LaPSuS – A Lattice-Based Private Stream Aggregation Scheme under Scrutiny*

Johannes Ottenhues and Alexander Koch

University of St. Gallen; CNRS, IRIF, Université Paris Cité
johannes.ottenhues@posteo.org, alexander.koch@irif.fr

Abstract. Private Stream Aggregation (PSA) allows clients to send encryptions of their private values to an aggregator that is then able to learn the sum of these values but nothing else. It has since found many applications in practice, e.g. for smart metering or federated learning. In 2018, Becker et al. proposed the first lattice-based PSA scheme LaPS (NDSS 2018), with putative post-quantum security, which has subsequently been patented. In this paper, we describe two attacks on LaPS that break the claimed aggregator obliviousness security notion, where the second attack even allows to recover the secret keys of the clients, given enough encryptions. Moreover, we review the PSA literature for other occurrences of the responsible flawed proof steps. By explicitly tracking down and discussing these flaws, we clarify and hope to contribute to the literature on PSA schemes, in order to prevent further insecure schemes in practice. Finally, we point out that a Real-or-Random variant of the security notion that is often used as a substitute to make proofs easier, is not well-defined and potentially weaker than the standard PSA security notion. We propose a well defined variant and show that it is equivalent to the standard security notion of PSA under mild assumptions.

Keywords: Security Notions, Private Stream Aggregation, Cryptanalysis, Aggregate Statistics

1 Introduction

The more data-driven our economies become, the more important it is for service providers to obtain statistics on the use of their services, to better fit them to the actual needs. However, valid privacy concerns, laws and user preferences constrain a free use of this data. As one solution to this, Shi et al. [14] devised so-called Private Stream Aggregation (PSA) schemes, where a server (the "aggregator") can obtain *aggregate statistics* on the clients data, in the sense that the (a priori fixed set of) clients encrypt their secret values in such a way that the server can compute the sum of their values, but does not learn anything else.

Since then, these have been proposed for numerous uses, including smart metering and federated learning [14, 4, 7, 11]. One of the important follow-up papers to [14], also published at NDSS, is a lattice-based PSA scheme [1], called

* This is an extended version of a paper published at SCN 2024.

"LaPS". The authors argue for its security based on the Augmented Learning with Errors (A-LWE) problem, defined in [6] (which is shown in [6] to be at least as hard as the original Learning with Errors (LWE) problem [13], with a reduction that assumes the random oracle model security heuristic), and hence claim that it is the first post-quantum secure PSA scheme. Subsequently, this scheme became patented [2] and influenced the discussion in the PSA literature, as almost all newer schemes are compared to this w.r.t. security and performance.

As PSA schemes have become attractive enough to be adopted in practice, it is of particular importance to ensure these uses to be actually secure. However, in this paper, we describe how larger parts of the literature on PSA schemes do not fare well when it comes to actual security or the correctness of proofs. More particularly, we describe a flaw in the security proof of LaPS, and corresponding attacks, which amount to the LaPS scheme being fully broken. This includes the possibility that an attacker can learn the user secret keys, given enough encryption queries (relevant due to the claimed encrypt-multiple-times security). The attack/flaw arises from a proof strategy (replacing certain oracle queries by others, see below) in [14], which is valid within their reduction, but due to a different reason than stated there, and hence is not as universally applicable as it appears. However, this strategy was used in [1] in a different context, where it is no longer permissible.

Additionally, we point out another flaw that is present in the security proof of [1]. Importantly, almost five years after its publication at the NDSS conference, this has remained unknown, and the flaws described within our paper have since spread to at least four other papers in the area, including one published in Journal of Cryptology [16]. See Table 1 for an overview of the issues as reported in our work.

In addition to the aforementioned proof issues, we point out another problem with a frequently used security definition. Many security proofs of PSA schemes substitute in their first step the aimed-for security notion with a different version, called Real-or-Random security (e.g. [14, 1, 10, 15, 16, 9]). However, throughout the literature, this notion is only vaguely defined in passing, and hence lacks a proper foundation: We argue in Section 5 that the Real-or-Random notion of PSA as it is usually described, is in fact not well-defined and potentially weaker than the standard security notion. We propose a proper definition that, however, makes use of the additional assumption of having a group structure on the message space, which might not be applicable in all use-cases. Then, we show equivalence to the usual aggregator obliviousness security. Note, however, that our definition does not automatically fix the affected proofs. By clearing up this situation, we contribute to clarify the proper ways of proving security for PSA schemes.

Related Work. The only other paper where weaknesses of PSA schemes are discussed that we are aware of is [17]. In Remark 5.3 of [17], the authors point out that the security notion claimed to be achieved by the LaPS scheme [1], can in fact not be achieved. This is due to the (incorrect) formulation of a

Table 1. Overview of proof flaws and attacks as presented in this paper.

Proof flaw 1	[1, 9, 10]
Proof flaw 2	[1, 9, 10, 15, 16]
Attack 1 & 2	[1]

certain non-triviality condition in [1].[1] In contrast, we give two different concrete attacks on the LaPS scheme, where the first allows the aggregator to compute the difference of two client plaintexts and the second to compute the secret key, given enough ciphertexts. These attacks also break the security notion even if the non-triviality condition is formalized correctly. Also, we are the first to describe the concrete flaws in the security proof of LaPS.

Furthermore, Waldner et al. [17] point out (in footnote 9) that the security guarantee provided by a restricted one-time-use variant of the standard PSA security notion can be achieved by a simple information-theoretic construction. In Appendix B, we formalize this construction and prove that it does indeed satisfy the corresponding one-time-use security notion. Additionally, Waldner et al. [17] devise a stronger notion of PSA security than commonly used, argue for its usefulness and provide a construction that satisfies this stronger notion.

2 Preliminaries

2.1 Notations

With $[n]$ and $[n]_0$ we denote the sets $\{1, \ldots, n\}$ and $\{0, \ldots, n\}$, respectively. We use λ as the security parameter. We write $x \leftarrow X$ to denote that x is sampled from distribution X or is the output of a randomized algorithm X. We write $x \leftarrow\!\!\$\ S$ when x is sampled uniformly at random from set S. We usually denote the adversary with $\mathcal{A}$.

2.2 Private Stream Aggregation

In private stream aggregation (PSA) there is one aggregator and many clients. At each time step, the clients encrypt their current value (e.g. power consumption or ML model update) and send the resulting ciphertext to the aggregator. When the aggregator has received one ciphertext from each client for the same time step, the aggregator can compute the sum of the client's values, but nothing more. The protocol is non-interactive insofar that the aggregator does not send any messages to the clients and the clients only send messages to the aggregator (and not to each other). In some definitions the clients use random noise in order to achieve differential privacy (cf. [5] for an introduction). However, we omit the

[1] In a setting where multiple encryptions per client and label are allowed, the non-triviality condition must also take into account the plaintexts of the encryption queries, as done e.g. in [17, Def. 5.1].

noise in our definition, as it can simply be added to the plaintext value before encryption. The following definitions of PSA and aggregator obliviousness were introduced by [14] and are taken verbatim from [7].

Definition 1 (Private Stream Aggregation (Def. 2 in [7])). *A* private stream aggregation *scheme* PSA *over* $\mathbb{Z}_R$ *(for* $R \in \mathbb{N}$*) and label space* $\mathcal{L}$ *consists of the following three PPT algorithms for the setup, the encryption and the decryption of the aggregate sum:*

- $\mathsf{Setup}(1^\lambda, 1^n)$*: Given the security parameter* λ *and the number of users* n*, it outputs public parameters* pp *and* $n+1$ *keys* $(\mathsf{sk}_i)_{i\in[n]_0}$*. The key* sk_0 *is the (secret) key of the aggregator, and each* sk_i *is a (secret) key of a user* $i \in [n]$*.*
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, l, x_i)$*: Given the public parameters* pp*, a key* sk_i *of user* $i \in [n]$*, a label* $l \in \mathcal{L}$ *and a value* $x_i \in \mathbb{Z}_R$*, it outputs an encryption* c_i *of* x_i *under key* sk_i *with label* l*. This algorithm is supposed to be executed by each user at every time step, where the time step is used as label. The user then sends* c_i *to the aggregator.*
- $\mathsf{AggrDec}(\mathsf{pp}, \mathsf{sk}_0, l, \{c_i\}_{i\in[n]})$*: Given the public parameters* pp*, the aggregator's key* sk_0*, a label* $l \in \mathcal{L}$*, and a set of* n *ciphertexts* $\{c_i\}_{i\in[n]}$ *that were encrypted under the same label* l*, it outputs* $\sum_{i\in[n]} x_i \pmod R$*.*

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

$$\mathsf{AggrDec}(\mathsf{pp}, \mathsf{sk}_0, l, \{c_i\}_{i\in[n]}) = \sum_{i\in[n]} x_i \bmod R.$$

The general use of PSA is as follows: A trusted third party executes the Setup algorithm and distributes the public parameters and the secret keys to the clients and the aggregator. Ernst and Koch [7] describe an approach to get a decentralized setup. At each time step t each client i calls the $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, t, x_i)$ algorithm and sends the resulting ciphertext to the aggregator. When the aggregator has received a ciphertext from each client, they call the $\mathsf{AggrDec}$ algorithm to compute the sum of the clients' values.

Next, we define Aggregator Obliviousness (AO), which is the security notion of PSA. We only define *encrypt-once* security, which restricts the clients to send only one message per label. This reduces the potential leakage in practice, because sending more than one ciphertext per label usually allows the aggregator to compute the differences of the plaintexts (cf. Section 2.3 for a discussion on this). Focusing on encrypt-once security is common in the literature [14, 3, 16].

Definition 2 (Aggregator obliviousness (Def. 3 in [7])). *The game-based security notion of aggregator obliviousness (AO) is defined via the following security experiment* $\mathsf{AO}_b(\lambda, n, \mathcal{A})$*, given in Figure 1. Here,* $b \in \{0, 1\}$ *indicates which of the messages are encrypted and returned by* $\mathsf{QChallenge}$*. First, the challenger runs* Setup *and passes the public parameters* pp *to the adversary* $\mathcal{A}$*. Then,* $\mathcal{A}$ *can adaptively ask queries to the following oracles:*

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

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

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

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

$\mathsf{AO}_b(\lambda, n, \mathcal{A})$

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

$\alpha \leftarrow \mathcal{A}^{\mathsf{QCorrupt}(\cdot),\mathsf{QEnc}(\cdot,\cdot,\cdot),\mathsf{QChallenge}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp})$

`if` condition $(*)$ is satisfied (see p. 5)

    `output` α

`else`

    `output` 0

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

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

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

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

1. $\mathcal{U} \cap \mathcal{CS} = \emptyset$*. This means that all users for which* $\mathcal{A}$ *receives a challenge ciphertext must stay uncorrupted during the entire game.*
2. $\mathcal{A}$ *has not queried* $\mathsf{QEnc}(i, x_i, l)$ *twice for the same* (i, l)*. Doing so would violate the encrypt-once restriction.*
3. $\mathcal{U} \cap \mathcal{E}_{l^*} = \emptyset$*. This means that* $\mathcal{A}$ *is not allowed to get a challenge ciphertext from users for which they ask an encryption query on the challenge label* l^**. Doing this would violate the encrypt-once restriction.*

4. If $\mathcal{A}$ has corrupted the aggregator and $\mathcal{Q}_{l^*} \cup \mathcal{CS} = [n]$ *then we require that*

$$\sum_{i \in \mathcal{U}} x_i^0 = \sum_{i \in \mathcal{U}} x_i^1.$$

We will call this condition the balance-condition*.*

We define $\mathcal{A}$'s advantage as

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

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

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

The balance condition applies in the case where the aggregator (or an attacker who has corrupted the aggregator) can use their aggregation capability to legitimately compute the sum of the clients' values. Thus, the balance condition prevents the adversary from trivially winning the game. However, note that the balance-condition does only apply if $\mathcal{A}$ got a ciphertext from *every* honest user for the challenge label l^*. If there is a single ciphertext missing, then the balance condition does not apply, because in the real world the aggregator would also not be able to compute the sum of the plaintext values. The preconditions for the balance-condition are sometimes stated incorrectly. Some papers (e.g. [1, 15, 16]) define that the balance-condition already applies if the aggregator is corrupted, regardless of whether there are enough ciphertexts. This places an unnecessary restriction on the adversary.

Next we define a modified version of the AO game, which we call *corruption-only* aggregator obliviousness ($\mathsf{AO}^{\mathsf{co}}$). In this game the challenger treats all encryption queries as corruption queries. We define this, because several papers in the literature [1, 9, 10] have made this modification in the security proof. However, as we point out in Section 3.2, this definition gives much weaker security guarantees and should not be used.

Definition 3 (Corruption-only aggregator obliviousness). *We define the corruption-only aggregator obliviousness ($\mathsf{AO}^{\mathsf{co}}(\lambda, n, \mathcal{A})$) game identical to the AO game with the only difference that all encryption queries are treated as corruption queries. That is, we change the definition of the encryption oracle as follows:*

$\underline{\mathsf{QEnc}}(i, x_i, l)$*: Given a user index $i \in [n]$, a value $x_i \in \mathbb{Z}_R$, and a label l, it answers with $\mathsf{QCorrupt}(i) = \mathsf{sk}_i$ (and updates the list $\mathcal{CS}$ of corrupted users accordingly).*

Note that treating encryption queries as corruption queries is equivalent to ignoring the encryption queries, as the adversary would then simply ask corruption queries instead of encryption queries.

2.3 On Labels, Encrypt-Once and Inherent Leakage

When the clients are not bound by the encrypt-once restriction, i.e. clients are allowed to send multiple ciphertexts per label, then the aggregator can already learn the difference of the same user's plaintexts. To see this, suppose the aggregator has one ciphertext from each client $\{c_i \coloneqq \mathsf{Enc}(\mathsf{sk}_i, x_i, l)\}_{i\in[n]}$ which are encryptions of plaintexts x_i under label l. Then suppose the aggregator gets another ciphertext $c_1' = \mathsf{Enc}(sk_1, x_1', l)$ which is an encryption of x_1' under the same label l. Now the aggregator can compute $\mathsf{AggrDec}(\mathsf{sk}_0, (c_1, c_2, \ldots, c_n), l) - \mathsf{AggrDec}(\mathsf{sk}_0, (c_1', c_2, \ldots, c_n), l) = x_1 + x_2 + \cdots + x_n - (x_1' + x_2 + \cdots + x_n) = x_1 - x_1'$. Because this leakage is inherent to the aggregation capability of the aggregator it is often called *inherent* leakage. Note, however, that this is only possible if the aggregator gets at least one ciphertext from every client, because otherwise the aggregator cannot use the $\mathsf{AggrDec}$ function. This example shows that the aggregator can already learn information about individual clients' plaintexts if they obtain two ciphertexts of the same client and at least one ciphertext of every other client that were created under the same label. Therefore, it is advisable to adhere to the encrypt-once restriction and use a new label for each new message.

3 Attack on the LaPS Scheme

We begin by pointing out a flaw in the security proof that reduces the security guarantee to a setting where only one ciphertext can be encrypted. (Note that this does not yet show that the scheme is actually insecure, for this we also need the next step.) We proceed by presenting two attacks on the scheme, which violate the claimed security property [1, Def. 6] of [1]. In particular, the authors stress that they do not need to rely on the encrypt-once restriction (cf. [1, p. 3]) and the aggregator obliviousness definition they claim for their scheme allows for as many encryption queries as the adversary chooses. Our first attack allows the aggregator to decrypt messages under certain circumstances. The second attack enables the aggregator to compute the secret keys of the clients, given sufficient ciphertexts.

The first attack only needs two ciphertexts of the same client, the second attack needs a lot more ciphertexts. Nevertheless, e.g. in the setting of privacy preserving smart-metering new ciphertexts are sent regularly over a long period of time. Thus, also the second attack is of practical concern.

3.1 The LaPS Scheme

Here we give a rough overview of the LaPS PSA scheme, cf. [1] for more details. Let $D_{\mathbb{Z}_q^a,\sigma}$ be the Gaussian distribution over $\mathbb{Z}_q^a$ with standard deviation σ. Let $\mathbf{G}$ be a gadget matrix and $D_{\Lambda^\perp_{\mathbf{v}_i}(\mathbf{G}),\sigma}$ the Gaussian distribution over the lattice $\Lambda^\perp_{\mathbf{v}_i}(\mathbf{G})$. For $\mathbf{e}_i \leftarrow D_{\Lambda^\perp_{\mathbf{v}_i}(\mathbf{G}),\sigma}$ an important property is that $\mathbf{G}\cdot\mathbf{e}_i = \mathbf{v}_i$. The vector $\mathbf{v}_i$ is the encryption of the message x_i under an asymmetric additively homomorphic encryption scheme $(\mathsf{Gen}_{\mathsf{ahom}}, \mathsf{Enc}_{\mathsf{ahom}}, \mathsf{Dec}_{\mathsf{ahom}})$ that has

pseudorandom ciphertexts and where the addition of plaintexts corresponds to the addition of ciphertexts. In [1] it is instantiated with the BGV encryption scheme which satisfies these properties. Note that LaPS does not have labels.

The scheme with security parameter λ is shown in Figure 2. The public parameters are a uniformly random matrix $\mathbf{A}$ and the public key of the homomorphic encryption scheme. The secret keys of the clients are simply random vectors in $\mathbb{Z}_q^\lambda$. The aggregator's key is the sum of these vectors and the secret key of the homomorphic encryption scheme. To encrypt a value, a client first encrypts this value with the homomorphic encryption scheme and then samples an error term from a Gaussian distribution that is defined via the ciphertext of the homomorphic encryption scheme. The PSA ciphertext then is the LWE sample created with the client's secret vector, the matrix $\mathbf{A}$ and the error term. To decrypt, the aggregator adds all ciphertexts and subtracts their secret key multiplied with $\mathbf{A}$. They then multiply the resulting vector with the gadget matrix $\mathbf{G}$ and decrypt the result with the secret key of the homomorphic encryption scheme.

$\underline{\mathsf{Setup}(1^\lambda, 1^n):}$

sample $\mathbf{A}$ as uniformly random matrix

`for` $i \in [n]$: $\mathbf{s}_i \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda$

$(\mathsf{pk}_{\mathsf{ahom}}, \mathsf{sk}_{\mathsf{ahom}}) \leftarrow \mathsf{Gen}_{\mathsf{ahom}}(1^\lambda)$

$\mathsf{pp} := (\mathbf{A}, \mathsf{pk}_{\mathsf{ahom}})$

$\mathsf{sk}_0 := (\mathsf{sk}_{\mathsf{ahom}}, \sum_{i\in[n]} \mathbf{s}_i)$

`return` $(\mathsf{pp}, \mathsf{sk}_0, \{\mathbf{s}_i\}_{i\in[n]})$

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathbf{s}_i, x_i):}$

$\mathbf{v}_i \leftarrow \mathsf{Enc}_{\mathsf{ahom}}(\mathsf{pk}_{\mathsf{ahom}}, x_i)$

$\mathbf{e}_i \leftarrow D_{\Lambda^\perp_{\mathbf{v}_i}(\mathbf{G}),\sigma}$

$\mathbf{c}_i = \mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top \bmod q$

`return` $\mathbf{c}_i$

$\underline{\mathsf{AggrDec}(\mathsf{pp}, \mathsf{sk}_0, \{\mathbf{c}_i\}_{i\in[n]}):}$

`parse` $\mathsf{sk}_0 = (\mathsf{sk}_{\mathsf{ahom}}, \mathbf{s} = \sum_{i\in[n]} \mathbf{s}_i)$

$\mathbf{e} = \sum_{i\in[n]} \mathbf{c}_i - \mathbf{s}^\top \mathbf{A}$

`return` $\mathsf{Dec}_{\mathsf{ahom}}(\mathsf{sk}_{\mathsf{ahom}}, \mathbf{G} \cdot \mathbf{e} \bmod q)$

Fig. 2. The LaPS scheme.

3.2 Flaw: Treating Encryption Queries as Corruption Queries

Here we describe a flaw in the security proof of [1, pp. 15 sq.], which comes about, because encryption queries are treated as corruption queries in the proof. This makes the adversary much weaker and, therefore, invalidates the proof.

In the proof of AO security of the LaPS scheme, queries of the form $\mathsf{Enc}(i, x_i)$ are treated as corruption queries $\mathsf{Corrupt}(i)$. We will refer to this modified game as the corruption-only aggregator obliviousness ($\mathsf{AO}^{\mathsf{co}}$, cf. Definition 3) game, and to the regular aggregator obliviousness game as AO game[2]. In [1] the authors claim that using the $\mathsf{AO}^{\mathsf{co}}$ game instead of AO game is a valid adaption, because it gives more power to the adversary. However this is *not* true. It reduces the choice of the adversary and makes them therefore weaker.

They then show security by a reduction of the augmented learning with errors (A-LWE) problem from [6] to the $\mathsf{AO}^{\mathsf{co}}$ game. This means that an adversary on the $\mathsf{AO}^{\mathsf{co}}$ game can be turned into an adversary on the A-LWE problem. To show AO security, it would have been necessary to additionally show that an adversary on the AO game can be turned into an adversary on the $\mathsf{AO}^{\mathsf{co}}$ game. By transitivity this would have meant that an adversary on the AO game could be turned into an adversary on the A-LWE problem. However, in [1] the authors claim that an adversary on the $\mathsf{AO}^{\mathsf{co}}$ game can be turned into an adversary on the AO game. This is correct, but is exactly the opposite direction of what would have been necessary to show. Figure 3 shows how the relation between the attackers is in [1] and how the relation would have had to be for the proof to go through.

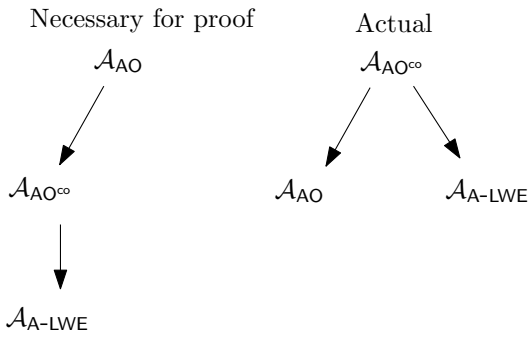

Fig. 3. Graphs showing which kind of adversary can be turned into which kind of adversary. On the left is the situations that would have been necessary for the proof to be valid. On the right is the actual situation from the proof.

[2] In Appendix B, we show in Proposition 1 that a "trivial" one-time-pad PSA scheme already satisfies the $\mathsf{AO}^{\mathsf{co}}$-security notion, and in Proposition 2 we show that this OTP construction is not AO-secure.

So far, we have argued that the proof is lacking a reduction from the $\mathsf{AO}^{\mathsf{co}}$ game to the AO game. Next we argue that the reduction is not only missing, but that there exists no such reduction.

For an intuition of this fact, let us argue as follows: Restricting the adversary to only ask corruption queries, instead of also allowing encryption queries, is indeed a restriction, because corruption queries lead to stricter conditions on what the adversary is allowed to do. If the adversary corrupts a user then they are not allowed to include this user in the set $\mathcal{U}$ of the challenge query (due to the first item of $(*)$). Whereas if the adversary only asks an encryption query, they are allowed to include this user in the challenge query[3]. Therefore, restricting an adversary to only ask corruption queries severely reduces their options.

In more detail, the adversary $\mathcal{A}$ can only use honest users in the challenge query and the only way that $\mathcal{A}$ can get ciphertexts from clients is through the challenge query and through encryption queries. When encryption queries are treated as corruption queries, the only way to get ciphertexts from honest users is through the challenge query. However, since the challenge query can only be asked once, $\mathcal{A}$ can get at most one ciphertext per honest user. Only guaranteeing security as long as $\mathcal{A}$ does not see more than one ciphertext per client is a much weaker security guarantee than allowing arbitrarily many ciphertexts. Hence, by treating encryption queries as corruption queries the security model is essentially diminished to a model in which a one-time-pad construction is secure, shown more verbosely in Appendix B.

The above flaw appears in the security proofs of two other papers on PSA [9, 10]. In both cases encryption queries are treated as corruption queries, which highlights the importance of clarifying the situation and showing that this simplification is not valid.

The origin of the flaw. We note that Shi et al. [14] also treat encryption queries as corruption queries in one step in their proof of AO security. However, they do so *only* for the challenge label, i.e. the label that is specified in the challenge query. They justify this with the same wrong argument that this step makes the adversary stronger. However, in their situation this query replacement is actually legitimate.

This is because their scheme uses labels and requires the encrypt-once restriction, which only allows one encryption or challenge query per user and label. They treat encryption queries as corruption queries only for the label for which the challenge query is asked. Because of the encrypt-once restriction, the adversary is not allowed to ask encryption queries on the challenge-label for users that are included in the challenge query. Therefore, asking a corruption or encryption query for the challenge-label results in the same conditions for the adversary. Hence, this step is valid, albeit for a different reason than given in [14].

[3] Although with encrypt-once the challenge query must be for a different label as the encryption query

3.3 First Attack: Decrypting Messages

Here, we describe the attack that allows the aggregator to decrypt individual messages. It also enables an adversary to win the AO game. Waldner et al. [17] already pointed out a weakness, not of the scheme itself (as it only makes black-box use of the 3-tuple of PSA algorithms), but of the exact security notion as formalized in [1] that allows an attacker to win the AO security game. Note, however, that this is due to the incorrect formulation of the balance condition in [1]. In contrast our attack also works with respect to the correct formulation of the AO security game.

With our attack, the aggregator (or, more formally, the adversary corrupting the aggregator) can compute the difference of two plaintexts of any individual client. For the attack they only need two ciphertexts of that client. This also means that when the attacker knows one of the plaintexts they can immediately compute the other plaintext. Note that getting the aggregation key does not directly allow an attacker to decrypt individual ciphertexts but only to compute the sum of the client's values for epochs in which all clients submit a ciphertext. Our attack already works if only one client submits two ciphertexts, even if all other clients do not submit a ciphertext. Thereby, our attack in fact violates the guarantees provided by the security notion (and is not covered by the inherent leakage as given in Section 2.3).

Let c_i and c'_i be the encryptions of x_i and x'_i of client i as described in Figure 2. The aggregator computes

$$\mathbf{c}_i - \mathbf{c}'_i = \mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top - \mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i'^\top \bmod q = \mathbf{e}_i^\top - \mathbf{e}_i'^\top \bmod q.$$

Then they use the result to compute

$$\mathbf{G}(\mathbf{e}_i^\top - \mathbf{e}_i'^\top) \bmod q = \mathbf{G}\mathbf{e}_i^\top - \mathbf{G}\mathbf{e}_i'^\top \bmod q = \mathbf{v}_i - \mathbf{v}'_i \bmod q.$$

Because the aggregator knows $\mathsf{sk}_{\mathsf{ahom}}$, they can decrypt this value to get

$$\begin{aligned}\mathsf{Dec}_{\mathsf{ahom}}(\mathsf{sk}_{\mathsf{ahom}}, \mathbf{v}_i - \mathbf{v}'_i \bmod q) &= \mathsf{Dec}_{\mathsf{ahom}}(\mathsf{sk}_{\mathsf{ahom}}, \mathbf{v}_i) - \mathsf{Dec}_{\mathsf{ahom}}(\mathsf{sk}_{\mathsf{ahom}}, \mathbf{v}'_i) \\ &= x_i - x'_i \bmod q.\end{aligned}$$

To win the AO game using this attack, the adversary proceeds as follows. First they corrupt the aggregator. Then they ask an encryption query for an arbitrary message x_i and arbitrary user i, and receive the ciphertext c_i as answer. Then they ask a challenge query for user i and an arbitrary user j, that is with $\mathcal{U} = \{i, j\}$ and $x_i^0 \neq x_i^1$ and $x_i^0 + x_j^0 = x_i^1 + x_j^1$ and get the ciphertexts c_i, c_j as answer. The challenge messages for user j are only there to satisfy the balance condition. The adversary now uses the attack as described above to decrypt the challenge message c_i.

Discussion. Although this attack seems quite generic, its usefulness is often limited. The common scenario for PSA is that the aggregator gets ciphertexts from all clients and is then able to compute the sum of the clients' values.

Suppose the aggregator has ciphertexts c_i from all clients and additionally a second ciphertext c'_{i^*} from user i^* for a known plaintext x'_{i^*}. Then, the aggregator can decrypt the sum of the ciphertext, once with i^*'s first ciphertext c_{i^*} and once with i^*'s second ciphertext c'_{i^*}. The difference of the sums is $x_{i^*} - x'_{i^*}$, from which the aggregator gets x_{i^*} by adding x'_{i^*}. This works for all PSA schemes by the property of the sum functionality. We have discussed this issue in Section 2.3. Thus, the only situation in which the aggregator has an advantage through the described attack is when at least one client does not send a ciphertext. In that case the aggregator can compute the difference of the client's plaintexts with our attack, but not by using their aggregation capability.

3.4 Second Attack: Getting the Secret Key

In this section we show that, given enough ciphertexts, the aggregator can compute the individual clients' secret keys. This is possible, because the scheme uses the product of the LWE secret and matrix $\mathbf{s}^\top \mathbf{A}$ multiple times with different error terms. However, the LWE assumption only promises pseudorandomness, if a fresh $\mathbf{a}$ is used for each sample. Therefore, it is not surprising that this allows for an attack on the scheme. This construction mistake was not caught because, as illustrated above, the reduction in the proof is missing a tackling of the encryption queries (as it is only showing $\mathsf{AO}^{\mathsf{co}}$ security).

When there are enough samples $\mathbf{s}^\top \mathbf{A} + \mathbf{e}_l$ and one takes their average, the errors cancel out with high probability and the aggregator is able to recover $\mathbf{s}^\top \mathbf{A}$. Intuitively, when more and more samples are added to the average, the error distribution gets narrower and narrower until the error is likely to be zero. From $\mathbf{s}^\top \mathbf{A}$ the aggregator can than compute $\mathbf{s}$ by Gaussian elimination.

The attack. Remember that a ciphertext for user i is computed as $c_i = \mathbf{s}_i^\top \mathbf{A} + \mathbf{e}_i^\top$ (the message is encoded in the error term). The attack works as follows: The adversary $\mathcal{A}$ collects all ciphertexts for a certain user i^*. Let $\{c_{i^*}^1, \ldots, c_{i^*}^t\}$ be the ciphertexts that $\mathcal{A}$ collected. The adversary computes

$$\frac{1}{t} \cdot \sum_{k \in [t]} c_{i^*}^k = \frac{1}{t}(t \cdot \mathbf{s}_{i^*}^\top \mathbf{A} + \sum_{k \in [t]} \mathbf{e}_{i^*}^k) = \mathbf{s}_{i^*}^\top \mathbf{A} + \frac{1}{t} \sum_{k \in [t]} \mathbf{e}_{i^*}^k.$$

The adversary then hopes that $\frac{1}{t} \sum_{k \in [t]} \mathbf{e}_{i^*}^k = 0$ and uses Gaussian elimination to recover $\mathbf{s}_{i^*}$. Next we provide a rough sketch of an analysis of the success probability of the attack.

The analysis. Note that this is only a sketch and that we make several simplifying assumptions. In [1], the error vectors $\mathbf{e}_{i^*}$ are drawn from a multidimensional Gaussian distribution with variance σ^2. Let X be a random variable. We write $X \sim \mathcal{N}(\mu, \sigma^2)$ to express that X follows a Gaussian distribution with mean μ and variance σ^2. By $\mathrm{Var}[X]$ we denote the variance of X. Let $X_1, X_2 \sim \mathcal{N}(\mu, \sigma^2)$ be independent and identically distributed, then $X_1 + X_2 \sim \mathcal{N}(2 \cdot \mu, 2 \cdot \sigma^2)$ and

$a \cdot X_1 \sim \mathcal{N}(a \cdot \mu, a^2 \cdot \sigma^2)$. This implies that averaging the sum of t independent Gaussian distributions $X_j \sim \mathcal{N}(\mu, \sigma^2)$ is $\frac{1}{t} \sum_{j \in [t]} X_j \sim \mathcal{N}(\mu, \frac{1}{t} \cdot \sigma^2)$. Chebyshev's inequality states for random variables X with finite mean μ and finite non-zero variance $\mathrm{Var}[X]$ that for every $\epsilon > 0$,

$$\Pr[|\mathrm{X} - \mu| \geq \epsilon] \leq \frac{\mathrm{Var}[X]}{\epsilon^2}.$$

We make a few simplifying assumptions. First, we use the continuous Gaussian distribution in the analysis, although the LaPS scheme uses a discretized Gaussian distribution. Second, for $X \sim \mathcal{N}(0, \sigma^2)$ we assume that the probability of the discretized version of X being 0 is roughly $\Pr[|X| < 1/2]$, which is essentially rounding X to integer values.

For the attack to succeed we need that the error terms vanish, i.e. $\frac{1}{t} \sum_{k \in [t]} \mathbf{e}_{i*}^k = 0$. Since this is a vector with λ entries, we first analyze for each entry the probability to be zero. The entries are drawn from $X_j \sim \mathcal{N}(0, \sigma^2)$, so for the average of t samples we have $\frac{1}{t} \cdot \sum_{j \in [t]} X_j \sim \mathcal{N}(0, \frac{1}{t} \cdot \sigma^2)$. Let $Y := \frac{1}{t} \cdot \sum_{j \in [t]} X_j$, then $Y \sim \mathcal{N}(\mu, \frac{1}{t} \cdot \sigma^2)$ as explained above. We need to analyze the probability that the discretized version of Y is equal to 0, which we assume to be $\Pr[|Y| < 1/2]$. By Chebyshev's inequality we get

$$\Pr[|Y| < 1/2] = 1 - \Pr[|Y| \geq 1/2] \geq 1 - \frac{\mathrm{Var}[Y]}{(1/2)^2} = 1 - 4 \cdot \mathrm{Var}[Y] = 1 - \frac{4 \cdot \sigma^2}{t}.$$

This is the probability that a specific entry of Y is 0. The probability that all λ entries of Y are zero is $(1 - \frac{4 \cdot \sigma^2}{t})^\lambda$. When the adversary has $t = \sigma^2 \cdot \lambda$ ciphertexts, then the success probability is $(1 - \frac{4}{\lambda})^\lambda$, which for large λ approaches e^{-4} from below. For $\lambda = 128$ the success probability is ≈ 0.017.

For an LWE modulus q the standard deviation σ is strictly smaller than q, so the variance σ^2 is strictly smaller than q^2. When the LWE modulus q is polynomial in the security parameter λ, then $\sigma^2 \cdot \lambda$ is polynomial in λ. This implies that polynomially many ciphertext are sufficient for the attack to succeed with probability roughly e^{-4}.

4 Another Flaw in AO Security Proofs: Answer to the Challenge Query

In this section we describe another flaw in AO proofs that appears in [16, 15, 1, 10, 9]. Our goal is to clarify the problem and, thereby, help to avoid it in the future. In all of these papers the authors use a real-or-random version of the AO game that we denote by $\mathsf{AO}^{\mathsf{RoR}}$. In the $\mathsf{AO}^{\mathsf{RoR}}$ game the adversary submits only one set of plaintexts in the challenge query and gets back encryptions either of these plaintexts or of random plaintexts. The notion $\mathsf{AO}^{\mathsf{RoR}}$ has some issues, which we discuss in Section 5, but the flaw that we discuss in this section is independent of these issues. The general proof structure is that the authors want to use a hypothetical attacker $\mathcal{A}$ on the $\mathsf{AO}^{\mathsf{RoR}}$ security of their PSA scheme

to construct an attacker $\mathcal{B}$ that plays with a challenger $\mathcal{C}$ and can solve a hard problem P (such as LWE). The attacker $\mathcal{B}$ answers $\mathcal{A}$'s $\mathsf{AO}^{\mathsf{RoR}}$-game queries (sometimes by asking queries to $\mathcal{C}$) and at some point embeds $\mathcal{C}$'s challenge in the answers. At some other point, $\mathcal{A}$ will submit a guess b' of whether they are in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ or the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game. $\mathcal{B}$ simply forwards this guess b' to their own challenger $\mathcal{C}$. The argument then is that $\mathcal{B}$ has the same advantage in solving P as $\mathcal{A}$ has in the $\mathsf{AO}^{\mathsf{RoR}}$ game. However, this argument does only hold, if $\mathcal{B}$ perfectly simulates (depending on $\mathcal{C}$'s challenge bit) either $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ or $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$. In all the above mentioned papers this is *not* the case. This makes it possible that $\mathcal{A}$ realizes that they are being used in a security reduction and then behave in a way so that $\mathcal{B}$ cannot use them to break the problem P. This would not impact $\mathcal{A}$'s advantage on the $\mathsf{AO}^{\mathsf{RoR}}$ game, but render $\mathcal{A}$ useless for breaking P, which invalidates the proof.

Let us zoom in on the technical details. We will describe the exact problem at the example [16], which uses the Ring-LWE assumption [12], but the other papers ([1, 15, 10, 9]) have the same issue, although sometimes with different hardness assumptions. We begin by summarizing the proof and then describe the flaw. The Ring-LWE challenger $\mathcal{C}$ will select a random bit $b_{\mathcal{C}}$ and provide $\mathcal{B}$ with Ring-LWE samples (if $b_{\mathcal{C}} = 0$) or with uniform samples (if $b_{\mathcal{C}} = 1$). $\mathcal{B}$ tries to guess $b_{\mathcal{C}}$ by using an $\mathsf{AO}^{\mathsf{RoR}}$ attacker $\mathcal{A}$. $\mathcal{B}$ starts by randomly selecting two users j, k and chooses PSA secret keys for all users except of j, k, i.e. for users $i \in [n] \setminus \{j, k\}$. Let us look at the way $\mathcal{B}$ answers the (real-or-random) challenge query from $\mathcal{A}$. Upon receiving a query $\mathsf{QChallenge}(\mathcal{U}, \{x_i\}_{i\in\mathcal{U}}, l^*)$[4], $\mathcal{B}$ encrypts all $\{x_i\}_{i\in\mathcal{U}\setminus\{j,k\}}$ with the previously chosen secret keys as $\{c_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, x_i, l^*)\}_{i\in\mathcal{U}\setminus\{j,k\}}$. For user j, $\mathcal{B}$ encrypts x_j by adding a sample v received from $\mathcal{C}$ as $c_j := v + x_j$. For user k, $\mathcal{B}$ sets c_k to the value that ensures that the aggregator can correctly decrypt the sum of all plaintexts.

When $b_{\mathcal{C}} = 0$ then the sample v that $\mathcal{B}$ uses to encrypt x_j is a Ring-LWE sample, whereby c_j is indeed a valid encryption of x_j. The ciphertexts of the other clients are also valid encryptions of x_i and, thus, $\mathcal{B}$ perfectly simulates $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$. When $b_{\mathcal{C}} = 1$ then v is a uniform sample and $c_j := v + x_i$ is uniformly random as well. This can also be seen as an encryption of a random value and, thus, c_j (and also c_k) is as in $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$. However, the ciphertexts $\{c_i\}_{i\in\mathcal{U}\setminus\{j,k\}}$ are created as $\{c_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, x_i, l^*)\}_{i\in\mathcal{U}\setminus\{j,k\}}$ regardless of $b_{\mathcal{C}}$. Thus, these ciphertexts are always as in $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$. Therefore, in the case that $b_{\mathcal{C}} = 1$, two ciphertexts are as in $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ and the others are as in $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$. Hence, $\mathcal{B}$ does not properly simulate $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$. As described above, this can allow $\mathcal{A}$ to realize that they are being used in a security reduction and then behave as described next.

Let us illustrate the above problem by describing an attacker $\mathcal{A}$ with advantage (close to) 1 in the $\mathsf{AO}^{\mathsf{RoR}}$ game, where the above described Ring-LWE attacker $\mathcal{B}$ has advantage 0 in distinguishing Ring-LWE samples from uniform. Let $\mathcal{A}$ be an attacker that always wins the $\mathsf{AO}^{\mathsf{RoR}}$ game, e.g. because they are able to reliably decrypt the PSA ciphertexts of individual clients. $\mathcal{A}$ would choose plaintexts $\{x_i\}_{i\in[n]}$ and a label l^* and send a challenge query $\mathsf{QChallenge}(\mathcal{U} = [n], \{x_i\}_{i\in\mathcal{U}}, l^*)$

[4] We omit the noise r_i as it plays no role in the security analysis.

to $\mathcal{B}$. After receiving back the challenge ciphertexts $\{c_i\}_{i\in[n]}$, $\mathcal{A}$ would decrypt all of them. If the decrypted plaintexts match the $\{x_i\}_{i\in[n]}$, then $\mathcal{A}$ outputs $b' = 0$ indicating that they think they are in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ game. If all but two of the decrypted plaintexts match the $\{x_i\}_{i\in[n]}$ then $\mathcal{A}$ knows that they are neither in $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ nor in $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ and outputs $b' = 0$. Thus, B will always guess that $b_{\mathcal{C}} = 0$, regardless of the actual value of $b_{\mathcal{C}}$, which gives $\mathcal{B}$ an advantage of 0. If all the challenge ciphertexts turn out to be encryptions of random plaintexts, $\mathcal{A}$ guesses $b' = 1$ and, therefore, has advantage close to 1.

In [16, 15], an additional and very similar flaw is present. In these papers additionally the encryption queries are answered incorrectly. The encryption queries for all clients except for j, k are answered with proper encryptions of the plaintext, which is correct. However, the encryption queries for clients j, k are answered by using the challenge from the Ring-LWE challenger $\mathcal{C}$. When $\mathcal{C}$ did send real Ring-LWE samples, then the ciphertexts of j, k are proper encryptions as well. The problem arises when $\mathcal{C}$ sends random samples, because then the ciphertexts of clients j, k are essentially random values. However, both in $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ and in $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ the encryption queries should be answered with proper encryptions and not with random values. As in the case of the incorrectly answered challenge queries above, this can allow the adversary to avoid being used in a security reduction.

However, note that despite of the flaws in the security proofs, we did not find attacks on [16, 15, 10, 9] (as opposed to the attack on [1]).

5 On Real-or-Random Security Notions for Aggregator Obliviousness

In [14], the security notion of PSA (aggregator obliviousness) has been defined as a game where the adversary can submit two sets of challenge plaintexts and gets the encryptions of the plaintexts from one of the sets. The adversary then has to guess to which set the ciphertexts belong. However, in many papers, e.g. [14, 1, 10, 15, 16, 9], a real-or-random version of the game is used in the security proof, where the adversary submits only one set of plaintexts and the challenger chooses to either encrypt these plaintexts or random plaintexts. We will denote this game with $\mathsf{AO}^{\mathsf{RoR}}$ and use $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ and $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ to denote whether the challenger encrypts the *real* plaintexts or *random* plaintexts (this is analogous to the definition of AO_0 and AO_1).

In this section we point out that $\mathsf{AO}^{\mathsf{RoR}}$ as used in the literature is not well defined and that it seems hard to fix the definition without changing at least the input to the challenge query/oracle. We also strongly recommend not to use the real-or-random version of aggregator obliviousness without first giving an exact definition of it that solves the issue that we describe here.

In all the above-mentioned papers, $\mathsf{AO}^{\mathsf{RoR}}$ is only described in a few sentences, but not formally defined.[5] The crucial question is how the balance-condition is

[5] E.g. in [1] this is explained with only the following statement: "Secondly, we change the challenge phase to its real-or-random version i.e. instead of having the adversary

handled. Remember that in the AO game the balance-condition requires that, if $\mathcal{A}$ can use the aggregator's capability to compute the sum of the plaintext, then the sum of both sets of plaintexts submitted by $\mathcal{A}$ in the challenge query needs to be equal. Thus, in the AO game, the adversary is responsible for adhering to the balance condition.

However, at the time when $\mathcal{A}$ asks the challenge query it is not necessarily clear whether the balance condition needs to be fulfilled. For example $\mathcal{A}$ can first ask a challenge query for all clients, i.e. $\mathcal{U} = [n]$ and later corrupt the aggregator. At the time of the challenge query the balance condition did not need to be fulfilled, but after $\mathcal{A}$ corrupted the aggregator, it does need to be fulfilled. In this case it was the choice of $\mathcal{A}$ whether to submit a challenge query that would adhere to the balance condition, i.e. satisfying $\sum_{i\in\mathcal{U}} x_i^0 = \sum_{i\in\mathcal{U}} x_i^1$. However, it is unclear how the challenger $\mathcal{C}_{\mathsf{RoR}}$ in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game would answer the challenge query $\mathsf{QChallenge}(\mathcal{U}, \{x_i\}_{i\in\mathcal{U}}, l^*)$. One option is that $\mathcal{C}_{\mathsf{RoR}}$ would encrypt random $\{r_i\}_{i\in\mathcal{U}}$. But then $\mathcal{A}$ could trivially win the game by corrupting the aggregator, decrypting the sum of the plaintexts and checking if it matches the sum of the submitted challenge plaintexts $\{x_i\}_{i\in\mathcal{U}}$. The challenger could also choose random r_i with the constraint that $\sum_{i\in\mathcal{U}} r_i = \sum_{i\in\mathcal{U}} x_i$, thus, always adhering to the balance condition. However, this is a restriction compared to the normal AO game, where $\mathcal{A}$ can specify two sets of challenge messages with *different* sums. In particular, when $\mathcal{A}$ submits a challenge query for a single user i^*, the challenger in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game would have to choose $r_i^* = x_i^*$ making it impossible for $\mathcal{A}$ to distinguish between $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ and $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$. So it is unclear whether such a version of real-or-random security still implies AO security.

One way to fix the definition is to let $\mathcal{A}$ specify a bit b_{sum} in the challenge query, indicating whether $\mathcal{A}$ wants to receive challenge ciphertexts that adhere to the balance condition. In more detail, in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game, when $b_{\text{sum}} = 1$, the challenger chooses random plaintexts $\{r_i\}_{i\in\mathcal{U}}$ with $\sum_{i\in\mathcal{U}} r_i = \sum_{i\in\mathcal{U}} x_i$. In the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game, when $b_{\text{sum}} = 0$, the challenger chooses completely random plaintexts. In the $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ game, the challenger ignores b_{sum}. Thereby, as in the AO game, it would be $\mathcal{A}$'s choice whether to adhere to the balance condition or not. Formally, we have the following definition:

Definition 4 (Real-or-random aggregator obliviousness). *Let the message space of the PSA scheme be a group $(\mathbb{G}, +)$. We define the real-or-random security notion of aggregator obliviousness via the security experiments $\mathsf{AO}^{\mathsf{RoR}}_{real}$ and $\mathsf{AO}^{\mathsf{RoR}}_{random}$ given in Figures 4 and 5. In both games, the challenger runs Setup and passes the public parameters pp to the adversary $\mathcal{A}$. Then, the adversary may adaptively query three oracles, where queries to the QEnc and the $\mathsf{QCorrupt}$ oracle are handled exactly as in Definition 2. At the end, $\mathcal{A}$ outputs a guess α of whether $b = 0$ or $b = 1$.*

---

specify two sets of plaintext-randomness pairs $\{d_i, r_i\}$ and $\{d_i', r_i'\}$ and have her distinguish between encryptions of either one, we let the adversary pick one set $\{d_i, r_i\}$ and have her distinguish between a set of valid encryptions and a set of random values in $\mathbb{Z}_q^\lambda$."

Here, instead of the normal balance condition (which is item 4 of condition (∗)), we require the following condition (∗∗) in order to output α: If $\mathcal{A}$ has corrupted the aggregator and $\mathcal{Q}_{l^*} \cup \mathcal{CS} = [n]$ *then we require that* $b_{sum} = 1$.

We define $\mathcal{A}$'s advantage as

$$\mathsf{Adv}^{\mathsf{AO\text{-}RoR}}_{\mathcal{A},\mathsf{PSA}}(\lambda, n) = |\Pr[\mathsf{AO}^{\mathsf{RoR}}_{real} = 1] - \Pr[\mathsf{AO}^{\mathsf{RoR}}_{random} = 1]|$$

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

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

$\mathsf{AO}^{\mathsf{RoR}}_{\mathrm{real}}$

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

$\alpha \leftarrow \mathcal{A}^{\mathsf{QCorrupt}(\cdot),\mathsf{QEnc}(\cdot,\cdot,\cdot),\mathsf{QChallenge}^{\mathsf{RoR}}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp})$

`if` items 1 to 3 of condition $(*)$ are satisfied (see p. 5)

  `if` condition $(**)$ is satisfied (see p. 17)

    `output` α

`else`

  `output` 0

$\underline{\mathsf{QChallenge}^{\mathsf{RoR}}}(\mathcal{U} = \{u_1, \ldots, u_m\}, \{x_i\}_{i\in\mathcal{U}}, b_{\mathrm{sum}}, l^*)$:

`return` $\{\mathsf{Enc}(\mathsf{sk}_i, x_i, l^*)\}_{i\in\mathcal{U}}$

Fig. 4. The real part of the real-or-random aggregator obliviousness experiment and the modified challenge oracle that we propose.

Note that our definition requires that the message space of the PSA scheme is a group. Let us explain the problem that arises, when this is not the case. For this paragraph, w.l.o.g. let $\mathcal{U} = [m] = \{1, \ldots, m\}$. In the case of $\mathsf{AO}^{\mathsf{RoR}}_{\mathrm{random}}$, when the challenger $\mathcal{C}$ wants to choose random $r_i \in \mathbb{Z}_R$ with $\sum_{i\in\mathcal{U}} r_i = \sum_{i\in\mathcal{U}} x_i =: X$, they choose $\{r_i \leftarrow\!\$\ \mathbb{Z}_R\}_{i\in[m-1]}$ and set $r_m := X - \sum_{i\in[m-1]} r_i$. This ensures that the $\{r_i\}_{i\in\mathcal{U}}$ are uniformly random with the only constraint that $\sum_{i\in\mathcal{U}} r_i = X$. The problem arises when the message space is not a group. For example consider the case where the message space is $\{0, \ldots, 10\}$ and the sum shall not be computed modulo 10, but over the integers. When X is e.g. 4 and $\mathcal{C}$ chooses $\{r_i \leftarrow\!\$\ \{0, \ldots, 10\}\}_{i\in[m-1]}$, then it is very likely that $\sum_{i\in[m-1]} r_i > 4$, whereby r_m would need to be a negative number, which it not in the message space. One option for the challenger would be to uniformly sample from the set of all tuples $(r_i \in \{0, \ldots, 10\})_{i\in\mathcal{U}}$ that satisfy $\sum_{i\in\mathcal{U}} r_i = X$. However, it is unclear how to do this efficiently. Therefore, if one wants to define $\mathsf{AO}^{\mathsf{RoR}}$ in a setting where the

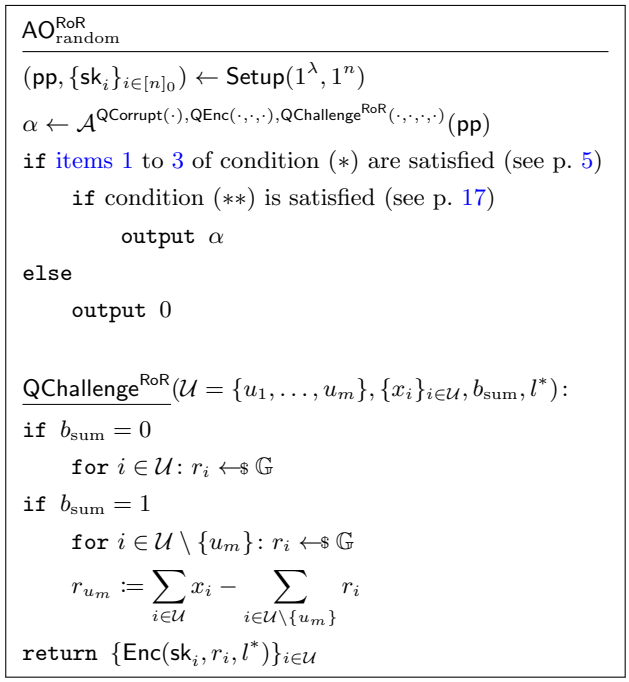

Fig. 5. The random part of the real-or-random aggregator obliviousness experiment and the modified challenge oracle that we propose. Here, in the penultimate line of the challenge query, we require the message space to be a group. Otherwise, it is not clear how to efficiently sample random elements, such that they sum to the sum of the x_i.

message space is not a group one has to specify how the challenger can choose the r_i efficiently in a way that they are properly distributed. All in all, the definition of AO security seems to be more robust than the definition of $\mathsf{AO}^{\mathsf{RoR}}$ security.

Nevertheless, because our real-or-random definition may be useful in security proofs, let us show that our version of the real-or-random AO security is equivalent to AO security. Importantly, this does not automatically fix the affected security proofs, because our version of $\mathsf{AO}^{\mathsf{RoR}}$ is different in that it requires the attacker to specify the bit b_{sum} in the challenge query. Exploring how our definition of $\mathsf{AO}^{\mathsf{RoR}}$ can replace the not-well-defined definition of real-or-random security would be worthwhile for future work.

Theorem 1. *Let* $\mathsf{PSA} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{AggrDec})$ *be a private stream aggregation scheme over a group* $\mathbb{G}$*. Then,* PSA *is AO secure if and only if it is Real-or-Random AO secure.*

Proof. First we show that $\mathsf{AO} \implies \mathsf{AO}^{\mathsf{RoR}}$, by giving a reduction $\mathcal{B}$ that uses an attacker $\mathcal{A}$ on $\mathsf{AO}^{\mathsf{RoR}}$ to win the AO game. First, $\mathcal{B}$ forwards the public parameters pp to $\mathcal{A}$. When $\mathcal{A}$ asks a corruption or encryption query, $\mathcal{B}$ simply forwards this query to their AO challenger $\mathcal{C}$. When $\mathcal{A}$ asks a query $\mathsf{QChallenge}^{\mathsf{RoR}}(\mathcal{U} =$

$\{u_1, \ldots, u_m\}, \{x_i\}_{i \in \mathcal{U}}, b_{\text{sum}}, l^*)$, then $\mathcal{B}$ chooses $\{r_i\}_{i \in \mathcal{U}}$ as in the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game. Concretely, if $b_{\text{sum}} = 0$, then the r_i are completely random and if $b_{\text{sum}} = 1$, then the r_i are random conditioned on $\sum_{i \in \mathcal{U}} r_i = \sum_{i \in \mathcal{U}} x_i$. Next, $\mathcal{B}$ asks a challenge query $\mathsf{QChallenge}(\mathcal{U} = \{u_1, \ldots, u_m\}, \{x_i\}_{i \in \mathcal{U}}, \{r_i\}_{i \in \mathcal{U}}, l^*)$ to $\mathcal{C}$ and simply forwards the answer to $\mathcal{A}$. When $\mathcal{A}$ submits a guess b whether they play $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ or $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$, $\mathcal{B}$ simply forwards this guess to $\mathcal{A}$. Note that if $\mathcal{C}$'s challenge bit is 0, then $\mathcal{C}$ answers with an encryption of the left messages, i.e. $\{x_i\}_{i \in \mathcal{U}}$ and, thus, $\mathcal{B}$ perfectly simulates $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ to $\mathcal{A}$. If $\mathcal{C}$'s challenge bit is 1, then $\mathcal{C}$ answers with an encryption of the right messages, i.e. $\{r_i\}_{i \in \mathcal{U}}$ and, thus, $\mathcal{B}$ perfectly simulates $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ to $\mathcal{A}$. Also note that $\mathcal{B}$ needs to fulfill the balance condition if and only if $\mathcal{A}$ has to fulfill the balance condition, as they both ask the corruption encryption and challenge queries for the same users. Finally, observe that $\mathcal{B}$ does indeed fulfill the balance condition if $\mathcal{A}$ fulfills condition $(**)$, because if $b_{\text{sum}} = 1$, $\mathcal{B}$ makes sure that $\sum_{i \in \mathcal{U}} r_i = \sum_{i \in \mathcal{U}} x_i$. Thus, $\mathcal{B}$ has the same advantage as $\mathcal{A}$.

Next, we show that $\mathsf{AO}^{\mathsf{RoR}} \implies \mathsf{AO}$, by giving a reduction $\mathcal{B}$ that uses an attacker $\mathcal{A}$ on AO to win the $\mathsf{AO}^{\mathsf{RoR}}$ game. As before, $\mathcal{B}$ simply forwards the corruption and encryption queries from $\mathcal{A}$ to $\mathcal{C}$ and gives the answers back to $\mathcal{A}$. When $\mathcal{A}$ asks a challenge query $\mathsf{QChallenge}(\mathcal{U}, \{x_i^0\}_{i \in \mathcal{U}}, \{x_i^1\}_{i \in \mathcal{U}}, l^*)$, $\mathcal{B}$ sets $b_{\text{sum}} = 1$, if $\sum_{i \in \mathcal{U}} x_i^0 = \sum_{i \in \mathcal{U}} x_i^1$ and $b_{\text{sum}} = 0$, otherwise. Also, $\mathcal{B}$ chooses $b' \leftarrow\!\!\$\ \{0, 1\}$. Then $\mathcal{B}$ sends a challenge query $\mathsf{QChallenge}^{\mathsf{RoR}}(\mathcal{U}, \{x_i^{b'}\}_{i \in \mathcal{U}}, b_{\text{sum}}, l^*)$ to $\mathcal{C}$ and forwards the answer to $\mathcal{A}$. When $\mathcal{A}$ submits a guess b, $\mathcal{B}$ sets $b^* = 0$ (meaning that $\mathcal{B}$ guesses $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$), if $b = b'$. If $b \neq b'$, then $\mathcal{B}$ sets $b^* = 1$ (meaning that $\mathcal{B}$ guesses $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$). Finally, $\mathcal{B}$ sends b^* to $\mathcal{C}$. Because $\mathcal{B}$ sets $b_{\text{sum}} = 1$ iff $\sum_{i \in \mathcal{U}} x_i^0 = \sum_{i \in \mathcal{U}} x_i^1$ and $\mathcal{B}$ asks the corruption, encryption and challenge query for the same set of users as $\mathcal{A}$, $\mathcal{B}$ fulfills condition $(**)$ if $\mathcal{A}$ fulfills the balance condition. The advantage of $\mathcal{B}$ is non-negligible, if $\mathcal{A}$'s advantage is non-negligible. To see this, observe that in the case that $\mathcal{C}$ plays the $\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}$ game, $\mathcal{B}$ perfectly simulates the $\mathsf{AO}_{b'}$ game to $\mathcal{A}$ and that $\mathcal{C}$ correctly guesses $b^* = 0$ iff $\mathcal{A}$ correctly guesses b'. Hence, $\Pr[\mathsf{AO}^{\mathsf{RoR}}_{\text{real}}(\lambda, n, \mathcal{A}) = 1]$ equals the probability that $\mathcal{A}$ wins the AO game, which is at least $\frac{1}{2} + \mu(\lambda)$ for a non-negligible μ.

In the case that $\mathcal{C}$ plays the $\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}$ game, b will be independent from the randomly chosen b' and hence $b = b'$ (i.e. the output b^*) is uniformly random. Hence, in this case, $\Pr[\mathsf{AO}^{\mathsf{RoR}}_{\text{random}}(\lambda, n, \mathcal{A}) = 1] = \frac{1}{2}$ and $\mathcal{B}$'s overall advantage is $\frac{1}{2} + \mu(\lambda) - \frac{1}{2} = \mu(\lambda)$, which is non-negligible. □

To summarize, although the real-or-random version of AO security is widely used in the literature, it turns out to not be well defined. We recommend to use the standard AO security definition. However, for the case that a real-or-random definition is desired *and* the message space is a group, with Definition 4 we offer such a definition, which is equivalent to standard AO security.

6 Conclusion

There is an argument to be made that the nearer to actual employment a published scheme is, the more important it is to make sure that its privacy

guarantees actually hold true. When a patented scheme that was published at a top-level security conference is attacked almost five years after its publication, this might be particularly alarming.

While we are not aware of actual employments of this scheme, it used to be the natural choice, if one wanted to use a post-quantum-secure private stream aggregation. More importantly, the proof mistakes have spread to other papers. We aim to rectify this situation, by not only attacking the original scheme and tracking down the effects of the proof flaw in the overall PSA literature, but also by furthering a discussion on the formulation and specification of the security guarantees, such as on how or whether to use the real-or-random version of the security game.

Acknowledgments. We thank the anonymous reviewers for their helpful and constructive feedback. This work was partly supported by KASTEL Security Research Labs. Johannes Ottenhues was supported by the EU-funded Marie Curie ITN TReSPAsS-ETN project under the grant agreement 860813 and Alexander Koch was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

References

[1] D. Becker, J. Guajardo, and K.-H. Zimmermann. "Revisiting Private Stream Aggregation: Lattice-Based PSA." In: *NDSS 2018*. The Internet Society, 2018. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02B-3_Becker_paper.pdf.

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

[3] F. Benhamouda, M. Joye, and B. Libert. "A new framework for privacy-preserving aggregation of time-series data". In: *ACM Transactions on Information and System Security (TISSEC)* 18.3 (2016), 10:1–10:21. DOI: 10.1145/2873069.

[4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *CCS 2017*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM, 2017, pp. 1175–1191. DOI: 10.1145/3133956.3133982.

[5] C. Dwork, A. Roth, et al. "The algorithmic foundations of differential privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. DOI: 10.1561/0400000042.

[6] R. El Bansarkhani, Ö. Dagdelen, and J. Buchmann. "Augmented learning with errors: The untapped potential of the error term". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 333–352. DOI: 10.1007/978-3-662-47854-7_20.

[7] J. Ernst and A. Koch. "Private Stream Aggregation with Labels in the Standard Model". In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 117–138. DOI: 10.2478/popets-2021-0063.

[8] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu. "Efficient and Privacy-Enhanced Federated Learning for Industrial Artificial Intelligence". In: *IEEE Trans. Ind. Informatics* 16.10 (2020), pp. 6532–6542. DOI: 10.1109/TII.2019.2945367.

[9] R. Karl, J. Takeshita, A. Mohammed, A. Striegel, and T. Jung. "CryptoGram: Fast Private Calculations of Histograms over Multiple Users' Inputs". In: *Distributed Computing in Sensor Systems, DCOSS 2021*. IEEE, 2021, pp. 25–34. DOI: 10.1109/DCOSS52077.2021.00017.

[10] R. Karl, J. Takeshita, A. Mohammed, A. Striegel, and T. Jung. "Cryptonomial: A Framework for Private Time-Series Polynomial Calculations". In: *Security and Privacy in Communication Networks, SecureComm 2021*. Vol. 398. Full version with flawed proof at https://eprint.iacr.org/2021/473. Springer, 2021, pp. 332–351. DOI: 10.1007/978-3-030-90019-9_17.

[11] H. Karthikeyan and A. Polychroniadou. "OPA: One-shot Private Aggregation with Single Client Interaction and its Applications to Federated Learning". In: *IACR Cryptol. ePrint Arch.* (2024), p. 723. URL: https://eprint.iacr.org/2024/723.

[12] V. Lyubashevsky, C. Peikert, and O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *EUROCRYPT 2010*. Ed. by H. Gilbert. Vol. 6110. LNCS. Springer, 2010, pp. 1–23. DOI: 10.1007/978-3-642-13190-5_1.

[13] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *STOC 2005*. Ed. by H. N. Gabow and R. Fagin. ACM, 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

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

[15] J. Takeshita, Z. Carmichael, R. Karl, and T. Jung. "TERSE: Tiny Encryptions and Really Speedy Execution for Post-Quantum Private Stream Aggregation". In: *Security and Privacy in Communication Networks, SecureComm 2022*. Ed. by F. Li, K. Liang, Z. Lin, and S. K. Katsikas. Vol. 462. Springer, 2022, pp. 331–352. DOI: 10.1007/978-3-031-25538-0_18.

[16] J. Takeshita, R. Karl, T. Gong, and T. Jung. "SLAP: Simpler, Improved Private Stream Aggregation from Ring Learning with Errors". In: *Journal of Cryptology* 36.8 (2 2023). DOI: 10.1007/s00145-023-09450-w.

[17] H. Waldner, T. Marc, M. Stopar, and M. Abdalla. "Private Stream Aggregation from Labeled Secret Sharing Schemes". In: *IACR Cryptol. ePrint Arch.* (2021), p. 81. URL: https://eprint.iacr.org/2021/081.

A Another paper with the same problems as [1]

The authors of [8] use a PSA scheme to make federated learning more secure. In federated learning multiple clients locally train a machine learning model on their local data and then send the model updates to a server. The server then

computes the average of the model updates and sends the updated model back to the clients. This process is usually repeated until the model is considered good enough. A problem is that the aggregator learns the model updates of the individual clients. This may allow the server to get information about the (potentially privacy sensitive) data of the clients via model inversion attacks. By using a PSA scheme to allow the server to only learn the sum of all clients' updates, this threat can be alleviated.

In [8] the authors essentially use the PSA scheme of [1] and the same attacks are possible there. This means that the server (i.e. the aggregator) can learn the differences of the clients' model updates. Furthermore, the server can, given enough ciphertexts, compute the secret keys and, thus, decrypt all ciphertexts. This makes the additional security guarantees void and again allows the server to perform model inversion attacks to recover the sensitive training data of the clients.

This can be quite problematic. Assume for example that the machine learning model is a neural network that is being trained for use in an biometric authentication system. The training data would consist of the biometric data of the clients (e.g. face images or voice data). Keeping this data private is very important, as it can easily reveal sensitive attributes like age and gender, but may also allow to infer the client's health condition. Another, less immediate, danger is that the server may learn the biometric information, which the clients would later use for biometric authentication. Thus, the server (or anyone who steals the servers data) can potentially impersonate many clients.

B $\mathsf{AO}^{\mathsf{co}}$ security is trivially achievable

In this section, we remark that $\mathsf{AO}^{\mathsf{co}}$ security is an insufficient, weak security notion and that a PSA scheme with this level of security can already be achieved using a very simple one-time pad construction. Importantly, this scheme would become very insecure in actual practice, where the secret keys (one-time pads) of the clients are likely to be used multiple times. (This multiple use in the $\mathsf{AO}^{\mathsf{co}}$ game will lead to the adversary loosing, and hence they cannot win with such a move). The scheme has already been described textually in a footnote 9 in [17]. Here we formally write down the scheme and give a security proof.

For illustration, let us give the construction in Figure 6 and the corresponding simple security proof. At the setup, each client gets a random value, and the aggregator gets the (additive inverse of the) sum of these (the aggregator key). For encryption of the clients plaintexts, we just mask them with the respective key. The decryption of the aggregator proceeds by summing all ciphertexts and adding their key. Syntactically, we define the scheme with support for labels, but these are ignored in the encryption step. Hence, even if one were to only encrypt once per label, the one-time-pad would be used multiple times, which makes the scheme insecure in actual practice. (As discussed before, the security notion is not strong enough to capture this.)

$\underline{\mathsf{Setup}(1^\lambda, 1^n):}$	$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i, l):}$
for $i \in [n]$: $\mathsf{sk}_i \leftarrow\!\!\$\ \mathbb{Z}_q$	**return** $x_i + \mathsf{sk}_i$
$\mathsf{sk}_0 := -\sum_{i\in[n]} \mathsf{sk}_i$	
$\mathsf{pp} := q$ (the modulus)	$\underline{\mathsf{AggrDec}(\mathsf{pp}, \mathsf{sk}_0, l, \{c_i\}_{i\in[n]}):}$
return $(\mathsf{pp}, \mathsf{sk}_0, \{\mathsf{sk}_i\}_{i\in[n]})$	**return** $\sum_{i\in[n]} c_i + \mathsf{sk}_0$

Fig. 6. A "trivial" PSA construction, where the clients' ciphertext is encrypted via a one-time pad.

Proposition 1. *The PSA scheme given in Figure 6 is a perfectly $\mathsf{AO}^{\mathsf{co}}$-secure PSA scheme (with labels).*

Proof. Correctness holds, as for any security parameter λ, and any number of clients n, any $(\mathsf{pp}, \mathsf{sk}_0, \{\mathsf{sk}_i\}_{i\in[n]})$ in the image of $\mathsf{Setup}(1^\lambda, 1^n)$, and for any $\{c_i\}_{i\in[n]}$, where for each $i \in [n]$, c_i is from the image of $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}_i, x_i, l)$, where $x_i \in \mathbb{Z}_q$ and l is a label, it holds that

$$\mathsf{AggrDec}(\mathsf{pp}, \mathsf{sk}_0, l, \{c_i\}_{i\in[n]}) =$$
$$\sum_{i\in[n]} (x_i + \mathsf{sk}_i) + \mathsf{sk}_0 = \sum_{i\in[n]} x_i + \left(\sum_{i\in[n]} \mathsf{sk}_i + \mathsf{sk}_0\right) = \sum_{i\in[n]} x_i.$$

For security, observe that everything is perfectly masked with a one-time pad and re-use of the keys is prevented due to the balance condition and by disallowing Enc-queries. More formally, we argue that the games $\mathsf{AO}_0^{\mathsf{co}}(\lambda, n, \mathcal{A})$ and $\mathsf{AO}_1^{\mathsf{co}}(\lambda, n, \mathcal{A})$ are perfectly indistinguishable for any (unbounded) adversary $\mathcal{A}$, as follows.

For notation, note that the set $\mathcal{Q}_{l^*}$ of users from the challenge query and the encryption queries for label l^* (see Definition 2) simplifies to $\mathcal{Q}_{l^*} = \mathcal{U}$ (because there are no encryption queries), where $\mathcal{U} \subseteq [n]$ is the set of clients in the challenge query. As before, $\mathcal{CS} \subseteq [n]$ denotes the set of indices asked in corruption queries. We distinguish two cases.

Case 1: The balance-condition needs to be fulfilled. In this case, the adversary $\mathcal{A}$ has corrupted the aggregator and $\mathcal{U} \cup \mathcal{CS} = [n]$, i.e. $\mathcal{A}$ got either the secret key or a ciphertext from every party. Now, $\{\mathsf{sk}_i\}_{i\in[n]_0}$ is a perfect additive $(n+1)$-out-of-$(n+1)$-secret sharing of 0. Therefore, the challenge ciphertexts together with the other secret keys

$$S_0 := \{c_i^0 = x_i^0 + \mathsf{sk}_i\}_{i\in\mathcal{U}} \cup \{\mathsf{sk}_i\}_{[n]_0\setminus\mathcal{U}} \text{ and}$$
$$S_1 := \{c_i^1 = x_i^1 + \mathsf{sk}_i\}_{i\in\mathcal{U}} \cup \{\mathsf{sk}_i\}_{[n]_0\setminus\mathcal{U}}$$

is a perfect secret sharing of $\sum_{i\in\mathcal{U}} x_i^0$ and $\sum_{i\in\mathcal{U}} x_i^1$, respectively. Because the balance-condition requires both sums to be equal, S_0 and S_1 are both a perfect secret sharing of the same value and, thus, perfectly indistinguishable.

Case 2: The balance-condition does not need to be fulfilled. In this case, the aggregator is not corrupted or $\mathcal{U} \cup \mathcal{CS} \neq [n]$. The intuition for the proof is that the secret keys are an additive $(n+1)$-out-of-$(n+1)$-secret sharing of 0. However, $\mathcal{A}$ is lacking at least one share and, thus, has no information about the secret keys, whereby the challenge ciphertexts are proper one-time-pad encryptions.

In the $\mathsf{AO}^{\mathsf{co}}$ game the adversary can only ask corruption queries and one challenge query and $\mathcal{A}$ is not allowed to specify corrupted users in the challenge query. If the aggregator is not corrupted, the client secret keys sk_i are independent and uniformly random values for $\mathcal{A}$. Therefore, before asking the challenge query, $\mathcal{A}$ has no information about the secret keys of the non-corrupted clients. In the challenge query, $\mathcal{A}$ gets exactly one ciphertext per (non-corrupted) user in $\mathcal{U}$. Because Enc is a one-time-pad, the challenge ciphertexts $\{c_i^0 = x_i^0 + \mathsf{sk}_i\}_{i \in \mathcal{U}}$ and $\{c_i^1 = x_i^1 + \mathsf{sk}_i\}_{i \in \mathcal{U}}$ are identically distributed.

If the aggregator is corrupted, but $\mathcal{U} \cup \mathcal{CS} \neq [n]$, there is at least one client i^* that is neither corrupted nor in $\mathcal{U}$. Therefore, $\{\mathsf{sk}_i\}_{i \in [n]_0 \setminus \{i^*\}}$ are independent and uniformly random values from $\mathcal{A}$'s point of view. Thus, the above argument applies again that the answers to the challenge query $\{c_i^0 = x_i^0 + \mathsf{sk}_i\}_{i \in \mathcal{U}}$ and $\{c_i^1 = x_i^1 + \mathsf{sk}_i\}_{i \in \mathcal{U}}$ are one-time-pad encryptions where $\mathcal{A}$ has no previous information about the secret keys $\{\mathsf{sk}_i\}_{i \mathcal{U}}$ (due to $(*)$) and, thus, are identically distributed. □

Note that this scheme could even be said to support (arbitrary many) labels (which are effectively ignored, however), as the labels are only relevant for encryption and challenge queries, where the former are not allowed in the weak security game and the latter are allowed exactly once, hence ciphertexts are learned for exactly one label. In the construction, the secret keys are exactly one element in $\mathbb{Z}_q$ (where the client secret keys could be compressed to a string of length λ using a pseudorandom number generator (PRG), if q is larger that λ, reducing the security to the computational notion).

However, we remark that the trivial scheme does not feature full AO-security as follows:

Proposition 2. *The PSA scheme given in Figure 6 is not (computationally) AO-secure (with labels).*

Proof. Consider the following attack of a PPT adversary $\mathcal{A}$:

1. $\mathcal{A}$ receives the public parameters pp from the challenger and asks an encryption query for an arbitrary user i^* on message $x_{i^*} = 0$ and an arbitrary label l. The challenger returns $c_{i^*} = x_{i^*} + \mathsf{sk}_{i^*} = \mathsf{sk}_{i^*}$, so now $\mathcal{A}$ knows sk_{i^*}.
2. $\mathcal{A}$ asks a challenge query for $\mathcal{U} = \{x_{i^*}\}$ on a different label l^* with $x_{i^*}^0 \neq x_{i^*}^1$. The challenger returns $c_{i^*}^b = x_{i^*}^b + \mathsf{sk}_{i^*}$.
3. Because $\mathcal{A}$ knows sk_{i^*}, they can compute $c_{i^*}^b - \mathsf{sk}_{i^*} = x_{i^*}^b$, check whether this is equal to $x_{i^*}^0$ or $x_{i^*}^1$ and then submit b to the challenger.

Because $\mathcal{A}$'s queries adhere to condition $(*)$, $\mathcal{A}$ always wins the AO game. □