SLAP: Simple Lattice-Based Private Stream Aggregation Protocol

Jonathan Takeshita, *Student Member, IEEE,* Ryan Karl, Ting Gong, and Taeho Jung, *Member, IEEE,*

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 or a lack of post-quantum security. 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 lattice-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 lattice-based PSA.

Index Terms—Public key cryptosystems, Lattice-based Cryptography, Private Stream Aggregation, SIMD, RNS.

1 Introduction

Many common 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 data privacy of individual data holders is paramount. Examples of this include health care, education, and advertising, where an individual's privacy is often protected by law [1]. Due to such demands, secure aggregation of user-generated datasets has gained interest in the industry where large-scale user-generated datasets need to be analyzed. Facebook Research and Sony Research call for proposals in secure aggregation [2] and secure time-based data aggregation [3], and Google has adopted secure aggregation in their secure federated learning framework [4]. It is thus desirable to consider schemes that allow data aggregation to be computed without violating users' privacy efficiently.

Simply put, the secure data aggregation problem is defined 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; however, this does not prevent the data 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 [5], smart metering [6], and can be seen in areas including healthcare, education, and finance that require computation over users' private data.

A simple solution to this problem is to simply 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 very 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), which is the secure and private aggregation of time-series data, shown in Figure 2. As originally introduced by Shi et al. [7], PSA encompassed security and privacy with the idea of *aggregator obliviousness*. Aggregator obliviousness captures security and privacy, requiring 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 [7], [8], [9], [10], [11] due to its wide applicability to secure computation.

Other approaches include the use of 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 [12], [13], [14]. However, there are many issues with using existing HE schemes, including key management, computational overhead, and complexity. Trusted

- *J. Takeshita, R. Karl, and T. Jung are with the Department of Computer Science and Engineering, The University of Notre Dame, Notre Dame, IN, 46556.*
  E-mail: [jtakeshi,rkarl,tjung]@nd.edu
- *T. Gong is with the Department of Mathematics, The University of Notre Dame, Notre Dame, IN, 46556.*
  E-mail: [tgong]@nd.edu

Manuscript received May 5, 2021; revised May 5, 2021.

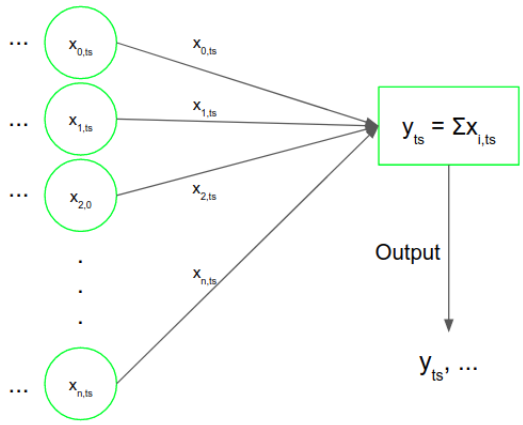

Fig. 1. Time-series Plain Aggregation

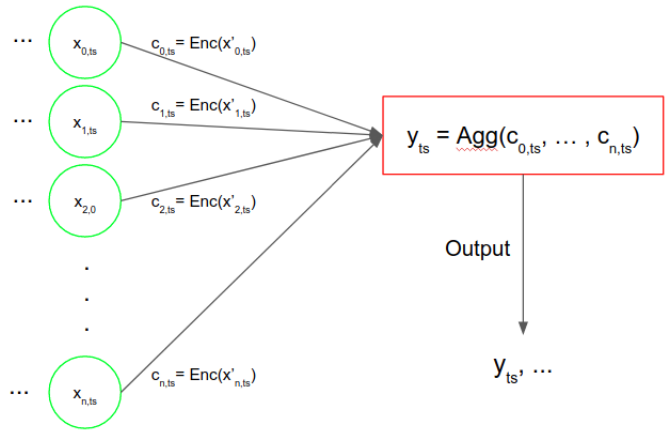

Fig. 2. Time-series Private Stream Aggregation with Noisy Data

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 [15]. Secure Multiparty Computation (MPC) protocols are used to allow a set of parties to securely compute a function over their inputs [16], [17]. 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 [7], [8], [18]. 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.

Later work in secure aggregation worked towards constructing aggregation schemes secure against quantum adversaries [19], [20], [21]. The state-of-the-art LaPS scheme [8] achieves additive aggregation with quantum security by utilizing lattice-based cryptography. 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.

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 [22], [23]. To create these schemes, we take a white-box approach and design lattice-based PSA protocols comprised directly of ring operations, without using other HE cryptosystems as primitives. We take advantage of the trusted setup model of PSA schemes to create ciphertexts that possess additive homomorphism with correct decryption when (and only when) all parties' ciphertexts are aggregated. Further, our schemes allow for security through multiple aggregations without key redistribution, fixing weaknesses in previous work [8].

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 can use differential privacy similarly to prior work. We show that our optimized implementation can improve upon previous work 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.

Our Contributions

- We present SLAP, a novel lattice-based PSA scheme with two variations, built using a white-box approach. SLAP draws its ideas from the RLWE difficulty assumption [22], granting it quantum security. By taking this approach, we can construct lattice-based schemes optimally tailored to the application of PSA. Both variants are simpler and more efficient than previous related protocols, 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 99.96% as compared to other lattice-based PSA. We show that SLAP is an aggregator oblivious PSA scheme, and can attain differential privacy.
- We further integrate the NTT, full-RNS variants, 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.
- Both implementations (basic and fully optimized) of our scheme are made available (anonymously) as open-source code.

Organization In Section 2, we discuss related work in PSA, lattice-based cryptography, 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

2.1 Pre-quantum PSA

The seminal works of Shi et al. [7] and Rastogi et al. [24] introduced the concepts of PSA and aggregator obliviousness, as well as presenting schemes based upon Diffie-Hellman and Fourier perturbation with differential privacy. The work of Joye et al. [9] expanded upon this work by improving the limited plaintext space size, using the hardness assumption of Decisional Composite Residuosity. Erkin et al. [25] propose a PSA-like framework for aggregation of time- and space-series data (intended for use in smart metering), which utilizes the Pallier cryptosystem. Protocols for secure distributed polynomial computations based in the difficulty of the discrete logarithm problem have also been formulated [10], [26], 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 [27].

PSA schemes using secret sharing [6], [28] do not have the same weakness to quantum adversaries. However, these protocols have requirements including multiple rounds of communication and multiple trusted parties that hinder their scalability.

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 [19], [20], [21]. 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 pseudo-random functions can be used to construct quantum-secure PSA protocols such as the leading LaPS protocol without the requirement of a trusted third party [8], [29]. However, such protocols have extremely complex decryption procedures or other weaknesses. SLAP is much conceptually simpler and more efficient than other lattice-based PSA schemes.

2.3 Reliability and Fault-Tolerance in PSA

Related research in PSA generally does not explore issues in robustness such as fault tolerance, reliability, input poisoning, or fully malicious participants who might craft nonsensical messages or even refuse to communicate. Instead, PSA focuses on the theoretical construction of protocols more than practical concerns [7], [8]. Other lines of research do provide solutions for fault tolerance, dynamic join/leave, maliciously chosen inputs, and error detection [30], [31], [32], [33], [34].

2.4 Algorithmic optimization in lattice-based encryption

Lattice-based quantum-secure cryptographic schemes such as the fully homomorphic encryption schemes BGV [13], B/FV [14], and CKKS [35] schemes deal with ring polynomials that are large in two senses: both their degree and coefficients may be large, with polynomial degrees on order of $N = 2^{15}$ and coefficients that are hundreds of bits wide. Large polynomial degrees may make polynomial multiplication highly computationally intensive; to mitigate this the Number-Theoretic Transform (NTT) can be used to decrease the theoretical complexity of polynomial multiplication [36]. For large coefficients, Residue Number System (RNS) representations can be used to break these large numbers down into smaller, more manageable components. Full-RNS variants of lattice-based cryptosystems have been used to reduce the complexity of those cryptosystems' most intensive operations to the complexity of the NTT [37], [38], [39]. Both the NTT and full-RNS variants bring a great practical improvement to these cryptosystems. We apply both of these optimizations in our work, and discuss them in more detail in Section 6.

2.5 Differential Privacy

Differential Privacy refers to the technique of adding specially-distributed noise to data to mask the exact values [24], [29], [40]. It is commonly used in work in aggregation to hide the exact result of a computation [7], [8], [34], [41]. 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

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 $\mathbf{Z}_q = [\frac{-q}{2}, \frac{q}{2})$. 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 elements rings. 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 [14], [22]. 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. 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, due to a reduction from the Shortest Vector Problem [22] to RLWE. 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 [42], [43], [44], and mathematically convenient for some schemes.

3.3 Differential Privacy and Computational Differential Privacy

We recall relevant definitions from differential privacy [7], [8], [24], [29] 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 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 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. *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 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 $x \in \mathbb{Z}$.*

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

3.4 Private Stream Aggregation and Aggregator Obliviousness

In secure aggregation, we usually consider the case of a set of n users and a single untrusted aggregator. Semi-honest (also known as honest-but-curious) adversaries may carry out arbitrary additional computation to try to learn other parties' private data, but will execute the protocol faithfully without giving malicious or malformed input. We consider semi-honest parties (including users and the aggregator) for PSA and aggregator obliviousness. Each user i possesses 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 are designed to allow a third party (referred to as the aggregator) to perform this computation while providing semantic security to the time-series data the aggregator receives from other users. Further, 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. [7] 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 will run $NoisyEnc$ on their data (once per timestamp), and send their results $c_{i,ts}$ to the aggregator. The aggregator can then call Agg on the ciphertexts $c_{0,ts}, \cdots c_{n-1,ts}$ it has collected from users 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 [7], [8], [9]. 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 is more efficient in communication than most existing MPC protocols [17], [29]. 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 wish to 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 5 [7], [8]:

Definition 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 [7], [8] 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 ith 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 $c_{i,ts} = NoisyEnc(parms, s_i, ts, \tilde{x}_{i,ts}, \tilde{r}_{i,ts})$ 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 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. Such vulnerabilities are inherent in scenarios with powerful adversaries; thus aggregator obliviousness is constructed to require that no *additional* information is learned in such a case, as discussed at length in [7]. For example, there is the situation where an adversary compromising the aggregator and all but one of the users may be able to learn the last user's private input; aggregator obliviousness accounts for this by requiring that in such cases of unavoidable vulnerabilities, that the protocol not leak any additional information.

Issues not Addressed in PSA PSA schemes, including this work, are primarily concerned with efficient and functional constructions of the actual aggregation protocol [6], [7], [8], [28]. Differential Privacy techniques are used in PSA, but are considered to be orthogonal to the aggregation. 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 [30], [31], [32], [33], [34]. In particular, PSA assumes 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 Schemes

We first present our PSA schemes without specifying the details of added noise, and discuss the use of differentially private noise in Section 5. We present two variants 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 FHE has been used in previous lattice-based PSA as a black-box building block [8], 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 LPR and NTRU cryptosystems [22], [23], adapted to the specific application of PSA. This allows for a more efficient application of the solution for the specific purpose of PSA. One example of this is that ciphertexts in SLAP are only a single element of R_q, while ciphertexts in RLWE FHE schemes are tuples of elements of R_q [13], [14]. 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 [8]. However, doing 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. SLAP thus uses a random oracle to prevent this attack [45]. In practice, this can be instantiated by distributing hash function keys among users.

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 [23] 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 $log_2(3)+log_2(n)+log_2(t) < log_2(q)$ and q, t are coprime. Choose the polynomial modulus N such that λ bits of security are provided for the RLWE problem with ring polynomial coefficients in $\mathbb{Z}_q$ [46], [47]. (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 [45]. Return $parms = (R_q, t, n, H)$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $NoisyEnc_{NS}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, \mathbf{r}_{i,ts} \in R_t, ts)$: 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 Let all noise values $\mathbf{r}_{i,ts} = 0$, as it can be assumed for this discussion that the noise has already been added to user input. 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 $\frac{3 \cdot n \cdot t}{2} < \frac{q}{2}$, $\sum_{i=0}^{n-1} (t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts})$ 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, 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 [22] and B/FV [14] 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). 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 $log_2(3) + log_2(n) + 2 \cdot log_2(t) < log_2(q)$. (Note that these bounds are stricter than those 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 $\mathbb{Z}_q$ [46], [47]. 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 [45]. Return $parms = (R_q, t, n, \{\mathbf{A}_{ts}\})$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $NoisyEnc_{MS}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, \mathbf{r}_{i,ts} \in R_t, ts)$: 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 $\mathbf{r}_{i,ts} = 0$, as it can be assumed for this discussion that the noise has already been added to user input. 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 $\frac{t}{q} \cdot || \sum_{i=0}^{n-1} \mathbf{e}_{i,ts} - \frac{q \mod t}{t} \sum_{i=0}^{n-1} \mathbf{x}_{i,ts} || < \frac{q}{2}$ [14]. 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 Semantic Security

The semantic security of the ciphertexts $\mathbf{c}_i$ generated in $SLAP_{NS}$ and $SLAP_{MS}$ follows trivially from the RLWE problem: the terms $\mathbf{A}_{ts} \cdot \mathbf{s}_i + \mathbf{e}_{i,ts}$ and $\mathbf{A}_{ts} \cdot \mathbf{s}_i + t \cdot \mathbf{e}_{i,ts}$ are RLWE terms, indistinguishable from random by the randomness (or indistinguishability thereof) of the random oracle hash from which $\mathbf{A}_{ts}$ is derived. See the extended version of [22] for more details on the security of RLWE. Note that ciphertexts at different times from the same user use different $\mathbf{A}_{ts}$ and $\mathbf{e}_{i,ts}$, so security across multiple executions is given.

Informally, suppose for contradiction that the ciphertexts $\mathbf{c}_{i,ts}$ in either variant of SLAP were not indistinguishable from random, and that there existed some probabilistic polynomial-time (PPT) distinguisher algorithms $\mathcal{D}$. Then, we could use this to construct an attacking algorithm $\mathcal{D}'$ against the pseudorandomness of RLWE terms. Given a term $\mathbf{b}$ that is either an RLWE term or random, an adversary $\mathcal{D}'$ can choose a message $\mathbf{x}$ and call $\mathcal{D}$ on $\mathbf{b}+\mathbf{x}$. If $\mathcal{D}$ indicates that this is a SLAP ciphertext, then $\mathcal{D}'$ can conclude that $\mathbf{b}$ was a RLWE term. On the other hand, if $\mathcal{D}$ indicates that its input is random, then $\mathcal{D}'$ can conclude that it was given a random value. Thus $\mathcal{D}'$ would be able to break the pseudorandomness of RLWE. This shows a reduction from RLWE to SLAP.

4.4 Benefits and Novelty of SLAP

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

- **Simplicity:** The theory underlying SLAP is vastly simpler than prior state-of-the-art work in quantum-proof PSA [8], with straightforward operations and parameter requirements. This is due to our white-box approach, which directly applies lattice-based homomorphism instead of relying on preexisting fully homomorphic encryption schemes. SLAP is thus easier and more lightweight to implement and deploy, making it more practically useful. This simplicity also leads to very straightforward proofs of security and privacy.

- **Smaller Ciphertexts:** As shown in Sections 7.1 and 7.3, SLAP has smaller ciphertexts as compared to previous work [8], with reductions in communication overhead of over 99% for larger parameter settings. The lighter communication load of SLAP make it more practical for deployment among users with less computing power, e.g., smartphones and IoT nodes.
- **Extensibility:** Thanks to the simplicity of SLAP, it is easy to apply other work in lattice-based cryptography, including the common optimizations of RNS, NTT, and batching (discussed in Section 6), and the complex canonical embedding used in the CKKS scheme [35] to allow approximate floating-point computation.
- **Noninteractivity:** SLAP has no need for repeated distribution of public values or secret keys, and after setup has occurred, no user needs to perform any interaction with the aggregator or any other party beyond sending their ciphertext to the aggregator. This setup provides security across multiple iterations of aggregation, improving on previous work that did not provide this security guarantee [8].
- **Efficiency and Throughput:** As seen in Section 7, SLAP can outperform optimized implementations of previous quantum-proof PSA by an order of magnitude, and the throughtput is further improved when batching is considered. Even our unoptimized implementation of SLAP showed comparable performance to the LaPS scheme.

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 [7], [8], [41]; 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 1. The PSA schemes $SLAP_{NS}$ and $SLAP_{MS}$ using this mechanism of noisy encryption achieve differential privacy and aggregator obliviousness:

Theorem 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 [8], Theorem 3, which itself is from [7], 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. The term α is approximately $O(\frac{\Delta}{\epsilon}\sqrt{n})$, so this approach to differential privacy is most effective with a smaller number of users whose inputs have a low variance.

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). This proof is very similar to existing PSA proofs [7], [8] that use aggregator obliviousness as their security notion, and we adapt these existing techniques for our own protocol.

Theorem 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.*

Proof. We follow previous work [7], [8], [41] 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 distinguish between semantically secure RLWE ciphertexts in either variant of SLAP. 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). Throughout this proof, we write

A Slightly Modified Game: In this proof, we modify the game of aggregator obliviousness defined in Section 3.4 (similarly to LaPS and Shi et al.) as follows [7], [8]: First, we change any Encrypt query as a Compromise query from the adversary (which actually strengthens the adversary), and we change the Challenge phase to a real-or-random version. Second, in the original game of aggregator obliviousness, the adversary is asked to specify two sets of plaintext/randomness pairs $(\mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}), (\tilde{\mathbf{x}}_{i,ts}, \tilde{\mathbf{r}}_{i,ts})$ and then to distinguish between encryptions of either of the pairs. In this proof, we let the adversary choose one pair $(\mathbf{x}_{i,ts}, \mathbf{r}_{i,ts})$ and have them distinguish between valid encryptions of $(\mathbf{x}_{i,ts}, \mathbf{r}_{i,ts})$ or random values. Any adversary with more than negligible advantage in winning this modified game will also be able to win the original game of aggregator obliviousness aggregator obliviousness security game with more than negligible advantage. Therefore, it is enough to show that with any PPT adversary $\mathcal{A}$ with a greater than negligible advantage in winning the modified game can be used to construct an algorithm $\mathcal{B}$ that can distinguish

RLWE ciphertexts from random, thus solving the decisional version of RLWE. As noted by Shi et al. [7], when the aggregator has not been compromised, real-or-random security suffices, and if the aggregator has been compromised, then aggregator obliviousness shows that a challenger cannot distinguish between an argued plaintext and a randomly chosen argument with the same sum. This modification allows us to use an adversary that can break the security of SLAP as an attacker against the Decision version of RLWE.

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.

Reducing to Semantic Security: First, we define a game for $\mathcal{B}$ that indicates $\mathcal{B}$'s ability to break the semantic security of RLWE ciphertexts. Suppose $\mathcal{B}$ receives the parameters (R_q, t, n). Then with a challenger $\mathcal{C}$ testing the ability of $\mathcal{B}$ to break the indistinguishability from random of RLWE terms, $\mathcal{B}$ will play the modified game described above. In this, $\mathcal{B}$ can make Sample queries by arguing $(\mathbf{m}_{ts} \in R_t, ts)$ to $\mathcal{C}$ and will receive back the pair $(\mathbf{A}_{ts}, \mathbf{M}_{ts})$, where $\mathbf{A}_{ts}$ is a publicly known element of R_q and $\mathbf{M}_{ts}$ is an encryption under the secret key $\mathbf{s}^*$ of $\mathbf{m}_{ts}$ at the time ts. Then, $\mathcal{B}$ sends $(\mathbf{m}^*_{ts} \in R_t, ts)$ to $\mathcal{C}$ (with ts not previously argued, as required in the encrypt-once model). Based on a random bit b chosen by $\mathcal{C}$, $\mathcal{C}$ will choose $\mathbf{M}^*_{ts}$ either as an encryption of $\mathbf{m}^*_{ts}$ (if $b = 0$) or a random element of R_q (if $b = 1$). Then $\mathcal{B}$ must guess the value of b, winning if correct.

Reduction: We now show how $\mathcal{B}$ can simulate the modified game of aggregator obliviousness to $\mathcal{A}$. In the Setup phase, $\mathcal{B}$ will first choose distinct $j, k \in [0, n) \cup \{\square\}$. (With probability $\frac{1}{n^2}$, $\mathcal{A}$ will not select these parties to be compromised.) Then, $\mathcal{B}$ implicitly sets $\mathbf{s}_k = \mathbf{s}^*$, chooses secret keys $\mathbf{s}_i$ for all $i \neq j, k$, and implicitly sets $\mathbf{s}_j = [-(\sum_{i \neq j,k} \mathbf{s}_i) - \mathbf{s}_k]_q$. (Note that $\mathcal{B}$ does not actually know either of $\mathbf{s}_k, \mathbf{s}_j$, which are the aggregation scheme's secret keys for users j, k.) Finally, $\mathcal{B}$ chooses the aggregation key $\mathbf{s}'$ randomly from $\{\mathbf{s}_i\}_{i \in [0,n)} \setminus \{\mathbf{s}_j, \mathbf{s}_k\}$.

In a Compromise query, $\mathcal{A}$ will send a query for a user i to $\mathcal{B}$. If $i \notin \{j, k\}$, then $\mathcal{B}$ returns $\mathbf{s}_i$ to $\mathcal{A}$. Also, if $i = \square$, then $\mathbf{s}'$ will be returned to $\mathcal{A}$.

In a Challenge query, $\mathcal{A}$ will choose a set of uncompromised users $U \subseteq [0, n) \setminus \{j, k\}$. They will then send plaintext-randomness tuples $\{(\mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}, ts)\}$ with $i \in U$ to $\mathcal{B}$. Because we chose earlier to abort if a query was for either of j, k, we know that $j, k \in U$. Then, $\mathcal{B}$ computes $\mathbf{c}_i = NoisyEnc\,(parms, \mathbf{s}_i, \mathbf{x}_{i,ts}, \mathbf{r}_{i,ts}, ts)$ for $i \in U \setminus \{j, k\}$.

Now $\mathcal{B}$ will attempt to guess the bit b. It sends $\mathbf{m}_{k,ts} = [\mathbf{x}_{k,ts} + \mathbf{r}_{k,ts}]_q$ and ts to $\mathcal{C}$ who returns the tuple $(\mathbf{A}_{ts}, \mathbf{M}^*_{ts})$. Then, $\mathcal{B}$ sets $\mathbf{c}_{k,ts} := \mathbf{M}^*_{ts}$. $\mathcal{B}$ then computes an encryption of the sum of the plaintext-random pairs, with $\mathbf{v}_{ts} = [\sum_{i \in U} \mathbf{x}_{i,ts} + \mathbf{r}_{i,ts}]_q$ and $\mathbf{c}_{j,ts} = NoisyEnc(parms, \mathbf{A}_{ts}, \mathbf{x}_{j,ts}, \mathbf{v}_{ts})$. Now $\mathcal{B}$ has $\mathbf{c}_{i,ts}$ for $i \in U$, including $\mathbf{c}_{j,ts}$ and $\mathbf{c}_{k,ts}$. $\mathcal{B}$ then returns these values $\mathbf{c}_{i,ts}$ to $\mathcal{A}$.

We now move to the Guess phase. If $\mathcal{A}$ has more than negligible advantage in winning the aggregator obliviousness security game, they can distinguish the ciphertexts from random. Specifically, if $\mathbf{c}_{k,ts} = \mathbf{M}^*_{ts}$ is a valid encryption of $\mathbf{x}_{k,ts} + \mathbf{r}_{k,ts}$, i.e., $\mathbf{M}^*_{ts} = \mathbf{M}_{ts}$, then $\mathcal{A}$ will return 0, otherwise they return 1. Therefore, by forwarding $\mathcal{A}$'s output to $\mathcal{C}$ as their guess, $\mathcal{B}$ wins the game, and they can distinguish $\mathbf{M}_{ts}$ from random and break the semantic security of the RLWE ciphertexts. This completes the proof by reduction. □

6 Performance Optimizations and Practical Considerations

In this section, we discuss various practical optimizations common to lattice-based cryptography that can be utilized to accelerate SLAP. The simple and limited lattice operations of SLAP allow these optimizations to be directly "plugged-in" to SLAP with no modifications required. Further, the purely additive nature of SLAP allows for a simpler method of batching (though the common one can still be used). We also discuss parameter choices for different scenarios.

6.1 RNS Variants

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, 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 [37], [39]. The full-RNS variant using floating-point operations [39] is much simpler and as efficient as the full-RNS variant using integer-only operations [37]. We thus adapt the integer-only RNS variant [39] 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 [39]. 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 [39]. 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 $\frac{t \cdot q_i}{q_i} = \omega_i + \theta_i$ 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 $[\omega + v]_t$.

6.2 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 [36]. 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 [36]. 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), all moduli q_i must satisfy this condition. In practice, these moduli are often precomputed as prime values.

6.3 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) [48]. The common method of batching could also be used, though encoding and decoding is more complex.

Another method of batching is to use RNS decomposition (see Section 6.1) 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 affect the allowable range of users' inputs and the number of users for a given ciphertext modulus, as discussed in Section 6.4.

Besides the above methods of batching, our scheme is also amenable to the application of a complex canonical embedding [35] that can be used to encode an array of floating-point values into a plaintext element of R_t, and perform floating-point aggregation.

6.4 Practical Parameter Selection

In the event that users wish to compute aggregation without overflow, we suppose users have arguments that are in $\mathbb{Z}_w$ where $\frac{w}{2} < n \cdot \frac{t}{2}$, so that a computation can correctly sum users' values without wrapping modulo t. Suppose we decompose t into k' RNS moduli as in Sections 6.1 and 6.3, the smallest of which is $\tilde{t}$. ($k' = 1$ without packing.) Suppose there are n users, with input as large as $\frac{w}{2}$ (so that all inputs are in an interval of width w). Then to avoid wrapping from modular reduction, we require that $n \cdot w/2 \leq \tilde{t} \approx \frac{t}{k'}$. This can be written as $k' \leq \frac{t}{n \cdot w}$, $n \leq \frac{t}{k' \cdot w}$, $w \leq \frac{t}{k' \cdot n}$, or $w \cdot k' \cdot n \leq t$, allowing one to choose the number of packed values, users, input size, or plaintext modulus size based on the values of the other variables. This analysis is made for the case of exact aggregation without any noise. When noise is added, parameter selection should take the possible addition of that noise into account.

7 Experimental Evaluation

In this section, we present experimental evaluations of our work. The LaPS scheme [8] is the most closely related work to ours, so we compare SLAP against LaPS to best understand the improvements gained in performance and communication. In our experimental evaluations, we consider the same parameters for differential privacy as LaPS, i.e., $\epsilon = 1$, $\delta = 0.1$, 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 $\mathcal{O}(w\sqrt{N})$). 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.

7.1 Example Parameters

Table 1 shows minimal parameter choices to guarantee correctness and 128-bit security. 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 (q_0) [8]. Also, our required polynomial modulus degree is smaller. This continues the improvement of smaller parameters that LaPS presented over related work [49], and shows that SLAP can be more efficient than previous work in communication overhead.

TABLE 1
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 2
Ciphertext Size Comparison of LaPS and $SLAP_{NS}$ (128 bits 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×

TABLE 3
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 4
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

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, both utilizing the differentially private noise mechanism of Section 5.1. In the optimized implementation, the full-RNS variant presented in Section 6.1 is implemented, and the Number-Theoretic Transform is used to accelerate polynomial multiplication (as described in Section 6.2). The basic implementation does not integrate these optimizations, but uses NTL [50] for integer and polynomial arithmetic. Our implementations (in C++14) are published at https://gitlab.com/jtakeshi/lattices/. The code is single-threaded, though SLAP and our implementation can be easily 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. The computer had 128GB memory. We use the standard ciphertext modulus and polynomial modulus specifications from HomomorphicEncryption.org [46]. Our tests took average runtimes over 50 trials of the operation in question.

7.3 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 [8], [51].

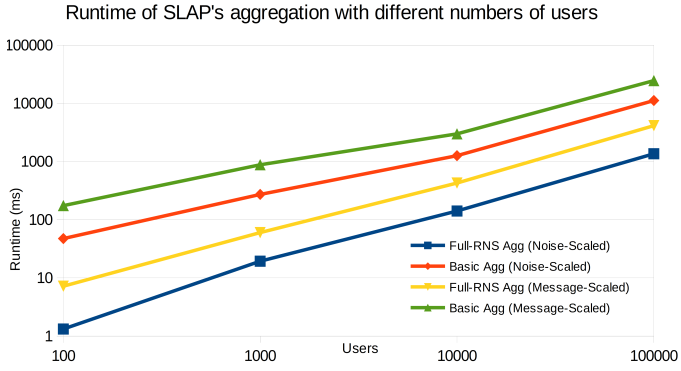

Fig. 3. Scalability with a 32-bit plaintext space

We first compare SLAP directly to LaPS, matching the largest set of parameters they considered. LaPS is implemented using the HElib homomorphic encryption library [52], 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) 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. Table 3 shows the performance of SLAP with these parameters, along with the time it takes to do the computation in plaintext. Table 4 shows

the speedup of SLAP as compared to LaPS. 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 2) 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. Table 2 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, it is assumed that ciphertext moduli are stored in 64-bit words - if 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.

As a practical example, Table 9 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.

We can also 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.

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 6.3, 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 5 and 6. 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 5 and 6, 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 6.3, encryption time, network latency, and aggregation time. We show the results of this analysis in Tables 7 and 8. 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.

7.5 Comparison to Other PSA

Early work by Shi et al. [7] 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 [21] and [19] are highly similar; both are lattice-based PSA-like work specifically tailored to the scenario of smart meters, which is different from general PSA. [21] fixes some security holes in [19], 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. Our experimental comparison to the state-of-the-art lattice-based PSA scheme LaPS [8] shows that SLAP is more efficient as compared to previous work, and would require less overhead in communication due to smaller ciphertexts. This is shown in Tables 3, 4 and 9. In [20], the runtime of

TABLE 5
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 6
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 7
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 8
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 9
Ciphertext Upload Time (seconds) of LAPS and $SLAP_{NS}$ (128 bits 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

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 [7], [9], [10], [11] 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 10, 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. We conclude that SLAP brings both theoretical and practical improvements to the current state-of-the-art in lattice-based PSA.

References

[1] D. Archer, L. Chen, J. H. Cheon, R. Gilad-Bachrach, R. A. Hallman, Z. Huang, X. Jiang, R. Kumaresan, B. A. Malin, H. Sofia *et al.*, "Applications of homomorphic encryption," *HomomorphicEncryption. org, Redmond WA, Tech. Rep.*, 2017.

[2] "Role of applied cryptography in a privacy-focused advertising ecosystem request for proposals," Jan 2020. [Online]. Available: https://research.fb.com/programs/research-awards/proposals/cryptography-rfp-2019/

[3] "Sony research award program." [Online]. Available: https://www.sony.com/electronics/research-award-program#FacultyInnovationAward

[4] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Apr 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

TABLE 10
Runtime (μs) of Schemes without Quantum Security (2048-bit plaintext space)

Operation	Joye agg. [9]	Joye agg. (per user)	Jung 2013 agg. (per user) [10]	Jung PDA agg. (per user) [11]	Shi agg. (per user) [7]
Time (μs)	3563.46	35.79	32.02	12.83	0.16
Operation	Joye enc.	Jung 2013 enc.	Jung PDA enc. (per user)	Jung PDA enc. of Special User 2 (per user)	Shi enc.
Time (μs)	2.07	32.02	0.06	1776.15	491.20

[6] G. Danezis, C. Fournet, M. Kohlweiss, and S. Zanella-Béguelin, "Smart meter aggregation via secret-sharing," in *Proceedings of the first ACM workshop on Smart energy grid security*, 2013, pp. 75–80.

[7] E. Shi, T. H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Proc. NDSS*, vol. 2, 2011, pp. 1–17.

[8] D. Becker, J. Guajardo, and K.-H. Zimmermann, "Revisiting Private Stream Aggregation: Lattice-Based PSA," in *NDSS*, 2018.

[9] M. Joye and B. Libert, "A scalable scheme for privacy-preserving aggregation of time-series data," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 111–125.

[10] T. Jung, X. Mao, X.-Y. Li, S.-J. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2634–2642.

[11] T. Jung, J. Han, and X.-Y. Li, "PDA: semantically secure time-series data analytics with dynamic user groups," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, pp. 260–274, 2016.

[12] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–33, 2017.

[13] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[14] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[15] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "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*, 2018, pp. 1–8.

[16] Y. Yang, X. Huang, X. Liu, H. Cheng, J. Weng, X. Luo, and V. Chang, "A comprehensive survey on secure outsourced computation and its applications," *IEEE Access*, vol. 7, pp. 159 426–159 465, 2019.

[17] R. Karl, T. Burchfield, J. Takeshita, and T. Jung, "Non-interactive MPC with trusted hardware secure against residual function attacks," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2019, pp. 425–439.

[18] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[19] A. Abdallah and X. S. Shen, "A lightweight lattice-based homomorphic privacy-preserving data aggregation scheme for smart grid," *IEEE Transactions on Smart Grid*, vol. 9, no. 1, pp. 396–405, 2016.

[20] A. A. Agarkar, M. Karyakarte, and H. Agrawal, "Post quantum security solution for data aggregation in wireless sensor networks," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–8.

[21] R. B. Romdhane, H. Hammami, M. Hamdi, and T.-H. Kim, "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, 2019, pp. 1067–1072.

[22] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM (JACM)*, vol. 60, no. 6, pp. 1–35, 2013.

[23] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International Algorithmic Number Theory Symposium*. Springer, 1998, pp. 267–288.

[24] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 735–746.

[25] Z. Erkin and G. Tsudik, "Private computation of spatial and temporal power consumption with smart meters," in *International Conference on Applied Cryptography and Network Security*. Springer, 2012, pp. 561–577.

[26] T. Jung, X.-Y. Li, and M. Wan, "Collusion-tolerable privacy-preserving sum and product calculation without secure channel," *IEEE Transactions on Dependable and secure computing*, vol. 12, no. 1, pp. 45–57, 2014.

[27] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[28] J. Bell, K. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," *IACR Cryptol. ePrint Arch*, 2020.

[29] F. Valovich and F. Aldà, "Computational differential privacy from lattice-based cryptography," in *International Conference on Number-Theoretic Methods in Cryptology*. Springer, 2017, pp. 121–141.

[30] J. Won, C. Y. Ma, D. K. Yau, and N. S. Rao, "Proactive fault-tolerant aggregation protocol for privacy-assured smart metering," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 2804–2812.

[31] K. Xue, Q. Yang, S. Li, D. S. Wei, M. Peng, I. Memon, and P. Hong, "PPSO: A privacy-preserving service outsourcing scheme for real-time pricing demand response in smart grid," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2486–2496, 2018.

[32] R. Karl, J. Takeshita, and T. Jung, "Cryptonite: A framework for flexible time-series secure aggregation with online fault tolerance," Cryptology ePrint Archive, Report 2020/1561, 2020, https://eprint.iacr.org/2020/1561.

[33] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure information aggregation in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 255–265.

[34] Q. Li and G. Cao, "Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2013, pp. 60–81.

[35] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[36] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *International Conference on Cryptology and Network Security*. Springer, 2016, pp. 124–139.

[37] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 423–442.

[38] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 347–368.

[39] S. Halevi, Y. Polyakov, and V. Shoup, "An improved RNS variant of the BFV homomorphic encryption scheme," in *Cryptographers' Track at the RSA Conference*. Springer, 2019, pp. 83–105.

[40] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 429–438.

[41] G. Ács and C. Castelluccia, "I have a dream!(differentially private smart metering)," in *International Workshop on Information Hiding*. Springer, 2011, pp. 118–132.

[42] I. Blanco-Chacón, "On the RLWE/PLWE equivalence for cyclotomic number fields," *Applicable Algebra in Engineering, Communication and Computing*, pp. 1–19, 2020.

[43] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Annual cryptology conference*. Springer, 2011, pp. 505–524.

[44] M. Rosca, D. Stehlé, and A. Wallet, "On the ring-LWE and polynomial-LWE problems," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 146–173.

[45] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, 2017.

[46] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.
[47] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer, "Estimate all the {LWE, NTRU} schemes!" in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 351–367.
[48] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Annual Cryptology Conference*. Springer, 2012, pp. 850–867.
[49] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.
[50] V. Shoup *et al.*, "NTL: A library for doing number theory," 2001.
[51] M. Crane, A. Trotman, and R. O'Keefe, "Maintaining discriminatory power in quantized indexes," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 1221–1224.
[52] S. Halevi and V. Shoup, "Helib," *Retrieved from HELib: https://github.com/shaih/HElib*, 2014.



Jonathan Takeshita is a Ph.D. student in the Department of Computer Science and Engineering at The University of Notre Dame. He holds a B.S.E. in Computer Science from The University of Michigan - Ann Arbor and a B.A. in Engineering Physics and Music from Albion College. His recent research includes work on lattice-based cryptography, secure deduplication, hardware acceleration of fully homomorphic encryption, and privacy-preserving contact tracing.



Ryan Karl is a 4th year graduate student in the Department of Computer Science and Engineering at The University of Notre Dame. He is broadly interested in cryptography and computer security, and focuses on leveraging trusted computing and cryptography to provide practical solutions to problems in privacy-preserving data collection and analysis. He holds a M.S. in Computer Science and Engineering from The University of Notre Dame and a B.S. in Mathematics from Saint Vincent College.



Ting Gong is a graduating undergraduate student majoring in mathematics and philosophy at The University of Notre Dame. His research interests include number theory, arithmetic geometry and cryptography.



Taeho Jung is an assistant professor of Computer Science and Engineering at The University of Notre Dame. He received his Ph.D. from Illinois Institute of Technology in 2017 and B.E. from Tsinghua University in 2011. His research area includes data security, user privacy, and applied cryptography. His paper has won a best paper award (IEEE IPCCC 2014), and two of his papers were selected as best paper candidate (ACM MobiHoc 2014) and best paper award runner up (BigCom 2015).