Private Stream Aggregation from Labeled Secret Sharing Schemes

Hendrik Waldner[1], Tilen Marc[2,3], Miha Stopar[2], and Michel Abdalla[4,5]

[1] University of Edinburgh, Edinburgh, UK
hendrik.waldner@ed.ac.uk
[2] XLAB d.o.o., Ljubljana, Slovenia
{tilen.marc,miha.stopar}@xlab.si
[3] University of Ljubljana, Ljubljana, Slovenia
[4] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[5] INRIA, Paris, France

Abstract. The concept of *private stream aggregation* (PSA) has been proposed by Shi et al. (NDSS 2011) to allow for data analysis in a privacy-preserving manner. In this work, we introduce the notion of *labeled secret sharing* (LaSS) schemes and show how to use it to construct PSA schemes. We also show how to realize LaSS using pseudorandom functions or alternatively with a hash function modeled as a random oracle and how it can be used to construct PSA schemes. Additionally, we revisit the security model of Becker et al. (NDSS 2018) and describe stronger security notions for PSA. We then present additional constructions achieving the stronger security notions by relying on recent results on multi-client functional encryption. For all of our constructions, we present implementations to show their practicality and the performance gains over existing solutions.

1 Introduction

The concept of *private stream aggregation* (PSA) has been initially introduced by Shi et al. [SCR+11] to provide a cryptographic solution that allows an untrusted aggregator the statistical evaluation over data provided by different users. In more detail, we consider a setting where we have n different users $U_1, \ldots, U_n$ and an aggregator (denoted as U_0). The users U_i generate data d_i with respect to different time frames t. To allow the aggregator U_0 to compute statistics over the input of the users in the different time frames t, the users U_i take their data d_i, randomize it using some random noise r_i and encrypt the result using their encryption key sk_i under the time step t, which results in a ciphertext $\mathsf{ct}_{i,t}$. The aggregator then evaluates the decryption procedure using its aggregator key sk_0 on all the ciphertexts it receives from the different users $\mathsf{ct}_{1,t}, \ldots, \mathsf{ct}_{n,t}$, during a time step t, to receive as an output the sum of the input data together with the added noise $\sum_{i\in[n]} d_i + r_i$. A graphical illustration of the PSA setting can be found in Fig. 1.

To guarantee privacy for the different users in the described setting, a PSA scheme needs to fulfill *aggregator obliviousness* (AO). AO ensures that the aggregator does not learn anything about the input of an individual user U_i beyond what is leaked from the sum of the data of the users. This results in data privacy for an individual user U_i. Besides AO, we also want to guarantee that the output statistics are not significantly influenced by a missing database record. Therefore, we also require that the PSA scheme fulfills some notion of *differential privacy*, called *distributed differential privacy*. In the setting of distributed differential privacy, every user adds some noise to its data input and then sends their data to an untrusted aggregator, while still preserving differential privacy. This stands in contrast to the classical differential privacy setting in which a trusted aggregator adds some noise to the evaluated statistics.

In this work, we introduce the notion of *labeled secret sharing* (LaSS) schemes and present a compiler that turns a LaSS scheme into a PSA scheme. Besides this, we also show how to construct a LaSS scheme from *pseudorandom functions* (PRF), a standard cryptographic primitive or, alternatively, from a hash function modeled as a random oracle.

Besides the presentation of the LaSS-based PSA scheme, we also analyze the relation between *functional encryption* (FE) [O'N10, BSW11] and PSA. Functional encryption is a more general notion than PSA. In FE, different functional keys sk_f (associated with different functions f) can be generated using a so called master secret key msk. These functional keys can be used together with a ciphertext ct, that encrypts a message m. The result of the decryption procedure is the function f, associated with the functional key sk_f, applied on m, i.e. $f(m)$. This stands in contrast to the classical notion of encryption where it is only possible to decrypt the whole message or nothing.

To allow for the interaction between multiple users in this setting, the notion of multi-client functional encryption (MCFE) has been introduced [GGG+14]. In the setting of multi-client functional encryption, several users U_i can provide data by encrypting their input x_i using their personal encryption keys sk_i. Decryption then requires a functional key sk_f and n different ciphertexts, where n is the number of users. A graphical description of this setting can be found in Fig. 2. We further distinguish between MCFE schemes that are *labeled* and MCFE schemes that are *without labels*. In the labeled setting, every ciphertext ct_i is generated under a specific label ℓ and the decryption of ciphertexts is only possible if all of the ciphertexts that are used in the decryption procedure are generated under the same label ℓ. These labels have the purpose of preventing mix-and-match between ciphertexts. In the unlabeled setting, none of these restrictions are enforced.

In recent years, a lot of research has been done in constructing more practical functional encryption schemes for specific function classes [ALS16, Gay20, BCFG17, ABDP15, BJK15, KLM+18].[6] Especially for the functionality class of inner-products a lot of functional encryption schemes [ALS16, Gay20, BCFG17, ABDP15, BJK15], as well as multi-client schemes in the labeled and unlabeled setting have been proposed [ABM+20, ABG19, LT19, ACF+18, AGRW17, CDG+18a, CDG+18b, CDSG+20]. As already mentioned above, we investigate the implication of MCFE for the inner-product functionality on PSA and show how to construct PSA from

[6] It is not known how to construct practical functional encryption for a general class of functions, since all known constructions for a general class of functions require non standard assumptions [GGG+14, GGH+13, ABG+13, GGHZ16, Wat15].

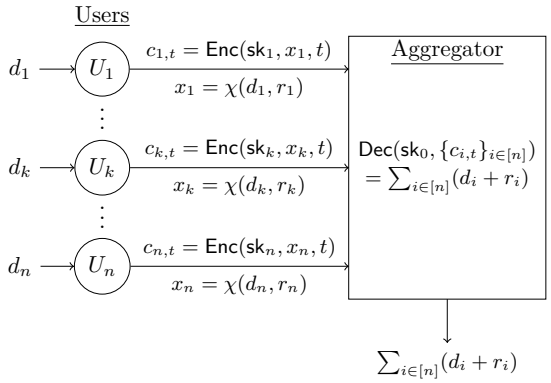


Fig. 1: The PSA Setting, where d_i denotes the data, r_i the random noise and χ the randomization function.

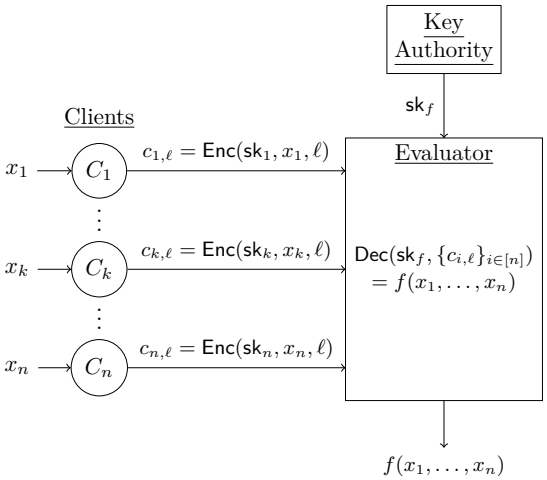


Fig. 2: The MCFE Setting, where x_i denotes the messages.

inner-product MCFE that fulfills a stronger security definition than the standard notion of AO-security. We denote this new security notion as "many-times" security and the initial AO-security notion as "encrypt-once" security.

Beyond this, we present some possible extensions of our constructions in Section 7.

1.1 Applications

In this section, we describe some application scenarios in which function evaluations over encrypted data using PSA and FE can be useful. These application scenarios have already been mentioned in [SCR+11].

Smart metering. Compared to traditional meters, which read the electrical usage once a month, smart meters read the electrical usage every 15 minutes. This more frequent reading allows to obtain precise information about, for example, the number of people in a household, their sleep and work habits, as well as their use of common household appliances. To allow for the desired data analysis while preserving the privacy of each individual household, a PSA scheme or a MCFE scheme can be used.

Public health and clinical research. The analysis of medical data is necessary to conduct medical research. The extent to which medical data is collected and disseminated is restricted due to privacy concerns. To protect the privacy of the medical data but nevertheless allow for some computation a PSA scheme or a MCFE scheme can be used.

Cloud services. The number of people and organizations that store their data in the cloud has been increasing over the last few years. To analyze the behavior of their users, the cloud providers wish to compute statistics over the data they store. A PSA scheme or a MCFE scheme can be deployed to allow for the computation of the statistics while protecting the privacy of every individual user.

For more application scenarios of PSA and FE, such as sensor network aggregation, population monitoring and sensing, we refer to [SCR+11].

1.2 Our Contributions

We can summarize our contributions as follows:

– The introduction of labeled secret sharing (LaSS) schemes and its realization from pseudorandom functions or hash functions modeled as random oracles.
– The construction of a simpler PSA scheme that fulfills the standard security notion of aggregator obliviousness under adaptive corruptions from a LaSS scheme.
– Description of significant shortcomings in the PSA construction and security model by Becker et al. [BGZ18].
– The presentation of a stronger security model for private stream aggregation with and without time steps.
– A black-box MCFE-based construction that achieves the stronger security notions and allows for richer functionalities than summation.
– Implementations of the PSA scheme based on a LaSS scheme using AES and SHA-3, as well as an implementation of the MCFE-based construction without time steps based on DDH, Paillier and LWE. In addition we present a comparison between other implemented PSA schemes.

All the constructions we present in this work can be instantiated using quantum-safe assumptions.

1.3 Technical Overview

In this section, we describe the techniques used to obtain our results. We start with labeled secret sharing schemes, followed by the contributions in the encrypt-once security model and we conclude with our results in the many-times security model.

Labeled Secret Sharing Schemes. In a labeled secret sharing scheme, we want to allow n different parties to non-interactively generate multiple secret sharings of 0.[7] All of these different secret sharings are generated with respect to a label ℓ. In more detail, every party is in possession of a secret key sk_i that allows, together with the label ℓ, the generation of a share $s_{i,\ell}$. Combining the shares of all the different users U_i, with $i \in [n]$, under the same label ℓ yields 0. We say that a LaSS scheme is secure, if the secret sharings generated under the different labels are indistinguishable from "real" secret sharing schemes.

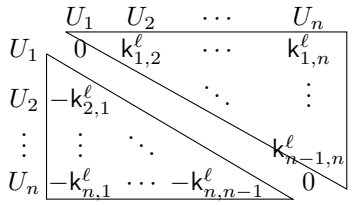
A LaSS scheme can be realized using pseudorandom functions and several shared keys between the different parties. Now, we give an informal description about how our construction works. The detailed construction is described in Section 3.

The idea of our construction is that every user U_i is in possession of a secret key sk_i that can be used to generate a new share, depending on the label ℓ. This can easily be realized for a single label by just generating a secret sharing of 0.

The problem of this approach is that it only allows the generation of a single secret sharing. A possible way to allow the user to generate multiple secret sharings under different lables would be to generate several secret sharings of 0 during the setup procedure. All of this secret shares would then be a part of the encryption key sk_i of ever user. The drawback of this straw-man solution is that the secret keys of the clients become very big and that the number of secret sharings is bounded. A better solution for this problem is to allow different users to generate a fresh secret sharing of 0 for each label on the fly. To achieve this non-interactive agreement, we distribute shared keys between all the users during the setup phase. In more detail, every user U_i is in possession of $n-1$ keys $\mathsf{k}_{i,1}, \ldots, \mathsf{k}_{i,n-1}$, where $\mathsf{k}_{i,j}$ is the shared key between user U_i and user U_j. This results in $\frac{n(n-1)}{2}$ keys overall. These keys can then be used together with a pseudorandom function PRF to generate the label-dependent shares $s_{i,\ell}$, by computing $s_{i,\ell} := \sum_{j \neq i} (-1)^{j<i} \mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell)$. To illustrate that these keys together generate a fresh secret sharing of 0, where the reconstruction simply consists of summing

[7] In the formal definition, we are more general and allow a LaSS scheme to generate fresh secret sharings for a pre-determined secret s. For simplicity, we only focus on the case $s = 0$ here.

up all the different $s_{i,\ell}$, we consider the following visualization:



Here, $\mathsf{k}^\ell_{i,j}$ are the "intermediate" keys defined by the PRF evaluation, i.e. $\mathsf{k}^\ell_{i,j} := \mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell)$ with $\mathsf{k}_{i,j}$ as the shared key between user U_i and user U_j. We can easily see that summing up all the entries of the matrix yields 0. Using this matrix representation, we see that the sum $\sum_{j\neq i}(-1)^{j<i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell)$ is the sum of all the entries in a column of the matrix. This overall results in the desired secret sharing of 0, with the sum $\sum_{i\in[n]} s_{i,\ell}$ being the reconstruction procedure.

To instantiate our scheme in the random-oracle model, we can simply replace the PRF with a hash function modeled as a random oracle.

The approach that we use to construct the LaSS scheme has already been described in previous works [ÁC11, KDK11] in the context of smart metering. The authors of these works only present an informal security analysis of their schemes but no formal security proof. In this work, we close this gap by presenting a formal security proof of this construction by relying on [ABG19].

Encrypt-Once Security Model. Our PSA scheme that achieves security in the encrypt-once model is based on a LaSS scheme. In more detail, we present a compiler that turns any LaSS scheme for $n+1$ parties into a PSA scheme. The main idea of the compiler is that every user U_i uses the share $s_{i,t}$ as its one-time pad encryption key for time step t to encrypt the message $x_{i,t}$, i.e. $\mathsf{ct}_{i,t} \coloneqq s_{i,t} + x_{i,t}$. The aggregator U_0 can then generate its share $s_{0,t}$ for time step t and decrypt by computing $s_{0,t} + \sum_{i\in[n]} \mathsf{ct}_{i,t}$.

For our implementation we instantiate the PRF-based LaSS using AES and the LaSS scheme in the random oracle model using SHA-3.

Many-Times Security Model. In addition to the LaSS-based PSA scheme, we also present how to obtain a PSA scheme in a more general security model from inner-product multi-client functional encryption. The fact that multi-client functional encryption implies private stream aggregation has already been mentioned in several works [CDG+18a, ABKW19, BJL16, LT19]. In the recent work of [LT19] the authors describe concretely how to obtain a PSA scheme from a MCFE scheme with labels by generating the functional key for the all one vector as the aggregator key. In this work, we formally define this straightforward construction and show the different grades of security it fulfills.

In the initial security definition of PSA, time steps t in the encryption procedure are required and only a single encryption of every user is possible. We can think of two variants of this security definition:

1. No time steps and multiple encryptions for each user.
2. Time steps and multiple encryptions for each user under every time step.

In Section 5, we present the formal definition of these notions, as well as a black-box construction based on MCFE that achieves theses security levels. For the implementation of the black-box MCFE-based construction, we use the MCFE scheme of Abdalla et al. [ACF+18], which allows for instantiations based on DDH, Paillier, and LWE. More details on this can be found in Section 6.1.

In [BGZ18], Becker et al. present another security notion that also does not rely on time steps. In their security game, the adversary is only allowed to ask a single challenge query for every user. This results in an even weaker security model than the encrypt-once model, since several encryptions, one per time step, can be generated in the encrypt-once model. Therefore, the security model of Becker et al. can be seen as an

equivalent to the encrypt-once model with only a single time step, which overall results in a one-time secure scheme.[8] To achieve this level of security, however, no complexity assumptions would be needed since the information-theoretic construction described in Section 1.3 on Page 4 would already suffice.

To show why the restriction on the challenge queries is necessary, we present, in Remark 5.3, a concrete attack on the scheme by Becker et al. when two challenge queries are allowed. The attack is indeed generic and applies to any scheme in which users only encrypt a scalar.

The "All-Ciphertext" Condition. Another restriction that is enforced on the challenge oracle queries of the adversary in the security game is that the adversary needs to specify a set U' of users in advance for which a query needs to be submitted and in the case that the adversary compromises the aggregator, it must hold that the sum of the submitted challenge messages for the determined subset of users U' is equal. This is an unnatural requirement which is enforced due to information leakage that occurs when the adversary does not have all the ciphertexts for non-compromised users. A similar problem occurs in the setting of multi-client functional encryption, where it is required that the adversary asks a challenge oracle query in every uncorrupted position. This is also due to possible information leakage that can occur if challenge queries are not submitted in every uncorrupted position. To also achieve security in the case that a challenge query has not been submitted in every uncorrupted positions several security compilers have been introduced in the context of MCFE. The first of these compilers has been proposed in the work of Abdalla et al. [AGRW17]. This compiler is restricted to MCFE schemes without labels. Since then, several other compilers have been proposed for the more general setting of MCFE with labels [ABG19, ABKW19, CDG+18b]. These compilers can directly be adapted to the setting of private stream aggregation and achieve security in a game where no restrictions regarding the challenge queries are enforced on the adversary.

The second security definition described above together with the mentioned security compiler can be seen as the strongest possible security definition in the setting of PSA.

1.4 Related Work

In this section, we present the main related work, which contains of work in smart metering, functional encryption and the recent results in private stream aggregation.

Smart Metering. The constructions we present have been already considered in the context of smart metering [ÁC11, KDK11]. Smart metering and PSA differ in a few points. In the setting of smart metering, the definition explicitly considers node and communication failures, whereas this has only been considered in follow up works on private stream aggregation [CSS12, JK12] as an additional security property. In the smart metering setting, they also do not consider a specific aggregator but allow a public computation of the outcome. It has already been noted in [SCR+11] that it is possible for some schemes to allow public aggregation, also our scheme fulfills this property. We describe this in more detail in Section 7. In the two works [ÁC11, KDK11], the authors only present an informal security analysis, but no formal proof of security. In this work we close this gap by providing a formal proof of AO-security.

Functional Encryption. As already mentioned above, in this work, we rely on the recent results on inner-product MCFE, since we need it for the instantiation of our MCFE-based PSA scheme in the many-times security model. Our construction can be instantiated using the following works [ABM+20, ABG19, LT19, ACF+18, AGRW17, CDG+18b, CDSG+20] for the setting without time steps and the works [ABM+20, ABG19, LT19, CDG+18b, CDSG+20] for the setting with time steps. For the implementation of the MCFE-based PSA scheme in the many-times security model without time steps, we rely on the work of Abdalla et al. [ACF+18],

[8] Interestingly, this level of security can also be achieved by our construction, if the underlying LaSS scheme only works for a single label ℓ, which, as mentioned above, can be achieved information-theoretically using a secret sharing of 0. More detail on this is given in Section 1.3 on Page 4 and in Remark 5.3.

which has been proven to fulfill the MCFE notion in [ABKW19]. We present the implementation results in Section 6.2.

To get rid of the requirement in the challenge queries, we can use one of the security compilers presented in [AGRW17, ABKW19, CDG+18b, ABG19]. For the many-times security setting without time steps, the compilers of Abdalla et al. [AGRW17, ABKW19] can be used, this would only require the existence of a symmetric encryption scheme as the underlying assumption. The more general setting of many-times security with time steps the compiler of Abdalla et al. [ABG19] can be used. This compiler only requires the existence of PRFs.

Private Stream Aggregation. After the introductory paper of PSA [SCR+11] many other schemes in the same setting have been proposed. These contributions can be divided into different categories: PSA schemes under stronger security notions, such as dynamic joins and leaves [CSS12, JK12], PSA schemes from lower assumptions, such as DDH [BJL16, VA18], Paillier [JL13] or LWE [BGZ18, VA18] and works that focus on the analysis of the differential privacy mechanisms used in the setting of PSA [VA18]. Besides this, there are also several works that use PSA as a building block in larger protocols [SSL+15, RZL+13, LSL+13, BGZ17]. For a survey of the different types of aggregation schemes, we refer to [JKD12].

The main related works are the ones by Becker et al. [BGZ18] and by Valovic and Aldà [VA18]. Becker et al. [BGZ18], present a new PSA scheme using additive homomorphic encryption and the augmented learning with errors (A-LWE) assumption, which they call *LaPS*. Their construction is secure in the one-time security setting without time steps, as we point out in this work, and the enforcement of the "all-ciphertexts" condition on the adversary, which can be removed using one of the compilers mentioned above. The authors also present an implementation of their construction together with some benchmarks. In Section 6, we compare their results with our implementation.

The scheme by Valovich and Aldà [VA18] uses a similar approach, but requires a weak key-homomorphic PRF. Their scheme only requires a secret sharing of 0 between the different parties instead of shared keys between every user. Their security proof in the encrypt-once model only works in the selective corruption setting, a more restrictive security setting where the adversary needs to announce all its corruption queries at the beginning of the security game. Security against adaptive corruption of this scheme can also be proven using the techniques of Abdalla et al. [ABG19], which we also use in this work.

1.5 Concurrent Work

Concurrently to our work, Takeshita et al. [TKGJ20] proposed a new lattice-based PSA scheme called *SLAP*. The SLAP scheme is based on the underlying ideas of homomorphic encryption schemes and lattice-based hardness assumptions. This stands in contrast to other constructions which rely on homomorphic encryption schemes in a more general way such as LaPS [BGZ18]. This tailored approach to the application of PSA leads to more efficient constructions and is more optimally tailored to the application of PSA. The SLAP scheme has two variants. The first scheme draws its core ideas from the BGV scheme [BGV12] and is also based on NTRU [HPS98]. The second scheme is based on the B/FV schemes [FV12] and also relies on the LPR hardness [LPR10].

Additionally, the authors also implement their scheme and highlight the improvements compared to the work of Becker et al. [BGZ18]. We evaluate their results in context with our results in Section 6.

2 Preliminaries

Notation. We denote the set $\{1, \ldots, n\}$ as $[n]$ and the set $[n] \cup \{0\}$ as $[n]_0$. For vectors we write $\boldsymbol{x}$ and denote the i-th element as x_i. We use $(-1)^{j<i}$ to denote $\frac{j-i}{|j-i|}$. The winning probability of an adversary $\mathcal{A}$ in a game or experiment G, which is $\Pr[\mathsf{G}(\lambda, n, \mathcal{A}) = 1]$, is denoted as $\mathsf{Win}^{\mathsf{G}}_{\mathcal{A}}(\lambda, n)$ with λ the security parameter and n an additional parameter. The probability is taken over the random coins of G and $\mathcal{A}$. In this context, we define the distinguishing advantage between games G_0 and G_1 of an adversary $\mathcal{A}$ as $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{A}}(\lambda, n) = \left|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)\right|$.

2.1 Private Stream Aggregation

In this section, we recap the definition of private stream aggregation as introduced in [SCR+11].

Definition 2.1 (Private Stream Aggregation). *Let $\mathcal{T} = \{0,1\}^*$ or $\{\bot\}$ be a set of time steps. A private stream aggregation (PSA) scheme for the time step set $\mathcal{T}$ is a tuple of three algorithms $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$:*

- $\mathsf{Setup}(1^\lambda, n)$: *Takes as input a unary representation of the security parameter λ and the number of parties n and outputs n secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$ for the parties and an aggregator secret key sk_0.*
- $\mathsf{Enc}(\mathsf{sk}_i, x_i, t)$: *Takes as input a secret key sk_i, some message $x_i \in \mathcal{X}_i$ to encrypt, and a time step $t \in \mathcal{T}$, and outputs ciphertext $\mathsf{ct}_{i,t}$.*
- $\mathsf{Dec}(\mathsf{sk}_0, \{\mathsf{ct}_{i,t}\}_{i\in[n]})$: *Takes as input the aggregator key sk_0 and n ciphertexts encrypted under the same time step t and outputs a value y.*

A scheme PSA is correct, if for all $\lambda, n \in \mathbb{N}$, $\{\mathsf{sk}_i\}_{i\in[n]_0} \leftarrow \mathsf{Setup}(1^\lambda, n)$, $t \in \mathcal{T}$, $x_i \in \mathcal{X}_i$, we have

$$\Pr[\mathsf{Dec}(\mathsf{sk}_0, (\mathsf{Enc}(\mathsf{sk}_i, x_i, t))_{i\in[n]}) = \textstyle\sum_{i\in[n]} x_i] = 1.$$

When $\mathcal{T} = \{0,1\}^*$, we say that the scheme is *with time steps*. When $\mathcal{T} = \{\bot\}$, we say that the scheme is *without time steps*, and we often omit t.

We also define PSA in the context without time steps, this is not captured in the initial definition of PSA by Shi et al. [SCR+11]. This is due to the fact that we define in Section 5.1 a stronger notion of AO-security, in which multiple left-or-right oracle queries without any time steps are possible.[9]

In our adaptation of the security definition, we allow the adversary to also ask challenge queries for the compromised positions under the condition that both of the challenge messages in this case are the same. This does not give the adversary any distinguishing advantage compared to the initial definition of Shi et al. [SCR+11].

$\mathbf{AO}_\beta^{\mathsf{PSA}}(\lambda, n, \mathcal{A})$

$\{\mathsf{sk}_i\}_{i\in[n]_0} \leftarrow \mathsf{Setup}(1^\lambda, n)$

$\alpha \leftarrow \mathcal{A}^{\mathsf{QComp}(\cdot),\mathsf{QEnc}(\cdot,\cdot,\cdot),\mathsf{QLeftRight}(\cdot,\cdot,\cdot,\cdot)}(1^\lambda, n)$

Output: α if Condition (*) is satisfied, or a uniform bit otherwise

Fig. 3: Aggregator Obliviousness Security Game.

Definition 2.2 (Aggregator Obliviousness). *Let PSA be a PSA scheme and $\mathcal{T}$ a set of time steps. We define the experiment $\mathrm{AO}_\beta^{\mathsf{PSA}}$ in Fig. 3, where the oracles are defined as:*

- **Compromise oracle** $\mathsf{QComp}(i)$: *Outputs the private key sk_i of user i. For $i = 0$, it outputs the aggregator key sk_0. We denote by $\mathcal{CS}$ the set of compromised users at the end of the experiment.*
- **Encryption oracle** $\mathsf{QEnc}(i, x_i, t)$: *Outputs $\mathsf{ct}_{i,t} = \mathsf{Enc}(\mathsf{sk}_i, x_i, t)$ on a query (i, x_i, t).*
- **Left-or-Right oracle** $\mathsf{QLeftRight}(i, x_i^0, x_i^1, t^\star)$: *Outputs $\mathsf{ct}_{i,t^\star} = \mathsf{Enc}(\mathsf{sk}_i, x_i^\beta, t^\star)$ on a query $(i, x_i^0, x_i^1, t^\star)$. This oracle can be queried on at most one time step $t^\star$, the adversary determines a subset U of users and the adversary is required to query it once for every $i \in U$. Further queries that hurt this condition will be ignored.*

and where Condition (*) *holds if all the following conditions hold:*

[9] The definition of no time steps in the encrypt once model is easily achievable by generating an additive secret sharing of 0 and use the secret shares as the secret keys of the different users and the aggregator.

– *If $i \in \mathcal{CS}$ (i.e., user i is compromised): for any query* $\mathsf{QLeftRight}(i, x_i^0, x_i^1, t^\star), x_i^0 = x_i^1$.
– *The queries to the encryption oracle* QEnc *must be of the form $(\cdot, \cdot, t)$ with $t \neq t^\star$, where $t^\star$ is the time step queried to the left-right oracle* QLeftRight.
– *If $0 \in \mathcal{CS}$ (i.e., the aggregator is compromised): for time step $t^\star$, for any family of queries $\{\mathsf{QLeftRight}(i, x_i^0, x_i^1, t^\star)\}_{i\in[n]}$, for any family of inputs $\{x_i \in \mathcal{X}_{i,\sigma}\}_{i\in\mathcal{CS}}$ we define $x_i^0 = x_i^1 = x_i$ for any slot $i \in \mathcal{CS}$, and we require that:*

$$\sum_{i\in[n]} x_i^0 = \sum_{i\in[n]} x_i^1 \ .$$

We define the advantage of an adversary $\mathcal{A}$ in the following way:

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

A private stream aggregation scheme PSA *is aggregator oblivious (*AO*-secure), if for any n, for any polynomial-time adversary $\mathcal{A}$, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathrm{AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) \leq \mathrm{negl}(\lambda)$.

In the description of the left-or-right oracle, we make some additional requirements that this oracle needs to be queried on a determined subset of users. We describe how this restriction can be removed in the next section.

In addition to the notion of aggregator obliviousness, a PSA scheme involves the application of a differential privacy mechanism. In more detail, the input x_i, that is taken as an input to the encryption procedure of the PSA scheme, is derived by applying a randomization function χ on the data input d_i, as well as some noise r_i, i.e. $x_i = \chi(d_i, r_i)$. Differential privacy in the context of PSA has already been extensively analyzed [SCR+11, BGZ18, VA18]. In [VA18], the authors show that every differential privacy preserving mechanism preserves computational differential privacy when executed through a PSA scheme. We rely on this result in our work and do not consider differential privacy specifically.

2.2 Multi-Client Functional Encryption

In this section, we define the notion of MCFE [GGG+14].

Definition 2.3 (Multi-Client Functional Encryption). *Let $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ be a family (indexed by ρ) of sets $\mathcal{F}_\rho$ of functions $f\colon \mathcal{X}_{\rho,1} \times \cdots \times \mathcal{X}_{\rho,n_\rho} \to \mathcal{Y}_\rho$.*[10] *Let* $\mathsf{Labels} = \{0,1\}^*$ *or* $\{\bot\}$ *be a set of labels. A multi-client functional encryption scheme (MCFE) for the function family $\mathcal{F}$ and the label set* Labels *is a tuple of four algorithms* $\mathsf{MCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

– $\mathsf{Setup}(1^\lambda, n)$: *Takes as input a unary representation of the security parameter λ and the number of parties n, and outputs n secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$ and a master secret key* msk.
– $\mathsf{KeyGen}(\mathsf{msk}, f)$: *Takes as input the master secret key* msk *and a function $f \in \mathcal{F}_\rho$, and outputs a functional decryption key sk_f.*
– $\mathsf{Enc}(\mathsf{sk}_i, x_i, \ell)$: *Takes as input a secret key sk_i, a message $x_i \in \mathcal{X}_{\rho,i}$ to encrypt, a label $\ell \in$* Labels*, and outputs a ciphertext $\mathsf{ct}_{i,\ell}$.*
– $\mathsf{Dec}(\mathsf{sk}_f, \{\mathsf{ct}_{i,\ell}\}_{i\in[n]})$: *Takes as input a functional key sk_f and n ciphertexts under the same label ℓ and outputs a value $y \in \mathcal{Y}_\rho$.*

A scheme MCFE *is correct, if for all $\lambda, n \in \mathbb{N}$, $f \in \mathcal{F}_\rho$, $\ell \in$* Labels*, $x_i \in \mathcal{X}_{\rho,i}$, when $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$ and $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, (\mathsf{Enc}(\mathsf{sk}_i, x_i, \ell))_{i\in[n]}) = f(x_1, \ldots, x_n)\right] = 1 \ .$$

When ρ is clear from context, the index ρ is omitted. When $\mathsf{Labels} = \{0,1\}^*$, we say that the scheme is *labeled* or *with labels*. When $\mathsf{Labels} = \{\bot\}$, we say that the scheme is *without labels*, and we often omit ℓ.

We also recap the security definition:

[10] All the functions inside the same set $\mathcal{F}_\rho$ have the same domain and the same range.

Definition 2.4 (Security of MCFE). *Let* MCFE *be an MCFE scheme,* $\mathcal{F} = \{\mathcal{F}_\rho\}_\rho$ *a function family indexed by* ρ *and* Labels *a label set. For* $\mathrm{xx} \in \{\mathrm{pos}^+, \mathrm{any}\}$ *and* $\beta \in \{0,1\}$*, we define the experiment* $\text{xx-IND}_\beta^{\mathsf{MCFE}}$ *in Fig. 4, where the oracles are defined as:*

- **Corruption oracle** $\mathsf{QCor}(i)$: *Outputs the encryption key* sk_i *of slot* i*. We denote by* $\mathcal{CS}$ *the set of corrupted slots at the end of the experiment.*
- **Left-Right oracle** $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$: *Outputs* $\mathsf{ct}_{i,\ell} = \mathsf{Enc}(\mathsf{sk}_i, x_i^\beta, \ell)$ *on a query* (i, x_i^0, x_i^1, ℓ)*. We denote by* $Q_{i,\ell}$ *the number of queries of the form* $\mathsf{QLeftRight}(i, \cdot, \cdot, \ell)$*.*
- **Encryption oracle** $\mathsf{QEnc}(i, x_i, \ell)$: *Outputs* $\mathsf{ct}_{i,\ell} = \mathsf{Enc}(\mathsf{sk}_i, x_i, \ell)$ *on a query* (i, x_i, ℓ)*.*
- **Key generation oracle** $\mathsf{QKeyG}(f)$: *Outputs* $\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{msk}, f)$*.*

and where Condition (*) *holds if all the following conditions hold:*

- *If* $i \in \mathcal{CS}$ *(i.e., slot* i *is corrupted): for any query* $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$*,* $x_i^0 = x_i^1$*.*
- *For any label* $\ell \in \mathsf{Labels}$*, for any family of queries* $\{\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$ *or* $\mathsf{QEnc}(i, x_i, \ell)\}_{i \in [n] \setminus \mathcal{CS}}$*, for any family of inputs* $\{x_i \in \mathcal{X}_{\rho,i}\}_{i \in \mathcal{CS}}$*, for any query* $\mathsf{QKeyG}(f)$*, we define* $x_i^0 = x_i^1 = x_i$ *for any slot* $i \in \mathcal{CS}$ *and any slot queried to* $\mathsf{QEnc}(i, x_i, \ell)$*, and we require that:*

$$f(\boldsymbol{x}^0) = f(\boldsymbol{x}^1) \quad \text{where } \boldsymbol{x}^b = (x_1^b, \ldots, x_n^b) \text{ for } b \in \{0,1\} \ .$$

  We insist that if one index $i \notin \mathcal{CS}$ *is not queried for the label* ℓ*, there is no restriction.*
- *If* $\mathrm{xx} = \mathrm{pos}^+$*: For any label* $\ell \in \mathsf{Labels}$*, either the adversary makes no left-right oracle query or makes at least one left-right encryption query for each slot* $i \in [n] \setminus \mathcal{CS}$*.*

We define the advantage of an adversary $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}^{\text{xx-IND}}_{\mathsf{MCFE},\mathcal{A}}(\lambda, n) = \left| \Pr[\text{xx-IND}_0^{\mathsf{MCFE}}(\lambda, n, \mathcal{A}) = 1] - \Pr[\text{xx-IND}_1^{\mathsf{MCFE}}(\lambda, n, \mathcal{A}) = 1] \right| \ .$$

A multi-client functional encryption scheme MCFE *is* xx*-IND secure, if for any* n*, for any polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\text{xx-IND}}_{\mathsf{MCFE},\mathcal{A}}(\lambda, n) \leq \mathrm{negl}(\lambda)$*.*

We omit n when it is clear from the context. We also often omit $\mathcal{A}$ from the parameter of experiments or games when it is clear from context.

To turn a MCFE scheme that is pos^+-IND-secure into an MCFE scheme that is any-IND-secure, we can use one of the compilers presented in [ABG19, ABKW19, CDG+18b]. The most current compiler has been presented in [ABG19]. It only relies on pseudorandom functions, but has a quadratic blow up in the size of the ciphertexts.

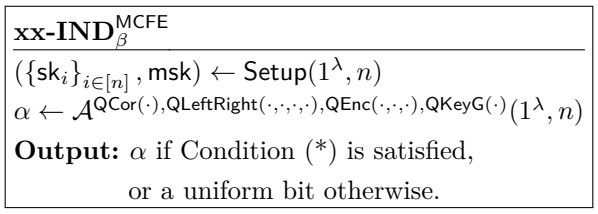


Fig. 4: Security games for MCFE.

2.3 Inner-Product Functionality

We describe the functionalities supported by the constructions in this paper, by considering the index ρ of $\mathcal{F}$ in more detail.

The index of the family is defined as $\rho = (\mathcal{R}, n, m, X, Y)$ where $\mathcal{R}$ is either $\mathbb{Z}$ or $\mathbb{Z}_L$ for some integer L, and n, m, X, Y are positive integers. If X, Y are omitted, then $X = Y = L$ is used (i.e., no constraint).

This defines $\mathcal{F}_\rho = \{f_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n} : (\mathcal{R}^m)^n \to \mathcal{R}\}$ where

$$f_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n}(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n) = \sum_{i=1}^{n} \langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle \ ,$$

where the vectors satisfy the following bounds: $\|\boldsymbol{x}_i\|_\infty < X, \|\boldsymbol{y}_i\|_\infty < Y$ for $i \in [n]$, and where $\boldsymbol{x} \in \mathcal{R}^{mn}$ and $\boldsymbol{y} \in \mathcal{R}^{mn}$ are the vectors corresponding to the concatenation of the n vectors $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ and $\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n$ respectively.

2.4 Pseudorandom Functions

We recap the definition of a pseudorandom function (PRF) [GGM86].

Definition 2.5 (Pseudorandom Function). *Let* $\mathsf{PRF} : \mathcal{K} \times \mathcal{V} \to \mathcal{W}$ *be a deterministic polynomial-time algorithm, with key space* $\mathcal{K}$*, domain* $\mathcal{V}$ *and range* $\mathcal{W}$*. For* $\beta \in \{0,1\}$*, we define the experiment* $\mathrm{IND}_\beta^{\mathsf{PRF}}$ *in Fig. 5, where the oracle* $\mathcal{O}_{\mathsf{PRF}}$ *is defined as:*

$$\mathcal{O}_{\mathsf{PRF}}(t) = \begin{cases} \mathsf{PRF}_{\mathsf{K}}(t) & \text{if } \beta = 0 \\ \mathsf{RF}(t) & \text{if } \beta = 1 \end{cases}$$

with $\mathsf{RF}(t)$ *denoting a random function. We define the advantage of an adversary* $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{A}}(\lambda) = |\Pr[\mathrm{IND}_0^{\mathsf{PRF}}(\lambda,\mathcal{A})] - \Pr[\mathrm{IND}_1^{\mathsf{PRF}}(\lambda,\mathcal{A})]|$$

A pseudorandom function PRF *is secure, if for any polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$.

$\mathbf{IND}_\beta^{\mathsf{PRF}}(\lambda,\mathcal{A})$

$\mathsf{K} \leftarrow \mathcal{K}$

$\alpha \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PRF}}(\cdot)}(1^\lambda)$

Output: α

Fig. 5: Security Games for PRF

2.5 Symmetric Encryption and One-time Pad

In this section, we recap the security definition for symmetric encryption and the one-time pad. First, we formally define a symmetric encryption scheme.

Definition 2.6 (Symmetric Encryption). *A symmetric encryption scheme (SE) for the key space* $\mathcal{K}$ *and the message space* $\mathcal{M}$ *is a couple of algorithms* $\mathsf{SE} = (\mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Enc}(\mathsf{K}, m)$*: Takes as input the symmetric key* K*, a message* $m \in \mathcal{M}$ *to encrypt, and outputs a ciphertext* ct.
- $\mathsf{Dec}(\mathsf{K}, \mathsf{ct})$*: Takes as input the symmetric key* K *and a ciphertext* ct *and outputs a message or* $\perp$ *if decryption fails.*

A scheme SE *is correct, if for all* $\rho \in \mathbb{N}$, $\mathsf{K} \leftarrow \mathcal{K}$, $m \in \mathcal{M}$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{K}, \mathsf{Enc}(\mathsf{K}, m)) = m\right] = 1$$

Security for a symmetric encryption scheme is defined in the following way.

Definition 2.7 (IND-CPA Security of SE). *Let* $\mathsf{SE} = (\mathsf{Enc}, \mathsf{Dec})$ *be an SE scheme, for the message space* $\mathcal{M}$*. We define the experiment* $\text{IND-CPA}_\beta^{\mathsf{SE}}$ *in Fig. 6, where the oracle is defined as:*

- **Left-or-Right Oracle** $\mathsf{QLeftRight}(m^0, m^1)$*: Outputs* $\mathsf{ct} = \mathsf{Enc}(\mathsf{K}, m^\beta)$.

We define the advantage of an adversary $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = |\Pr[\text{IND-CPA}_0^{\mathsf{SE}}(\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-CPA}_1^{\mathsf{SE}}(\lambda, \mathcal{A}) = 1]| \ .$$

A symmetric encryption scheme SE *is called IND-CPA secure, if for any PPT adversary* $\mathcal{A}$ *it holds that* $\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \text{negl}(\lambda)$*. A scheme* SE *is called perfectly secure, if for any adversary* $\mathcal{A}$ *with a single oracle query to* QLeftRight *it holds that* $\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = 0$

$\boxed{\begin{array}{l}\textbf{IND-CPA}_\beta^{\mathsf{SE}}(\lambda, \mathcal{A}) \\ \hline \mathsf{K} \leftarrow \mathcal{K} \\ \alpha \leftarrow \mathcal{A}^{\mathsf{QLeftRight}(\cdot,\cdot)}(1^\lambda) \\ \textbf{Output: } \alpha\end{array}}$

Fig. 6: IND-CPA Security Game for a symmetric encryption scheme

$\boxed{\begin{array}{l}\underline{\mathsf{Enc}(\mathsf{K}, m \in \mathcal{M}) :} \\ \mathsf{ct} := m + \mathsf{K} \\ \text{Return } \mathsf{ct} \\ \underline{\mathsf{Dec}(\mathsf{K}, \mathsf{ct}) :} \\ m := \mathsf{ct} - \mathsf{K} \\ \text{Return } m\end{array}}$

Fig. 7: The One-Time Pad

A specific symmetric encryption scheme that fulfills perfect security is the one-time pad (Fig. 7). The perfect security has first been proven in [Sha01] for the XOR operation. An adaption of this proof to finite groups is straightforward and can be found for example in [Wic15].

Theorem 2.8 (One-Time Pad). *The scheme* $\mathsf{SE} = (\mathsf{Enc}, \mathsf{Dec})$ *defined in Fig. 7 is perfectly secure. Namely, for any adversary* $\mathcal{A}$ *it holds that* $\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}} = 0$ *for a single oracle query to* QLeftRight.

3 Labeled Secret Sharing Scheme

In this section, we introduce the notion of a labeled secret sharing (LaSS) scheme, together with its security definition.

3.1 Definition of Labeled Secret Sharing Schemes

Definition 3.1 (Labeled Secret Sharing Scheme). *Let* $\mathsf{Labels} = \{0,1\}^*$ *be a set of labels. A labeled secret sharing (LaSS) scheme for the labels* Labels *is a tuple of three algorithms* $\mathsf{LaSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Reconstruct})$*:*

- $\mathsf{Setup}(1^\lambda, s, n)$*: Takes as input a unary representation of the security parameter* λ*, a secret* s *and the number of parties* n *and outputs* n *secret keys* $\{\mathsf{sk}_i\}_{i\in[n]}$.
- $\mathsf{ShareGen}(\mathsf{sk}_i, \ell)$*: Takes as input a secret key* sk_i*, and a label* $\ell \in \mathsf{Labels}$*, and outputs a share* $s_{i,\ell}$.
- $\mathsf{Reconstruct}(\{s_{i,\ell}\}_{i\in[n]_0})$*: Takes as input* $n+1$ *shares generated under the same label* ℓ *and outputs a value* t.

$\mathbf{xx\text{-}IND}^{\mathsf{LaSS}}_{\beta}$

$s \leftarrow \mathcal{A}(1^\lambda, n)$

$(\{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, s, n)$

$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot),\mathsf{QShare}(\cdot,\cdot)}(1^\lambda, s, n)$

Output: α

Fig. 8: Security games for LaSS.

A LaSS *is correct, if for all* $\lambda, s, n \in \mathbb{N}$, $\{\mathsf{sk}_i\}_{i\in[n]_0} \leftarrow \mathsf{Setup}(1^\lambda, , s, n)$, $\ell \in \mathsf{Labels}$, *we have*

$$\Pr[\mathsf{Reconstruct}((\mathsf{ShareGen}(\mathsf{sk}_i, \ell))_{i\in[n]}) = s] = 1.$$

In a similar flavor to Adballa et al. [ACF+18], we define an Add procedure for our scheme.

Definition 3.2 (Linear Homomorphism). *A LaSS scheme* $\mathsf{LaSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Reconstruct})$ *satisfies the linear encryption property if there exists a deterministic algorithm* Add *that takes a share* $s_{i,\ell}$ *and a value* $x_{i,\ell}$ *as an input, such that the following are identically distributed:*

$$\mathsf{Add}(s_{i,\ell}, x_{i,\ell}) \text{ and } s_{i,\ell} \ .$$

When these modified shares are used in the reconstruction procedure, it outputs $s + \sum_{i\in[n]} x_{i,\ell}$, where $x_{i,\ell}$ are the values added to the shares $s_{i,\ell}$.

Now, we are ready to state the security for LaSS. We distinguish between two security notions: semi-adaptive and adaptive security. In the semi-adaptive security game, the adversary is required to submit an additional bit with the first challenge query of every slot that indicates whether the slot can be corrupted later in the security game or not.

Definition 3.3 (Security of LaSS). *Let* LaSS *be a LaSS scheme and* Labels *a label set. For* $\mathrm{xx} \in \{\mathrm{semi}, \mathrm{ad}\}$ *and* $\beta \in \{0, 1\}$, *we define the experiment* $\text{xx-IND}^{\mathsf{LaSS}}_{\beta}$ *in Fig. 8, where the oracles are defined as:*

- **Corruption oracle** $\mathsf{QCor}(i)$: *Outputs the encryption key* sk_i *of slot* i. *We denote by* $\mathcal{CS}$ *the set of corrupted slots at the end of the experiment.*
- **Share generation oracle** $\mathsf{QShare}(i, \ell)$: *Outputs*

$$\mathsf{ShareGen}(\mathsf{sk}_i, \ell) \text{ if } \beta = 0 \text{ and } \mathsf{PS}(i, \ell) \text{ if } \beta = 1 \ ,$$

where $\mathsf{PS}(i, \ell)$ *denotes the generation of the* i*-th share of a perfect secret sharing.*

  - *If* $\mathrm{xx} = \mathrm{semi}$*: The adversary is required to submit an additional bit* b *for the first query to every slot* i. *In the case that* $b = 0$ *for a slot* i, *the adversary is not allowed to corrupt the corresponding slot later in the security game. We call such a slot an* explicitly honest slot. *For the case that* $b = 1$ *for a slot* i, *the adversary is allowed to corrupt the corresponding slot later in the security game but the output of the oracle is* $\mathsf{ShareGen}(\mathsf{sk}_i, \ell)$ *on a query* $(i, \ell, 1)$.

We define the advantage of an adversary $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}^{\text{xx-IND}}_{\mathsf{LaSS},\mathcal{A}}(\lambda, n) = \left|\Pr[\text{xx-IND}^{\mathsf{LaSS}}_0(\lambda, n, \mathcal{A}) = 1] - \Pr[\text{xx-IND}^{\mathsf{LaSS}}_1(\lambda, n, \mathcal{A}) = 1]\right| \ .$$

A labeled secret sharing scheme LaSS *is* xx*-IND secure, if for any* n, *for any polynomial-time adversary* $\mathcal{A}$, *there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\text{xx-IND}}_{\mathsf{LaSS},\mathcal{A}}(\lambda, n) \leq \mathrm{negl}(\lambda)$.

It trivially holds that ad-IND $\Rightarrow$ semi-IND for a LaSS scheme.[11]

[11] The reduction just ignores the additional bit and forwards the challenge queries.

3.2 Our Constructions

In this section, we present our LaSS scheme. We consider two different instantiations of this construction, a PRF-based construction in the standard model and a construction in the ROM, which are both described in Fig. 9. The scheme in the standard model achieves semi-adaptive security and the scheme in the ROM adaptive security. An informal description of this scheme can be found in Section 1.3.

$\underline{\mathsf{Setup}(1^\lambda, s, n):}$	$\underline{\mathsf{ShareGen}(\mathsf{sk}_i, \ell):}$
For $i \in [n], j > i$:	Parse $\mathsf{sk}_i = (\{\mathsf{k}_{i,j}\}_{j\in[n]_0}, t_i)$
$\mathsf{k}_{i,j} = \mathsf{k}_{j,i} \leftarrow \{0,1\}^\lambda$	$v_{i,\ell} := \sum_{j\neq i}(-1)^{j<i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell) \in \mathbb{Z}_p$
If $s \neq 0$	$v_{i,\ell} := \sum_{j\neq i}(-1)^{j<i}\mathsf{H}(\mathsf{k}_{i,j}\|\ell) \in \mathbb{Z}_p$
$t_i \leftarrow \mathbb{Z}_p$ for all $i \in [n-1]$	$s_{i,\ell} := v_{i,\ell} + t_i$
$t_n := s - \sum_{i\in[n-1]} t_i$	Return $s_{i,\ell}$
$\mathsf{sk}_i = (\{\mathsf{k}_{i,j}\}_{j\in[n]}, t_i)$	$\underline{\mathsf{Reconstruct}(\{s_{i,\ell}\}_{i\in[n]}):}$
Return $\{\mathsf{sk}_i\}_{i\in[n]}$	Return $\sum_{i\in[n]} s_{i,\ell}$

Fig. 9: Our LaSS scheme in the Standard Model and the Random Oracle Model.

Correctness. The correctness of our scheme in the standard model follows from the generation of the $v_{i,\ell}$ values for all $i \in [n]$ under a specific label ℓ. We can see this by considering the following sum:

$$\sum_{i\in[n]} v_{i,\ell} = \sum_{i\in[n]}\sum_{j\neq i}(-1)^{j<i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell) = \sum_{i\in[n]}\sum_{j\neq i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell) - \sum_{j\neq i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell) = 0 \ .$$

Taking this into account, we can directly see what happens if the additional shares t_i are introduced:

$$\sum_{i\in[n]} v_{i,\ell} + t_i = \sum_{i\in[n]} v_{i,\ell} + \sum_{i\in[n]} t_i = 0 + \sum_{i\in[n]} t_i = s \ ,$$

which shows the correctness of our construction.

For the correctness of the scheme in the ROM, we can argue in the exact same way, i.e. we can show that all the keys $\mathsf{K}_{i,t}$ for all the users U_i with $i \in [n]$ together with the aggregator key $\mathsf{K}_{0,t}$ sum up to 0:

$$\sum_{i\in[n]} v_{i,\ell} = \sum_{i\in[n]}\sum_{j\neq i}(-1)^{j<i}\mathsf{H}(\mathsf{k}_{i,j}\|\ell) = \sum_{i\in[n]}\sum_{j\neq i}\mathsf{H}(\mathsf{k}_{i,j}\|\ell) - \sum_{j\neq i}\mathsf{H}(\mathsf{k}_{i,j}\|\ell) = 0 \ .$$

Together with the analysis from above for the shares t_i, the correctness of our scheme follows.

The correctness for the Add procedure follows accordingly.

Security. Now, we present the proof of security for both of the presented schemes. For the proof of security of the PRF-based construction we rely on a hybrid argument where the different game transitions are described in Fig. 10. For the proof of the security of the construction in the random oracle model, we can directly use the programmability of the random oracle.

For the scheme in the standard model, we make use of a technique from Abdalla et al. [ABG19] which has also been used in the subsequent work of Ciampi et al. [CSW20]. The main difficulty that we have to overcome in this proof is that the adversary is able to ask corruption queries adaptively. To ensure that a reduction to the security of the PRF still works, we use a hybrid strategy in which some, but not all, of the

corrupted users need to be guessed in advance and make use of the corruption bit that is submitted by the adversary.[12]

Theorem 3.4 (Semi-Security in the Standard Model). *Let* PRF *be an* IND *secure pseudorandom function, then the PSA scheme* $\mathsf{LaSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Reconstruct})$ *described in Fig. 9 is* semi-IND*-secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that:*

$$\mathsf{Adv}^{\text{semi-IND}}_{\mathsf{LaSS},\mathcal{A}}(\lambda) \leq (n+1)n(n-1)^2 \cdot \mathsf{Adv}^{\text{IND}}_{\mathsf{PRF},\mathcal{B}}(\lambda) \ .$$

Proof. To prove this theorem, we proceed via a hybrid argument, using the games described in Fig. 10. Note that G_0 corresponds to semi-$\text{IND}_0^{\mathsf{LaSS}}(\lambda, n, \mathcal{A})$, and $\mathsf{G}^\star_1$ corresponds to semi-$\text{IND}_1^{\mathsf{LaSS}}(\lambda, n, \mathcal{A})$. Thus, we have:

$$\mathsf{Adv}^{\text{semi-IND}}_{\mathsf{LaSS},\mathcal{A}}(\lambda, n) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}^\star_1}_{\mathcal{A}}(\lambda, n)| \ .$$

We describe the different intermediate games in more detail:

– **Game** $\mathsf{G}^\star_0$: The game is the same as the semi-IND_0 game, with the difference that the number of explicitly honest slots is guessed initially, by choosing a uniformly random $\kappa^\star \leftarrow \{0, \ldots, n\}$. In the case that the guess is incorrect, it ignores $\mathcal{A}$'s output α and outputs 0, instead. Since the guess is correct with probability $\frac{1}{n+1}$, we have

$$\mathsf{Win}^{\mathsf{G}^\star_0}_{\mathcal{A}}(\lambda, n) = \frac{1}{(n+1)} \cdot \mathsf{Win}^{\text{semi-IND}_0}_{\mathsf{LaSS}}(\lambda, n) \ .$$

– **Game** $\mathsf{G}^\star_1$: We change the distribution of the share generated by the QShare oracle in the case that $\kappa^\star \geq 2$. For these, the $v_{i,\ell^\star}$ values are computed as usual, but a share of a perfect $\kappa^\star$-out-of-$\kappa^\star$ secret sharing of 0 is added. We justify this transition using the security of the PRF and by guessing the explicitly honest slots. Since guessing the entire set of explicitly honest slots would incur an exponential security loss, we introduce the shares gradually. Starting with a 2-out-of-2 perfect secret sharing, then a 3-out-of-3 until we reach a $\kappa^\star$-out-of-$\kappa^\star$ secret sharing among all the queried slots. This gradual introduction happens via a hybrid argument that is described in Fig. 10. To go from one hybrid to another, we only require to guess a slot pair (i, j) (the first and the last slot to be revealed) correct and rely on the security of the PRF using the key $\mathsf{k}_{i,j}$. Namely, in Lemma 3.5, we show that there exists a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}^\star_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}^\star_1}_{\mathcal{A}}(\lambda, n)| \leq n(n-1)^2 \cdot \mathsf{Adv}^{\text{IND}}_{\mathsf{PRF},\mathcal{B}_0}(\lambda, n) \ .$$

Putting everything together, we obtain the theorem.

□

[12] Guessing all of the corrupted users would result in an exponential security loss in the reduction and an adaptively secure scheme.

$\underline{\mathsf{G}^\star_{0.k} \text{ for } k \in \{0,\dots,n\}:}$	We denote by $\{i_1,\dots,i_\kappa\}$ the set of explicitly honest slots in the order they are revealed, and we set $\theta := \min(\kappa^\star, k)$.
$\kappa^\star \leftarrow \{0,\dots,n\}$, for all $w \in \{2,\dots,\kappa^\star\}$, $u_w \leftarrow \mathbb{Z}_p$	
$s \leftarrow \mathcal{A}(1^\lambda)$	
$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{Setup}(1^\lambda, s, n)$	If $\theta \geq 2$ then do the following:
$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot),\mathsf{QShare}(\cdot,\cdot,\cdot)}(1^\lambda, s, n)$	– If $i = i_1$, then $s_{i,\ell^\star} := v_{i,\ell^\star} + \sum_{w=2}^{\theta} u_w$
Output α if Condition (*) is satisfied AND	– If $i = i_s$, for $w \in \{2,\dots,\theta\}$,
the guess $\kappa^\star$ is correct, or 0 otherwise.	then $s_{i,\ell^\star} := v_{i,\ell^\star} - u_w$
	– If $i = i_s$, for $w \in \{\theta+1,\dots,\kappa^\star\}$,
$\underline{\mathsf{QCor}(i):}$	then $s_{i,\ell^\star} := v_{i,\ell^\star}$
Return sk_i	– If $i = i_s$, for $w > \kappa^\star$, that means $k > \kappa^\star$,
	the guess was incorrect.
$\underline{\mathsf{QShare}(i,\ell^\star,b):}$	Ends the game and outputs 0.
Parse $\mathsf{sk}_i := \{\mathsf{k}_{i,j}\}_{j\in[n]}$,	If $\theta < 2$, then $s_{i,\ell^\star} := v_{i,\ell^\star}$.
$v_{i,\ell^\star} := \sum_{j\neq i}(-1)^{j<i}\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell^\star)$.	Return $s_{i,\ell^\star} + t_i$

Fig. 10: Games for the proof of Lemma 3.5. The guess $\kappa^\star$ is correct if it equals the size of the set of explicitly honest users.

Lemma 3.5 (Transition from $\mathsf{G}^\star_0$ to $\mathsf{G}^\star_1$). *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$|\mathsf{Win}^{\mathsf{G}^\star_0}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_1}_{\mathcal{A}}(\lambda,n)| \leq n(n-1)^2 \cdot \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{B}'}(\lambda) \ .$$

Proof. To prove that $\mathsf{G}^\star_0$ is indistinguishable from $\mathsf{G}^\star_1$ we need to apply a hybrid argument over the explicitly honest users by relying on the security of the PRF.

Using the definition of the games in Fig. 10 and the triangular inequality, we can see that

$$|\mathsf{Win}^{\mathsf{G}^\star_0}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_1}_{\mathcal{A}}(\lambda,n)| \leq \sum_{k=2}^{n} |\mathsf{Win}^{\mathsf{G}^\star_{0.k-1}}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_{0.k}}_{\mathcal{A}}(\lambda,n)| \ ,$$

where $\mathsf{G}^\star_0$ corresponds to game $\mathsf{G}^\star_{0.0}$ (and $\mathsf{G}^\star_{0.1}$) and whereas $\mathsf{G}^\star_1$ is identical to game $\mathsf{G}^\star_{0.n}$. Since $\mathsf{G}^\star_{0.0} = \mathsf{G}^\star_{0.1}$, we do not analyze the transition between these two games.

Now, we can bound the difference between each consecutive pair of games for every $k \in \{2,\dots,n\}$.

Lemma 3.6. *For every $k \in \{2,\dots,n\}$, there exists a PPT adversary $\mathcal{B}_k$ against the* IND *property of* PRF *such that*

$$|\mathsf{Win}^{\mathsf{G}^\star_{0.k-1}}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_{0.k}}_{\mathcal{A}}(\lambda,n)| \leq n(n-1) \cdot \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{B}_k}(\lambda) \ .$$

Proof. The proof for this transition works mainly as described in [ABG19].

We build an adversary $\mathcal{B}_k$ that simulates $\mathsf{G}^\star_{0.k-1+\beta}$ for $k \in \{2,\dots,n\}$ to $\mathcal{A}$ when interacting with the underlying $\mathrm{IND}^{\mathsf{PRF}}_\beta$ experiment.

As already mentioned above, if $\kappa^\star < 2$, the games $\mathsf{G}^\star_{0.k-1}$ and $\mathsf{G}^\star_{0.k}$ are the same. Therefore, we only consider the case where $\kappa^\star \geq 2$.

The adversary $\mathcal{B}_k$ starts by guessing the pair of the first and k'th honest slot $(i^\star, j^\star)$, by sampling random values $i^\star, j^\star \leftarrow [n]$, with $i^\star < j^\star$. Whenever $\mathcal{A}$ asks a share generation oracle query, $\mathcal{B}_k$ replies as described in Fig. 10 for every explicitly honest slot in the order they are revealed. Since $\mathcal{B}_k$ has guessed the first and k'th explicitly honest slot correctly, it knows how to answer the queries for every explicitly honest slot that is revealed in between. If it turns out that the guess of $\mathcal{B}_k$ is incorrect, the simulation ends and returns 0. If the guess is correct, we can rely on the security of the PRF on the key $\mathsf{k}_{i^\star,j^\star}$ and exchange the PRF evaluation with

a random function evaluation $\mathsf{RF}(\ell^\star)$. Then we argue that $\mathsf{RF}(\ell^\star)$ is identically distributed to $\mathsf{RF}(\ell^\star)+u_{s,\ell^\star}$ and therefore, since the former distribution corresponds to $\mathsf{G}^\star_{0.k-1}$ and the latter to game $\mathsf{G}^\star_{0.k}$, the computational indistinguishability between $\mathsf{G}^\star_{0.k}$ and $\mathsf{G}^\star_{0.k-1}$ follows. Since the guessing of the two honest slots happens with probability $\frac{2}{(n+1)n}$, we have a resulting security loss of $\frac{n(n-1)}{2}$, i.e. $\frac{2}{n(n-1)} \cdot |\mathsf{Win}^{\mathsf{G}^\star_{0.k-1}}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_{0.k}}_{\mathcal{A}}(\lambda,n)| \leq \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{B}_k}(\lambda) \Leftrightarrow |\mathsf{Win}^{\mathsf{G}^\star_{0.k-1}}_{\mathcal{A}}(\lambda,n) - \mathsf{Win}^{\mathsf{G}^\star_{0.k}}_{\mathcal{A}}(\lambda,n)| \leq \frac{n(n-1)}{2} \cdot \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{PRF},\mathcal{B}_k}(\lambda)$. Finally, we switch back from a random function evaluation $\mathsf{RF}(\ell^\star)$ to the PRF evaluation $\mathsf{PRF}_{\mathsf{k}_{i^\star,j^\star}}(\ell^\star)$ by relying on the security of the PRF on the key $\mathsf{k}_{i^\star,j^\star}$ a second time. This results in the advantage described in the lemma.

For every corruption query $\mathsf{QCor}(i)$, $\mathcal{B}_k$ just returns the corresponding secret key sk_i. □

□

The Add procedure for this scheme is the simple addition procedure, i.e. $\mathsf{Add}(s_{i,\ell}, x_{i,\ell}) \coloneqq s_{i,\ell} + x_{i,\ell}$. We can show that the Add procedure fulfills the linear homomorphism property.

Lemma 3.7 (Linear Homomorphism). *Let* $\mathsf{LaSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Reconstruct})$ *be the LaSS scheme described in Fig. 9 and* Add *the algorithm described above, then* Add *is a linear homomorphism.*

Proof. This lemma can be proven with a reduction to the one-time pad.

We build a PPT adversary $\mathcal{B}$ that outputs a share $s_{i,\ell}$ to $\mathcal{A}$ as a reply to a challenge query $x_{i,\ell}$, when interacting with the one-time pad in the $\text{IND-CPA}^{\mathsf{OTP}}_\beta$ experiment. We describe the behavior of adversary $\mathcal{B}$.

For the challenge query $x_{i,\ell}$ submitted by $\mathcal{A}$, the adversary $\mathcal{B}$ submits $(0, x_{i,\ell})$ as the challenge query to the one-time pad and receives as a reply the share $s_{i,\ell}$. For the case that challenger used the left challenge, i.e. 0, the output is a random share $s_{i,\ell}$ and for the case that the challenger encrypted the right value, i.e. $x_{i,\ell}$, it corresponds to the output of $\mathsf{Add}(s_{i,\ell}, x_{i,\ell})$.

This argument can be applied to every label $\ell \in \mathsf{Labels}$ and every slot $i \in [n]$, which proves the lemma. □

The security of the ROM construction follows immediately from the security for the scheme in Fig. 9, by just instantiating the PRF using the random oracle, i.e. $\mathsf{PRF}_{\mathsf{k}_{i,j}}(*) = \mathsf{H}[\mathsf{k}_{i,j} \| *]$.

To obtain stronger security bounds, we provide a security proof that relies on the programmability of the ROM.

Theorem 3.8 (Adaptive-Security in the Random Oracle Model). *Let* H *be a hash function, then the LaSS scheme* $\mathsf{LaSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{Reconstruct})$ *described in Fig. 9 is* ad-IND*-secure. Namely, when the hash function* H *is modeled as a random oracle, for any PPT adversary* $\mathcal{A}$ *it holds that:*

$$\mathsf{Adv}^{\text{ad-IND}}_{\mathsf{LaSS},\mathcal{A}}(\lambda,n) \leq \frac{q_{\mathsf{H}}}{2^\lambda} \ ,$$

where q_{H} *is the numbers of queries to the oracle* H*.*

Proof. In G_1, the values $v_{i,\ell}$, and therefore the resulting shares $s_{i,\ell}$ are generated uniformly at random on the fly. The random oracles are programmed to explain these values, when the adversary corrupts a new slot i or calls the share generation oracle QShare for the last remaining honest slot.

To complete the proof, we need to bound the probability of the event **Abort**, which comes from collisions.

Let us consider the q-th query $\mathsf{QCor}(i)$ and bound the probability of the event **Abort**. Let $j \notin \mathcal{CS} \setminus \{i\}$, we start by bounding the probability that HT contains a key of the form $\mathsf{k}_{i,j} \| \star$. Since $\mathsf{k}_{i,j}$ is drawn uniformly random and independently from $\{0,1\}^\lambda$, for any key of HT of the form $\star' \| \star$, the probability that $\star' = \mathsf{k}_{i,j}$ is exactly $1/2^\lambda$. Thus, the probability that HT_1 contains a key of the form $\mathsf{k}_{i,j} \| \star$ is $\frac{q_{\mathsf{H}}}{2^\lambda}$, with q_{H} the number of queries to H of the form $\star' \| \star$. (Even if keys of HT are also added by the challenger and not only when the adversary queries H, the keys added by the challenger can never create an abort, so we are ignoring them.)

By remarking that only the first query $\mathsf{QCor}(i)$ for a given i might abort (if the query happens for an already queried i, the same result gets returned) and by union bound, the probability for the execution of **Abort** is at most $\frac{q_{\mathsf{H}}}{2^\lambda}$. □

Games $\mathsf{G}_0, \mathsf{G}_1$:

HT is an empty array

If $\mathsf{HT}[z]$ does not exist, $\mathsf{HT}[z] \leftarrow \mathbb{Z}_p^m$

Return $\mathsf{HT}[z]$

G_β for $\beta \in \{0,1\}$:

$s \leftarrow \mathcal{A}(1^\lambda, n)$

$(\{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, s, n)$

$\alpha \leftarrow \mathcal{A}^{\mathsf{QCor}(\cdot),\mathsf{QShare}(\cdot,\cdot)}(1^\lambda, s, n)$

Output: α

$\mathsf{QCor}(i)$:

If already called for the same i,
return same answer

Add i to $\mathcal{CS}$

For all $j \notin \mathcal{CS}$,

Define $\mathsf{k}_{i,j} = \mathsf{k}_{j,i} \leftarrow \{0,1\}^\lambda$

Abort if HT contains a key of the form
$\mathsf{k}_{i,j}\|\star$ for $j \notin \mathcal{CS}$

For all previous queries $\mathsf{QShare}(i,\ell)$ for some time step ℓ,

Sample a random $j^* \leftarrow [n] \setminus \mathcal{CS}$

For all $j \in [n] \setminus (\mathcal{CS} \cup \{j^*\})$

$\mathsf{k}_{i,j,\ell} := \mathsf{H}(\mathsf{k}_{i,j}\|\ell)$

Define $\mathsf{k}_{i,j^*,\ell} := (-1)^{j^*<i}(v_{i,\ell} - \sum_{j\neq i,j^*}(-1)^{j<i}\mathsf{k}_{i,j,\ell})$

$\mathsf{HT}[\mathsf{k}_{i,j^*}\|\ell] := \mathsf{k}_{i,j^*,\ell}$

For all previous queries $\mathsf{QShare}(j,\ell)$ that have been asked for all $j \notin \mathcal{CS}$

For all $j \in \mathcal{CS}$

$v_{j,\ell} := \sum_{k\neq j}(-1)^{j<k}\mathsf{HT}[\mathsf{k}_{j,k}\|\ell]$

Sample a random $j^* \leftarrow [n] \setminus \mathcal{CS}$

For all $j \in [n] \setminus (\mathcal{CS} \cup \{j^*\})$

$\mathsf{k}_{i,j,\ell} := \mathsf{H}(\mathsf{k}_{i,j}\|\ell)$

Define $\mathsf{k}_{i,j^*,\ell} := (-1)^{j^*<i}(-\sum_{j\neq i} v_{j,\ell} - \sum_{j\neq i,j^*}(-1)^{j<i}\mathsf{k}_{i,j,\ell})$

$\mathsf{HT}[\mathsf{k}_{i,j^*}\|\ell] := \mathsf{k}_{i,j^*,\ell}$

$\mathsf{QShare}(i,\ell)$:

If $i \in \mathcal{CS}$

$v_{i,\ell} := \sum_{j\neq i}(-1)^{j<i}\mathsf{H}[\mathsf{k}_{i,j}\|\ell]$

$s_{i,\ell} := v_{i,\ell} + t_i$

Return $s_{i,\ell}$

If $\mathsf{QShare}(i,\ell)$ has already been queried

$s_{i,\ell} := v_{i,\ell} + t_i$

Return $s_{i,\ell}$

If $\mathsf{QShare}(j,\ell)$ has already been queried for all $j \notin \mathcal{CS}$

For all $j \in \mathcal{CS}$

$v_{j,\ell} := \sum_{k\neq 0}(-1)^{j<k}\mathsf{HT}[\mathsf{k}_{j,k}\|\ell]$

$v_{i,\ell} := -\sum_{j\neq i} v_{j,\ell}$

$s_{i,\ell} := v_{i,\ell} + t_i$

Return $s_{i,\ell}$

Define $v_{i,\ell} \leftarrow \{0,1\}^\lambda$

$s_{i,\ell} := v_{i,\ell} + t_i$

Return $s_{i,\ell}$

Fig. 11: Share generation queries QShare of the games for the proof of Theorem 3.8. Corruption queries QCor of the games for the proof of

The Add procedure of the construction in the ROM is the same as for the construction in the standard model. The proof is the same as for Lemma 3.7.

4 PSA in the Encrypt-Once Model

In the following sections, we prove the correctness of this construction and give an overview of the security proof.

$\underline{\mathsf{Setup}(1^\lambda, n):}$
$\{\mathsf{sk}_i\}_{i\in[n]_0} \leftarrow \mathsf{LaSS.Setup}(1^\lambda, 0, n+1)$
For $i \in [n]_0, j > i$:
Return $\{\mathsf{sk}_i\}_{i\in[n]_0}$
$\underline{\mathsf{Enc}(\mathsf{sk}_i, x_i \in \mathbb{Z}_p, t):}$
$s_{i,t} \coloneqq \mathsf{LaSS.ShareGen}(\mathsf{sk}_i, t)$
Return $\mathsf{ct}_{i,t} \coloneqq \mathsf{LaSS.Add}(s_{i,t}, x_i) \in \mathbb{Z}_p$
$\underline{\mathsf{Dec}(\mathsf{sk}_0, \{\mathsf{ct}_{i,t}\}_{i\in[n]}):}$
$s_{0,t} \coloneqq \mathsf{LaSS.ShareGen}(\mathsf{sk}_0, t)$
Return $\mathsf{LaSS.Reconstruct}(s_{0,t} \cup \{\mathsf{ct}_{i,t}\}_{i\in[n]})$

Fig. 12: The LaSS-based PSA scheme.

4.1 Correctness

The correctness of our scheme follows from the correctness and the linear homomorphism property of the LaSS scheme:

$$\mathsf{Reconstruct}(s_{0,t} \cup \{\mathsf{ct}_{i,t}\}) = \mathsf{Reconstruct}(s_{0,t} \cup \{\mathsf{Add}(s_{i,t}, x_{i,t})\}) = 0 + \sum_{i\in[n]} x_{i,t} = \sum_{i\in[n]} x_{i,t} \ .$$

4.2 Security

In this section, we give an overview of the proof of security for the compiler described in Fig. 12. For the proof of security, we rely on a hybrid argument, where the different game transitions are described in Fig. 13

Game	$\mathsf{ct}_{i,t}$	Justification
G_0	$s_{i,t} \coloneqq \mathsf{ShareGen}(\mathsf{sk}_i, t)$ $\mathsf{ct}_{i,t} \coloneqq \mathsf{Add}(s_{i,t}, x^0_{i,t})$	
G_1	$\boxed{s_{i,t} \leftarrow \mathbb{Z}_p}$ $\mathsf{ct}_{i,t} \coloneqq \mathsf{Add}(s_{i,t}, x^0_{i,t})$	semi-IND security of LaSS
G_2	$s_{i,t} \leftarrow \mathbb{Z}_p$ $\mathsf{ct}_{i,t} \coloneqq \mathsf{Add}(s_{i,t}, \boxed{x^1_{i,t}})$	information-theoretic argument
G_3	$\boxed{s_{i,t} \coloneqq \mathsf{ShareGen}(\mathsf{sk}_i, t)}$ $\mathsf{ct}_{i,t} \coloneqq \mathsf{Add}(s_{i,t}, x^1_{i,t})$	semi-IND security of LaSS

Fig. 13: Overview of the games to prove the security of the PSA scheme using a semi-IND secure LaSS.

Theorem 4.1 (AO-Security). *Let* LaSS *be a* semi-IND *secure labeled secret sharing scheme, then the PSA scheme* $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 12 is* AO*-secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that:*

$$\mathsf{Adv}^{\mathrm{AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{LaSS},\mathcal{B}}(\lambda, n+1) \ .$$

Proof. To prove this theorem, we proceed via a hybrid argument, using the games described in Fig. 13. Note that G_0 corresponds to $\mathrm{AO}_0^{\mathsf{PSA}}(\lambda, n, \mathcal{A})$, and G_3 corresponds to $\mathrm{AO}_1^{\mathsf{PSA}}(\lambda, n, \mathcal{A})$. Thus, we have:

$$\mathsf{Adv}^{\mathrm{AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(\lambda, n)| \ .$$

We describe the different intermediate games in more detail:

- **Game G_1**: We change the way the shares of each client are generated. Instead of executing the $\mathsf{ShareGen}$ every client generates a perfect share of a secret sharing of 0. We justify this transition using the semi-adaptive security of LaSS together with the fact that we can determine the bit b for every slot, by checking if the first challenges are different ($b = 0$) or the same ($b = 1$). Namely, in Lemma 4.2, we show that there exists a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{LaSS},\mathcal{B}_0}(\lambda, n+1) \ .$$

- **Game G_2**: We change the generation of the ciphertext from an encryption of x_i^0 to an encryption of x_i^1. The transition from G_1 to G_2 is justified by an information-theoretic argument using the Add procedure for all the different slots $i \in [n]$. In more detail, we prove the transition by relying on the perfect security of several instances of the one-time pad. Namely, in Lemma 4.3, we show that

$$|\mathsf{Win}^{\mathsf{G}^\star_2}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}^\star_3}_{\mathcal{A}}(\lambda, n)| = 0 \ ,$$

  for all adversaries $\mathcal{A}$.
- **Game G_3**: The transition from G_2 to G_3 is almost symmetric to the transition from G_0 to G_1, justified by the semi-adaptive security of LaSS. Namely, it can be proven as in Lemma 4.2 that there exists a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\mathrm{IND}}_{\mathsf{LaSS},\mathcal{B}_0}(\lambda, n) \ .$$

Putting everything together, we obtain the theorem. □

Lemma 4.2 (Transition from G_0 to G_1). *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that*

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n)| \leq \mathsf{Adv}^{\text{semi-IND}}_{\mathsf{LaSS},\mathcal{B}_0}(\lambda, n+1)$$

Proof. We prove this theorem by construction an adversary $\mathcal{B}$ against the semi-IND security game.

Whenever $\mathcal{A}$ asks a left-or-right oracle query $(i, x^0_{i,t}, x^1_{i,t}, t)$, $\mathcal{B}$ sets $b = 0$ if $x^0_{i,t} \neq x^1_{i,t}$ and $b = 1$ otherwise. $\mathcal{B}$ then submits (i, t, b) to its share generation oracle and receives $s_{i,t}$ as a reply. It computes $\mathsf{ct}_{i,\ell} \coloneqq \mathsf{Add}(s_{i,t}, x^0_{i,t^\star})$ and sends it as a reply to $\mathcal{A}$.

For every compromise query for i asked by $\mathcal{A}$, $\mathcal{B}$ forwards it to its corruption oracle, receives sk_i and sends it to $\mathcal{A}$.

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. This concludes the proof of the lemma. □

Lemma 4.3 (Transition from G_1 to G_2). *For any adversary $\mathcal{A}$ it holds that*

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(\lambda, n) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(\lambda, n)| = 0$$

Proof. This theorem follows directly from the linear homomorphism property of the LaSS scheme. In more detail, the linear homomorphism property states that for any value $x_{i,t}$ the values $s_{i,t}$ and $\mathsf{Add}(s_{i,t}, x_{i,t})$ are identically distributed for all $i \in [n]$. This also implies that $s_{i,t}$ is identically distributed to $\mathsf{Add}(s_{i,t}, x^0_{i,t})$ and $\mathsf{Add}(s_{i,t}, x^1_{i,t})$, which proves the theorem. □

5 PSA in the Many-Times Model

In this section, we first introduce a stronger security definition for PSA that allows for multiple left-or-right oracle queries. Second, we show that this security definition can be realized using functional encryption in the setting with and without time steps.

5.1 Stronger Security for PSA

The stronger notion of AO security that we present in this section allows an adversary to ask multiple left-or-right queries under different time steps, as well as queries to the encryption oracle on the same time steps that are queried to the left-or-right oracle. We also get rid of the restriction that an adversary has to query the left-or-right oracle for a specific determined subset of users.

Definition 5.1 (Stronger Security for PSA). *Let* PSA *be a PSA scheme and* $\mathcal{T}$ *a set of time steps. We define the experiment* $\text{str-AO}_\beta^{\mathsf{PSA}}$ *in Fig. 14, where the oracles are defined as:*

- **Compromise oracle** $\mathsf{QComp}(i)$: *works as in Definition 2.2.*
- **Left-Right oracle** $\mathsf{QLeftRight}(i, x_i^0, x_i^1, t)$: *Outputs* $\mathsf{ct}_{i,t} = \mathsf{Enc}(\mathsf{sk}_i, x_i^\beta, t)$ *on a query* (i, x_i^0, x_i^1, t).
- **Encryption oracle** $\mathsf{QEnc}(i, x_i, t)$: *as in Definition 2.2.*

and where Condition (*) *holds if all the following conditions hold:*

- *If* $i \in \mathcal{CS}$ *(i.e., user* i *is compromised): for any query* $\mathsf{QLeftRight}(i, x_i^0, x_i^1, t), x_i^0 = x_i^1$.
- *If* $0 \in \mathcal{CS}$ *(i.e., the aggregator is compromised): for any time step* $t \in \mathcal{T}$*, for any family of queries* $\{\mathsf{QLeftRight}(i, x_i^0, x_i^1, t)$ *or* $\mathsf{QEnc}(i, x_i, t)\}_{i \in [n] \setminus \mathcal{CS}}$*, for any family of inputs* $\{x_i \in \mathcal{X}_i\}_{i \in \mathcal{CS}}$*, we define* $x_i^0 = x_i^1 = x_i$ *for any slot* $i \in \mathcal{CS}$ *and any slot queried to* $\mathsf{QEnc}(i, x_i, t)$*, and we require that:*

$$\sum_{i \in [n]} x_i^0 = \sum_{i \in [n]} x_i^1 \ .$$

We insist that if one index $i \notin \mathcal{CS}$ *is not queried there is no restriction.*

We define the advantage of an adversary $\mathcal{A}$ *in the following way:*

$$\mathsf{Adv}^{\text{str-AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) = \left| \Pr[\text{str-AO}_0^{\mathsf{PSA}}(\lambda, n, \mathcal{A}) = 1] - \Pr[\text{str-AO}_1^{\mathsf{PSA}}(\lambda, n, \mathcal{A}) = 1] \right| \ .$$

An aggregator oblivious encryption scheme PSA *is strong-AO-secure, if for any* n*, for any polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:* $\mathsf{Adv}^{\text{str-AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) \leq \text{negl}(\lambda)$.

$\underline{\textbf{str-AO}_\beta^{\mathsf{PSA}}(\lambda, n, \mathcal{A})}$

$\{\mathsf{sk}_i\}_{i \in [n]_0} \leftarrow \mathsf{Setup}(1^\lambda, n)$

$\alpha \leftarrow \mathcal{A}^{\mathsf{QComp}(\cdot), \mathsf{QLeftRight}(\cdot,\cdot,\cdot,\cdot), \mathsf{QEnc}(\cdot,\cdot,\cdot)}(1^\lambda, n)$

Output: α if Condition (*) is satisfied, or a uniform bit otherwise

Fig. 14: Stronger AO Security Game.

Remark 5.2 (On the Number of Challenge Queries). In the case of a corrupted aggregator, the number of linearly independent challenge queries the adversary is allowed to ask is defined by the dimension of the input of each individual party. In the setting where every user encrypts a scalar, which is the case in the standard definition of PSA, every further challenge query that the adversary asks for a user needs to be linearly dependent to the first query that the adversary has asked for this user. In more detail, let $(x_i^{1,0}, x_i^{1,1})$ denote the first challenge query for the user U_i than it must hold for every further challenge query $(x_i^{j,0}, x_i^{j,1})$ asked for U_i that $x_i^{1,0} - x_i^{1,1} = x_i^{j,0} - x_i^{j,1}$. For the adapted setting where every user U_i encrypts a vector $\boldsymbol{x}_i$ of dimension m instead of scalar, the adversary can ask up to m linearly independent challenge queries, where the only restriction that is enforced on the queries is that $\sum_{k \in [n]} x_{i,k}^{1,0} - \sum_{k \in [n]} x_{i,k}^{1,1} = \sum_{k \in [n]} x_{i,k}^{j,0} - \sum_{k \in [n]} x_{i,k}^{j,1}$ where $(\boldsymbol{x}_i^{1,0}, \boldsymbol{x}_i^{1,1})$, with $\boldsymbol{x}_i^{1,b} = (x_{i,1}^{1,b}, \ldots, x_{i,m}^{1,b})$ for $b \in \{0,1\}$, is the first challenge query asked by the adversary for user U_i and $(\boldsymbol{x}_i^{j,0}, \boldsymbol{x}_i^{j,1})$, with $\boldsymbol{x}_i^{j,b} = (x_{i,1}^{j,b}, \ldots, x_{i,m}^{j,b})$ for $b \in \{0,1\}$, the j-th query asked by the adversary for user U_i. Note that in the setting with time steps this restriction applies for every time step separately.

Since in the scalar case for the stronger notion of AO-security with time steps only one challenge query per label is allowed, this notion is equivalent to the encrypt-once security.

After the general discussion on the number of challenge queries, we concretely consider the security notion of [BGZ18].

Remark 5.3 (On the Security of LaPS [BGZ18]). Since the encryption procedure of the LaPS scheme presented in [BGZ18], takes as an input a scalar x_i and since the scheme does not rely on time steps, it follows from the observation in Remark 5.2 that only a single challenge query for linearly independent messages is allowed. To highlight why this is the case, we describe a concrete attack in this setting when two challenge queries of linearly independent messages are allowed, i.e. $(x_{i^*}^{1,0}, x_{i^*}^{1,1}), (x_{i^*}^{2,0}, x_{i^*}^{2,1})$ with $x_{i^*}^{1,0} - x_{i^*}^{1,1} \neq x_{i^*}^{2,0} - x_{i^*}^{2,1}$, for a specific user U_{i^*}. The attack works as follows:

1. The adversary compromises the aggregator, i.e. it queries the compromise oracle $\mathsf{QComp}(i)$ for $i = 0$ and receives sk_0.
2. The adversary determines a random subset of users $U \subset [n]$, with $i^* \in U$ and submits a set of challenge queries $(x_i^{1,0}, x_i^{1,1})_{i \in U}$ with $\sum_{i \in U} x_i^{1,0} = \sum_{i \in U} x_i^{1,1}$ for which it receives as a reply $(\mathsf{ct}_i^1)_{i \in U}$.
3. The adversary submits the challenge query $(x_{i^*}^{2,0}, x_{i^*}^{2,1})$, which is linearly independent from the query $(x_{i^*}^{1,0}, x_{i^*}^{1,1})$ for which it receives as a reply $\mathsf{ct}_{i^*}^2$.
4. The adversary computes $\mathsf{Dec}(\mathsf{sk}_0, \{\mathsf{ct}_i^1\}_{i \in U \setminus \{i^*\}} \cup \{\mathsf{ct}_{i^*}^2\})$ and if it is equal to $x_{i^*}^{2,0} + \sum_{i \in U \setminus \{i^*\}} x_i^{1,0}$ it outputs 0, otherwise it outputs 1.

The security model that only allows for a single linearly independent challenge query is equivalent to the encrypt-once model with only a single time-step, and therefore it is weaker than the encrypt-once model. Moreover, as already mentioned in the introduction, we do not need to rely on a computational assumption to construct a scheme that fulfills this one-time security definition, as done in [BGZ18], since the information-theoretic construction in Section 1.3 on Page 3 already suffices.

5.2 Constructing PSA from Inner-Product MCFE

We formalize the generic construction of a PSA scheme that fulfills our strong AO security notion from inner-product MCFE in Fig. 15, as already observed by Libert and Titiu [LT19, Page 8-9]. To allow for several linearly independent challenge queries (see Remark 5.2), we instantiate our construction using an MCFE scheme that allows a user to encrypt a vector of dimension m ([ABM+20, ABG19, LT19, ACF+18, AGRW17, CDG+18b] fulfill this property) instead of a scalar. The security proof of this construction is straightforward and for completeness it can be found in Appendix A.

$\underline{\mathsf{Setup}(1^\lambda, n):}$
$(\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{msk}) \leftarrow \mathsf{Setup}'(1^\lambda, n)$
$\mathsf{sk}_0 \leftarrow \mathsf{KeyGen}'(\mathsf{msk}, \mathbf{1})$, $\mathbf{1} = (1, \ldots, 1) \in \mathbb{Z}_p^{mn}$
Return $\{\mathsf{sk}_i\}_{i \in [n]_0}$
$\underline{\mathsf{Enc}(\mathsf{sk}_i, \boldsymbol{x}_i \in \mathbb{Z}_p^m, t):}$
$\mathsf{ct}_{i,t} = \mathsf{Enc}'(\mathsf{sk}_i, \boldsymbol{x}_i, t)$
Return $\mathsf{ct}_{i,t}$
$\underline{\mathsf{Dec}(\mathsf{sk}_0, \{\mathsf{ct}_{i,t}\}_{i \in [n]}):}$
Return $\mathsf{Dec}'(\mathsf{sk}_0, \{\mathsf{ct}_{i,t}\}_{i \in [n]})$

Fig. 15: Compiler from an any-IND-secure MCFE scheme MCFE with labels into an PSA scheme that is str-AO-secure.

Theorem 5.4. *Let* $\mathsf{MCFE} = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *be an* any-IND*-secure MCFE scheme with labels* Labels *for a family of functions* $\mathcal{F}$*. Then the PSA scheme* $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *scheme described in Fig. 15 is* str-AO*-secure PSA scheme for time steps* $\mathcal{T} = \mathsf{Labels}$*. Namely, for all PPT adversaries* $\mathcal{A}$ *there exists a PPT adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{str-AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{any-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) \ .$$

The construction described in Fig. 15 can be adapted to any inner-product functionality by defining the aggregator key as a functional key for a different vector $\boldsymbol{y} \in \mathbb{Z}_p^m$ than the all one vector. This allows the aggregator to compute richer functionalities than the sum.

6 Implementation

In this section, we present the implementation results of our schemes, the LaSS-based scheme and the MCFE-based scheme, and a comparison with the results of Becker et al. [BGZ18]. The implementation is (anonymously) available at `https://anonymous.4open.science/r/aa436f15-e630-44b7-85ca-1dd2103ae3dc`.

6.1 Encrypt-Once Model

For the implementation of the LaSS-based construction (Section 4), we use AES as the instantiation for the PRF. For the construction in the ROM, we use SHA-3 as an instantiation for the hash function. The LaSS-based schemes are implemented in Go language using its native Crypto library for the AES and the SHA-3 implementation with keys of size 256 bits and benchmarked on a laptop with Intel Core i5-6200 2.3GHz processor.[13] We compare the performance of our scheme with the evaluation times reported in [BGZ18] for the LaPS scheme with 80 bit security[14] and for the SLAP scheme [TKGJ20] with 128 bit security .

	AES		SHA-3		LaPS [BGZ18]		SLAP [TKGJ20]	
p	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
2^2	0.539	0.509	1.783	1.654	3.576	1.868	n/a	n/a
2^{16}	0.539	0.509	1.783	1.654	3.724	1.964	1.17	3.26
2^{64}	0.539	0.509	1.783	1.654	n/a	n/a	n/a	n/a

Table 1: Comparison for $n = 1000$ of our PRF-based PSA scheme, instantiated using AES and SHA-3. Times given in ms.

In Table 1, we present our running time together with the running time of the LaPS scheme [BGZ18] and the SLAP scheme [TKGJ20], which, to the best of our knowledge, are the only other implemented PSA schemes. The table shows that the encryption and decryption times of our LaSS-based construction, instantiated using AES, are much less than the encryption and decryption times of the LaPS and SLAP scheme. In addition to that, the encryption and decryption time of the LaSS-based scheme is independent of the size of the underlying plaintext space. This is not the case for the LaPS and SLAP scheme, since a larger plaintext space requires larger LWE parameters and therefore results in longer encryption and decryption times.

The evaluation times of the LaPS and SLAP scheme are dependent on the size of the plaintext space but they do not seem to be dependent on the number of users in the system. This is due to the fact that the size of the secret key of every user is constant, and overall linear, in the number of users in the system. On the other hand, the secret key size of the LaSS-based construction is linear, and overall quadratic, in the number

[13] To allow for a better comparison with the results of Becker et al. [BGZ18] and Takeshita et al. [TKGJ20], we also implement our scheme using the Laplace distribution.

[14] Note that their scheme was implemented in C++ and benchmarked using slightly better hardware which only boosts its performance.

of users in the system. This results in a dependence between the encryption and decryption time and the number of users. Therefore we present additional performance evaluations depending on the number of users in Table 2.

	AES		SHA-3	
n	Enc.	Dec.	Enc.	Dec.
10^2	0.057	0.043	0.158	0.137
10^3	0.539	0.509	1.868	1.654
10^4	7.418	6.906	18.920	17.323

Table 2: Performance comparison for $p = 2^{64}$ of our PRF-based PSA scheme, instantiated using AES and SHA-3. Times given in ms.

6.2 Many-Times Model

For the implementation of the MCFE-based PSA scheme, we used the implementations of the GoFe library [GOF].[15] The 80-bit security is achieved by using a standard choice of 2048-bit groups in which the DDH assumption or the DCR assumption is believed to be hard. For the LWE instantiation, the parameters are chosen as described in the work of Agrawal et al. [ALS16].

	DDH		Paillier		LWE	
p	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
2^2	24.83	12798	39.06	47342	673	2355
2^{16}	24.83	12802	39.06	47342	80245	220112
2^{64}	24.83	n/a	39.06	47342	n/a	n/a

Table 3: Comparison for $n = 1000$ of our MCFE-based construction instantiated using DDH, Pallier and LWE. Times given in ms.

Table 3 shows that the performance of the MCFE-based constructions is much worse than the performance of the PRF-based construction. This is due to the more complex structure of MCFE schemes compared to the simple PRF-based construction. For the DDH instantiation, it is also worth mentioning that a discrete logarithm computation is required, which takes approximately 3ms for $p = 2^{16}$ and 1s for $p = 2^{32}$, using the baby-step giant-step algorithm. For $p = 2^{64}$, this computation is out of reach for a laptop. Since the MCFE-based construction is black-box and several inner-product MCFE schemes have recently been proposed, it is likely that more efficient inner-product MCFE schemes will be available in the future. Using these schemes as the instantiation for our MCFE-based construction will result in better performance.

7 Extensions

In this section, we highlight some extensions that are easily achievable by the presented schemes.

Extended Functionality. The presented LaSS scheme can easily be adapted to allow for a multiplicative secret sharing of 1

[15] More details on the library can also be found in [MSH+19].

The shares in this scheme would then be computed as $s_{i,\ell} := \prod_{j \neq i}(\mathsf{PRF}_{\mathsf{k}_{i,j}}(\ell))^{(-1)^{j<i}}$ (or $s_{i,\ell} := \prod_{j \neq i}(\mathsf{H}(\mathsf{k}_{i,j} \| \ell))^{(-1)^{j<i}}$ for the scheme in the ROM) and the Add procedure using x_i would be defined as $\mathsf{Add}(s_{i,\ell}, x_i) := s_{i,\ell} \cdot x_i$. The presented proof of security for the additive scheme can be straightforwadly adapted to the presented multiplicative one.

Instantiating our black-box compiler (Fig. 12) using the described scheme results in a PSA scheme that allows for the multiplication of the plaintexts of the users.

Such a straightforward adaption is not possible for every PSA scheme.

Vector Inputs. We can also extend our LaSS scheme such that the shares are vectors of dimension m ($s_{i,\ell} \in \mathbb{Z}_p^m$) instead of scalars ($s_{i,\ell} \in \mathbb{Z}_p$). This would result in the fact that the Add procedure can also take vectors $\boldsymbol{x}$ of dimension m as an input. Using such a LaSS scheme to instantiate our compiler in Fig. 12 would allow for the construction of a PSA scheme that is able to handle vectors.

To still achieve (semi-)adaptive security for the LaSS construction, we need to replace the PRF, or respectively the hash function, with a PRF, or a hash function, that has $\mathbb{Z}_p^m$ as a target space.

Public Aggregation. To allow for public aggregation in the LaSS-based construction, we only need to establish shared keys between the different users that encrypt messages. The decryption function is then the reconstruction procedure of the LaSS scheme taking all the ciphertexts of the different users as an input.

For the MCFE-based construction, we can achieve public aggregation by publishing the functional key for the one vector $\mathbf{1}$, this enables every user in the system to decrypt a set of ciphertexts.

Decentralization and Dynamic Joins and Leaves. Another possible extension is the decentralization of the setup procedure. To achieve this in the LaSS-based construction we simply execute several key exchange protocols between the different users. This ensure that every user is in the possession of a shared key with all the other users but their exists no trusted party that has knowledge of all the generated keys[16].

For the MCFE-based PSA scheme we can achieve decentralization by using a decentralized multi-client functional encryption (DMCFE) scheme [ABG19, LT19, CDG+18a, CDG+18b, ABKW19] as the underlying MCFE scheme. In a DMCFE scheme the setup is decentralized and the generation of functional keys require the participation of all the users in the system. Combining these two aspects results in a decentralized setup in our PSA scheme. By relying on a dynamic decentralized multi-client functional encryption (DDMCFE) scheme [CDSG+20], it is even possible to allow for dynamic joins and leaves. This has already been mentioned in [CDSG+20].

8 Conclusion

In this work, we showed how to construct a simple PSA scheme that relies on minimal assumptions and achieves the standard notion of AO-security. We presented stronger notions of AO-security and showed how to construct schemes that fulfill these stronger notions based on MCFE. In the process, we revisited the work by Becker et al. [BGZ18] and pointed out important shortcomings in their scheme and security analysis, which does not provide security guarantees when users are allowed to encrypt multiple messages. Finally, we compared an implementation of our results with other existing PSA schemes. Taking all into account, it shows that our work significantly improves over prior work in terms of practical efficiency, security and complexity assumptions.

Acknowledgments

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme FENTEC (Grant Agreement no. 780108).

[16] In the case that there exists a single party that generates the shared keys, this entity can also break the privacy of all the users in the system.

References

ABDP15. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015*, *LNCS* 9020, pages 733–751. Springer, Heidelberg, March / April 2015. (Page 1.)

ABG+13. P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/2013/689. (Page 1.)

ABG19. M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In *ASIACRYPT 2019, Part III*, *LNCS* 11923, pages 552–582. Springer, Heidelberg, December 2019. (Pages 1, 4, 5, 6, 9, 13, 15, 21, and 24.)

ABKW19. M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In *PKC 2019, Part II*, *LNCS* 11443, pages 128–157. Springer, Heidelberg, April 2019. (Pages 4, 5, 6, 9, and 24.)

ABM+20. M. Abdalla, F. Bourse, H. Marival, D. Pointcheval, A. Soleimanian, and H. Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *SCN 20*, *LNCS* 12238, pages 525–545. Springer, Heidelberg, September 2020. (Pages 1, 5, and 21.)

ÁC11. G. Ács and C. Castelluccia. I have a dream! (differentially private smart metering). In *Information Hiding*, pages 118–132, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Pages 4 and 5.)

ACF+18. M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *CRYPTO 2018, Part I*, *LNCS* 10991, pages 597–627. Springer, Heidelberg, August 2018. (Pages 1, 4, 5, 12, and 21.)

AGRW17. M. Abdalla, R. Gay, M. Raykova, and H. Wee. Multi-input inner-product functional encryption from pairings. In *EUROCRYPT 2017, Part I*, *LNCS* 10210, pages 601–626. Springer, Heidelberg, April / May 2017. (Pages 1, 5, 6, and 21.)

ALS16. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 333–362. Springer, Heidelberg, August 2016. (Pages 1 and 23.)

BCFG17. C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 67–98. Springer, Heidelberg, August 2017. (Page 1.)

BGV12. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, January 2012. (Page 6.)

BGZ17. D. Becker, J. Guajardo, and K.-H. Zimmermann. Somar: Privacy-preserving social media advertising architecture. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, WPES '17, page 21–30, New York, NY, USA, 2017. Association for Computing Machinery. (Page 6.)

BGZ18. D. Becker, J. Guajardo, and K.-H. Zimmermann. Revisiting private stream aggregation: Lattice-based PSA. In *NDSS 2018*. The Internet Society, February 2018. (Pages 3, 4, 6, 8, 21, 22, and 24.)

BJK15. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 470–491. Springer, Heidelberg, November / December 2015. (Page 1.)

BJL16. F. Benhamouda, M. Joye, and B. Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3), March 2016. (Pages 4 and 6.)

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011. (Page 1.)

CDG+18a. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 703–732. Springer, Heidelberg, December 2018. (Pages 1, 4, and 24.)

CDG+18b. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018. https://eprint.iacr.org/2018/1021. (Pages 1, 5, 6, 9, 21, and 24.)

CDSG+20. J. Chotard, E. Dufour-Sans, R. Gay, D. H. Phan, and D. Pointcheval. Dynamic decentralized functional encryption. In *CRYPTO 2020, Part I*, *LNCS* 12170, pages 747–775. Springer, Heidelberg, August 2020. (Pages 1, 5, and 24.)

CSS12. T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *FC 2012*, *LNCS* 7397, pages 200–214. Springer, Heidelberg, February / March 2012. (Pages 5 and 6.)

CSW20. M. Ciampi, L. Siniscalchi, and H. Waldner. Multi-client functional encryption for separable functions. Cryptology ePrint Archive, Report 2020/219, 2020. https://eprint.iacr.org/2020/219. (Page 13.)

FV12. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144. (Page 6.)

Gay20. R. Gay. A new paradigm for public-key functional encryption for degree-2 polynomials. In *PKC 2020, Part I*, *LNCS* 12110, pages 95–120. Springer, Heidelberg, May 2020. (Page 1.)

GGG+14. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In *EUROCRYPT 2014*, *LNCS* 8441, pages 578–602. Springer, Heidelberg, May 2014. (Pages 1 and 8.)

GGH+13. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. (Page 1.)

GGHZ16. S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Functional encryption without obfuscation. In *TCC 2016-A, Part II*, *LNCS* 9563, pages 480–511. Springer, Heidelberg, January 2016. (Page 1.)

GGM86. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986. (Page 10.)

GOF. Gofe - functional encryption library. (Page 23.)

HPS98. J. Hoffstein, J. Pipher, and J. H. Silverman. Ntru: A ring-based public key cryptosystem. In *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. (Page 6.)

JK12. M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *PETS 2012*, *LNCS* 7384, pages 221–238. Springer, Heidelberg, July 2012. (Pages 5 and 6.)

JKD12. M. Jawurek, F. Kerschbaum, and G. Danezis. Privacy technologies for smart grids - a survey of options. Technical Report MSR-TR-2012-119, Microsoft, November 2012. (Page 6.)

JL13. M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *FC 2013*, *LNCS* 7859, pages 111–125. Springer, Heidelberg, April 2013. (Page 6.)

KDK11. K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS 2011*, *LNCS* 6794, pages 175–191. Springer, Heidelberg, July 2011. (Pages 4 and 5.)

KLM+18. S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu. Function-hiding inner product encryption is practical. In *SCN 18*, *LNCS* 11035, pages 544–562. Springer, Heidelberg, September 2018. (Page 1.)

LPR10. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, *LNCS* 6110, pages 1–23. Springer, Heidelberg, May / June 2010. (Page 6.)

LSL+13. M. Lu, Z. Shi, R. Lu, R. Sun, and X. S. Shen. Pppa: A practical privacy-preserving aggregation scheme for smart grid communications. In *2013 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 692–697, 2013. (Page 6.)

LT19. B. Libert and R. Titiu. Multi-client functional encryption for linear functions in the standard model from LWE. In *ASIACRYPT 2019, Part III*, *LNCS* 11923, pages 520–551. Springer, Heidelberg, December 2019. (Pages 1, 4, 5, 21, and 24.)

MSH+19. T. Marc, M. Stopar, J. Hartman, M. Bizjak, and J. Modic. Privacy-enhanced machine learning with functional encryption. In *ESORICS 2019, Part I*, *LNCS* 11735, pages 3–21. Springer, Heidelberg, September 2019. (Page 23.)

O'N10. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556. (Page 1.)

RZL+13. Ruixue Sun, Zhiguo Shi, R. Lu, Min Lu, and X. Shen. Aped: An efficient aggregation protocol with error detection for smart grid communications. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 432–437, 2013. (Page 6.)

SCR+11. E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*. The Internet Society, February 2011. (Pages 1, 2, 5, 6, 7, and 8.)

Sha01. C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 2001. (Page 11.)

SSL+15. Z. Shi, R. Sun, R. Lu, L. Chen, J. Chen, and X. Sherman Shen. Diverse grouping-based aggregation protocol with error detection for smart grid communications. *IEEE Transactions on Smart Grid*, 6(6):2856–2868, Nov 2015. (Page 6.)

TKGJ20. J. Takeshita, R. Karl, T. Gong, and T. Jung. Slap: Simple lattice-based private stream aggregation protocol. Cryptology ePrint Archive, Report 2020/1611, 2020. https://eprint.iacr.org/2020/1611. (Pages 6 and 22.)

VA18. F. Valovich and F. Aldà. Computational differential privacy from lattice-based cryptography. In *Number-Theoretic Methods in Cryptology*, pages 121–141, Cham, 2018. Springer International Publishing. (Pages 6 and 8.)

Wat15. B. Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 678–697. Springer, Heidelberg, August 2015. (Page 1.)

Wic15. D. Wichs. Lecture 1: Perfect secrecy and statistical authentication. In *CS 7880 Graduate Cryptography*, 2015. (Page 11.)

A Proof of Theorem 5.4

Proof. We construct an adversary $\mathcal{B}$ against the any-IND security of the scheme MCFE.

If $\mathcal{A}$ asks a query $\mathsf{QComp}(i)$, $\mathcal{B}$ distinguishes between two different cases, depending on the user i queried to the oracle:

1. For $i \in [n]$, $\mathcal{B}$ asks a query $\mathsf{QCor}'(i)$ to its corruption oracle and obtains the key sk_i, which it forwards to $\mathcal{A}$.
2. If $i = 0$, $\mathcal{B}$ asks a query $\mathsf{QKeyG}'(\mathbf{1})$, with $\mathbf{1} = (1, \ldots, 1)$ to its key derivation oracle. The challenger replies with $\mathsf{sk}_{\mathbf{1}}$, which $\mathcal{B}$ sets as the aggregator key, $\mathsf{sk}_0 := \mathsf{sk}_{\mathbf{1}}$ and forwards it to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks encryption oracle queries $\mathsf{QEnc}(i, x_i, t)$ or left-right oracle queries $\mathsf{QLeftRight}(i, x_i^0, x_i^1, t^\star)$, $\mathcal{B}$ forwards the queries to its own corresponding oracles QEnc' or $\mathsf{QLeftRight}'$. As an answer for an encryption oracle query, $\mathcal{B}$ receives $\mathsf{ct}_{i,t}$ and for a left-right oracle query it receives $\mathsf{ct}_{i,t}^\beta$. $\mathcal{B}$ forwards the received ciphertext to $\mathcal{A}$.

It is straightforward to see that the adversary $\mathcal{B}$ perfectly simulates the str-IND security game for PSA to $\mathcal{A}$. Hence, we have:

$$\mathsf{Adv}^{\text{str-AO}}_{\mathsf{PSA},\mathcal{A}}(\lambda, n) \leq \mathsf{Adv}^{\text{any-IND}}_{\mathsf{MCFE},\mathcal{B}}(\lambda, n) \ .$$

□