# 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

Smart grid proposals threaten user privacy by potentially disclosing fine-grained consumption data to utility providers, primarily for time-of-use billing, but also for profiling, settlement, forecasting, tariff and energy efficiency advice. We propose a privacy-preserving protocol for general calculations on fine-grained meter readings, while keeping the use of tamper evident meters to a strict minimum. We allow users to perform and prove the correctness of computations based on readings on their own devices, without disclosing any fine grained consumption. Applying the protocols to time-of-use billing is particularly simple and efficient, but we also support a wider variety of tariff policies. Cryptographic proofs and multiple implementations are used to show the proposed protocols are secure and efficient.

## 1 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 [12]. Expected electricity savings depend on matching generation and demand, achieved partly through dynamic tariffs with higher rates during peak consumption periods. Further savings are expected through the use of smart meter data for more accurate forecasting, more accurate settlement of costs between suppliers and producers (in the UK energy market) as well as customised energy efficiency advice. 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 [1]: meters send all fine-grained measurements to the utilities or a centralised database. Yet, it is recognised that meter readings leak personal information. For example, load monitoring [25, 26] allows the identification of specific electrical appliances. As a result, detailed consumption data would facilitate the creation of user lifestyle profiles, including but not limited to house occupancy, meal times, working hours, or prayer or fasting patterns.

To alleviate such concerns, privacy impact assessments (PIA) are included in ongoing standardization processes. The National Institute of Standards and Technology (NIST) [38] lists fine-grained readings as being used for load monitoring, forecasting, demand-response, efficiency analysis and billing. Time-of-use billing is a major reason for collecting and storing all fine-grained readings, and thus we use it to illustrate our techniques. Other computations on readings are also supported.

Consumer privacy concerns have already jeopardised the mandatory deployment of smart meters in the Netherlands [15], leading to a deployment deadlock. This deadlock stems from the assumption that smart metering is necessarily privacy invasive and that a balance needs to be struck between privacy and the social utility of fine-grained billing. Our work refutes this assumption: we demonstrate an architecture that guarantees privacy and high integrity for a very broad set of smart-metering and billing applications.

**Our Contribution.** We propose a set of protocols amongst a provider, a user agent and a simple tamper-evident meter. The meter outputs certified readings of measurements and gives them to the user, either directly or through a wide area network secure channel. For billing, 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 ensures the calculation to be correct and leaks no 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).

Other calculations can also be performed and certified

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

to support forecasting, profiling, settlement, or fraud detection. Complex calculations are enabled by our scheme for applying non-linear functions as well as look-ups to certified readings, with efficient zero-knowledge proofs based on re-randomizable signatures.

The need for certifying meter readings is the only modification necessary to the meters. Users can delegate the calculation of their bill or other computations to any device or service they trust without compromising the integrity of the scheme. Our key aim is for users to be able to perform all privacy friendly operations within a web-browser, keeping their experience of interacting on-line with their provider unchanged.

Variants of the scheme eliminate covert channels and ensure privacy even if the meter actively attempts to leak information to the provider. On the other hand, when a simple tariff policy is applied, we can construct a very efficient protocol for billing that requires no zero-knowledge proofs and is particularly well suited for time-of-use billing.

Although no other information needs to be leaked for billing purposes, we allow individual or aggregate meter readings to be disclosed according to a privacy policy, in order to facilitate further smart-grid functions such as abuse prevention or load prediction.

## 2 Related Work

The damage that smart meters can cause to users' privacy has previously been studied both from a technical [28, 30] and a legal perspective [12, 34]. These works propose enforcement of privacy properties based on organizational means, codes of conduct and regulations, subject to current legislation. These works assume that billing inevitably requires the sharing of detailed meter readings.

Little work exists on the design of technical solutions to protect privacy in the smart grid. Wagner et al. [40] propose a privacy-aware framework for the smart grid based on semantic web technologies. Garcia and Jacobs [21] 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 NIST privacy subgroup [38] suggests anonymizing traces of readings, as proposed by Efthymiou et al. [18], but also warns of the ease of re-identification. Molina et al. [32] highlight the private information that current meters leak, and sketch a protocol that could use zero-knowledge proofs to achieve privacy in metering.

Some work focuses on more general aspects of smart grid security. Anderson and Fuloria [1] analyze the security economics of electricity metering. In addition to privacy issues, they discuss pricing policies, the behavioural economics needed to understand how smart meters reduce electricity usage, and the conflict of interests among the different entities. McLaughlin et al. [31] 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 [24]. Proposals to enhance the security of the smart grid infrastructure include Fatemieh et al. [19]. No complete and thorough solution exists for computing privately individual bills when complex time-of-use tariffs are applied, or perform general private computations needed to run a modern grid.

Smart-metering is a special case of metering. LeMay et al. propose an architecture for attested metering [27] based on calculations performed on trusted hardware. Our protocol follows an approach similar to the one described in [3, 16] for the design of a privacy-friendly electronic toll pricing system. Whereas Balasch et al. use 'spot checks' to ensure the correctness of the calculation, we use simple tamper evident meters. This introduces no additional trust assumptions as current smart-grid proposals already rely on tamper-resistant meters. We extend their paradigm of proving some aspects of a metering system using cryptography by providing full end-to-end verifiability for computations.

Troncoso et al. [39] propose an architecture in which secure meters are used to calculate final bills for pay-as-you-drive insurance. Although this architecture could be used in our secure metering setting, it has drawbacks. Meters are larger and more complex than in our scheme, making them more expensive and 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. Most importantly, the black box model might misalign incentives [2]: 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].

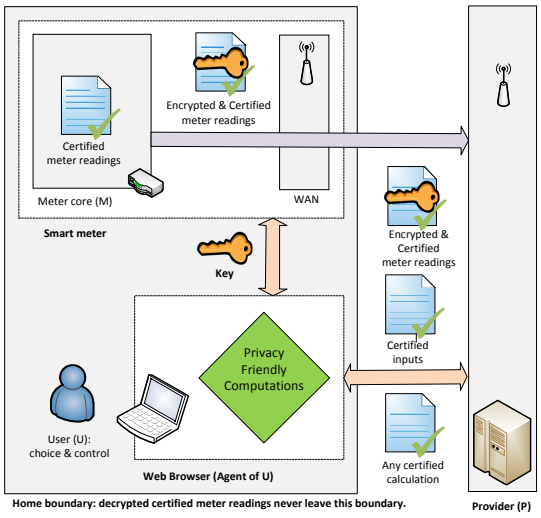


Figure 1: Interactions between parties.

## 3 Design Goals & Rationale

We propose a protocol to preserve user privacy in smart metering applications that is flexible enough to be applied in a number of settings including electricity, water and gas metering. Our protocol guarantees the following security properties. First *integrity:* the utility provider is assured that the user reports the correct results of calculations. Second *privacy*: the provider does not learn any information but the result of computations. For the case of billing, the provider is ensured the correct fee is calculated based on the actual readings and time-of-use tariffs, without learning any fine grained readings. Finally, the provider cannot claim that a user must pay an incorrect fee.

The aim of our protocols is to keep meters extremely simple and to rely on cryptographic calculations outside the tamper-evident part of the meter for the integrity or specific calculations like billing. Meters need to be cheap and as a result have limited connectivity and bandwidth. They offer only a very limited user-interface that cannot deliver information about energy usage, efficiency advice, or detailed billing. Our protocols have been designed to impose a small computational overhead on meters, and a negligible communication overhead – both of which should be achievable without any additional hardware.

Once meter readings are certified and output from the meter, our protocols provide flexibility about where calculations are performed without compromising integrity. This flexibility means that devices and software performing the actual billing can evolve over time while the meters remain the same. In particular additional unforeseen computations on certified readings can be performed without changing and re-certifying meters.

Figure 1 illustrates a key use case we would like to support: meter readings are certified by the meter, encrypted using a local symmetric key and uploaded to remote servers using a wide-area network. A customer simply uses a web-browser to connect to their supplier's site, at which point the meter readings are downloaded and decrypted with the key. Privacy-friendly calculations are performed in the browser, along with the proofs they are correct, for billing, settlement, fraud detection and profiling. The results and proofs are then relayed back to the provider for verification and further processing. The client side web-application can make further use of the meter readings to generate efficiency reports and a rich user interface based on the actual energy consumption of the user. The provider never learns the detailed readings, yet is able to provide a rich user experience, as well as compute highly reliable results on readings to support billing and other processes.

Alternative user agents for computations could include smart phones, standalone software clients, as well as third party service providers trusted by the users. In all those cases, as above, certified bills or other computations can be proved correct, ensuring high integrity.

**System Model.** Without loss of generality we will describe our protocols in terms of billings, where certified fine-grained readings and certified time-of-use tariffs are used to calculate how much money a customer owes an electricity supplier for some period of consumption.

We describe our protocol in an abstract setting that comprises three parties, as illustrated in Figure 1: a tamper-resistant meter $\mathrm{M}$ that outputs consumption data $cons$ and related information $other$; a service provider $\mathrm{P}$ that establishes a pricing policy $\Upsilon$ and that, at each billing period, requests the user to pay the fee $fee$ corresponding to her total consumption; finally a user $\mathrm{U}$ that receives consumption readings from meter $\mathrm{M}$ and pays a fee to provider $\mathrm{P}$. 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, to charge a tariff as a function of consumption per day or per week.

The basic operation of the system is as follows: the provider $\mathrm{P}$ sends the user $\mathrm{U}$ a pricing policy $\Upsilon$. During

[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.

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, on a device or service they trust, the total fee and sends it to the provider P. The user U also produces and sends a proof that the fee has been correctly computed using the pricing policy $\Upsilon$ and all the consumption measurements output by the meter M.

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 composed readily without leaking additional information.

## 4 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 probabilistic polynomial time (p.p.t.) adversary should be able to output a message-signature pair $(s, m)$ unless he has previously obtained a signature on $m$.

**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. 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 use homomorphic commitment schemes extensively. 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 [5] is a two-party protocol between a prover and a verifier. The prover demonstrates to the verifier her 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. Second, it should be zero-knowledge, i.e., the verifier learns nothing but the truth of the statement. 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 [37]; proof of knowledge of the equality of some element in different representations [13]; proof with interval checks [33], range proof [8] and proof of the disjunction or conjunction of any two of the previous [14]. These results are often given in the form of $\Sigma$-protocols but they can be turned into non-interactive zero-knowledge arguments in the random oracle model via the Fiat-Shamir heuristic [20].

When referring to the proofs above, we follow the notation introduced by Camenisch and Stadler [10] 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 $\alpha$, $\beta$, and $\delta$ 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. The convention is that letters in the parenthesis, in this example $\alpha$, $\beta$, and $\delta$, 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 $\mathsf{NIPK}\{(x, s_x) : \mathsf{Verify}(pk, x, s_x) = \mathrm{accept}\}$.

## 5 Construction

**Intuition Behind Our Construction.** We consider a setting with the entities presented in Section 3, 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 $\Upsilon$ 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. Using 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 $(cons, other)$. U only 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 $\Upsilon$;

2. a non-interactive zero-knowledge proof $\pi$ that she (a) knows the openings of the signed commitments, (b) knows the opening of the commitment to the price, and (c) possesses a signature computed by P on $(cons, other, price)$ that states that $price$ is the price to be paid for $(cons, other)$.

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

[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.

Upon receiving the payment message, P verifies the signature by U, and the signatures by M on the commitments to $(cons, other)$ and on $d$. P also verifies the proofs $\pi$. P then uses the homomorphic property of the commitment scheme to aggregate all the commitments to the prices and get a commitment to $fee$. Finally P checks whether $(fee, open_{fee})$ is a valid opening for it before accepting the reported bill. 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 agrees to P learning this information, U reveals to P the openings of some commitments to $(cons, other)$.

**Security.** In Appendix 11 we describe the security model we employ, the ideal-world/real-world paradigm [11], and we propose an ideal functionality $\mathcal{F}_{\text{PSM}}$, which defines the security properties for privacy-preserving smart metering. Any construction that realizes $\mathcal{F}_{\text{PSM}}$ ensures that U pays the right fee for the consumption data output by M, i.e., that the fee is computed following $\Upsilon$. It also ensures that, if M and P do not collude, P only learns the fee paid, not the consumption data $cons$ nor the other information $other$ used to compute $fee$. Additionally, it ensures that a malicious provider cannot claim that the fee that U must pay is different from the one computed following $\Upsilon$.

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 $\Upsilon$ 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. P's security rests 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, owing 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.

### 5.1 Construction

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

In the setup phase, M runs $\mathsf{Mkeygen}(1^k)$ to obtain a key pair $(sk_{\text{M}}, pk_{\text{M}})$, U runs $\mathsf{Ukeygen}(1^k)$ to get a key pair $(sk_{\text{U}}, pk_{\text{U}})$ and P runs $\mathsf{Pkeygen}(1^k)$ 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

**Protocol** PSM

- **Initialization.** When P is activated with $(\mathsf{policy}, \Upsilon)$, P runs $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$ to get a signed policy $\Upsilon_s$. P sends $\Upsilon_s$ 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 $\Upsilon_s$.
- **Consumption.** When M is activated with $(\mathsf{consume}, cons, other)$, M increments a counter $d_{\mathrm{M}}$ (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 $d_{\mathrm{U}}$ and runs $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, SC, d_{\mathrm{U}})$ 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 $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, Q, d_{\mathrm{P}})$ to obtain $(b, d'_{\mathrm{P}})$. If $b = 0$, P rejects the bill, otherwise the bill is accepted $d_{\mathrm{P}} = d'_{\mathrm{P}}$ is set.
- **Reveal.** When P is activated with $(\mathsf{reveal}, i)$, 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 $\mathsf{VerifyReveal}(pk_{\mathrm{U}}, par_c, Q, R, i)$ to get a bit $b$. If $b = 0$, P sends $(\mathsf{reject}, Q, R)$ to U, otherwise it sends $(\mathsf{accept})$ to U.

Figure 2: The concrete implementation of privacy-friendly metering.

$td$, computes a proof $\pi = \mathsf{NIPK}\{(td) : (par_c, td) \leftarrow \mathsf{ComSetup}(1^k)\}$ and sends $(par_c, \pi)$ to U and $(par_c)$ to M. U verifies $\pi$.

The outline of our concrete construction is presented in Figure 2, and the functions called are detailed in Figure 3.

## 5.2 Policies

We detail here the computation of the signed policy $\Upsilon_s$, and how to prove that the price for a tuple $(cons, other)$ is computed in accordance with the right entry specified in the policy $\Upsilon$. They differ depending on different types of policies $\Upsilon$. 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 in order to calculate their contribution to the bill (see Figure 4). 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.

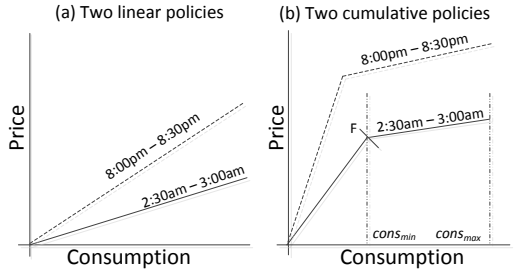


Figure 4: (a) A linear policy specifies the rate per unit consumption that is applied to determine the price to be paid for each measurement. The rate can be selected through information associated with the reading, like the time of the day or the location of a vehicle (without revealing this information). (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.

Three other policies considered are the discrete policy, which looks up a tariff in a table, and the interval policy which charges a fixed premium per different ranges of consumption (see Appendix 14). These are special cases of the cumulative policy or linear policy, with some efficiently improvements, and are not discussed 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 pricing policies can be composed to express complex composite policies, e.g. to apply a different non-linear policy to the total consumption of each day, and subtracting from the final bill a rebate of 10% if the total units of consumption exceeds a threshold.

**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 units, 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.

To sign this policy, for $i = 1$ to $n$, P runs

**Cryptographic functions used by Protocol** PSM

- $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$. For each tuple $(cons, other, price) \in \Upsilon$, compute $sp = \mathrm{Psign}(sk_{\mathrm{P}}, \langle cons, other, price\rangle)$.[4] Let $\Upsilon_s = (cons_i, other_i, price_i, sp_i)_{i=1}^n$ be the set of message-signature tuples. Output $\Upsilon_s$.
- $\mathsf{VerifyPolicy}(pk_{\mathrm{P}}, \Upsilon_s)$. For $i = 1$ to $n$, parse $\Upsilon_s$ as $(cons_i, other_i, price_i, sp_i)_{i=1}^n$, and, for $i = 1$ to $n$, run $\mathrm{Pverify}(pk_{\mathrm{P}}, sp_i, \langle cons_i, other_i, price_i\rangle)$. If any of the outputs is reject, output $b = 0$, else output $b = 1$.
- $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, cons, other, d_{\mathrm{M}})$. Execute both $(c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons)$ and $(c_{other}, open_{other}) = \mathsf{Commit}(par_c, other)$. Run $sc = \mathrm{Msign}(sk_{\mathrm{M}}, \langle d_{\mathrm{M}}, c_{cons}, c_{other}\rangle)$ and output $SC = (d_{\mathrm{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$.
- $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, SC, d_{\mathrm{U}})$. Parse message $SC$ as $(d_{\mathrm{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$. 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 $\mathrm{Mverify}(pk_{\mathrm{M}}, sc, \langle d_{\mathrm{U}}, c_{cons}, c_{other}\rangle)$ and output $b = 0$ if the output is reject. Otherwise output $b = 1$.
- $\mathsf{Pay}(sk_{\mathrm{U}}, par_c, \Upsilon_s, T)$. For each entry $(d_{\mathrm{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc) \in T$, calculate $price = \Upsilon(cons, other)$, run $(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price)$ and calculate a non-interactive witness-indistinguishable proof $\pi$:[5]

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

  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 message $p = (fee, open_{fee}, \{sc_i, d_{\mathrm{M}_i}, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^N)$. Compute a signature[6] $s_p = \mathrm{Usign}(sk_{\mathrm{U}}, p)$ and set a payment message $Q = (p, s_p)$.
- $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, Q, d_{\mathrm{P}})$. Parse $Q$ as $(p, s_p)$ and run $\mathrm{Uverify}(pk_{\mathrm{U}}, s_p, p)$. Output $b = 0$ if it rejects. Else parse $p$ as $(fee, open_{fee}, \{sc_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^N)$ and, for $i = 1$ to $N$, increment $d_{\mathrm{P}}$, run $\mathrm{Mverify}(pk_{\mathrm{M}}, sc_i, \langle d_{\mathrm{P}}, c_{cons_i}, c_{other_i}\rangle)$ and verify $\pi_i$. 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 else $b = 0$. Output $(b, d_{\mathrm{P}})$.
- $\mathsf{Reveal}(sk_{\mathrm{U}}, T, i)$. Pick the tuple $r = (i, cons, open_{cons}, other, open_{other})$ in the entry $(i, \ldots) \in T$, sign $s_r = \mathrm{Usign}(sk_{\mathrm{U}}, r)$ and output $R = (r, s_r)$.
- $\mathsf{VerifyReveal}(pk_{\mathrm{U}}, par_c, Q, R, j)$. Parse $Q$ as $(p, s_p)$ and $p$ as $(fee, open_{fee}, \{sc_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^N)$. Pick the tuple $(sc_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i)$ 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 else $b = 0$.

Figure 3: Cryptographic functions called by Protocol PSM.

$sp_i = \mathrm{Psign}(sk_{\mathrm{P}}, \langle other_i, price_i\rangle)$, and sets $\Upsilon_s = (other_i, price_i, sp_i)_{i=1}^n$. To compute a proof $\pi$, U uses the commitments to the consumption $c_{cons}$ and to other parameters $c_{other}$ included in $sc$, and commits to the total price $price_t$ ($(c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t)$). U then computes a proof of possession of a signature $sp \in \Upsilon_s$ 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 $price \cdot cons$:

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

**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 $\Upsilon$ over the interval $[0, cons]$. For instance, let $\Upsilon$ be a policy as follows[7]: $[0, 3] \to 2$, $(3, 7] \to 5$, $(7, \infty) \to 8$, 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

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

required that intervals defined by $[cons_{min}, cons_{max}]$ be disjoint. $F$ is the definite integral of $\Upsilon$ over the interval $[0, cons_{min}]$.

To sign this policy, for $i = 1$ to $n$, P runs $sp_i = \mathsf{Psign}(sk_{\mathrm{P}}, \langle cons_{min_i}, cons_{max_i}, F_i, other_i, price_i \rangle)$, and sets $\Upsilon_s = (cons_{min_i}, cons_{max_i}, F_i, other_i, price_i, sp_i)_{i=1}^{n}$. 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 $\pi$, 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 $sp \in \Upsilon_s$ 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 $price_t = (cons - cons_{min}) \times price + F$:

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

**Other Policies.** Another possible policy $\Upsilon$ 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 6). The price due is the evaluation of $\Upsilon$ for $x = cons$.

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 $sp_i = \mathsf{Psign}(sk_{\mathrm{P}}, \langle a_{Ni}, \ldots, a_{0i}, other_i \rangle)$, and sets $\Upsilon_s = (a_{Ni}, \ldots, a_{0i}, other, sp_i)_{i=1}^{n}$. To compute a proof $\pi$, 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 $sp \in \Upsilon_s$ 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 $price_t = \sum_{i=0}^{N} a_i cons^i$:

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

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.

## 5.3 Fast-PSM for public linear policies

In the previous construction, the policy $\Upsilon$ consisted of several formula 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 $(cons, other)$.

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 $\Upsilon$ consists of linear formula 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[8]. This construction is based on the use of a commitment scheme provided with two operations $\otimes$ and $\odot$ (see Section 4) that allow the computation of a commitment to the price, given a commitment to the consumption value.

The protocol is described in Figure 5, and the functions called are detailed in Figure 6.

The 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 $id_{\Upsilon}$ is introduced to ensure that U and P employ the policy published previously by P to compute and verify the payment message.

## 5.4 Discussion

We discuss here possible optimizations of the scheme, as well as modifications needed when it is applied to certain settings or if an adversary corrupts more than one party.

[8] This is the case for time-of-use billing in smart-grids.

**Protocol Fast-**PSM

- **Initialization.** When P is activated with $(\mathsf{policy}, \Upsilon)$, where $\Upsilon$ is a linear policy, P publishes a unique policy identifier $id_\Upsilon$ and sends $(id_\Upsilon, \Upsilon)$ to U.
- **Consumption.** Works as in the Protocol PSM.
- **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 $\mathsf{EffVerifyPayment}(pk_\mathrm{M}, pk_\mathrm{U}, par_c, id_\Upsilon, Q, d_\mathrm{P})$ to obtain $(b, d'_\mathrm{P})$. If $b = 0$, P rejects the payment, and otherwise accepts it and sets $d_\mathrm{P} = d'_\mathrm{P}$.
- **Reveal.** Works as in the Protocol PSM.

Figure 5: The fast protocol for billing in smart-grids.

**Efficient functions for Protocol Fast-**PSM

- $\mathsf{EffPay}(sk_\mathrm{U}, par_c, id_\Upsilon, \Upsilon, T)$. For each table entry $(d_\mathrm{M}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc) \in T$, 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 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 $p = (id_\Upsilon, fee, open_{fee}, \{sc_i, d_{\mathrm{M}i}, c_{cons_i}, c_{other_i}\}_{i=1}^{N})$. Compute a signature[9] $s_p = \mathrm{U}\mathsf{sign}(sk_\mathrm{U}, p)$ and set a payment message $Q = (p, s_p)$.
- $\mathsf{EffVerifyPayment}(pk_\mathrm{M}, pk_\mathrm{U}, par_c, id_\Upsilon, Q, d_\mathrm{P})$. Parse $Q$ as $(p, s_p)$ and run $\mathrm{U}\mathsf{verify}(pk_\mathrm{U}, s_p, p)$. Output $b = 0$ if it rejects. Otherwise parse $p$ as $(id_\Upsilon{}', fee, open_{fee}, \{sc_i, d_i, c_{cons_i}, c_{other_i}\}_{i=1}^{N})$, check that $id_\Upsilon = id_\Upsilon{}'$ and, for $i = 1$ to $N$, increment $d_\mathrm{P}$ and run $\mathrm{M}\mathsf{verify}(pk_\mathrm{M}, sc_i, \langle d_\mathrm{P}, c_{cons_i}, c_{other_i}\rangle)$. 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 by computing $c_{fee} = \otimes_{i=1}^{N} c_{price_i}$ and execute $\mathsf{Open}(par_c, c_{fee}, fee, open_{fee})$. If the output is $\mathsf{accept}$, set $b = 1$ else $b = 0$. Output $(b, d_\mathrm{P})$.

Figure 6: Efficient pay & verify functions.

**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 value 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 12) in order to improve efficiency.

Additionally, in the construction above, for each tuple $(cons, other)$ output by M, U 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 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 primarily to allow U to start the computation of the payment from the beginning of the billing period, when the total fee is still 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.

**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 fewer tuples in order to pay less. This is suitable in applications like electricity metering, where 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 number 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 13), 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 were already corrupted at the setup phase (later on, no communication from the meter to the provider 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, at the cost of proving possession of more signatures. In the modified construction, M outputs signatures on the tuples $(d, cons, other)$ and does not compute any commitment. Then, instead of revealing the signature to P, U 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 combined with the proof of possession of a signature on $(cons, other, price)$ given in the signed policy $\Upsilon_s$. In this modified construction, the zero-knowledge property of proofs and the hiding property of commitments (computed with randomness chosen by U), ensure no information is revealed to P.

Finally, we note that in the construction described above, the pricing policy $\Upsilon$ is sent by P to U at the initialization phase and is not updated later. 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.

# 6 Implementation

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 5.

**Commitment Scheme.** We choose the integer commitment scheme proposed by Groth [23].

**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 [9].

**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 5. 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 [17]. 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:

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

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 by Groth [23] to prove that an integer $m \geq 0$.

# 7 Security Evaluation

In Appendix 13 we prove that protocol PSM realizes the ideal functionality $\mathcal{F}_{\mathrm{PSM}}$. We do not consider the cases where all the parties are honest, where all the parties are dishonest or where the user U and the provider P are 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 5.4.

When P is dishonest, we claim and prove indistinguishability between real and ideal world under the unforgeability of the signature schemes $(\mathrm{M}\mathsf{keygen}, \mathrm{M}\mathsf{sign}, \mathrm{M}\mathsf{verify})$ and $(\mathrm{U}\mathsf{keygen}, \mathrm{U}\mathsf{sign}, \mathrm{U}\mathsf{verify})$, under the hiding property of the commitment scheme and the extractability and witness-indistinguishability of proofs of knowledge. 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 of the consumption readings and the pricing policy.

When U is dishonest, we prove indistinguishability under the unforgeability of the signature schemes $(\mathrm{M}\mathsf{keygen}, \mathrm{M}\mathsf{sign}, \mathrm{M}\mathsf{verify})$ and $(\mathrm{P}\mathsf{keygen}, \mathrm{P}\mathsf{sign}, \mathrm{P}\mathsf{verify})$, under the binding property of the commitment scheme and under the extractability and zero-knowledge property of proofs of knowledge. This proof ensures that the total fee calculated by U is correct.

# 8 Performance Evaluation

We implemented all the functionality required to generate keys, policies, prove bills and verify bills in C++. 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 specific code spans about 250 lines of code – including measurement code.

The reference platform for our measurements is an 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

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

Table 1: 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 | $\text{sec}^{-1}$ | ticks | $\text{sec}^{-1}$ |
| Prove bill | 48 ticks | (298295) | 59 ticks | (242681) |
| Verify bill | 158 ticks | (90621.4) | 504 ticks | (28409.1) |

Table 2: Fast protocol timings. Proof and verification per reading (amortised and averaged over 1000 readings). Policy packaging requires no cryptography.

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. For each reading the user needs to compute a zero-knowledge proof of several statements: possession of a CL signature, range proof including proving twice the decomposition of integers into 3 sums of squares, proof that a value is the result of multiplying two committed values and proofs of linear operations.

- **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. In this case the user only uses the homomorphisms of commitments to calculate the final bill.

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 Appendix 12.)

Table 1 illustrates the time required to create policies, prove and verify bills for the generic protocol setting, while Table 2 illustrates the fast protocol setting. Table 3 describes the sizes of the bill and its proof for different settings.

| **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 3: 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).

Verifying bills is about twice as slow as generating bills in the generic protocol, due to aggressive precomputations 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 weeks of billing data depending on the security parameter.

The fast protocol is extremely efficient: generating bill proofs requires a few tens of ticks since it does not involve any exponentiation. 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 these will have a dramatic impact on the performance of the protocols. Similarly, if proof size was a crucial factor, an elliptic curve could be used to implement the fast protocol to reduce bill sizes to about 20 KBytes. Since these proofs are transmitted over commodity broadband networks, so we did not implement the techniques for minimizing their size.

**Minimal meter overhead.** We optimized our protocols to impose a minimal communication, storage and computation overhead on meters.

Assuming a non privacy preserving meter outputs readings in clear in batches with a single signature over the batch, we can augment it to be privacy preserving with *no communication overhead*. The meter computes commitments to the readings, using opening values derived from a pseudo-random stream keyed with a sym-

metric key shared with the user. The commitments are signed, but *not transmitted*. The meter transmits only readings (possibly encrypted if a WAN is to be used) and a single batch signature, leading to no overhead. The user reconstruct the commitments using the readings and the derived opening values, for use in further computations and proofs.

The only additional storage required in the meter is a symmetric key shared with the user (i.e. about 128 to 160 bytes). Computations of the commitments and signature are done on the fly, updating the state of a hash function, and discarding the commitments once they are part of the signature.

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) per 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, tamper resistant smart-cards or microcontrollers as documented in [7].

**Web-deployment evaluation.** The design rational for privacy preserving metering includes deployment of the schemes using web technologies, as illustrated in Figure 1. We implemented a billing back-end using off-the-shelf web technologies: the meter registers encrypted readings with a server in the cloud. Users can then access a billing portal, running on an ASP.NET server, that delivers an HTMP page with an embedded Silverlight 4 control to perform the privacy preserving computations for billing. The control runs on the web-client: it downloads and decrypts the readings and the tariff policy, computes and proves the bill, and uploads it for verification back to the server. The decrypted readings never leave the client side Silverlight 4 control.

To evaluate the performance of the web-deployment we used an Intel Core 2 Duo P9600 CPU, running at 2.66GHz, with 4GB RAM, running the 32 bit Windows 7 OS. The machine was running both the client (Internet Explorer 8 Browser with Silverlight 4) and the server software (bundled with Visual Studio 2010). The .NET 4 and Silverlight 4 big number library was used to implement both the proofs and verification of the Fast-PSM scheme in 641 lines of C# code.

The performance of computing, proving and verifying bills within the browser and on the server side is entirely consistent with responsiveness requirements of web applications. Proving the bill for 7 days (336 readings) in the Silverlight 4 control took 190ms, while verifying the bill took on the server side 107ms. It is clear that the web-platform big number performance on the client lags behind the server side libraries. Yet, proving the bill for a few months of readings takes only a couple of seconds, demonstrating that our scheme is practical, and can be deployed within current client side web-applications.

## 9 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, display, local or wide area network telecommunications and remote upgrade capabilities. According to the PIA performed by NIST, time-of-use billing calculations require detailed readings. Other functions, such as load forecasting, efficiency analysis or demand-response can be performed through private calculations on certified readings or by pushing data to customers or using sampled, anonymized or aggregated readings volunteered by users.

**Meters.** Our scheme requires only a small sub-system to be adapted and secured, 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. We chose not to rely on the meters further to minimise the requirements for storage, communications, upgradability, and generic computations. This keeps the meter Trusted Computing Base small and as a result cheap and amenable to verification [29]. As argued by Garcia and Jacobs [21], independently verified tamper-resistant meters are also required for consumer protection.

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; 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.

Our system model differs from ongoing smart grid projects as we restrict the unidirectional communication from the metering core $\mathrm{M}$ and the provider $\mathrm{P}$ to protect privacy. A multi-level security policy [4] is defined to protect confidentiality: raw readings in the meter $\mathrm{M}$ are classified as "high" and the provider $\mathrm{P}$ systems are cleared only for "low" information, i.e. the final billing information. It is safe for information to flow "up" from the provider $\mathrm{P}$ to the meter $\mathrm{M}$, for example to push updates or commands. Any flow of information from the meter $\mathrm{M}$ to the provider $\mathrm{P}$ has to be mediated and declassified through the user $\mathrm{U}$ running our protocols to ensure it leaks only the final bill and, optionally, other information according to the policy.

Functions of the smart meter not related to consump-

tion 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.

**Private Computations.** Computations 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 on-line services.

A variety of tariff policies can be applied to the certified readings to calculate the final bill without changes to 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 all cases the integrity of the billing is guaranteed by the verification process. This property is similar to the "software independence" [36] that is sought in electronic election protocols.

Privacy 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. They can delegate the calculation to any third party entity they trust with their data, use any vendors product, or even write their own. Users can switch platform at will. This is in stark contrast to 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.

We implemented and evaluated a web-platform deployment setting, where the private computations are performed on a web-client using a downloaded control. Alternative deployments on smart devices, standalone applications on personal computers or fully delegating all computations (and privacy) to third party services are possible. They offer different trade-offs in terms of availability, interactivity and privacy for customers.

**Interoperability.** A key consideration when it comes to rolling out smart metering infrastructure is interoperability of meters among 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[10] to produce readings trusted by all parties. The rest of the smart meter can act in arbitrary ways without compromising the integrity of the computations. Different calculations can be performed on meter readings by different providers without updating the metrology unit. Such a scheme future-proofs the infrastructure: even if new methods are developed 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.

**Other applications.** The privacy-preserving metering and billing protocols are quite generic. They accommodate any situation in which certified readings are billed according to a policy, without revealing other information. In particular we can implement pay-as-you-drive insurance schemes with much simpler on-board units than assumed in previous solutions [39]. Similarly, we can implement complex road charging and tolling schemes without the need for spot-checks [3], but instead relying on the correctness of a simpler on-board unit. These applications are enabled by our use of re-randomizable signatures to efficiently implement look-up table functionality, also used to implement arbitrary non-linear functions as splines.

## 10 Conclusion

Privacy is a serious concern raised by smart-metering and failure to protect it has jeopardised smart-meter deployments in the Netherlands. Naively, it appears that a balance must be struck between the intrusion necessary for time-of-use 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 but provide 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 high-integrity calculations to be done on any device under the control of the user. The integrity of the bill is software independent and correctness is ensured through cryptographic verification. Our evaluation ensures the security of the scheme

[10] For example the UK SI 2006 No. 1679 "The Measuring Instruments (Active Electrical Energy Meters) Regulations 200".

through rigorous proofs and its practicality through a complete software implementation. It is striking that the Fast-PSM algorithm could verify 3 weeks of bills from 27 million UK homes in a few days on a single core of a modern PC.

An advantage of the proposed schemes is their flexibility: they offer different options on how to certify meter readings depending on the level 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 transmitting 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. Contrary to common wisdom dictates, they do not have to lead to privacy-invasion.

## Acknowledgment

We would like to thank Carmela Troncoso, Cedric Fourner, Markulf Kohlweiss, Dan Shumo, Mira Belenki, Tolga Acar, Brian LaMaccia, Manuel Costa, Miguel Castro, Eno Thereska, Thomas Simpson, Richard Harper, Rebecca Murphy, Ross Anderson and Fabian Uhse for discussions and comments that were most helpful in the preparation and presentation of this work.

## References

[1] Ross Anderson and Shailendra Fuloria. On the security economics of electricity metering. In *The Ninth Workshop on the Economics of Information Security*, 2010.

[2] Ross J. Anderson. Why information security is hard-an economic perspective. In *ACSAC*, pages 358–365, 2001.

[3] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. Pretp: Privacy-preserving electronic toll pricing. In *19th Usenix Security Symposium*, August 2010.

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

[5] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO '92*, volume 740, pages 390–420. Springer-Verlag, 1992.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS, year = 1993, pages = 62-73, ee = http://doi.acm.org/10.1145/168588.168596, bibsource = DBLP, http://dblp.uni-trier.de*.

[7] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard java card. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 600–610. ACM, 2009.

[8] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

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

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

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

[12] Ann Cavoukian, Jules Polonetsky, and Christopher Wolf. Smartprivacy for the smart grid: embedding privacy into the design of electricity conservation. In *Identity in the Information Society*, 2009.

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

[14] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[15] Colette Cuijpers.

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

[17] Morris Dwork. Cryptographic protocols of the identity mixer library, v. 2.3.0. IBM research report RZ3730.

[18] Costas Efthymiou and Georgios Kalogridis. Smart grid privacy via anonymization of smart metering data. In *First IEEE International Conference on Smart Grid Communications*. IEEE, October, 4-6 2010.

[19] Omid Fatemieh, Ranveer Chandra, and Carl 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.

[20] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[21] Flavio D. Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. Technical report, Radboud Universiteit Nijmegen, February 2010.

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

[23] J. Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, pages 467–482, 2005.

[24] Amir hamed Mohsenian-rad, Vincent W. S. Wong, Juri Jatskevich, and Robert Schober. 1 optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid.

[25] George W. Hart. Nonintrusive appliance load monitoring. In *Proceedings of the IEEE*, pages 1870–1891, December 1992.

[26] C. Laughman, Kwangduk Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong. Power signature analysis. *Power and Energy Magazine, IEEE*, (2):56–63.

[27] Michael LeMay, George Gross, Carl A. Gunter, and Sanjam Garg. Unified architecture for large-scale attested metering. In *Hawaii International Conference on System Sciences*, Big Island, Hawaii, January 2007. ACM.

[28] Mikhail Lisovich and Stephen Wicker. Privacy concerns in upcoming residential and commercial demand-response systems. In *2008 Clemson University Power Systems Conference*. Clemson University, March 2008.

[29] Jonathan M. Mccune. Reducing the trusted computing base for applications on commodity systems, 2009.

[30] Patrick McDaniel and Stephen McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7:75–77, 2009.

[31] Stephen McLaughlin, Patrick McDaniel, and Dmitry Podkuiko. Energy theft in the advanced metering infrastructure. In *4th International Workshop on Critical Information Infraestructures Security*, 2009.

[32] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 61–66, New York, NY, USA, 2010. ACM.

[33] T. Okamoto. An efficient divisible electronic cash scheme. In *CRYPTO*, pages 438–451, 1995.

[34] Elias L. Quinn. Privacy and the new energy infrastructure. *SSRN eLibrary*, 2009.

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

[36] R.L. Rivest. On the notion of software independence in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759, 2008.

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

[38] The Smart Grid Interoperability Panel. Smart Grid Cyber Security Strategy and Requirements. Technical Report 7628, National Institute of Standards and Technology.

[39] C. Troncoso, G. Danezis, E. Kosta, and B. Preneel. PriPAYD: privacy friendly pay-as-you-drive insurance.

[40] Andreas Wagner, Sebastian Speiser, Oliver Raabe, and Andreas Harth. Linked data for a privacy-aware smart grid. In *INFORMATIK 2010 Workshop - Informatik fr die Energiesysteme der Zukunft*, 2010.

APPENDIX

# 11 Definitions

## 11.1 Security Model

We define security following the ideal-world/real-world paradigm [11]. 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 $\psi$ is secure if there exists no environment $\mathcal{Z}$ that can distinguish whether it is interacting with adversary $\mathcal{A}$ and parties running protocol $\psi$ 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 $\psi$ 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 $\text{IDEAL}_{\mathcal{F}_\psi,\mathcal{E},\mathcal{Z}}$ and $\text{REAL}_{\psi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. We refer to [11] 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.

## 11.2 Privacy-Preserving Smart Metering

We first define an ideal functionality $\mathcal{F}_{\text{PSM}}$ for privacy-preserving smart metering (see Figure 7). Any construction that realizes $\mathcal{F}_{\text{PSM}}$ ensures that U pays the right fee for the consumption data output by M, i.e., that the fee is computed following $\Upsilon$. It also ensures that, if M and P do not collude, P only learns the fee paid, not the consumption data *cons* nor the other information *other* used to compute *fee*. Additionally, it ensures that a malicious provider cannot claim that the fee that U must pay is different from the one computed following $\Upsilon$.

Our construction operates in the $\mathcal{F}_{\text{REG}}$-hybrid model [11] (Figure 8), where parties register their public keys at a trusted registration entity. Below we depict the ideal functionality $\mathcal{F}_{\text{REG}}$, which is parameterized with a set of participants $\mathcal{P}$ that is restricted to contain M, U and P only. This functionality abstracts key management, which is a separate concern from privacy.

# 12 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 5. More details can be found in Appendix 12.

**Functionality $\mathcal{F}_{\text{REG}}$**

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

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

Figure 8: The $\mathcal{F}_{\text{REG}}$ functionality abstracting key management.

**Commitment Scheme.** We choose the integer commitment scheme due to Groth [23]. 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 $c = c'$.

**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 [9].

- $\mathsf{Keygen}(1^k)$. 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 $\pi = \mathsf{NIPK}\{(x_Z, x_{R_1}, \ldots, x_{R_l}) : Z = S^{x_Z} \wedge R_1 =$

**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 (policy, $\Upsilon$) from P, $\mathcal{F}_{\text{PSM}}$ stores $\Upsilon$ and sends (policy, $\Upsilon$) to U.
- On input (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 (consume, $cons$, $other$) to U.
- On input (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 (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 (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 (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 (revresp, $cons$, $other$, $b$) to P.

Figure 7: The $\mathcal{F}_{\text{PSM}}$ functionality defining the security properties of our scheme.

$S^{x_{R_1}} \wedge \ldots \wedge R_l = S^{x_{R_l}}\}$. 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 $l_n = 2048$, $l_r = 80$, $l_m = 256$, $l_e = 597$, $l_v = 2724$ ([17]).

**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 5. 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 [17]. 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 [6]. 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 [23] 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 [35]. The proof is:

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

# 13 Security Analysis

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

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 $\mathcal{E}$.

Therefore, in order to prove Theorem 1, 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 divide our proof of Theorem 1 into several claims. We prove security under static corruptions, and each of the claims proves indistinguishability when a different party is corrupted: when only the provider P is corrupted, when only the user U is corrupted, and when only the meter M is corrupted.

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.

## 13.1 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* (Mkeygen, Msign, Mverify) *and* (Ukeygen, Usign, 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.

- **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 $pk_{\mathrm{M}}$ and $pk_{\mathrm{U}}$ 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 $pk_{\mathrm{M}}$ and $pk_{\mathrm{U}}$, then |Pr [**Game** 1] − Pr [**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 $pk_{\mathrm{M}}$ and $\mathcal{A}$ did not receive a signature on $m$ verifiable with $pk_{\mathrm{M}}$. The probability that **Game** 2 aborts is bounded by the following lemma:

  **Lemma 1.** *Under the unforgeability of the signature scheme defined by algortithms* (Mkeygen, Msign, Mverify)*,* |*Pr* [***Game*** 2] − *Pr* [***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 $pk_{\mathrm{U}}$ and $\mathcal{A}$ did

not receive a signature on $m$ verifiable with $pk_{\mathrm{U}}$. The probability that **Game** 3 aborts is bounded by the following lemma:

**Lemma 2.** *Under the unforgeability of the signature scheme defined by algorithms* (Ukeygen, Usign, Uverify)*,* $|Pr\ [\textbf{\textit{Game}}\ 3] - Pr\ [\textbf{\textit{Game}}\ 2]| = \nu_2(\kappa)$.

- **Game** 4: This game proceeds as **Game** 3, except that **Game** 4 extracts the witness $td$ from the proof $\pi$. 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 $\Upsilon$. The proofs $\pi_i$ are replaced by proofs that prove knowledge of the opening of those commitments and of a signature $sp \in \Upsilon_s$ 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,* $|Pr\ [\textbf{\textit{Game}}\ 5] - Pr\ [\textbf{\textit{Game}}\ 4]| = \nu_4(\kappa)$.

$\mathcal{E}$ 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 $pk_{\mathrm{P}}$, $\mathcal{E}$ stores $pk_{\mathrm{P}}$. 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 $(par_c, \pi)$, $\mathcal{E}$ verifies $\pi$, extracts $td$ and stores $par_c$.

- **Initialization.** When $\mathcal{A}$ sends $\Upsilon_s$, $\mathcal{E}$ gets $\Upsilon$ from $\Upsilon_s$ 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 $(\mathsf{pay}, fee, N, b)$ from $\mathcal{F}_{\mathrm{PSM}}$, $\mathcal{E}$ 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 $\Upsilon$. (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 $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, cons'_i, other'_i, i)$ to obtain $SC$. $\mathcal{E}$ creates a table $T$ with the signed consumptions $SC$ and runs $\mathsf{Pay}(sk_{\mathrm{U}}, par_c, \Upsilon_s, T)$ 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 $(\mathsf{reveal}, i)$ to $\mathcal{F}_{\mathrm{PSM}}$. Upon receiving $(\mathsf{revresp}, cons, other, b)$ from $\mathcal{F}_{\mathrm{PSM}}$, $\mathcal{E}$ 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 $\mathsf{accept}$. $\mathcal{E}$ sets $r = (i, cons, open_{cons}, other, open_{other})$, signs $s_r = \mathsf{Usign}(sk_{\mathrm{U}}, r)$ and sends $R = (r, s_r)$ to $\mathcal{A}$. $\mathcal{E}$ aborts if $\mathcal{A}$ returns $(\mathsf{reject}, Q, R)$ 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 $|\Pr\ [\textbf{Game}\ 5] \leq \nu_5$. □

## 13.2 Security Analysis When User Is Corrupted

**Claim 2.** *When* U *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* (Mkeygen, Msign, Mverify) *and* (Pkeygen, Psign, 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 $\mathrm{REAL}_{\mathrm{PSM},\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ with non-negligible probability.

- **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 $pk_{\mathrm{M}}$ and $pk_{\mathrm{P}}$ and the commitment parameters $par_c$ are replaced by other values $pk_{\mathrm{M}}{}'$, $pk_{\mathrm{P}}{}'$ and $par_c{}'$ that are obtained by running Mkeygen, Pkeygen and ComSetup respectively. Since these values have the same distribution as $pk_{\mathrm{M}}$, $pk_{\mathrm{P}}$ 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 $\pi$ 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 $(price, open_{price}, cons, open_{cons}, other, open_{other}, sp)$ includes a message-signature pair $(\langle cons, other, price\rangle, sp)$ 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* $(\mathrm{Pkeygen}, \mathrm{Psign}, \mathrm{Pverify})$, $|Pr[\textbf{Game } 3] - Pr[\textbf{Game } 2]| = \nu_2(\kappa)$.

- **Game** 4: This game proceeds as **Game** 3, except that **Game** 4 aborts if any of the message-signature pairs $(\langle d, c_{cons}, c_{other}\rangle, sc)$ 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* $(\mathrm{Mkeygen}, \mathrm{Msign}, \mathrm{Mverify})$, $|Pr[\textbf{Game } 4] - Pr[\textbf{Game } 3]| = \nu_3(\kappa)$.

- **Game** 5: This game proceeds as **Game** 4, except that the proof of knowledge $\pi$ in the messages $(par_c, \pi)$ 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 $(sc_i, i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i)$ 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 $SC = (i, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$ 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 $\mathcal{F}_{\mathrm{PSM}}$ as described in our simulation below.

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

- **Initialization.** When $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{policy}, \Upsilon)$, $\mathcal{E}$ runs $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$ to get $\Upsilon_s$ and sends $\Upsilon_s$ to $\mathcal{A}$.

- **Consumption.** When $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{consume}, cons, other)$, $\mathcal{E}$ 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 $(\mathsf{payment}, fee, N)$, $\mathcal{E}$ sends $(\mathsf{payment})$ to $\mathcal{A}$. Upon receiving $Q = (fee', open_{fee'}, \{sc_i, d_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i=1}^{N'})$ from $\mathcal{A}$, $\mathcal{E}$ extracts the witness $(price, open_{price}, cons, open_{cons}, other, open_{other}, sp)$ of proofs $\pi$ and aborts if any of the conditions described in **Game** 3, **Game** 4 or **Game** 6 are fulfilled. $\mathcal{E}$ sends $(\mathsf{pay}, fee', N')$ to $\mathcal{F}_{\mathrm{PSM}}$.

- **Reveal.** When $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{reveal}, i)$, $\mathcal{E}$ 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 $(\mathsf{revresp}, cons, other)$ to $\mathcal{F}_{\mathrm{PSM}}$.

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

## 13.3 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 $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PSM}},\mathcal{E},\mathcal{Z}}$ with non-negligible probability.

- **Game** 0: This game corresponds to the execution of the real-world protocol with honest U and P. Thus Pr [**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 [**Game** 1] − Pr [**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 $(\mathsf{register}, pk_{\mathrm{M}})$ to register the public key $pk_{\mathrm{M}}$, $\mathcal{E}$ stores $pk_{\mathrm{M}}$. $\mathcal{E}$ 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 $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, SC, d)$ to get a bit $b$. If $b = 1$, $\mathcal{E}$ parses $SC$ as $(d_{\mathrm{M}}, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$ and sends $(\mathsf{consume}, cons, other)$ to $\mathcal{F}_{\mathrm{PSM}}$.

The distribution produced in **Game** 1 is identical to that of our simulation. We have that |Pr [**Game** 1] = 0. □

# 14 Other specific policies

### 14.0.1 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 $sp_i = \mathsf{Psign}(sk_{\mathrm{P}}, \langle cons_i, other_i, price_i\rangle)$, and sets $\Upsilon_s = (cons_i, other_i, price_i, sp_i)_{i=1}^n$. To compute the proof $\pi$, 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 $sp \in \Upsilon_s$ on $(cons, other, price)$ and equality between the signed values and the values committed to in $(c_{cons}, c_{other}, c_{price})$:

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

### 14.0.2 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 $sp_i = \mathrm{Psign}(sk_{\mathrm{P}}, \langle cons_{min_i}, cons_{max_i}, other_i, price_i\rangle)$, and sets $\Upsilon_s = (cons_{min_i}, cons_{max_i}, other_i, price_i, sp_i)_{i=1}^n$.[11] To compute a proof $\pi$, 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 $sp \in \Upsilon_s$ 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[12] $cons \in [cons_{min}, cons_{max}]$:

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

[11] We note that if $\Upsilon$ 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).

[12] 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 $cons \in [cons_{min}, \infty)$.