SLAP: Simple Lattice-Based Private Stream Aggregation Protocol

Jonathan Takeshita, Ryan Karl, Ting Gong, and Taeho Jung

University of Notre Dame, Notre Dame IN 46556, USA
{jtakeshi,rkarl,tgong,tjung}@nd.edu

Abstract. Today, users' data is gathered and analyzed on a massive scale. While user data analytics such as personalized advertisement need to make use of this data, users may not wish to divulge their information without security and privacy guarantees. Private Stream Aggregation (PSA) allows the secure aggregation of time-series data, affording security and privacy to users' private data, which is much more efficient 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 lattice-based PSA protocol. SLAP features two variants with post-quantum security, with simpler and more efficient computations enabled by (1) the white-box approach that builds the encryption directly from the Ring Learning With Error assumption and (2) the state-of-the-art algorithmic optimization in lattice-based cryptography. We show that SLAP meets the security and privacy requirements of PSA, 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 over 600,000 aggregations per second.

Keywords: 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, the data privacy of individual data holders is paramount. Examples of this include health care and education, where an individual's privacy is often protected by law [5]. It is thus desirable to consider schemes that allow data aggregation to be computed without violating users' privacy.

Simply put, the problem we wish to solve 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? In this scenario, both the security and privacy of each data element should be protected. Security guarantees should prevent eavesdroppers from seeing the original data. This 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. A simple solution is to simply use a trusted third party to collect and aggregate users' data; however this is a very strong assumption that we wish to eliminate.

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 [28, 9, 17]. However, there are many issues with using existing HE schemes as-is. For example, key management is nontrivial: an aggregating party must be trusted to not decrypt ciphertexts pre-aggregation, or the duties of aggregation and decryption should be separated between two servers. Further, HE schemes are mathematically complex and computationally intensive, making them unattractive to clients who want simple, lightweight solutions to the less general problem of aggregation. 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 [29]. Secure Multiparty Computation (MPC) protocols are used to allow a set of parties to securely compute a function over their inputs [36, 25]. 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.

The shortcomings of more general approaches have led to the creation of the idea of Private Stream Aggregation (PSA), which is the secure and private aggregation of time-series data. As originally introduced by Shi et al. [31], PSA encompassed security and privacy with the idea of *aggregator obliviousness*. Aggregator obliviousness captures security and privacy, requiring that no adversary can learn the values of a target's data, even when compromising all other participating parties. Later work in secure aggregation worked towards constructing aggregation schemes secure against quantum adversaries. The state-of-the-art LaPS scheme [7] achieves additive aggregation with quantum security by utilizing lattice-based cryptography. However, the LaPS scheme has several drawbacks: it introduces needless complexity and overhead by relying upon fully homomorphic encryption as a primitive operation. It also relies on the BGV cryptosystem 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 work 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 underlying principles of HE schemes and lattice-based hardness assumptions. 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. 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. This is possible through an additive correlation of users' secret keys and the aggregation key. 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. 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 812,396 aggregations per second. We also show that our schemes are scalable, and more efficient in communication overhead than previous work.

Our Contributions

- We present SLAP, a PSA scheme with two variations, built using a white-box approach. SLAP draws its ideas from the core ideas of the BGV [9] and B/FV [17] fully homomorphic encryption schemes, and the NTRU [21] and LPR [27] difficulty assumptions. By taking this approach, we can create novel lattice-based schemes optimally tailored to the application of PSA. Both variants are simpler and more efficient than previous related protocols.
- 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 the use of 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 628,605 aggregations per second) 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.

2 Related Work

2.1 Pre-quantum PSA

The seminal work of Shi et al. [31] and Rastogi et al. [16] introduced the concepts of PSA and aggregator obliviousness, as well as presenting a Diffie-Hellman-based scheme with differential privacy. The work of Joye et al. [22] expanded upon this work by improving the limited plaintext space size, using the hardness assumption of Decisional Composite Residuosity. Erkin et al. [15] 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 [24, 23], 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 [32].

PSA schemes using secret sharing [14, 8] do not have the same weakness to quantum adversaries. However, these schemes suffer from a severe lack of scalability: the computation required by each user increases linearly with the number of users, which is clearly untenable for cases with millions of users.

2.2 Post-quantum PSA

There exists some work in PSA-like protocols relying on lattice-based cryptography, which is quantum-proof [1, 2, 30]. However, these works have the disadvantage of 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 without the requirement of a trusted third party, but such protocols have extremely complex decryption procedures or other weaknesses [7, 35]. SLAP is much conceptually simpler and more efficient than other lattice-based PSA schemes. Further, we integrate state-of-the-art algorithmic optimizations of lattice-based cryptography (SIMD, RNS) into SLAP to further enhance the efficiency and throughput.

2.3 Algorithmic optimization in lattice-based encryption

Lattice-based quantum-secure fully homomorphic encryption schemes such as the BGV [9], B/FV [17], and CKKS [11] 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 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 [26]. 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 [6, 10, 19]. 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 5.

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 $\mathbb{Z}_t = [\frac{-t}{2}, \frac{t}{2}) \cap \mathbb{Z}$. 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

Consider coprime numbers q, p, with $q \gg p$, and let $\mathbf{s}$ be a random element of R_q with coefficients bounded by b. Let $\mathbf{a}_i, \mathbf{e}_i$ be a polynomially bounded number of elements of R_q, with $\mathbf{a}_i$ chosen randomly and $\mathbf{e}_i$ random and also b-bounded. An adversary is given the set of pairs $(\mathbf{a}_i, \mathbf{b}_i) \in R_q^2$. Unknown to the adversary is whether $(\mathbf{a}_i, \mathbf{b}_i)$ are RLWE terms, i.e. $\mathbf{b}_i = [\mathbf{a}_i \cdot \mathbf{s}_i + p\mathbf{e}_i]_q$, 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; its difficulty comes from reduction to the Shortest Vector Problem [27].

3.3 Differential Privacy and Computational Differential Privacy

We recall relevant definitions from differential privacy 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, $Pr[M(D_0) \in U] \leq exp(\epsilon) \cdot Pr[M(D_1)] + \delta$. (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 $x \in \mathbb{Z}$.*

3.4 Private Stream Aggregation and Aggregator Obliviousness

In secure aggregation, we consider the case of a set of n users and a single untrusted aggregator. Each user i possesses a piece of data x_i, corresponding to some timestamp ts. The users wish to calculate the aggregation $\sum_{i=0}^{n-1} x_i$.

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 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 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 and 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'.

– $NoisyEnc(parms, x_{i,ts}, s_i, ts, \cdots)$: Takes the scheme's parameters, and a user's secret key s_i and time-series input $x_{i,ts}$, along with a timestamp ts. The input should be obscured with differentially private noise specified by additional parameters. The noise term may be directly specified, or a distribution from which to draw the noise may be given. Returns an encryption c_i of the user's noisy input under their secret key.
– $Agg(parms, s', ts, c_{0,ts}, \cdots c_{n-1,ts})$: Takes the scheme's parameters, the aggregation key, a timestamp, and the n time-series ciphertexts from the users (with timestamp ts). Returns $y_{ts} = x_{0,ts} \otimes x_{1,ts} \otimes \cdots \otimes x_{n-1,ts}$.

Users will run $NoisyEnc$ on their data, 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 aggregation result y_{ts}. In PSA schemes, the algorithm *Setup* is considered to be run in a trusted manner [7]. This can be accomplished through the use of an additional trusted third party, secure hardware, or secure multiparty computation. In PSA, only a single round of input-dependent communication must be performed, making it more efficient in communication than most existing MPC protocols [35, 25].

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 new information about uncompromised users' data. The idea of *aggregator obliviousness* encompasses these ideas. We restate the definition of aggregator obliviousness [7, 31] in Definition 5:

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 [7, 31] 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 which 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 argues $(i, x_{i,ts}, r_{i,ts})$ to the challenger and receives back $NoisyEnc(parms, sk_i, ts, x_{i,ts}, r_{i,ts})$ to the adversary. (Note that the adversary explicitly specifies the noise to be added in this case.)*

***Compromise**: The adversary argues $i \in [0, n) \cup \{\square\}$. 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 argues a set of participants $S \subset [0, n)$, with $i \in S$ not previously compromised. For each user $i \in S$, the adversary chooses two plaintext-noise pairs $(x_{i,ts}, r_{i,ts}), (\tilde{x}_{i,ts}, \tilde{r}_{i,ts})$ and sends them to the challenger. The challenger then chooses a random bit b. If $b = 0$, the challenger computes $c_{i,ts} = NoisyEnc(parms, s_i, ts, x_{i,ts}, r_{i,ts})$ 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 they can correctly guess the bit b chosen during the Challenge.

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 situations are inherent in many such scenarios with powerful adversaries, and are managed by constructing the security definition to require that no *additional* information is learned in such a case [7, 34].

4 Basic Aggregation without Differential Privacy

We first present basic schemes without differential privacy, and complete the construction of full PSA schemes in Section 6. We present two variants of our PSA scheme SLAP, each similar to popular lattice-based fully homomorphic encryption schemes (BGV [9] and B/FV [17]). These variants are evaluated against each other in Section 7. While the full BGV scheme has been used in previous lattice-based PSA as a black-box building block, we take a white-box approach, constructing a scheme similar to BGV but specially formulated for PSA. Further, we extend this approach to the B/FV cryptosystem, which has not been applied to PSA.

As a preliminary, let $\mathbf{PRF}_q(\cdot)$ be a pseudorandom function with its domain being the set of possible timestamps, and its range being R_q.

4.1 First Additive Scheme (BGV-like)

The first scheme we present is similar to the BGV fully homomorphic encryption scheme [9], and is also based on NTRU [21]. While the basic idea of embedding messages in ring polynomials is drawn from BGV, the actual scheme is quite different: operands are ring elements, not matrices or vectors. Further, the key distribution/correlation mechanism differs from BGV key generation. In this scheme $SLAP_{BGV}$, we consider the plaintext domain to be the ring R_t and the ciphertext domain to be the ring R_q, with $q \gg t$ and an appropriate value of the polynomial modulus degree N to allow for a desired level of security. Secret keys and error terms are drawn from a 1-bounded distribution on R_q. The scheme is defined as follows:

- $Setup_{BGV}(\lambda, t, n)$: 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 $\mathbb{Z}_q$ [3, 4]. (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 a 1-bounded distribution on R_q. Construct the aggregator's key as $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Return $parms = (R_q, t, n)$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $Enc_{BGV}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, ts)$: Choose the user's error $\mathbf{e}_{i,ts}$ from a 1-bounded distribution on R_q. Return the user's ciphertext $\mathbf{c}_{i,ts} = [\mathbf{PRF}(ts) \cdot$

$\mathbf{s}_i + t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts}]_q$ (based upon the secret key, the user's input, a small random error, and the timestamp ts).

- $Agg_{BGV}(parms, \mathbf{s}', ts, \mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts})$:
  Compute $\mathbf{y}_{ts} = [[\mathbf{PRF}(ts) \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q]_t$

Correctness When adding n ciphertexts $\mathbf{c}_{i,ts}$, we find $[\mathbf{PRF}(ts) \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q = [\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{x}_{i,ts})]_q$. 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 (B/FV-like)

In LaPS, BGV was used as a black-box building block, and in $SLAP_{BGV}$ we created a BGV-lke PSA scheme. We also explore the possibility of constructing a lattice-based PSA scheme whose mathematics are similar to the popular B/FV scheme and LPR hardness [17, 27]. As before, while the basic idea of message encoding is from the B/FV scheme, the particulars of our scheme are different: as one example, ciphertexts are singleton elements of R_q, not arbitrary-length tuples with elements from R_q. 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 5.1). Secret keys and error terms are drawn from a 1-bounded distribution on R_q. Denote $\Delta = \lfloor \frac{q}{t} \rfloor$. The scheme $SLAP_{B/FV}$ is defined as follows:

- $Setup_{B/FV}(\lambda, t, n)$: Takes in the security parameter λ, the plaintext modulus t, and the number of users n. Choose the ciphertext modulus q such that $log_2(3) + log_2(n) + 2 \cdot log_2(t) < log_2(q)$. Choose the polynomial modulus N such that λ bits of security are provided for the RLWE problem with ring polynomial coefficients in $\mathbb{Z}_q$ [3, 4]. Choose users' secret keys $\mathbf{s}_0 \cdots \mathbf{s}_{n-1}$ from a 1-bounded distribution on R_q. Construct the aggregator's key as $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Return $parms = (R_q, t, n)$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $Enc_{B/FV}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, ts)$: Choose the user's error $\mathbf{e}_{i,ts}$ from a 1-bounded distribution on R_q. Return the user's ciphertext $\mathbf{c}_{i,ts} = [\mathbf{PRF}(ts) \cdot \mathbf{s}_i + \mathbf{e}_{i,ts} + \Delta \cdot \mathbf{x}_{i,ts}]_q$ (based upon the secret key, the user's input, a small random error, and the timestamp ts).
- $Agg_{B/FV}(parms, \mathbf{s}', ts, \mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts})$: Compute $\mathbf{y}_{ts} = [\lfloor \frac{t}{q}(\mathbf{PRF}(ts) \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}) \rceil]_t$

Correctness When adding n ciphertexts $\mathbf{c}_{i,ts}$, we find $\mathbf{y}^{tmp} = [\mathbf{PRF}(ts) \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q = [\sum_{i=0}^{n-1} \mathbf{e}_{i,ts} + \Delta \mathbf{x}_{i,ts}]_q$. 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 $\frac{t}{q} \cdot || \sum_{i=0}^{n-1} \mathbf{e}_{i,ts} - \frac{q \mod t}{t} \sum_{i=0}^{n-1} \mathbf{x}_{i,ts} || < \frac{q}{2}$ [17]. 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 $SLAP_{BGV}$ and $SLAP_{B/FV}$ follow directly from the security of the BGV and B/FV cryptosystems on which they are based [9, 17]. The ciphertexts $\mathbf{c}_i$ are of exactly the same form of ciphertexts in the BGV and B/FV cryptosystems. Thus post-quantum security follows directly from reduction to the RLWE problem [27]. (While correctness may also similarly follow from the BGV and B/FV schemes, the parameter bounds are greatly relaxed when considering only additively homomorphic operations, which is why we explicitly state the bounds needed for correctness in $SLAP_{BGV}$ and $SLAP_{B/FV}$.)

5 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. We also discuss parameter choices for different scenarios.

5.1 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 $Agg_{B/FV}$) 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 [6, 19]. The full-RNS variant using floating-point operations [19] is much simpler and as efficient as the full-RNS variant using integer-only operations [6]. We thus adapt the operations of [19] 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 [19]. 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 $Agg_{B/FV}$. To accomplish this, we use the procedure of Simple Scaling from Section 2.3 of [19]. 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 $[\omega + v]_t$.

5.2 Number-Theoretic Transform

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 greatly desirable to reduce this complexity. To accomplish this, the Number-Theoretic Transform (NTT) can be applied [26]. This strategy uses two operations: the forward NTT transformation NTT and the inverse NTT $INTT$. 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 [26]. 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 5.1), all moduli q_i must satisfy this condition. In practice, these moduli are often precomputed.

5.3 Batching

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 only 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) [18].

Another method of batching is to use RNS decomposition (see Section 5.1) 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 5.4.

5.4 Practical Parameter Selection

In the scenario where 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 5.1 and 5.3, 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. When differentially private noise is added as in Section 6, parameter selection should take the possible addition of that noise into account.

A different scenario would be in the case of computing a binary aggregation (i.e., an OR over inputs in $\{0, 1\}$), in which case $t' = t = 2$ ($\mathbb{Z}_2$ is usually taken to be $\{0, 1\}$). De Morgan's Law can also be applied to calculate an AND over users' binary inputs, by negating each input and the final result.

6 Achieving Computational Differential Privacy

To construct PSA using the exact additive aggregation schemes presented in Section 4, we require that users add differentially private noise to their inputs before calling *Enc*. For the distribution of added noise, we utilize the discrete Laplacian distribution. In particular, we replace the basic *Enc* function of each scheme with the differentially private encryption function $NoisyEnc(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, ts, s, \epsilon, \delta, \alpha, \beta)$: From DL_s, add differentially private noise to each coefficient of $\mathbf{x}_{i,ts}$, and then simply call *Enc*. The noise added is specified for (ϵ, δ)-privacy and (α, β)-accuracy. Concretely, these functions are:

- $NoisyEnc_{BGV}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, ts, s, \epsilon, \delta, \alpha, \beta)$: Choose $\mathbf{z}_{i,ts} \in R_t$, where with probability β, $\mathbf{z}_{i,ts}$ has coefficients drawn from DL_s; with probability $1 - \beta$, $\mathbf{z}_{i,ts}$ will be zero. Set $\mathbf{x}'_{i,ts} = \mathbf{x}_{i,ts} + \mathbf{z}_{i,ts}$.
  Return the result of $Enc_{BGV}(parms, \mathbf{s}_i, \mathbf{x}'_{i,ts}, ts)$.
- $NoisyEnc_{BGV}(parms, \mathbf{s}_i, \mathbf{x}_{i,ts} \in R_t, ts, s, \epsilon, \delta, \alpha, \beta)$: Choose $\mathbf{z}_{i,ts} \in R_t$, where with probability β, $\mathbf{z}_{i,ts}$ has coefficients drawn from DL_s; with probability $1 - \beta$, $\mathbf{z}_{i,ts}$ will be zero. Set $\mathbf{x}'_{i,ts} = \mathbf{x}_{i,ts} + \mathbf{z}_{i,ts}$.
  Return the result of $Enc_{BGV}(parms, \mathbf{s}_i, \mathbf{x}'_{i,ts}, ts)$.

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. We prove the guarantee of privacy and accuracy for any (ϵ, δ) under reasonable conditions taken from LaPS [7], and apply it to our aggregation schemes. We show that the PSA schemes $SLAP_{BGV}$ and

$SLAP_{B/FV}$ using 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 $SLAP_{BGV}$ and $SLAP_{B/FV}$ using $NoisyEnc$ achieve (ϵ, δ)-differential privacy. Further, these PSA schemes achieve (α, β)-accuracy, where $\beta \geq 2\delta^{\frac{1}{\gamma}}$ and $\alpha = \frac{4w}{\epsilon}\sqrt{\frac{1}{\gamma}ln(\frac{1}{\delta})ln(\frac{2}{\beta})}$.*

Proof. This follows directly from [7], Theorem 3, which itself is from [31], Lemma 1. Our desired function 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.

Having constructed the complete, differentially private scheme SLAP, we can now state that SLAP satisfies the security requirement of aggregator obliviousness given in Section 3.4.

Theorem 2. *SLAP is an aggregator oblivious PSA scheme.*

Proof. See Appendix A.

7 Experimental Evaluation

In this section, we present experimental evaluations of our work. The LaPS scheme [7] 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., $\epsilon = 1$, $\delta = 0.1$, $\gamma \geq 0.0023$. 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 for some parameter settings. The ciphertext moduli required for $SLAP_{B/FV}$ is generally larger, which also necessitates a larger polynomial modulus degree. The ciphertext modulus size needed for $SLAP_{BGV}$ is smaller than the outer ciphertext modulus of LaPS (q_1), and is often even smaller than the inner ciphertext modulus (q_0) [7]. Also, our required polynomial modulus degree is smaller. This continues the improvement of smaller parameters that LaPS presented over related work [13], and shows that SLAP can be more efficient than previous work in communication overhead.

Table 1. Parameter Requirements for 128-bit security

Users	$\|t\|$	$\|q\|$ $(SLAP_{BGV})$	$\|q\|$ $(SLAP_{B/FV})$	N $(SLAP_{BGV})$	N $(SLAP_{B/FV})$
100	32	42	75	2,048	2,048
1000	32	45	78	2,048	2,048
10000	32	49	82	2,048	2,048
10000	128	145	274	8,192	16,384
10^{15}	128	181	310	8,192	16,384
10^{21}	128	201	330	8,192	16,384

Table 2. Latency of SLAP with 1000 users and 16-bit messages

Variant	Full-RNS $NoisyEnc$	Full-RNS Agg	NTL $NoisyEnc$	NTL Agg
BGV-like	1.17 ms	3.26 ms	43.53 ms	95.38 ms
B/FV-like	5.91 ms	16.98 ms	166.40 ms	272.26 ms

7.2 Implementation and Experiments

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 6. Our implementations were in C++14, and utilized NTL [33] for integer and polynomial arithmetic in the basic implementation. In the optimized implementation, the full-RNS variant presented in Section 5.1 is implemented, and the Number-Theoretic Transform is used to accelerate polynomial multiplication (as described in Section 5.2). Our implementations are published (anonymously) at *https://anonymous.4open.science/r/43ecd86b-44c2-4ad9-97e2-a8c6a020c746/*. 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 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 [7, 12]. Our tests took average runtimes over 50 trials of the operation in question. We use the standard ciphertext modulus and polynomial modulus specifications from `HomomorphicEncryption.org` [3].

We first compare SLAP directly to LaPS, matching the largest set of parameters they considered. LaPS is implemented using the HElib homomorphic encryption library [20], 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 2 shows the performance of SLAP with these parameters, along with the time it takes to do the computation in plaintext. Table 3 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 $SLAP_{BGV}$). Even without any optimizations such as those present in HElib, a basic implementation of SLAP still shows performance comparable to LaPS.

We next tested the scalability of the aggregation and decryption of SLAP with respect to the number of users in the aggregation. Taking $|t| = 32$, we evaluated the runtime of Agg for $n = 100, 1000, 10000, 100000$. As Figure 1

Table 3. Speedup of SLAP vs. LaPS with 1000 users and 16-bit messages

Variant	Full-RNS $NoisyEnc$	Full-RNS Agg	Basic $NoisyEnc$	Basic Agg
BGV-like	65.97x	20.76x	1.79x	0.71x
B/FV-like	13.09x	3.98x	0.46x	0.25x

Table 4. 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
BGV-like	0.0139x	0.0005x	28.39x	0.97x
B/FV-like	0.0027x	0.0002x	5.45x	0.34x

shows, the runtime of aggregation increases linearly with the number of users, showing the scalability of SLAP.

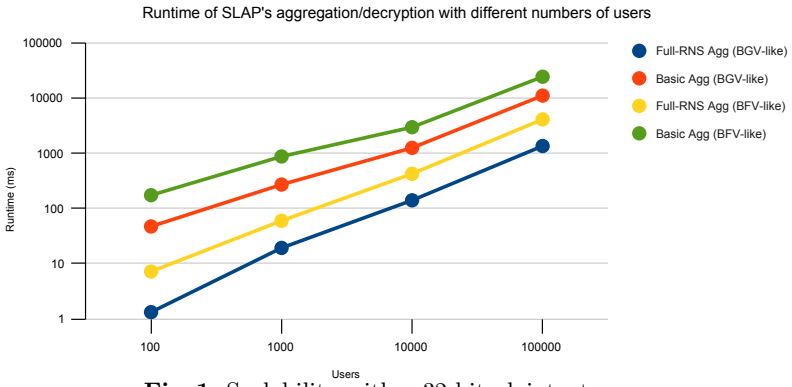

Fig. 1. Scalability with a 32-bit plaintext space

The direct comparison to LaPS only compared the case when a single message is included in a ciphertext. However, as discussed in Section 5.3, 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 Table 5. 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 Table 5, introducing even simple batching greatly increases the throughput SLAP can achieve. With this, the throughput of PSA can be improved to 628605 aggregations/second, reducing our amortized runtime to only microseconds.

To fully utilize our scheme's possible throughput, we then tested the use of the double-packing method from Section 5.3. We tested plaintext domains of 32, 64, 128, and 196 bits, each of which exceed 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. We show the results from such packing in Table 6. For larger plaintext spaces, SLAP is still reasonably performant, only requiring tens of milliseconds for $SLAP_{BGV}$ to run $NoisyEnc$ and Agg for $|t|$ of up to 192 bits. When analyzing throughput, SLAP is able to perform up to 812,396 aggregations per second (for a 128-bit plaintext space), and even shows a speedup relative to a naive non-PSA data aggregation.

Table 5. Throughput 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
BGV-like	1,747,079	628,605	47,369	21,470
B/FV-like	346,726	120,626	12,308	7,522

Table 6. Throughput of full-RNS SLAP with 1000 users and 16-bit messages (calculations/second)

Variant	$\|t\|$	k'	N	$NoisyEnc$ (batched)	Agg (batched)	Agg (batched) Speedup vs. Plain
BGV-like	32	1	2,048	1,727,284	626,321	20.0x
B/FV-like	32	1	4,096	692,389	240,753	7.7x
BGV-like	64	2	2,048	668,259	211,954	6.8x
B/FV-like	64	2	4,096	368,887	136,507	4.4x
BGV-like	128	4	8,192	1,403,887	546,029	17.4x
B/FV-like	128	4	16,384	741,917	277,198	8.8x
BGV-like	192	7	8,192	2,305,269	812,396	25.9x
B/FV-like	192	7	16,384	1,197,435	469,138	15.0x

From these experiments, we can conclude that $SLAP_{BGV}$ is generally superior to $SLAP_{B/FV}$. The BGV-based variant of SLAP has better parameter bounds and performance than the B/FV-based 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 (see Section 7.1), SLAP also requires less communication, which can further decrease the latency that end users of PSA will experience. Overall, these experiemnts show the high performance of SLAP, especially as compared to other work.

7.3 Comparison to Other Lattice-based PSA

Early work by Shi et al. [31] 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 [30] 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. [30] fixes some security holes in [1], but does not provide any experimental evaluation for a direct comparison. In both cases, the communication overhead their scheme incurs 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 shows that SLAP is more efficient and as compared to previous work, and would require less overhead in communication due to smaller ciphertexts. In [2], the runtime of aggregation is dominated by LWE decryption, which runs at 0.14 ms; however the parameter settings of $|q| < 15$ and $N = 256$ used in this evaluation is extremely small as compared to parameters used in SLAP or LaPS.

8 Conclusion

In this work, we presented SLAP, a PSA scheme with two variations that features efficiency, simplicity, and quantum security. We prove both the security and privacy of SLAP, which combined allows SLAP to achieve the security notion of aggregator obliviousness. Our implementation of both variants of SLAP (both basic and full-RNS implementations) shows improvements of over 20x for aggregation against previous work. Our experiments also show that SLAP can

achieve aggregation with a throughput of up to 628,605 aggregations/second for parameters matching previous research, and actually brings a speedup as compared to non-PSA aggregation. We conclude that SLAP brings both theoretical and practical improvements to the current state-of-the-art in lattice-based PSA.

References

1. Asmaa Abdallah and Xuemin Sherman Shen. A lightweight lattice-based homomorphic privacy-preserving data aggregation scheme for smart grid. *IEEE Transactions on Smart Grid*, 9(1):396–405, 2016.
2. Aarti Amod Agarkar, Mandar Karyakarte, and Himanshu Agrawal. Post quantum security solution for data aggregation in wireless sensor networks. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–8. IEEE, 2020.
3. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.
4. Martin R Albrecht, Benjamin R Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *International Conference on Security and Cryptography for Networks*, pages 351–367. Springer, 2018.
5. David Archer, Lily Chen, Jung Hee Cheon, Ran Gilad-Bachrach, Roger A Hallman, Zhicong Huang, Xiaoqian Jiang, Ranjit Kumaresan, Bradley A Malin, Heidi Sofia, et al. Applications of homomorphic encryption. *HomomorphicEncryption. org, Redmond WA, Tech. Rep.*, 2017.
6. Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.
7. Daniela Becker, Jorge Guajardo, and Karl-Heinz Zimmermann. Revisiting private stream aggregation: Lattice-based psa. In *NDSS*, 2018.
8. James Bell, K Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. *IACR Cryptol. ePrint Arch*, 2020.
9. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
10. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.
11. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
12. Matt Crane, Andrew Trotman, and Richard O'Keefe. Maintaining discriminatory power in quantized indexes. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1221–1224, 2013.
13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking

the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.
14. George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Smart meter aggregation via secret-sharing. In *Proceedings of the first ACM workshop on Smart energy grid security*, pages 75–80, 2013.
15. Zekeriya Erkin and Gene Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *International Conference on Applied Cryptography and Network Security*, pages 561–577. Springer, 2012.
16. Rastogi et al. Differentially private aggregation of distributed time-series with transformation and encryption, 2010.
17. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.
18. Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.
19. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the bfv homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*, pages 83–105. Springer, 2019.
20. Shai Halevi and Victor Shoup. Helib. *Retrieved from HELib: https://github. com. shaih/HElib*, 2014.
21. Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.
22. Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *International Conference on Financial Cryptography and Data Security*, pages 111–125. Springer, 2013.
23. Taeho Jung, Xiang-Yang Li, and Meng Wan. Collusion-tolerable privacy-preserving sum and product calculation without secure channel. *IEEE Transactions on Dependable and secure computing*, 12(1):45–57, 2014.
24. Taeho Jung, XuFei Mao, Xiang-Yang Li, Shao-Jie Tang, Wei Gong, and Lan Zhang. Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation. In *2013 Proceedings IEEE INFOCOM*, pages 2634–2642. IEEE, 2013.
25. Ryan Karl, Timothy Burchfield, Jonathan Takeshita, and Taeho Jung. Non-interactive mpc with trusted hardware secure against residual function attacks. In *International Conference on Security and Privacy in Communication Systems*, pages 425–439. Springer, 2019.
26. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.
27. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.
28. Paulo Martins, Leonel Sousa, and Artur Mariano. A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–33, 2017.
29. Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong 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*, pages 1–8, 2018.
30. Rihem Ben Romdhane, Hamza Hammami, Mohamed Hamdi, and Tai-Hoon 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)*, pages 1067–1072. IEEE, 2019.
31. Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer, 2011.
32. Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
33. Victor Shoup et al. Ntl: A library for doing number theory, 2001.
34. Jonathan Takeshita, Ryan Karl, and Taeho Jung. Secure single-server nearly-identical image deduplication. In *IoTSPT-ML at ICCCN 2020*. IEEE, 2020.
35. Filipp Valovich and Francesco Aldà. Computational differential privacy from lattice-based cryptography. In *International Conference on Number-Theoretic Methods in Cryptology*, pages 121–141. Springer, 2017.
36. Yang Yang, Xindi Huang, Ximeng Liu, Hongju Cheng, Jian Weng, Xiangyang Luo, and Victor Chang. A comprehensive survey on secure outsourced computation and its applications. *IEEE Access*, 7:159426–159465, 2019.

A Proof

Note our proof is very similar to existing PSA proofs [7, 31], and we adapt these existing techniques for our own protocol.

Theorem 3. *(Aggregator Obliviousness Security): Let the output of Enc be indistinguishable from random. Then both variants of SLAP are secure under aggregator obliviousness. problems.*

Proof. We follow previous work in assuming that a potential adversary can choose the noise $\mathbf{r}_i$ 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 RLWE ciphertexts in the BGV or B/FV schemes.

A Slightly Modified Game: In this proof, we modify the game of aggregator obliviousness (defined in Section 3.4) as follows: 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 $(\mathbf{x}_i, \mathbf{z}_i), (\tilde{\mathbf{x}}_i, \tilde{\mathbf{z}}_i)$ and then to distinguish between encryptions of either of the pairs. In this proof, we let the adversary choose one pair $(\mathbf{x}_i, \mathbf{z}_i)$ and have them distinguish between valid encryptions of $(\mathbf{z}_i, \mathbf{z}_i)$ 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.

For simplicity in the proof, we consider the protocol's operation at a single timestamp ts, and write $\mathbf{A} = \mathbf{PRF}(ts)$. We also omit the timestamp identifier of plaintexts, ciphertexts, and other variables.

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 learns 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 briefly define a game for $\mathcal{B}$ that describes their 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 either of the BGV or B/FV cryptosystems, $\mathcal{B}$ will play the modified game described above. In this, $\mathcal{B}$ can make Sample queries by arguing $\mathbf{m} \in R_t$ to $\mathcal{C}$ and will receive back the pair $(\mathbf{A}, \mathbf{M})$, where $\mathbf{A}$ is a publicly known element of R_q and $\mathbf{M}$ is an encryption under the secret key $\mathbf{s}^*$ (either in the BGV or B/FV styles, as appropriate) of $\mathbf{m}$. Then in the Distinguish part, $\mathcal{B}$ argues $\mathbf{m}^* \in R_t$ to $\mathcal{C}$. Based on a random bit b chosen by $\mathcal{C}$, $\mathcal{C}$ will choose $\mathbf{M}^*$ either as an encryption of $\mathbf{m}^*$ (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 the Compromise phase, $\mathcal{A}$ will send a query 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 the Challenge phase, $\mathcal{A}$ will choose a set of uncompromised users $U \subseteq [0, n) \setminus \{j, k\}$. They will then send plaintext-randomness pairs $\{(\mathbf{x}_i, \mathbf{z}_i)\}$ 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 $\{\mathbf{c}_i = NoisyEnc(parms, \mathbf{s}_i, \mathbf{x}_i, \mathbf{z}_i)\}$ for $i \in U \setminus \{j, k\}$

Now $\mathcal{B}$ enters the Distinguish phase and sends $\mathbf{m}_k = [\mathbf{x}_k + \mathbf{z}_k]_q$ to $\mathcal{C}$ who returns the tuple $(\mathbf{A}, \mathbf{M})$. Then, $\mathcal{B}$ sets $\mathbf{c}_k := \mathbf{M}$. $\mathcal{B}$ then computes an encryption of the sum of the plaintext-random pairs, with $\mathbf{v} = [\sum_{i \in U} \mathbf{x}_i + \mathbf{z}_i]_q$ and $\mathbf{c}_j = FHE.Enc(parms, \mathbf{A}, \mathbf{v})$, where FHE is either of the BFV or B/FV schemes, as appropriate. Now $\mathcal{B}$ has $\mathbf{c}_i$ for $i \in U$, including $\mathbf{c}_j$ and $\mathbf{c}_k$. $\mathcal{B}$ then returns these values $\mathbf{c}_i$ 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 $\mathbf{c}_k = \mathbf{M}$ is a valid encryption of $\mathbf{x}_k + \mathbf{r}_k$ and $\mathcal{A}$ 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 $\mathbf{M}$ from random and break the semantic security of the FHE scheme being used. This completes the proof by reduction. $\square$