Johannes Ernst* and Alexander Koch

Private Stream Aggregation with Labels in the Standard Model†

Abstract: A private stream aggregation (PSA) scheme is a protocol of n clients and one aggregator. At every time step, the clients send an encrypted value to the (untrusted) aggregator, who is able to compute the sum of all client values, but cannot learn the values of individual clients. One possible application of PSA is privacy-preserving smart-metering, where a power supplier can learn the total power consumption, but not the consumption of individual households.

We construct a simple PSA scheme that supports labels and which we prove to be secure in the standard model. Labels are useful to restrict the access of the aggregator, because it prevents the aggregator from combining ciphertexts with different labels (or from different timesteps) and thus avoids leaking information about values of individual clients.

The scheme is based on key-homomorphic pseudorandom functions (PRFs) as the only primitive, supports a large message space, scales well for a large number of users and has small ciphertexts.

We provide an implementation of the scheme with a lattice-based key-homomorphic PRF (secure in the ROM) and measure the performance of the implementation. Furthermore, we discuss practical issues such as how to avoid a trusted party during the setup and how to cope with clients joining or leaving the system.

Keywords: Private Stream Aggregation, Aggregator Obliviousness, Standard Model, Pseudorandom Function, Lattice-Based Cryptography, Learning With Rounding, Smart-Meters

DOI 10.2478/popets-2021-0063

Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Smart meters are becoming more and more ubiquitous in many countries. This has advantages for the power suppliers, because they get near real-time power consumptions from their clients, which they can use for load-balancing and prediction in their networks. However, this raises the question of privacy. Sensitive information like the work schedule can easily be guessed from the variation in the power consumption of a household. In practice, it is often sufficient for the power supplier to know the sum of the consumptions of all clients within a certain area, and this is exactly what a protocol for private stream aggregation (PSA) [22] can offer. PSA considers the scenario where an aggregator wishes to periodically compute the sum of values that are supplied by different clients. The values are encrypted in such a way that the aggregator can only compute their sum, but not the individual values. This is captured in the game-based security definition of aggregator obliviousness (AO), which is given in Definition 3.

It is desirable for PSA schemes to support the use of *labels*. Labels restrict the aggregator to only be able to compute the sum of values which were encrypted under the same label. This prevents the aggregator from mixing ciphertexts of different time steps and thereby learning more about the individual values than would otherwise be possible.

A clear advantage of PSA schemes is that they do not require the clients to exchange messages, nor does the aggregator need to send messages to the clients. After the keys have been distributed, the only messages in the protocol are the ciphertexts which the clients send to the aggregator. PSA stays secure even if the aggregator colludes with an arbitrary subset of the clients. In that case, the aggregator only learns the sum of the non-colluding clients' values.

When we apply a PSA protocol to the smart meter scenario, this means that every smart meter encrypts (e.g. every fifteen minutes) its current power consumption and sends it to the supplier. The supplier then is able to compute the sum and thereby learns the power consumption of all households in the specific area. Because of the way the clients' values are encrypted, the

***Corresponding Author: Johannes Ernst:** University of St. Gallen (most of the work done while at KIT, Karlsruhe)
Alexander Koch: Competence Center for Applied Security Technology (KASTEL), Karlsruhe Institute of Technology

† An extended abstract of this work appeared in [19]

only information the power supplier gets is the sum of all values.

To further protect the privacy of each client, techniques from differential privacy can be used. In this case, it means that every client adds a small amount of noise to their value before encrypting it. This induces a small error in the resulting sum, but in many cases a small error is tolerable. However, in this paper we focus on the encryption part. Differential privacy can then be added by standard techniques, e.g. as described in [22].

Apart from privacy-preserving smart metering, PSA has a lot more of possible applications. For example it can be used in federated learning in a similar way to the protocol of [11], to compute a global model update from the local updates that are supplied by the clients. This can help to prevent an adversary from using the model to infer information on the, possibly sensitive, data which the clients used to train the model.

PSA (without differential privacy) can be seen as special case of inner-product multi-client functional encryption (IP-MCFE) first introduced by [16]. In inner-product multi-client functional encryption there are several clients and one or more aggregators. The aggregators can ask for functional decryption keys associated with an arbitrarily chosen vector y. The functional decryption key then enables them to compute the inner product of the clients' values with the vector y. When we only allow the vector $y = (1, \ldots, 1)$, then this is exactly the case of PSA (without differential privacy). The main challenge in IP-MCFE is that the scheme must be secure, although the vectors y are not known at the beginning and an aggregator can hold keys for many different vectors.

1.1 Contribution

Provably secure PSA scheme with labels in the standard model: The scheme we propose is the first PSA scheme that both supports labels and is proven to be aggregator oblivious (AO) *with adaptive corruptions* in the *standard model.* Although, strictly speaking, the IP-MCFE scheme of [1] can also be used as PSA scheme with the same properties, compared to their scheme, ours is more efficient. The size of each user secret key and the length of the ciphertexts in their scheme grows linearly with the number of clients, whereas ours are constant, i.e. independent of the number of clients. Becker et al. have also proposed a PSA scheme in the standard model, but without labels [9]. They roughly explain how to extend their scheme to support labels, but they provide no security proof of this extension. Furthermore, their scheme seems to be subject to a patent [8]. Our scheme is very similar to the PSA scheme of [24] who used key-homomorphic *weak* PRFs but only proved non-adaptive security in the standard model. Also, as opposed to our scheme, the labels have to be precomputed at setup time and distributed to all clients. Thus, the scheme does not support an unbounded number of labels and the key size grows linearly with the number of labels.

Our scheme needs as its only building block a key-homomorphic pseudorandom function (PRF) whose output space is $\mathbb{Z}_R$ for some integer R. It thus relies only on secret-key primitives and is quite flexible. It makes efficient use of the key-homomorphic PRF, as both encryption and decryption only require one PRF evaluation. The scheme can be instantiated with a lattice-based key-homomorphic PRF, which are assumed to be secure against quantum adversaries.

Additionally, we implemented the scheme and a simple key-homomorphic PRF based on the learning with rounding (LWR) problem. For simplicity and efficiency, we chose a PRF that relies on the random oracle model (ROM). The performance tests show that both encryption and decryption are very efficient, in this case.

One restriction of our scheme is that it only supports the encryption of one message per label. However, this is a mild restriction, because any (correct) PSA scheme leaks information about individual messages, if a user encrypts more than one message per label.

Furthermore, we concretely describe how the scheme can be used for privacy-preserving smart-meter aggregation. We discuss issues that arise in that setting, such as clients joining or leaving the system and how to execute the setup without a trusted party.

1.2 Related Work

In this section, we give an overview of other work that is related to ours.

1.2.1 Privacy Preserving Aggregation

Shi et al. [22] were the first to formalize the notion of PSA together with the security definition of AO. They propose a scheme that is based on the Decisional Diffie–Hellman (DDH) problem and prove it to be aggregator oblivious in the ROM. Despite its simplicity, the decryption procedure is inefficient because it has to compute a discrete logarithm. This limits the size of the message space such that the discrete logarithm can be computed in reasonable time.

Subsequently, Benhamouda et al. [10] propose a general way to build PSA schemes from key-homomorphic smooth projective hash functions (SPHF). Their construction yields schemes that are aggregator oblivious in the ROM. They give concrete instantiations from the DDH-, DCR- and several other assumptions. An advantage over previous schemes is the low reduction loss, which does not depend on the number of users, but only on the maximum number of labels used. This allows for smaller keys and thereby makes the schemes more efficient.

Valovich constructs a PSA scheme from key-homomorphic weak PRFs [24]. In contrast to the other PSA schemes, the author only considers semi-honest adversaries. He proves the scheme to be aggregator oblivious for non-adaptive corruptions in the standard model. For proving adaptive security, the author resorts to the ROM. The main differences to our scheme are that [24] has a weaker security model and that the author uses *weak* PRFs. Because of the use of *weak* PRFs, the labels need to be uniformly random. Therefore, the set of labels is created at setup time and given to all parties, to ensure that everyone uses the same labels. This means that the scheme does not support an unbounded number of labels and that the key size grows linearly with the number of labels.

Becker et al. propose a generic PSA scheme [9] that can be instantiated with an additively homomorphic encryption scheme, where the addition of ciphertexts corresponds to the addition of plaintexts, with the additional property that the ciphertexts are indistinguishable from random strings. The security of their scheme relies on the Learning with Errors (LWE) assumption, or its ring variant, and is proven to be secure in the standard model. Two advantages over the scheme of Shi et al. [22] are the prospective post-quantum security and the more efficient decryption algorithm. In contrast to the other PSA schemes, they provide an implementation and give performance results. Their implementation has a message space of size 2^{16}. The scheme does not directly support labels which limits the practical use cases. Although the authors sketch how to extend the scheme to work with labels, they provide no security proof for that.

Except for [9] all other PSA schemes, including ours, have the restriction that every client must only encrypt one message per label. However, this restriction is also reasonable from a security perspective, because even a perfectly secure PSA scheme leaks information about individual client values, if a client encrypts more than one message per label. We elaborate on this in Section 2.4.1.

The advantage of our scheme over [22] and the DDH version of [10] is that the size of the message space is not restricted. The advantage over [10, 22, 24] is that we prove our scheme to be secure under adaptive corruptions in the standard model. A disadvantage of our scheme is the larger key size. Benhamouda et al. [10] report key sizes of 592 bit for 128 bit security for their DDH based scheme when using elliptic curves. For our choices of parameters our scheme provides a security of 114 bit[1] with key-sizes of 268288 bit. Note however, that this is mainly due to the use of a post-quantum secure PRF. We give more details on this in Section 4.4. Advantages of our scheme over [9] are that our scheme has a security proof for the case that labels are used and that it is much simpler. Our scheme relies on key-homomorphic PRFs while theirs needs an additively homomorphic public key encryption scheme with ciphertexts indistinguishable from random.

The following works are not directly comparable to ours, because their focus is a bit different.

Emura considers the verifiability of aggregated sums [18]. This means that, when the aggregator publishes the sum, they must provide a publicly verifiable proof that the sum is correct. The author proposes two schemes, which are based on the DDH version of [10] and require pairings. In this paper we do not consider the verifiability of the result.

Ács and Castelluccia [3] propose an aggregation scheme that uses similar pair-wise masking as the MCFE scheme of [1]. The users in their scheme agree on shared keys via the Diffie–Hellman key-exchange. Additionally, the scheme offers a mechanism by which the aggregator can decrypt the sum, even when some clients drop out. As opposed to PSA, this mechanism needs interaction between the clients and the aggregator. The authors do not provide a security proof, but argue that the scheme is secure against certain attacks.

Bonawitz et al. construct a protocol for the aggregation of model updates in distributed machine learning [11]. Their protocol is resistant to user failures and is proven secure against malicious adversaries in the ROM. They use pairwise masks that are set up by Diffie–Hellman key exchanges between the users. The protocol has four rounds of communication to aggre-

[1] when used with 1000000 clients. With 10000 clients the security is 132 bit)

gate one model update. The key difference to PSA is that their protocol is resistant to user failures, but requires several rounds of communication, whereas PSA is non-interactive.

1.2.2 Multi-Client Functional Encryption

Chotard et al. [16] are the first to define (decentralized) multi-client functional encryption (DMCFE) and show two instantiations for the inner product functionality. Their schemes have several practical limitations and rely on pairings and the ROM. Abdalla et al. [2] address these limitations and remove the need for pairings, while still relying on the ROM.

Finally, Abdalla et al. [1] construct the first MCFE scheme with labels that is secure in the standard model. Their scheme works with any PRF and (single-input) functional encryption scheme for inner products. Due to that, their scheme can be based upon many different mathematical problems including LWE, DDH and DCR. Our security proof strongly relies on techniques from the security proof of their MCFE scheme.

The main differences between PSA and MCFE are that in MCFE the aggregator(s) can compute inner products with many different vectors and not just the vector consisting of one-entries. Furthermore, in MCFE these vectors can be chosen by the aggregator(s) adaptively during the protocol execution. This make schemes for MCFE harder to construct and usually less efficient. Nevertheless, both areas have a lot of techniques in common.

1.3 Concurrent Work

Independent of and concurrent to our work, two more papers on PSA have been published in online pre-print archives recently.

The authors of [23] propose two PSA schemes based on variants of two fully homomorphic encryption schemes. The security of both schemes relies on the ring-LWE assumption. They also implement the schemes and provide a performance analysis. According to this analysis our scheme seems to be faster, however.

Waldner et al. propose a PSA scheme based on PRFs [25]. As opposed to our scheme, the PRF does not need to be key-homomorphic. This enables the use of very efficient PRFs. In [25], the authors use AES and SHA3. However, this comes at the cost of requiring n evaluations of the PRF for encrypting one message, where n is the number of users. The authors also provide an implementation and performance results.

In Section 4, we compare the running time of the aforementioned schemes with our implementation.

Table 1 shows a comparison of the different properties of the PSA schemes that we described in this section.

Scheme	Proof in standard model	Number supported labels	Adaptive corruptions	Encryption cost per client
Shi et al. [22]	✗	unbounded	✓	$\mathcal{O}(1)$
Benhamouda et al. [10]	✗	unbounded	✓	$\mathcal{O}(1)$
Valovich [24] (1st scheme)	✓	bounded	✗	$\mathcal{O}(1)$
Valovich [24] (2nd scheme)	✗	bounded	✓	$\mathcal{O}(1)$
LaPS [9]	✓	none	✓	$\mathcal{O}(1)$
SLAP [23]	✗	unbounded	✓	$\mathcal{O}(1)$
LaSS [25]	✓	unbounded	✓	$\mathcal{O}(n)$
Our scheme	✓	unbounded	✓	$\mathcal{O}(1)$

Table 1. Comparison of PSA schemes. Note that the DDH based schemes in [22] and [10] have the limitation that the message space needs to be small in order to allow taking a discrete logarithm in reasonable time.

1.4 Outline

We give the necessary definitions and background in Section 2. In Section 3 we explain our PSA scheme and prove its security according to the game-based security definition of AO. In Section 4 we describe the implementation and choices of parameters and give performance results. In Section 5 we discuss several issues related to the deployment of the scheme in practice, with a focus on smart-meters. The last section summarizes our paper.

2 Preliminaries

In this section, we explain our basic notation and define the cryptographic problems and primitives we use in this paper.

2.1 Notation

Here, we quickly explain some of the notation that we use. By $[n]$ we denote the set $\{1, \ldots, n\}$ and with $[n]_0$ we mean $\{0, \ldots, n\}$. With $\log(x)$ we mean the logarithm to base 2 and with ln(x) we mean the logarithm to base e. As security parameter we use λ. Lower-case boldface letters such as $\mathbf{v}$ denote vectors. We use the terms *client* and *user* synonymously. By a *PPT Turing machine*, we mean a probabilistic Turing machine that runs in polynomial time. By $x \leftarrow_{\$} \mathcal{X}$ we mean that x is chosen uniformly random from the set $\mathcal{X}$. With $\langle \mathbf{x}, \mathbf{y} \rangle$ we denote the inner-product of two vectors $\mathbf{x}$ and $\mathbf{y}$. Let $q, p \in \mathbb{N}$ with $q > p$. Then, for a value $x \in \mathbb{Z}_q$ we define $\lfloor x \rceil_p := \lfloor x \cdot p/q \rceil$.

2.2 Learning With Rounding (LWR)

We will define the learning with rounding (LWR) problem, which can be seen as a deterministic version of the learning with errors (LWE) problem. Learning with rounding was introduced in [7] and has turned out to be very useful to construct secret-key primitives such as pseudorandom functions.

Definition 1. Let $\lambda, q, p \in \mathbb{N}$, with $q > p$ and $\mathbf{s} \leftarrow_{\$} \mathbb{Z}_q^{\lambda}$. Let $L_{\mathbf{s}}$ be the following distribution over $\mathbb{Z}_q^{\lambda} \times \mathbb{Z}_p$: Choose $\mathbf{a} \leftarrow_{\$} \mathbb{Z}_q^{\lambda}$ and output $(\mathbf{a}, \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p)$. The (decision) LWR problem then is to distinguish between the distribution $L_{\mathbf{s}}$ and the uniform distribution over $\mathbb{Z}_q^{\lambda} \times \mathbb{Z}_p$.

We will use a key-homomorphic pseudorandom function that is based on the LWR problem to instantiate our scheme. Next we define pseudorandom functions and key-homomorphic pseudorandom functions.

2.3 Pseudorandom Functions

Intuitively, a pseudorandom function (PRF) is a function that is indistinguishable from a random function (RF). A random function is a function that returns truly random values on all distinct inputs. We use pseudorandom functions $\mathsf{PRF}_k \colon \mathcal{X} \to \mathcal{Y}$ that are indexed by a key $k \in \mathcal{K}$. For a PPT-adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage in distinguishing a pseudorandom function PRF from a random function as

$$\begin{aligned}&\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},\mathsf{PRF}}(\lambda)\\ :=&|\Pr[\mathrm{Exp}^{\mathsf{PRF}}(\lambda,\mathcal{A}) = 1] - \Pr[\mathrm{Exp}^{\mathsf{RF}}(\lambda,\mathcal{A}) = 1]|,\end{aligned}$$

where the experiments $\mathrm{Exp}^{\mathsf{PRF}}(\lambda,\mathcal{A})$, and $\mathrm{Exp}^{\mathsf{RF}}(\lambda,\mathcal{A})$ are defined as follows:

```
$\mathrm{Exp}^{\mathsf{PRF}}(\lambda,\mathcal{A})$
$k \leftarrow_{\$} \mathcal{K}$
$b \leftarrow \mathcal{A}^{\mathsf{PRF}_k(\cdot)}$
return $b$
```

```
$\mathrm{Exp}^{\mathsf{RF}}(\lambda,\mathcal{A})$
$k \leftarrow_{\$} \mathcal{K}$
$b \leftarrow \mathcal{A}^{\mathsf{RF}(\cdot)}$
return $b$
```

In the first case, $\mathcal{A}$ has oracle access to PRF indexed by a random key k, whereas in the second case $\mathcal{A}$ has oracle access to a random function. Intuitively, $\mathcal{A}$'s goal can be seen as finding out whether they are in $\mathrm{Exp}^{\mathsf{PRF}}$ or $\mathrm{Exp}^{\mathsf{RF}}$. A pseudorandom-function is *computationally indistinguishable from a random function*, if for all PPT-adversaries $\mathcal{A}$, there exists a negligible function negl such that for all sufficiently large λ it holds that

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},\mathsf{PRF}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.3.1 Key-Homomorphic Pseudorandom Functions

A useful special case of pseudorandom functions are key-homomorphic pseudorandom functions. A pseudorandom function $\mathsf{PRF}_k \colon \mathcal{X} \to \mathcal{Y}$ is *key-homomorphic*, if for all $x \in \mathcal{X}$, $\mathsf{PRF}_{(\cdot)}(x)$ is a group homomorphism between the key space $\mathcal{K}$ and $\mathcal{Y}$. To define this formally, let $(\mathcal{K}, *)$ and $(\mathcal{Y}, \bullet)$ be groups. Then, for all $x \in \mathcal{X}$, $k_1, k_2 \in \mathcal{K}$

$$\mathsf{PRF}_{k_1}(x) \bullet \mathsf{PRF}_{k_2}(x) = \mathsf{PRF}_{k_1 * k_2}(x)$$

must hold. A key-homomorphic PRF must fulfill the same security definition as a PRF.

A PRF is *almost* key-homomorphic if $\mathsf{PRF}_{k_1+k_2}(x) = \mathsf{PRF}_{k_1}(x) + \mathsf{PRF}_{k_2}(x) + e$ for a small $e \in \mathbb{N}$. The PRF we use as main building block in our PSA scheme is *almost* key-homomorphic with $e \in \{0, 1\}$.

2.4 Private Stream Aggregation

In private stream aggregation (PSA) we have an aggregator and several clients. In each time step, the clients send an encrypted value to the aggregator. The aggregator is then able to compute the sum of these values but no individual client value. It is important that the aggregator can only compute the sum of values that were encrypted under the same time-stamp or label. There is no further interaction beyond the messages that the clients send to the aggregator.

Often in PSA *differential privacy* is additionally used. For this, every client adds a small amount of noise to their value before encrypting it. The aggregator can then compute the resulting noisy sum of the plaintexts. When the noise is chosen appropriately, and enough clients honestly add noise, the noisy sum maintains differential privacy. However, in this paper we are only concerned with the encryption and will leave out the noise in the definition of PSA. Nevertheless, it is no problem to add differential privacy via standard techniques (e.g. as in [22]).

Our definition roughly follows the definition of [10], as it is also without noise.

Definition 2 (Private Stream Aggregation)**.** A *private stream aggregation* scheme PSA over $\mathbb{Z}_R$ (for $R \in \mathbb{N}$) and label space $\mathcal{L}$, consists of the following three PPT algorithms for the setup, the encryption and the decryption of the aggregate sum:

- $\mathsf{Setup}(1^\lambda, 1^n)$: Given the security parameter λ and the number of users n in unary, it outputs public parameters pp and $n+1$ keys $(k_i)_{i \in [n]_0}$. The key k_0 is the (secret) key of the aggregator, and each k_i is a (secret) key of a user $i \in [n]$.
- $\mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$: Given the public parameters pp, a key k_i of user $i \in [n]$, a label $l \in \mathcal{L}$ and a value $x_i \in \mathbb{Z}_R$, it outputs an encryption c_i of x_i under key k_i with label l. This algorithm is supposed to be executed by each user at every time step, where the time step is used as label. The user then sends c_i to the aggregator.
- $\mathsf{AggrDec}(\mathsf{pp}, k_0, l, \{c_i\}_{i \in [n]})$: Given the public parameters pp, the aggregator's key k_0, a label $l \in \mathcal{L}$, and a set of n ciphertexts $\{c_i\}_{i \in [n]}$ that were encrypted under the same label l, it outputs $\sum_{i \in [n]} x_i \pmod R$.

We additionally require $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{AggrDec})$ to satisfy *correctness*, i.e. that for any $n, \lambda \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathbb{Z}_R$ and any label $l \in \mathcal{L}$, that for $(\mathsf{pp}, \{k_i\}_{i \in [n]_0}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, and $c_i \leftarrow \mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$, we have

$$\mathsf{AggrDec}(\mathsf{pp}, k_0, l, \{c_i\}_{i \in [n]}) = \sum_{i \in [n]} x_i \bmod R.$$

In most PSA schemes (including ours) the sum is computed modulo a public integer R. When the goal is to compute the sum over $\mathbb{Z}$ instead of $\mathbb{Z}_R$ then the clients must be restricted to only encrypt values smaller than a certain value ω and R must be chosen to be greater than $n \cdot \omega$. This difference can be important, because some proofs only go through, when the message space is a group. However, in our proofs it makes no difference whether the clients are allowed to encrypt values from $\mathbb{Z}_R$ or $\{0, \ldots, \omega\}$.

Usually in a PSA scheme, a trusted third party executes the setup algorithm and gives the secret keys to the clients and the aggregator. The clients then regularly encrypt some value and send the ciphertext to the aggregator. By calling $\mathsf{AggrDec}$ the aggregator is then able to decrypt the sum of the values. In Section 5.1 we will describe approaches how the trusted setup can be avoided.

Next we define the security notion of aggregator obliviousness. We only define *encrypt-once* security, which is security in the case that every client encrypts only one message per label. This is a reasonable restriction, because it can be easily enforced in practice. Furthermore, encrypting two messages per label leaks the difference of the messages as explained in Section 2.4.1. The PSA schemes of [22] and [10] both have this restriction as well.

Definition 3 (Aggregator obliviousness)**.** The game-based security notion of aggregator obliviousness (AO) is defined via the following security experiment $\mathsf{AO}_b(\lambda, n, \mathcal{A})$, $b \in \{0,1\}$ given in Figure 1. First, the challenger runs Setup and passes the public parameters pp to the adversary $\mathcal{A}$. Then, $\mathcal{A}$ can adaptively ask queries to the following oracles:

$\underline{\mathsf{QEnc}(i, x_i, l)}$: Given a user index $i \in [n]$, a value $x_i \in \mathbb{Z}_R$, and a label l, it answers with $c = \mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$.

$\underline{\mathsf{QCorrupt}(i)}$: Given a user index $i \in [n]_0$ (including the aggregator's index 0), it returns the secret key k_i.

$\underline{\mathsf{QChallenge}(\mathcal{U}, \{x_i^0\}_{i \in \mathcal{U}}, \{x_i^1\}_{i \in \mathcal{U}}, l^*)}$: The adversary specifies a set of users $\mathcal{U} \subseteq [n]$, a label l^* and two challenge messages for each user from $\mathcal{U}$. The oracle answers with encryptions of x_i^b, that is $\{c_i \leftarrow \mathsf{Enc}(\mathsf{pp}, k_i, l^*, x_i^b)\}_{i \in \mathcal{U}}$. This oracle can only be queried once during the game. (If it is not queried, we set $\mathcal{U} = \emptyset$ in the discussion below.)

At the end, $\mathcal{A}$ outputs a guess α, of whether $b = 0$ or $b = 1$.

```
$\mathsf{AO}_b(\lambda, n, \mathcal{A})$
$(\mathsf{pp}, \{k_i\}_{i \in [n]_0}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot), \mathsf{QEnc}(\cdot,\cdot,\cdot), \mathsf{QChallenge}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp})$
if condition $(*)$ is satisfied (see p. 123)
    output $\alpha$
else
    output 0
```

Fig. 1. Aggregator obliviousness experiment for PSA schemes. Depending on the bit b, the oracle QChallenge answers with encryptions of x_i^0 or x_i^1.

To formally define the condition $(*)$, we introduce the following sets:

- Let $\mathcal{E}_l \subseteq [n]$ be the set of all users for which $\mathcal{A}$ has asked an encryption query on label l.
- Let $\mathcal{CS} \subseteq [n]$ be the set of *users* for which $\mathcal{A}$ has asked a corruption query. Even if the aggregator is corrupted, we define this set to only contain the corrupted *users* and not the aggregator.
- Let $\mathcal{Q}_{l^*} := \mathcal{U} \cup \mathcal{E}_{l^*}$ be the set of users for which $\mathcal{A}$ asked a challenge or encryption query on label l^*.

We say that *condition* $(*)$ *is satisfied* (as used in Figure 1), if all of the following conditions are satisfied:

- $\mathcal{U} \cap \mathcal{CS} = \emptyset$. This means that all users for which $\mathcal{A}$ receives a challenge ciphertext must stay uncorrupted during the entire game.
- $\mathcal{A}$ has not queried $\mathsf{QEnc}(i, x_i, l)$ twice for the same (i, l). Doing so would violate the encrypt-once restriction.
- $\mathcal{U} \cap \mathcal{E}_{l^*} = \emptyset$. This means that $\mathcal{A}$ is not allowed to get a challenge ciphertext from users for which they ask an encryption query on the challenge label l^*. Doing this would violate the encrypt-once restriction.
- If $\mathcal{A}$ has corrupted the aggregator *and* $\mathcal{Q}_{l^*} \cup \mathcal{CS} = [n]$ then we require that
$$\sum_{i \in U} x_i^0 = \sum_{i \in U} x_i^1.$$
We will call this condition the *balance-condition.*

The balance condition captures the fact that if $\mathcal{A}$ has corrupted the aggregator and received a ciphertext from every uncorrupted user, then they can compute the sum of the plaintexts. If the plaintexts submitted in the challenge query would sum to different values, then $\mathcal{A}$ could trivially win the game by using their aggregation capability. Note that the balance-condition does not apply if there is a single honest user for which $\mathcal{A}$ did not get a ciphertext on label l^*.

We say that corruptions are *adaptive*, because $\mathcal{A}$ can ask corruption queries depending on previously asked queries. If $\mathcal{A}$ has to decide at the beginning of the game which users they want to corrupt, the term *static* corruptions is used in the literature. In this paper we only consider adaptive corruptions, because it is a more realistic assumption and because security under adaptive corruptions implies security under static corruptions. We define $\mathcal{A}$'s advantage as

$$\begin{aligned}\mathsf{Adv}^{\mathrm{AO}}_{\mathcal{A},\mathsf{PSA}}(\lambda, n) =& |\Pr[\mathsf{AO}_0(\lambda, n, \mathcal{A}) = 1] \\ & - \Pr[\mathsf{AO}_1(\lambda, n, \mathcal{A}) = 1]|.\end{aligned}$$

A PSA scheme is *aggregator oblivious*, if for every PPT adversary $\mathcal{A}$ there is a negligible function negl such that for all sufficiently large λ

$$\mathsf{Adv}^{\mathrm{AO}}_{\mathcal{A},\mathsf{PSA}}(\lambda, n) \leq \mathsf{negl}(\lambda).$$

2.4.1 Inherent Leakage of Sum Queries

Here we will briefly explain why it is dangerous in a PSA scheme, when a client encrypts more than one message per label, even though, the scheme may be formally secure. Imagine that user $i \in [n]$ encrypts both x_i and x_i' as ciphertexts c_i and c_i', respectively, with the same label l. When the aggregator got ciphertexts for the same label l from the other users as well, they can use $\mathsf{AggrDec}$ to compute

$$\begin{aligned}&\mathsf{AggrDec}(\mathsf{pp}, k_0, l, (c_1, \ldots, c_i, \ldots, c_n)) \\ &-\mathsf{AggrDec}(\mathsf{pp}, k_0, l, (c_1, \ldots, c_i', \ldots, c_n)) \\ &= (\sum_{j \in [n]\setminus\{i\}} x_j) + x_i - (\sum_{j \in [n]\setminus\{i\}} x_j) - x_i' = x_i - x_i'.\end{aligned}$$

With this, the aggregator learns the difference of the two messages of user i. It also means that if the aggregator knows one of the two messages, they can compute the other one. If the aggregator has two ciphertexts from more than one client, then they can combine them in arbitrary ways to get even more information. This leakage cannot be avoided, because it is leaked by the sum functionality itself. This is also a reason why, in this paper, we restrict the clients to only encrypt one message per label (encrypt-once).

3 Adaptively Secure PSA

In this section, we construct a scheme for private stream aggregation and prove that it is aggregator oblivious under adaptive corruptions in the standard model. We will define the scheme without noise. The noise can be added via standard techniques (e.g. as in [22]), to ensure differential privacy.

In the security proofs, we use diagrams to illustrate the game hops. Figure 2 shows how to read these diagrams. In this example there are the four games G_0 to G_3. In game G_0, only the unmodified lines are executed, that is the lines which are neither framed nor gray. Thus, in G_0 only line 1 is executed. Game G_1 additionally executes the lines that are framed by a rectangular box, but that are not gray. In our example, G_1 executes

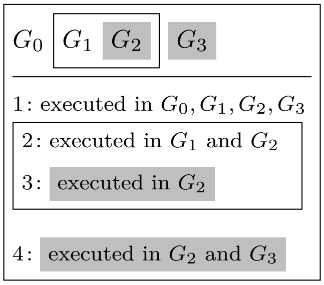
G_0
G_1
G_2
G_3
1: executed in G_0, G_1, G_2, G_3
2: executed in G_1 and G_2
3: executed in G_2
4: executed in G_2 and G_3

Fig. 2. Figure showing how to read the game hop diagrams.

lines 1 and 2. Game G_2 executes all unmodified lines, all framed lines and all gray lines, which in this case are all four lines. In game G_3, only the unmodified lines are executed and the lines that are gray, but not framed. Therefore, in G_3 the lines 1 and 4 are executed.

3.1 The Construction

Our scheme makes use of a key-homomorphic PRF to create pseudorandom pads which are added to the messages as encryption.

Let $\mathsf{PRF}_k \colon \mathcal{X} \to \mathcal{Y}$ be the key-homomorphic PRF, where the key spaces $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are abelian groups. Thus, we have that for all $x \in \mathcal{X}$, $\sum_i \mathsf{PRF}_{k_i}(x) = \mathsf{PRF}_{\sum_i k_i}(x)$ holds. For our use we require that $(\mathcal{Y}, +)$ is the group $(\mathbb{Z}_R, +)$, for some integer R. In Section 4, we describe how to instantiate the scheme with a lattice-based key-homomorphic PRF. Throughout this section we will often write $\sum_{i\in[n]} x_i$ and omit the mod R, when it is clear from the context. That is, $\sum_{i\in[n]} x_i$ denotes the sum in $\mathbb{Z}_R$ and not the sum in $\mathbb{Z}$. Also we will write $\sum_{i\in[n]} k_i$, with which we mean the sum in the group $\mathcal{K}$.

We define the PSA scheme $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{AggrDec})$ in Figure 3. The setup algorithm chooses n random keys for the key-homomorphic PRF and defines the aggregation key as the sum of the client keys. The encryption algorithm uses the key-homomorphic PRF to create a pseudorandom pad and adds it to the message modulo the public modulus R. The decryption algorithm sums together all ciphertexts which yields the sum of the client values plus the sum of the pseudorandom pads. Because the key of the aggregator is the sum of the client keys, the aggregator can compute the sum of the pseudorandom pads and subtract it from the ciphertexts' sum to obtain the sum of the plaintexts.

In Section 3.2, we show that this construction is aggregator oblivious under adaptive corruptions if the key-homomorphic PRF is indistinguishable from a random function. If the PRF is secure in the standard model, then our construction is also secure in the standard model.

```
$\mathsf{Setup}(1^\lambda, 1^n)$:
for $i \in [n]$: $k_i \leftarrow\$ \mathcal{K}$
$k_0 := \sum_{i\in[n]} k_i$
$\mathsf{pp} := R$ (the modulus)
return $(\mathsf{pp}, \{k_i\}_{i\in[n]_0})$

$\mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$:
$t_{i,l} := \mathsf{PRF}_{k_i}(l)$
$c_{i,l} := x_i + t_{i,l} \bmod R$
return $c_{i,l}$

$\mathsf{AggrDec}(\mathsf{pp}, l, \{c_{i,l}\}_{i\in[n]})$:
$t_{0,l} = \mathsf{PRF}_{k_0}(l)$
return $\sum_{i\in[n]} c_{i,l} - t_{0,l} \bmod R$
```

Fig. 3. The PSA scheme that uses key-homomorphic PRFs.

Next, we show that PSA is correct. Note that because PRF is key-homomorphic, we have $\sum_{i\in[n]} \mathsf{PRF}_{k_i}(l) = \mathsf{PRF}_{\sum_{i\in[n]} k_i}(l) = \mathsf{PRF}_{k_0}(l)$.

Correctness: Let $n, \lambda \in \mathbb{N}$, $(\mathsf{pp}, \{k_i\}_{i\in[n]_0}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, $l \in \mathcal{L}$, $x_i \in \mathbb{Z}_R$, $c_{i,l} \leftarrow \mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$. Then we have

$$\begin{aligned}
&\mathsf{AggrDec}(\mathsf{pp}, k_0, \{c_{i,l}\}_{i\in[n]}, l)\\
&= \textstyle\sum_{i\in[n]} c_{i,l} - \mathsf{PRF}_{k_0}(l) \bmod R\\
&= \textstyle\sum_{i\in[n]} (x_i + \mathsf{PRF}_{k_i}(l)) - \mathsf{PRF}_{k_0}(l) \bmod R\\
&= \textstyle\sum_{i\in[n]} x_i + \sum_{i\in[n]} \mathsf{PRF}_{k_i}(l) - \mathsf{PRF}_{k_0}(l) \bmod R\\
&= \textstyle\sum_{i\in[n]} x_i + \mathsf{PRF}_{\sum_{i\in[n]} k_i}(l) - \mathsf{PRF}_{k_0}(l) \bmod R\\
&= \textstyle\sum_{i\in[n]} x_i \bmod R
\end{aligned}$$

If $\sum_{i\in[n]} x_i < R$, then we get the sum over the integers $\sum_{i\in[n]} x_i$ as result.

3.2 Security Proof

In this section, we prove the aggregator obliviousness of the above scheme. For this, we follow the proof strategy of [1], who used this strategy to show the security of an inner-product MCFE scheme.[2]

In the proof of this theorem we show that PSA is aggregator oblivious, if the key-homomorphic PRF is indistinguishable from a random function.

Theorem 1. For any PPT adversary $\mathcal{A}$ on the aggregator obliviousness game, there is a PPT adversary $\mathcal{B}$ on the PRF such that

$$\begin{aligned}&\mathsf{Adv}^{\mathrm{AO}}_{\mathcal{A},\mathsf{PSA}}(\lambda)\\ &\leq 2(4n^2(n-1)+2n(n-1)+2n^2)\cdot\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda)\\ &\leq (8n^3+8n^2)\cdot\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda),\end{aligned}$$

where n is the number of users. The adversary $\mathcal{B}$ has roughly the same running time as $\mathcal{A}$.

Proof. We use four intermediate games to go from AO_0 to AO_1. A description of the games is depicted in Figure 4. We provide the lemmas for the transition between the games in Appendix A.

Game G_0: This is the AO_0 game, in which the challenge query is answered with encryptions of x_i^0.

Game G_1: This game still answers the challenge query with encryptions of x_i^0, but changes the pseudorandom pads that are used for the encryption. For correct decryption, these changes must not affect the sum of the pseudorandom pads. Therefore, to each pseudorandom pad, we add a share of a perfect η-out-of-η secret sharing of zero, where η is the number of users in the challenge query. This makes the ciphertexts useless, unless they are all summed together. This fact enables us to make the change of the next game.

Game G_2: This game answers the challenge query with encryptions of x_i^1 instead of x_i^0. This is possible because the secret shares of the previous game hide all information on the individual ciphertexts.

Game G_3: Here we remove the secret shares from the pseudorandom pads again. Therefore, this game is identical to AO_1, in which the challenge query is answered with encryptions of x_i^1.

```
$G_0$  $\boxed{G_1\ \ G_2}$  $G_3$
----------------------------------------------------------------
$(\mathsf{pp}, \{k_i\}_{i\in[n]_0}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{QChallenge}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp})$
Output $\alpha$ if condition $(*)$ (see p. 123) is satisfied, 0 otherwise

$\mathsf{QEnc}(i, x_i, l)$:
return $\mathsf{Enc}(\mathsf{pp}, k_i, l, x_i)$

$\mathsf{QCorrupt}(i)$:
return $k_i$

$\mathsf{QChallenge}(\mathcal{U}, \{x_i^0\}_{i\in\mathcal{U}}, \{x_i^1\}_{i\in\mathcal{U}}, l^*)$:
Let $\mathcal{U} = \{i_1, \ldots, i_\eta\} \subseteq [n]$
for all $\tau \in \{2, \ldots, \eta\}$: $u_\tau \leftarrow\$ \mathbb{Z}_R$
for all $i \in \mathcal{U}$
    $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*)$
    ┌───────────────────────────────────────────────────────────
    │ if $\eta \geq 2$, then:
    │     if $i = i_1$ then $t_{i,l^*} := t_{i,l^*} + \sum_{\tau=2}^{\eta} u_\tau$
    │     if $i = i_\tau$ for $\tau \in \{2, \ldots, \eta\}$ then $t_{i,l^*} := t_{i,l^*} - u_\tau$
    └───────────────────────────────────────────────────────────
    $w_i := x_i^0 + t_{i,l^*}$
    $w_i := x_i^1 + t_{i,l^*}$
return $\{w_i\}_{i\in\mathcal{U}}$
```

Fig. 4. Game hops of the proof of Theorem 1.

We distinguish two cases. In the *first case* the adversary $\mathcal{A}$ corrupts the aggregator. This is the more challenging case, because it allows the adversary to decrypt the sum of ciphertexts. Care must be taken to introduce the changes for the games in a way that cannot be recognized by the adversary. In Lemma 1, we use a hybrid argument over all users and several intermediate games to show that

$$\begin{aligned}&|\Pr[G_0(\lambda,\mathcal{A})=1]-\Pr[G_1(\lambda,\mathcal{A})=1]|\\ &\qquad\leq\ 2n^2(n-1)\cdot\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).\end{aligned}$$

To get from G_1 to G_2 we show in Lemma 2 that

$$\begin{aligned}&|\Pr[G_1(\lambda,\mathcal{A})=1]-\Pr[G_2(\lambda,\mathcal{A})=1]|\\ &\qquad\leq 2n(n-1)\cdot\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).\end{aligned}$$

[2] The definition of the games G_0 to G_3 is very similar to theirs. However, there are differences in the proof. Because in the game of aggregator obliviousness, the adversary is only allowed to ask one challenge query, the games do not have to guess the number of honest users. Also we need one game less than [1], because we do not have a layer of functional encryption. Because we use key-homomorphic PRFs instead of general PRFs, the transition from G_0 to G_1 is also different. Lastly, we directly prove aggregator obliviousness without relying on lemmas to upgrade the security. Doing so adds an extra case distinction to the proof, but reduces the reduction loss. This is possible, because PSA is a simpler primitive than MCFE.

Finally, to get from G_2 to G_3, we apply Lemma 3 in which we show that

$$|\Pr[G_2(\lambda,\mathcal{A})=1]-\Pr[G_3(\lambda,\mathcal{A})=1]| \\ \leq 2n^2(n-1)\cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).$$

In the *second case*, $\mathcal{A}$ does not corrupt the aggregator. This enables us to directly go from G_0 to G_3 by a hybrid argument over all users. Thus, in Lemma 4 we show that

$$|\Pr[G_0(\lambda,\mathcal{A})=1]-\Pr[G_3(\lambda,\mathcal{A})=1]| \\ \leq 2n^2\cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).$$

The reduction $\mathcal{B}$ uses an unbiased coin to decide whether to simulate case 1 or case 2, so in conclusion we get

$$\mathsf{Adv}^{\mathrm{AO}}_{\mathcal{A},\mathsf{PSA}}(\lambda) \\ \leq 2(4n^2(n-1)+2n(n-1)+2n^2)\cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda) \\ \leq (8n^3+8n^2)\cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda). \qquad \square$$

In this section, we proposed a PSA scheme that is based on key-homomorphic PRFs. We proved that the scheme is aggregator oblivious in the standard model. In the next section, we describe how to instantiate the scheme with a lattice-based PRF and explain our implementation.

4 Implementation

In this section, we describe the implementation of our scheme, the choice of parameters and performance results. The implementation in Go can be found here https://github.com/johanernst/khPRF-PSA. Both the encryption and the decryption algorithm are fast, so that they can also be executed on computationally limited devices such as smart-meters. The setup algorithm is slower, because for our parameters it needs to draw $\lambda = 2096$ random numbers per client. However it is only executed very rarely.

4.1 Choice of the Pseudorandom Function

For the implementation, we chose to use an *almost* key-homomorphic PRF mentioned in [12]. It relies on the LWR assumption and is secure in the random oracle model. Therefore, with this concrete instantiation our scheme is also only secure in the ROM. We chose a ROM-based PRF for its simplicity and efficiency. The standard model PRF of Boneh et al. [12] requires quite large parameters to be secure. The public parameters are two $\lambda' \times \lambda'$ matrices. Because the matrices are sampled from $\{0,1\}^{\lambda'\times\lambda'}$ instead of $\mathbb{Z}_q^{\lambda'\times\lambda'}$, the dimension of the matrices must be increased by a factor of $\log_2(q)$. Since we use $\lambda = 2096$ and $q = 2^{128}$, this means that $\lambda' = 2096\cdot 128$, thus the square matrix would have $\lambda'^2 \approx 7\cdot 10^{10}$ entries. Even if each entry is stored as single bit, these are 70 gigabits. Also, for evaluating the PRF, the matrices need to be multiplied together multiple times, whereby the intermediate entries which need to be kept in memory get much larger. Thus, this PRF does not seem practical because of both running time and memory constraints.

While the key-homomorphic PRF of Banerjee and Peikert [6] is more efficient, it is also more complex and thus, more prone to implementation errors, which endangers the security of the implementation. Kim proposes an approach that allows a smaller modulus q at the cost of larger keys [21]. Since we need $q > p = 2^{85}$ for a message space of 2^{64}, we would not gain very much from a smaller modulus. When our scheme is instantiated with any of the above mentioned standard model key-homomorphic PRFs, then we obtain a PSA scheme that is secure in the standard model.

Next, we describe the ROM-based key-homomorphic PRF of [12], which we use in the following. For $\lambda, q, p \in \mathbb{N}$, with $q > p$, $\mathbf{k} \in \mathbb{Z}_q^\lambda$ and a hash function $H\colon \mathcal{X} \mapsto \mathbb{Z}_q^\lambda$, the PRF is defined as:

$$F_{\mathbf{k}}(x) := \lfloor \langle H(x), \mathbf{k}\rangle \rfloor_p.$$

Because in [12] there is no security proof for this function, we provide a short proof in Appendix B. Because the output of F is from $\mathbb{Z}_p$, we set the public modulus R in our scheme equal to p.

The hash function H is required to map to $\mathbb{Z}_q^\lambda$. We construct such a hash function using a standard hash function H', such as SHA3, as follows:

$$H(x) := \begin{pmatrix} H'(\mathsf{byte}(x \,\|\, \texttt{" "} \,\|\, \texttt{"1"})) \bmod q \\ \vdots \\ H'(\mathsf{byte}(x \,\|\, \texttt{" "} \,\|\, \texttt{"}\lambda\texttt{"})) \bmod q \end{pmatrix}, \qquad (1)$$

where byte converts its argument to a byte array. The space between x and $i \in \{1,\dots,\lambda\}$ is necessary to ensure that all inputs to H' are different. Note that if q does not divide the size of the output space of H', extra analysis is needed to make sure that the mod q operation does not induce any bias. However in our case we choose q as power of 2, whereby it divides the size of the output space of H'. In our implementation we used SHA3-512 as H'.

The rounding function $\lfloor a \rceil$ is not exactly linear, but almost linear, which means that:

$$\lfloor a+b \rceil = \lfloor a \rceil + \lfloor b \rceil + e,$$

for $e \in \{0,1\}$. This entails that the PRF is only *almost* key-homomorphic:

$$\begin{aligned} F_{\mathbf{k}_1+\mathbf{k}_2}(x) &= \lfloor \langle H(x), \mathbf{k}_1 + \mathbf{k}_2 \rangle \rceil_p \\ &= \lfloor \langle H(x), \mathbf{k}_1 \rangle + \langle H(x), \mathbf{k}_2 \rangle \rceil_p \\ &= \lfloor \langle H(x), \mathbf{k}_1 \rangle \rceil_p + \lfloor \langle H(x), \mathbf{k}_2 \rangle \rceil_p + e \\ &= F_{\mathbf{k}_1}(x) + F_{\mathbf{k}_2}(x) + e \end{aligned}$$

for $e \in \{0,1\}$. For our use-case this is not a problem. Because in the decryption algorithm the PRF values of n clients are summed, the error from the non-linearity is at most $n-1$. The idea is to use a larger message space where all legitimate messages have a difference larger than n. The decrypted message that potentially contains an error of up to $n-1$ is then rounded to the next legitimate message. In Section 4.3, we describe this in more detail.

4.2 Choice of Parameters

In this section, we describe how we chose the parameters for the PRF and approximately which security level we get from these parameters.

The PRF is parameterized by $\lambda, q, p \in \mathbb{N}$ and reduces tightly to $\mathrm{LWR}_{\lambda,q,p}$. We used the LWE-estimator from [5] to estimate the security level for certain choices of parameters. The value of $1/p$ corresponds to the error rate α in LWE, so we need to choose $\lambda, q, p \in \mathbb{N}$ such that $\mathrm{LWE}_{\lambda,q,1/p}$ is hard. For $\lambda = 2096, q = 2^{128}$ and $p = 2^{85}$, the program estimates a hardness of over 2^{178}. Note that these parameters also satisfy the recommendation of [7] that $q/p > \sqrt{\lambda}$.

According to Theorem 1, the reduction loss is less than $8n^3+8n^2$. When we suppose we have $n = 2^{20}$ users, then the reduction loss is less than 2^{64}. This yields a security of $178 - 64 = 114$ bit.[3]

4.3 Concrete Instantiation

As described in the previous section, we set $\lambda := 2096$, $q := 2^{128}$ and $p := 2^{85}$. In our implementation, we set the public modulus $R = p = 2^{85}$ and $\mathsf{PRF} := F_{\mathbf{k}}(x) = \lfloor \langle H(x), \mathbf{k} \rangle \rceil_p$ with key space $\mathbb{Z}_q^\lambda$ and H as defined in (1). As labels we use strings that are converted to byte arrays before given to the hash function.

Next, we describe how to mitigate the error introduced by the non-linearity of the rounding function: Because $\sum_{i\in[n]} \mathbf{k}_i = \mathbf{k}_0$, we have

$$\sum_{i\in[n]} F_{\mathbf{k}_i}(l) + e \bmod R = F_{\mathbf{k}_0}(l),$$

for $e \in \{0, \ldots, n-1\}$. This means that

$$\sum_{i\in[n]} (x_i + F_{\mathbf{k}_i}(l)) - F_{\mathbf{k}_0}(l) \bmod R = \sum_{i\in[n]} x_i - e \bmod R.$$

To ensure correctness, each client, before calling $\mathsf{Enc}(\mathsf{pp}, \mathbf{k}_i, l, x_i')$, computes $x_i' := n \cdot x_i + 1$. The multiplication with n ensures that all legitimate messages are apart by $n-1$ and the addition of 1 ensures that the non-linearity error does not cause an underflow $\bmod R$. The aggregator, after executing $\bar{s} = \mathsf{AggrDec}(\mathsf{pp}, \mathbf{k}_0, l, \{c_i\}_{i\in[n]})$, rounds $\bar{s}$ up to the next multiple s' of n and computes $s := (s'-n)/n$.

Correctness: After encryption we have

$$c_i = n \cdot x_i + 1 + F_{\mathbf{k}_i}(l) \bmod R.$$

$\mathsf{AggrDec}(\mathsf{pp}, \mathbf{k}_0, l, \{c_i\}_{i\in[n]})$ yields

$$\begin{aligned} \bar{s} &= \textstyle\sum_{i\in[n]} (n \cdot x_i + 1 + F_{\mathbf{k}_i}(l)) - F_{\mathbf{k}_0}(l) \bmod R \\ &= \textstyle\sum_{i\in[n]} (n \cdot x_i + 1) - e \bmod R \\ &= n \cdot \textstyle\sum_{i\in[n]} x_i + n - e \bmod R \\ &= n \cdot \textstyle\sum_{i\in[n]} x_i + n - e. \end{aligned}$$

For the last equality to hold, we need that

$$0 \le n \cdot \textstyle\sum_{i\in[n]} x_i + n - e < R,$$

which means that e neither creates an underflow nor an overflow $\bmod R$. Because e is at most $n-1$, we have $0 < n \cdot \sum_{i\in[n]} x_i + n - e$ and if $\sum_{i\in[n]} x_i < (R-n)/n$, then we also have

$$n \cdot \sum_{i\in[n]} x_i + n - e < R.$$

When we suppose we have at most $n = 2^{20}$ users, this means that $\sum_{i\in[n]} x_i$ must be smaller than $(R - 2^{20})/2^{20} = 2^{85}/2^{20} - 1 = 2^{65} - 1$. In the next step, the aggregator rounds $\bar{s}$ up to the next multiple of n, which is $s' = n \cdot \sum_{i\in[n]} x_i + n$. After computing $s = (s'-n)/n$, they get $s = \sum_{i\in[n]} x_i$, which is the desired result. Therefore,

[3] For the 10000 users in our implementation the security is at least 132 bit

if the total sum is at most 2^{64}, the scheme works correctly.

Security: Because the clients input $n \cdot x_i + 1$ to the encryption algorithm of the PSA scheme, Theorem 1 guarantees that only $\sum_{i \in [n]}(n \cdot x_i + 1) \bmod R$ can be computed by the aggregator. Because n is known publicly, this value does not contain more information than $\sum_{i \in [n]} x_i$. Thus, our instantiation inherits the security of the general scheme.

4.4 Performance

In this section, we analyze the performance of our scheme both in theory and by performing running time measurements.

Every secret key consists of $\lambda = 2096$ elements from $\mathbb{Z}_q = \mathbb{Z}_{2^{128}}$, which means that every secret key needs $2096 \cdot 128 = 268288$ bits of memory. These are roughly 33.5 kilobyte. For a security level of 128 bit, Benhamouda et al. [10] report a key size of 592 bit for their scheme and of 416 bit for the scheme of Shi et al. [22]. The size of a ciphertext in our scheme is $\log(p) = 85$ bit. Again for a security level of 128 bit, Benhamouda et al. [10] report a ciphertext size of 296 bit for their PSA scheme and of 416 bit for the scheme of [22]. Both values were computed for 2^{20} users and 2^{20} labels. Neither Takeshita et al. [23] nor Waldner et al. [25] report their key sizes, however the key size of [25] increases linearly with the number of clients, because every client needs a shared secret with every other client. In many cases, smaller ciphertexts are preferable over small keys, because the ciphertexts have to be sent over the network at every time-step. In our scheme the cost for encryption mainly is the cost for evaluating the PRF. The PRF needs 2096 evaluations of the underlying hash function, 2096 modulo operations and 2096 additions and multiplications for evaluating the inner product of the hash and the key.

We executed the performance tests on a laptop on a single thread of an Intel Core i5-10210U CPU. We measured the running time of both our scheme and [25]. For a better comparison we executed their scheme with a message space of 2^{64} and without noise. We executed the tests 40 times and took the average. In every test we run the encryption algorithm once for every client and executed the decryption algorithm 1000 times. Figure 5 shows the average running time of a single execution of the encryption and decryption algorithm of both our scheme and [25]. As expected the running time for the encryption algorithm of our scheme does not depend on the number of users, whereas the running time of [25] grows linearly. Somewhat surprisingly the running time of our decryption algorithm increases with the number of users. This means that the running time is not completely dominated by the cost of evaluating the PRF, but summing together all ciphertexts also takes significant time. As the figure clearly shows, our scheme outperforms [25] starting from about 3500 clients. Table 2 shows the exact numbers of the average running time of our scheme and [25] for different numbers of users. Table 3 shows the running time of [23] and [9] taken from the respective papers.

In both, our scheme and [25], the evaluation of the PRF does not depend on the plaintext. Therefore, encryption could be sped up by computing the PRF beforehand. Then, when the plaintext is available, encryption only consists of adding the PRF output to the plaintext and one modulo operation. The same can be done for decryption as well.

	Our scheme		**LaSS (AES variant)**	
Users	Enc	Dec	Enc	Dec
1000	0.913(0.010)	0.875(0.002)	0.295(0.011)	0.277(0.007)
5000	0.929(0.007)	1.209(0.006)	1.590(0.022)	1.508(0.042)
10000	0.901(0.004)	1.805(0.007)	3.941(0.046)	3.643(0.201)

Table 2. Running time in milliseconds of one execution of the encryption/decryption algorithm of our scheme and the AES version of LaSS [25]. The value in parentheses is the standard deviation. For both schemes we executed 40 measurements and took the average.

	SLAP$_{\text{BGV}}$		**LaPS**	
Users	Enc	Dec	Enc	Dec
1000	1.17	3.26	3.724	1.964

Table 3. Running time in milliseconds of the optimized version of SLAP$_{\text{BGV}}$ [23] and of LaPS [9]. This version of LaPS only provides a security of 80 bit. For a security of 128 bit Becker et al. [9] report a running time of 77.33ms for encryption and 67.62 for decryption. Both implementations were measured with a message space of size 2^{16}. Note that the numbers are taken from the respective papers. Thus, a comparison is not entirely reliable.

The setup algorithm of our scheme has to draw 2096 random elements from $\mathbb{Z}_q$ for each user and then compute the sum of the user keys to get the key for the aggregator. As shown in Figure 6, the running time of our setup algorithm grows linearly in the number of users. The running time of [25] grows quadratically with the number of users, because every pair of users needs a shared key. The generation of the random numbers can trivially be parallelized and thereby be made much faster in practice. Also, since the setup algorithm is executed very rarely, its running time is not as critical as the running time of the encryption or decryption algorithm.

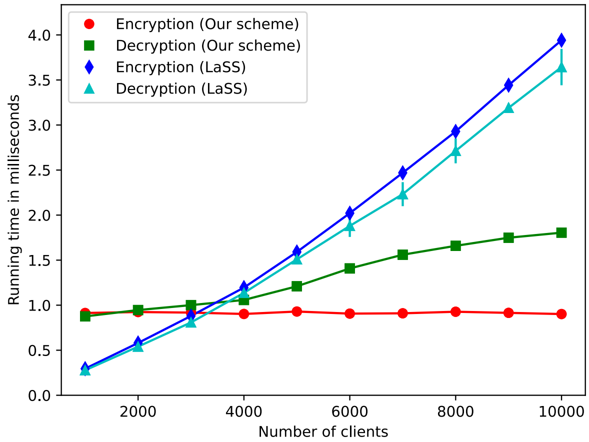

Fig. 5. This figure shows the running time of one execution of the encryption and decryption algorithm of both our scheme and [25]. The vertical bars show the standard deviation, which is large enough to be seen only for the decryption algorithm of LaSS.

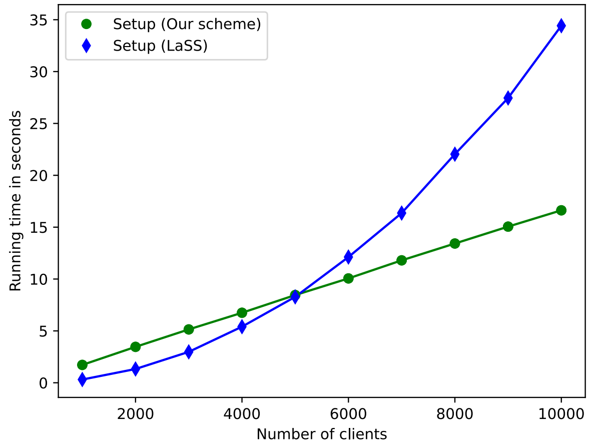

Fig. 6. This figure shows the running time of the setup algorithm of both our scheme and LaSS. As in Figure 5 the error bars are too small to be seen.

5 Deployment Considerations

In this section, we discuss practical issues when deploying our scheme, with a special focus on the smart-meter application.

5.1 Setup and Key Management

In the PSA literature the setup procedure is usually considered to be executed by a trusted party who distributes secret keys to the clients and the aggregator. This often means that the trusted party is able to decrypt all messages sent by the clients. In the following, we discuss techniques to overcome this limitation.

One approach to achieve a decentralized setup is to use techniques very similar to the ones of Chotard et al. [17]. The idea is that we let each client choose their PRF key k_i at random. The aggregator's key is supposed to be the sum of the keys chosen by the clients. So we only need to let the aggregator know the sum of the client keys in a secure way. The solution for this is to combine non-interactive key exchange (NIKE) ([13], [20]) with a technique from [15] as done in [17]. First the clients execute the NIKE, i.e. each client generates a public key and a secret key and uploads the public key to a key server. Each client downloads the public keys of all other clients and uses each other client's public key together with their own secret key to derive a shared secret with that client. Then the clients use the shared pairwise keys to generate random pads with the property that the pads sum to zero. The clients then add these pads to their PSA secret keys and send the resulting ciphertext to the aggregator. The aggregator will obtain the sum of the client keys by adding all ciphertexts, but learns nothing else about the keys. The process is essentially the same as the DSum functionality of [17], but without the layer of All-or-Nothing Encapsulation and is shown in Figure 7.

In principle, one can use the same key-homomorphic PRF as in our PSA scheme. However, the property of key-homomorphism is not needed here. Hence, we recommend using a block cipher such as AES as PRF.

One may now argue, that we do not need the rest of the PSA scheme anymore, because we already have a way of letting the aggregator know the sum of client values. Indeed this would basically give us the PSA scheme of [25]. However then the encryption of each value requires n invocations of the PRF, where n is the number of users. Doing this for every encryption becomes

```
$\mathsf{ComputeKeyShare}(k_i, \mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in [n]}, l)$:
$\{\mathsf{ss}_j\}_{j \in [n] \setminus \{i\}} := \mathsf{NIKE.SharedKey}(\mathsf{sk}_i, \mathsf{pk}_j)$
$t_i := \sum_{j \in [n] \setminus \{i\}} (-1)^{i<j} \cdot \mathsf{PRF}_{\mathsf{ss}_j}(\{\mathsf{pk}_k\}_{k \in [n]} \,\|\, l)$
return $(k_i + t_i) \bmod q$
```

Fig. 7. The decentralized setup algorithm as executed by every client. The algorithm takes as input the client's PSA secret key k_i, the NIKE secret key sk_i and the NIKE public keys of the other clients pk_j.

inefficient as the number of clients becomes large. So it is preferable to only execute this step once for the decentralized setup and then continue with our PSA scheme that only requires one invocation of the key-homomorphic PRF in the encryption algorithm.

For illustration, let us present a simple example of how a NIKE scheme can be built from the Diffie–Hellman key exchange. Every client i publishes their public key $\mathsf{pk}_i := g^{x_i}$ and downloads the public keys of all other clients. To compute a shared key with client j, client i computes $\mathsf{pk}_j^{x_i} = g^{x_j x_i}$ and hashes this together with their identities $H(i, j, g^{x_j x_i})$. For more details on constructions and security models see [13] and [20].

The most efficient way to distribute the public keys is to use a key server which all clients use to upload and download their public keys. The key server needs to be semi-honest, i.e. it can share all its knowledge with the adversary without compromising security, but must follow the protocol. A malicious key server could perform a person-in-the-middle attack by replacing all client public keys with its own public keys. The key server would then compute a shared secret with every client and would be able to decrypt all messages. However such an attack can be detected when clients manually compare their keys with each other.

The communication costs for the decentralized setup of one client is uploading their NIKE public-key to the key-server, downloading $n-1$ public-keys and sending one aggregation key share (the encrypted PSA secret key) to the aggregator. The computational costs are $n-1$ calls to $\mathsf{NIKE.SharedKey}$ to compute the pairwise shared keys and $n-1$ PRF or AES evaluations to compute the pseudorandom pad that encrypts the PSA secret key. These operations only have to be executed for the setup and have no influence on the cost for encryption, which is still independent of the number of clients.

An alternative to relying on a key server would be to let the clients broadcast their public key to all other clients. However, here a person-in-the-middle attack is also possible. Furthermore this approach would cause roughly n^2 messages in the setup phase, which can be too much in the smart-meter scenario, where the number of clients can be large.

Having a decentralized setup as described above has the additional advantage that the setup can be repeated at regular intervals to achieve some sort of forward secrecy. Repeating a centralized setup would mean that the trusted party would have to be available at each time the setup is repeated. Only relying on a semi-honest key server which needs to be online once every interval is much easier to assure.

Another approach from literature to decentralize the setup is adhoc multi-input functional encryption (adhoc-MIFE) [4]. In adhoc-MIFE the clients also generate shares of the aggregation key in a decentralized way, by getting as input the public keys of the other parties. The authors consider the computation of inner-products, which can be seen as a generalization of the computation of sums in our setting. They use 2-round MPC protocols with specific properties in their construction. However their construction does not support labels and is less efficient due to the use of MPC.

5.2 Client Failures

If one client fails to submit a valid ciphertext, then the aggregator cannot compute the overall sum, because that client's PRF term is missing. The result will then look like a random value. In the context of smart-metering this is a problem, because there are many devices and it is quite possible that one device fails due to technical problems, loss of network connection or active manipulation by the user. We have several options to cope with such failures.

A straightforward approach is to partition the users in several groups and run one instance of the PSA scheme for each group. For example, if we have 1000 users, we can divide them in ten groups of 100 users each. When one user fails, we only lose the values of 100 users, instead of 1000. Two disadvantages of this approach are that a few failing users can cause a lot of lost values and that it reduces the privacy of the users, because their values are only aggregated with a smaller set of other users.

To solve this problem, Chan et al. [14] have proposed a generic solution that incorporates differential privacy in an essential way. They let the aggregator learn partial sums of the users' inputs in such a way that the sum of the values of the non-failing users can always be reconstructed. They use a binary tree, where the clients are the leaf nodes. Each inner node corresponds to the partial sum of the values of the clients beneath that node. So each client produces $\log(n)$ ciphertexts of which each one corresponds to an inner node on the path from that client to the root of the tree. When all clients send ciphertexts, then the aggregator uses the ciphertexts corresponding to the root node to compute the total sum. If some clients fail, the aggregator has to use ciphertexts corresponding to the inner nodes to be able to reconstruct the sum of the remaining users as shown in Figure 8. The noise for differential privacy is essential here, because otherwise the aggregator would get all the users' values in the clear.

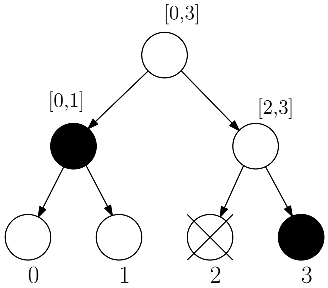

Fig. 8. If client 2 fails, the aggregator uses the ciphertexts corresponding to the black nodes to compute the sum of the remaining clients' values. The notation [i,j] means the (noisy) sum of the values of clients {i,. . . ,j}. This figure is a smaller version of Figure 1b from [14].

With this approach each client has $\log(n)$ secret keys, one corresponding to each node on the path to the root. Also, instead of sending one message, each client sends $\log(n)$ messages. The aggregator holds one aggregation key for each inner node in the tree. Another way to view this is that we are running one instance of the PSA scheme for each inner node. This approach does not add additional rounds of communication. The aggregator will always be able to compute the overall sum, no matter how many clients fail, however, when more clients fail, the resulting sum is noisier. The scheme is generic and does not pose any special requirements on the underlying PSA scheme, so it can be used to make our scheme failure tolerant. Only one small adaption has to be made. The scheme of Chan et al. only considers user values in $\{0, 1\}$, whereas in our case the values are from $\{0, \dots, \Delta\}$, where Δ is the largest reasonable power consumption. To accommodate for this we need to multiply the ϵ and δ parameters for differential privacy with $1/\Delta$. In Appendix C we describe in a bit more detail how the adapted encryption algorithm works and also provide figures with pseudocode.

5.3 Client Joins and Leaves

If clients leave the system, for example if they changed their power supplier, we can treat them as permanently failed (cf. Section 5.2), as suggested by [14]. This will slightly increase the noise relative to the number of remaining clients. Therefore, when many clients have left, we can repeat the setup for the remaining clients and start over with a new tree. Repeating the setup is more practical when using the decentralized setup described in Section 5.1, because then we do not need to entrust a third party with the key generation.

To accommodate for joining clients, [14] propose to create a tree that has more leaves than there are clients, where the trusted party creates secret keys for every leave node. The not-yet-present clients are treated as failed, until they join. When a new client joins, they get the secret keys corresponding to their leave node from the trusted party. This has the advantage that neither the other clients nor the aggregator need to be notified when a client joins. However it has the disadvantage that the trusted party needs to be available whenever a new client joins.

In the following we describe how client-joins can work together with the decentralized setup from Section 5.1. Since we are using a binary tree now, we are essentially running one instance of the PSA scheme for every inner node, where the clients of each instance are the clients below the respective inner node. This means that in the beginning, each client creates $\log(n)$ aggregation key shares and sends them to the aggregator.

Whenever a new client joins they broadcast their public key to the other clients by uploading it to the key server and downloads the public keys of the other clients and computes a shared key with each of them. The other clients download the public key of the new client and use it to compute a shared secret with the new client. Then a new aggregation key is created for each node on the path from the new client to the root. This means that each client, which shares an inner node with the new client, chooses a new secret key for the respective node and sends the aggregation key share to the aggregator. The aggregator receives all the aggregation key shares for each node on the path from the new client to the root and combines them to obtain a completely new aggregation key for all these $\log(n)$ nodes.

Another way to view this is that whenever a new client joins, (a part of) the decentralized setup is repeated for the inner nodes on the path from the new client to the root.

The cost for the new $(n+1)$th client is computing a shared secret with the n other clients and computing aggregation key shares for each node on the path to the root. The number of PRF/AES evaluations depends on the position of the new node in the tree, but is at least n (for the aggregation key associated with the root node) and at most $2n$. The other clients have to compute one new shared secret. The number of PRF/AES evaluations depends on their position in relation to the new node, but is again at least n and at most $2n$.

5.4 Concrete Smart-Meter Example

In this section we summarize how our PSA scheme together with the above extensions can be used for privacy preserving smart-metering. Every smart-meter is preconfigured with the IP addresses of the key server and the power-supplier, the public key of the power-supplier and with an upper bound on the number of other smart-meter devices that are expected to join the system[4]. Every smart-meter creates a random key for the PSA scheme and a public and secret key for the NIKE scheme. The smart-meters then upload their public keys to the key-server and compute pairwise shared keys as described in Section 5.1. They then compute their key share as described in Figure 7, encrypt it with the power-suppliers public key and send the resulting ciphertext to the power-supplier. The power-supplier is then able to compute the aggregation key for each inner node. This concludes the setup.

At every time step (e.g. every fifteen minutes) the smart-meters encrypt their current power consumption and send it to the power-supplier. There are two straightforward ways of choosing the label which is needed as input for the PRF. One possibility is to use a counter that starts with 0 and is incremented at every time step. This has the disadvantage that if a client malfunctions and misses a time step, its counter becomes asynchronous to the counters of the other smart-meters, whereby its ciphertexts cannot be decrypted anymore. The other possibility provides a solution for this. The label can be chosen as the current date and hour and a value between 0 and 3 which indicates the quarter of the hour we are currently in. Then every smart-meter can deduce the next label from the current time. The clocks of the smart-meters only need to have the same time up to fifteen minutes precision and the current time can be easily obtained via the network. Note that this ensures that every label is only used once, whereby the encrypt-once condition is satisfied and thus, our security proof applies.

To accommodate for users joining the system, every smart-meter checks the key server for new public keys at regular intervals such as once every day. If there are new public keys, the clients recompute some of their key-shares as described in Section 5.3 and send them to the aggregator. Failing or leaving smart-meters are also treated as described in Section 5.3. Since we use the generic approach of [14] to provide fault-tolerance, the resulting power-consumption, which is decrypted by the power-supplier, contains a small amount of noise. However as stated by [14] the additive error is only slightly larger than logarithmic in the number of users.

The smart-meters can also be configured to repeat the setup at regular intervals to provide better forward secrecy or to reduce the noise, when too many smart-meters have left the system.

6 Conclusion

PSA is a useful protocol for privately letting an aggregator know the sum of client-supplied values. It has applications from privacy-preserving smart metering to distributed machine learning.

In this paper, we proposed a PSA scheme that is based on key-homomorphic PRFs as the only building block. It supports a large message space and scales well for a large number of users. Also it has very small ciphertexts (85 bit in our implementation with a message space of 2^{64}). Both encryption and decryption mostly consist of only one PRF evaluation.

We proved the security of our scheme in the standard model. Furthermore we implemented our scheme using a lattice-based key-homomorphic PRF in the ROM. We analyzed the performance both in theory and by measuring the running time in practice and thereby showed that the scheme is quite efficient. Moreover we discussed possible solutions for practical issues as how to decentralize the setup and how to accommodate for clients joining or leaving the system.

[4] When more users than expected join the system, then the (non-interactive) setup can simply be repeated

Acknowledgments

We would like to thank the anonymous reviewers. Alexander Koch was supported by the Competence Center for Applied Security Technology (KASTEL).

References

[1] M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Proceedings, Part III*, volume 11923 of *LNCS*, pages 552–582. Springer, 2019. 10.1007/978-3-030-34618-8_19.

[2] M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In D. Lin and K. Sako, editors, *Public-Key Cryptography, PKC 2019, Proceedings, Part II*, volume 11443 of *LNCS*, pages 128–157. Springer, 2019. 10.1007/978-3-030-17259-6_5.

[3] G. Ács and C. Castelluccia. I have a DREAM! (DiffeRentially privatE smArt Metering). In T. Filler, T. Pevný, S. Craver, and A. D. Ker, editors, *Information Hiding, IH 2011*, volume 6958 of *LNCS*, pages 118–132. Springer, 2011. 10.1007/978-3-642-24178-9_9.

[4] S. Agrawal, M. Clear, O. Frieder, S. Garg, A. O'Neill, and J. Thaler. Ad hoc multi-input functional encryption. In T. Vidick, editor, *Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 40:1–40:41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. 10.4230/LIPIcs.ITCS.2020.40.

[5] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. 10.1515/jmc-2015-0016.

[6] A. Banerjee and C. Peikert. New and improved key-homomorphic pseudorandom functions. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Proceedings, Part I*, volume 8616 of *LNCS*, pages 353–370. Springer, 2014. 10.1007/978-3-662-44371-2_20.

[7] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012, Proceedings*, volume 7237 of *LNCS*, pages 719–737. Springer, 2012. 10.1007/978-3-642-29011-4_42.

[8] D. Becker and J. G. Merchan. Post-quantum secure private stream aggregation, Apr. 21 2020. URL https://patents.google.com/patent/US10630655B2/en. US Patent 10,630,655.

[9] D. Becker, J. Guajardo, and K.-H. Zimmermann. Revisiting private stream aggregation: Lattice-based PSA. In *NDSS 2018*. The Internet Society, 2018. URL https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02B-3_Becker_paper.pdf.

[10] F. Benhamouda, M. Joye, and B. Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Transactions on Information and System Security (TISSEC)*, 18(3):10:1–10:21, 2016. 10.1145/2873069.

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1175–1191. ACM, 2017. 10.1145/3133956.3133982.

[12] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Proceedings, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, 2013. 10.1007/978-3-642-40041-4_23.

[13] D. Cash, E. Kiltz, and V. Shoup. The Twin Diffie-Hellman problem and applications. In N. P. Smart, editor, *EUROCRYPT 2008, Proceedings*, volume 4965 of *LNCS*, pages 127–145. Springer, 2008. 10.1007/978-3-540-78967-3_8.

[14] T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In A. D. Keromytis, editor, *Financial Cryptography and Data Security, FC 2012*, volume 7397 of *LNCS*, pages 200–214. Springer, 2012. 10.1007/978-3-642-32946-3_15.

[15] M. Chase and S. S. Chow. Improving privacy and security in multi-authority attribute-based encryption. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security, CCS 2009*, pages 121–130. ACM, 2009. 10.1145/1653662.1653678.

[16] J. Chotard, E. D. Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In T. Peyrin and S. D. Galbraith, editors, *ASIACRYPT 2018, Proceedings, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, 2018. 10.1007/978-3-030-03329-3_24.

[17] J. Chotard, E. Dufour-Sans, R. Gay, D. H. Phan, and D. Pointcheval. Dynamic decentralized functional encryption. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Proceedings, Part I*, volume 12170 of *LNCS*, pages 747–775. Springer, 2020. 10.1007/978-3-030-56784-2_25.

[18] K. Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In J. Pieprzyk and S. Suriadi, editors, *Australasian Conference on Information Security and Privacy, ACISP 2017, Proceedings, Part II*, volume 10343 of *LNCS*, pages 193–213. Springer, 2017. 10.1007/978-3-319-59870-3_11.

[19] J. Ernst and A. Koch. Efficient private stream aggregation with labels in the standard model. In S.-L. Gazdag, D. Loebenberger, and M. Nüsken, editors, *crypto day matters 32*, Bonn, 2021. Gesellschaft für Informatik e.V. / FG KRYPTO. 10.18420/cdm-2021-32-16.

[20] E. S. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography, PKC 2013, Proceedings*, volume 7778 of *LNCS*, pages 254–271. Springer, 2013. 10.1007/978-3-642-36362-7_17.

[21] S. Kim. Key-homomorphic pseudorandom functions from LWE with small modulus. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Proceedings, Part II*, volume 12106 of *LNCS*, pages 576–607. Springer, 2020. 10.1007/978-3-030-45724-2_20.

[22] E. Shi, T. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*. The Internet Society, 2011. URL https://www.ndss-symposium.org/ndss2011/privacy-preserving-aggregation-of-

time-series-data.

[23] J. Takeshita, R. Karl, T. Gong, and T. Jung. SLAP: Simple lattice-based private stream aggregation protocol. *Cryptology ePrint Archive*, Report 2020/1611, 2020. URL https://eprint.iacr.org/2020/1611/20210513:151621.

[24] F. Valovich. Aggregation of time-series data under differential privacy. In T. Lange and O. Dunkelman, editors, *LATINCRYPT 2017, Revised Selected Papers*, volume 11368 of *LNCS*, pages 249–270. Springer, 2017. 10.1007/978-3-030-25283-0_14.

[25] H. Waldner, T. Marc, M. Stopar, and M. Abdalla. Private stream aggregation from labeled secret sharing schemes. *Cryptology ePrint Archive*, Report 2020/81, 2021. URL https://eprint.iacr.org/2020/081.

A Lemmas for the Security Proof of the PSA Scheme

Lemma 1. (Transition from G_0 to G_1): For every PPT adversary $\mathcal{A}$, which corrupts the aggregator, there is a PPT adversary $\mathcal{B}$, on the PRF with

$$|\Pr[G_0(\lambda, \mathcal{A}) = 1] - \Pr[G_1(\lambda, \mathcal{A}) = 1]| \leq 2n^2(n-1) \cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).$$

Proof. We prove the transition by a hybrid argument. As in the proof of [1], the goal in each hybrid step is to add a random pad to the PRF value of one more user. Each hybrid step consists of three game hops as depicted in Figure 9.

We define hybrid games $G_{0,\mu}$ for $\mu \in [n]$. Let $\eta = |\mathcal{U}|$ and $\mathcal{U} = \{i_1, \ldots, i_\eta\}$ be the set of users for which $\mathcal{A}$ asks the challenge query. Let $\Theta = \min(\eta, \mu)$. If $\Theta \geq 2$, in hybrid step μ, the game adds random pads to the PRF values of the first Θ users in $\mathcal{U}$. The condition that $\Theta \geq 2$ is necessary, because we need two users in $\mathcal{U}$ to change the PRF to a RF, because we must not change the overall sum of the ciphertexts. The pads are set up such that they are a perfect Θ-out-of-Θ secret sharing of 0. This makes sure that the pads have no effect on the sum of the ciphertexts. If $\mu > \eta$, then there are already random pads on the PRF values of all users from $\mathcal{U}$, therefore, the subsequent games are the same. We have that $G_0 = G_{0,1}$ and $G_{0,n} = G_1$. In the following we will shortly describe the different games of the transition from $G_{0,\mu-1}$ to $G_{0,\mu}$:

Game $G_{0,\mu-1}$: This is step $\mu - 1$ of the hybrid argument between G_0 and G_1. Let $\Theta = \min(\eta, \mu)$. If $\Theta \geq 2$, then in this game there are random pads added to the PRF-values of the first $\Theta - 1$ honest users.

```
$G_{0,\mu-1}$   $G'_{0,\mu-1}$   $G''_{0,\mu-1}$   $G_{0,\mu}$

//Note that the game can guess $i^*_\Theta$, without yet
// knowing the value of $\Theta$.
$i^*_1 \leftarrow\$ [n]$, $i^*_\Theta \leftarrow\$ [n] \setminus \{i^*_1\}$
$(\mathsf{pp}, \{k_i\}_{i\in[n]_0}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{QChallenge}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp})$
Output $\alpha$ if condition $(*)$ is satisfied AND
  the game was not aborted; 0 otherwise

$\mathsf{QEnc}(i, x_i, l)$:
$t_{i,l} := \mathsf{PRF}_{k_i}(l)$
if $i = i^*_1$ then $t_{i,l} := \mathsf{RF}(l)$
if $i = i^*_\Theta$ then $t_{i,l} := -\sum_{j\in[n]\setminus\{i\}} t_{j,l}$
return $x_i + t_{i,l}$

$\mathsf{QCorrupt}(i)$:
return $k_i$

$\mathsf{QChallenge}(\mathcal{U}, \{x^0_i\}_{i\in\mathcal{U}}, \{x^1_i\}_{i\in\mathcal{U}}, l^*)$:
Let $\mathcal{U} = \{i_1, \ldots, i_\eta\}$
$\Theta := \min(\eta, \mu)$
for all $\tau \in \{2, \ldots, \Theta\}$: $u_\tau \leftarrow\$ \mathbb{Z}_R$
for all $i \in \mathcal{U}$
    $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*)$
    if $\Theta \geq 2$ then
        if $i_1 \neq i^*_1$ or $i_\Theta \neq i^*_\Theta$ then abort game
        if $\mu \leq \eta$ then
            if $i = i_1$ then $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*) + \sum_{\tau=2}^{\Theta-1} u_\tau$
            if $i = i_\tau$ for $\tau \in \{2, \ldots, \Theta - 1\}$ then:
                $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*) - u_\tau$
            if $i = i_1$ then $t_{i,l} := \mathsf{RF}(l^*) + \sum_{\tau=2}^{\Theta-1} u_\tau$
            if $i = i_\Theta$ then $t_{i,l} := -\sum_{j\in[n]\setminus\{i\}} t_{j,l}$
            if $i = i_1$ then $t_{i,l^*} := t_{i,l^*} + u_\Theta$
            if $i = i_\Theta$ then $t_{i,l^*} := t_{i,l^*} - u_\Theta$
        else
            if $i = i_1$ then $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*) + \sum_{\tau=2}^{\Theta} u_\tau$
            if $i = i_\tau$ for $\tau \in \{2, \ldots, \Theta\}$ then:
                $t_{i,l^*} := \mathsf{PRF}_{k_i}(l^*) - u_\tau$
return $\{x^0_i + t_{i,l^*}\}_{i\in\mathcal{U}}$
```

Fig. 9. Game hops for one step of the hybrid argument

Game $G'_{0,\mu-1}$: In this game, the PRF of user i_1 is replaced by a RF. The pseudorandom pad $t_{i_\Theta,l}$ of user i_Θ is set such that $\sum_{i\in[n]} t_{i,l} = t_{0,l}$ still holds. The game has to guess the first and Θ'th user of $\mathcal{U}$ before seeing the challenge query, because it must be able to answer encryption queries before seeing the challenge query.

Game $G''_{0,\mu-1}$: Here we still have the RF of the previous game. When answering the challenge query, the game adds a random pad u_Θ to the RF. Now, the first Θ honest users have a random pad added to their PRF/RF values.

Game $G_{0,\mu}$: This game again uses a PRF instead of a RF for honest users i_1 and i_Θ. Because of the changes in the previous games, there are random pads added to the PRF value of the first Θ users of $\mathcal{U}$, so this is the μ'th hybrid game.

Next we give the reductions for the transitions between the games.

Transition from $G_{0,\mu-1}$ **to** $G'_{0,\mu-1}$: The difficulty in this step is that $\sum_{i\in[n]} \mathsf{PRF}_{k_i}(l^*) = \mathsf{PRF}_{k_0}(l^*)$. Replacing one PRF with a RF while holding the other keys fixed would violate this property. This is the reason why in each hybrid step of the transition from G_0 to G_1 the game guesses *two* honest users. The pad of one user is determined by the answer of the PRF challenger and the pad of the other user is set, such that the pads still sum to $\mathsf{PRF}_{k_0}(l^*)$.

Now we show the indistinguishability of $G_{0,\mu-1}$ and $G'_{0,\mu-1}$ by a reduction to the security of the PRF. We assume that there is an adversary $\mathcal{A}$ which can distinguish $G_{0,\mu-1}$ and $G'_{0,\mu-1}$. The reduction $\mathcal{B}$ guesses the honest users i_1^* and i_Θ^*. They generate the secret keys $\{k_i\}_i \in [n]_0 \setminus \{i_1^*, i_\Theta^*\}$ for all users and the aggregator except of users i_1^* and i_Θ^*. They send the public parameters pp to $\mathcal{A}$ and answer the queries as follows:

$\underline{\mathsf{QCorrupt}(i)}$: If the guess of i_1^* and i_Θ^* was correct, these two users stay uncorrupted. Since $\mathcal{B}$ generated the keys of all the other clients, they can simply answer with the corresponding secret key. If $i = 0$, then $\mathcal{B}$ returns the aggregation key k_0. Note that here k_0 is not the sum of the client keys, but chosen randomly as well. The pads of i_1^* and i_Θ^* will be chosen accordingly such that $\sum_{i\in[n]} t_{i,l} = t_{0,l}$ still holds.

$\underline{\mathsf{QEnc}(i, x_i, l)}$: If $i = i_1^*$ or $i = i_\Theta^*$, $\mathcal{B}$ queries l to their own PRF challenger and receives a_l, which is either $\mathsf{PRF}_{k'}(l)$, for some k', or $\mathsf{RF}(l)$. They set $t_{i_1^*,l} := a_l$ and $t_{i_\Theta^*,l} := \mathsf{PRF}_{k_0} - \sum_{j\in[n]\setminus\{i_1^*,i_\Theta^*\}} \mathsf{PRF}_{k_j}(l) - a_l = t_{0,l} - \sum_{j\in[n]\setminus\{i_\Theta^*\}} t_{j,l}$. This ensures that all $t_{i,l}$ still sum to $t_{0,l}$. Note that $t_{i_\Theta^*,l} = t_{0,l} - \sum_{j\in[n]\setminus\{i_\Theta^*\}} t_{j,l}$ also holds in the unmodified scheme (Figure 3) due to the key-homomorphism of the PRF. If $i = i_1^*$, $\mathcal{B}$ sends $x_i + t_{i_1^*,l}$ to $\mathcal{A}$. If $i = i_\Theta^*$, $\mathcal{B}$ sends $x_i + t_{i_\Theta^*,l}$ to $\mathcal{A}$ and stores $t_{i_1^*,l}$ until $\mathcal{A}$ asks an encryption query for i_1^* and label l. For the other clients $\mathcal{B}$ knows the corresponding secret keys and can, therefore, answer the queries without asking their PRF challenger.

$\underline{\mathsf{QChallenge}(\mathcal{U}, \{x_i^0\}_{i\in\mathcal{U}}, \{x_i^1\}_{i\in\mathcal{U}}, l^*)}$: Here $\mathcal{B}$ encrypts x_i^0 the same way as in the QEnc queries.

If the PRF challenger uses $\mathsf{PRF}_{k'}$ instead of a RF then $k_{i_1^*}$ is implicitly set to k' and $k_{i_\Theta^*}$ is set such that $\sum_{i\in[n]} k_i = k_0$. In that case the k_i are a perfect secret sharing of k_0, which is exactly as in $G_{0,\mu-1}$. So in that case, $\mathcal{B}$ perfectly simulates $G_{0,\mu-1}$.

If the PRF challenger uses a RF, then $t_{i_1^*,l} = \mathsf{RF}(l)$ and $t_{i_\mu^*}$ is set such that all $t_{i,l}$ sum to $t_{0,l}$. So in this case $\mathcal{B}$ perfectly simulates game $G'_{0,\mu-1}$.

Transition from $G'_{0,\mu-1}$ **to** $G''_{0,\mu-1}$ In this step the goal is to add random pads u_Θ to the RF of users i_1 and i_Θ in the answers of the challenge query. Because we consider encrypt-once security, $\mathcal{A}$ cannot ask an encryption query for user i_1 or i_Θ on label l^*. Therefore, the only information that $\mathcal{A}$ has about $\mathsf{RF}(l^*)$, comes from the answer to the challenge query. Since $\mathsf{RF}(l^*)$ is identically distributed as $\mathsf{RF}(l^*) + u_\Theta$, $\mathcal{A}$ cannot realize that they received $\mathsf{RF}(l^*) + u_\Theta$ instead of $\mathsf{RF}(l^*)$. Thus, $G'_{0,\mu-1}$ and $G''_{0,\mu-1}$ are perfectly indistinguishable.

Transition from $G''_{0,\mu-1}$ **to** $G_{0,\mu}$ In this step we need to change back the RF of users i_1 and i_Θ to a PRF. This works exactly as the transition from $G_{0,\mu-1}$ to $G'_{0,\mu-1}$.

After $\eta - 1$ of these steps we reached Game G_1. Now in G_1 the challenge query $(\mathcal{U}, \{x_i^0\}_{i\in U}, \{x_i^1\}_{i\in\mathcal{U}}, l^*)$ is answered with $x_i^0 + t_{i,l^*} + \bar{u}_i$, where

$$\bar{u}_i = \begin{cases} \sum_{j\in\mathcal{U}\setminus\{i_1\}} u_j & \text{if } i = i_1, \\ -u_i & \text{if } i \in \mathcal{U} \setminus \{i_1\}, \\ 0 & \text{else.} \end{cases}$$

Therefore, the $\{\bar{u}_i\}_{i\in\mathcal{U}}$ are a perfect η out of η secret sharing of 0.

The guessing of the two honest users in each hybrid step incurs a reduction loss of $n(n-1)$ and using n hybrid games leads to a loss of $n^2(n-1)$ for the hybrid argument. Since the hybrid argument is applied twice, once to add and once to remove the random pads, we get a total reduction loss of $2n^2(n-1)$. □

Lemma 2. (Transition from G_1 to G_2): For every PPT adversary $\mathcal{A}$, which corrupts the aggregator, there is a

PPT adversary $\mathcal{B}$ on the PRF with

$$|\Pr[G_1(\lambda,\mathcal{A})=1]-\Pr[G_2(\lambda,\mathcal{A})=1]| \leq 2n(n-1)\cdot \mathsf{Adv}^{\text{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).$$

Proof. The goal in this step is to change the answer of the challenge query from encryptions of x_i^0 to encryptions of x_i^1. We distinguish two cases here. Remember that $\mathcal{Q}_{l^*}$ is the set of clients for which $\mathcal{A}$ has received a ciphertext on label l^*, either by an encryption or a challenge query. In the first case $\mathcal{Q}_{l^*}=\mathcal{HS}$, which means that $\mathcal{A}$ gets a ciphertext of every honest user for the challenge label l^*. Then we have $\mathcal{Q}_{l^*}\cup\mathcal{CS}=[n]$, whereby $\mathcal{A}$'s challenge query is restricted by the balance-condition. We then argue that, since $\sum x_i^0=\sum x_i^1$, the change is covered by the $\bar{u}_i$.

In the second case there is an honest user of whom $\mathcal{A}$ does not get a ciphertext on label l^*. Therefore, the challenge messages are not restricted by the balance condition. Here we use the fact that $\mathcal{A}$ is lacking a ciphertext of an honest user i_q and thereby has no information about $\mathsf{PRF}_{k_{i_q}}(l^*)$.

Case 1 ($\mathcal{Q}_{l^*}=\mathcal{HS}$): In this case $\mathcal{A}$ knows k_0, because they corrupted the aggregator. Furthermore $\mathcal{Q}_{l^*}=\mathcal{HS}$, whereby $\mathcal{A}$'s challenge messages are restricted by the balance-condition. We argue as the authors in [1]. For $t_{0,l}:=\mathsf{PRF}_{k_0}(l)$, we have $\sum t_{i,l^*}=t_{0,l^*}$ and that $\{\bar{u}_i\}_{i\in\mathcal{U}}$ is a perfect η out of η secret sharing of 0. Therefore, $\{x_i^0+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ and $\{x_i^1+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ are perfect secret sharings of $\sum_{i\in\mathcal{U}}(x_i^0+t_{i,l^*})$ and $\sum_{i\in\mathcal{U}}(x_i^1+t_{i,l^*})$, respectively. The balance-condition requires that $\sum_{i\in\mathcal{U}}x_i^0=\sum_{i\in\mathcal{U}}x_i^1$. So $\{x_i^0+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ and $\{x_i^1+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ are both perfect secret sharings of the same value and are, therefore, perfectly indistinguishable.

Case 2 ($\mathcal{Q}_{l^*}\neq\mathcal{HS}$): Other than in case 1, $\mathcal{A}$'s messages in the challenge query are *not* restricted by the balance-condition, because there is at least one honest user for which $\mathcal{A}$ has no ciphertext on label l^*. Therefore, in this case we need another reduction. If $\mathcal{A}$ asks no challenge query or a challenge query with $\mathcal{U}=\{\}$, $\mathcal{A}$ has no information about the challenge bit and, therefore, $\mathcal{A}$'s advantage is 0. Thus, we can assume that $\mathcal{U}$ contains at least one user. Additionally, in this case $\mathcal{Q}_{l^*}\neq\mathcal{HS}$, therefore, $\mathcal{HS}\setminus\mathcal{Q}_{l^*}$ contains an honest user that is different from the user in $\mathcal{U}$. The reduction $\mathcal{B}$ guesses the users $i_q^*\in\mathcal{HS}\setminus Q_{l^*}$ and $i_u^*\in\mathcal{U}$. Then $\mathcal{B}$ changes the PRF of these two users to a RF by setting $t_{i_u^*,l}:=\mathsf{RF}(l)$ and $t_{i_q^*,l}:=t_{0,l}-\sum_{i\in[n]\setminus i_{q^*}}t_{i,l}$. This is done the same way as in the transition from $G_{0,\mu-1}$ to $G'_{0,\mu-1}$.

In the next step we change all ciphertexts in the challenge query from $x_i^0+t_{i,l^*}$ to $x_i^1+t_{i,l^*}$. Because of the $\bar{u}_i$, $\{x_i^0+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ and $\{x_i^1+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ are perfect η out of η secret sharings of $\sum_{i\in\mathcal{U}}(x_i^0+t_{i,l^*})$ and $\sum_{i\in\mathcal{U}}(x_i^1+t_{i,l^*})$ respectively. For both $b=0$ and $b=1$ we have $\sum_{i\in\mathcal{U}}(x_i^b+t_{i,l^*})=\sum_{i\in\mathcal{U}\setminus\{i_u^*\}}(x_i^b+t_{i,l})+x_{i_u^*}+t_{i_u^*,l^*}$. Because $t_{i_u^*,l^*}=\mathsf{RF}(l^*)$, both $\{x_i^0+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ and $\{x_i^1+t_{i,l^*}+\bar{u}_i\}_{i\in\mathcal{U}}$ are secret sharings of a truly random value and are, therefore, perfectly indistinguishable.

In the last step we change back the RF to a PRF again.

The reduction loss of $n(n-1)$ comes from guessing the two users i_u^* and i_q^*. The factor of two is there, because the RF needs to be changed back into a PRF. Therefore, the total reduction loss is $2n(n-1)$. □

Lemma 3. (Transition from G_2 to G_3): For every PPT adversary $\mathcal{A}$, that corrupts the aggregator, there is a PPT adversary $\mathcal{B}$ on the PRF with

$$|\Pr[G_2(\lambda,\mathcal{A})=1]-\Pr[G_3(\lambda,\mathcal{A})=1]| \leq 2n^2(n-1)\cdot \mathsf{Adv}^{\text{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda)$$

Proof. This transition is just applying the G_0-G_1 transition backwards. □

Lemma 4. For every PPT adversary $\mathcal{A}$, which does *not* corrupt the aggregator, there is a PPT adversary $\mathcal{B}$ on the PRF with

$$|\Pr[G_0(\lambda,\mathcal{A})=1]-\Pr[G_3(\lambda,\mathcal{A})=1]| \leq 2n^2\cdot \mathsf{Adv}^{\text{prf}}_{\mathcal{B},\mathsf{PRF}}(\lambda).$$

Proof. In this case we can directly go from G_0 to G_3 via a hybrid argument over all users. Let $\{i_1,\dots,i_\eta\}=\mathcal{U}$ be the set of users specified in the challenge query. In hybrid game H_μ the challenge query for $(i_1,\dots,i_\mu)$ is answered with an encryption of x_i^1, whereas for the other users it is answered with an encryption of x_i^0.

Formally, in H_μ we have:

```
$\underline{\mathsf{QChallenge}(\mathcal{U}=\{i_1,\dots,i_\eta\},\{x_i^0\}_{i\in\mathcal{U}},\{x_i^1\}_{i\in\mathcal{U}},l^*):}$
$c_{i,l^*} := \begin{cases}\mathsf{Enc}(\mathsf{pp},k_i,x_i^0) & \text{if } i=i_\tau \text{ for } \tau>\mu\\ \mathsf{Enc}(\mathsf{pp},k_i,x_i^1) & \text{if } i=i_\tau \text{ for } \tau\leq\mu\end{cases}$
return $\{c_{i,l^*}\}_{i\in\mathcal{U}}$
```

We have that $G_0=H_0$ and $G_3=H_n$. To get from $H_{\mu-1}$ to H_μ we use the two intermediate games $H'_{\mu-1}$ and $H''_{\mu-1}$. In $H'_{\mu-1}$, the PRF of user i_μ is replaced by a RF and in $H''_{\mu-1}$ the challenge query of user $i_{\mu+1}$ is answered with an encryption of x^1_{q+1} instead of x^0_{q+1}. In Lemma 1 we needed two honest users in order to change the PRF to a RF. This was necessary, because we had to make sure that the sum of the ciphertexts remained unchanged. Here we are in the case that $\mathcal{A}$ does not corrupt the aggregator and, therefore, $\mathcal{A}$ is unable to recognize when the sum of the ciphertexts changes. Thus, we only need one honest user to exchange the PRF for a RF. We now describe the games in a bit more detail:

Game $H_{\mu-1}$: In this game the challenge query for users i_τ with $\tau \leq \mu - 1$ is answered with encryptions of $x^1_{i_\tau}$, whereas the challenge query for the users i_τ with $\tau > \mu - 1$ is answered with encryptions of $x^0_{i_\tau}$.

Game $H'_{\mu-1}$: This game guesses user i_μ and uses a RF instead of a PRF to answer the encryption and challenge queries for this user.

Game $H''_{\mu-1}$: In this game the challenge query for user i_μ is answered with an encryption of $x^1_{i_\mu}$ instead of $x^0_{i_\mu}$.

Game H_μ: This game uses a PRF instead of a RF for user i_μ again. The answer to the challenge query is an encryption of $x^1_{i_\tau}$ for all users with $\tau \leq \mu$.

Next, we prove the transitions between the games by reductions to the security of the PRF.

Transition from $H_{\mu-1}$ **to** $H'_{\mu-1}$: The reduction $\mathcal{B}$ guesses the user i^*_μ and generates keys k_i for all users, including the aggregator, except of user i^*_μ. Then $\mathcal{B}$ answers the queries as follows:

$\underline{\mathsf{QCorrupt}(i)}$: If $i \neq i^*_\mu$ then $\mathcal{B}$ returns the self generated key k_i. If $i = i^*_\mu$ then the guess that i^*_μ would be honest was wrong and $\mathcal{B}$ aborts. If $i = 0$, i.e. $\mathcal{A}$ wishes to corrupt the aggregator, $\mathcal{B}$ aborts, because we are in the case where $\mathcal{A}$ does not corrupt the aggregator.

$\underline{\mathsf{QEnc}(i, x_i, l)}$: If $i \neq i^*_\mu$ then $\mathcal{B}$ simply answers with $x_i + \mathsf{PRF}_{k_i}(l) \bmod R$. If $i = i^*_\mu$ then $\mathcal{A}$ asks l to their PRF challenger C_{PRF}, gets the answer a_l and sends $x_{i^*_\mu} + a_l \bmod R$ to $\mathcal{A}$.

$\underline{\mathsf{QChallenge}(\mathcal{U} = \{i_1, \ldots, i_\eta\}, \{x^0_i\}_{i \in \mathcal{U}}, \{x^1_i\}_{i \in \mathcal{U}}, l^*)}$: The reduction asks l^* to C_{PRF} and receives the answer a_{l^*}. Then $\mathcal{B}$ answers with $x^1_{i_\tau} + \mathsf{PRF}_{k_{i_\tau}}(l^*)$ for $\tau < \mu$, with $x^0_{i_\tau} + \mathsf{PRF}_{k_{i_\tau}}(l^*)$ for $\tau > \mu$ and with $x^0_{i_\tau} + a_{l^*}$ for $\tau = \mu$. If $i_\mu \neq i^*_\mu$, then $\mathcal{B}$'s guess of i^*_μ was wrong and they abort the game.

The reduction $\mathcal{B}$ can directly use $\mathcal{A}$'s output, as guess for C_{PRF}. If C_{PRF} used a PRF to generate its answers, then $\mathcal{B}$ perfectly simulates $H_{\mu-1}$ and if C_{PRF} used a RF, then $\mathcal{B}$ perfectly simulates $H'_{\mu-1}$.

Transition from $H'_{\mu-1}$ **to** $H''_{\mu-1}$: The games $H'_{\mu-1}$ and $H''_{\mu-1}$ are perfectly indistinguishable, because $\mathsf{RF} + x^0_\mu$ is identically distributed as $\mathsf{RF} + x^1_\mu$.

Transition from $H''_{\mu-1}$ **to** H_μ: Here we only need to change back the RF of user i_μ to a PRF. So this transition is the transition from $H_{\mu-1}$ to $H'_{\mu-1}$ applied backwards.

Guessing user i^*_μ entails a reduction loss of n, for both the transition from $H_{\mu-1}$ to $H'_{\mu-1}$ and from $H''_{\mu-1}$ to H_μ. Together with the n hybrid steps leads to a reduction loss of $2n^2$. □

B Security Proofs of the PRFs

In this section we prove that the PRF used in Section 4.1 is secure.

Theorem 2. Let $\lambda, q, p \in \mathbb{N}$ with $q > p$, $\mathbf{k} \xleftarrow{\$} \mathbb{Z}^\lambda_q$ and $H : \mathcal{X} \mapsto \mathbb{Z}^\lambda_q$. When we model H as a random oracle, then for any PPT adversary $\mathcal{A}$ on the PRF $F_{\mathbf{k}}(x) := \lfloor \langle H(x), \mathbf{k} \rangle \rceil_p$, there is an adversary $\mathcal{B}$ on $\mathrm{LWR}_{\lambda,q,p}$ with

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},F}(\lambda) \leq \mathsf{Adv}^{\mathrm{LWR}}_{\mathcal{B}}(\lambda).$$

Proof. The idea is for an LWR sample $(\mathbf{a}, b) \in \mathbb{Z}^\lambda_q \times \mathbb{Z}_p$, to interpret $\mathbf{a}$ as $H(x)$ for some x and the LWR secret $\mathbf{s}$ as the PRF-key $\mathbf{k}$. Then $b = \lfloor \langle H(x), \mathbf{k} \rangle \rceil_p$. This is possible, because $\mathcal{B}$ can program the random oracle accordingly.

The reduction $\mathcal{B}$ works as follows. $\mathcal{B}$ maintains a table of triples $(\cdot, \cdot, \cdot)$. If $\mathcal{A}$ asks a PRF query for value x, $\mathcal{B}$ looks for an entry $(x, \mathbf{a}, b)$ in the table and returns b if such an entry is present. If there is no such entry, then $\mathcal{B}$ requests an LWR sample from their LWR challenger, receives $(\mathbf{a}, b)$, stores $(x, \mathbf{a}, b)$ in the table and returns b to $\mathcal{A}$.

If $\mathcal{A}$ asks a RO query for value x, $\mathcal{B}$ again looks for an entry $(x, \mathbf{a}, b)$ in the table, but this time returns $\mathbf{a}$ if an entry is found. If there is no such entry, $\mathcal{B}$ again queries their LWR challenger, stores the answer $(\mathbf{a}, b)$ in the table as $(x, \mathbf{a}, b)$ and returns $\mathbf{a}$ to $\mathcal{B}$.

In LWR the values of $\mathbf{a}$ are uniformly random, therefore, $\mathcal{B}$'s answers to the RO queries of $\mathcal{A}$ are uniformly random as well. Furthermore, by maintaining the table of triples $(x, \mathbf{a}, b)$, $\mathcal{B}$ ensures that the answer to the RO queries are consistent with the answers of the PRF queries. If the LWR challenger returns random values b, then $\mathcal{B}$ perfectly simulates a random function. If the LWR challenger returns actual LWR samples then $\mathcal{B}$ perfectly simulates the PRF $F_{\mathbf{k}}(x) := \lfloor \langle H(x), \mathbf{k} \rangle \rceil_p$, where $\mathbf{k}$ is the LWR secret $\mathbf{s}$. Therefore, $\mathcal{B}$ can directly forward $\mathcal{A}$'s guess to their LWR challenger and wins the game, if $\mathcal{A}$ wins. □

```
$\mathsf{Encrypt}(\mathsf{pp}, \{k_{i,B}\}_{B \in \mathcal{B}(i)}, l, x_i, \epsilon, \delta)$:
$K := \lfloor \log_2(n) \rfloor + 1$
$\epsilon_0 := \frac{\epsilon}{K\Delta},\ \delta_0 := \frac{\delta}{K}$
for $B \in \mathcal{B}(i)$
    $\beta := \min\left(\frac{1}{|B|} \ln(\frac{1}{\delta_0}), 1\right)$
    $r \leftarrow \mathsf{Geom}^{\beta}(e^{\epsilon_0})$
    $x_i' = x_i + r$
    $c_{i,B} := \mathsf{PSA.Encrypt}(\mathsf{pp}, k_{i_B}, l, x_i')$
$c_i := \{c_{i,B}\}_{B \in \mathcal{B}(i)}$
```

Fig. 10. The encrypt procedure for the fault-tolerant scheme, where $\mathcal{B}(i)$ is the set of all $\log_2(n)$ nodes of the tree from client i to the root. For each of these nodes it calls PSA.Encrypt which is the encrypt algorithm of our proposed scheme in Figure 3.

```
$\mathsf{Geom}^{\beta}(\alpha)$:
$r \leftarrow \begin{cases} \mathsf{Geom}(\alpha) & \text{with probability } \beta \\ 0 & \text{with probability } 1-\beta \end{cases}$
return $r$
```

Fig. 11. The diluted geometric mechanism from [14]. The function $\mathsf{Geom}(\alpha)$ returns a value k with probability $\frac{\alpha-1}{\alpha+1}\alpha^{-|k|}$.

C Noise and Fault Tolerance

In this section we quickly describe how the noise is added in the fault tolerant version of our scheme. This is only a slight adaption of the method of Chan et al. especially of their Figure 2. They only considered user values in $\{0, 1\}$, whereas we consider values in $\{0, \ldots, \Delta\}$, where Δ is the largest possible power consumption for which privacy shall still hold. Therefore we have an additional factor of $1/\Delta$ in the computation of ϵ_0, the rest remains unchanged. Note that the factor of $1/\Delta$ is also present in [22], which introduced PSA. In the literature of differential privacy, Δ is also called the sensitivity of the function.

Figure 10 shows the encryption algorithm of the fault tolerant scheme. For each node on the path from client i to the root, i.e. for each instance of the PSA scheme in which client i is part of, the algorithm creates one ciphertext by calling the encrypt function from our PSA scheme in Figure 3. The parameters ϵ and δ are the same for all clients. One execution of the encrypt algorithm in Figure 10 provides (ϵ, δ) computational differential privacy. The values ϵ_0 and δ_0 are then set to accommodate for the fact, that the aggregator gets K ciphertexts per client.