INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

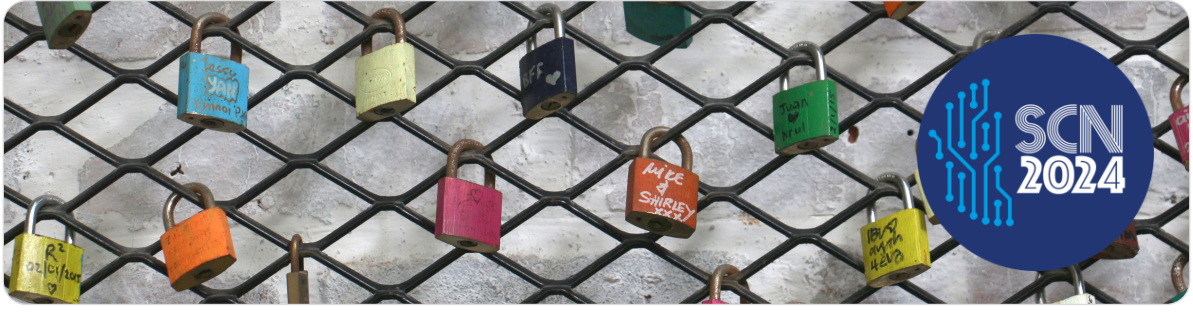
Université Paris Cité | cnrs

University of St.Gallen

LaPSuS: A Lattice-Based PSA Scheme under Scrutiny

SCN 2024, International Conference on Security and Cryptography for Networks

Johannes Ottenhues (Univ. St. Gallen), Alexander Koch (IRIF, Univ. Paris Cité, CNRS) | 2024-09-11



Private Stream Aggregation (PSA)

Setting: Each client has a sensor, producing a private value $x_i \in \mathbb{Z}_q$

x_1

x_2

x_3

x_4

What is $\sum_{i \in [n]} x_i$**?**

Private Stream Aggregation (PSA)

Setting: Each client has a sensor, producing a private value $x_{i,\ell} \in \mathbb{Z}_q$ at time step ℓ

$\ell =$ 


$x_{1,\ell}$


$x_{2,\ell}$


$x_{3,\ell}$


$x_{4,\ell}$

What is $\sum_{i \in [n]} x_{i,\ell}$ for a given time step ℓ?

Private Stream Aggregation (PSA)

Setting: Each client has a sensor, producing a private value $x_{i,\ell} \in \mathbb{Z}_q$ at time step ℓ

$\ell =$ 


$x_{1,\ell}$


$x_{2,\ell}$


$x_{3,\ell}$


$x_{4,\ell}$

What is $\sum_{i \in [n]} x_{i,\ell}$ for a given time step ℓ?

Private Stream Aggregation (PSA)

Setting: Each client has a sensor, producing a private value $x_{i,\ell} \in \mathbb{Z}_q$ at time step ℓ

Aggregator

Image “server multiple” by RRZE, CC BY-SA 3.0

Private Stream Aggregation (PSA)

PSA: A 3-tuple of efficient algorithms (Setup, Enc, AggrDec)

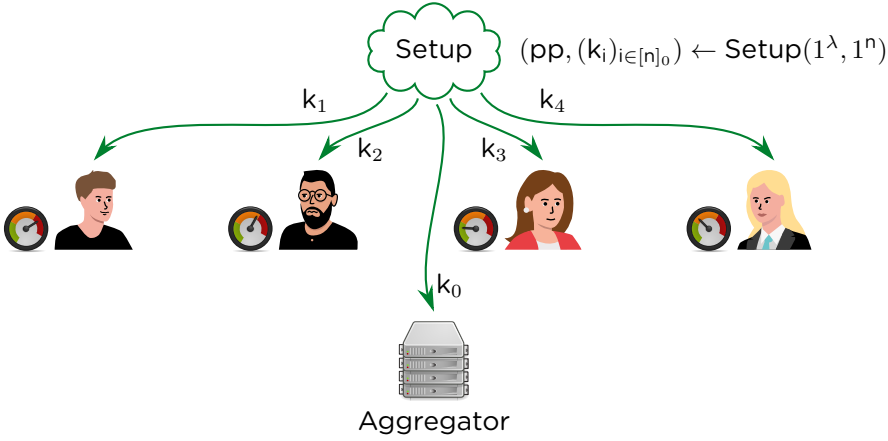


Image “server multiple” by RRZE, CC BY-SA 3.0

Private Stream Aggregation (PSA)

PSA: A 3-tuple of efficient algorithms (Setup, Enc, AggrDec)

$\ell =$ 

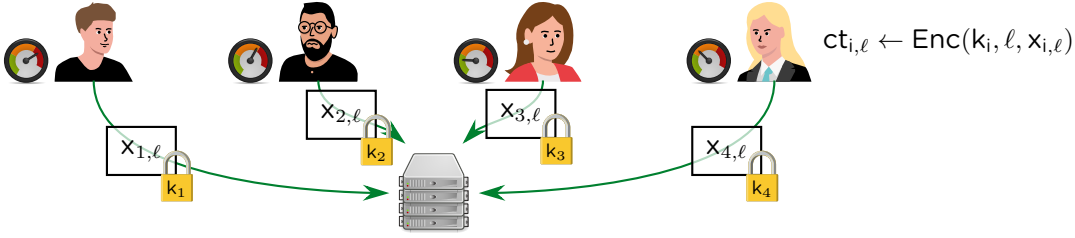


Image “server multiple” by RRZE, CC BY-SA 3.0

Private Stream Aggregation (PSA)

PSA: A 3-tuple of efficient algorithms (Setup, Enc, AggrDec)

$\ell =$ 

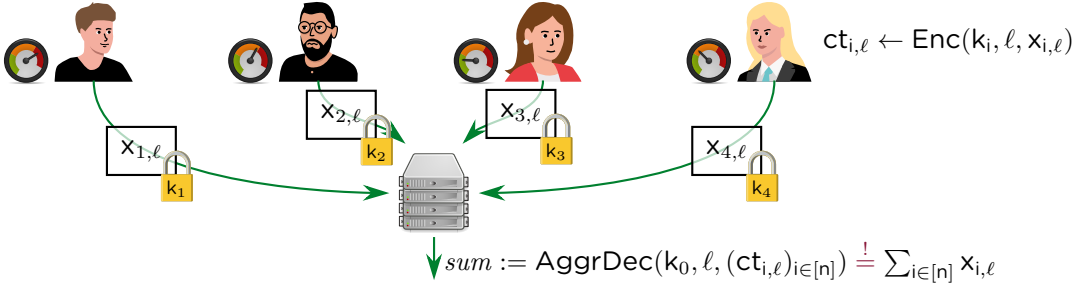


Image "server multiple" by RRZE, CC BY-SA 3.0

Private Stream Aggregation (PSA)

A note on **inherent leakage** if you get more than one ct per client and label:

$\ell =$ 

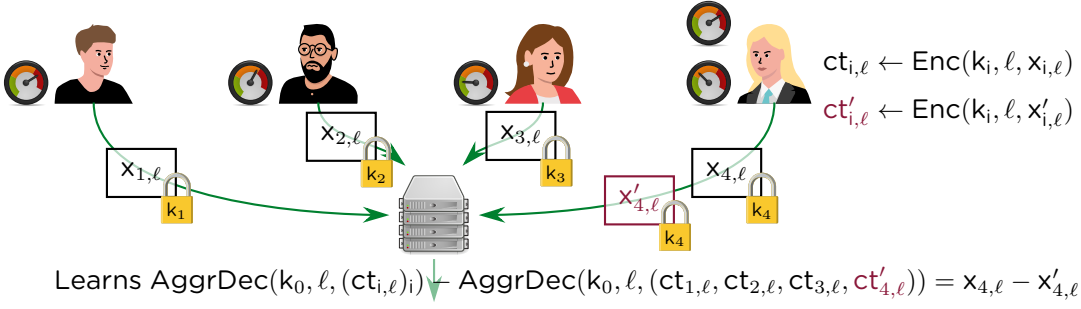


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Attacker gets pp and oracles: QEnc(i, x_i, ℓ), QCorrupt(i), QChallenge$(\mathcal{U}, (x_i^0)_{i\in\mathcal{U}}, (x_i^1)_{i\in\mathcal{U}}, \ell^*)$

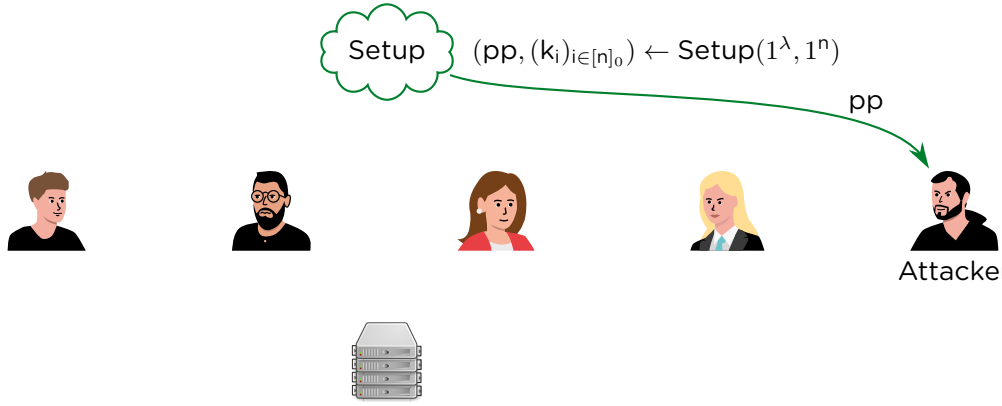


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Attacker gets pp and oracles: QEnc(i, x_i, ℓ), QCorrupt(i), QChallenge($\mathcal{U}$, $(x_i^0)_{i\in\mathcal{U}}$, $(x_i^1)_{i\in\mathcal{U}}$, ℓ^*)

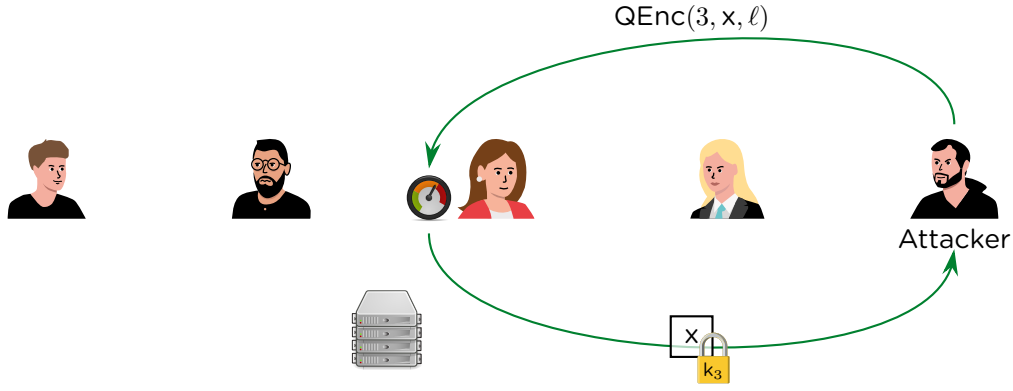


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Attacker gets pp and oracles: QEnc(i, x_i, ℓ), QCorrupt(i), QChallenge$(\mathcal{U}, (x_i^0)_{i\in\mathcal{U}}, (x_i^1)_{i\in\mathcal{U}}, \ell^*)$

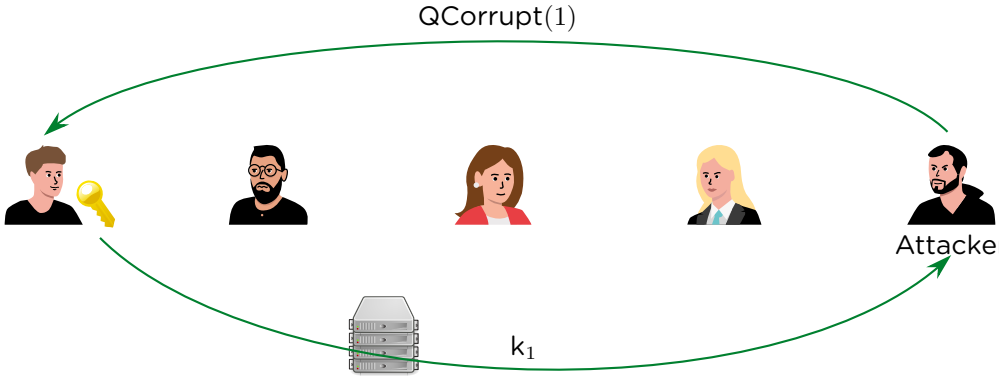


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Attacker gets pp and oracles: QEnc(i, x_i, ℓ), QCorrupt(i), QChallenge$(\mathcal{U}, (x_i^0)_{i\in\mathcal{U}}, (x_i^1)_{i\in\mathcal{U}}, \ell^*)$

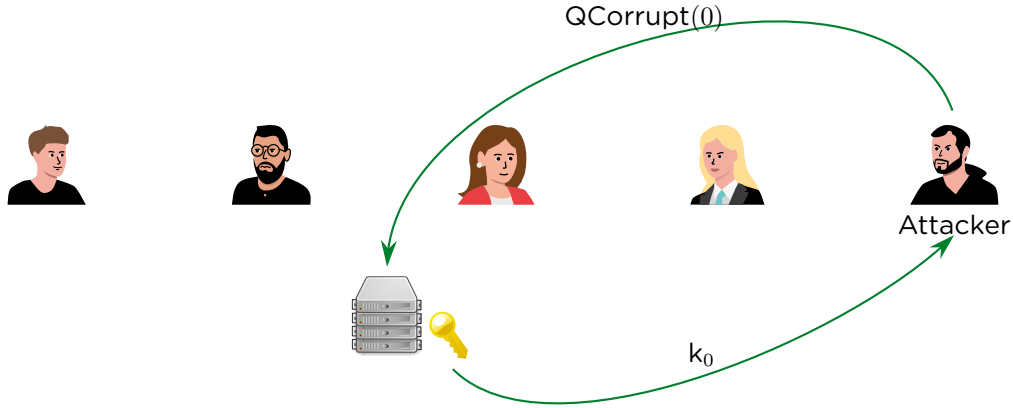


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Attacker gets pp and oracles: QEnc(i, x_i, ℓ), QCorrupt(i), QChallenge($\mathcal{U}$, $(x_i^0)_{i\in\mathcal{U}}$, $(x_i^1)_{i\in\mathcal{U}}$, ℓ^*)

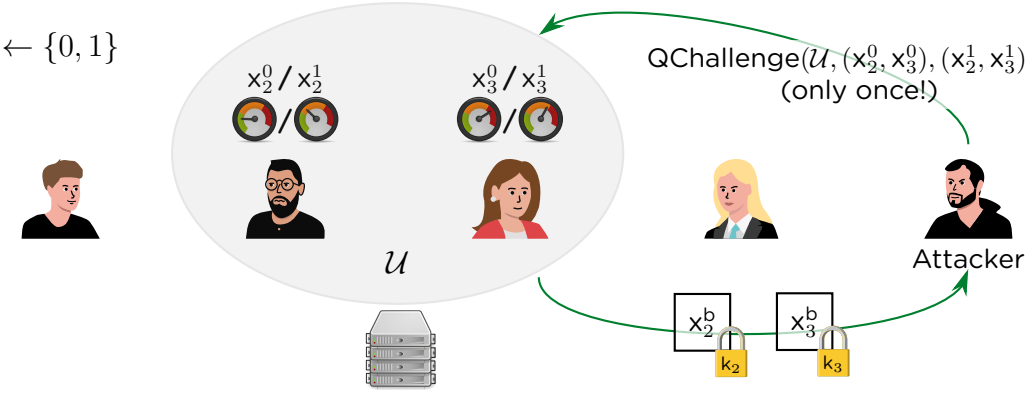


Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Aim of the attacker: Guess bit b.

$b \leftarrow \{0, 1\}$

Attacker Rules

- Keep $\mathcal{U}$ disjoint from $\{\text{corrupted}\} =: \mathcal{CS}$!
- If you corrupted the aggregator **and** $\mathcal{U}$, $\mathcal{CS}$, and the set of users from which you got an encryption for ℓ^* covers everyone, **enforce** $\sum x_i^0 = \sum x_i^1$

... **or you loose!**

Attacker

Image “server multiple” by RRZE, CC BY-SA 3.0

Security Notion: Aggregator Obliviousness (AO)

Aim of the attacker: Guess bit b.

b ←

Encrypt-Once-Rules

- Don't ask for more than one encryption per user and label.
- This includes the challenge query!

Attacker Rules

- Keep $\mathcal{U}$ disjoint from $\{\text{corrupted}\} =: \mathcal{CS}$!
- If you corrupted the aggregator **and** $\mathcal{U}$, $\mathcal{CS}$, and the set of users from which you got an encryption for ℓ^* covers everyone, **enforce** $\sum x_i^0 = \sum x_i^1$

... **or you loose!**

Attacker

LaPS Scheme (Becker et al., NDSS 2018)

- $(\mathsf{Gen_{ah}}, \mathsf{Enc_{ah}}, \mathsf{Dec_{ah}})$: additively homomorphic PKE with pseudorandom ciphertexts.
- $\mathbf{G}$: gadget matrix, i.e. $\mathbf{G} = \mathbf{I} \otimes (1\ 2\ \ldots\ 2^{d-1})$, with $d = O(\lceil \log q \rceil)$
- $\mathbf{e} \leftarrow \mathcal{D}_{\Lambda^{\perp}_{\mathbf{v}}(\mathbf{G}),\sigma}$: error distribution (with $\sigma > 2\sqrt{\lambda}$); defined so that $\mathbf{G} \cdot \mathbf{e} \bmod q = \mathbf{v}$

Assumption: "Augmented LWE" (El Bansarkhani et al.): additionally hides message x inside error-term of LWE sample, via: encode x into pseudorandom $\mathbf{v}$, sample $\mathbf{e}$ from $\mathcal{D}_{\Lambda^{\perp}_{\mathbf{v}}(\mathbf{G}),\sigma}$

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

$\mathbf{A} \leftarrow \mathbb{Z}_q^{\lambda \times d}$

`for` $i \in [n]$: $k_i \leftarrow \mathbb{Z}_q^\lambda$

$(\mathsf{pk_{ah}}, \mathsf{sk_{ah}}) \leftarrow \mathsf{Gen_{ah}}(1^\lambda)$

$pp := (\mathbf{A}, \mathsf{pk_{ah}})$

$k_0 := (\mathsf{sk_{ah}}, \sum_{i \in [n]} k_i)$

`return` $(pp, (k_i)_{i \in [n]_0})$

$\underline{\mathsf{Enc}(pp, k_i, x_i):}$

`parse` $pp = (\mathbf{A}, \mathsf{pk_{ah}})$

$\mathbf{v}_i \leftarrow \mathsf{Enc_{ah}}(\mathsf{pk_{ah}}, x_i)$

$\mathbf{e}_i \leftarrow \mathcal{D}_{\Lambda^{\perp}_{\mathbf{v}_i}(\mathbf{G}),\sigma}$

$ct_i = k_i^\top \mathbf{A} + \mathbf{e}_i^\top \bmod q$

`return` ct_i

$\underline{\mathsf{AggrDec}(pp, k_0, (ct_i)_{i \in [n]}):}$

`parse` $k_0 = (\mathsf{sk_{ah}}, k)$

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

`return` $\mathsf{Dec_{ah}}(\mathsf{sk_{ah}}, \mathbf{G} \cdot \mathbf{e} \bmod q)$

First Attack on LaPS using two Ciphertexts of a Client

Given two encryptions ct_i and ct'_i of x_i and x'_i and $\mathsf{k}_0 = (\mathsf{sk}_{\mathsf{ah}}, \mathsf{k})$.

$$\mathsf{ct}_i - \mathsf{ct}'_i = \mathsf{k}_i^\top \mathbf{A} + \mathbf{e}_i^\top - \mathsf{k}_i^\top \mathbf{A} + \mathbf{e}_i'^\top \bmod \mathsf{q} = \mathbf{e}_i^\top - \mathbf{e}_i'^\top \bmod \mathsf{q}.$$

Then:

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

We can then decrypt to get

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

This is without knowing any other ciphertexts, i.e. goes beyond inherent leakage.

First Attack on LaPS using two Ciphertexts of a Client

Given two encryptions ct_i and ct'_i of x_i and x'_i and $\mathsf{k}_0 = (\mathsf{sk}_{\mathsf{ah}}, \mathsf{k})$.

$$\mathsf{ct}_i - \mathsf{ct}'_i = \mathsf{k}_i^\top \mathbf{A} + \mathbf{e}_i^\top - \mathsf{k}_i^\top \mathbf{A} + \mathbf{e}_i'^\top \bmod \mathsf{q} = \mathbf{e}_i^\top - \mathbf{e}_i'^\top \bmod \mathsf{q}.$$

Then:

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

We can then decrypt to get

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

This is without knowing any other ciphertexts, i.e. goes beyond inherent leakage.

A Problem with the Proof of LaPS: Removing QEnc

Shi et al. (p. 15):

Proof of Theorem 1: First, we will make a small modification to the aggregator oblivious security game. In the **Encrypt** queries, if the adversary submits a request for some tuple (i, x, t^*) where t^* is the time step specified in the **Challenge** phase, the challenger treats this as a **Compromise** query, and simply returns the sk_i to the adversary. Given sk_i, the adversary can compute the requested ciphertext herself. Therefore, this modification actually gives more power to the adversary. From now on, we will assume that the adversary does not make any **Encrypt** queries for the time t^*.

LaPS (Becker et al., p. 16):

We define the following intermediate game Game similar to [40] that is indistinguishable from the aggregator obliviousness security game according to Definition 6. First, we treat any **Encrypt** query as a **Compromise** query from the adversary. Clearly, this turns the adversary actually more powerful. Secondly, we change the **Challenge** phase to its real-

Problem Intuition: Corruptions lead to stricter conditions on what is allowed: corrupted users cannot be included in $\mathcal{U}$, but those from which we queried an encryption (for $\ell \neq \ell^*$ if encrypt-once) can be. Hence, LaPS is shown only to be "corruption-only-AO-secure".

How Weak is corruption-only-AO Security?

The following "trivial" PSA scheme is (perfectly) corruption-only-AO secure:[1]

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

Intuition: The adversary cannot get anyone to use their OTP key twice ($\mathcal{U} \cap \mathcal{CS} = \emptyset$)

Problem: In practice, a PSA setup is supposed to be used for more than one use per party. Hence, security guarantees are very weak.

Also: The scheme is not AO-secure.

[1] This was already mentioned in a footnote in Waldner et al. (2021)

How Weak is corruption-only-AO Security?

The following "trivial" PSA scheme is (perfectly) corruption-only-AO secure:[1]

$\underline{\mathsf{Setup}(1^\lambda, 1^n):}$	$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{k_i}, \mathsf{x_i}, \ell):}$
for $\mathsf{i} \in [\mathsf{n}]$: $\mathsf{k_i} \leftarrow \mathbb{Z}_\mathsf{q}$	**return** $\mathsf{x_i} + \mathsf{k_i}$
$\mathsf{k}_0 := -\sum_{\mathsf{i}\in[\mathsf{n}]} \mathsf{k_i}$	$\underline{\mathsf{AggrDec}(\mathsf{pp}, \mathsf{k}_0, \ell, (\mathsf{ct_i})_{\mathsf{i}\in[\mathsf{n}]}):}$
$\mathsf{pp} := \mathsf{q}$ (the modulus)	**return** $\sum_{\mathsf{i}\in[\mathsf{n}]} \mathsf{ct_i} + \mathsf{k}_0$
return $(\mathsf{pp}, (\mathsf{k_i})_{\mathsf{i}\in[\mathsf{n}]_0})$	

Intuition: The adversary cannot get anyone to use their OTP key twice ($\mathcal{U} \cap \mathcal{CS} = \emptyset$)

Problem: In practice, a PSA setup is supposed to be used for more than one use per party. Hence, security guarantees are very weak.

Also: The scheme is not AO-secure.

Intuition: Using $\mathsf{QEnc}(\mathsf{i}^*, \mathsf{x} = 0, \ell)$ you get $\mathsf{k_{i^*}} = \mathsf{k_{i^*}} + 0$ without corrupting.

[1] This was already mentioned in a footnote in Waldner et al. (2021)

Key Recovery Attack on LaPS using Averaging

- $(\mathsf{Gen_{ah}}, \mathsf{Enc_{ah}}, \mathsf{Dec_{ah}})$: additively homomorphic PKE with pseudorandom ciphertexts.
- $\mathbf{G}$: gadget matrix, i.e. $\mathbf{G} = \mathbf{I} \otimes (1\ 2\ \ldots\ 2^{d-1})$, with $d = O(\lceil \log q \rceil)$
- $\mathbf{e} \leftarrow \mathcal{D}_{\Lambda^{\perp}_{\mathbf{v}}(\mathbf{G}),\sigma}$: error distribution (with $\sigma > 2\sqrt{\lambda}$); defined so that $\mathbf{G} \cdot \mathbf{e} \bmod q = \mathbf{v}$

Note: Each $\mathsf{Enc}(pp, k_i, x_i)$ uses same $\mathbf{A}$ and k_i.
Attack: Run QEnc many (poly(λ)) times to average out the error.

$\underline{\mathsf{Setup}(1^\lambda, 1^n)\text{:}}$
$\mathbf{A} \leftarrow \mathbb{Z}_q^{\lambda \times d}$
`for` $i \in [n]$: $k_i \leftarrow \mathbb{Z}_q^\lambda$
$(pk_{ah}, sk_{ah}) \leftarrow \mathsf{Gen_{ah}}(1^\lambda)$
$pp := (\mathbf{A}, pk_{ah})$
$k_0 := (sk_{ah}, \sum_{i\in[n]} k_i)$
`return` $(pp, (k_i)_{i\in[n]_0})$

$\underline{\mathsf{Enc}(pp, k_i, x_i)\text{:}}$
`parse` $pp = (\mathbf{A}, pk_{ah})$
$\mathbf{v}_i \leftarrow \mathsf{Enc_{ah}}(pk_{ah}, x_i)$
$\mathbf{e}_i \leftarrow \mathcal{D}_{\Lambda^{\perp}_{\mathbf{v}_i}(\mathbf{G}),\sigma}$
$ct_i = k_i^\top \mathbf{A} + \mathbf{e}_i^\top \bmod q$
`return` ct_i

$\underline{\mathsf{AggrDec}(pp, k_0, (ct_i)_{i\in[n]})\text{:}}$
`parse` $k_0 = (sk_{ah}, k)$
$\mathbf{e} = \sum_{i\in[n]} \mathbf{c}_i - k^\top \mathbf{A}$
`return` $\mathsf{Dec_{ah}}(sk_{ah}, \mathbf{G} \cdot \mathbf{e} \bmod q)$

Another Proof Problem: Changing to Real-or-Random

The proof switches to a **Real-or-Random** (RoR) variant of the AO-game without really defining it. And, doing it naively leads to a potentially **weaker definition**.

Main Issue: How should the random x_i be chosen? In AO, the attacker can determine adaptively whether the balance condition needs to hold, here this is not possible.

Way out: Define RoR-AO by giving a choice b_{sum} in QChallenge on whether the randomness should add up to the same value as the provided x_i's. We show this to be equivalent to AO.

LaPS (Becker et al., p. 16):

> We define the following intermediate game Game similar to [40] that is indistinguishable from the aggregator obliviousness security game according to Definition 6. First, we treat any **Encrypt** query as a **Compromise** query from the adversary. Clearly, this turns the adversary actually more powerful. Secondly, we change the **Challenge** phase to its real-or-random version, i.e. instead of having the adversary 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 $\mathbb{Z}_q^\lambda$. It is straightforward that any adversary with more than negligible advantage in winning Game will also win the aggregator obliviousness security game with more than negligible advantage. Therefore, it suffices to show that

Summary and Discussion

Summary:

1. The LaPS scheme (NDSS 2018) was the first with claimed post-quantum and encrypt-multiple times security, and subsequently patented.
2. However, the proof contains at least two shaky/wrong steps
3. There are attacks that let you recover differences of their messages and the user's keys.
4. **Importantly**: First mention of problems was in a remark in Waldner et al. (ePrint 2021) and the proof mistakes have also spread to other papers.

Main Takeaway/Discussion:

1. Getting all the details of a security proof right is hard, but crucial.
2. Maybe: avoid putting the main proof in the appendix.

References

- Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn 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.
- Rachid El Bansarkhani, Özgür Dagdelen, and Johannes Buchmann. “Augmented learning with errors: The untapped potential of the error term”. In: FC 2015. Springer. 2015, pp. 333–352. DOI: 10.1007/978-3-662-47854-7_20.
- Daniela Becker, Jorge Guajardo, and Karl-Heinz 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.
- Hendrik Waldner, Tilen Marc, Miha Stopar, and Michel Abdalla. Private Stream Aggregation from Labeled Secret Sharing Schemes. Cryptology ePrint Archive, Report 2021/081.