Privacy-Preserving Smart Metering

Alfredo Rial
K.U.Leuven, ESAT/COSIC & IBBT
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.

Categories and Subject Descriptors

K.4.1 [**Public Policy Issues**]: Privacy; K.4.4 [**Electronic Commerce**]: Payment schemes

General Terms

Security

Keywords

Billing, Smart metering, Cryptographic Protocol, Verifiable Computing

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 [9]. 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 [2]: 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 [23, 27] 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) [37] 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 [12], 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 privacy-preserving 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.

Our solution has the following advantages compared with other approaches.

- Complex non-linear tariff policies can be applied over individual meter readings or arbitrary periods of time

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

(i.e. per day, per week). Other calculations can also be performed and certified 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. We provide concrete constructions for complex non-linear policies, and an evaluation of their performance.

- 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. We have implemented a web-browser Silverlight control that performs private billing computations to illustrate the practicality of this approach on deployed web technologies.

- We have optimized protocols to require *no addtional communications* by the meter making it practical for immediate deployment. In fact, we have implemented and tested meter modifications on real-world meters in collaboration with a team from the Elster Group SE.

- When a simple tariff policy is applied, we can construct a very efficient protocol Fast-PSM for billing that requires no zero-knowledge proofs and is particularly well suited for time-of-use billing.

- Finally, our schemes have been shown to be cryptographically correct – i.e. our concrete protocols comply with an ideal functionality that expresses the privacy and integrity properties claimed.

Outline of the Paper. We discuss related work in smart metering privacy in Sect. 2. Then we present the requirements and purpose of our protocols in Sect. 3. In Sect. 4, we describe the cryptographic building blocks employed, we define security and we depict our schemes, including special cases for specific billing calculations. We evaluate our scheme for performance, as well as compatibility with current meters and web-technologies in Sect. 5. In Sect. 6, we describe applications of the scheme beyond the electricity metering setting. We conclude in Sect. 7.

Due to space constraint a number of aspects of our privacy metering schemes are not described in this paper, but are available in the full report[2]. These include:

- Full cryptographic proofs in the ideal-world/real-world model for our schemes.

- Different billing policies, including policies to encode splines and approximate arbitrary functions.

- A discussion of deployment and interoperability issues.

[2]`http://research.microsoft.com/apps/pubs/?id=141726`

2. RELATED WORK

Smart meters privacy concerns have previously been studied both from a technical [29, 30] and a legal perspective [9, 34]. These works propose enforcement of privacy properties based on procedural means and assume that fine-grained 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. [39] propose a privacy-aware framework for the smart grid based on semantic web technologies. Garcia and Jacobs [19] design a multiparty computation to compute the sum of their consumption privately. The NIST privacy subgroup [37] suggests anonymizing traces of readings, as proposed by Efthymiou et al. [16], 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 uses zero-knowledge proofs to achieve privacy in metering. Kumari et al. [25] propose usage control mechanisms for data shared by smart meters connected to web based social networks.

Some work focuses on more general aspects of smart grid security. Anderson and Fuloria [2] analyze the security economics of electricity metering. 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 [22]. Proposals to enhance the security of the smart grid infrastructure include Fatemieh et al. [17]. 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 (the special case of linear policies was independently studied in [24] – yet it does not include the opimizations to reduce meter communication costs we present, making their approach very expensive).

Smart metering is a special case of metering. LeMay et al. propose an architecture for attested metering [28] based on calculations performed on trusted hardware. Troncoso et al. [38] propose an architecture in which secure meters are used to calculate final bills for pay-as-you-drive insurance. Our protocol follows an approach similar to the one described in [3, 14] for the design of a privacy-friendly electronic toll pricing system. We extend their paradigm of proving some aspects of a metering system using cryptography by providing full end-to-end verifiability for computations.

Further Work. The work presented in this paper has already been extended in several works. In Kohlweiss et al. [13] it is extended to obscure inferences that can be drawn from the final bill using a combination of differentially private mechanisms and oblivious payments. Kursawe et al. [26] propose an aggregation protocol to privately sum readings from multiple meters, including protocols that are compatible with the protocols presented in this work, as well as making use of our low-communication overhead techniques to make aggregation practical. Finally, Fournet et al. have verified an implementation of the Fast-PSM protocol in Fine [36], and Aizatulin et al. [1] have done verification work on the concrete C implementation of our protocols for real-world meters. We will not be discussing further these extensions to the basic protocols presented here.

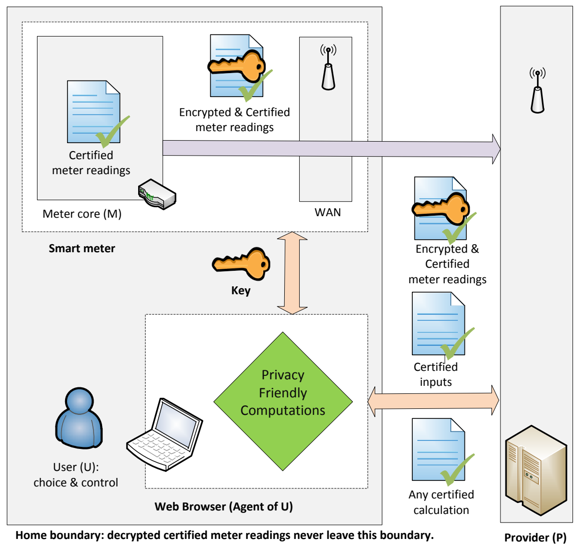
Encrypted & Certified meter readings
Certified meter readings
Meter core (M)
WAN
Smart meter
Key
Encrypted & Certified meter readings
Certified inputs
Privacy Friendly Computations
User (U): choice & control
Web Browser (Agent of U)
Any certified calculation
Home boundary: decrypted certified meter readings never leave this boundary.
Provider (P)

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 of 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, or other computations on readings, can evolve over time while the meters remain the same.

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. This allows users to choose clients and software they trust to perform the computations.

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

The basic operation of the system is as follows. First, the provider P sends the user U a pricing policy Υ. During a billing period, the meter M outputs consumption readings *cons* along with other information *other* that influences the cost. This output is collected by the user U who computes, 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 Υ and all the consumption measurements output by the meter M.

We present in Figure 2 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 to any time interval, i.e. per day, week, month, and policies can be composed readily without leaking additional information.

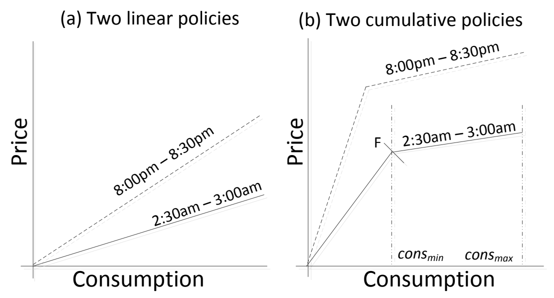

Figure 2: (a) A linear policy specifies the rate per unit consumption that is applied to determine the price to be paid for each measurement. The rate can be selected through information associated with the reading, like the time of the day or the location of a vehicle (without revealing this information). (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.

4. CONSTRUCTION

4.1 Cryptographic Building Blocks

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 [20] 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.

The signature schemes of M and U can be instantiated with any existentially unforgeable signature scheme – we have used the NIST Digital Signature Algorithm (DSA). For P's signature scheme, we choose the signature scheme proposed by Camenisch and Lysyanskaya [6].

Commitment schemes. A non-interactive commitment scheme consists of the algorithms $\mathsf{ComSetup}$, Commit and Open. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme par_c. $\mathsf{Commit}(par_c, x)$ outputs a commitment c_x to x and auxiliary information $open_x$. (We also employ the notation $\mathsf{Commit}(par_c, x, open_x)$, which outputs a commitment c_x to 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.

For our implementation we choose the integer commitment scheme proposed by Groth [21].

Proofs of Knowledge. A zero-knowledge proof of knowledge [4] 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 [35]; proof of knowledge of the equality of some element in different representations [10]; proof with interval checks [33], range proof [5] and proof of the disjunction or conjunction of any two of the previous [11]. These results are often given in the form of Σ-protocols but they can be turned into non-interactive zero-knowledge arguments in the random oracle model via the Fiat-Shamir heuristic [18].

When referring to the proofs above, we follow the notation introduced by Camenisch and Stadler [7] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. $\mathsf{NIPK}\{(\alpha, \beta, \delta) : y = {g_0}^{\alpha} {g_1}^{\beta} \wedge \tilde{y} = \tilde{g_0}^{\alpha} \tilde{g_1}^{\delta} \wedge A \le \alpha \le B\}$ denotes a "*zero-knowledge Proof of Knowledge of integers α, β, and δ such that $y = {g_0}^{\alpha} {g_1}^{\beta}$, $\tilde{y} = \tilde{g_0}^{\alpha} \tilde{g_1}^{\delta}$ and $A \le \alpha \le B$ 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 α, β, and δ, denote quantities whose knowledge is being proven, while all other values are known to the verifier. We denote a non-interactive proof of signature possession as $\mathsf{NIPK}\{(x, s_x) : \mathsf{Verify}(pk, x, s_x) = \text{accept}\}$.

For the specific policies implemented, 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. All the proofs in our implementation are computed via the Fiat-Shamir heuristic.

To prove possession of a Camenisch-Lysyanskaya signature, we employ the proof described in [15]. To prove that a message m_3 committed in $c_{m_3} = g_1^{m_3} h^{open_{m_3}}$ is the product of two messages m_1 and m_2 committed 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:

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

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 [21], based on expressing integers as the sum of 3 squares, to prove that an integer $m \geq 0$.

4.2 Construction Sketch

We consider a meter M, a user U and a provider P. 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 an additively homomorphic commitment scheme and reveals them to U and to M.

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

During a billing period, M obtains consumption values *cons* and outputs tuples $(bp, cons, other)$, where *bp* denotes the billing period. These tuples are signed as follows. First, M commits to *cons* and to *other*, and then computes signatures *sc* on the commitments and on *bp*. U is given the message-signature pairs and the openings of the commitments, i.e. $(bp, cons, other, \mathsf{Commit}(cons), \mathsf{Commit}(other), sc)$.

At the end of a billing period, U stores the signed policy given by P and a set of tuples $(bp, cons, other)$ 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 Υ;
2. a non-interactive zero-knowledge proof π 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 π.

Upon receiving the payment message, P verifies the signature by U, and the signatures by M on the commitments to $(cons, other)$ and on *bp*. P also verifies the proofs π. 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.

P is only given the total fee, but he is assured that the fee is correct in accordance with the pricing policy Υ and the consumption values output by M. U's privacy relies on the hiding property of commitments and on the zero-knowledge property of proofs [7]. 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.

4.3 Security Definition

Full definitions, constructions and proofs of security are available in the extended technical report[3].

We define security following the ideal-world/real-world paradigm [8]. In the real world, a set of parties interact according to the protocol description in the presence of a real adversary $\mathcal{A}$, while in the ideal world dummy parties interact with an ideal functionality that carries out the desired task in the presence of an ideal adversary $\mathcal{E}$. A protocol ψ is secure if there exists no environment $\mathcal{Z}$ that can distinguish whether it is interacting with adversary $\mathcal{A}$ and parties running protocol ψ or with the ideal process for carrying out the desired task, where ideal adversary $\mathcal{E}$ and dummy parties interact with an ideal functionality $\mathcal{F}_\psi$.

In the extended version of this work we show that the Protocol PSM is indistinguishable from the following abstract Functionality PSM:

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

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

- On input $(\mathsf{policy}, \Upsilon)$ from P, $\mathcal{F}_{\mathrm{PSM}}$ stores Υ and sends $(\mathsf{policy}, \Upsilon)$ to U.
- On input $(\mathsf{consume}, bp, cons, other)$ from M, $\mathcal{F}_{\mathrm{PSM}}$ appends $(cons, other)$ to a table T_{bp} that stores the consumptions of billing period bp. $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{consume}, bp, cons, other)$ to U.
- On input $(\mathsf{payment}, bp)$ from P, $\mathcal{F}_{\mathrm{PSM}}$ computes the total fee *fee* as follows. For all rows $i \in T_{bp}$, $\mathcal{F}_{\mathrm{PSM}}$ calculates $price_i = \Upsilon(cons_i, other_i)$. The fee is $fee = \sum_{i \in T_{bp}} price_i$. $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{payment}, fee, bp)$ to U and, if U is corrupted, $\mathcal{F}_{\mathrm{PSM}}$ receives $(\mathsf{pay}, fee', bp')$. If $fee \neq fee'$ or $bp \neq bp'$, $\mathcal{F}_{\mathrm{PSM}}$ sets $fee = fee'$, $bp = bp'$ and $b = 0$, and otherwise it sets $b = 1$. $\mathcal{F}_{\mathrm{PSM}}$ sends $(\mathsf{pay}, bp, fee, b)$ to P.

Any construction that realizes $\mathcal{F}_{\mathrm{PSM}}$ ensures that U pays the right fee for the consumption data output by M, i.e., that the fee is computed following Υ. It also ensures that, if M and P do not collude, P only learns the fee paid, 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 Υ. Our construc-

[3]`http://research.microsoft.com/apps/pubs/?id=141726`

tion operates in the $\mathcal{F}_{\mathrm{REG}}$-hybrid model [8], where parties register their public keys at a trusted registration entity.

4.4 Detailed Scheme

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 $(sk_{\mathrm{M}}, pk_{\mathrm{M}})$, U runs $\mathsf{Ukeygen}(1^k)$ to get a key pair $(sk_{\mathrm{U}}, pk_{\mathrm{U}})$ and P runs $\mathsf{Pkeygen}(1^k)$ to get a key pair $(sk_{\mathrm{P}}, pk_{\mathrm{P}})$. Each party registers its public key with $\mathcal{F}_{\mathrm{REG}}$ and retrieves public keys from other parties by querying $\mathcal{F}_{\mathrm{REG}}$.[4] P runs $\mathsf{ComSetup}(1^k)$ to get par_c and a trapdoor td, computes a proof $\pi = \mathsf{NIPK}\{(td) : (par_c, td) \leftarrow \mathsf{ComSetup}(1^k)\}$ and sends (par_c, π) to U and (par_c) to M. U verifies π.

Figure 3 illustrates the key interactions among P, U and M as well as the algorithms they execute. More formally, we define Protocol PSM as follows:

Protocol PSM

- **Initialization.** When P is activated with $(\mathsf{policy}, \Upsilon)$, P runs $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$ to get a signed policy Υ_s. P sends Υ_s to U. U runs $\mathsf{VerifyPolicy}(pk_{\mathrm{P}}, \Upsilon_s)$ to get a bit b. If $b = 0$, U rejects the policy. Otherwise U stores Υ_s.
- **Consumption.** When M is activated with the message $(\mathsf{consume}, bp, cons, other)$, M runs $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, bp, cons, other)$ to obtain a signed consumption SC. M sends SC to U. U runs $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, SC)$ 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_{bp} that stores the consumptions of the billing period bp.
- **Payment.** When P is activated with $(\mathsf{payment}, bp)$, P sends (bp) to U. U runs $\mathsf{Pay}(sk_{\mathrm{U}}, par_c, \Upsilon_s, T_{bp})$ to obtain a payment message Q and sends (Q) to P. P runs $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, bp, Q)$ to obtain a bit b. If $b = 0$, P rejects the payment, and otherwise accepts it.

The specific cryptographic algorithms used by the Protocol PSM are defined as follows:

- $\mathsf{SignPolicy}(sk_{\mathrm{P}}, \Upsilon)$. For each tuple $(cons, other, price) \in \Upsilon$, compute $sp = \mathsf{Psign}(sk_{\mathrm{P}}, \langle cons, other, price\rangle)$.[5] Let $\Upsilon_s = (cons_i, other_i, price_i, sp_i)_{i=1}^{n}$ be the set of message-signature tuples. Output Υ_s.
- $\mathsf{VerifyPolicy}(pk_{\mathrm{P}}, \Upsilon_s)$. For $i = 1$ to n, parse Υ_s as $(cons_i, other_i, price_i, sp_i)_{i=1}^{n}$, and, for $i = 1$ to n, run $\mathsf{Pverify}(pk_{\mathrm{P}}, sp_i, \langle cons_i, other_i, price_i\rangle)$. If any of the outputs is reject, output $b = 0$, else output $b = 1$.
- $\mathsf{SignConsumption}(sk_{\mathrm{M}}, par_c, bp, cons, other)$. Execute both $(c_{cons}, open_{cons}) = \mathsf{Commit}(par_c, cons)$ and $(c_{other}, open_{other}) = \mathsf{Commit}(par_c, other)$. Run $sc = \mathrm{Msign}(sk_{\mathrm{M}}, \langle bp, c_{cons}, c_{other}\rangle)$ and output $SC = (bp, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$.
- $\mathsf{VerifyConsumption}(pk_{\mathrm{M}}, par_c, SC)$. Parse message SC as $(bp, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc)$. Run $\mathsf{Open}(par_c, c_{cons}, cons, open_{cons})$ and $\mathsf{Open}(par_c, c_{other}, other, open_{other})$ and output $b = 0$ if any of them outputs reject. Run $\mathrm{Mverify}(pk_{\mathrm{M}}, sc, \langle bp, c_{cons}, c_{other}\rangle)$ and output $b = 0$ if the output is reject. Otherwise output $b = 1$.
- $\mathsf{Pay}(sk_{\mathrm{U}}, par_c, \Upsilon_s, T_{bp})$. For each table entry $(bp, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc) \in T$, calculate $price = \Upsilon(cons, other)$, run $(c_{price}, open_{price}) = \mathsf{Commit}(par_c, price)$ and calculate a proof π:[6]

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

  Compute the total fee $fee = \sum_{i\in T_{bp}} price_i$ and add all the openings $open_{fee} = \sum_{i\in T_{bp}} open_{price_i}$ to get an opening to the commitment to the fee. Set a message $p = (fee, open_{fee}, bp, \{sc_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i\in T_{bp}})$. Compute a signature[7] $s_p = \mathsf{Usign}(sk_{\mathrm{U}}, p)$ and set a payment message $Q = (p, s_p)$.
- $\mathsf{VerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, pk_{\mathrm{P}}, par_c, bp, Q)$. Parse Q as (p, s_p) and run $\mathsf{Uverify}(pk_{\mathrm{U}}, s_p, p)$. Output $b = 0$ if it rejects. Else parse p as $(fee, open_{fee}, bp, \{sc_i, c_{cons_i}, c_{other_i}, c_{price_i}, \pi_i\}_{i\in T_{bp}})$ and, for $i \in T_{bp}$ run $\mathrm{Mverify}(pk_{\mathrm{M}}, sc_i, \langle bp, c_{cons_i}, c_{other_i}\rangle)$ and verify π_i. Output $b = 0$ if any of the signatures or the proofs is not correct. Add the commitments to the prices $c'_{fee} = \otimes_{i=1}^{N} c_{price_i}$ and execute $\mathsf{Open}(par_c, c'_{fee}, fee, open_{fee})$. If the output is accept, set $b = 1$ and else $b = 0$. Output b.

4.5 Policies

We detail here the computation of the signed policy Υ_s, and how to prove that the price for a tuple $(cons, other)$ is computed in accordance with the right entry tuple $(cons, other, price)$ specified in the policy Υ. They differ depending on different types of policies Υ. We provide details of two policies: a linear policy that can be used to apply a different rate to each measurement and a "cumulative" policy that allows the application of a non-linear function to measurements in order to calculate their contribution to the bill (see Figure 2). The linear policy implements the tariff policy for smart-metering, where a different rate is applied per

[4]The key distribution authority is an abstraction: in practise all parties in the protocol, namely the Meter, User and Provider have previous off-line relationships that allow them to exchange signature keys securely. Our scheme does not require a Public Key Infrastructure (PKI), as no parties ever need to verify signatures of previously unknown entities.

[5]The way the tuples $(cons, other, price)$ are signed depends on the particular policy Υ to be signed. Examples are given in Section 4.5.

[6]The proof π also depends on the policy Υ. See Section 4.5.

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

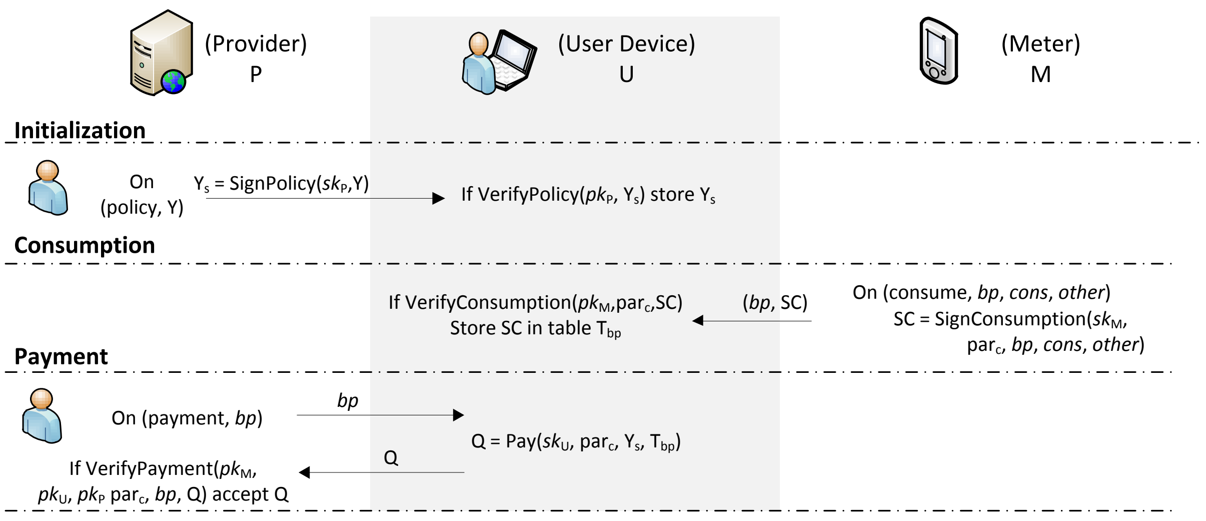
(Provider)
P
(User Device)
U
(Meter)
M
Initialization
On
(policy, Y)
Y_s = SignPolicy(sk_P,Y)
If VerifyPolicy(pk_P, Y_s) store Y_s
Consumption
If VerifyConsumption(pk_M,par_c,SC)
Store SC in table T_bp
(bp, SC)
On (consume, bp, cons, other)
SC = SignConsumption(sk_M, par_c, bp, cons, other)
Payment
On (payment, bp)
bp
Q = Pay(sk_U, par_c, Y_s, T_bp)
Q
If VerifyPayment(pk_M, pk_U, pk_P par_c, bp, Q) accept Q

Figure 3: Structure of protocols and use of cryptographic algorithms

half-hour according to the policy a customer has subscribed to. The cumulative policy illustrates the generality of the scheme.

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. These are special cases of the cumulative policy or linear policy, with some efficiency 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 discrete policy is only suitable when the set of possible consumption values is finite and small. Otherwise, signing all possible tuples $(cons, other)$ is inefficient. 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 $sp_i = \mathsf{Psign}(sk_\mathrm{P}, \langle other_i, price_i\rangle)$, and sets $\Upsilon_s = (other_i, price_i, sp_i)_{i=1}^{n}$. To compute a proof π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the total price $price_t$ $((c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t))$. 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 $price \cdot cons$:

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

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

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

To sign this policy, for $i = 1$ to n, P runs $sp_i = \mathsf{Psign}(sk_\mathrm{P}, \langle cons_{min_i}, cons_{max_i}, F_i, other_i, price_i\rangle)$, 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 π, U uses the commitments to the consumption c_{cons} and to other parameters c_{other} included in sc, and commits to the price $price_t$

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

$((c_{price_t}, open_{price_t}) = \mathsf{Commit}(par_c, price_t))$ 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 $price_t = (cons - cons_{min}) \times price + F$:

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

Other Policies. It is possible to combine a linear policy with an interval policy. Such a policy would describe a price per unit that depends on the interval in which the consumption value lies.

Another possible policy Υ is that defined by a polynomial function $\sum_{i=0}^{N} a_i x^i$ over a commutative ring R, which in our implementation is given by the integers modulo a composite. The price due is the evaluation of Υ for $x = cons$. 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.

4.6 Security Evaluation

THEOREM 1. *Protocol* PSM *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 $\mathcal{E}$.

We divide our proof of Theorem 1 into several claims.[9] We prove security under static corruptions, and each of the claims proves indistinguishability when a different subset of parties is corrupted. 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 4.8.

[9]The full security proof is available in the extended version at `http://research.microsoft.com/apps/pubs/?id=141726`

When P is dishonest, we claim and prove indistinguishability between real and ideal world under the unforgeability property 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. 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 property 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. This proof ensures that the total fee calculated by U is correct.

4.7 Fast-PSM for public linear policies.

In the previous construction, the policy Υ 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*). (consider the example policy "if consumption > 10 then fee = consumption * 5". The rate (5) has to be hidden to hide the fact that the consumption exceed the threshold).

In case the policy is simple, and only depends on parameters known by the provider (e.g. the application of a dynamic rate to each consumption period) an optimization is possible.

As the choice of formula depends on parameters already known by P, then the formula used does not need to be hidden. When Υ consists of linear formula of the form $price = r \cdot cons$, we provide an efficient construction that avoids the use of non-interactive zero-knowledge proofs. This construction is based on the use of a commitment scheme with an additive homomorphism that allows the computation of a commitment to the price, given a commitment to the consumption value. We note that U only needs to open the commitment to $\mathsf{Commit}(fee) = \mathsf{Commit}(\sum_{i=0}^{n} r_i \cdot cons_i)$ to P. This commitment can be independently computed and verified by P as $\mathsf{Commit}(fee) = \otimes_{i=0}^{n} \mathsf{Commit}(cons_i) \odot r_i$. In this case the computations for the prover and verifier are very efficient. We define Protocol Fast-PSM as follows:

Protocol Fast-PSM

- **Initialization.** When P is activated with (policy, Υ), where Υ is a linear policy, P publishes a unique policy identifier id_Υ and sends (id_Υ, Υ) to U.
- **Consumption.** Works as in the Protocol PSM.
- **Payment.** When P is activated with (payment, bp), P sends (bp) to U. U runs $\mathsf{EffPay}(sk_{\mathrm{U}}, par_c, id_\Upsilon, \Upsilon, T_{bp})$ to obtain a payment message Q and sends (Q) to P. P runs $\mathsf{EffVerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, par_c, id_\Upsilon, bp, Q)$ to obtain b. If $b = 0$, P rejects the payment, and otherwise accepts it.

- $\mathsf{EffPay}(sk_{\mathrm{U}}, par_c, id_{\Upsilon}, \Upsilon, T_{bp})$. For each table entry $(bp, cons, open_{cons}, c_{cons}, other, open_{other}, c_{other}, sc) \in T_{bp}$, calculate $price = a_1 \cdot cons + a_0$ and $open_{price} = open_{cons} \cdot a_1$. Calculate the total fee $fee = \sum_{i \in T_{bp}} price_i$ and add the openings $open_{fee} = \sum_{i \in T_{bp}} open_{price_i}$ to get an opening to the commitment to the fee fee. Set a payment message $p = (id_{\Upsilon}, fee, open_{fee}, bp, \{sc_i, c_{cons_i}, c_{other_i}\}_{i \in T_{bp}})$. Compute a signature $s_p = \mathsf{Usign}(sk_{\mathrm{U}}, p)$ and set a payment message $Q = (p, s_p)$.

- $\mathsf{EffVerifyPayment}(pk_{\mathrm{M}}, pk_{\mathrm{U}}, par_c, id_{\Upsilon}, bp, Q)$. Parse Q as (p, s_p) and run $\mathrm{Uverify}(pk_{\mathrm{U}}, s_p, p)$. Output $b = 0$ if it rejects. Otherwise parse p as $(id_{\Upsilon}', fee, open_{fee}, bp, \{sc_i, c_{cons_i}, c_{other_i}\}_{i \in T_{bp}})$, check that $id_{\Upsilon} = id_{\Upsilon}'$ and for all i run $\mathsf{Mverify}(pk_{\mathrm{M}}, sc_i, \langle bp, c_{cons_i}, c_{other_i} \rangle)$. Output $b = 0$ if any of the signatures 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 $c_{fee} = \otimes_{i \in T_{bp}} c_{price_i}$ and execute $\mathsf{Open}(par_c, c_{fee}, fee, open_{fee})$. If the output is accept, set $b = 1$ else $b = 0$. Output b.

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_{Υ} is introduced to ensure that U and P employ the policy published previously by P to compute and verify the payment message.

4.8 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.

Fewer commitments. In the construction depicted above, M commits separately to *cons* and to *other*. However, in applications where both parameters are always disclosed together M can commit to both values in a single commitment in order to improve efficiency.

Batch signatures & proofs. 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.

Explicit meta-data. We note that the scheme described above only works when P knows the amount of tuples that M outputs at each billing period, as it is the case for electricity metering. However, this may not be the case in other applications and U may report fewer tuples in order to pay less. To solve this problem, we can 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. Alternatively, a counter or the real time of readings can be encoded in *other*, and used in the proof to ensure no reading is omitted.

Corrupt Meter. A collusion between M and P against U is possible, in which the meter attempts to leak information through the commitments. Such collusion makes sense in practical applications, in which P is likely to provide the meters and thus can manipulate them. 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 $(bp, cons, other)$ 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 $(bp, cons, other)$ (bp 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 Υ_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.

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

Table 1: Fast protocol timings. Proof and verification per reading (amortised and averaged over 1000 readings). Policy packaging requires no cryptography.

5. PERFORMANCE EVALUATION

We implemented all the functionality required to generate keys, policies, prove bills and verify bills in 8200 lines of C++. The libraries provide generic support for expressing, generating and verifying computations on certified inputs. The smart-metering specific code spans about 250 lines of code – including measurement code.

Two reference billing problems on 1000 readings were considered, one very complex and one simple:

- **Generic Protocol.** A user needs to certify a bill using a complex pricing policy. This illustrates a non-linear policy composed of 100 linear segments 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 a bill using a linear public policy. This maps 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.

Two reference security parameters are evaluated, namely a 1024 bit and 2048 bit RSA modulus.

Table 2 illustrates the time required to create policies, prove and verify bills for the generic protocol setting, while Table 1 illustrates the fast protocol setting[10]. Table 3 describes the sizes of the bill and its proof for different settings.

[10]The reference platform for our measurements is a sin-

Generic PSM	1024 bits		2048 bits	
(per reading)	ticks	$\sec^{-1}$	ticks	$\sec^{-1}$
Gen. policy	147522	(97.0)	489754	(29.2)
Prove bill	162816	(87.9)	586586	(24.4)
Verify bill	344703	(41.5)	1270456	(11.2)

Table 2: Generic protocol timings. Policy generation per line of policy (amortised over 100 lines). Proof and verification per reading (amortised and averaged over 1000 readings).

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 kilobytes 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 pre-computations that are not available to the verifier (the cost of pre-computations is folded into the timing measurements). In real terms it takes from a few seconds to a few minutes to calculate and verify 3 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 computations only involve very small exponents. In real terms, 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.

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, which might be unecessary if U and P communicate over commodity broadband networks.

Minimal meter communication overhead & real meter implementation. We optimized our protocols to impose a minimal communication, storage and computation overhead on meters. A non privacy preserving meter that outputs signed batches of readings can be augmented to be privacy preserving with *no communication overhead*.

As readings are recorded, the meter computes commitments to the readings, using opening values derived from a pseudo-random stream keyed with a symmetric key shared with the user. The commitments are incrementally hashed and finally signed with DSA, but *never transmitted*. The meter transmits only (possibly encrypted) 4-byte readings and the single, batch, signature on the commitments, leading to no overhead (assuming the 4-byte readings and signature were transmitted even without our protocol). The user reconstructs the commitments using the readings, the shared key and the derived opening values, for use in further computations and proofs.

gle core of 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 executing 14318180 ticks/s. The reference platform is typical of the systems we expect providers and other verifiers to use.

The only additional storage required in the meter is a symmetric key shared with the user (i.e. 16 bytes). Computations of the commitments and signature are done on the fly, requiring very short buffers.

The computation at the meter consists of one commitment per reading per 15 or 30 minutes, and one signature on a set of commitments per billing period. We prototyped our protocol on current smart meters with the help of a team from the Elster Group SE, on their current generation of meters. The computation of each commitment took significantly less than 10 seconds using the naive OpenSSL exponetiation functions, demonstrating the approach is practical on current hardware[11]. We expect an optimized implementation of the cryptographic libraries to reduce this to less than 1 second per commitment. The protocol was fully integrated into their data collection platform, and confirmed to require no communication overhead.

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 HTML 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.

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 a bill for 7 days (336 readings) using the fast protocol in the Silverlight 4 control took 190ms, while verifying the bill took on the server side 107ms[12]. This demonstrates our scheme is practical, and can be deployed within current client side web-applications.

6. OTHER METERING AND BILLING 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. Many other applications require similar functionality.

In particular, a number of automotive applications can be satisfied:

- **Pay-as-you-drive (PAYD) insurance**: A meter is fixed to a car that records its GPS coordinates, speed,

[11]Due to commercial restrictions we cannot report extact timings, or the exact specifications of the metering platform.

[12]Our evaluation platform was 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.

distance travelled, time and date, and provides these as certified readings. A policy maps regions of GPS coordinates to a charging regime per distance travelled to calculate an insurance premium. This provides a full cryptographic solution to the PAYD problem, with a smaller trusted computing base than the one considered in PriPAYD [38].

- **Road charging and tolling**: As for the PAYD application, a black box in the car records its position and distance travelled. The car position can be used as key to look-up a table mapping rectangles to the tax premium or toll charge to be paid. Tariffs can change according to the time of day or predicted road congestion, incentivising drivers to avoid certain times or roads respectively. Our scheme avoids the need for the spot-checks necessary in [3], relying instead on the tamper-resistance of the meter for integrity.

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

7. CONCLUSION

Smart-meter privacy is a serious concern and failure to protect it has jeopardised the smart-meter deployments. 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 to be false and present a practical privacy-friendly metering system that does not leak any information while providing unforgeable bills based on complex dynamic tariff policies.

Our schemes use simple cryptography on the meters to certify readings and then off-load high-integrity calculations to any user device. The integrity of the bill is software independent and correctness is ensured through cryptographic verification. Our evaluation demonstrates the scheme's practicality: 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: 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.

Acknowledgment. We would like to thank Carmela Troncoso, Cedric Fournet, Markulf Kohlweiss, Mira Belenkiy, Dan Shumow, Tolga Acar, Brian LaMacchia, Miguel Castro, Eno Thereska, Manuel Costa, Thomas Simpson, Richard Harper, Rebecca Murphy, Michael John, Gergely Budai, Ross Anderson and Fabian Uhse for discussions and comments that were most helpful in the preparation and presentation of this work.

This work was supported in part by the Research Council K.U.Leuven: GOA TENSE (GOA/11/007), and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). Alfredo Rial is a Research Foundation - Flanders (FWO) doctoral researcher.

8. REFERENCES

[1] Mihhail Aizatulin, Andrew D. Gordon, and Jan Jürjens. Extracting and verifying cryptographic models from c protocol code by symbolic execution. 2011.

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

[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] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

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

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

[12] Colette Cuijpers. No to mandatory smart metering does not equal privacy!

[13] G. Danezis, M. Kohlweiss, and A. Rial. Differentially private billing with rebates. *Information Hiding*, 2011.

[14] 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.

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

[16] 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.

[17] 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.

[18] A. Fiat and A. Shamir. How to prove yourself:

Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

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

[20] 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.

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

[22] 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.

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

[24] Marek Jawurek, Martin Johns, and Florian Kerschbaum. Plug-in privacy for smart metering billing. *CoRR*, abs/1012.2248, 2010.

[25] Prachi Kumari, Florian Kelbert, and Alexander Pretschner. Data protection in heterogeneous distributed systems: A smart meter example. In *Dependable Software for Critical Infrastructures*, October 2011.

[26] K. Kursawe, M. Kohlweiss, and G. Danezis. Privacy-friendly aggregation for the smart-grid. *Privacy Enhancing Technologies*, 2011.

[27] 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.

[28] 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.

[29] 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.

[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 *BuildSys '10*. ACM, 2010.

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

[36] N. Swamy, J. Chen, C. Fournet, P.Y. Strub, and K.B.J. Yang. Secure distributed programming with value-dependent types. *Technical Report MSR-TR-2010-149, Microsoft Research Cambridge*, November 2010.

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

[38] Carmela Troncoso, George Danezis, Eleni Kosta, and Bart Preneel. Pripayd: privacy friendly pay-as-you-drive insurance. In Peng Ning and Ting Yu, editors, *WPES*, pages 99–107. ACM, 2007.

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