Smart Meter Aggregation via Secret-Sharing

George Danezis, Cédric Fournet, Markulf Kohlweiss, Santiago Zanella-Béguelin
Microsoft Research, Cambridge, UK
{gdane,fournet,markulf,santiago}@microsoft.com

ABSTRACT

We design and prototype protocols for processing smart-meter readings while preserving user privacy. We provide support for computing non-linear functions on encrypted readings, implemented by adapting to our setting efficient secret-sharing-based secure multi-party computation techniques.

Meter readings are jointly processed by a (public) storage service and a few independent authorities, each owning an additive share of the readings.

For non-linear processing, these parties consume pre-shared materials, produced by an off-line trusted third party. This party never processes private readings; it may be implemented using trusted hardware or somewhat homomorphic encryption.

The protocol involves minimal, off-line support from the meters—a few keyed hash computations and no communication overhead.

Categories and Subject Descriptors

K.4.1 [**Public Policy Issues**]: Privacy

1. INTRODUCTION

Smart metering deployments have been plagued with privacy concerns [9, 10]. As a result, data protection authorities have recognized that fine grained readings are personal information, and as such should be processed in line with data protection principles, such as "data minimization" [15] that dictates the minimal amount of information should be collected or made available to third parties to fulfill their business needs. Specific privacy technologies have been developed in the context of smart metering architectures, to minimize the information necessary to perform common tasks such as generating aggregate statistics [16, 24] or computing bills [12, 23] (see [13] for a survey).

This work aims to expand the reach of these privacy technologies significantly, by allowing the computation of complex, non-linear functions on encrypted meter readings. This functionality goes beyond linear bills and aggregates [16, 23, 24], and does not require interaction with a user device, as prior work did [23]. Yet, it uses a similar security model as previous mechanisms [11, 16, 18] with multiple honest-but-curious parties. Our solution is based on extending these mechanisms with ideas from modern secret-sharing-based secure multi-party computation literature [4, 6].

SEGS'13, November 8, 2013, Berlin, Germany.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2492-2/13/11 ...$15.00.
http://dx.doi.org/10.1145/2516930.2516944.

Throughout this paper we use motivating examples directly inspired by problems related to smart metering [8]:

- **Line Fraud detection.** Given encrypted readings of the voltage across the meters of two homes adjacent on a power line, a utility company wishes to determine if the voltage drop is suspiciously large. This may indicate someone stealing electricity directly from the line. Yet, ideally, the exact voltage at each meter or their exact difference should not be made available, unless there is a significant drop.
- **Meter short-cut detection.** In a normal setting, households adjacent on a power line should see a decreasing voltage across their meters. Yet, when someone shorts their meter to commit fraud, the reading will be lower than the two households on either side. This check needs to be made without revealing any other information about voltages.
- **Non-linear billing.** Demand-response systems direct or broadcast "back-off" signals to households at times of peak demand. Following such a signal, energy consumed above a predetermined threshold is billed at a very high tariff, to provide incentives to reduce demand. We show how such a non-linear billing mechanism can be implemented.
- **Advanced statistics.** Current aggregation schemes provide linear statistics, in the form of weighted sums over collections of readings. We show how to extract higher order statistics such as the variance of a collection of readings.

We will describe all building blocks we use to support the applications above, and evaluate their specific performance.

In the rest of this paper, we first review related work, on which we heavily rely; we then go on describing our system assumptions, adversary models, and security assumptions; we then give a description of the types of computations we support, their cost and how they compose; finally we present our preliminary experimental results.

2. BACKGROUND & RELATED WORK

A thorough survey of privacy technologies applicable to smart metering exists [13], so here we limit ourselves to works that are directly relevant to the problem at hand.

Smart metering privacy protocols. Privacy-friendly aggregation has received a lot of attention: the aim of those protocols is for a utility to reveal the sum of readings from multiple meters without learning the readings themselves. Kursawe et al. [16] concurrently with Shi et al. [24] have presented a set of efficient aggregation protocols. In both schemes, meters generate readings (r_i) that are blinded by additive shares (s_i) summing to zero (i.e. $\sum_i s_i = 0$, where all arithmetic is done modulo 2^{32}). Thus when the blinded readings ($b_i = s_i + r_i$) are revealed and summed, shares cancel out and the sum of the readings is revealed (i.e. $\sum_i b_i = \sum_i r_i$). These protocols are very efficient but suffer from inflexibility: the groups of meters that can be aggregated are static, and missing readings prevent the computation of the aggregate. Furthermore, the details of computing the shares at each meter require some data per other meter in each group, and thus the size of groups is restricted and their membership is static. Finally, only simple sums of readings can be computed, which is a valuable but restricted statistic. Yet, this protocol is advantageous enough to form the basis of a mid-size trial in the Netherlands [8].

A number of protocols attempt to improve the robustness of aggregation, and allow for missing readings without jeopardizing computations. Jawurek and Kerschbaum [11] propose a novel model for such protocols, which also forms the basis of our threat model and security assumptions. They consider the participation of additional "authorities" that facilitate the protocol, but do not learn any secrets (or readings) in the process. The key trust assumption is that at least one of the authorities is honest—if they are all corrupt and collude, then the protocol does not provide any privacy.

Also in a security model with authorities, Barthe et al. [1] propose an efficient robust aggregation protocol, as follows: for each reading r_i, the meter and each authority compute a set of shares that sum up to zero (i.e. $\sum_j s_{ij} = 0$). The meter then simply outputs a version of the reading blinded with its share (i.e. $b_i = r_i + s_{i0}$). With the help of all authorities, any linear aggregate of readings can be computed. For example, if the utility wishes to compute $R = \alpha \cdot r_1 + \beta \cdot r_2$, it would compute the linear sum on the blinded readings $B = \alpha \cdot b_1 + \beta \cdot b_2$, and request from each authority the same linear sum on the shares they hold for these readings $S_a = \alpha \cdot s_{1a} + \beta \cdot s_{2a}$, where a ranges over the authorities. The aggregate can then be computed as $R = B + \sum_a S_a$, and their approach generalizes to computing any weighted sum. The selection of readings to be summed can be dynamic, and thus missing readings can simply be omitted (or implicitly be assigned a weight of zero). Furthermore, multiple readings per meter can be used, leading to a very efficient implementation of a linear time-of-use billing mechanism. All values exchanged are small, computations are linear and on small integers, and the contributions from the authorities can be collected ahead of time. Finally, additive noise can be added to make the aggregate differentially private [1].

The obvious limitation of these protocols is their inability to compute anything non-linear, for instance involving multiplications of secret readings, thresholds, comparisons between secret readings, etc. Three broad approaches have been proposed to run arbitrary computations on encrypted readings. The obvious approach is to provide all necessary readings to a trusted third party, and allow them to be decrypted to compute any function in clear. Relying on a single fully-trusted party is dangerous since implementing them well is expensive, and they could be compromised, collude with an adversary, or be compelled to reveal secrets. A second approach is to rely on fully homomorphic encryption [25] but, in practice, such schemes remain too expensive for industrial applications, and still rely on the controlled decryption of final results [7]. The final approach is to reflect the encrypted readings to a user device, decrypt them there, compute any function, and return the result alongside a zero-knowledge proof of correctness [23]. This approach is surprisingly general, but its proofs can be expensive, it is only available for computing on data from a single household, and it requires interaction with some user device.

Computations using secret shares. The approach for performing computations on "encrypted" readings we present consists of a cut-down and optimized set of protocols inspired by recent work on secure multi-party computation using secret shares. In this setting, a user splits a secret into shares adding up to the secret value, i.e. $r_i = \sum_j s_{ij}$, where the shares are usually denoted as $\langle r_i \rangle$, and distributes them to a number of authorities. Unless all authorities collude, the secret remains confidential. The aim of those authorities is to use their shares to jointly compute on the secret values, without revealing any intermediate results—a general problem in the realm of *secure multi-party computation.*

Applying linear functions in the secrets shared is a relatively easy task, that relates with the efficient aggregation protocol of Barthe et al. [1]. It can be easily derived that the share of the sum of two secrets is simply the sum of shares of each secret (i.e. $\langle v + w \rangle = \langle v \rangle + \langle w \rangle$). Similarly, the share of a secret multiplied with a public value is simply the product of the share of the secret multiplied by the same value, i.e. $\langle \alpha \cdot v \rangle = \alpha \cdot \langle v \rangle$. While linear functions can be computed by each authority using only local shares of secrets that are combined to reveal the result, as in the linear sum example above, the same does not hold for multiplications of secrets.

The multiplication of two secrets requires some interaction between the parties that hold shares, as well as pre-shared state [5]. First of all, we assume that all authorities have pre-shared, ahead of time and independently of the computation to be performed, a large number of triplets of shares $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$ where a and b are random and $c = a \cdot b$. We call those secrets "multiplication triplets"; §6 discusses options for deriving them in a smart metering setting. Using one of those triplets makes multiplication of two shared secrets possible through a single round of interaction: to multiply $\langle x \rangle$ and $\langle y \rangle$, all authorities first compute and reveal their shares of $\langle \epsilon \rangle = \langle x \rangle + \langle a \rangle$ and $\langle \delta \rangle = \langle y \rangle + \langle b \rangle$, enabling them to reconstruct ϵ and δ (which are statistically independent of x and y). The authorities then privately compute their shares of the product $\langle x \cdot y \rangle = \langle c \rangle - \epsilon \cdot \langle b \rangle - \delta \cdot \langle a \rangle + \epsilon \cdot \delta$. This enables us to evaluate general arithmetic circuits, consuming a pre-shared triplet for each multiplication, with as many rounds of interaction as the multiplicative depth of the circuit.

In particular, the availability of multiplication on shared secrets enables the computation of any boolean circuit. Assuming secrets ranging over $\{0, 1\}$ we can express NOT, AND, NAND, NOR and XOR gates directly as $\text{NOT}(a) = 1 - a$; $\text{AND}(a, b) = a \cdot b$; $\text{NAND}(a, b) = 1 - a \cdot b$; $\text{NOR}(a, b) = (1 - a)(1 - b)$ [22] and $\text{XOR}(a, b) = (a - b)^2$. Each of these gates uses (at most) one multiplication, together with linear operations; thus evaluating a boolean circuit on shared secrets requires as many triplets as gates, and requires as many rounds of interaction as the depth of the circuit.

Boolean circuits operate on secret bits, but readings are usually larger integers. It is trivial to convert a vector of bit shares to an integer share using linear operations (i.e., $\langle v \rangle = \sum_i 2^i \langle b_i \rangle$). The converse decomposition of an integer into its binary representation is a more complex operation, for which we use a variant of the protocol by Damgård et al. [4]. This protocol relies on pre-shared random bits $\langle b_i \rangle$. These, like the multiplication triplets, can be generated and shared before the computation takes place. Converting an n-bit shared number v to shares of its constituent n bits requires n pre-shared bits b_i. The protocol proceed as follows: all authorities compute and reveal their share of $\langle d \rangle = \langle v \rangle - \sum_i 2^i \langle b_i \rangle$ (which is statistically independent from v), then decompose d into bits u_i ($d = \sum_i 2^i u_i$). The authorities use these bits, together with the secret bits $\langle b_i \rangle$, as inputs to a boolean circuit performing addition (implemented using any logic gates available). This circuit outputs shared bits $\langle h_i \rangle$ such that $v = \sum_i 2^i h_i$. The choice of the addition circuit can have profound implications on the efficiency of this protocol; we use a variant of Ladner and Fischer's adder with parallel prefix computation [17] that minimizes the circuit depth, while increasing the fan out (which is not an issue for us).

Other general-purpose implementations of secret-sharing schemes and multi-party computation [2, 20] also embed integrity-protection mechanisms, and sometimes support faster comparisons. We are investigating how to integrate and adapt their features to the context of smart metering.

3. SYSTEM MODEL

Protocol participants & goals. Smart metering infrastructures involve the installation of next generation energy meters in user homes, to serve diverse needs. In our protocols, we assume that each *smart meter* is installed in a specific household (or *user*) and monitors a number of variables such as volume of energy consumed, or voltage across the supply line regularly (e.g. 2 times per hour) to generate *fine grained readings*. Optionally, we assume that users have access to some form of modern computing device connected to the Internet. This *user device* can be a computer, a smart phone, a set-top box, a game console, or a user controlled online server or service. Meter readings are periodically (e.g. once a day) collected and uploaded to a *storage service*. This service can be a traditional large scale distributed database that keeps track of readings and their meta-data.

Queries can be submitted to the storage service, requesting the computation of certain statistics on the stored data, by a number of entities in the energy industry (producers, suppliers, distributors, or the grid). The storage service is in charge of initiating and orchestrating all protocols required to answer a query. It is useful to split queries into two categories: *single-household* and *multi-household*. Single household queries are executed on data from a single household, for example to compute its time-of-use bill, or to profile a single household's consumption pattern into one of many categories. A multi-household query, such as the total volume of energy consumed in an area, or by the customers of a single supplier, uses data from many households.

Our protocols are concerned with privacy-friendly computations. Thus the storage service will need to enlist the help of a number of other parties to authorize and compute queries. We assume the availability of a small number of *authorities* that help answer queries. Those authorities are only interacting with the storage service and, given a query, release just enough information to allow the result to be computed. We assume that the authorities are connected to the storage service through a low-latency high-bandwidth network. In addition to the authorities, our protocols optionally uses one or multiple *randomness services* with specific properties—we discuss these in some depth later.

Security properties, assumptions & threat model. Our fundamental aim relates to privacy: to answer authorized queries without leaking any other information about stored readings. More precisely, we assume that meters are certified devices that act honestly in accordance to our protocols—this is an inherent limitation of any technology in this space.

If the meter acts outside the policy, or could be controlled by an adversary, they could bypass any protection for their readings. (This important security assumption may not hold in practice, inasmuch as current architectures allow meters to be remotely re-configured and reprogrammed, and their software security is rather poor [21].) We also assume that the meters are tamper evident through passive or active triggers, to ensure the correctness of the readings. We do not suggest obscuring in any way the reporting of tamper switches—we aim to protect only the privacy of readings of energy consumption and associated information. Finally, we assume that meters are capable of storing and securing long-term keys for signing readings as well as protecting their privacy.

The storage service is entrusted to reliably store readings, but should not be in a position to access them in clear. We consider a storage service that is honest but curious, namely it follows the protocol but attempts to draw inferences from its observations. Such a service should not learn anything besides the result of authorized queries. There are a number of ways of extending the protocols of this paper to fully malicious storage services, but we leave this to future work. Yet, to practically discourage malicious behaviour, we design the storage service protocols to be auditable: the service does not hold any secret keys, and thus any transcript of its interactions can be verified for compliance to the protocol by anyone. Such a transcript does not violate privacy.

Authorities are entrusted to be honest but curious, i.e., to follow the protocol and not deviate from it. Some colluding authorities can, however, pool their shares. As long as all authorities follow the protocol and at least one authority does not collude privacy is not violated. More specifically, all operations an authority performs are parameterized by secret keys, and these long term secrets must not be divulged.

The integrity and availability of the protocols also depends on the authorities executing the protocol faithfully. In this paper we assume the authorities to be reputable enough to not deviate from the protocol. We are exploring a number of avenues to relax this "honest-but-curious" assumption that permeates the threat model of this work.

Optionally, our protocols will make use of random data from user devices or randomness services. We assume those to keep the random data confidential. Still, this data does not depend on the readings or queries, so it can be generated and distributed ahead of the shared-secret computation.

4. PROTOCOLS

The protocol involves meters M_m, authorities A_a, a storage service S, and a randomness service R. For simplicity, we assume a single, fixed series of authorities for all readings.

Cryptographic set-up. The set-up ensures that, upon completion, every meter m shares a secret symmetric key $s_{m,a}$ with every authority a. Although any means can be used to establish such secrets for privacy-friendly computations (see e.g. [1]), we present a technique that minimizes key management for the authorities, allowing them to potentially handle a very large number of meters.

Upon initialization, each meter generates a key pair of a suitable public-key-encryption scheme. The public key is then certified, by the manufacturer or through a public-key infrastructure, as belonging to a meter with fixed, unique identifier m. Upon initialization, every authority a generates a master secret key s_a, that will be used to derive all other keys, and a key pair for a suitable signature scheme.

To *pair* a meter m with authorities , each authority a computes their shared secret $s_{m,a} = H_1(s_a|m)$, where H_1 is a key derivation function (modelled as a pseudo-random function or PRF), then it encrypts it under the public key of m and signs it using the public key of a. The message can be relayed through any channel; the meter will check the signature, ensure it corresponds to a valid authority (presumably through a root certificate from the manufacturer or certifier) and then decrypt and install $s_{m,a}$. As a result, each meter stores one symmetric key for each authority, whereas each authority only needs to store a single master key s_a. (In the following for simplicity we assume that the keys $s_{m,a}$ are communicated over a secure channel.)

Producing readings. Meters use the keys shared with authorities to 'blind' their readings, in a way that requires the cooperation of every authority to recover any information about them (as in [1]). We assume each reading $r_{m,\ell}$ from meter m is associated with a unique, public label ℓ (the sensor identifier and the time period, for example); the corresponding blinded reading is computed as:

$$c_{m,\ell} = r_{m,\ell} - \textstyle\sum_a H_2(s_{m,a}|\ell) \mod p \qquad (1)$$

where p is the native machine word size (e.g. 2^{32} for a 32 bit processor) and H_2 is a keyed hash function (also modelled as a PRF) that returns values in $0 \ldots p-1$. The meter, upon request, then transmits series of pairs $(\ell, c_{m,\ell})$ to the storage service, using whichever authenticated channel is available.

Importantly, the label ℓ must never be used for blinding any other reading. A safe implementation may for instance rely on a hardware counter included in ℓ.

By construction, the integers $\{c_{m,\ell}, H_2(s_{m,a}|\ell)$ for $a \in \vec{a}\}$ form shares of an additive secret-sharing scheme: their sum modulo p yields the secret reading $r_{m,\ell}$. Yet, as long as one share remains secret, the others do not leak any information about the secret reading. The shared keys have allowed us to reduce the cost of transporting a share to a single integer modulo p (while sacrificing perfect secrecy and reducing it to the security of two PRFs and the master keys).

Preparing random values. The randomness source uses the same approach to efficiently distribute and store multiplication triplets and random bits. Like meters, it follows the setup protocol and shares a symmetric key $s_{r,a}$ with each authority. To share a triplet $x, y, z = xy$ with labels ℓ_x, ℓ_y, ℓ_z, it

shares					
A_0	$H_2(s_{m,a_0}\|\ell)$	$H_2(s_{r,a_0}\|\ell_x)$	$H_2(s_{r,a_0}\|\ell_z)$	$H_2(s_{r,a_0}\|\ell_d)$	0
A_1	$H_2(s_{m,a_1}\|\ell)$	$H_2(s_{r,a_1}\|\ell_x)$	$H_2(s_{r,a_1}\|\ell_z)$	$H_2(s_{r,a_1}\|\ell_d)$	0
...					
S	$c_{m,\ell}$	0	$c_{\ell,z}$	$c_{\ell,d}$	v
$\sum$	$r_{m,\ell}$	x	xy	d	v

Table 1: Shares for private readings $r_{m,\ell}$, random values x, y, products z, bits d, and public values v.

sets $x = \sum_a H_2(s_{r,a}|\ell_x)$ and $y = \sum_a H_2(s_{r,a}|\ell_y)$, and computes the blinded value for z as: $c_{\ell,z} = xy - \sum_a H_2(s_{r,a}|\ell_z)$. Similarly, to share a random bit $d \in \{0, 1\}$ with label ℓ_d, it computes the blinded value $c_{\ell,d} = d - \sum_a H_2(s_{r,a}|\ell_d)$. As x and y are random secrets, the storage service does not require values for them and instead uses 0 as its share.

The whole distribution of random shares is summarized in Table 1. The last column lets the storage service S add public values only to its local share. We note that the cost of transporting a reading, a random bit, or a whole triplet is merely a single element in p. We are now ready to apply all the techniques from §2 to compute on these secret shares.

Computations. Upon receiving a query, the storage service communicates it to all authorities. A query consists of instructions on secret shares with specific labels—these instructions are executed on the stored shares by the storage service, and on the derived shares by each authority. The instruction set consists merely of linear weighted sums of shares, revelation of shares, and loading of pre-computed values. Revelation is done through the storage service, which gathers all shares revealed at a specific stage, adds them and broadcasts the result in clear to all authorities. Those few instructions allow us to execute interesting high level computations, using previously discussed techniques, in particular:

- Weighted sums of shared secrets and public values.
- Multiplications of shared secrets, at the cost of a shared triplet and a round of revelation.
- All boolean operations, including OR, AND, NAND, XOR, at the cost of a single multiplication each.
- Conversions between shares of integers mod p and vectors of shares of their binary representation, which requires pre-shared bits and triplets.

This set of operations allows us to express any boolean circuit and therefore support any fixed-depth computation. The cost of computing on shares is in terms of the number of rounds of revelation of shares (latency), the number of shares to be revealed (bandwidth), and the number of pre-shared values needed to facilitate operations. Careful arithmetic and circuit optimizations are applied to minimize those.

Main results. We outline the main properties of our protocol, but refer to prior work for detailed proofs.

Correctness: the protocol returns the same results as those returned by the source query on the readings.

Privacy: provided at least one authority is securely implemented, the protocol leaks no other information on the readings. More precisely, consider the following game

1. After set-up, given the private states for some meters and for all-but-one authorities, the adversary chooses two collections of readings that (1) coincide on corrupted meters; and (2) yield the same query result;

2. Given the trace of the protocol for one of these two collections, the adversary wins if it guesses which of the two is used. The advantage of the adversary reduces to those of H_1 and H_2 being PRFs.

5. APPLICATIONS & COST

We illustrate the feasibility and practicality of using computations on secret shares on the four sample applications discussed in §1. To this end we have implemented the key distribution and secret-sharing-based computation engine in 1,600 lines of Python. Table 2 summarizes the performance for each of the protocols, detailed next. The number of *round trips* of revelations of intermediate shares measures the impact of network latency between authorities and the storage service (this latency can be shared between many runs of the same protocol on different readings); the number of *revealed shares* indicates the volume of data transferred between authorities and the storage service (4 bytes per share); finally, the numbers of *pre-shared triplets* and *pre-shared bits* gives the intensity of use of the randomness service.

```
def meanvar(c, readings):
    rsum = c.linear([1] * 100, readings)
    rsquares = [c.mult(x, x) for x in readings]
    rsumsquares = c.linear([1] * 100, rsquares)
    return [rsum, rsumsquares]
```

The first protocol meanvar computes and reveals all necessary values to extract the mean and the variance or standard deviation of a set of 100 readings. It consists of a linear sum of all readings, then 100 squarings of each of the readings, followed by the sum of all squares. From these the variance can be publicly computed as $\mathrm{Var}[X] = E[X^2] - (E[X])^2$. Since the squarings do not depend on one another, a shallow multiplication circuit of depth one minimizes latency. Hence, only 200 shares need to be exchanged, in a single 800-byte message from each A. The same number of pre-shared triplets as multiplications are needed, and each only squares the reading from a single meter.

```
def compare(c, rA, rB, Thld):
    diff = c.linear([1, -1], [rA, rB], -Thld)
    bits, _ = c.tobits(diff)
    return c.gneg(bits[-16])
```

The second protocol compare implements a simple theft prevention mechanism. It accepts readings of the voltage across meters of two consecutive houses on the same distribution line, and checks whether the voltage drop is above a threshold. If the drop is too high, it might indicate that someone is absconding electricity from the line between the two homes. The comparison is performed by subtracting the second reading and the threshold from the first one, and comparing the result with zero. Currently, this is done by decomposing the difference into bits, and testing the high-end bit to determine the sign of the number (assuming that the difference fits within 16 bits). We are aware of more efficient comparison protocols [3]. The resulting protocol takes 11 rounds of communication within which 117 shares are exchanged by each authority and the storage service. The bit decomposition protocol uses 32 pre-shared bits as expected.

```
def compare3way(c, rA, rB, rC):
    diff1 = c.linear([1, -1], [rA, rB])
    diff2 = c.linear([1, -1], [rB, rC])
    bits1, _ = c.tobits(diff1)
    bits2, _ = c.tobits(diff2)
    return c.gxor(bits1[-16], bits2[-16])
```

Sample query	meanvar	compare	compare3way	demrespbill
Round-trips	1	11	12	12
Revealed shares	200	117	236	8,700
Pre-shared triplets	100	58	117	4,300
Pre-shared bits	0	32	64	3,200

Table 2: Performance evaluation

Another theft prevention protocol compare3way is designed to detect abnormal sequences of voltages cross consecutive households. It takes readings from three consecutive homes on the same line and ensures they are ordered, either in increasing or decreasing order. According to this use case, described in [8], if the middle household is shorting the meter, its reading will be lower than the two readings on either side. The protocol compares the difference between the adjacent pairs of households to zero, and requires their sign to be the same (through an XOR illustrating boolean gates). The two comparisons are run in parallel, and the XOR gate requires one round of interaction, resulting in 12 rounds of communication, and 236 shares being exchanged.

```
def greaterthan(c, readingA, cutoff):
    diff = c.linear([1], [readingA], -cutoff)
    bits, _ = c.tobits(diff)
    return c.gneg(bits[-16])
def demrespbill(c, readings, cutoff, penalty):
    free, cut = penalty * cutoff, []
    for r in readings:
        gt = greaterthan(c, r, cutoff)
        excess = c.linear([penalty], [r], -free)
        cut += [c.mult(excess, gt)]
    return c.linear([1] * 100, cut)
```

Finally, demrespbill implements a time-of-use billing mechanism to support demand-response. It takes readings from a single household during 100 overloaded periods for which a "back-off" signal was sent. Each reading is compared with a cut-off consumption and any units of over consumption are billed at a punitive tariff. The sum of all bills over all periods is returned, to further hide when over-consumption occurred. This is achieved by building a binary vector indicating whether there is over-consumption in each period, and multiplying it with the amount to be paid. In case of no over consumption this yields zero, otherwise the amount to be billed. The computation is repeated for 100 periods, and the results summed. All comparisons are run in parallel leading to only 12 rounds of computation. In total 8700 shares are exchanged, and there is a need for 4300 pre-shared triplets and 3200 pre-shared bits. The pre-shared information is used only to blind readings from a single household.

6. SOURCES OF RANDOMNESS

The availability of pre-shared "multiplication triplets" and random bits makes the multi-party protocols efficient and straightforward. Yet, the efficient provision of such values is a key open problem. A corrupt source of randomness that reveals values to the adversary can seriously harm privacy (and less so integrity, since their correctness can be checked at some cost). Here we somewhat depart from the orthodoxy that uses secret-sharing-based multiplication protocols and somewhat homomorphic encryption schemes to compute those values, and consider other available options:

For single household computations, such as the demand-response billing protocol, any user device (laptop, mobile phone, game console, or even the meter itself) can provide ahead of time a large number of random values for use in subsequent computations on the household data. Thus we can very cheaply acquire a supply of randomness to cover all use-cases where in past proposals computation was offloaded to a user device [23]. This option is also applicable to the computation of variance, since the squaring of each meter reading involves only data from a single household.

A second option is to rely on either one or multiple secure hardware solutions to provide the random data. This can be done ahead of time and efficiently. All opportunities, but also worries associated with hardware secure modules would apply to this option.

Finally, we are exploring a collaborative approach to generating random data, whereas a number of users (or maybe their meters) collaborate to create a pool of randomness to be used across diverse applications. The probability of attack is minimized by making it unlikely that a single adversary can learn a significant or targeted amount of information. We are also working on ways of combining random sources to create a single high-quality random source.

7. CONCLUSIONS

We have demonstrated that complex privacy-friendly computations on smart meter readings are practical, resolving a number of open challenges in billing, statistics and fraud detection. Furthermore, the cost on the meters and the storage required are minimal, and similar to the cost of producing and storing plain unprotected readings. Evaluating linear functions is simple. Thanks to offline sources of correlated randomness, evaluating more complex functions is comparable in terms of computation, but incurs communications bandwidth and latency overheads proportional to the number and depth of multiplications in the circuit, respectively.

We argued that, for smart metering, user devices can produce cheaply random values needed to fuel some protocols that compute on a single household. This pattern of private outsourced computation is likely to apply to other settings. Yet, how to acquire secure random values efficiently in general remains an open problem. Similarly, extending the protocols to the setting where the storage service or authorities are actively dishonest is left for future work. For both, recent progress in implementation of general-purpose multi-party computation offers valuable insights and reasons for optimism. Given sufficient business interest, there is evidence of a Moore's law for privacy-friendly secure computations.

References

[1] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella-Béguelin. Verified computational differential privacy with applications to smart metering. In *26th IEEE Computer Security Foundations Symposium, CSF 2013*. IEEE, 2013.

[2] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security – ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

[3] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks, SCN 2010*, volume 6280 of *LNCS*, pages 182–199. Springer, 2010.

[4] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *3rd Theory of Cryptography Conference, TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, 2006.

[5] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority — or: Breaking the SPDZ limits. *IACR Cryptology ePrint Archive*, 2012:642, 2012.

[6] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[7] G. Danezis and B. Livshits. Towards ensuring client-side computational integrity. In *3rd ACM Cloud Computing Security Workshop, CCSW 2011*, pages 125–130. ACM, 2011.

[8] B. Defend and K. Kursawe. Implementation of privacy-friendly aggregation for the smart grid. In *HotPETS*, 2013.

[9] Future of privacy summary of California public utilities commission proposed decision on smart grid privacy and security. On-line `http://www.futureofprivacy.org/issues/smart-grid/`, May 9, 2011.

[10] W. Heck. Smart energy meter will not be compulsory. NRC Handelsblad, April 2009.

[11] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *12th Intl. Symp. on Privacy Enhancing Technologies, PETS 2012*, volume 7384 of *LNCS*, pages 221–238. Springer, 2012.

[12] M. Jawurek, M. Johns, and F. Kerschbaum. Plug-in privacy for smart metering billing. In *11th Intl. Symp. on Privacy Enhancing Technologies, PETS 2011*, volume 6794 of *LNCS*, pages 192–210. Springer, 2011.

[13] M. Jawurek, F. Kerschbaum, and G. Danezis. Privacy technologies for smart grids - a survey of options. Online `http://research.microsoft.com/apps/pubs/?id=178055`, 2012.

[15] R. Knyrim and G. Trieb. Smart metering under EU data protection law. *Intl. Data Privacy Law*, 1(2):121–128, 2011.

[16] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *11th Intl. Symp. on Privacy Enhancing Technologies, PETS 2011*, volume 6794 of *LNCS*, pages 175–191. Springer, 2011.

[17] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, 1980.

[18] Q. Li and G. Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In *13th Intl. Symp. on Privacy Enhancing Technologies, PETS 2013*, volume 7981 of *LNCS*, pages 60–81. Springer, 2013.

[20] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[21] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *Security & Privacy, IEEE*, 7(3): 75–77, 2009.

[22] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symp. on Security and Privacy*, pages 238–252. IEEE, 2013.

[23] A. Rial and G. Danezis. Privacy-preserving smart metering. In *10th annual ACM workshop on Privacy in the electronic society, WPES 2011*, pages 49–60. ACM, 2011.

[24] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *2011 Network and Distributed System Security Symposium, NDSS 2011*. The Internet Society, 2011.

[25] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography – PKC 2010*, volume 6056 of *LNCS*, pages 420–443. Springer, 2010.