M. O. Rabin and J. O. Shallit, "Randomized algorithms in number theory," *Communications on Pure and Applied Mathematics*, vol. 39, no. S1, pp. 239–256, 1986.

Appendixes are included for the purpose of double blind-review – not for publication.

APPENDIX
IMPLEMENTATION IN DETAIL

We propose an efficient instantiation for the commitment scheme, for the signatures schemes used by M, U and P, and for the non-interactive proofs of knowledge that are used in the construction described in Section VI.

1) Commitment Scheme.: We choose the integer commitment scheme due to Groth [32]. Let l_n be the bit-length of a special RSA modulus n and l_r be the bit-length of the security parameter. Typical values are $l_n = 2048$ and $l_r = 80$.

- $\mathsf{ComSetup}(1^k)$. Given a special RSA modulus, pick a random generator $h \in QR_n$. Pick random $\alpha_1, \ldots, \alpha_k \leftarrow \{0,1\}^{l_n+l_r}$ and, for $i = 1$ to k, compute $g_i = h^{\alpha_i}$. Output commitment parameters $par_c = (g_1, \ldots, g_k, h, n)$ and trapdoor $td = (\alpha_1, \ldots, \alpha_k)$.
- $\mathsf{Commit}(par_c, \langle m_1, \ldots, m_k \rangle)$. On input integers $(m_1, \ldots, m_k)$ of length l_m, choose a random $open \in \{0,1\}^{l_n+l_r}$, and compute $C = g_1^{m_1} \cdot \ldots \cdot g_k^{m_k} h^{open} (\text{mod } n)$. Output the commitment c and the opening $open$.
- $\mathsf{Open}(par_c, c, \langle m'_1, \ldots, m'_k \rangle, open')$. On inputs integers $(m'_1, \ldots, m'_k)$ and $open'$, compute $c' = g_1^{m'_1} \cdot \ldots \cdot g_k^{m'_k} h^{open'} (\text{mod } n)$ and check whether $c = c'$.

2) Signature Schemes.: The signature schemes of M and U can be instantiated with any existentially unforgeable signature scheme. For P's signature scheme, we choose the signature scheme proposed by Camenisch and Lysyanskaya [33].

- $\mathsf{Keygen}(1^k)$. On input 1^k, generate two safe primes p, q of length k such that $p = 2p'+1$ and $q = 2q'+1$, where p' and q' are also primes. The special RSA modulus of length l_n is defined as $n = pq$. Output secret key $sk = (p, q)$. Choose uniformly at random $S \leftarrow QR_n$, and $R_1, \ldots, R_k, Z \leftarrow \langle S \rangle$. Compute a non-interactive zero-knowledge proof $\pi = \mathsf{NIPK}\{(x_Z, x_{R_1}, \ldots, x_{R_l}) : Z = S^{x_Z} \wedge R_1 = S^{x_{R_1}} \wedge \ldots \wedge R_l = S^{x_{R_l}}\}$. Output public key $pk = (n, R_1, \ldots, R_k, S, Z, \pi)$.
- $\mathsf{Sign}(sk, \langle m_1, \ldots, m_k \rangle)$. On input messages $(m_1, \ldots, m_k)$ of length l_m, choose a random prime number e of length $l_e > l_m + 2$, and a random number v of length $l_v = l_n + l_m + l_r$. Compute the value A such that $Z = A^e R_1^{m_1} \cdot \ldots \cdot R_k^{m_k} S^v (\text{mod } n)$. Output the signature $s = (e, A, v)$.
- $\mathsf{Verify}(pk, s, \langle m_1, \ldots, m_k \rangle)$. On inputs messages $(m_1, \ldots, m_k)$ and signature $s = (e, A, v)$, check that $Z \equiv A^e R_1^{m_1} \cdot \ldots \cdot R_k^{m_k} S^v (\text{mod } n)$, that $m_i \in \pm\{0,1\}^{l_m}$, and that $2^{l_e} \leq e \leq 2^{l_e}-1$.

Typical values are $l_n = 2048$, $l_r = 80$, $l_m = 256$, $l_e = 597$, $l_v = 2724$ ([34]).

3) Non-Interactive Zero-Knowledge Proof.: We describe the basic buildings blocks that compose the non-interactive zero-knowledge proofs utilized in our construction in Section VI. Such non-interactive zero-knowledge proofs consist of the conjunction of some of those building blocks. The basic building blocks are a non-interactive zero-knowledge proof of possession of a Camenisch-Lysyanskaya signature, a proof that a committed value is the product of two committed values and a proof that a committed value lies in an interval.

To prove possession of a Camenisch-Lysyanskaya signature, we employ the proof described in [34]. Given a signature $s = (e, A, v)$ on messages $(m_1, \ldots, m_k)$, randomize the signature s by picking random $r \leftarrow \{0,1\}^{l_n+l_\oslash}$ and computing $(e, A' = AS^{-r} (\text{mod } n), v' = v + er)$. Additionally set $e' = e - 2^{l_e-1}$. Send A' to the verifier along with the following non-interactive zero-knowledge proof:

$$\begin{aligned} \mathsf{NIPK}\{ & (e, v, m_1, \ldots, m_k) : \\ & Z \equiv \pm A^e R_1^{m_1} \cdot \ldots \cdot R_k^{m_k} S^v (mod\ n) \wedge \\ & m_i \in \{0,1\}^{l_m+l_H+l_\oslash+2} \wedge \\ & e - 2^{l_e-1} \in \{0,1\}^{l'_e+l_H+l_\oslash+2}\}. \end{aligned}$$

We turn this proof into a non-interactive zero-knowledge argument via the Fiat-Shamir heuristic as follows. (All the proofs in our implementation are computed via the Fiat-Shamir heuristic in a similar way.) Let H be a hash function modeled as a random oracle [35]. The prover picks random values:

$$\begin{array}{lcl} r_e & \leftarrow & \{0,1\}^{l'_e+l_H+l_\oslash} \\ r_{v'} & \leftarrow & \{0,1\}^{l_v+l_H+l_\oslash} \\ \{r_{m_i}\}_{i=1}^k & \leftarrow & \{0,1\}^{l_m+l_H+l_\oslash} \end{array}$$

where l_H is the size of the challenge, $l_\oslash$ controls statistical zero-knowledge and $l'_e < l_e - l_H - l_\oslash - 3$ is the bit-length that determines the interval from which e must be taken in order to succeed in the proof with interval checks $e - 2^{l_e-1} \in \{0,1\}^{l'_e+l_H+l_\oslash+2}$. The prover computes a commitment $t_Z = A'^{r_e} R_1^{r_{m_1}} \cdot \ldots \cdot R_k^{r_{m_k}} S^{r_v}$ and a challenge $ch = H(n||A'||R_1|| \ldots ||R_k||S||Z||t_Z)$. The prover computes responses:

$$\begin{array}{lcl} s_e & = & r_e - ch \cdot e' \\ s_{v'} & = & r_{v'} - ch \cdot v' \\ \{s_{m_i}\}_{i=1}^k & = & r_{m_i} - ch \cdot m_i \end{array}$$

and sends to the verifier $\pi = (A', ch, s_e, s_{v'}, \{s_{m_i}\}_{i=1}^k)$. The verifier computes $t'_Z = (Z/(A')^{2^{l_e-1}})^{ch} A'^{s_e} R_1^{s_{m_1}} \cdot \ldots \cdot R_k^{s_{m_k}} S^{s_{v'}}$, verifies if $ch = H(n||A'||R_1|| \ldots ||R_k||S||Z||t'_Z)$, and runs the interval checks $s_e \in \pm\{0,1\}^{l'_e+l_H+l_\oslash+1}$ and $\{s_{m_i}\}_{i=1}^k \in \pm\{0,1\}^{l_m+l_H+l_\oslash+1}$. Typical values for the parameters are $l_H = 256$, $l_\oslash = 80$ and $l'_e = 120$.

To prove that a message m_3 committed to in $c_{m_3} = g_1^{m_3} h^{open_{m_3}}$ is the product of two messages m_1 and m_2 committed to in $c_{m_1} = g_1^{m_1} h^{open_{m_1}}$ and $c_{m_2} = g_1^{m_2} h^{open_{m_2}}$

respectively, the following proof can be used:

$$\begin{aligned}\mathsf{NIPK}\{\ &(m_1, open_{m_1}, m_2, open_{m_2}, m_3, open_{m_3},\\ &m_2 \cdot open_{m_1}) : c_{m_1} = g_1^{m_1} h^{open_{m_1}} \wedge\\ &c_{m_2} = g_1^{m_2} h^{open_{m_2}} \wedge c_{m_3} = g_1^{m_3} h^{open_{m_3}} \wedge\\ &1 = c_{m_1}^{m_2} (1/g_1)^{m_3} (1/h)^{m_2 \cdot open_{m_1}}\}.\end{aligned}$$

To prove that a committed value x lies in an interval $[a, b]$, it is necessary to prove that $x - a \geq 0$ and $b - x \geq 0$. We employ the non-interactive zero-knowledge proof due to Groth [32] 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, d) can be computed via the Rabin-Shallit algorithm [38]. The proof is:

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

APPENDIX
SECURITY ANALYSIS

Theorem 1: This PSM scheme securely realizes $\mathcal{F}_{\mathrm{PSM}}$.

In order to prove this theorem, we need to build a simulator $\mathcal{E}$ that invokes a copy of adversary $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathrm{PSM}}$ and environment $\mathcal{Z}$ in such a way that ensembles $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ and $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. We analyze formally the security of our scheme when only the provider P is corrupted, when only the user U is corrupted, and when only the meter M is corrupted.

A. Security Analysis When Provider Is Corrupted

Claim 1: When P is corrupted, the distribution ensembles $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ and $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable under the unforgeability of the signature schemes $(\mathrm{Mkeygen}, \mathrm{Msign}, \mathrm{Mverify})$ and $(\mathrm{Ukeygen}, \mathrm{Usign}, \mathrm{Uverify})$, under the hiding property of the commitment scheme and the extractability and witness-indistinguishability of proofs of knowledge.

Proof: We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ 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 honest M and U. Thus Pr [**Game** 0] $= 0$.
- **Game** 1: This game proceeds as **Game** 0, except that the public keys pk_{M} and pk_{U} are replaced by other public keys $pk_{\mathrm{M}}{}'$ and $pk_{\mathrm{U}}{}'$ that are obtained by running Mkeygen and Ukeygen respectively. Since these public keys have the same distribution as pk_{M} and pk_{U}, then $|\Pr\ [\textbf{Game}\ 1] - \Pr\ [\textbf{Game}\ 0]| = 0$.
- **Game** 2: This game proceeds as **Game** 1, except that **Game** 2 aborts if $\mathcal{A}$ sends a message-signature pair (m, s) verifiable with public key pk_{M} and $\mathcal{A}$ did not receive a signature on m verifiable with pk_{M}. The probability that **Game** 2 aborts is bounded by the following lemma:
  Lemma 1: Under the unforgeability of the signature scheme defined by algortithms $(\mathrm{Mkeygen}, \mathrm{Msign}, \mathrm{Mverify})$, $|\Pr\ [\textbf{Game}\ 2] - \Pr\ [\textbf{Game}\ 1]| = \nu_1(\kappa)$.
- **Game** 3: This game proceeds as **Game** 2, except that **Game** 3 aborts if $\mathcal{A}$ sends a message-signature pair (m, s) verifiable with public key pk_{U} and $\mathcal{A}$ did not receive a signature on m verifiable with pk_{U}. The probability that **Game** 3 aborts is bounded by the following lemma:
  Lemma 2: Under the unforgeability of the signature scheme defined by algorithms $(\mathrm{Ukeygen}, \mathrm{Usign}, \mathrm{Uverify})$, $|\Pr\ [\textbf{Game}\ 3] - \Pr\ [\textbf{Game}\ 2]| = \nu_2(\kappa)$.
- **Game** 4: This game proceeds as **Game** 3, except that **Game** 4 extracts the witness td from the proof π. Since extraction fails with negligible probability, $|\Pr\ [\textbf{Game}\ 4] - \Pr\ [\textbf{Game}\ 3]| = \nu_3(\kappa)$.
- **Game** 5: This game proceeds as **Game** 4, except that the commitments c_{price_i} that are sent to $\mathcal{A}$ in the payment message Q are replaced by commitments to prices $price_i'$ that add to the fee fee, and commitments $(c_{cons_i}, c_{other_i})$ are replaced by commitments to tuples $(cons_i', other_i')$ that map to $price_i'$ following Υ. The proofs π_i are replaced by proofs that prove knowledge of the opening of those commitments and of a signature $\mathrm{sp} \in \Upsilon_s$ that signs $(cons_i', other_i', price_i')$.
  Lemma 3: Under the assumption that the commitment scheme is hiding and that the non-interactive proofs of knowledge are witness indistinguishable, $|\Pr\ [\textbf{Game}\ 5] - \Pr\ [\textbf{Game}\ 4]| = \nu_4(\kappa)$.

$\mathcal{E}$ performs all the changes described in **Game** 5, and forwards and receives messages from $\mathcal{F}_{\mathrm{PSM}}$ as described in our simulation below.

- **Setup.** When $\mathcal{A}$ sends a request $(\mathsf{register}, pk_{\mathrm{P}})$ to register the public key pk_{P}, $\mathcal{E}$ stores pk_{P}. When $\mathcal{A}$ sends a request $(\mathsf{retrieve}, \mathrm{M})$, $\mathcal{E}$ runs Mkeygen in order to generate a key pair $(pk_{\mathrm{M}}, sk_{\mathrm{M}})$ and sends $(\mathsf{retrieve}, \mathrm{M}, pk_{\mathrm{M}})$ to $\mathcal{A}$. When $\mathcal{A}$ sends a request $(\mathsf{retrieve}, \mathrm{U})$, $\mathcal{E}$ runs Ukeygen in order to generate a key pair $(pk_{\mathrm{U}}, sk_{\mathrm{U}})$ and sends $(\mathsf{retrieve}, \mathrm{U}, pk_{\mathrm{U}})$ to $\mathcal{A}$. When $\mathcal{A}$ sends (par_c, π), $\mathcal{E}$ verifies π, extracts td and stores par_c.
- **Initialization.** When $\mathcal{A}$ sends Υ_s, $\mathcal{E}$ gets Υ from Υ_s and sends $(\mathsf{policy}, \Upsilon)$ to $\mathcal{F}_{\mathrm{PSM}}$.
- **Payment.** When $\mathcal{A}$ sends $(\mathsf{payment})$, $\mathcal{E}$ sends $(\mathsf{payment})$ to $\mathcal{F}_{\mathrm{PSM}}$. Upon receiving (pay, fee, N, b)

from $\mathcal{F}_{\text{PSM}}$, $\mathcal{E}$ picks a set of N prices $price_i'$ that add to fee and N tuples $(cons_i', other_i')$ that map to $price_i'$ following Υ. (We note that such values always exist.) Let d be a counter initialized at 0. For $i = d$ to $d+N-1$, $\mathcal{E}$ runs $\mathsf{SignConsumption}(sk_{\text{M}}, par_c, cons_i', other_i', i)$ to obtain SC. $\mathcal{E}$ creates a table T with the signed consumptions SC and runs $\mathsf{Pay}(sk_{\text{U}}, par_c, \Upsilon_s, T)$ to obtain a payment message Q. $\mathcal{E}$ sends (Q) to $\mathcal{A}$ and updates $d = d + N$.

- **Reveal.** When $\mathcal{A}$ sends (i), $\mathcal{E}$ sends (reveal, i) to $\mathcal{F}_{\text{PSM}}$. Upon receiving $(\mathsf{revresp}, cons, other, b)$ from $\mathcal{F}_{\text{PSM}}$, $\mathcal{E}$ picks the tuple $(c_{cons'}, c_{other'})$ in the message SC $= (d, \ldots)$ such that $d = i$, and, by means of trapdoor td, computes $open_{cons}$ and $open_{other}$ such that $\mathsf{Open}(par_c, c_{cons'}, cons, open_{cons})$ and $\mathsf{Open}(par_c, c_{other'}, other, open_{other})$ output accept. $\mathcal{E}$ sets $r = (i, cons, open_{cons}, other, open_{other})$, signs $s_r = \mathsf{Usign}(sk_{\text{U}}, r)$ and sends $R = (r, s_r)$ to $\mathcal{A}$. $\mathcal{E}$ aborts if $\mathcal{A}$ returns $(\mathsf{reject}, \text{Q}, R)$ such that Q or R contain a message-signature pair that was not sent to $\mathcal{A}$.

The distribution produced in **Game** 5 is identical to that of our simulation. By summation we have that $|\Pr[\textbf{Game } 5] \leq \nu_5$. ■

B. Security Analysis When User Is Corrupted

Claim 2: When U is corrupted, the distribution ensembles $\text{IDEAL}_{\mathcal{F}_{\text{PSM}},\mathcal{E},\mathcal{Z}}$ and $\text{REAL}_{\text{PSM},\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable under the unforgeability of the signature schemes $(\text{Mkeygen}, \text{Msign}, \text{Mverify})$ and $(\text{Pkeygen}, \text{Psign}, \text{Pverify})$, under the binding property of the commitment scheme and under the extractability and zero-knowledge property of proofs of knowledge.

Proof: We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\text{REAL}_{\text{PSM},\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $\text{IDEAL}_{\mathcal{F}_{\text{PSM}},\mathcal{E},\mathcal{Z}}$.

- **Game** 0: This game corresponds to the execution of the real-world protocol with honest M and P. Thus $\Pr[\textbf{Game } 0] = 0$.
- **Game** 1: This game proceeds as **Game** 0, except that the public keys pk_{M} and pk_{P} and the commitment parameters par_c are replaced by other values $pk_{\text{M}}{}'$, $pk_{\text{P}}{}'$ and $par_c{}'$ that are obtained by running Mkeygen, Pkeygen and ComSetup respectively. Since these values have the same distribution as pk_{M}, pk_{P} and par_c, then $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| = 0$.
- **Game** 2: This game proceeds as **Game** 1, except that **Game** 2 extracts the witness of the proofs of knowledge π included in the payment messages Q. Since extraction fails with negligible probability, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| = \nu_1(\kappa)$.
- **Game** 3: This game proceeds as **Game** 2, except that **Game** 3 aborts if the witness $(price, open_{price}, cons, open_{cons}, other, open_{other}, \text{sp})$ includes a message-signature pair $(\langle cons, other, price\rangle, \text{sp})$ that was not sent to $\mathcal{A}$. The probability that **Game** 3 aborts is bounded by the following lemma:
  Lemma 4: Under the unforgeability of the signature scheme defined by algorithms $(\text{Pkeygen}, \text{Psign}, \text{Pverify})$, $|\Pr[\textbf{Game } 3] - \Pr[\textbf{Game } 2]| = \nu_2(\kappa)$.
- **Game** 4: This game proceeds as **Game** 3, except that **Game** 4 aborts if any of the message-signature pairs $(\langle d, c_{cons}, c_{other}\rangle, \text{sc})$ in the payment messages Q was not sent to $\mathcal{A}$. The probability that **Game** 4 aborts is bounded by the following lemma:
  Lemma 5: Under the unforgeability of the signature scheme defined by algorithms $(\text{Mkeygen}, \text{Msign}, \text{Mverify})$, $|\Pr[\textbf{Game } 4] - \Pr[\textbf{Game } 3]| = \nu_3(\kappa)$.
- **Game** 5: This game proceeds as **Game** 4, except that the proof of knowledge π in the messages (par_c, π) is replaced by a simulated proof. Under the zero-knowledge property of proofs of knowledge, $|\Pr[\textbf{Game } 5] - \Pr[\textbf{Game } 4]| = \nu_4(\kappa)$.
- **Game** 6: This game proceeds as **Game** 5, except that **Game** 6 aborts if $\mathcal{A}$ sends a payment message Q in which $(fee, open_{fee})$ is a correct opening of the commitment $\otimes_{i=1}^{N} c_{price_i}$, but $fee \neq \sum_{i=1}^{N} price_i$, where $price_i$ is in the witness of proofs $\pi \in$ Q. The probability that **Game** 6 aborts is bounded by the following lemma:
  Lemma 6: Under the binding property of the commitment scheme, we have that $|\Pr[\textbf{Game } 6] - \Pr[\textbf{Game } 5]| = \nu_5(\kappa)$.
- **Game** 7: This game proceeds as **Game** 6, except that **Game** 7 aborts if $\mathcal{A}$ sends an opening message $R = (i, cons, open_{cons}, other, open_{other}, s_r)$ such that the tuple $(\text{sc}_i, i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i)$ of the payment message Q contains a commitment c_{cons_i} that is opened correctly $(cons, open_{cons})$ or a commitment c_{other_i} that is opened correctly by $(other, open_{other})$, but where $cons$ or $other$ do not equal those in message SC $= (i, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \text{sc})$ that was previously sent to $\mathcal{A}$. The probability that **Game** 7 aborts is bounded by the following lemma:
  Lemma 7: Under the binding property of the commitment scheme, we have that $|\Pr[\textbf{Game } 7] - \Pr[\textbf{Game } 6]| = \nu_6(\kappa)$.

$\mathcal{E}$ performs all the changes described in **Game** 7, and forwards and receives messages from $\mathcal{F}_{\text{PSM}}$ as described in our simulation below.

- **Setup.** When $\mathcal{A}$ sends a request $(\mathsf{register}, pk_{\text{U}})$ to register the public key pk_{U}, $\mathcal{E}$ stores pk_{U}. When $\mathcal{A}$ sends a request $(\mathsf{retrieve}, \text{M})$, $\mathcal{E}$ runs Mkeygen in order to generate a key pair $(pk_{\text{M}}, sk_{\text{M}})$ and sends $(\mathsf{retrieve}, \text{M}, pk_{\text{M}})$ to $\mathcal{A}$. When $\mathcal{A}$ sends a request

(retrieve, P), $\mathcal{E}$ runs Pkeygen in order to generate a key pair $(pk_{\mathrm{P}}, sk_{\mathrm{P}})$ and sends (retrieve, P, pk_{P}) to $\mathcal{A}$. $\mathcal{E}$ runs $\mathsf{ComSetup}(1^k)$ to get par_c and a trapdoor td and sends par_c and a simulated proof π to $\mathcal{A}$.

- **Initialization.** When $\mathcal{F}_{\mathrm{PSM}}$ sends (policy, Υ), $\mathcal{E}$ runs $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$ to get Υ_s and sends Υ_s to $\mathcal{A}$.
- **Consumption.** When $\mathcal{F}_{\mathrm{PSM}}$ sends (consume, $cons$, $other$), $\mathcal{E}$ increments a counter d, runs $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, cons, other, d)$ to get SC and sends SC to $\mathcal{A}$.
- **Payment.** When $\mathcal{F}_{\mathrm{PSM}}$ sends (payment, fee, N), $\mathcal{E}$ sends (payment) to $\mathcal{A}$. Upon receiving $\mathrm{Q} = (fee', open_{fee'}, \{\mathrm{sc}_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^{N'})$ from $\mathcal{A}$, $\mathcal{E}$ extracts the witness $(price, open_{price}, cons, open_{cons}, other, open_{other}, \mathrm{sp})$ of proofs π and aborts if any of the conditions described in **Game** 3, **Game** 4 or **Game** 6 are fulfilled. $\mathcal{E}$ sends (pay, fee', N') to $\mathcal{F}_{\mathrm{PSM}}$.
- **Reveal.** When $\mathcal{F}_{\mathrm{PSM}}$ sends (reveal, i), $\mathcal{E}$ sends (i) to $\mathcal{A}$. Upon receiving R from $\mathcal{A}$, $\mathcal{E}$ parses R as $(i, cons, open_{cons}, other, open_{other}, s_r)$ and aborts if the condition described in **Game** 7 is fulfilled. Otherwise $\mathcal{E}$ sends (revresp, $cons$, $other$) to $\mathcal{F}_{\mathrm{PSM}}$.

The distribution produced in **Game** 7 is identical to that of our simulation. By summation we have that $|\Pr[\textbf{Game } 7] \leq \nu_7$. ■

C. Security Analysis When Meter Is Corrupted

Claim 3: When the meter is corrupted, the distribution ensembles $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ and $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ are unconditionally indistinguishable.

Proof: We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$.

- **Game** 0: This game corresponds to the execution of the real-world protocol with honest U and P. Thus $\Pr[\textbf{Game } 0] = 0$.
- **Game** 1: This game proceeds as **Game** 0, except that the the commitment parameters par_c are replaced by other parameters $par_c{}'$ that are obtained by running ComSetup respectively. Since these values have the same distribution as par_c, then $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| = 0$.

$\mathcal{E}$ performs all the changes described in **Game** 1, and forwards and receives messages from $\mathcal{F}_{\mathrm{PSM}}$ as described in our simulation below.

- **Setup.** When $\mathcal{A}$ sends a request (register, pk_{M}) to register the public key pk_{M}, $\mathcal{E}$ stores pk_{M}. $\mathcal{E}$ runs $\mathsf{ComSetup}(1^k)$ to get par_c and sends par_c to $\mathcal{A}$.
- **Consumption.** When $\mathcal{A}$ sends SC, $\mathcal{E}$ increments a counter d and runs $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, \mathrm{SC}, d)$ to get a bit b. If $b = 1$, $\mathcal{E}$ parses SC as $(d_{\mathrm{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, \mathrm{sc})$ and sends (consume, $cons$, $other$) to $\mathcal{F}_{\mathrm{PSM}}$.

The distribution produced in **Game** 1 is identical to that of our simulation. We have that $|\Pr[\textbf{Game } 1] = 0$. ■

APPENDIX
OTHER SPECIFIC POLICIES

1) Discrete Policy.: The simplest policy to be expressed is a policy that considers a discrete domain described by n tuples $(cons, other)$. Each tuple is mapped to a price $price$. To sign the policy, for $i = 1$ to n, P runs $\mathrm{sp}_i = \mathsf{Psign}(sk_{\mathrm{P}}, \langle cons_i, other_i, price_i \rangle)$, and sets $\Upsilon_s = (cons_i, other_i, price_i, \mathrm{sp}_i)_{i=1}^n$. To compute the proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the price $(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price)$ specified in the policy for $(cons, other)$. Then U proves possession of a signature $\mathrm{sp} \in \Upsilon_s$ on $(cons, other, price)$ and equality between the signed values and the values committed to in $(c_{cons}, c_{other}, c_{price})$:

$$\begin{aligned}\mathsf{NIPK}\{\ & (price, open_{price}, cons, open_{cons}, other, open_{other}, \mathrm{sp}) : \\ & (c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge \\ & (c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge \\ & (c_{price}, open_{price}) = \mathsf{Commit}(par_c, price) \wedge \\ & \mathrm{Pverify}(pk_{\mathrm{P}}, \mathrm{sp}, \langle cons, other, price \rangle) = \mathsf{accept}\}.\end{aligned}$$

2) Interval Policy.: In an interval policy, the consumption values domain is divided into intervals and each interval is mapped to a price. For instance, if the policy says that all the consumptions between 4 and 7 must pay price 3 and your consumption is 5, the price due is 3. Therefore, an interval policy is given by $\Upsilon : (cons_{min}, cons_{max}, other) \rightarrow price$, where it is required that intervals defined by $[cons_{min}, cons_{max}]$ be disjoint.

To sign this policy, for $i = 1$ to n, P runs $\mathrm{sp}_i = \mathsf{Psign}(sk_{\mathrm{P}}, \langle cons_{min_i}, cons_{max_i}, other_i, price_i \rangle)$, and sets $\Upsilon_s = (cons_{min_i}, cons_{max_i}, other_i, price_i, \mathrm{sp}_i)_{i=1}^n$.[10] To compute a proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the price $(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price)$ specified in the policy for $(cons_{min}, cons_{max}, other)$ such that $cons \in [cons_{min}, cons_{max}]$. Then U computes a proof of possession of a signature $\mathrm{sp} \in \Upsilon_s$ on $(cons_{min}, cons_{max}, other, price)$, a proof of equality between $(other, price)$ and the values committed to in (c_{other}, c_{price}), and a proof that[11] $cons \in$

[10] We note that if Υ is a monotonic function, then it is enough to sign $cons_{max}$ (when the function is increasing) or $cons_{min}$ (when the function is decreasing).

[11] If the policy is monotonically increasing, it suffices to prove that $cons \in [0, cons_{max}]$, while if it is monotonically decreasing, it suffices to prove that $cons \in [cons_{min}, \infty)$.

$[cons_{min}, cons_{max}]$:

$$
\begin{aligned}
\mathsf{NIPK}\{\ & (price, open_{price}, cons, open_{cons}, other, open_{other}, cons_{min}, cons_{max}, \mathrm{sp}) : \\
& (c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons) \wedge \\
& (c_{other}, open_{other}) = \mathsf{Commit}(par_c, other) \wedge \\
& (c_{price}, open_{price}) = \mathsf{Commit}(par_c, price) \wedge \\
& \mathrm{Pverify}(pk_{\mathrm{P}}, \mathrm{sp}, \langle cons_{min}, cons_{max}, other, price \rangle) = \mathsf{accept} \wedge \\
& cons \in [cons_{min}, cons_{max}]\}.
\end{aligned}
$$