International Journal of Information Security manuscript No.
(will be inserted by the editor)

SLAP: Simple Lattice-Based Private Stream Aggregation Protocol

Jonathan Takeshita [1], **Ryan Karl** [1], **Ting Gong** [2], **Taeho Jung** [1,a]

[1]Department of Computer Science
[2] Department of Mathematics
University of Notre Dame
Notre Dame, Indiana, USA

Received: date / Accepted: date

Abstract Private Stream Aggregation (PSA) protocols allow for the secure aggregation of time-series data, affording security and privacy to users' private data, with significantly better efficiency than general secure computation such as homomorphic encryption, multiparty computation, and secure hardware based approaches. Earlier PSA protocols face limitations including needless complexity, a lack of post-quantum security, or other practical issues. In this work, we present SLAP, a **S**imple **L**attice-based Private Stream **A**ggregation **P**rotocol. SLAP features two variants with post-quantum security, with simpler and more efficient computations enabled by (1) our white-box approach that builds the encryption directly from the Ring Learning With Errors assumption and (2) application of state-of-the-art algorithmic optimizations in lattice-based cryptography. We prove that SLAP with differentially private inputs is an aggregator oblivious PSA scheme. We implement SLAP, and show experimentally the improvements of SLAP over similar work. We show a speedup of 20.76x over the previous state-of-the-art RLWE-based PSA work's aggregation, and apply techniques including RNS, NTT, and batching to obtain a throughput of 390,691 aggregations per second for 1000 users. The communication overhead of SLAP is less than in previous work, with decreases of up to 99.96% in ciphertext sizes as compared to previous work in RLWE-based PSA. We also show the improvement of SLAP over other state-of-the-art post-quantum PSA with regards to throughput, and compare and contrast our RLWE-based approach with other work based upon secret sharing and Learning-With-Rounding.

Keywords Public key cryptosystems · Lattice-based Cryptography, · Private Stream Aggregation.

[a]e-mail: tjung@nd.edu

1 Introduction

1.1 Motivations

Many real-life applications can be posed as a problem of aggregation. For example, to compute an average over a set of data, the data can be additively aggregated (i.e. summed), and then divided by the number of data elements. In some such contexts where statistical analysis is essential, the privacy of individual data holders is paramount. Examples of this include health care, education, and advertising, where individuals' privacy is often protected by law [7]. Due to such demands, secure aggregation of large-scale user-generated datasets has gained interest in industry, where such datasets need to be analyzed. Facebook Research and Sony Research have called for proposals in secure aggregation [32, 28], and Google has adopted secure aggregation in their secure federated learning framework [44]. Smart metering is another application of aggregation in related work [23, 24]. It is thus desirable to consider schemes that allow data aggregation to be efficiently computed without risking users' security against external eavesdroppers or privacy against other participants.

One possible application of secure and private aggregation is in aggregating data from hospitals, whose patients' privacy is protected by laws such as HIPAA. Hospitals may want to jointly compute the total of sensitive figures such as patient deaths or malpractice incidents in a locality, but also want to conceal their individual figures, both from external attackers (security) or the authority computing the total (privacy). Similarly, homeowners using smart metering may wish to not disclose the exact details of their power usage, as it may leak information about their personal habits or travel patterns. Online advertising is another use

case; users' web browsers seeing an advertisement can privately send whether or how quickly a user clicked on the ad. These are just a few possible applications of secure and private aggregation.

Simply put, the secure data aggregation problem is as follows: considering n users with data x_i, how can we allow a third party to compute $\sum_{i=0}^{n-1} x_i$ while preserving users' security and privacy? Semantic security against eavesdroppers is easy enough to achieve with symmetric encryption, but this does not prevent the aggregator from learning individual users' data. Privacy guarantees are thus required to make sure the aggregator cannot learn anything about individual data. This problem is a general one that is applicable to privacy-preserving statistical calculation (e.g., count, mean, standard deviation), histogram calculation, federated deep learning [58], smart metering [17], and can be seen in areas including healthcare, education, and finance that require computation over users' private data. In the modern era, the problem also implicitly includes efficiency of both user-side and server-side operations, as well as minimizing communication overhead. Considering practical efficiency and throughput is especially important at the scale that Big Tech operates in - Facebook and Google serve billions of users and their queries daily that they might with to perform statistical analysis upon.

A simple solution to this problem is to use a trusted third party to collect and aggregate users' data as in Figure 1; however, the existence of a fully trustworthy third party is a strong assumption that we wish to eliminate. The shortcomings of more general approaches to this problem have led to the creation of the idea of Private Stream Aggregation (PSA), as shown in Figure 2. As originally introduced by Shi et al. [51], PSA encompasses security and privacy in the idea of *aggregator obliviousness*. Aggregator obliviousness requires that no adversary can learn any new information about a target's data, even when compromising all other participating parties. PSA is an area of active research [9, 51, 33, 34, 36, 24, 55] due to its wide applicability to secure computation.

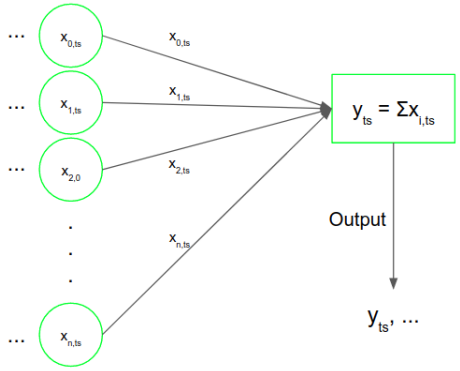

Fig. 1 Time-series Plain Aggregation

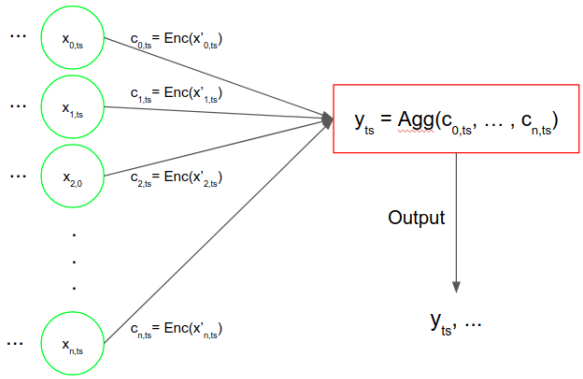

Fig. 2 Time-series Private Stream Aggregation with Noisy Data

1.2 Other Methods

Other approaches include the use of (fully) homomorphic encryption, secure multiparty computation, or trusted hardware to allow a third party to collect and aggregate data without gaining any knowledge about the users' data besides the final aggregation. While these approaches can achieve a high degree of security, they incur a high overhead in computation and/or communication, and may have other issues rendering them undesirable for ordinary users. Homomorphic Encryption (HE) can be applied to compute over encrypted data, and can guarantee quantum security to encrypted data [43, 13, 26]. However, there are many issues with using existing HE schemes, including key management, computational overhead, and complexity. Trusted Execution Environments, including Intel SGX, AMD Secure Execution Environment, and TrustZone, can be used to facilitate secure computing. However, these hardware solutions are vulnerable to practical attacks, incur runtime penalties, and introduce nontrivial implementation difficulties [45]. Secure Multiparty Computation (MPC) protocols are used to allow a set of parties to securely compute a function over their inputs [59, 37]. These functionalities may require multiple rounds of interactive communication between parties, where for a single piece of input data, multiple messages must be exchanged between parties.

Differential Privacy aims to provide dataset-level privacy by adding a small amount of noise to each element, so that the computation of a function over the dataset doesn't reveal whether or not a particular element is included in the dataset [9, 21, 51]. This can give users some privacy, obscuring information about

their individual data that might otherwise be gleaned from the final result. The normal model of Differential Privacy assumes a trusted aggregator who aggregates user input, then adds differentially private noise to mask the exact result. PSA eliminates this assumption of a trusted aggregator, replacing it with one of an untrusted aggregator who receives inputs with value-obscuring noise already added.

1.3 Private Stream Aggregation

Later work in secure aggregation worked towards constructing aggregation schemes secure against quantum adversaries [1, 3, 49]. The state-of-the-art Ring Learning With Errors (RLWE)-based LaPS scheme [9] achieves additive aggregation with quantum security. However, the LaPS scheme introduces needless complexity and overhead in its operations, and its basic version only guarantees security for a single round of aggregation. It also relies on the BGV cryptosystem [13] as a black-box primitive, which hinders the possibility of optimizing the underlying system for the task of aggregation. Other quantum-secure PSA not based on RLWE has been proposed [55, 24], but faces issues of security [18] or practicality with many users due to key size growth. There exists other highly efficient recent work in quantum-secure PSA [55, 24]. Though these works do not use RLWE as their cryptographic assumption, they provide more state-of-the-art benchmarks to compare our work against.

In this work, we improve upon previous constructions in secure aggregation by presenting simpler and more efficient lattice-based quantum-secure aggregation schemes. Instead of relying on HE schemes such as BGV directly as a primitive, we create novel schemes that arise directly from the LPR/NTRU assumptions and their underlying principles of lattice-based hardness assumptions [42, 31]. To create these schemes, we take a white-box approach and design RLWE-based PSA protocols, without using other HE cryptosystems as primitives. Further, our schemes allow for security through multiple aggregations without key redistribution, fixing weaknesses in previous work [9].

Our scheme uses one-time key generation to provide security without key redistribution across multiple aggregations. This simple construction helps create a PSA scheme that is comparable to the state-of-the-art even with a naive implementation, and is capable of greatly improving upon the state-of-the-art with various optimizations. Our schemes satisfy the PSA security definition of aggregator obliviousness, and apply differential privacy similarly to prior work. We show that our optimized implementation can improve upon previous work in RLWE-based PSA [9] by 65x for encryption and 20x for aggregation, and can achieve a throughput of over 390,691 aggregations per second for 1000 users. We also show that our schemes are scalable, and more efficient in communication overhead than previous work, with very large improvements (over 99%) in ciphertext size. We also compare our work to other state-of-the-art post-quantum PSA protocols [24, 55], and show that our work is comparable in latency and shows a great improvement in practical throughput over these schemes.

Our Contributions

- We present SLAP, a novel lattice-based and quantum-secure PSA scheme with two versions, built using a white-box approach directly from the RLWE problem. By taking this approach, we can construct lattice-based schemes optimally tailored to the application of PSA. Both versions are simpler and more efficient than previous related work, and only need a single trusted setup to allow for secure aggregations across multiple timestamps. SLAP is particularly efficient with regards to communication, with ciphertext size smaller by up to 99.96% as compared to other RLWE-based PSA. We show that SLAP is an aggregator oblivious PSA scheme, and can attain differential privacy.
- We further integrate the Number-Theoretic Transform [41], Residue Number System [30], and double-batching to greatly improve the practical performance and throughput of SLAP. To our knowledge, SLAP is the first work to fully explore and implement these optimizations in PSA.
- We implement simple and optimized versions of both variants of SLAP, and analyze the results to show the practical efficiency and scalability of SLAP. Our results show performance improvements of an order of magnitude against previous work, which become much greater when integrating our double-batching strategy. We can achieve a latency as low as 3.26ms for a single aggregation (with throughput of up to 390,691 aggregations per second with 1000 users, and only one order of magnitude slower than plain aggregation) with the *largest* set of parameters evaluated by LaPS. This is the first work in quantum-secure to consider the practical issues of throughput and communication overhead in depth. SLAP is shown to be comparable in terms of operation latency with the leading state-of-the-art post-quantum PSA schemes [9, 55, 24], and can greatly outperform preexisting post-quantum PSA schemes in throughput due to its batching. We are also able to conclude from our experiments the relative superiority of noise-scaled message encoding against message-scaled encoding for the use of additive PSA.

– Both implementations (basic and fully optimized) of our scheme are made available (anonymously) as open-source code. We provide a software library that can be used in future research in PSA and post-quantum cryptography.

Organization In Section 2, we discuss related work in PSA and differential privacy. In Section 3, we give relevant background on the RLWE problem and differential privacy, as well as the scenario, security notion, and orthogonal issues of PSA. In Section 4 we define the basic operations of SLAP, and discuss its security, novelty, and benefits. In Section 5 we introduce differential privacy and prove aggregator obliviousness for SLAP. Section 6 discusses optimizations and practical issues for SLAP. Section 7 contains our experimental evaluation of SLAP, and Section 8 summarizes our work and contributions.

2 Related Work

In this section, we review related work in PSA. Both pre-quantum and post-quantum PSA works are included, to show the progression of research. A brief overview of Differential Privacy is also included here and in Section 3, as it is a technique frequently used to obscure user inputs to provide privacy. Like other work in PSA [51, 9, 24, 33], we focus primarily on novel and efficient constructions, and are not primarily concerned with robustness. Nevertheless, we include works on reliability in PSA for reference to this orthogonal area.

2.1 Pre-quantum PSA

The seminal works of Shi et al. [51] and Rastogi et al. [48] introduced the concepts of PSA and aggregator obliviousness, presenting schemes based upon Diffie-Hellman and Fourier perturbation with differential privacy. The work of Joye et al. [33] improved limits on plaintext spaces, using the hardness assumption of Decisional Composite Residuosity. Erkin et al. [23] propose a PSA-like framework for use in smart metering using the Pallier cryptosystem. Protocols for secure distributed polynomial computations based on the discrete logarithm problem have also been presented [34, 35], which have the added advantage of not relying upon secure channels of communication. These works are vulnerable to quantum-capable attackers, who can utilize Shor's algorithm for solving integer factorization and the discrete logarithm problem [52].

2.2 Post-quantum PSA

There exists some work in PSA-like protocols relying on lattice-based cryptography, which is secure against quantum-capable adversaries [1, 3, 49]. However, these works have the disadvantage of heavily relying on a trusted third party, and are less versatile, being designed for the specific scenario of smart metering. Key-homomorphic pseudorandom functions can be used to construct quantum-secure PSA protocols such as the leading RLWE-based LaPS protocol without the requirement of a trusted third party (beyond initial setup) [9, 54]. However, such protocols have extremely complex decryption procedures or other weaknesses. In particular, LaPS has a "double encryption", where homomorphically encrypted user input is again concealed within an Augmented learning-with-errors term.

The work of Ernst et al. [24] uses approximately key-homomorphic PRFs and the learning-with-rounding (LWR) problem to construct quantum-secure PSA. This scheme is relatively simple and features small ciphertexts with large plaintext spaces. The security of the LWR and Ring-LWR problems is still of concern due to the deterministic nature of the rounding [18].

PSA schemes using secret sharing [17, 10, 55] are not weak to quantum adversaries. The LaSS protocol of Waldner et al. [55] is of particular note, as it is most recent and highly efficient. All of these protocols have requirements including multiple rounds of communication, multiple trusted parties, and key sizes quadratic in the number of users that hinder their scalability and practical applicability. However, LaSS is highly effective for the scenario of lower numbers of users under a stricter security model than that of LaPS.

Multi-Key Fully Homomorphic Encryption [6, 46] can be used to implement PSA, along with many other types of multi-user computations; such schemes are more general and much more complex that what is needed for PSA. Especially for weaker devices such as IoT nodes, the extra overhead and storage may not be desirable. In particular, the use of multiple encryptions [6] or constructing matrices of ciphertexts [46] would require large amounts of memory, storage, communication, and computation, making these general schemes less practical than purpose-built PSA schemes for applications involving aggregation.

LaPS [9], LaSS [55], and Ernst et al. [24] are the works that we primarily compare and contrast our work to. These works each use a different strategy to provide quantum security, with LaPS' use of RLWE-based homomorphic encryption being the closest to our direct use of RLWE. All of these works provide some experimental evaluation, with that of LaPS being the most extensive,

allowing us to make experimental comparisons of the protocols.

2.3 Reliability and Fault-Tolerance in PSA

Related research in PSA does not always explore practical issues around robustness such as fault tolerance, reliability, input poisoning, or fully malicious participants who might craft nonsensical messages or even refuse to communicate. Often, PSA focuses on the theoretical construction of protocols more than practical concerns [51, 9, 55]. Other lines of research do provide solutions for fault tolerance, dynamic join/leave, maliciously chosen inputs, and error detection [56, 57, 38, 47, 39].

2.4 Differential Privacy

Differential Privacy refers to the technique of adding specially-distributed noise to data to mask the exact values [20, 48, 54]. It is commonly used in work in aggregation to hide the exact result of a computation [9, 51, 39, 2]. PSA schemes have users add differentially private noise user-side, so that users do not have to trust the aggregator to add noise after aggregation and the final noise added in the aggregate result is bounded above by a constant (with overwhelming probability).

3 Background

3.1 Notation

We summarize common notation in SLAP in Table 1.

For a number x, let $\lfloor x \rceil$ be the integer closest to x (rounding up if the fractional portion of x is $\frac{1}{2}$). Let $[x]_t$ be the centered modular reduction of $x \mod t$, such that $[x]_t = x - \lfloor \frac{x}{t} \rceil \cdot t \in \mathbb{Z}_t$, where $\mathbb{Z}_t = [\frac{-t}{2}, \frac{t}{2}) \cap \mathbb{Z}$, and similarly $R_q = \mathbb{Z}_q[X]/\Phi(X)$ for $\mathbb{Z}_q = [\frac{-q}{2}, \frac{q}{2})$. We use R to denote the quotient ring of $\mathbb{Z}[X]/\Phi(X)$. Here, $\Phi(X)$ is the $M = 2N$-th cyclotomic polynomial with degree $N = 2^d$ for some positive integer d. Define $R_t = \mathbb{Z}_t[X]/\Phi(X)$, the ring with all coefficients in $\mathbb{Z}_t$. Boldface lowercase letters (e.g. $\mathbf{a}$) denote ring elements. Centered modular reduction can be applied coefficientwise to ring elements, i.e. $[\mathbf{a}]_t \in R_t$.

3.2 Ring Learning With Error (RLWE) Problem

Let $\mathbf{s}$ be a random element of R_q drawn from a distribution χ. The distribution χ may be bounded by a small number, or it can be uniformly random over R_q, with no effect on the security of RLWE [26, 42]. Let $\mathbf{a}_i, \mathbf{e}_i$ be a polynomially bounded selection of elements of R_q, with $\mathbf{a}_i$ chosen uniformly at random[1] and $\mathbf{e}_i$ chosen randomly from a distribution ζ on R_q. Here, ζ may be the same as χ, or it may also be bounded by a small number. In practice, 1-bounded distributions are often used for χ and ζ. An adversary is given the set of pairs $(\mathbf{a}_i, \mathbf{b}_i) \in R_q^2$. Unknown to the adversary is whether the values $\mathbf{b}_i$ are RLWE terms, i.e., $\mathbf{b}_i = [\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i]_q$, or if $\mathbf{b}_i$ was randomly chosen from R_q. The decisional RLWE problem is then to determine whether the terms $\mathbf{b}_i$ are RLWE terms or random elements of R_q.

The RLWE problem is believed to be intractable for quantum computers [42]. RLWE terms of the form $\mathbf{b}_i = [\mathbf{a}_i \cdot \mathbf{s} + t \cdot \mathbf{e}_i]_q$ with t, q coprime are also pseudorandom [11, 12, 50], and mathematically convenient for some schemes.

3.3 Differential Privacy and Computational Differential Privacy

We recall relevant definitions from differential privacy [54, 51, 48, 9] in this section. Let $exp(\cdot)$ be the exponential function. Denote the l_1 norm of a database D as $||D||_1 = \sum_{i=1}^{|D|} |D_i|$.

Definition 3.1 The l_1 distance between databases D_0, D_1 is $||D_0 - D_1||_1$. We say that D_0, D_1 are adjacent when the distance between the databases is no more than 1.

Definition 3.2 A function M is (ϵ, δ)-differentially private if for all adjacent databases D_0, D_1 and all U that are subsets of the range of M, $Pr[M(D_0) \in U] \leq exp(\epsilon) \cdot Pr[M(D_1) \in U] + \delta$. (This is abbreviated to M being ϵ-differentially private when δ is zero.)

Definition 3.3 A function M is (α, β)-accurate with respect to a query function f (with the same domain and range as M) when for all D in the domain of M, $Pr[|M(D) - f(D)| \leq \alpha] \geq 1 - \beta$.

Definition 3.4 The discrete Laplacian distribution is defined for a scale parameter $s > 1$. Let $\sigma = exp(-1/s) \in (0, 1)$. Then the discrete Laplacian distribution DL_s is definied to be the function with a probability mass function of $DL_s(x) = \frac{1-\sigma}{1+\sigma} \cdot \sigma^{|x|}$, for arguments $x \in \mathbb{Z}$.

3.4 Private Stream Aggregation and Aggregator Obliviousness

We consider semi-honest (honest-but-curious) parties for PSA and aggregator obliviousness. Each user i possesses

[1] While an element $\mathbf{a}_i$ could be zero, this happens with only negligible probability.

Table 1 Common Important Notation

Notation	Meaning
λ	Security parameter, measured in bits.
R_q	The polynomial ring that ciphertexts and other terms reside in.
q	Ciphertext modulus, an integer that is commonly hundreds of bits.
N	The polynomial modulus degree, a power of two commonly ranging from 2^{10} to 2^{15}.
t	The plaintext modulus, an integer much less than q. Commonly ranges from 16 to 192.
n	Number of users in the PSA instantiation.
χ, ζ	Small error distributions on R_q.
ts	The timestamp at which a particular aggregation takes place.
$x_{i,ts}, \mathbf{x}_{i,ts}$	User i's PSA input at timestamp ts.
$r_{i,ts}, \mathbf{r}_{i,ts}$	User i's PSA noise at timestamp ts.
$\mathbf{e}_{i,ts}$	User i's PSA error term at timestamp ts.
$c_{i,ts}, \mathbf{c}_{i,ts}$	User i's ciphertext at timestamp ts.
$s_i, \mathbf{s}_i$	User i's PSA key.
$s', \mathbf{s}'$	The aggregator's key.
H	A random-oracle hash function mapping from timestamps to R_q.
$\mathbf{A}_{ts} = H(ts)$	The element of R_q derived from H and ts, used as a public term in the aggregation.
$parms$	The public PSA parameters R_q, t, n, H.
Δ	Equal to $\lfloor \frac{q}{t} \rfloor$, used in $SLAP_{MS}$ encryption and decryption.
$y_{ts}, \mathbf{y}_{ts}$	The final aggregation result, equal to the sum of users' inputs and noise.
β	A parameter controlling the probability that differentially private noise drawn from a Discrete Laplacian distribution will be zero.
ϵ, δ	Differential Privacy parameters related to privacy.
α, β	Differential Privacy parameters related to accuracy.
s, σ	Parameters of the Discrete Laplacian distribution used to draw differentially private noise.
γ	Proportion of users honestly adding differentially private noise.
w	Maximum difference in user inputs when applying Differential Privacy.
k	Number of RNS moduli used for the ciphertext.
$q_0 \cdots q_{k-1}$	Coprime ciphertext moduli used in RNS, whose product is q. For use in the NTT, these should each be equivalent to 1 modulo $2N$.
$q_i^*, \tilde{q}_i, \omega_i, \theta_i$	Precomputed parameters used in RNS base conversion and division-with-rounding.
v, z_i, y_i, ω, v	Penultimate and final results in RNS base conversion and division-with-rounding.
k'	Number of RNS moduli used for the plaintext.

a piece of data x_i, corresponding to some timestamp ts. The users wish to calculate the aggregation $\sum_{i=0}^{n-1} x_i$. (Other types of aggregation such as multiplicative aggregation are possible; in this work we only present additive aggregation.)

Private Stream Aggregation schemes allow a third party (the aggregator) to perform this aggregation while providing users privacy for their time-series data sent to the aggregator. A PSA scheme should provide privacy to individual users, preventing the aggregator from learning their individual data even when compromising or colluding with other users. PSA was first introduced by Shi et al. [51] as the combination of a scheme for aggregation secure against outside adversaries and the use of privacy-preserving noise to obscure user inputs. PSA schemes are formalized as the following 3 algorithms:

- $Setup(\lambda, \cdots)$: Takes a security parameter λ as input, along with any other required parameters, e.g., the number of users n, or the range of their data. Returns a set of parameters $parms$, users' secret keys $s_i, i \in [0, n-1]$, and the aggregation key s'.
- $NoisyEnc(parms, x_{i,ts}, r_{i,ts}, s_i, ts, \cdots)$: Takes the scheme's parameters, and a user's secret key s_i, time-series input $x_{i,ts}$, and noise $r_{i,ts}$, along with a timestamp ts and any other required parameters. Returns an encryption $c_{i,ts}$ of the user's noisy input under their secret key at timestamp ts. Users should only run this function once for any given timestamp.
- $Agg(parms, s', ts, c_{0,ts}, \cdots c_{n-1,ts})$: Takes the scheme's parameters, the aggregation key, a timestamp, and the n time-series ciphertexts $c_{i,ts} = NoisyEnc(parms, x_{i,ts}, r_{i,ts}, s_i, ts, \cdots)$ from the users (with timestamp ts). Returns the noisy sum $y_{ts} = \sum_{i=0}^{n-1} x_{i,ts} + r_{i,ts}$.

Users run $NoisyEnc$ on their data (once per timestamp), and send ciphertexts $c_{i,ts}$ to the aggregator. The aggregator calls Agg on the ciphertexts $c_{0,ts}, \cdots c_{n-1,ts}$ to get the noisy aggregation result y_{ts}, which is desired to be approximately equal to $\sum_{i=0}^{n-1} x_{i,ts}$. In PSA schemes, the algorithm $Setup$ is run only once and in a trusted manner [9, 51, 33]. This can be accomplished through the use of an additional trusted third party, secure hardware, or secure multiparty computation. Because it is a one-time process, it has negligible influence to the scalability of the entire protocol.

In PSA, because only a single round of input-dependent communication must be performed, PSA can be more efficient in communication than most existing MPC protocols [54, 37]. This is an advantage of PSA with semi-honest parties; schemes with robustness and security against malicious and colluding adversaries commonly require more communication.

Aggregator Obliviousness Informally, we require that an adversary able to compromise any number of the aggregator and other users is unable to learn any additionally revealed information about uncompromised users' data from the protocol. The idea of *aggregator*

obliviousness encompasses these ideas. Encrypt-once security places the additional restriction that an attacker can only gain access to a single encryption for a particular user and timestamp. We restate the definition of aggregator obliviousness with encrypt-once security in Definition 3.5 [9, 51]:

Definition 3.5 Suppose we have a set of n users, who wish to compute an aggregation at a time point specified by the timestamp ts. An aggregation scheme AS is aggregator oblivious with encrypt-once security [9, 51] if no polynomially bounded adversary has an advantage greater than negligible in the security parameter λ in winning the following game:

The challenger runs the Setup algorithm and returns the public parameters $parms$ to the adversary. Then the adversary will guess which of two unknown inputs was a users' data, by performing the following queries:

Encrypt: The adversary sends the values $(i, x_{i,ts}, r_{i,ts}, ts)$ to the challenger and receives back $NoisyEnc(parms, x_{i,ts}, r_{i,ts}, s_i, ts)$ to the adversary. For a given user, only one query at a particular timestamp ts can be made.

Compromise: The adversary sends $i \in [0, n) \cup \{\square\}$ to the challenger. If $i = \square$, the challenger gives the aggregator's decryption key s' to the adversary. Otherwise, the challenger returns the i^{th} user's secret key s_i to the adversary.

Challenge: The adversary may only make this query once. The adversary sends a set of participants $S \subset [0, n)$ to the challenger, with $i \in S$ not previously compromised. For each user $i \in S$, the adversary chooses two plaintext-noise pairs at a new timestamp ts not previously used in any Encrypt query as $(x_{i,ts}, r_{i,ts}), (\tilde{x}_{i,ts}, \tilde{r}_{i,ts})$ and sends them to the challenger. The challenger then chooses a random bit b. If $b = 0$, the challenger computes $c_{i,ts} = NoisyEnc(parms,\ x_{i,ts}, r_{i,ts}, s_i, ts)$ for every $i \in S$. If $b = 1$, the challenger computes $c_{i,ts} = NoisyEnc(parms,\ \tilde{x}_{i,ts}, \tilde{r}_{i,ts}, s_i, ts,)$ for every $i \in S$. The challenger returns the ciphertexts $\{c_{i,ts}\}_{i \in S}$ to the adversary.

We say that the adversary wins the game if (1) they can correctly guess the bit b chosen during the Challenge and (2) if the aggregator is compromised, then $\sum_{i \in S} x_{i,ts} + r_{i,ts} = \sum_{i \in S} \tilde{x}_{i,ts} + \tilde{r}_{i,ts}$.

In this game, an adversary can trivially learn information about a single user's data, by compromising the aggregator and all other users, decrypting the compromised users' data, and computing the difference of the aggregator's sum and the sum of the compromised users' data. These vulnerabilities are inherent in scenarios with powerful adversaries; thus aggregator obliviousness requires that no *additional* information is learned in such cases of unavoidable leakage, as discussed at length in [51].

Other Issues in PSA PSA constructions, including this work, are primarily concerned with efficient and functional constructions of the actual aggregation protocol [9, 51, 10, 17]. Differential Privacy techniques are used in PSA, and can be used with many different PSA protocols. Many practical issues such as fault tolerance, data pollution, robustness, readiness of input, and fully malicious participants are not considered in PSA constructions, though some lines of research do explore these properties [38, 39, 47, 56, 57]. In particular, many PSA schemes assume that for n users, at each epoch every user will have exactly one input in the specified range prepared for aggregation, and that no user will fail to participate.

4 Basic Aggregation

We first present our PSA schemes without specifying the details of differentially private noise, and discuss the use of differentially private noise in Section 5. We present two versions of our PSA scheme SLAP, whose differences are in which portion of a ciphertext is scaled and by what term. This leads to slight differences in encryption and decryption, though the actual additive aggregation in decryption (simply summing all ciphertexts) is the same. Both variants have identical key correlation requirements. These variants are evaluated against each other in Section 7.

While lattice-based fully homomorphic encryption (FHE) has been used in previous lattice-based PSA as a black-box building block [9], we instead take a white-box approach, constructing purpose-built lattice-based cryptographic procedures specially formulated for PSA. Our new schemes are key-homomorphic adaptations of the ideas underlying the LPR and NTRU cryptosystems [42, 31], adapted to the specific application of PSA. This allows for a more efficient solution for the specific purpose of PSA. One example of this is that ciphertexts in SLAP are only a single polynomial from R_q, while ciphertexts in RLWE FHE schemes are tuples of elements of R_q [13, 26]. This property allows for less communication overhead and more efficient encryption and decryption.

LaPS mentions the possibility of generating a new public key for each timestamp, but did not go into details and did not make it an explicit requirement of their scheme [9]. However, this should be done for security across multiple executions - without a fresh public matrix for each aggregation, an aggregator can learn the difference between a users' inputs by subtracting LaPS

ciphertexts and decrypting the result. This attack is described in detail in Remark 5.3 of [55]. SLAP thus uses a random oracle to prevent this attack [25]. In practice, this can be instantiated by defining a hash function for users to utilize.

4.1 First Additive Scheme (Noise-Scaled)

The first variant of our scheme we present is noise-scaled, meaning that in encryption, the small amount of random noise added is scaled by the plaintext modulus t. A modular reduction by t in decryption thus removes the noise terms, leaving only the message. This idea of scaling up the noise to allow for decryption (so that noise is removed with a modular reduction) is used in the NTRU [31] and BGV [13] lattice-based cryptosystems. In this scheme $SLAP_{NS}$, we consider the plaintext domain to be the ring R_t and the ciphertext domain to be the ring R_q, with $q \gg t$, q and t coprime, and an appropriate value of the polynomial modulus degree N to allow for a desired level of security. The scheme $SLAP_{NS}$ is defined as follows:

- $Setup_{NS}(\lambda, t, n)$: Takes in the security parameter λ, the plaintext modulus t, and the number of users n. Choose q such that $log_2(3) + log_2(n) + log_2(t) < log_2(q)$ (1) and q, t are coprime. (This condition on n, t, q ensures that the bound of Inequality 2 holds for correctness of decryption.) Choose the polynomial modulus N such that λ bits of security are provided for the RLWE problem with ring polynomial coefficients in $\mathbb{Z}_q$ [4, 5]. (While a larger value of q allows for more utility, it also decreases security - choosing larger values for N can offset this.) Choose users' secret keys $\mathbf{s}_0 \cdots \mathbf{s}_{n-1}$ from χ. Construct the aggregator's key as $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Choose a random oracle hash function H, mapping from the domain of all timestamps to the range of R_q [25]. Return $parms = (R_q, t, n, H)$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $NoisyEnc_{NS}(parms, \mathbf{x}_{i,ts} \in R_t, \mathbf{r}_{i,ts} \in R_t, \mathbf{s}_i, ts)$: Choose the user's error $\mathbf{e}_{i,ts}$ from ζ. Let $\mathbf{A}_{ts} = H(ts)$. Return the user's ciphertext for this timestamp $\mathbf{c}_{i,ts} = [\mathbf{A}_{ts} \cdot \mathbf{s}_i + t\mathbf{e}_{i,ts} + [\mathbf{x}_{i,ts} + \mathbf{r}_{i,ts}]_t]_q$ (based upon the secret key, the user's input, a small random error, and the public value).
- $Agg_{NS}(parms, \mathbf{s}', ts, \mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts})$: Let $\mathbf{A}_{ts} = H(ts)$. Return $\mathbf{y}_{ts} = [[\mathbf{A}_{ts} \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q]_t$

Correctness Suppose for simplicity that $\mathbf{r}_{i,ts} = 0$, as it can be assumed for this discussion that the noise has already been added to user input, as in the basic schemes we consider exact correctness. (We follow the same reasoning as LaPS; parameter choices for applying differential privacy are discussed in Section 7.) When adding n ciphertexts $\mathbf{c}_{i,ts}$, we find $[\mathbf{A}_{ts} \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q = [\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts})]_q$. The magnitude of the sum of the errors is bounded by $n \cdot t$, and the magnitude of the sum of the inputs is bounded by $n \cdot \frac{t}{2}$. Then so long as $\frac{3 \cdot n \cdot t}{2} < \frac{q}{2}$, $\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts})$ (2) will not overflow modulo q, guaranteeing correctness. Then reducing $\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts})$ modulo t removes the error terms (recall that the noise terms $\mathbf{e}_i$ are integral elements of R_q), leaving us with the sum of the users' inputs modulo t.

4.2 Second Additive Scheme (Message-Scaled)

The second variant of our scheme we present is message-scaled, where in encryption, the message is scaled by $\Delta = \lfloor \frac{q}{t} \rfloor$. This method of scaling up the message to allow for correct decryption (so noise disappears during rounding) is used in the LPR [42] and B/FV [26] lattice-based cryptosystems. In this scheme, we again consider the plaintext domain of R_t and the ciphertext domain of R_q, with N chosen to ensure the desired level of security. We require $q \gg t$, but do not require that q, t are coprime (though this may be needed for efficient implementation - see Section 6.1.1). The scheme $SLAP_{MS}$ is defined as follows:

- $Setup_{MS}(\lambda, t, n)$: Takes in the security parameter λ, the plaintext modulus t, and the number of users n. Choose the ciphertext modulus q such that $log_2(3) + log_2(n) + 2 \cdot log_2(t) < log_2(q)$ (3). (Note that this bound, required for correctness of Inequality 4, is stricter than Inequality 2 of $SLAP_{NS}$.) Choose the polynomial modulus N such that λ bits of security are provided for the RLWE problem with ring polynomial coefficients in $\mathbb{Z}_q$ [4, 5]. Choose users' secret keys $\mathbf{s}_0 \cdots \mathbf{s}_{n-1}$ from χ. Construct the aggregator's key as $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Choose a random oracle hash function H, mapping from the domain of all timestamps to the range of R_q [25]. Return $parms = (R_q, t, n, H)$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $NoisyEnc_{MS}(parms, \mathbf{x}_{i,ts} \in R_t, \mathbf{r}_{i,ts} \in R_t, \mathbf{s}_i, ts)$: Choose the user's error $\mathbf{e}_{i,ts}$ from ζ. Let $\mathbf{A}_{ts} = H(ts)$. Return the user's ciphertext for this timestamp $\mathbf{c}_{i,ts} = [\mathbf{A}_{ts} \cdot \mathbf{s}_i + \mathbf{e}_{i,ts} + \Delta[\mathbf{x}_{i,ts} + \mathbf{r}_{i,ts}]_t]_q$ (based upon the secret key, the user's input, a small random error, and the public value).
- $Agg_{MS}(parms, \mathbf{s}', ts, \mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts})$: Let $\mathbf{A}_{ts} = H(ts)$. Return $\mathbf{y}_{ts} = [\lfloor \frac{t}{q}(\mathbf{A}_{ts} \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}) \rceil]_t$

Correctness Again, let all noise values $\mathbf{r}_{i,ts} = 0$. When adding n ciphertexts $\mathbf{c}_{i,ts}$, we find $\mathbf{y}^{tmp} = [\mathbf{A}_{ts} \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q = [\sum_{i=0}^{n-1}(\mathbf{e}_{i,ts} + \Delta\mathbf{x}_{i,ts})]_q$. Then $\lfloor \frac{t}{q}\mathbf{y}^{tmp} \rceil = \lfloor \frac{t}{q}(\sum_{i=0}^{n-1}(\mathbf{e}_{i,ts} + \Delta\mathbf{x}_{i,ts})) \rceil$ will be equal to $\sum_{i=0}^{n-1} \mathbf{x}_{i,ts}$

when $\frac{t}{q} \cdot || \sum_{i=0}^{n-1} \mathbf{e}_{i,ts} - \frac{q \mod t}{t} \sum_{i=0}^{n-1} \mathbf{x}_{i,ts} || < \frac{q}{2}$ (4) [26]. Noting that $\frac{q \mod t}{t} < 1$, this is satisfied when $t^2 \cdot n \cdot \frac{3}{2} < \frac{q}{2}$.

4.3 Benefits and Novelty of SLAP

We highlight some of the novel and beneficial properties of SLAP:

- **Simplicity:** SLAP is vastly simpler than prior state-of-the-art work in RLWE-based PSA [9], with straightforward operations and parameter requirements. This is due to our white-box approach, which directly applies RLWE-based homomorphism instead of relying on preexisting FHE schemes. This approach allows for 1-element ciphertexts, as compared to the 2-element ciphertexts common in FHE, saving memory and computation. SLAP is thus easier and more lightweight to implement and run, making it more practical.
- **Direct Use of RLWE:** LaPS's use of RLWE was indirect, coming about through their use of the BGV FHE scheme [12]. SLAP is the first PSA scheme to rely directly on RLWE, showing how to apply the additive homomorphism inherent in RLWE terms to additive aggregation.
- **Smaller Ciphertexts:** As shown in Sections 7.1 and 7.3, SLAP has smaller ciphertexts as compared to previous RLWE-based PSA [9], with reductions in communication overhead of over 99% for larger parameter settings. The lighter communication load of SLAP makes it more practical for deployment among users with less bandwidth, e.g., smartphones and IoT nodes. As noted in Section 7.3, SLAP is not as efficient in communication as non-RWLE post-quantum PSA [24, 55], though it is comparable.
- **Extensibility and Modularity:** Thanks to the simplicity of SLAP, it is easy to apply other work in lattice-based cryptography, including the optimizations of RNS, NTT, and batching (discussed in Section 6), and the complex canonical embedding used in the CKKS scheme [14] to allow approximate computations. Further, SLAP can seamlessly be integrated with other work in fault tolerance [38], further improving its feasibility for practical use.
- **Efficiency and Throughput:** As seen in Section 7, SLAP can outperform optimized implementations of previous RLWE-based PSA by an order of magnitude, and the throughtput is further improved when batching is considered. Even considering the much larger ciphertexts of SLAP as compared to non-RLWE post-quantum PSA [55, 24], SLAP achieves comparable latency and much higher throughput.

5 Guarantees of Privacy and Security

5.1 Achieving Differential Privacy

To construct privacy-preserving PSA using the exact additive aggregation schemes presented in Section 4, we require that users add differentially private noise to their inputs before calling *NoisyEnc*. Adding differential privacy in this way is commonly utilized in other PSA works [9, 51, 2]; we adopt the procedures of LaPS because it is the most closely related work to ours in post-quantum PSA. For the distribution of added noise, we utilize the Discrete Laplacian distribution. In particular, we specify the noise added in each variant of *NoisyEnc* to be chosen in a differentially private manner: for an input $\mathbf{x}_{i,ts} \in R_t$ and a Discrete Laplacian parameter s, choose $\mathbf{r}_{i,ts} \in R_t$, where with probability β, $\mathbf{r}_{i,ts}$ has coefficients drawn from DL_s, and with probability $1-\beta$, $\mathbf{r}_{i,ts}$ will be zero.

The parameter s of the discrete Laplacian distribution used to draw the noise terms is determined by the number of users, range of the users' inputs, number of users adding differentially private noise, and desired level of privacy. This is formalized in Theorem 5.1. The PSA schemes $SLAP_{NS}$ and $SLAP_{MS}$ using this mechanism of noisy encryption achieve differential privacy and aggregator obliviousness:

Theorem 5.1 *Consider a scenario with n users, whose inputs fit in an interval of width w. Let the desired privacy level (ϵ, δ) satisfy $\epsilon > 0$ and $\delta \in (0,1)$. Define the discrete Laplacian parameter s as $s = \frac{w}{\epsilon}$. Let the proportion of honest users γ (i.e., the number of users adding differentially private noise) be at least $\frac{1}{n}\ln(\frac{1}{\delta})$. Then if $w \geq \frac{\epsilon}{3}$, then the schemes $SLAP_{NS}$ and $SLAP_{MS}$ using $NoisyEnc$ with differentially private noise drawn from DL_s achieve (ϵ, δ)-differential privacy. Further, these PSA schemes achieve (α, β)-accuracy, where $\beta \geq (\frac{2}{\delta})^{-\frac{1}{\gamma}}$ and $\alpha = \frac{4w}{\epsilon}\sqrt{\frac{1}{\gamma} ln(\frac{1}{\delta}) ln(\frac{2}{\beta})}$.*

Proof This follows directly from [9], Theorem 3, which itself is from [51], Lemma 1. Our desired function (summation) and method of differential privacy is the same.

This theorem gives a guarantee of privacy for the desired (ϵ, δ), and gives bounds on the probability and magnitude of the error resulting from the addition of differentially private noise as described above.

5.2 Privacy via Aggregator Obliviousness

We now prove that SLAP satisfies the security requirement of aggregator obliviousness with encrypt-once security (given in Section 3.4).

Theorem 5.2 *(Aggregator Obliviousness Security): Let the output of NoisyEnc be indistinguishable from random. Then both variants of SLAP are secure under aggregator obliviousness with encrypt-once.*

Our proof is very similar to existing PSA proofs [9, 51] that use aggregator obliviousness as their security notion, and we adapt these existing techniques for our own protocol. The security is essentially proved by showing that any adversary that can break the security of SLAP is able to leverage this to break the difficulty of RLWE.

We prove Theorem 5.2 as follows:

Proof We follow previous work [9, 51, 2] in assuming that a potential adversary can choose the noise $\mathbf{r}_{i,ts}$ as part of the Challenge phase in the security game of aggregator obliviousness. We aim to show that if there exists a PPT adversary $\mathcal{A}$ that wins the aggregator obliviousness security game, then there exists a PPT adversary $\mathcal{B}$ that can that can solve the RLWE problem. In the encrypt-once model, an adversary can only gain access to one encryption from a user at a particular timestamp, preventing the simple attack of taking the difference of two ciphertexts to learn about the plaintexts (as the terms $\mathbf{A}_{ts} \cdot \mathbf{s}_i$ would cancel out). $\mathcal{B}$ can make Sample queries to a challenger $\mathcal{C}$, who will provide $\mathcal{B}$ with sample values (pairs of ring polynomials). We omit modular notation for simplicity. We also modify the game of aggregator obliviousness to a real-or-random guess in the Challenge phase, following previous work [51, 9].

A fundamental property of aggregator obliviousness is that it acknowledges the case where the adversary compromises all but one participant, and allows that they inevitably learn the secret key of that participant and can therefore distinguish between valid encryptions and random values. To account for this, the definition of aggregator obliviousness requires that they do not learn any additional information about that participant. We note that if the adversary compromises all n or $n-1$ users and the aggregator, an adversary cannot choose differing challenge messages and thus cannot win the game, so we proceed assuming an adversary will always leave at least 2 parties uncompromised.

We now construct a reduction from SLAP to RLWE. We consider an adversary $\mathcal{B}$ trying to break the security of RLWE, and an adversary $\mathcal{A}$ who can purportedly break the security of SLAP. A challenger $\mathcal{C}$ evaluates the ability of $\mathcal{B}$ to break RLWE. $\mathcal{B}$ will formulate SLAP parameters $parms$ including R_q, t, n for a given security level λ to $\mathcal{A}$ as a response to a Setup query from $\mathcal{A}$. Here, R_q is the ring in which $\mathcal{B}$ is attempting to break RLWE. $\mathcal{B}$ will choose distinct values $j, k \in [0, \cup\{\square\}$. $\mathcal{B}$ then chooses secret keys $\mathbf{s}_i \leftarrow \chi$, where $i \notin \{j, k\}$. (As above, $\mathcal{B}$'s guess of uncompromised users j, k must be users $\mathcal{A}$ does not choose to compromise; this will occur with probability at least $\frac{1}{n^2}$.)

Supposing $\mathcal{A}$ will make Encrypt queries at up to $p = poly(\lambda)$ timestamps, $\mathcal{B}$ will make $p+1$ Sample queries to $\mathcal{C}$, receiving a set of tuples $\mathcal{S} = \{(\mathbf{a}_\sigma, \mathbf{b}_\sigma)\}_{\sigma \in \mathbb{Z}_{p+1}}$. $\mathcal{B}$ will then set the public hash $H(ts)$ in $parms$ such that for each of the $p+1$ possible values of ts that $\mathcal{A}$ will use, $H(ts)$ is some value $\mathbf{a}_\sigma$. Concretely, at every Encrypt query made, if the timestamp ts of that query has not been made, then $\mathcal{B}$ can select a not previously used pair $(\mathbf{a}_\sigma, \mathbf{b}_\sigma)$ from $\mathcal{S}$, and use $H(ts) = \mathbf{a}_\sigma$. On the other hand, if ts had previously been used in an Encrypt query for a different user, then the previously used value will be reused.

Upon an Encrypt query $(i, \mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}, ts)$ from $\mathcal{A}$ for a party $i \in [0, n) \cup \{\square\}$ at timestamp ts, if party i is not compromised and has not previously been the target of an Encrypt query at timestamp ts, $\mathcal{B}$ will return $NoisyEnc(parms, \mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}, \mathbf{s}_i, ts)$ if $i \notin \{j, k\}$. For the other values, our strategy for j, k is to let user j's secret key be the secret RLWE value and to let user k's secret key implicitly be the sum of all other users' keys. If $i = j$, then $\mathcal{B}$ finds the tuple $(\mathbf{a}_\sigma, \mathbf{b}_\sigma)$ such that $H(ts)$ was set to $\mathbf{a}_\sigma$, and returns $\mathbf{b}_\sigma + (\mathbf{x}_{j,ts} + \mathbf{r}_{j,ts})$ to $\mathcal{A}$. If $i = k$, $\mathcal{B}$ again finds the appropriate value $\mathbf{b}_\sigma$, and returns $-\mathbf{b}_\sigma - H(ts) \cdot \sum_{\ell \notin \{j,k\}} \mathbf{s}_\ell + \delta'(\mathbf{x}_{k,ts} + \mathbf{r}_{k,ts})$ to $\mathcal{A}$, where δ' is t if using $SLAP_{MS}$ and 1 if using $SLAP_{NS}$.

Upon a Compromise query from $\mathcal{A}$ for a party $i \in [0, n) \cup \{\square\}$, $\mathcal{B}$ will first check that $i \notin \{j, k\}$, and will abort if this is not the case. ($\mathcal{B}$ has probability approximately $\frac{1}{n^2}$ of correctly choosing j, k to not be users compromised by $\mathcal{A}$.) Otherwise, $\mathcal{B}$ responds to the query by returning party i's secret key $\mathbf{s}_i$ to $\mathcal{A}$. The set of uncompromised users is denoted as $K \subseteq [0, n) \cup \{\square\}$.

When $\mathcal{A}$ sends a Challenge query to $\mathcal{B}$, it chooses a set of not previously compromised users $U \in K$, and sends input-noise pairs $\{(\mathbf{x}_{u,ts}, \mathbf{r}_{u,ts})\}_{u \in U}$, where ts was not used in a previous Encrypt query. Then, $\mathcal{B}$ will once again find the tuple $(\mathbf{a}_\sigma, \mathbf{b}_\sigma)$ such that $H(ts)$ was set to $\mathbf{a}_\sigma$. Then, $\mathcal{B}$ computes $\mathbf{c}_{i,ts} = NoisyEnc(parms, \mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}, ts)$ for $i \in U \setminus \{j, k\}$. It also computes $\mathbf{c}_{j,ts} = \mathbf{b}_\sigma + (\mathbf{x}_{j,ts} + \mathbf{r}_{j,ts})$ and $\mathbf{c}_{k,ts} = -\mathbf{b}_\sigma - H(ts) \cdot \sum_{\ell \notin \{j,k\}} \mathbf{s}_\ell + \delta'(\mathbf{x}_{k,ts} + \mathbf{r}_{k,ts})$. Finally, $\mathcal{B}$ returns all ciphertexts $\mathbf{c}_{i,ts}$ for $i \in [0, n) \cup \{\square\}$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ can simply observe the response of $\mathcal{A}$ after receiving its Challenge results. If $\mathcal{A}$ responds that it has been given ciphertexts that are simply messages padded with a random sample, then $\mathcal{B}$ can also conclude the same for its game with $\mathcal{C}$. On the other hand, if $\mathcal{A}$ responds that it is in a real version of SLAP and had received a SLAP ciphertext, then $\mathcal{B}$ can conclude that

it received a RLWE sample from $\mathcal{C}$. Thus if $\mathcal{A}$ has an advantage greater than negligible in breaking aggregator obliviousness, then $\mathcal{B}$ can gain an advantage greater than negligible in breaking RLWE. This constitutes a reduction from RLWE to SLAP. □

6 Practical Considerations

6.1 Standard Lattice-Based Cryptography Optimizations

SLAP, like similar work in FHE [26, 13, 14], deals with ring polynomials where both their degree and coefficients may be large, with polynomial degrees up to $N = 2^{15}$ and coefficients that are hundreds of bits wide. As with similar work, we can apply optimizations to improve the runtime of the underlying polynomial arithmetic. Large polynomial degrees may make polynomial multiplication computationally intensive; to mitigate this the Number-Theoretic Transform (NTT) can improve the runtime of polynomial multiplication [41]. For large coefficients, Residue Number System (RNS) representations is used to break these large numbers down into smaller, more manageable components. Full-RNS variants of FHE schemes have reduced the complexity of those cryptosystems' most intensive operations to the complexity of the NTT [8, 15, 30], and bring a great practical benefit. The simple and limited lattice operations of SLAP allow these optimizations to be directly "plugged-in" to SLAP with no modifications required. These optimizations were also present in the HElib FHE library [29], which was used to implement LaPS.

6.1.1 Residue Number System

The Chinese Remainder Theorem states that given a number q that can can be written as a product of k coprime numbers $q = q_0 \cdot q_1 \cdot \cdots \cdot q_{k-1}$, the rings $\mathbb{Z}_q$ and $\bigtimes_{i=0}^{k-1} \mathbb{Z}_{q_i} = \mathbb{Z}_{q_0} \times \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_{k-1}}$ are isomorphic. This isomorphism can be applied to write a number $x \in \mathbb{Z}_q$ in Residue Number System (RNS) form as $([x]_{q_0}, [x]_{q_1}, \cdots, [x]_{q_{k-1}}) \in \bigtimes_{i=0}^{k-1} \mathbb{Z}_{q_i}$. Addition and multiplication on numbers in $\mathbb{Z}_q$ can then be carried out by simply performing the same operations coefficientwise on the operands in RNS form. This is most useful with a q that is significantly larger than a computer word (64 bits in modern systems) and can be factored into coprime q_i that can fit into a computer word. By writing numbers in RNS form, each of the k RNS components can fit into a computer word, so operations only require single-precision arithmetic.

Our schemes, like their analogues in fully homomorphic encryption (FHE), require some operations that cannot be directly performed in RNS. In particular, rounded division (required in Agg_{MS}) and modular reduction to another RNS base (used in both variants' Agg) are not easily implemented with numbers in RNS form. In fully homomorphic encryption, full-RNS variants have been created to allow for the use of RNS representations without having to reconstruct numbers in $\mathbb{Z}_q$ for the problematic operations [8, 30]. The full-RNS variant using floating-point operations [30] is much simpler and as efficient as the full-RNS variant using integer-only operations [8]. We thus adapt the integer-only RNS variant [30] for use in SLAP, and can directly apply their procedures.

Base Conversion Suppose we have a number x in RNS form with respect to q, written as $(x_i)_{i\in[0,k)} = ([x]_{q_0}, [x]_{q_1}, \cdots, [x]_{q_{k-1}})$. In both variants of SLAP, we wish to compute $[x]_t$ during aggregation as part of returning our result in R_t. We can then use the procedure of CRT Basis Extension from Section 2.2 of [30]. Let $q_i^* = \frac{q}{q_i} \in \mathbb{Z}$ and $\tilde{q}_i$ be the inverse of $q_i^* \pmod{q_i}$. Then our goal is to compute $[x]_t = [(\sum_{i=0}^{k-1} [x_i \cdot \tilde{q}_i]_{\tilde{q}_i} \cdot q_i^*) - v \cdot q]_t$, where $v \in \mathbb{Z}_k$ is equal to $\lceil \sum_{i=0}^{k-1} \frac{[x_i \cdot \tilde{q}_i]_{\tilde{q}_i}}{q_i} \rfloor$. This can be done by computing $y_i = [x_i \cdot \tilde{q}_i]_{\tilde{q}_i}$ (as an integer) and $z_i = \frac{y_i}{q_i}$ (in floating-point). Then $v = \sum_{i=0}^{k-1} z_i$, and we can compute $[x]_p = [(\sum_{i=0}^{k-1} y_i \cdot [q_i^*]_p]) - v \cdot [q]_p]_p$. To do this efficiently, the parameters $[q_i^*]_p$, $\tilde{q}_i$, and $[q]_p$ can all be precomputed.

Division with Rounding Now suppose that for x in RNS form, we wish to scale x by $\frac{t}{q}$ and round to the nearest integer, as in Agg_{MS}. To accomplish this, we use the procedure of Simple Scaling from Section 2.3 of [30]. We can then compute $y = \lceil \frac{t}{q} \cdot x \rfloor = [\lceil (\sum_{i=0}^{k-1} x_i \cdot (\tilde{q}_i \cdot \frac{t}{q_i})) \rfloor]_t$. To do this, we precompute $\frac{t \cdot \tilde{q}_i}{q_i} = \omega_i + \theta_i$ separated into integer and fractional parts, where $\omega_i \in \mathbb{Z}_t$ and $\theta_i \in [-\frac{1}{2}, \frac{1}{2})$. We can then compute the terms $\omega = [\sum_{i=0}^{k-1} x_i \cdot \omega_i]_t$ and $v = \lceil \sum_{i=0}^{k-1} x_i \cdot \theta_i \rfloor$ (where ω is computed with single-precision integer arithmetic, and v is computed with floating-point arithmetic). Then the final result is $[\omega + v]_t$.

6.1.2 Number-Theoretic Transform

In lattice-based cryptography, the most intensive low-level operation that is performed on polynomial operands is polynomial multiplication. Textbook algorithms for polynomial multiplication have complexity of $\mathcal{O}(N^2)$. Considering that values of N may commonly range from 2^{10} to 2^{15}, it is desirable to reduce this complexity. To accomplish this, the Number-Theoretic Transform (NTT) can be applied [41]. This strategy uses two operations: the forward transformation NTT and the inverse transformation $INTT$. For $\mathbf{a}, \mathbf{b} \in R_q$, the ring

polynomial product $\mathbf{a} \cdot \mathbf{b} \in R_q$ can be computed as $INTT(NTT(\mathbf{a}) \circ NTT(\mathbf{b}))$, where $\circ$ denotes coefficientwise modular multiplication (a linear-time operation). Because both NTT and $INTT$ can be computed in $\mathcal{O}(N \cdot log(N))$, the complexity of polynomial multiplication can thus be reduced from quadratic to loglinear. For polynomials modulo $x^N + 1$ and q, a negacyclic wrapped convolution can be used to perform the NTT [41]. For this, it is required that q satisfies the condition $q \equiv 1 \mod 2N$, so that primitive $2N$-th roots of unity can be easily found. When using RNS form (as in Section 6.1.1), all moduli q_i must satisfy this condition. In practice, these moduli are often precomputed as prime numbers.

6.2 Batching and Encoding

In many practical applications, users will have scalar data $x_{i,ts} \in \mathbb{Z}_t$, not polynomial inputs $\mathbf{x}_{i,ts} \in R_t$. A simple solution is to simply set the constant term (or any single coefficient) of $\mathbf{x}_{i,ts}$ equal to $x_{i,ts}$ and set all other coefficients to zero. However, this does not fully utilize all N coefficients of $\mathbf{x}_{i,ts}$. We can perform N batched aggregations in parallel by assigning each coefficient of the ciphertext to be a piece of data corresponding to a different computation. Because polynomial addition is a coefficientwise operation, we can use this simple batching method and do not require the common (and much more complex) method of batching using the polynomial version of the CRT (as in the the "double-CRT" form) [27]. The common method of batching could also be used, though encoding and decoding is more complex and would introduce more latency.

Another method of batching is to use RNS decomposition (see Section 6.1.1) on the plaintext, breaking coefficients modulo t into a tuple of coefficients modulo the coprime factors of t. This allows smaller messages to be batched together. Decomposing t into k' RNS moduli gives us a total of $N \cdot k'$ inputs packed together into a single ring polynomial when using both batching methods, allowing higher throughput with less computation and communication. This double batching will reduce the plaintext space, decreasing the allowable range of users' inputs and the number of users for a given ciphertext modulus.

Besides the above methods of batching, our scheme is also amenable to the application of a complex canonical embedding [14] that can be used to encode an array of floating-point values into a plaintext element of R_t, and perform floating-point aggregation.

7 Experimental Evaluation

In this section, we present experimental evaluations of our work. The LaPS scheme [9] is the most closely related work to ours, so we primarily compare SLAP against LaPS to best understand the improvements gained in performance and communication. We also compare SLAP to a simple non-PSA plain aggregation (that does not use SIMD or RNS batching), to analyze the slowdown in SLAP as compared to a realistic non-secure implementation. Further, we also consider other state-of-the-art post-quantum PSA [55, 24] when evaluating the practical throughput of SLAP.

In our experimental evaluations, we consider the same parameters for differential privacy as in previous work [9]: $\epsilon = 1$, $\delta = 0.1$, which gives a minimum required proportion of honest participants $\gamma \geq 0.0023$ for (ϵ, δ)-privacy. Choosing $\beta = \frac{2}{exp(10)}$ gives us (α, β)-accuracy with $\alpha = 4w\sqrt{\frac{1}{\gamma} ln(\frac{1}{\delta}) \cdot 10}$ (with total noise approximately $\mathcal{O}(w\sqrt{n})$). Concretely, this can be used to perform an aggregation with differential privacy over inputs within an interval of length 65 with 1000 users.

The noise of a differentially private aggregation inexorably grows with the number of users. Thus, for large-scale scenarios with many users, one should consider: either changing the parameters, requiring a larger proportion of honest users, or using other strategies such as rate-limiting to prevent adversaries from compromising honest participants' privacy by performing brute-force attacks [22, 21].

Our work's novelty is not in differential privacy, so our experimental results reflect runtime and memory consumption with respect to the number of users, overall plaintext space, and polynomial modulus degree. We do not further discuss noise or accuracy, as our work does not bring novel contribution in this area. Parameters for differential privacy have a negligible effect on runtime and memory consumption as compared to these. For a more detailed discussion, one can refer to the well-established literature applying differential privacy to aggregation and PSA [51, 9, 54, 48, 2].

7.1 Example Parameters

Table 2 shows minimal parameter choices to guarantee correctness and 128-bit security. These are derived directly from Equations (1) and (3). The ciphertext moduli required for $SLAP_{MS}$ is generally larger, which also necessitates a larger polynomial modulus degree. The ciphertext modulus size needed for $SLAP_{NS}$ is smaller than the outer ciphertext modulus of LaPS (q_1), and is often even smaller than the inner ciphertext modulus

Table 2 SLAP Parameter Requirements for 128-bit security

Users	$\|t\|$	$SLAP_{NS}$ $\|q\|$	$SLAP_{MS}$ $\|q\|$	$SLAP_{NS}$ N	$SLAP_{MS}$ N
100	32	42	75	2^{11}	2^{11}
1000	32	45	78	2^{11}	2^{11}
10000	32	49	82	2^{11}	2^{11}
10000	128	145	274	2^{13}	2^{14}
10^{15}	128	181	310	2^{13}	2^{14}
10^{21}	128	201	330	2^{13}	2^{14}

Table 3 Ciphertext Size Comparison of LaPS and $SLAP_{NS}$ ($SLAP_{NS}$ always at 128 bits of security)

Users	$\|t\|$	LaPS security	LaPS $\|q\|$	$SLAP_{NS}$ $\|q\|$	LaPS N	$SLAP_{NS}$ N	LaPS ciphertext bytes	$SLAP_{NS}$ ciphertext bytes	Percent Improvement	Improvement (LaPS size / SLAP size)
100	16	80	36	25	2^{11}	2^{10}	16384	8192	50.00%	2×
1000	16	80	39	28	2^{11}	2^{11}	16384	16384	0.00%	1×
10000	16	80	43	32	2^{11}	2^{11}	16384	16384	0.00%	1×
100	32	128	63	41	2^{20}	2^{11}	8388608	16384	99.80%	512×
1000	32	128	64	44	2^{20}	2^{11}	8388608	16384	99.80%	512×
10000	32	128	67	48	2^{20}	2^{11}	16777216	16384	99.90%	1024×
10000	128	80	196	144	2^{24}	2^{13}	536870912	196608	99.96%	2730.67×
10^{15}	128	80	201	184	2^{24}	2^{13}	536870912	196608	99.96%	2730.67×
10^{21}	128	80	221	204	2^{24}	2^{13}	536870912	262144	99.95%	2048×

(q_0) [9]. Also, our required polynomial modulus degrees are smaller. This shows that SLAP is more efficient than previous work in communication overhead, due to not doubly enclosing input in Augmented LWE terms and FHE ciphertexts.

7.2 Implementation Details

To evaluate the efficiency of our scheme, we analyzed the performance of both a basic implementation of the scheme as originally presented in Section 4 and a more optimized implementation using the optimizations discussed in Section 6, both utilizing the differentially private noise mechanism of Section 5.1. In the optimized library, the full-RNS optimizations (as in Section 6.1.1) are implemented, and the Number-Theoretic Transform is used to accelerate polynomial multiplication (as in Section 6.1.2). The basic implementation does not integrate these optimizations, but uses NTL [53] for integer and polynomial arithmetic. The optimized version of SLAP includes a submodule for the optimized ring polynomial arithmetic, which can be used independently of the implementations of SLAP. Our library and driver implementations (in C++14) are published anonymously at `https://anonymous.4open.science/r/slap-codaspy/`. The code is single-threaded, though SLAP and our library can be parallelized for high-performance applications. Our experiments were run on a server computer with an AMD EPYC 7451 CPU, running at up to 2.3GHz. We use the standard ciphertext modulus and polynomial modulus specifications from

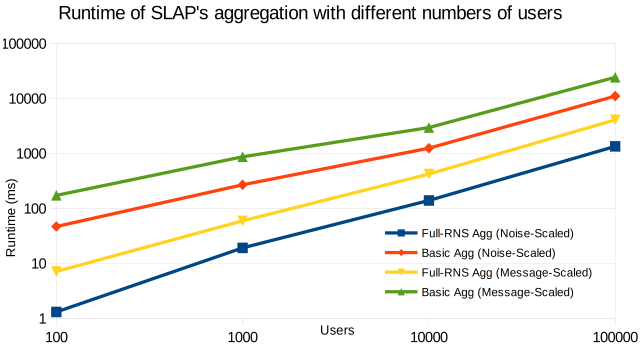

Fig. 3 Scalability with a 32-bit plaintext space

`HomomorphicEncryption.org` [4]. Our tests took average runtimes over 50 trials.

7.3 Basic Comparison to State-of-the-art PSA

We generally consider users to have a message space of 16 bits. This is in line with the experimental evaluations performed by LaPS, and is useful for many applications, e.g. quantized machine learning [9, 16]. LaSS [24] considers an 85-bit plaintext space, and Waldner et al. [55] considers plaintext spaces of 2, 16, and 64 bits.

We first compare SLAP directly to LaPS, matching exactly the largest set of parameters they considered in order to make a fair comparison against their reported results. LaPS is implemented using the HElib homomorphic encryption library [29], which incorporates optimizations such as the double-CRT representation and Discrete Fourier Transform for efficient arithmetic. We compare both the basic and optimized (full-RNS)

Table 4 Latency of SLAP with 1000 users and 16-bit messages

Variant	Full-RNS $NoisyEnc$	Full-RNS Agg	Basic $NoisyEnc$	Basic Agg
Noise-Scaled	1.17 ms	3.26 ms	43.53 ms	95.38 ms
Message-Scaled	5.91 ms	16.98 ms	166.40 ms	272.26 ms

Table 5 Speedup of SLAP vs. LaPS (using HElib) with 1000 users and 16-bit messages at 128 bits of security

Variant	Full-RNS $NoisyEnc$	Full-RNS Agg	Basic $NoisyEnc$	Basic Agg
Noise-Scaled	65.97x	20.76x	1.79x	0.71x
Message-Scaled	13.09x	3.98x	0.46x	0.25x

implementations of SLAP to LaPS, though comparison with the full-RNS implementation is the more direct one. We begin by considering the largest set of parameter settings used in LaPS, i.e., a 16-bit plaintext space, 128 bits of security, and 1000 users. Parameter settings for LaPS are taken directly from their publication. Table 4 shows the performance of SLAP with these parameters, along with the time it takes to do the computation in plaintext.

We note that our experiments are run on a server computer, while the experiments of LaPS, LaSS, and Ernst et al. were run on (presumably weaker) laptop computers. However, these differences in hardware do not account for the orders-of-magnitude improvements in latency and throughput that SLAP shows over these schemes. The vast majority of the improvement over LaPS in latency is due to the smaller operands and more efficient operations of SLAP, and our improvement in throughput over LaSS and Ernst et al. is due to the combination of comparable latency and amenability to batching.

Table 5 shows the speedup of SLAP as compared to LaPS for operation latency. From this, we see that our full-RNS implementation of SLAP is able to speed up $NoisyEnc$ by 65.97x and Agg by 20.76x (with $SLAP_{NS}$). *Even without any optimizations* such as those present in HElib, our basic implementation of SLAP still had performance comparable to LaPS. Notably, this comparison is for the parameters of $n = 1000$, $|t| = 16$, where (as shown in Table 3) there is no difference in ciphertext size as seen at larger parameter settings; this shows that the runtime improvements of SLAP are not solely due to smaller operands, but are a result of the simpler design of SLAP.

Besides the runtime comparison, SLAP shows significant improvements in the necessary communication overhead as compared to LaPS. Table 3 shows the sizes of ciphertexts in $SLAP_{NS}$ and LaPS for different parameter settings. ($SLAP_{MS}$ also showed improvement, but the improvement was less due to its stricter parameter bounds.) In this analysis, we assume an implementation would store ciphertext moduli in 64-bit words - when the actual minimum number of bytes is counted instead, then the improvement of $SLAP_{NS}$ becomes slightly greater. For every setting shown, the ciphertext size of $SLAP_{NS}$ is less than or equal to that of LaPS, and shows significant improvement for larger parameters. The decrease in ciphertext size is over 99% for larger parameter sets. This difference in size of operands implies greatly reduced communication time, and also helps to explain the runtime improvements of our scheme as compared to LaPS. In SLAP, ciphertexts and keys are the same size, as both are single elements of R_q. While the ciphertexts of SLAP are larger than those of LaSS and Ernst et al. [24], the plaintext elements transferred per ciphertext is comparable - these schemes have ciphertext sizes of 128 and 85 bits respectively, while SLAP has 39 bits per batched plaintext for 1000 users and a 16-bit plaintext space.

As a practical example, Table 11 shows the estimated communication times for a users' ciphertext for common Internet upload speeds in the United States, given in megabits/second (Mbps). Even at the slowest upload speeds and largest parameter sets, ciphertexts of $SLAP_{NS}$ can be sent in under a second. In contrast, ciphertexts from LaPS may take much longer to send to the point of human-noticeable latency, even taking over a minute to send ciphertexts from the largest set of parameters considered. This shows the practical improvements that SLAP can bring to users with slower upload speeds when using RLWE-based PSA.

We can thus surmise that the improvements of SLAP over LaPS are partially due to smaller parameters allowed for by a purpose-built design, where instead of using FHE as a black box, our specifically-designed and much simpler operations can be used.

Noting that $SLAP_{NS}$ is the more efficient variant in terms of latency, we can also compare it to other post-quantum PSA work, as shown in Table 6 (1000 users, 16-bit plaintexts or as close as possible). $SLAP_{NS}$ is comparable in latency to both LaSS and Ernst et al., and all of those show a great improvement as compared to LaPS. LaSS and Ernst et al. both feature aggregation time of about 1 millisecond for encryption and aggrega-

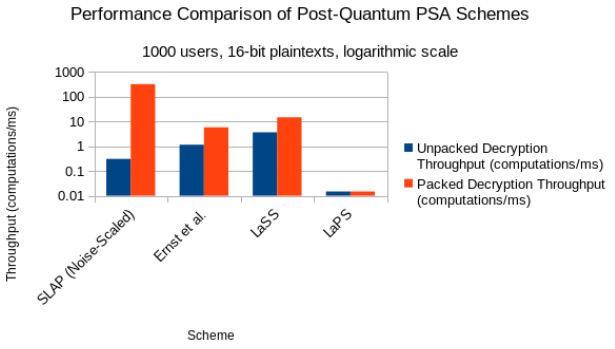

Fig. 4 Throughput Comparison of Post-Quantum PSA

tion. This comparison was only done for the small case of 1000 users, due to the diverse experiments between the 4 works. As shown in Section 7.4, the comparable latency of SLAP to other PSA work, combined with our propensity for batching, can lead to very strong practical performance.

While SLAP is capable of significant improvements in communication overhead as compared to LaPS, its ciphertexts are significantly larger than those of other state-of-the-art post-quantum PSA. LaSS's AES ciphertexts are 128 bits, while Ernst et al.'s scheme uses 85-bit ciphertexts. Even when accounting for batching, the larger per-input ciphertext expansion of SLAP leads to worse throughput for communication. Large ciphertext expansion is a common issue to RLWE-based cryptosystems, and addressing this is an active area of research [19, 40]. While this work's benefits are mostly in simplicity and server-side computational throughput, reducing the ciphertext expansion for RLWE-based PSA is a focus of our future work.

7.4 Scalability, Throughput, and Batching

We next tested the scalability of the aggregation and decryption of SLAP (both basic and full-RNS implementations) with respect to the number of users in the aggregation. Taking $|t| = 32$, we evaluated the runtime of Agg for $n = 100, 1000, 10000$, and 100000. As Figure 3 shows, the runtime of aggregation increases linearly with the number of users, showing the scalability of SLAP.

The direct comparison to LaPS only compared the case when a single message is included in a ciphertext. However, as discussed in Section 6.2, we can pack N inputs into a single scheme plaintext, by setting the coefficients of the plaintext polynomial to each of the N inputs. Doing this greatly increases our scheme's overall throughput, as seen in Tables 7 and 8. With the parameters of 1000 users and 16-bit messages, the ciphertext modulus degree is 2048, so we pack 2048 messages into one ciphertext. As shown in Tables 7 and 8, introducing even simple batching greatly increases the throughput SLAP can achieve. With this, the runtime of PSA's aggregation can be improved to 628,605 aggregations/second, reducing our amortized runtime to only microseconds.

We next examined the possible throughput of SLAP, accounting for using the double-packing method from Section 6.2, encryption time, network latency, and aggregation time. We show the results of this analysis in Tables 9 and 10. We tested 16-bit messages packed into plaintext domains of 32 and 64 bits, each of which exceeds the parameters experimentally evaluated for LaPS. Each plaintext polynomial with N coefficients and k' RNS components for t can hold $N \cdot k'$ data elements total, with accounting for overflow. In our analysis, we assume that the download speed of the aggregator is not a bottleneck (e.g., the aggregator is a server with a gigabit connection), and that users will have an unlimited supply of data ready that it can run $NoisyEnc$ on, then send to the aggregator. Then the limiting factor in SLAP's practical throughput is the greatest of the runtimes of $NoisyEnc$ and Agg, and the users' upload times. Thus both schemes are able to show a throughput of up to 78,138 aggregations per second at 5 Mbps, and 390,691 aggregations per second at 25 Mbps. In contrast, a plain aggregation's main bottleneck was computation, due to the lack of the high ciphertext expansion common in RLWE encryption. SLAP achieves throughput only one order of magnitude less than plain aggregation using RNS for SIMD operations.

From these experiments, we can conclude that $SLAP_{NS}$ is generally superior to $SLAP_{MS}$. The noise-scaled variant of SLAP has better parameter bounds, ciphertext sizes, and performance than the message-scaled variant. We can further conclude that PSA implemented with SLAP can be extremely computationally efficient in terms of computation. With ciphertexts that are smaller as compared to previous work, SLAP also requires less communication, which can further decrease the latency that end users of PSA will experience. Overall, these experiments show the high performance of SLAP, especially as compared to other work.

We can also compare SLAP to other quantum-secure PSA when considering throughput, which is another important practical metric besides simple latency that can be used to evaluate real-world performance. In Figure 4, we show the decryption throughput for SLAP as well as the protocols of Ernst et al., LaPS, and LaSS. As indicated by the schemes' decryption latencies for 1000 users (as shown in Table 6, $SLAP_{NS}$ has slightly worse throughput without batching at this small scale. However, when considering batching, $SLAP_{NS}$ shows orders-

Table 6 Latency Comparison of Post-Quantum PSA for 1000 users and 16-bit plaintexts (if possible)

Scheme	$SLAP_{NS}$	Ernst et al. [24]	LaSS [55]	LaPS [9]
Encryption Latency (ms)	1.17	0.913	0.509	77.3304
Decryption Latency (ms)	3.26	0.875	0.277	67.6243

Table 7 Speedup of SLAP vs. plain aggregation with 1000 users and 16-bit messages

Variant	Full-RNS Agg	Basic Agg	Full-RNS Agg, batched	Basic Agg, batched
Noise-Scaled	0.0139x	0.0005x	28.39x	0.97x
Message-Scaled	0.0027x	0.0002x	5.45x	0.34x

Table 8 Throughput (with batching) of SLAP with 1000 users, 16-bit messages, 2048 data points per ciphertext (calculations/second)

Variant	Full-RNS $NoisyEnc$	Full-RNS Agg	Basic $NoisyEnc$	Basic Agg
Noise-Scaled	1,747,079	628,605	47,369	21,470
Message-Scaled	346,726	120,626	12,308	7,522

Table 9 Throughput of full-RNS SLAP with 1000 users and 16-bit messages with 5 Mbps upload speed

Variant	$\|t\|$	k'	N	$NoisyEnc$ (ms)	Agg (ms)	Upload Time (ms) at 5 Mbps	Ciphertext Uploads/s	SLAP Throughput (aggs/s)	Slowdown vs. Plain
Noise-Scaled	32	1	2,048	1.05	2.70	26.21	38.15	78,138	14.69
Message-Scaled	32	1	4,096	4.33	14.62	104.84	9.54	39,069	29.38
Noise-Scaled	64	2	2,048	4.32	15.84	52.42	19.08	78,138	7.34
Message-Scaled	64	2	4,096	13.81	52.12	104.84	9.54	78,138	7.34

Table 10 Throughput of full-RNS SLAP with 1000 users and 16-bit messages with 25 Mbps upload speed

Variant	$\|t\|$	k'	N	Upload Time (ms) at 25 Mbps	Ciphertext Uploads/s	SLAP Throughput (aggs/s)	Slowdown vs. Plain
Noise-Scaled	32	1	2,048	5.24	190.77	39,061	2.94
Message-Scaled	32	1	4,096	20.97	47.69	195,345	5.88
Noise-Scaled	64	2	2,048	10.48	95.38	390,691	2.09
Message-Scaled	64	2	4,096	20.97	47.69	390,691	3.49

Table 11 Ciphertext Upload Time (seconds) of LAPS and $SLAP_{NS}$ ($SLAP_{NS}$ always at 128 bits of security)

Users	$\|t\|$ (bits)	LaPS Security (bits)	$SLAP_{NS}$ (s) at 5 Mbps	LaPS (s) at 5 Mbps	$SLAP_{NS}$ (s) at 25 Mbps	LaPS (s) at 25 Mbps	$SLAP_{NS}$ (s) at 50 Mbps	LaPS (s) at 50 Mbps
100	16	80	1.31E-02	2.62E-02	2.62E-03	5.24E-03	1.31E-03	2.62E-03
1000	16	80	2.62E-02	2.62E-02	5.24E-03	5.24E-03	2.62E-03	2.62E-03
10000	16	80	2.62E-02	2.62E-02	5.24E-03	5.24E-03	2.62E-03	2.62E-03
100	32	128	2.62E-02	1.34E+01	5.24E-03	2.68E+00	2.62E-03	1.34E+00
1000	32	128	2.62E-02	1.34E+01	5.24E-03	2.68E+00	2.62E-03	1.34E+00
10000	32	128	2.62E-02	2.68E+01	5.24E-03	5.37E+00	2.62E-03	2.68E+00
10000	128	80	3.15E-01	8.59E+02	6.29E-02	1.72E+02	3.15E-02	8.59E+01
10E15	128	80	3.15E-01	8.59E+02	6.29E-02	1.72E+02	3.15E-02	8.59E+01
10E21	128	80	4.19E-01	8.59E+02	6.29E-02	1.72E+02	4.19E-02	8.59E+01

of-magnitude improvement over the other 3 schemes. It should be noted that due to implementation differences in hardware, programming languages, etc., the comparison is not quite exact, though the improvement is much larger than can be attributed to the hardware or language differences alone. LaSS and Ernst et al. use Go, while LaPS and our implementation use C++.

In considering the throughput of communication, the relative disadvantage of SLAP as compared to LaSS and Ernst et al. is still present, but greatly reduced when considering the propensity of SLAP for batching. Recalling the 128 and 85 bits (16 and 11 bytes) used for plaintexts (and ciphertexts) for LaSS and Ernst et al. respectively, we suppose that for the application of computing sums of 1000 users' 16-bit inputs without overflow (requiring $16 + 10 = 26$ bits of plaintext space), these protocols can pack 4 and 3 inputs into a message, respectively. (In practical use, the inputs for the scheme of Ernst et al. may be even more limited due to the scaling up of inputs required by their use of an approximately homomorphic PRF.) SLAP can also use this packing, managing to fit $k' = 7$ 26-bit messages into a 192-bit plaintext space. These parameters lead to $|q| = 214$ and $N = 8192$ for $SLAP_{NS}$, leading to a ciphertext of approximately 262144 bytes, holding $N * k' = 7 * 8192 = 57344$ user inputs. With this, $SLAP_{NS}$ achieves about 86% and 80% of the communication throughput of LaSS and Ernst et al., respectively. (Batching is not considered in LaPS,

and the highly complex nature of their scheme makes assumptions such as those for LaSS and Ernst et al. difficult.) As noted above, the bottleneck of PSA schemes with aggregation and decryption latency on order of a few milliseconds is computation, not communication, so the relatively larger communication of SLAP as compared to LaSS and Ernst et al. is unlikely to bring a practical slowdown for high-bandwidth servers. However, communication overhead is still a practical issue important for the sake of client bandwidth, and we aim to address the ciphertext expansion in future research.

7.5 Comparison to Other PSA

Early work by Shi et al. [51] does not report experimental results, but estimate that their work should support 0.6ms encryption and 1.5s decryption on modern hardware. The schemes presented by [49] and [1] are highly similar; both are lattice-based PSA-like work specifically tailored to the scenario of smart meters, which is different from general PSA. [49] fixes some security holes in [1], but does not provide any experimental evaluation for a direct comparison. In both cases, the communication overhead (measured in rounds) their schemes incur is higher than the single round of communication per aggregation in a PSA scheme. In [3], the runtime of aggregation is dominated by LWE decryption, which runs at 0.14 ms; however, the parameter setting of $|q| < 15$ and $N = 256$ used in their evaluation is extremely small as compared to parameters used in SLAP or LaPS.

We wrote a test implementation of selected aggregation schemes without quantum security [51, 33, 34, 36] to examine the relative performance. Each scheme was run with a 2048-bit plaintext space, and had a number of users ranging from 4 to 1964, with each test having 40 more users. The results of these tests are reported in Table 12, with some operations' runtimes given as per-user time as appropriate. Aggregation in non-quantum secure formats can run in microseconds or milliseconds; SLAP thus shows performance at least comparable to that of non-quantum aggregation.

8 Conclusion

In this work, we presented SLAP, a new Private Stream Aggregation scheme with two variations featuring efficiency, simplicity, and quantum security. The setup of SLAP allows for security across multiple rounds of aggregation, improving upon previous work. We prove both the privacy and security of SLAP under differential privacy and aggregator obliviousness, and apply practical optimizations. Our implementations of both variants of SLAP shows improvements of up to over 20x for aggregation against previous work. Our experiments also show that SLAP can achieve aggregation with throughput of up to 390,691 aggregations/second at 25Mbps, and is only one order of magnitude slower than plain aggregation. SLAP is particularly efficient in its communication overhead as compared to the previous state-of-the-art lattice-based PSA, achieving a decrease in ciphertext size of up to 99.96% for large parameters, which greatly improves the practical viability of lattice-based PSA. Further, SLAP is comparable in latency and shows great improvements in practical throughput as compared to other leading post-quantum PSA schemes [24, 55]. We conclude that SLAP brings both theoretical and practical improvements to the current state-of-the-art in post-quantum PSA.

Declarations

Conflict of interest

The authors declare that they have no conflict of interest.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Funding

This work was supported by Facebook as a winner of the Role of Applied Cryptography in a Privacy-Focused Advertising Ecosystem Facebook RFP. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

1. Abdallah A, Shen XS (2016) A lightweight lattice-based homomorphic privacy-preserving data aggregation scheme for smart grid. IEEE Transactions on Smart Grid 9(1):396–405
2. Ács G, Castelluccia C (2011) I have a dream!(differentially private smart metering). In: International Workshop on Information Hiding, Springer, pp 118–132
3. Agarkar AA, et al. (2020) Post quantum security solution for data aggregation in wireless sensor networks. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, pp 1–8

Table 12 Runtime (μs) of Schemes without Quantum Security (2048-bit plaintext space)

Operation	Joye agg. [33]	Joye enc.	Jung 2013 agg. (per user) [34]
Time (μs)	3563.46	2.07	32.02

Operation	Jung 2013 enc.	Jung PDA enc. (per user) [36]	Jung PDA enc. of Special User 2 (per user)
Time (μs)	32.02	0.06	1776.15

Operation	Jung PDA agg. (per user)	Shi agg. (per user) [51]	Shi enc.
Time (μs)	12.83	0.16	491.20

4. Albrecht M, Chase M, Chen H, Ding J, Goldwasser S, Gorbunov S, Halevi S, Hoffstein J, Laine K, Lauter K, Lokam S, Micciancio D, Moody D, Morrison T, Sahai A, Vaikuntanathan V (2018) Homomorphic encryption security standard. HomomorphicEncryption.org, Toronto, Canada, Tech rep
5. Albrecht MR, et al. (2018) Estimate all the {LWE, NTRU} schemes! In: International Conference on Security and Cryptography for Networks, Springer, pp 351–367
6. Ananth P, Jain A, Jin Z, Malavolta G (2020) Multi-key fully-homomorphic encryption in the plain model. In: Theory of Cryptography Conference, Springer, pp 28–57
7. Archer D, et al. (2017) Applications of homomorphic encryption. HomomorphicEncryption org, Redmond WA, Tech Rep
8. Bajard JC, Eynard J, Hasan MA, Zucca V (2016) A full RNS variant of FV like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography, Springer, pp 423–442
9. Becker D, Guajardo J, Zimmermann KH (2018) Revisiting Private Stream Aggregation: Lattice-Based PSA. In: NDSS
10. Bell J, Bonawitz K, Gascón A, Lepoint T, Raykova M (2020) Secure single-server aggregation with (poly) logarithmic overhead. IACR Cryptol ePrint Arch
11. Blanco-Chacón I (2020) On the RLWE/PLWE equivalence for cyclotomic number fields. Applicable Algebra in Engineering, Communication and Computing pp 1–19
12. Brakerski Z, Vaikuntanathan V (2011) Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Annual cryptology conference, Springer, pp 505–524
13. Brakerski Z, Gentry C, Vaikuntanathan V (2014) (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6(3):1–36
14. Cheon JH, Kim A, Kim M, Song Y (2017) Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security, Springer, pp 409–437
15. Cheon JH, Han K, Kim A, Kim M, Song Y (2018) A full RNS variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography, Springer, pp 347–368
16. Crane M, Trotman A, O'Keefe R (2013) Maintaining discriminatory power in quantized indexes. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pp 1221–1224
17. Danezis G, Fournet C, Kohlweiss M, Zanella-Béguelin S (2013) Smart meter aggregation via secret-sharing. In: Proceedings of the first ACM workshop on Smart energy grid security, pp 75–80
18. Ding J, Gao X, Takagi T, Wang Y (2019) One sample ring-lwe with rounding and its application to key exchange. In: International Conference on Applied Cryptography and Network Security, Springer, pp 323–343
19. Dobraunig C, Grassi L, Helminger L, Rechberger C, Schofnegger M, Walch R (2021) Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive
20. Duchi JC, Jordan MI, Wainwright MJ (2013) Local privacy and statistical minimax rates. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, IEEE, pp 429–438
21. Dwork C (2008) Differential privacy: A survey of results. In: International conference on theory and applications of models of computation, Springer, pp 1–19
22. Dwork C, Yekhanin S (2008) New efficient attacks on statistical disclosure control mechanisms. In: Annual International Cryptology Conference, Springer, pp 469–480
23. Erkin Z, Tsudik G (2012) Private computation of spatial and temporal power consumption with smart meters. In: International Conference on Applied Cryptography and Network Security, Springer, pp 561–577
24. Ernst J, Koch A (2021) Private stream aggregation with labels in the standard model. Proceedings on Privacy Enhancing Technologies 4:117–138

25. Evans D, Kolesnikov V, Rosulek M (2017) A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security 2(2-3)
26. Fan J, Vercauteren F (2012) Somewhat practical fully homomorphic encryption. IACR Cryptol ePrint Arch 2012:144
27. Gentry C, Halevi S, Smart NP (2012) Homomorphic evaluation of the AES circuit. In: Annual Cryptology Conference, Springer, pp 850–867
28. Group S (????) Sony research award program
29. Halevi S, Shoup V (2014) Helib. Retrieved from HELib: `https://githubcom/shaih/HElib`
30. Halevi S, Polyakov Y, Shoup V (2019) An improved RNS variant of the BFV homomorphic encryption scheme. In: Cryptographers' Track at the RSA Conference, Springer, pp 83–105
31. Hoffstein J, Pipher J, Silverman JH (1998) NTRU: A ring-based public key cryptosystem. In: International Algorithmic Number Theory Symposium, Springer, pp 267–288
32. Inc F (2020) Role of applied cryptography in a privacy-focused advertising ecosystem request for proposals
33. Joye M, Libert B (2013) A scalable scheme for privacy-preserving aggregation of time-series data. In: International Conference on Financial Cryptography and Data Security, Springer, pp 111–125
34. Jung T, Mao X, Li XY, Tang SJ, Gong W, Zhang L (2013) Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation. In: 2013 Proceedings IEEE INFOCOM, IEEE, pp 2634–2642
35. Jung T, Li XY, Wan M (2014) Collusion-tolerable privacy-preserving sum and product calculation without secure channel. IEEE Transactions on Dependable and secure computing 12(1):45–57
36. Jung T, Han J, Li XY (2016) PDA: semantically secure time-series data analytics with dynamic user groups. IEEE Transactions on Dependable and Secure Computing 15(2):260–274
37. Karl R, Burchfield T, Takeshita J, Jung T (2019) Non-interactive MPC with trusted hardware secure against residual function attacks. In: International Conference on Security and Privacy in Communication Systems, Springer, pp 425–439
38. Karl R, Takeshita J, Jung T (2020) Cryptonite: A framework for flexible time-series secure aggregation with online fault tolerance. Cryptology ePrint Archive, Report 2020/1561
39. Li Q, Cao G (2013) Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In: International Symposium on Privacy Enhancing Technologies Symposium, Springer, pp 60–81
40. Li Y, Zhou J, Li Y, Au OC (2015) Reducing the ciphertext expansion in image homomorphic encryption via linear interpolation technique. In: 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP), IEEE, pp 800–804
41. Longa P, Naehrig M (2016) Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: International Conference on Cryptology and Network Security, Springer, pp 124–139
42. Lyubashevsky V, Peikert C, Regev O (2013) On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) 60(6):1–35
43. Martins P, Sousa L, Mariano A (2017) A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys (CSUR) 50(6):1–33
44. McMahan B, Ramage D (2017) Federated learning: Collaborative machine learning without centralized training data
45. Mofrad S, Zhang F, Lu S, Shi W (2018) A comparison study of intel sgx and amd memory encryption technology. In: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, pp 1–8
46. Mukherjee P, Wichs D (2016) Two round multiparty computation via multi-key fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp 735–763
47. Przydatek B, Song D, Perrig A (2003) SIA: Secure information aggregation in sensor networks. In: Proceedings of the 1st international conference on Embedded networked sensor systems, pp 255–265
48. Rastogi V, Nath S (2010) Differentially private aggregation of distributed time-series with transformation and encryption. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp 735–746
49. Romdhane RB, Hammami H, Hamdi M, Kim TH (2019) At the cross roads of lattice-based and homomorphic encryption to secure data aggregation in smart grid. In: 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE, pp 1067–1072
50. Rosca M, Stehlé D, Wallet A (2018) On the ring-LWE and polynomial-LWE problems. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp 146–173
51. Shi E, Chan TH, Rieffel E, Chow R, Song D (2011) Privacy-preserving aggregation of time-series data. In: Proc. NDSS, vol 2, pp 1–17

52. Shor PW (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science, IEEE, pp 124–134
53. Shoup V, et al. (2001) NTL: A library for doing number theory
54. Valovich F, Aldà F (2017) Computational differential privacy from lattice-based cryptography. In: International Conference on Number-Theoretic Methods in Cryptology, Springer, pp 121–141
55. Waldner H, Marc T, Stopar M, Abdalla M (2021) Private stream aggregation from labeled secret sharing schemes. IACR Cryptol ePrint Arch 2021:81
56. Won J, Ma CY, Yau DK, Rao NS (2014) Proactive fault-tolerant aggregation protocol for privacy-assured smart metering. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications, IEEE, pp 2804–2812
57. Xue K, Yang Q, Li S, Wei DS, Peng M, Memon I, Hong P (2018) PPSO: A privacy-preserving service outsourcing scheme for real-time pricing demand response in smart grid. IEEE Internet of Things Journal 6(2):2486–2496
58. Yang Q, Liu Y, Chen T, Tong Y (2019) Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST) 10(2):1–19
59. Yang Y, Huang X, Liu X, Cheng H, Weng J, Luo X, Chang V (2019) A comprehensive survey on secure outsourced computation and its applications. IEEE Access 7:159,426–159,465