

Designing Privacy-preserving Smart Meters with Low-cost Microcontrollers

Andres Molina-Markham, George Danezis[†],
Kevin Fu, Prashant Shenoy, and David Irwin

University of Massachusetts Amherst

[†]Microsoft Research Cambridge

Abstract. Smart meters that track fine-grained electricity usage and implement sophisticated usage-based billing policies, e.g., based on time-of-use, are a key component of recent smart grid initiatives that aim to increase the electric grid’s efficiency. A key impediment to widespread smart meter deployment is that fine-grained usage data indirectly reveals detailed information about consumer behavior, such as when occupants are home, when they have guests or their eating and sleeping patterns. Recent research proposes cryptographic solutions that enable sophisticated billing policies without leaking information. However, prior research does not measure the performance constraints of real-world smart meters, which use cheap ultra-low-power microcontrollers to lower deployment costs. In this paper, we explore the feasibility of designing privacy-preserving smart meters using low-cost microcontrollers and provide a general methodology for estimating design costs. We show that it is feasible to produce certified meter readings for use in billing protocols relying on Zero-Knowledge Proofs with microcontrollers such as those inside currently deployed smart meters. Our prototype meter is capable of producing these readings every 10 seconds using a \$3.30USD MSP430 microcontroller, while less powerful microcontrollers deployed in today’s smart meters are capable of producing readings every 28 seconds. In addition to our results, our goal is to provide smart meter designers with a general methodology for selecting an appropriate balance between platform performance, power consumption, and monetary cost that accommodates privacy-preserving billing protocols.¹

1 Introduction

The goal of recent smart grid initiatives is to increase the electric grid’s efficiency by reducing both its monetary and environmental cost. One way to increase efficiency is to alter electricity demand by either shifting some of it to off-peak hours or better aligning it with intermittent renewable generation. Since directly controlling the grid’s electricity consumption, e.g., by forcibly disconnecting loads, is infeasible, smart grids focus on incentivizing consumers to change their own consumption patterns by altering the price of electricity to accurately reflect generation costs and aggregate demand.

¹ Paper currently under review.

A variety of billing policies that properly incentivize consumers are available to utilities. For instance, time-of-use (TOU) pricing alters the price for electricity (\$/kWh) based on the time of day, with peak daytime rates more expensive than off-peak nighttime rates. Utilities implicitly assume that TOU pricing requires them to know not only how much electricity consumers use each month, but also *when* they use it. Unfortunately, prior research demonstrates that fine-grained usage data indirectly reveals sensitive private information about a consumer’s activity patterns, e.g., when they are home, when they have guests, their eating and sleeping patterns, etc. [20]. Vast collections of fine-grained electricity data for many buildings over long periods of time raise both legal and economic concerns. To address these issues, researchers have proposed a variety of privacy-preserving billing protocols that prevent utilities from linking fine-grained usage patterns to individual households, but still allow them to implement sophisticated billing policies. The solutions draw on common cryptographic techniques, including commitment schemes, digital signatures and Zero-Knowledge Proofs (ZKP).

A key impediment to the widespread adoption of privacy-preserving billing protocols is the computational and memory constraints of smart meters, which, due to cost, size, and power considerations, typically use embedded microcontrollers. Prior work does not measure these resource constraints, and, thus, implicitly assumes that meters are capable of executing protocols in a reasonable amount of time. In this paper, we explore the economic feasibility of implementing the cryptographic techniques required for privacy-preserving smart metering, and propose a general methodology for evaluating the cost of a solution. We take into account current smart meter deployments and look at the hardware technologies utilities are adopting over both the short- and long-term. Our focus is on implementing cryptographic techniques on smart meters such as those proposed by Rial et al. [24], Molina-Markham et al. [20], Kursawe et al. [17] and Jawurek et al. [16]. However, our methodology also applies to estimating the cost of similar metering systems that require privacy, including natural gas, water, and toll roads, such as the one proposed by Balasch et al. [2]. We summarize our contributions below:

Implementation. We implement a privacy-friendly smart meter using low-cost microcontrollers from both the MSP430 and ARM families. We present the first experimental results that actually measure the performance of a Camenisch-Lysyanskaya (CL) based scheme using elliptic curves in constrained environments. Previous work [24] discusses and estimates, but does not include implementation results. The most comparable realization of a CL based scheme uses a Java Card [5] and does not include an elliptic curve version.

Cost Evaluation. We outline a cost evaluation strategy for implementing privacy-preserving smart meters that accounts for the special characteristics of low-cost microcontrollers and industry trends. In particular, we list a set of system variables that designers may modify to balance security, privacy, and cost. We are the first to discuss the issues surrounding ultra-low-power implementations, which in some applications may make the difference between a meter that requires a battery replacement every few years versus every few days.

Feasibility Analysis. We present evidence to support the hypothesis that ZKP billing protocols are feasible on current deployments of smart meters and cost effective on deployments over both the short- and long-term. Because some smart meters can be remotely updated, it is plausible that a deployment may be implemented in one of these updates. In the long-term, our experimental results may help system designers to assess the performance and cost benefits of utilizing elliptic curve primitives. Our analysis takes into account the evolution of the storage and computational capabilities of low-cost microcontrollers and contrasts it to the evolution of personal computer processors.

2 Cryptographic Building Blocks

This work builds on protocols for privacy-preserving calculations of time-of-use based bills for smart electricity metering. In that setting a customer fitted with a smart meter proves to a utility provider the amount to be paid for their electricity consumption within a specific time period, without revealing any details about their fine-grained consumption. The bill is calculated on the basis of detailed readings, every half hour or fifteen minutes, that are each billed according to the dynamic price of electricity at that time, or a pre-defined but time variable tariff scheme. These protocols are applicable when consumers do not trust the utility with their detailed electricity usage information, and the utility does not rely on consumers to honestly report their usage. Our work focuses on efficient implementations of the meter components on different families of processors necessary to support those protocols.

2.1 Commitment Schemes & Zero-Knowledge Proofs

Commitment schemes are cryptographic primitives that enable a party to create the digital equivalent of an envelope for a secret. Commitments support two important properties: *hiding* protects the secrecy of the committed message, and *binding* ensures it can only be opened to the committed message. Pedersen commitments [22] are information-theoretically hiding, and binding under the discrete logarithm assumption. They rely on a set of global parameters, namely a group G of prime order p with generators g and h . Under that scheme a commitment C to message $r \in \mathbb{Z}_p$ is computed as $C = g^r h^o$ where o is an *opening* nonce chosen uniformly at random in \mathbb{Z}_p . Opening a commitment C involves disclosing the values r and o to a verifier. In addition to opening the commitment, efficient protocols exist for a prover to convince a verifier that they know the committed value without disclosing it. Fujisaki-Okamoto commitments [13] are similar to Pedersen commitments, except that they make use of a group of composite, hidden order instead of a group of prime order. They allow the committed value to be any integer, including negative integers. For our protocols we can use Pedersen or Fujisaki-Okamoto commitments depending on whether the meter needs to encode negative values or not.

For the purposes of time-of-use billing, the meter periodically commits to meter readings. Those commitments are signed and the customer can use the signature to prove functions of the bill to a verifier. Different signature schemes may be used to achieve different security properties. A standard signature scheme, such as DSA, can be used to ensure the integrity of any further statement proved on the basis of the meter readings. On the downside, it is not possible to eliminate covert channels that may allow a dishonest meter to signal some information back to the utility verifier. When meters are not trusted for privacy, a signature scheme that eliminates covert channels, such as Camenish-Lysyanskaya (CL) signatures [7], must be used to sign readings individually.

CL-signatures allow a requesting party to obtain a digital signature on a commitment from an authorized signer. In particular, Camenish and Lysyanskaya [7] provide efficient protocols for computing a signature on a commitment message, as well as for constructing zero-knowledge proofs of knowledge of a signature on a committed or encrypted message. Note that there are two digital signature schemes attributed to Camenish and Lysyanskaya; their earlier scheme [6] relies on the Strong RSA assumption, while the later scheme relies on a discrete-logarithm-based assumption (the LRSW assumption) [19]. CL-signatures [7] can be implemented using elliptic curve groups, as long as we have an efficient bilinear map that is non-degenerate. The key generation function, the signing function and the signature verification function for CL-signatures can be described as follows, using the notation in [25].

1. $CLKeyGen(1^k)$. Given a security parameter k , and the number of block messages to sign n , the signer generates the first part of their public key: $(p, \mathbb{G}, \mathbb{H}, g, h, e)$, such that there is a mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$, which is bilinear, non-degenerate and efficient to compute. The signer then chooses the following parameters for their private key: $x, y, z_1, \dots, z_n \in_R \mathbb{Z}_p$. Next, the signer uses these parameters to compute $X = g^x, Y = g^y$ and $Z_i = g^{z_i}$ for all $i \in [1, n]$. The public key is $pubkey = (p, \mathbb{G}, \mathbb{H}, g, h, e, X, Y, \{Z_i\}, \{W_i\})$, and the secret key is the public key concatenated with $(x, y, \{z_i\})$.
2. $CLSign((x, y, \{z_i\}), \{m_i\})$. To sign n blocks $\{m_i\}$, the signer first chooses $a \in_R \mathbb{G}$, and computes $b = a^y$. The signer then computes $A_i = a^{z_i}$ and $B_i = (A_i)^y$ for all $i \in [2, n]$. Finally, the signer computes

$$\sigma = a^{x+xy} \prod_{i=2}^n A_i^{xym_i}$$

The signature is $sig = (a, \{A_i\}, \{B_i\}, \sigma)$.

3. $CLVerifySign(pubkey, \{m_i\}, sig)$. The verifier performs the following computations and outputs *accept* if the following equalities hold:

$$\begin{aligned}
 e(a, Y) &= e(g, b) \\
 e(a, Z_i) &= e(g, A_i), \forall i \in [1, n] \\
 e(A_i, Y) &= e(g, B_i), \forall i \in [1, n] \\
 e(g, c) &= e(X, a) \cdot e(X, b)_1^m \cdot \prod_{i=2}^n e(X, B_i)m_i
 \end{aligned}$$

ZKPs make use of commitments and CL-signatures to prove to a third party an aggregate function of the committed readings without revealing the enclosed readings. The full billing protocol proposed by Rial et. al. [24] decomposes a bill’s proof of correctness into a small set of ZKPs – effectively proving the correctness of a commitment to the price component of each period of consumption separately before aggregating them and disclosing the final bill. All proofs in their scheme are non-interactive by using the well known Fiat-Shamir heuristic [12].

2.2 A Smart Metering Billing Protocol

Our study focuses on the efficient implementation of the meter cryptographic components for the Rial and Danezis [24] privacy preserving smart metering protocols. Proposals by [16] can easily be adapted to use the same meter components.

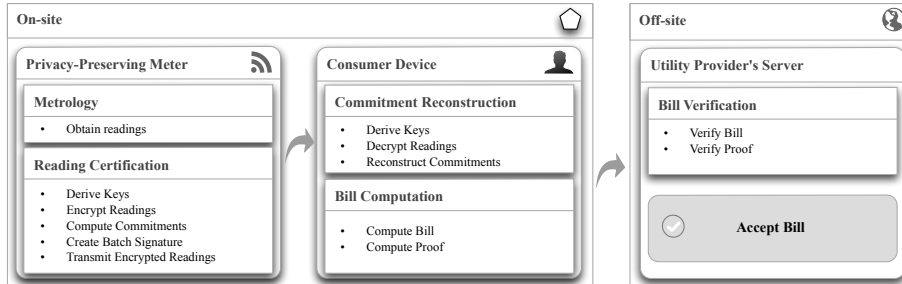


Fig. 1. Architecture of the privacy-preserving smart metering system. A smart meter, in addition to its metrologic unit, has a microcontroller capable of encrypting and certifying its readings. The meter also has a wireless transceiver used to send encrypted readings to the consumer’s device. The consumer uses the information from the meter for consumption planning, as well as in the computation of bills and corresponding proofs.

We illustrate the protocol with an example that includes three principals, as depicted in Figure 1: the smart meter, the prover, and the verifier. The smart

meter first measures and certifies consumer electricity readings, and then communicates them to the prover using a secure channel. The prover, a consumer-owned device, computes a bill along with a non-interactive ZKP that ensures the bill’s validity. The prover sends the bill and the proof to the utility company, which verifies the bill’s correctness before accepting it. Below, we describe in detail the computations the meter has to perform, and provide a brief outline of the protocols between the prover and the verifier.

Smart Meter Computations. To support privacy protocols, smart meters need to perform the following computations: sensing and measuring electricity usage, deriving session keys, certifying and encrypting readings, and finally transmitting readings to the consumer.

Sensing and measuring electricity. The meter’s primary function is sensing and measuring electricity usage. Thus, other computations must not interfere with this fundamental task. We denote Δt as the measurement interval, such that duration between meter readings $t_{i+1} - t_i = \Delta t$.

Deriving session keys. The protocol encrypts readings using a symmetric encryption algorithm before passing them to the user. To ensure the encrypted reading’s secrecy, each reading is encrypted with a distinct session key. For every t_i the meter encrypts reading r_i using key $K_i = H_0(K, t_i)$, where H_0 is a secure hash function and K is a master symmetric key known by the consumer. Additionally, the meter derives from the master key an opening value for the commitment $o_i = H_1(K, t_i)$ where H_1 is a hash function.

Certification and encryption. After deriving K_i and o_i , the meter both encrypts the reading r_i using K_i and computes a *commitment* c_i for the reading. More formally, the meter generates an encrypted reading $Er_i = E(K_i, r_i)$ using a symmetric encryption algorithm, and a commitment $c_i = g^{r_i} \cdot h^{o_i}$ using globally known constants g , h , and their group. The protocol also requires the meter to generate cryptographic signatures for each commitment c_i . To reduce the necessary computations, the protocol computes batch signatures Sig_j for multiple commitments $c_i, c_{i+1}, \dots, c_{i+k}$.

Network transmission. After the meter encrypts readings and computes batches of signatures, it transmits the batches to the consumer’s device (the prover) via the local network. More formally, for each batch j , the meter transmits the following tuples to the consumer: $\{\{t_i\}_j, \{Er_i\}_j, Sig_j\}$. The commitments need not be transmitted, which keeps the overheads of the protocol low.

Consumer Prover Computations. The prover computes both the bill’s payment and its corresponding proof of correctness. First, the prover derives the session keys $K_i = H_0(K, t_i)$ on the basis of times t_i and the master key K ; decrypts the readings r_i from $Er_i = E(K_i, r_i)$, and derives the opening values from each commitment as $o_i = H_1(K, t_i)$. Then all commitments to readings can be reconstructed as $c_i = g^{r_i} \cdot h^{o_i}$ using the public parameters of the commitment scheme and the recovered readings and openings. Finally, a batch of commitments can be accepted as authentic by checking the signature Sig_j . This ensures that the received encrypted readings have not been tampered with. After the readings and their signed commitments are available, an arbitrary billing func-

tion can be applied to each reading (or aggregates of readings) to establish the final bill. A ZKP of correctness has to be calculated by the prover, and provided to the verifier.

Summary. The details of those computations, and families of functions that can be practically proved and verified in zero-knowledge are provided in [24] along with the detailed security proofs for the protocol. To summarize, fine-grained meter readings are only available to the consumer, while simultaneously allowing the consumer to self-calculate their bill and ensuring the utility that the consumer has not manipulated or under-reported the payment. Thus, the utility has a guarantee over each bill’s authenticity, and the consumer has a guarantee over their data’s privacy. To resolve disputes, the meter may optionally store readings and decryption keys to permit audits by a trusted third party.

3 Implementation on Low-Cost Microcontrollers

In this section we describe a few different implementations of the cryptographic primitives discussed in Section 2 that would be required to run on a smart meter. We start by pointing out that the computational capabilities of low-cost and ultra-low-power microcontrollers have not developed at the same pace as high-performance microprocessors employed in servers and personal computers. System designers should, therefore, use different means to evaluate the economic feasibility of a cryptographic solution in the low-cost spectrum of embedded devices. We present the set of design variables that we control in our various implementations with the purpose of illustrating their effects on performances and costs in subsequent sections.

3.1 Computing Capabilities of Low-Cost Microcontrollers

Moore’s law predicted that the number of transistors placed in an integrated circuit would double approximately every two years. This prediction, however, does not directly address two issues that are pertinent to microcontrollers. First, the production costs associated with maintaining this trend have not remained constant. Second, with the addition of more transistors, the problem of efficient power management has significantly increased [11]. As a consequence, microcontrollers that are often constrained by production costs and power budgets have not increased their computational capabilities at the same rate as microprocessors for servers and personal computers. In Figure 2 we illustrate this by showing the evolution in processing capabilities across different microprocessor technologies.

3.2 Design Variables

In this section, we enumerate a set of design variables that we consider in our implementations with the purpose of illustrating their effect on various properties of the system. As we show, some of these design variables correspond to features,

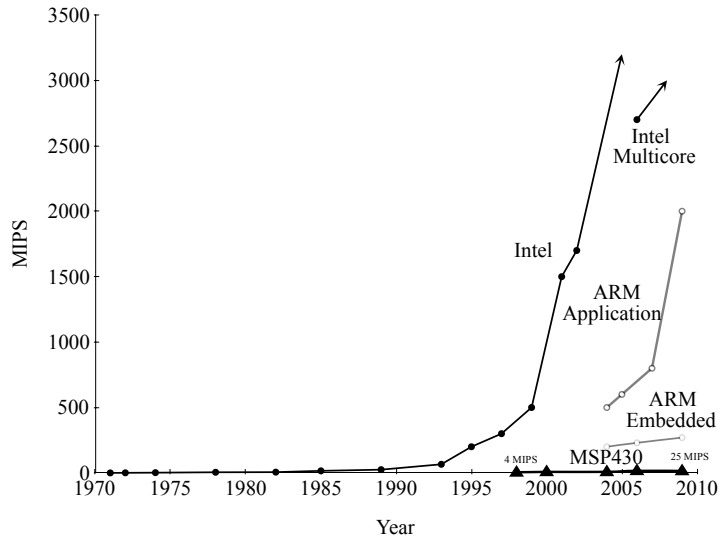


Fig. 2. While it is difficult to compare the performances of microprocessors using millions of instructions per second, this graph provides a visual representation of performance improvements as seen across a few popular architectures. The trends in microprocessors targeting desktop computers and servers, as well as the performance improvements observed in ARM application microprocessors have followed exponential curves. However, the performance improvements observed in embedded ARM microprocessors and MSP430 microcontrollers have followed linear curves [1, 8, 15].

such as qualitative privacy or security guarantees, e.g. properties of a trust or security model. Other design variables correspond to quantitative properties, for example computation performance, storage and communication requirements. The design variables that we consider in our implementation are in one of two categories, system variables or crypto variables. **System variables** include the selection of an MCU platform and a multitasking approach. **Crypto variables** include the selection of a digital signature scheme, and the selection of cryptographic primitives that rely on large integer multiplicative groups or elliptic curve cryptography.

3.3 Anatomy of a Smart Meter

In order to provide context for our discussion, we describe the generic anatomy of a smart meter. Figure 3 shows the schematics of a smart meter. In general, they are equipped with an analog front end, which is part of the metrologic unit used to convert the data coming from the load sensors and preprocess the measurements before they are passed to the microcontroller unit. The microcontroller unit handles this stream of data as well as the general functionality of storing the data in flash memory, and driving an LCD screen. More modern microcontroller units replace the analog front end with an integrated embedded signal processor.

Current deployments of smart meters have microcontroller units that can run at clock speeds ranging from 8-25 MHz and have storage ranging from 32 KB - 256 KB [14].

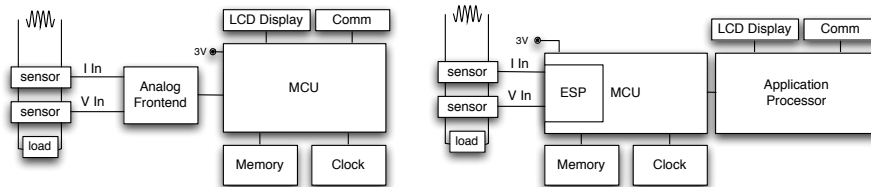


Fig. 3. Main components of a smart meter. The diagram on the left illustrates a simple meter with a single microcontroller unit (MCU) that controls the metrologic unit, storage and communication interfaces. The diagram on the right shows a smart meter that replaces the analog front end with an embedded signal processor (ESP) and has an additional application processor that controls communication, OS, power monitoring, and analytics.

3.4 Implementation Details

We implement the algorithms *Commit*, *CLSign* and *DSA*, using both large integer multiplicative groups and elliptic curve cryptography. We also implement the symmetric key derivation algorithm *DeriveAESKeys* to encrypt readings with AES for on-site wireless transmission. We integrate these algorithms to produce *certified readings*, as discussed in Section 2. In our experiments, we use the libraries `bnlib` [23] and `miracl` [26] to perform integer or elliptic curve arithmetic, together with one of the following Real-Time Operating Systems: `FreeRTOS` [4], `SYS/BIOS` [29] and `MicroC/OS-III` [18]. We write the rest of the implementation in C, with some minimal amount of assembly code. We describe the particular details of the ECC implementation in Section 3.5. We focus primarily on the MSP430 family of microcontrollers with a 16-bit RISC architecture. The motivation behind this focus is that current deployments already include microcontrollers in this family. We use the evaluation board MSP-EXP430F5438, in combination with the microcontrollers MSP430BT5190 and MSP430F5438A. The board includes an LCD screen and connectors for radio components. We also use the radio stack CC2567-PAN1327. Both microcontrollers are from the same family (MSP430x5xx). Shared characteristics include the availability of a hardware multiplier supporting 32-bit operations, size of flash (256 KB), frequency (25 MHz), and power consumption ($\sim 230 \mu\text{A}/\text{MHz}$ in active mode). The manufacturers designed the MSP430BT5190 for use with the radio stack; however, the MSP430F5438A has a larger RAM (16 KB).

3.5 Elliptic Curve Cryptography Details

The full ZKP based billing protocol requires the selection of various building blocks, such as commitment schemes and signatures. The security of these building blocks may depend on either the strong RSA (SRSA) assumption [6], or on the discrete logarithm (LRSW) assumption [7]. One important side-effect of the selection of these building blocks is that in order for the SRSA assumption to hold, the cryptographic operations need to be performed over multiplicative groups of integers with large moduli (1,024 to 2,048 bits in length). However, by leveraging modern Elliptic Curve Cryptography, the designer can use building blocks that rely on the discrete logarithm assumption employing considerably smaller key sizes. Therefore, for the ECC based commitments and ECDSA implementations, we use the NIST curves P-192 and P-224 [9]. For the ECC versions of the CL Signatures, we use the pairing-friendly elliptic curves $E(\mathbb{F}_{2^{379}}) : y^2 + y = x^3 + x + 1$ and $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$ with a 512-bit prime p as presented in [28].

The criteria for choosing curve parameters for ECDSA and the commitment scheme that we used are well known. However, choosing appropriate parameters for pairing-based cryptography is still an active area of research. That is, to use an elliptic curve implementation for CL-signatures, we require an appropriate bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$ that is non-degenerate and easy to compute. There is no unique way to obtain this map using elliptic curve groups \mathbb{G}, \mathbb{H} . While most protocols, such as signatures and identity based encryption protocols, are designed using a *type-1* pairing, it is often possible to use a *type-3* pairing. The latter are typically more efficient in practice. In other words, protocols often assume the existence of a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$ (type-1). However, in some cases the designer can implement a protocol that assumes the existence of a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{H}$ with $\mathbb{G}_1 \neq \mathbb{G}_2$ such that there is no isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ (type-3). We choose to implement type-1 pairings on a supersingular curve defined over $GF(2^m)$ using the η_T pairing [3] and on a supersingular curve defined over $GF(p)$ using a modified Tate pairing [27]. Note that in order for the curves to provide an adequate security guarantee, the size of the key must be large enough so that the corresponding dilogarithm problem in \mathbb{H} is hard. For the purposes of the particular billing protocol described in this paper, we note that a smart meter needs to compute signatures and not necessarily verify them. Therefore, we want to make operations on the curve as cheap as possible, even if that means computing more expensive pairings on the consumer's device. For more details on pairings, we refer the reader to Devegili et al. [10].

4 Experimental Evaluation

In this section, we evaluate the impact of choosing each of the design variables overviewed in Section 3. We first describe the impact of choosing a family of microcontrollers on overall computing performance. Next, we discuss the impact of

choosing an approach for multitasking on the RAM requirements and total cost size. Finally, we discuss the impact of choosing various cryptographic primitives.

4.1 Impact of Platform Selection

We implemented the cryptographic operations *Commit* and *CLSign* using microcontrollers from two of the most popular families, specifically, a microcontroller MSP430F5438A with 256 KB flash, 16 KB RAM and a microcontroller Stellaris LM3S9B92 (ARM Cortex M) with 256 KB flash, 96 KB SRAM; and two ARM application microprocessors OMAP3 (ARM Cortex A8) and OMAP4 (ARM Cortex A9) capable of running full Linux operating systems. These two microprocessors are commonly used in smart phones. The performances of these operations on these platforms are summarized in Table 1.

Table 1. Running time of commitments and signatures across multiple platforms. The tasks are run exclusively and uninterrupted on each of the platforms. The signatures are performed on 16 bytes of data. DSA uses a 1,024-bit prime p , a 160-bit prime q , and SHA-256. The timing does not include the generation of randomness, which depends on the source. Prices are in USD (Sept., 2011).

	MSP430F5438A	LM3S9B92	Cortex-A8	Cortex-A9
Operating Freq	25 MHz	80 MHz	720 MHz	1 GHz
Operating Power	330 - 690 μ W	333 - 524 mW	0.4 W	1.9 W
Family Price Range	\$0.25 - \$9	\$1 - \$8	\$41 - \$46	+\$50
Commitments - Key Size 1,024 bits				
Avg. Running Time	19.56 s	0.82 s	51 ms	36 ms
DSA Signatures - Key Size 1,024 bits				
Avg. Running Time	2.71 s	0.13 s	8 ms	6 ms
CL Signatures - Key Size 1,024 bits				
Avg. Running Time	43.1 s	2.3 s	150 ms	81 ms

4.2 Impact of Multitasking Approach

Meters need to be able to interrupt cryptographic computations periodically to perform measurements, logging and communication. One way of handling multitasking is with the use of an RTOS. Another way is the modeling of an application using a finite state machine and the implementation of it using timers and interrupts. Generally, the footprint of an RTOS is larger than the footprint of a state machine approach. We explore the following three RTOS in our work: FreeRTOS, SYS\BIOS and μ C-OSIII. Our configurations for each of the RTOS use 4 KB, 16 KB and 12 KB of code size respectively. The finite state machine requires approximately 2 KB of code. RTOS have the capability of managing

memory; some by reserving particular regions of the stack for different applications, and some by allowing for the use of dynamic memory allocation even with multiple heaps, such as `SYS\BIOS`. It is typically not a trivial engineering exercise to fit each cryptographic algorithm in RAM. We should also note that the system designer should probably base the decision of whether or not to use an RTOS on the necessity of additional required functionality, such as occasional tasks like secure updates, secure audits, key exchange and key revocation, etc.

4.3 Impact of ECC Utilization

In this subsection, we evaluate the impact of using elliptic curve cryptography instead of cryptography relying on large integer multiplicative groups. The code sizes of the `bnlib` [23] and `Miracl` [26] libraries and their RAM requirements depend on the features that are included. In our experimental setting using a microcontroller `MSP430F5438A`, the code size of `Miracl` was 23 KB and the code size of `bnlib` was 18 KB. The performance of `bnlib` and `Miracl` on non-ECC arithmetic is comparable. In our experiments, the running times of the same operation using either library differed by less than 5% of the total computation time of the operation. The RAM footprint for various functions is summarized in Table 2. As we can see, given a security level, ECC cryptographic primitives utilize RAM more efficiently. Similarly, Table 2 shows that given a microcontroller and a security level, an improvement in performance of about one order of magnitude can be achieved by using elliptic curve primitives.

Table 2. On the left we show the running time of commitments (single reading) and signatures (4 reading batches) on an `MSP430F5438A` at 25 MHz. These times are obtained when the algorithms are running exclusively and uninterrupted. We use `Miracl` for the elliptic curve versions as described in Section 3. The key sizes are in bits. On the right we show the RAM utilization for the various algorithms we implement on an `MSP430F5438A` all using the `Miracl` library. The measurements do not include RAM utilization by an RTOS, a radio stack or I/O.

Algorithm	Key Size	Library	Time	Algorithm	Key Size	RAM
Commit	1,024	bnlib	19.9 sec	Commit	1,024	5.8 KB
Commit	2,048	bnlib	303.0 sec	Commit	2,048	10.2 KB
ECC Commit	192	miracl	5.6 sec	CLSign	1,024	6.3 KB
ECC Commit	224	miracl	8.3 sec	CLSign	2,048	11.3 KB
CLSign	1,024	bnlib	41.2 sec	ECC Commit	192	2.2 KB
CLSign	2,048	bnlib	313.8 sec	ECC Commit	224	2.5 KB
ECC CLSign	379	miracl	6.7 sec	ECC CLSign	379	3.1 KB
ECC CLSign	512	miracl	35.6 sec	ECC CLSign	512	3.6 KB
AES Key Gen	128	miracl	0.1 sec	AES Key Gen	128	2 KB

4.4 Impact of Signature Scheme Selection

Table 2 shows running times for performing a `CLSign` algorithm with four readings. We highlight in particular the benefit of using an elliptic curve based library. If a designer uses elliptic curves, he or she can reduce a monthly batch

signature with 1,440 readings (one reading every half hour) from 15.6 hours to 2.5 hours. Furthermore, if the designer assumes a different trust level in which zero-knowledge is not required, signatures are less expensive. On an MSP430F5438A @ 25 MHz, signing a 16-byte message using regular DSA with a 1,024-bit prime p , a 160-bit prime q , and SHA-256 takes 2.71 seconds excluding the generation of randomness, which depends on the source. Signing a 16-byte message using ECDSA using a curve in $GF(p)$ for a 192-bit prime and SHA-256 takes 3.78 seconds excluding the generation of randomness. DSA signatures scale better than CL-signatures because the only overhead for a larger message would be the cost of the hash, which for the computations above is less than 0.01% of the computation.

5 Feasibility & Costs in Real-World Deployments

We now discuss a strategy for estimating the cost of deploying privacy preserving smart meters by matching the system variables that we discussed in our implementation in Section 3 to the cost of features in a microcontroller using a two step process.

5.1 Cost Estimation Strategy

Step 1: Determine the performance and power requirements. The first step in estimating the cost of implementing the cryptographic techniques described in Section 2 for the designer is to determine the acceptable levels of general computational performance and the power requirements of the meter. In other words, depending on the specific application, meter readings may need to be certified with a frequency of seconds, minutes or hours. Also, the meter may need to operate on a battery. Thus, using an ultra-low-power microcontroller may be the difference between replacing the battery every few years or every few days. For example, the performance shown in Table 1 may make the MCU Stellaris LM3S9B92 look very attractive for its ratio of cost/performance. However, the power consumption is roughly three orders of magnitude greater than the MSP430 MCU. Mobile processors are still far from being ultra-low power, although their computational and storage capabilities are increasing faster than those of the MCUs.

Step 2: Determine the code and RAM requirements. Once the performance and power requirements are met by a family of microcontrollers, it is then necessary for the designer to estimate the code size and RAM requirements for the implementation of the reading certification functions in a meter, taking into account whether multitasking needs to be supported.

5.2 Economic Feasibility

The implementation results in Section 4 support the hypothesis that privacy-preserving billing protocols based on ZKP are economically feasible. First, we

note that existing smart meters that have the ability to be remotely updated rely on microcontrollers similar to those that were used in our implementation. Furthermore, if a microcontroller in the MSP430 family is used to compute certified readings, it is possible to generate commitments and CL-signatures every 10 seconds when running at 25 MHz or every 28 seconds when running at a more conservative 8 MHz. Thus, a remote update that enables smart meters with privacy preserving functionality appears feasible.

Other metering applications may require that readings be certified at a finer granularity, for example every one or two seconds. This would require higher computational performance and larger storage than is currently available on low-cost ultra-low-power microcontrollers. For this reason, while obtaining certified readings at fine granularities is technologically feasible, it is to this date a feature that may incur a greater cost. Finally, in some circumstances, billing transactions (including communication) may be required to take milliseconds (real-time). In that case, only high-end mobile processors could provide the required performance, and thus the cost of that application would be high based on current technological trends.

We should note that, while in our analysis we did not cover all manufacturers of low cost MCUs, other leading manufacturers have similar offerings. For example *Atmel* also has AVR ultra low power microcontrollers, and various ARM based MCUs comparable to those discussed here. *Microchip* has the PIC microcontroller line with 8-, 16- and 32-bit MCUs. In our discussion, we did not consider 8-bit microcontrollers because they are perhaps too constrained for the kind of crypto application described here.

5.3 Best Utilization of Resources

The measurements in Section 4 show that the best security/cost ratio can be achieved by using ECC primitives. If current MCUs are targeted, maximizing the use of RAM can be achieved via ECC. Looking toward the future, performance will most likely regain importance due to the increasing economic feasibility of Ferroelectric RAM (FRAM), a kind of memory that enables high-performance on ultra-low power microcontrollers, with a unified memory model. Texas Instruments has started to ship MCUs with 16 KB of FRAM (\$1.20 USD), and they are already producing chips with 4 MB of FRAM [21].

6 Conclusion

Our evidence supports the notion that ZKP-based billing protocols are economically feasible. We also show that evaluating the cost of a cryptographic solution in an embedded system such as a smart meter depends first on the family of microcontrollers used, then on the storage and RAM requirements, and finally on additional features such as communication and user interface. Our empirical analyses show that with the use of Elliptic Curve Cryptography, it is possible to

reduce the RAM requirements by about 50% and obtain performance improvements of about one order of magnitude, thus obtaining a better performance/cost ratio.

7 Acknowledgments

This material is supported by a Sloan Research Fellowship, the NSF under CNS-0136228, CNS-136650, CNS-1143655, CNS-0916577, CNS-0855128 and a gift from Cisco. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1. ARM: ARM Company Milestones (2011), <http://www.arm.com/about/company-profile/milestones.php>
2. Balasch, J., Rial, A., Troncoso, C., Geuens, C., Preneel, B., Verbauwhede, I.: PrETP: Privacy-preserving electronic toll pricing. In: USENIX Security (2010)
3. Barreto, P., Galbraith, S., Ó'hÉigartaigh, C., Scott, M.: Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography* (2007)
4. Barry, R.: FreeRTOS-a free RTOS for small embedded real time systems (2006)
5. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous Credentials on a Standard Java Card. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (2009)
6. Camenisch, J., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: Security in Communication Networks (2003)
7. Camenisch, J., Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: Advances in Cryptology (2004)
8. Chen, J.: MSP430 Overview and Key Applications (2008)
9. Daley, W.: Digital signature standard (DSS). Tech. rep., DTIC (2000)
10. Devegili, A., Scott, M., Dahab, R.: Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In: Pairing-Based Cryptography (2007)
11. Dreslinski, R., Wieckowski, M., Blaauw, D., Sylvester, D., Mudge, T.: Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. Proceedings of the IEEE (2010)
12. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology (1987)
13. Fujisaki, E., Okamoto, T.: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In: Advances in Cryptology (1997)
14. Gas, P., Company, E.: PG & E: Full Installation Schedule
15. Intel: Corporate Timeline (2011), <http://www.intel.com/about/companyinfo/museum/archives/timeline.htm>
16. Jawurek, M., Johns, M., Kerschbaum, F.: Plug-in privacy for Smart Metering billing. In: Privacy Enhancing Technologies (2011)
17. Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly Aggregation for the Smart-grid. In: Privacy Enhancing Technologies (2011)
18. Labrosse, J.: MicroC/OS-III: The Real-Time Kernel. Micrium Press (2010)

19. Lysyanskaya, A., Rivest, R., Sahai, A., Wolf, S.: Pseudonym Systems. In: Selected Areas in Cryptography (2000)
20. Molina-Markham, A., Shenoy, P., Fu, K., Cecchet, E., Irwin, D.: Private Memoirs of a Smart Meter. In: Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (2010)
21. Pearson, J., Moise, T.: The Advantages of FRAM-Based Smart ICs for Next Generation Government Electronic IDs (2007)
22. Pedersen, T.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Advances in Cryptology (1992)
23. Plumb, C., Zimmermann, P.: bnlib: Extended Precision Integer Math Library
24. Rial, A., Danezis, G.: Privacy-Preserving Smart Metering. In: Workshop on Privacy in the Electronic Society (2011)
25. Rosenberg, B.: Handbook of Financial Cryptography and Security. Chapman & Hall/CRC (2010)
26. Scott, M.: MIRACL Multiprecision Integer and Rational Arithmetic C Library
27. Scott, M.: Implementing Cryptographic Pairings. In: Pairing-Based Cryptography (2007)
28. Scott, M., Costigan, N., Abdulwahab, W.: Implementing Cryptographic Pairings on Smartcards. In: Cryptographic Hardware and Embedded Systems (2006)
29. Texas Instruments: SYS/BIOS Real-Time Operating System (2011)