Fault-Tolerant Privacy-Preserving Statistics

Marek Jawurek and Florian Kerschbaum

SAP Research
Karlsruhe, Germany
{marek.jawurek|florian.kerschbaum}@sap.com

Abstract. Real-time statistics on smart meter consumption data must preserve consumer privacy and tolerate smart meter failures. Existing protocols for this *private distributed aggregation model* suffer from various drawbacks that disqualify them for application in the smart energy grid. Either they are not fault-tolerant or if they are, then they require bi-directional communication or their accuracy decreases with an increasing number of failures. In this paper, we provide a protocol that fixes these problems and furthermore, supports a wider range of exchangeable statistical functions and requires no group key management. A key-managing authority ensures the secure evaluation of authorized functions on fresh data items using logical time and a custom zero-knowledge proof providing differential privacy for an unbounded number of statistics calculations. Our privacy-preserving protocol provides all the properties that make it suitable for use in the smart energy grid.

Keywords: Privacy, Smart Grid, Statistics, Aggregation, Stream, Fault-Tolerance

1 Introduction

In the smart energy grid there is a conflict between privacy of consumers and utility for service providers. Utility providers can use real-time household electricity consumption data for forecasting future consumption. This consumption forecasting allows them more efficient and more stable operation of the electricity grid. However, real-time consumption data also closely reflects any activity in the household involving electrical appliances. Thus, for the consumer, it represents a privacy invasion [10,11]. Previous studies [8,9,13,15,16,17,19,22] have shown that and how information about a household and its inhabitants can be inferred from its high-resolution energy consumption data. Furthermore, any viable solution for forecasting consumption must also anticipate failing smart meters or communication links. A single failure must not prevent the real-time calculation of statistics.

The ability to calculate statistics in real-time, i.e., in the presence of failures, can also benefit many other real-world applications like public health and clinical research on patient information or any collection and monitoring where privacy-sensitive data is processed.

In this paper we provide a protocol in the *fault-tolerant, private distributed aggregation model*: Many data producers (e.g. smart meters) constantly produce sensitive data items (e.g. hourly smart meter consumption measurements). An untrusted data consumer (e.g. service provider) calculates statistics, e.g., for forecasting future consumption, over subsets of these data items in the presence of failing data producers. Not even a collusion of malicious data producers and consumer may lead to the disclosure of sensitive data items.

Our protocol roughly works as follows: We use homomorphic encryption for aggregation and employ an (w.r.t. privacy) untrusted, possibly distributed key-managing authority that provides differentially private decryption services to the data consumer while neither learning data items nor statistics results.

Recently, [2] also presented a protocol for this model. Without fault-tolerance it has been considered in [23,25]. In comparison to these existing protocols our contributions can be summarized as follows:

- The accuracy of the calculated statistics is higher. In our protocol the accuracy is independent of the number of data producers, the number of data items and the number of failures.
- We do not require synchronized clocks, but only rely on logical time.
- Our protocol enables the calculation of a wider range of statistical functions (weighted sums). The statistical function can be chosen and exchanged intermittently by the data consumer without notification to the data producers.
- We do not require any group key management. Data producers may join or leave without interaction with other participants.
- We only require uni-directional communication channels between data producers and data consumers. This implies a reduced attack surface of the smart meter.

The remainder of this paper is structured as follows: Section 2 describes the contributions of our protocol in comparison to [2,23,25]. Section 3 introduces the prerequisites used in our protocol. In Section 4 we introduce a naive version of our protocol to achieve differentially private, fault-tolerant statistics in the semi-honest model. Then, in Section 5 we present the final protocol and our custom zero-knowledge proof for the malicious model. Finally, we present related work (Section 6) and conclude with a summary in Section 7.

2 Contributions

We compare our protocol to the protocols in [2,23,25]. We favorably compare in existing criteria, but also extend their set of criteria. Our extended Table 1 illustrates the differences.

2.1 Communication

We assume that many (unsynchronized) data producers constantly produce sensitive data items. They send these items to a data consumer over a uni-directional

Scheme	Avg comm. per user	Comm. model	Error	Fault-tolerant	Group key management required	Synchr. clocks	Security model
Naive DP	$O(1)$	$C \rightarrow S$	$O(\sqrt{n})$	Yes	No	No	DP
[23]	$O(1)$	$C \Leftrightarrow S$	$O(1)$	No	Yes	Yes	CDP AO
[25]	$O(1)$	$C \rightarrow S$	$O(1)$	No	Yes	Yes	CDP AO
[2] Sampling	$O(\frac{1}{\phi^2 * n})$	$C \Leftrightarrow S$	$O(\phi * n)$	Yes	Yes	Yes	DP
[2] Binary	$O(\log n)$	$C \rightarrow S$	$\tilde{O}((\log n)^{\frac{3}{2}})$	Yes	Yes	Yes	CDP
this paper	$O(1)$	$C \rightarrow S$	$O(1)$	Yes	No	No	DP AO

Table 1. DP: differential privacy CDP: Computational differential privacy AO: Aggregator Obliviousness $C \rightarrow S$: client-to-server uni-directional $C \Leftrightarrow S$: interactive between client and server

communication channel. Every data producer has constant communication cost per data item. The data consumer queries the key-managing authority for decryption over a bi-directional channel. Thus, the communication cost for the key-managing authority is linear in the number of calculated statistics.

[23] requires bi-directional communication between data producers (users) and the data consumer (aggregator) so that the users can cooperate in decrypting the threshold decryption system. [25] only requires uni-directional communication links between data producers and the consumer and has the same communication cost as our scheme. In [2] data producers also communicate uni-directionally with data consumers but have higher communication cost. This is due to redundant information provided by data producers for fault-tolerance.

2.2 Accuracy

In our protocol the accuracy of the data consumer's calculated statistics is independent of the number of data producers or data producer failures. The only error in accuracy is introduced deliberately to ensure differential privacy and that is $O(1)$ with respect to the number of data producers or failures.

[25] and [23] also introduce $O(1)$ error for differential privacy while [2] introduces polylogarithmic error dependent on the number of data producers.

2.3 Fault-Tolerance

Fault-tolerance in our protocol is introduced by a selection process of the data consumer. The data consumer can arbitrarily select a subset of available data items as input to the statistics calculation. Consequently, our scheme tolerates an arbitrary and unbounded number of failing data producers. The key-managing authority can also be distributed (as we describe in Section 4.3) and we only require a majority of key-managing authority instances to be available during a run of the protocol.

[25] and [23] do not tolerate any failures of data producers (users). The former relies on blinding shares of zero which requires all shares to be present. The latter requires the data producers to participate in a threshold decryption of the final result. This offers some fault-tolerance, but the accuracy of the result degrades with failures. [2] offers fault-tolerance, but the error of the results grows sublinearly in the number of absent users. A bound on the maximum number of failing users needs to be pre-arranged.

2.4 Group Key Management

In our protocol, there is no group or elaborate key management. Data producers may join or leave independently at any time without any new key distribution or setup phase.

[25] requires a new distribution of the blinding shares for all data producers whenever a new producer joins. [23] requires a new distribution of key shares for the threshold decryption. [2] introduce a dynamic join & leave protocol which tolerates joining of data producers up to a certain limit. Beyond, they need to increase the tree height to accommodate more data producers. In such a case, the trusted dealer needs to distribute one additional secret key to existing users and $O(\log n)$ secret keys to new users.

2.5 Synchronization

Our protocol does not require synchronized clocks. We use logical time in order to chronologically order and thus prevent re-use of data items. All data items are encrypted under the same key.

All protocols [2,23,25] depend on communication rounds and require synchronization, although they do not explicitly mention this.

2.6 Security

Our scheme offers aggregator obliviousness (AO), i.e., the aggregator will not learn anything else but the final result. Input data and intermediate results are not available to him. Furthermore, we ensure differential privacy (DP) [4] for statistics.

Although [23,25] provide aggregator obliviousness they only guarantee the weaker notion of computationally differential privacy (CDP). In these protocols the differential privacy has to be ensured by the data producers while in our scheme the key-managing authority ensures this. [2] offers protocols with differential privacy and computational differential privacy, but does not ensure aggregator obliviousness. The data consumer (aggregator) learns some intermediate values (even if there is no fault) in order to calculate the result in the presence of failures.

3 Prerequisites

3.1 Differential Privacy

The definition of differential privacy according to [4] is the following:

Definition 1 *A randomized function K gives ϵ-differential privacy if, for all data sets D_1 and D_2 differing on at most one element and all $S \subset Range(K)$,*

$$Pr[K(D_1) \in S] \leq \exp(\epsilon) \cdot Pr[K(D_2) \in S]$$

It means, that the probability for any result of K changes only slightly (less than $\exp(\epsilon)$ if single elements are included/excluded in the set of inputs to K. In consequence, even with knowledge of the data set, the function result and arbitrary auxiliary information, it is hard for an attacker to identify whether an element is present or not. Thus, also the actual value of the element is protected.

Applied to smart metering that means, that if every statistics function on smart meter consumption data is differentially private, the individual readings can not be recovered by the receiver of the statistics's results, e.g. the service provider. Thus, differentially private statistics functions protect consumer privacy.

We will transform any function f into an ϵ-differential private version of itself by adding random noise according to a symmetric geometric distribution [6]. Specifically, if δ is the f's sensitivity, we add a sample from distribution $Geom(exp(\frac{\epsilon}{\delta}))$ to each function result $f(x)$ to make it differentially private. The parameter ϵ must be chosen according to the use case at hand. It represents the desired trade off between accuracy of the function K and how well it preserves privacy.

3.2 Paillier Cryptosystem

We only give a basic introduction to the Paillier cryptosystem, further information – including security proofs – can be found in [20]. This cryptosystem defines two functions:

– $E(m, r) \to c$, encrypts a message $m \in \mathbb{Z}_n$ with random value $r \in \mathbb{Z}_n^*$ to the ciphertext $c \in \mathbb{Z}_{n^2}^*$. All encryptions in our protocol are performed by data producers and use the key-managing authority's public key.
– $D(c) \to (D_v(c), D_r(c)) \to (m, r)$, decrypts ciphertext c to a tuple (m,r). In our protocol, all decryptions are performed by the key-managing authority using its private key.

We use this cryptosystem's ability to also recover the random parameter r during decryption.

The Paillier cryptosystem also has the following homomorphic properties:

$$D(E(m_1, r_1)E(m_2, r_2) \mod n^2) = (m_1 + m_2, r_1 \cdot r_2) \mod n \tag{1}$$

$$D(E(m, r)^k \mod n^2) = (km, r^k) \mod n \tag{2}$$

In the following, whenever we refer to the encryption E or the decryption function D applied to either singular values or vectors of ciphertexts it yields what makes most sense in the respective context: Singular values are encrypted (decrypted) to singular values and a vector of values is encrypted (decrypted) to a vector of values. The function $\boldsymbol{X} \cup v$ appends the scalar value v to the vector $\boldsymbol{X}$. Also, for improved readability, we write $f \circ h(X)$ for the sequential composition of functions f and h.

4 Protocol Description

In this Section we first give a description (Section 4.1) of a naive version of our fault-tolerant, differentially private statistics protocol. It provides differential privacy for a semi-honest data consumer, but may fail in case of a malicious data consumer. Then, in Section 4.2 we analyze this deficiency and explain the necessary restrictions that we must employ, namely the freshness of used data items and correctness of computation.

Finally, in the next Section 5, we present the full protocol with a focus on our custom zero-knowledge proof that provides fault-tolerant differential privacy in the presence of malicious data consumers with a distributed key-managing authority.

4.1 Naive Protocol

We restrict ourselves, without loss of generality, to one round of communication from data producers to the data consumer and one subsequent function evaluation and decryption. In the general case, the protocol step *Preparation* i.e., the creation of data items, is executed repeatedly in parallel and unsynchronized to the protocol steps that implement calculation and the decryption (*Calculation* and *Decryption*). Therefore, in the general case, in protocol step *Calculation* the data consumer could choose among all input values that he has received during the entire system run time, i.e. multiple rounds of data item creation. Furthermore, also without loss of generality, we assume, that data consumer and key-managing authority have pre-arranged a function f that the data consumer wishes to evaluate. We also assume, that in a setup phase all data producers obtain the public key that corresponds to the private key only held by the key-managing authority.

Preparation: Every data producer j encrypts its value v_j with a random number r_j. Every data producer sends $E(v_j, r_j)$ to the data consumer.

Calculation: The data consumer now chooses a vector

$$\boldsymbol{V} = (E(v_1, r_1), \ldots, E(v_m, r_m))$$

out of all encrypted data items that it received and *has never used before.* The data consumer calculates any statistics function of the form

$$f((x_1, \ldots, x_m), c) = (\sum_{i=1}^{m} a_i \cdot x_i) + c$$

by evaluating f's homomorphic counterpart

$$f_h(\boldsymbol{V}, E(c, r_c = 1)) = (\prod_{i=1} E(v_i, r_i)^{a_i}) \cdot E(c, r_c)$$

f_h is derived from f using the homomorphic relationship (see Equations 1, 2), the a_i are constants and c is a data consumer-chosen variable.
Then, the data consumer sends $f_h(\boldsymbol{V}, E(c, 1))$ to the key-managing authority for decryption to receive the plaintext of the statistics result.

Decryption: The key-managing authority decrypts $f_h(\boldsymbol{V}, E(c, 1))$ which yields (according to Equations 1,2):

$$D_v \circ f_h(\boldsymbol{V}, E(c, 1)) = f \circ D_v(\boldsymbol{V}, E(c, 1))$$

Note that the data consumer can use its addend c to prevent the disclosure of the statistics result to the key-managing authority.
Then, it applies the function $m_f(s, x)$ which adds a random sample according to the sensitivity of f (see Section 3.1) and a seed s to the decrypted function result in order to make the statistics function evaluation differentially private. Finally, it returns the differentially private statistics result's plaintext: $m_f(s) \circ D_v \circ f_h(\boldsymbol{V}, E(c, 1))$

4.2 Malicious Behavior

The naive protocol version described in Section 4.1 will provide differential privacy in the semi-honest model [7]. We assume that the data consumer abides by the protocol and computes the function as agreed with the key-managing authority. This includes that he only chooses data items that he has never used before. However, this freshness property of the data items is not ensured by the protocol. Thus, in the presence of malicious data consumers, the naive protocol cannot protect data items from re-use by the data consumer.

The re-use of data items has severe implications for differential privacy of continuous statistics calculation: It is easy (see Section 3.1) to determine the necessary parameter for the distribution of the random noise that makes a single function evaluation ϵ-differentially private. In [18] McSherry has analyzed how the differential privacy of combined function evaluations (queries) over intersecting data sets relates to the differential privacy of the individual functions: Let a function f_i provide ϵ_i-differential privacy. If all functions f_i cover disjoint subsets, the differential privacy for the combined function is $max_i \epsilon_i$. However, if several functions span the same set of values, the differential privacy for the combination of those functions is $\sum_i \epsilon_i$.

Thus, if we do not prevent the re-use of data items, the differential privacy will add up. Eventually, it will be insufficient for the protection of data items. In the PINQ framework [18] the aggregate knowledge gain of the attacker, i.e. the sum of differential privacy over all combined functions, is limited. This is achieved by giving every user a specific, upfront budget of differential privacy.

Once this budget is used up, no more statistics calculations (and decryptions) are allowed.

This approach is incompatible with continuous calculation of statistics. In the smart meter statistics context but also for other applications the number of statistics calculations must be unbounded. Thus, in order to limit the aggregate knowledge gain of an attacker we have to prohibit the re-use of data items and thus make it equivalent to calculations over disjoint subsets.

In order to turn the naive protocol into a secure protocol in the malicious model, the data consumer must prove to the key-managing authority that it used fresh data items and that it correctly evaluated a known statistics function. The evaluated statistics function f must be part of the data consumer's proof, because f's sensitivity determines the variance of the random noise in differential privacy (see Section 3.1).

However, Goldreich's compiler approach [7] to turn the naive protocol secure in the malicious model would not be feasible: The resulting generic zero-knowledge proof (ZKP) would be prohibitively costly spanning the entire history of data items.

Therefore, we subdivide the proof in two parts: The first part covers the fault-tolerance of the calculated statistics, i.e., the choice of k fresh data items out of n total data items. We model this choice with the function $\mathrm{fresh}_k(n)$. Section 4.3 describes how a key-managing authority can guarantee this freshness by keeping state. Furthermore, in Section 4.4, we implement a distributed key-managing authority to make it resilient to failures.

The second part of the proof covers the the evaluated statistics function f. In Section 5 we present the final protocol that employs a custom zero-knowledge proof that covers the evaluated statistics function and completes the freshness guarantee.

Thus, in total, the data consumer proves to the key-managing authority with both parts that it honestly evaluated $f \circ \mathrm{fresh}_k(n)$.

4.3 Freshness of Data Items

As identified in Section 4.2 the key-managing authority must only allow decryption of function results that incorporate fresh data items.

This can only be accomplished by keeping state with the key-managing authority. The main challenges with keeping state are:

1. Keep the state as small as possible and not dependent on the number of decryption requests or the frequency of data item creation.
2. Allow for a distributed key-managing authority which makes the protocol resilient to failure of the key-managing authority.

In order to keep the state at the key-managing authority manageable we further restrict the freshness property: Data items originating from the same data producer can only be used in chronological order. Items may be skipped however. The function $P_j(v)$ at data producer j returns the position (logical time) of data item v within the total order of data items of data producer j.

The key-managing authority remembers for every data producer which data item was used last in any statistics calculation. At the key-managing authority, the function $C(j)$ returns for data producer j the position (logical time) of the latest data item that was used in a statistics calculation in j's total order of data items.

Preparation: In addition to the naive protocol, every data producer j also sends $P_j(v_j)$ to the data consumer.

Decryption request: Let p_i denote the data producer, then the data consumer compiles data item information $\boldsymbol{I}$ about the input values $\boldsymbol{V}$ (see the naive protocol) he used for the current statistics calculation:

$$\boldsymbol{I} = ((p_0, P_0(v_0), \ldots, (p_m, P_m(v_m)))$$

We provide integrity protection for $\boldsymbol{I}$ using signatures in Section 5.

The data consumer sends a decryption request with the encrypted statistics result $f_h(\boldsymbol{V}, E(c, 1))$ and $\boldsymbol{I}$ to the key-managing authority.

Validation of freshness: The key-managing authority verifies the freshness of every used data item: For every $(p_j, P_j(v_j)) \in \boldsymbol{I}$ it checks whether $C(p_j) < P_j(v_j)$. If successful, $C()$ is updated so that: $C(p_j) \leftarrow P_j(v_j)$ and the key-managing authority proceeds with the decryption and the addition of random noise as in the naive protocol.

4.4 Distributed Key-Managing Authorities

It may be desirable to distribute the key-managing authority to make it resilient against failure.

However, a key-managing authority like described in the previous Section 4.3 cannot be simply split into several instances because of its state $C()$. A malicious data consumer could send decryption requests over intersecting subsets of data items to different instances of the key-managing authority. These instances would be unable to guarantee freshness of data items which would break our measures for ensuring differential privacy (as explained in Section 4.2).

Communication between different instances, in order to synchronize state, would also be prohibitively costly: Upon every decryption request an instance would have to query all other instances for the current state of their function C. We also would have to cover merging of different functions C and how to cope with failed instances.

Therefore, we propose the following augmented protocol for a distributed key-managing authority:

The total number of key-managing authority instances is $n = 2t + 1$ and we assume that at least a majority of $t + 1$ instances are alive at any time. Every instance holds its own version of function $C()$.

Decryption request: The data consumer prepares the same decryption request for all key-managing authority instances. As in the non-distributed protocol, it sends the encrypted statistics result $f_h(\boldsymbol{V}, E(c,1))$ and the information about the used input values $\boldsymbol{I}$ to, unlike in the non-distributed protocol, all key-managing authority instances.

Validation of freshness: As in the non-distributed case, every instance verifies the freshness of used data items: For every $(p_j, P_j(v_j)) \in \boldsymbol{I}$ it checks whether $C(p_j) < P_j(v_j)$. If successful, $C()$ is updated: $C(p_j) \leftarrow P_j(v_j)$.
The k-th instance decrypts $f_h(\boldsymbol{V}, E(c,1))$ and applies the function $m_f(s,x)$ to make it differentially private. We describe at the end of this Section how all instances can ensure they apply the same random noise. This yields

$$x = m_f(s) \circ D_v \circ f_h(\boldsymbol{V}, E(c,1))$$

The k-th instance now creates a share s_k of secret x using a random polynomial in Shamir's secret sharing scheme. In order to ensure that all instances choose the same random polynomial, we apply the same mechanism as for the noise (see below). This resulting secret share s_k is finally returned to the data consumer.

Assembly: The data consumer collects the secret shares from at least $t+1$ key-managing authority instances and reassembles the differentially private statistics result's plaintext.

The idea of this protocol is that the data consumer has to contact at least $t+1$ instances of the key-managing authority to obtain enough shares for its statistics. Every instance in this majority will then also update their state C. This means, that subsequently, a malicious data consumer will fail to find a disjoint majority set of instances and thus cannot cannot create partitions of key-managing authority instances with different states.

Note that all key-managing instances operate completely independent of each other. On the one hand, this eliminates any synchronization problems. On the other hand, every instance must create valid shares of the same x in order to allow successful assembly of x at the data consumer. We propose a mechanism that chooses the randomness for the random noise and for the secret sharing polynomial based on a pseudo-random function with identical seed across all instances. We then use the commonly available information $\boldsymbol{I}$ as seed.

5 Zero Knowledge Proof

5.1 Properties

Already in the naive protocol version we shown how a key-managing authority can decrypt a homomorphically encrypted function result and add random noise in order to guarantee ϵ-differential privacy in the presence of a semi-honest data consumer. However, this relies on the key-managing authority's knowledge about the sensitivity of the evaluated function (see Section 3.1) and on the freshness of data items (Section 4.2). In Section 4.3 we describe how a key-managing authority can guarantee freshness with manageable state and how such a key-managing authority can be distributed (Section 4.4).

In order to complete the final protocol that ensures differential privacy over an unbounded number of statistics calculations in case of a malicious data consumer we require guarantees for the following properties:

1. The key-managing authority can verify the correctness of the provided data item information $\boldsymbol{I}$ in order to guarantee freshness.
2. The key-managing authority can verify the correct evaluation of a known statistical function on data items provided by data producers. This allows the key-managing authority an appropriate choice of random noise for making the statistics result differentially private.

In the following (Section 5.2) we describe the final fault-tolerant, privacy-preserving statistics protocol between the data consumer and the distributed key-managing authority that provides guarantees for these properties.

5.2 The Final Protocol

Like in the naive protocol version we assume, without loss of generality, a pre-arranged statistics function f and a pre-distributed key-managing authority's public key. The function $P_j(v)$ at data producer j returns the position (logical time) of data item v within the total order of data items of data producer j. At the key-managing authority (at every instance), the function $C(j)$ returns for data producer j the position (logical time) of the latest data item that was used in a statistics calculation in j's total order of data items.

Preparation: Every data producer $j \in 1, \ldots, n$ encrypts its data item v_j with a random number r_j. Furthermore, the data producers encrypt the r_j with another chosen random value r'_j and create signatures over the random's ciphertext and the data item identification tuple $(j, P_j(v_j))$: $S_j(E(r_j, r'_j), (j, P_j(v_j)))_j$. Every data producer sends $(j, P_j(v_j))$, $E(v_j, r_j)$, $E(r_j, r'_j)$ and $S_j(E(r_j, r'_j), (j, P_j(v_j)))_j)$ to the data consumer.

Calculation: The data consumer now chooses a vector

$$\boldsymbol{V} = (E(v_1, r_1), \ldots, E(v_m, r_m))$$

out of all encrypted data items that it received and corresponding vectors of encrypted randoms $\boldsymbol{R}$, of data item information $\boldsymbol{I}$ and signatures $\boldsymbol{S}$:

$$\begin{aligned}
\boldsymbol{V} =& (V_1, \ldots, V_m) | V_i \in \{E(v_1, r_1), \ldots, E(v_n, r_n)\} \\
\boldsymbol{R} =& (R_1, \ldots, R_m) | R_i \in \{E(r_1, r'_1), \ldots, E(r_n, r'_n)\} \\
\boldsymbol{I} =& (I_1, \ldots, I_m) | I_i \in \{(1, P_1(v_1)), \ldots, (n, P_n(v_n))\} \\
\boldsymbol{S} =& (S_1, \ldots, S_m) | \\
& S_i \in \{S(E(r_1, r'_1), (1, P_1(v_1))_1, \ldots, S(E(r_n, r'_n), (n, P_n(v_n)))_n\}
\end{aligned}$$

Then, exactly like in the naive protocol version, it calculates the pre-arranged function f of the form

$$f((x_1, \ldots, x_m), c) = (\sum_{i=1}^{m} a_i \cdot x_i) + c$$

by evaluating f's homomorphic counterpart

$$f_h(\boldsymbol{V}, E(c, r_c = 1)) = (\prod_{i=1}^{m} E(v_i, r_i)^{a_i}) \cdot E(c, r_c)$$

f_h is derived from f using Equations 1 and 2, the a_i are constants and c is a data consumer-chosen input variable.
If $\boldsymbol{V} = (E(v_1, r_1), \ldots, E(v_m, r_m))$ and f_h and f' have been derived from the same function f the following holds:

$$D_r \circ f_h(\boldsymbol{V}, E(c, 1)) = f' \circ D_r(\boldsymbol{V} \cup E(c, 1)) \tag{3}$$

Then, the data consumer sends $f_h(\boldsymbol{V}, E(c, 1))$, $\boldsymbol{R}$, $\boldsymbol{I}$ and $\boldsymbol{S}$ to each available key-managing authority instance for decryption.

Decryption: Every key-managing authority instance performs two checks:
First, for the freshness guarantee: For every $I_j = (j, P_j(v_j)) \in \boldsymbol{I}$ and $R_j \in \boldsymbol{R}$ it checks the signature S_j and the freshness of every used data item: $C(j) < P_j(v_j)$. If successful, $C()$ is updated so that: $C(j) \leftarrow P_j(v_j)$.
Second, for correct function evaluation with data producers' data items: It derives function f' from f. f' represents the homomorphic operations on the random part of the ciphertext of f (see Equations 1,2):

$$f'(\boldsymbol{V}) = \prod_{i=1}^{m} D_r(E(v_i, r_i))^{a_i}$$

It checks if (according to Equations 3):

$$D_r \circ f_h(\boldsymbol{V}, E(c, 1)) = f' \circ D_v(\boldsymbol{R})$$

The key-managing authority instance accepts the proof if both checks pass. Then, like in the naive protocol it applies the function $m_f(s, x)$ which adds a random sample according to the sensitivity of f (see Section 3.1) and seed s to the decrypted function result: $x = m_f(s) \circ D_v \circ f_h(\boldsymbol{V}, E(c, 1))$. We use a pseudo-random function on the data item information $\boldsymbol{I}$ as seed s in order to create the randomness, so that every instance of the distributed key-managing authority will add the same random sample and thus obtains the same result x. This allows the k-th key-managing authority instance to finally create a share s_k of x (using the same seed s, but a different pseudo-random function) and return that to the data consumer.

Remark 1. Note that, if the data consumer supplies different input $\boldsymbol{I'}$ to one key managing authority, then its resulting share will be uniformly randomly distributed. Our ZKP only covers malicious behavior in the computation of the statistical function f by the data consumer and not its decryption. The data consumer can always prevent the correct decryption of the statistics, but this only prevents him from obtaining the result.

Assembly: The data consumer assembles the decryption shares from at least $(t + 1)$ key-managing authority instances and reassembles the decrypted, differentially-private, statistics result:

$$D_v(m_f(\boldsymbol{I}) \circ_h f_h(\boldsymbol{V}, E(c, 1)))$$

Theorem 1. *Our zero knowledge proof is complete, sound and honest-verifier zero-knowledge.*

Proof. Following Remark 1, we can assume that the vector $\boldsymbol{I}$ is static and omit it from the proof. For completeness: Assume that the data consumer, according to protocol step *Calculation*, chooses $\boldsymbol{V}$,$\boldsymbol{R}$ and $\boldsymbol{S}$, computes the pre-arranged, linear function f_h with $\boldsymbol{V}$ and supplies the function result $f_h(\boldsymbol{V}, E(c, 1))$, $\boldsymbol{R}$ and $\boldsymbol{S}$ to the key-managing authority. Then the key-managing authority will verify that $\boldsymbol{R}$ only contains fresh entries and that those entries have valid signatures in $\boldsymbol{S}$ and according to the last check of the protocol's step *Decryption* it will accept the proof if: $D_r \circ f_h(\boldsymbol{V}, E(c, 1)) = f' \circ D_v(\boldsymbol{R})$
Which is true if $\boldsymbol{R}$ contains the corresponding entries to $\boldsymbol{V}$ (using Equation 3):

$$\begin{aligned} &D_r \circ f_h(\boldsymbol{V}, E(c, 1)) = f' \circ D_r(\boldsymbol{V} \cup E(c, 1)) \\ =&f'((D_v(R_1), \ldots, D_v(R_m)) \cup 1) \\ =&f' \circ D_v(\boldsymbol{R}) \\ &\square \end{aligned}$$

For soundness, we show by contradiction: If our ZKP was not sound, IND-CPA game for the Paillier cryptosystem would be solvable. By providing a simulator that reduces the IND-CPA challenge to the problem of forging the ZKP we will show that forging the ZKP must be hard as well.

For soundness of the prover we need to differentiate between data producers (system environment) and consumer (prover). We use – in this and the subsequent part of the proof – a signature oracle for the data consumers that returns any valid signature. In this part of the proof, the data consumer has no access to this oracle.

Our simulator first interacts with the IND-CPA challenger. It provides two values as input: r_0 and r_1. The challenger returns the encryption of a randomly chosen value $c_x = E(r_0|r_1, r'_x)$. Without loss of generality, assume that the function f takes m data items from data producers as inputs. Now, the simulator takes $(m - 1)$ triples $\{v_i, r_i, r'_i\}$ and encrypts and signs them, resulting in $(m-1) \times \{E(v_i, r_i), E(r_i, r'_i), S(E(r_i, r'_i), (j, P_j(v_i)))\}$. The simulator also signs c_x and some data item information tuple y (using the oracle) and supplies this incomplete ZKP $Z = \{(m - 1) \times \{E(v_i, r_i), E(r_i, r'_i), S(E(r_i, r'_i), (j, P_j(v_i)))\} \cup \{c_x, S(c_x, y)\}$ to the data consumer. Note that in this setup the data consumer needs to complete the ZKP, e.g. by supplying the missing (input value) ciphertext: $E(v_x, r_0|r_1)$. Then the data consumer engages in communication with the key-managing authority, with our simulator listening in their communication. The data consumer passes the full ZKP $Z \cup E(v_x, r_0|r_1)$ to the key-managing authority and obtains the decrypted function result $f(\boldsymbol{V})$. The simulator still knows all v_i (except v_x) and can then infer v_x from the function result. Consequently, the simulator can also infer a plausible $E(v_x, r_0|r_1)$. Now, the simulator performs the last check: It creates $E(v_x, r_0)$ and $E(v_x, r_1)$ and compares them to $E(v_x, r_0|r_1)$. It thus solves the IND-CPA problem. $\square$

For honest-verifier zero-knowledge we give a simulator of the authority's (verifier's) view only from its input and output. We stress, that for zero-knowledge of the verifier we can view the data producers and consumer as one entity. The verifier – and not the prover – holds the private keys for the encryption of the values submitted during the proof. Consequently we need to simulate the plaintexts of the encryption. Furthermore, we need to take care of the randomization of the encryption, since it is not always uniformly distributed.

In order to simulate $f_h(\boldsymbol{V}, E(c, 1))$ the simulator uniformly chooses two random values $x \in \mathbb{Z}_n$ and $r \in \mathbb{Z}_n^*$. The simulated value is $E(x, r)$. Note that, since c can be drawn uniformly from $\mathbb{Z}_n$, so can x. Then, in order to simulate $\boldsymbol{R}$ the simulator uniformly chooses $m - 1$ random values $r_i \in \mathbb{Z}_n^*$ $(1 \leq i \leq m - 1)$. It sets $r_m = r(\prod_{i=1}^{m-1} r_i)^{-1}$. The simulated values are the ciphertexts $E(r_i, r'_i)$ where the r'_i are drawn uniformly from $\mathbb{Z}_n^*$. All the random values in the message of the ZKP are identically distributed – including their arithmetic relationship.

Last, in order to simulate the signatures $\boldsymbol{S}$ the simulator invokes the signature oracle on the simulated vector $\boldsymbol{R}$. If this oracle works properly, all signatures will be valid. This completes our simulator.

6 Related Work

Attacks on smart metering privacy: Several works [8,9,15,16,17,19,22] have covered the area of behavior analysis from energy consumption traces. The authors of [17] developed a system which inferred behavior events from the electrical consumption data and evaluated the performance of their approach with control data from video surveillance. This enables them to construct a sample disclosure metric that "...associates data quality (accuracy of readings, time resolution, types of readings, and so on) from a particular source with the information that the data could reveal." [15] focuses on detecting and characterizing different appliances according to load signatures.

Privacy-preserving smart meter billing: A cryptographic approach to privacy in smart meter billing has been presented in [12]. A privacy component homomorphically calculates the price locally in the household and only reports the final price and a cryptographic proof over commitments on the meter readings to the supplier. The proof allows the supplier to verify the correct calculation and tariff without ever receiving plaintext values. Another cryptographic approach very similar to [12] is described in [24]. It focuses on realizing a variety of different tariff types with a cryptographic solution and reducing the complexity of the calculations in the smart meter.

Private distributed aggregation model: In the following we describe two general protocols [25,23] in the *private distributed aggregation model* and one [2] in the *fault-tolerant, private distributed aggregation model.*

[25] describes a system where users provide homomorphically encrypted data items with individually added random noise. The aggregator homomorphically aggregates these data items and decrypts them. During aggregation individual amounts of random noise cancel each other out except for a specific amount that guarantees computational differential privacy.

In [23] bi-directional communication between users and the aggregator is required. First, users supply their homomorphically encrypted and randomized value to the aggregator for aggregation. Then, the aggregator sends the aggregated perturbed value back to all users for decryption. Every user removes its individually added random noise (except for differential privacy) and replies with a decryption share of the the final result which are finally combined by the aggregator.

In [2] a system of intersecting user groups allows the aggregator to compensate for user failures with the help of redundant information. As every user's data item is represented in the aggregate of several groups it can be recovered even if some group aggregates cannot be recovered due to user failures.

Privacy-preserving smart metering statistics: Furthermore, we provide an overview on different (non-fault-tolerant) approaches that ensure privacy for different applications of smart metering data: leakage detection [5] by homomorphic encryption, general aggregation [14] and comparison by secret-sharing, aggregation by third party [1,21] or aggregation [1] by randomization.

In [5] a privacy-preserving detection algorithm for leakages in electricity distribution has been proposed. Using homomorphic encryption and secret sharing several smart meters engage in a protocol with a (potentially malicious) substation that searches for differences between its own measurement and the sum of all smart meter readings. While both compute the sum in a private manner, our approach (obviously) does not require a measurement of aggregate consumption, requires only uni-directional communication and tolerates failing smart meters.

In [14] the authors propose several protocols for privacy-preserving aggregation or comparison on smart metering data based on secret-sharing. They also provide an extensive analysis w.r.t. cryptographic verifiability, their computational and communicational complexities and their applicability for settlement and profiling applications. Their work differs from ours mainly in their requirement of smart meters to form groups over which aggregates are computed which also implicates meter to meter communication and implicitly assumes no failures.

Furthermore, in [1] a model for measuring privacy in smart metering is developed and subsequently two different solutions to privacy are presented: A trusted third party approach, where aggregation takes place at the third party and alternatively the approach of masking individual values with added noise directly at the smart meters which is canceled out in the sum over all meters at the supplier. The trusted third party of their first approach is able to calculate arbitrary statistics. However, as it has to perform all calculations instead of one decryption, as in our case, it requires more computational power and storage than in our protocol. Their second approach has reduced variance in the function result with increasing number of meters in contrast to our approach. Ours, however, is independent of the total amount of smart meters contributing readings and allows the calculation of arbitrary linear functions.

Finally, in [21] the authors suggest to use the electrical grid infrastructure as a trusted third party to anonymize up-to-date consumption values constantly sent out by smart meters. This third party verifies the authenticity of the data, removes the identifying information and forwards it to the consumer of this data. This solution provides privacy by anonymizing information at a trusted third party before forwarding them to the data consumer which requires a similar computational and storage effort as in [1].

Application of differential privacy in smart metering: In [3] the authors build upon a previous work [24] and additionally apply differential privacy to hide any information leakage that is implied by the billing amount itself. Using differential privacy the consumer presents a randomized bill to the supplier that is higher then the actual one. Their approach especially targets the remaining information leakage implied by the bill calculation function itself, in which it is similar to ours. However, we target forecasting where smart meter readings are required in real-time and thus aggregation over time is not an option.

7 Summary and Conclusion

We provide a protocol in the *fault-tolerant, private distributed aggregation model* that allows a data consumer to calculate unbounded statistics (weighted sums) over homomorphically encrypted sensitive data items from data producers. Our protocol is fault-tolerant, as the data consumer can choose to calculate over an arbitrary subset of all available data items, i.e., failing data producers do not prevent the statistics calculation. It is also privacy-preserving, because a (possibly distributed) key-managing authority ensures differential privacy before responding to the data consumer's decryption request for the homomorphically encrypted statistics result. Our protocol is secure against malicious data consumers (aggregators) and features aggregator obliviousness, differential privacy and a uni-directional communication model between data producers and data consumers. In comparison to the other existing protocol [2] in this model the accuracy of our statistics calculation is higher, particularly in the presence of failures. We are also more flexible, since we allow the non-interactive, intermittent change of the statistics function and we do not require group key management.

In summary, our protocol provides the best suitable foundation for privacy-preserving, real-time energy consumption forecasting using smart meters. Future work is to extend the availability of consumption data to other data consumers, such as regulators that need to verify single data items, and to incorporate other data sources, such as weather forecast and television programs.

8 Acknowledgments

Marek Jawurek's work in this paper was partly funded by the German Federal Ministry of Economics and Technology (BMWi) as part of the MEREGIOmobil project with reference number 01ME09007 and the MEREGIO project with reference number 01ME08006.

References

1. J.-M. Bohli, O. Ugus, and C. Sorge. A privacy model for smart metering. In *Proceedings of the First IEEE International Workshop on Smart Grid Communications (in conjunction with IEEE ICC 2010)*, 2010.
2. T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, FC '12, 2012.
3. G. Danezis, M. Kohlweiss, and A. Rial. Differentially private billing with rebates. Cryptology ePrint Archive, Report 2011/134, 2011. `http://eprint.iacr.org/`.
4. C. Dwork. Differential privacy. In *in ICALP*, pages 1–12. Springer, 2006.
5. F. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *Proceedings of the 6th International Workshop on Security and Trust Management*, 2010.
6. A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 351–360, New York, NY, USA, 2009. ACM.
7. O. Goldreich and A. Warning. Secure multi-party computation, 1998.
8. G. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870 –1891, dec 1992.
9. G. W. Hart. Residential energy monitoring and computerized surveillance via utility power flows. *IEEE Technology and Society Magazine*, June 1989.
10. W. Heck. Smart energy meter will not be compulsory. NRC Handelsblad (online), April 2009. `http://www.nrc.nl/international/article2207260.ece/Smart_energy_meter_will_not_be_compulsory`.
11. A. Jamieson. Smart meters could be 'spy in the home'. Telegraph (UK) (online), October 2009. `http://www.telegraph.co.uk/finance/newsbysector/energy/6292809/Smart-meters-could-be-spy-in-the-home.html`.
12. M. Jawurek, M. Johns, and F. Kerschbaum. Plug-in privacy for smart metering billing. *CoRR*, abs/1012.2248, 2010.
13. M. Jawurek, M. Johns, and K. Rieck. Smart metering de-pseudonymization. In *ACSAC*, pages 227–236, 2011.
14. K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In S. Fischer-Hübner and N. Hopper, editors, *PETS*, volume 6794 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2011.
15. H. Lam, G. Fung, and W. Lee. A novel method to construct taxonomy electrical appliances based on load signaturesof. *Consumer Electronics, IEEE Transactions on*, 53(2):653 –660, may 2007.
16. C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong. Power signature analysis. *Power and Energy Magazine, IEEE*, 1(2):56 – 63, mar-apr 2003.

17. M. A. Lisovich, D. K. Mulligan, and S. B. Wicker. Inferring personal information from demand-response systems. *IEEE Security and Privacy*, 8(1):11–20, 2010.
18. F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, 2010.
19. A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 61–66, New York, NY, USA, 2010. ACM.
20. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *IN ADVANCES IN CRYPTOLOGY — EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.
21. R. Petrlic. A privacy-preserving concept for smart grids. In *Sicherheit in vernetzten Systemen: 18. DFN Workshop*, pages B1–B14. Books on Demand GmbH, 2010.
22. A. Prudenzi. A neuron nets based procedure for identifying domestic appliances pattern-of-use from energy recordings at meter panel. In *Power Engineering Society Winter Meeting, 2002. IEEE*, volume 2, pages 941 – 946 vol.2, 2002.
23. V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.
24. A. Rial and G. Danezis. Privacy-preserving smart metering. Technical report, Microsoft Research, November 2010.
25. E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.