TERSE: Tiny Encryptions and Really Speedy Execution for Post-Quantum Private Stream Aggregation

Jonathan Takeshita[1], Zachariah Carmichael[1], Ryan Karl[2], and Taeho Jung[1]

[1]University of Notre Dame, Notre Dame, IN 46556, USA
[2]Carnegie Mellon University, Pittsbufgh, PA 15213, USA
{jtakeshi,zcarmich,tjung}nd.edu,ryankarl915gmail.com

Abstract. The massive scale and performance demands of privacy-preserving data aggregation make integration of security and privacy difficult. Traditional tools in private computing are not well-suited to handle these challenges, especially for more limited client devices. Efficient primitives and protocols for secure and private data aggregation are a promising approach for private data analytics with resource-constrained devices. However, even such efficient primitives may be much slower than computation with plain data (i.e., without security/privacy guarantees).
In this paper, we present TERSE, a new Private Stream Aggregation (PSA) protocol for quantum-secure time-series additive data aggregation. Due to its simplicity, low latency, and low communication overhead, TERSE is uniquely well-suited for real-world deployment. In our implementation, TERSE shows very low latency for both clients and servers, achieving encryption latency on a smartphone of 0.0003 ms and aggregation latency of 0.0067 ms for 1000 users. TERSE also shows significant improvements in latency over other state-of-the-art quantum-secure PSA, achieving improvements of 1796× to 12406× for encryption at the client's end and 848× to 5433× for aggregation and decryption at the server's end.

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

1 Introduction

Motivation In modern computing and data analytics, aggregating a sum on data from many users is a frequently encountered problem. Ensuring security and privacy of user data in such aggregations while maintaining enough performance for practical deployment is a challenging issue, and is necessary to consider in order to comply with regulations for user protection such as GDPR. Secure and private data aggregation plays an important role in modern data analysis [63,31,62,55], with applications in statistical computation, smart metering, voting, and advertising analytics. At the massive scale of the modern Internet, with billions of users and devices [1], there is a need for high-performance implementations to perform high-scale aggregations. The huge scale and unique characteristics of the modern era of computing world presents new challenges that require novel solutions.

There exist general cryptographic tools for secure and private computation. However, the generality and pitfalls of these tools make them unattractive or infeasible for real-world deployment. Homomorphic encryption [20,33,26] allows computation over encrypted data, but its computational intensity and ciphertext size are too high for use on resource-limited devices. Secure multiparty computation [41] requires robust lines of communication for use in multiple rounds of communication, which may not be available in all locales, such as those in developing nations. Trusted Execution Environments such as Intel SGX [28] offer confidential computing, but face challenges at scale [51]. These challenges necessitate the development of efficient custom-built protocols for secure data aggregation.

To facilitate efficient secure and private aggregation, the study of Private Stream Aggregation (PSA) protocols has been undertaken and advanced in recent years [58,22,61,42,43,15,29,65,32]. Research in this area has focused on efficiency, though there is also work in fault tolerance and robustness. Many solutions for secure and private data analytics and outsourced computing focus on throughput on a large body of data [54,27,9,8,61]. In real-world deployments, the *latency* of a single computation, as opposed to the *throughput* across many epochs of time-series data, is of vital importance in real-time monitoring and reporting. Previous advances in secure aggregation have faced issues such as limited plaintext space, a lack of quantum security for future protection against quantum-capable attackers, high complexity and overhead due to the large ciphertext expansion, or focusing on throughput at the expense of latency due to the inherent computational intensity coming from the large ciphertext expansion.

Our Work In this work, we present a more efficient PSA protocol with quantum security, minimal latency and communication overhead. Our new protocol TERSE: **T**iny **E**ncryptions and **R**eally **S**peedy **E**xecution for Post-Quantum PSA is truly practical for latency-critical applications, satisfying the requirements of high performance without sacrificing guarantees of security and privacy. As online (input-dependent) computations are most critical for latency, we consider this metric of latency of input-dependent operations as what users are most interested in. For this reason, we focus on reducing the online costs of computation and communication as much as possible. Our research goal is thus to construct efficient RLWE-based PSA overcoming these issues. TERSE' ciphertexts are quantum secure via RLWE, and its trusted setup can be implemented with quantum-secure TEE-based symmetric encryption [18,50] and quantum-secure signature schemes for TEE [17].

Our construction is enabled by three insights: 1) giving adversary RLWE samples one coefficient at a time does not improve their advantage, 2) no input-dependent ring polynomial multiplication is required for additive lattice-based PSA, and 3) lattice-based PSA inputs that are ring polynomials can have input encoded coefficientwise. Combining these, we construct a novel PSA protocol using single coefficients of ring polynomials, resulting in much smaller ciphertexts and extremely efficient input-dependent encryption and aggregation. Our protocol mitigates the practical disadvantages of RLWE-based cryptography by

performing PSA with ring polynomials *one coefficient at a time*, and by precomputing intensive computations in advance of having inputs ready. This novel construction addresses the large ciphertext expansion that was inherent in previous lattice-based secure aggregation schemes, significantly reducing the latency of each aggregation. We show the real-world practicality of our novel construction with implementation results of both users and aggregators.

Our Contributions

1. We present TERSE, the first RLWE-based PSA scheme that can provide both low latency and high throughput, greatly reducing the size of a ciphertext for a single input. These novel traits allow TERSE to achieve operation latency measured in microseconds, making TERSE uniquely well-suited for performance-critical deployments.
2. We discuss the extension of TERSE with both well-known and cutting-edge extensions such as efficient ring polynomial arithmetic through RNS and NTT, SGX-based fault tolerance, and differential privacy. These extensions further support our goal of making private stream aggregation practically feasible for real-world use.
3. We implement TERSE and show experimental results demonstrating its performance and comparing it with plaintext aggregation. For $n = 1000$ users and a plaintext space of $|t| = 32$ bits, TERSE encryption achieves a latency of only 0.0003 ms, and aggregation and decryption run in 0.0067 ms. Our experiments with increasing numbers of users shows that TERSE is practically scalable for real-world deployments.

2 Related Work

2.1 Pre-Quantum PSA

The work of Shi et al. [58] established the field of PSA, creating the basic definitions and the first construction. The work of Shi et al. and Joye et al. rely respectively on the Decisional Diffie-Hellman (DDH) and the Decisional Composite Residuosity assumptions [58,37]. Other work based on the Discrete Logarithm problem has been proposed [38,40]. PSA has also been constructed for use in smart metering [56,60]. Chen et al. [24] presented a PSA scheme with dynamic joins/leaves and input tampering detection, based upon the DDH assumption. Wang et al. [66] created a scheme based on the Pallier cryptosystem [53] with fault tolerance and dynamic joins/leaves. These protocols are not secure against quantum-capable adversaries [59]; recent research has turned towards post-quantum PSA.

2.2 Post-Quantum PSA

The LaPS protocol [15] presented a PSA protocol integrating quantum security, improving upon previous bounds on the plaintext space, a generic and modular protocol with an instantiation, and the first implementation of a lattice-based PSA scheme. However, LaPS has several issues: it is extremely and needlessly complicated, and requires the black-box use of an FHE scheme (BGV [20] was

used in their instantiation), reducing its practicality. Further, its security with the "encrypt-once" model is subject to a simple attack [65], though this can be mitigated by requiring fresh public matrices at each timestamp.
The SLAP protocol [61] presented many improvements over LaPS. Instead of using an FHE scheme as a black-box subprotocol, SLAP used custom-built RLWE-based cryptographic constructions for PSA. Due to this, SLAP is much simpler than LaPS and is much more lightweight, with computational improvements over LaPS of 20× for aggregation and 65× for user-side encryption, and ciphertexts that are up to 2730× smaller at larger parameters. SLAP showed large improvements over LaPS in throughput for communication and computation, as well as in complexity. However, SLAP, like LaPS, is still subject to the high degree of ciphertext expansion common to RLWE encryption, resulting in higher latency and communication overhead. Both LaPS and SLAP operate in the encrypt-once model, where an adversary only sees a single ciphertext for a user at a given time. While this model is sufficient for most practical purposes, stronger ones have been proposed [65].
Other quantum-secure schemes with even lower latency have also been presented that manage to have smaller ciphertexts by not using RLWE. The LaSS scheme of Waldner et al. [65] uses secret sharing, and the scheme of Ernst et al. [32] uses a "deterministic version of the LWE [Learning With Errors] problem" known as Learning With Rounding (LWR) and key-homomorphic pseudorandom functions. Both schemes achieve runtimes on the order of milliseconds and have smaller input-dependent communication overhead. These schemes do have some issues: LaSS's per-user keys are linear in the number of users, i.e., the total number of keys is quadratic in the total number of users, making practical deployment for memory-limited IoT devices infeasible for larger numbers of users. The security of LWR, upon which [32] is dependent, is still in contention due to the deterministic rounding used [30]. Multi-key fully homomorphic encryption can also be applied to PSA, but is too general and burdensome to be appealing for IoT deployments [6,52]. Bao et al. used AES with noninteractively generated keys for PSA with message integrity [13].

2.3 PSA for IoT and Limited Devices

Lu et al. [49] constructed a PSA scheme using modified Pallier encryption and message authentication codes to form IoT-friendly PSA with protection against input tampering. Zhuo et al. [67] created a cloud-assisted protocol to compute on users' aggregated data, relying on the Diffie-Hellman and discrete logarithm assumptions and the BGV homomorphic encryption scheme. He et al. [36] create a scheme with the discrete logarithm and Diffie-hellman assumptions aimed at smart grids, which is able to withstand many different types of internal attacks. Li et al. [46] construct private dual-function aggregation by relying on the BGV homomorphic encryption scheme [19]; their scheme's practical performance is difficult to infer, as they only give an asymptotic performance analysis.

3 Background

3.1 Private Stream Aggregation

We consider the scenario where n users send inputs to a cloud server that is tasked with summing all user inputs. We assume that channels of communication are authenticated and nonmalleable; attackers impersonating users or modifying their messages in transit are outside the scope of this work. We use an honest-but-curious adversary model, where an adversary may view compromised parties' data, but will otherwise faithfully execute the protocol. This attacker model is commonly used for work in PSA [15,65,32,58]. PSA schemes are formally described with the following three algorithms:

1. $Setup(\lambda \in \mathbb{N}, \cdots)$: Takes a security parameter $\lambda \in \mathbb{N}$, along with other parameters such as the number of users and the plaintext space. Distribute secret keys s_i to each user and an aggregation key s' to the aggregator, and distribute publicly known parameters to all parties.
2. $Enc(s_i, ts, x_{i,ts}, r_{i,ts})$: Takes a user's input $x_{i,ts}$ at a particular timestamp ts, possibly along with differentially private noise $r_{i,ts}$. User i will call this function with their secret key s_i. Returns a ciphertext c_i.
3. $Agg(s', ts, c_{0,ts}, \cdots, c_{n-1,ts})$: The aggregator will call this function at timestamp ts, using its aggregation key s'. It will aggregate the ciphertexts $c_{0,ts}, \cdots, c_{n-1,ts}$, and output $y_{ts} = \sum_{i=0}^{n-1} x_{i,ts} + r_{i,ts}$.

Intuitively, we want to require that any adversary against a PSA scheme learns nothing more than they would when executing an idealized, black-box protocol that allows the aggregator to learn the sum of users' data. Inherent in this definition is that an adversary compromising the aggregator and $n-1$ users can inevitably learn information about the last user's data. In general, collusions of users and the aggregator have the ability to learn about information from uncorrupted users. This is a common issue in privacy-preserving protocols; preserving user privacy in the face of such attacks is a problem left to differential privacy. The *Setup* functionality is assumed to be performed by a trusted party, or collaboratively in a trusted format (e.g., using secure multiparty computation).
In the encrypt-once model, we assume that users will only produce a single input per timestamp. This model is used in prior work [61,15], and is a reasonable model of how real-world PSA deployments would function.

3.2 Definition of Security

Definition 1. *A PSA scheme is aggregator oblivious in the encrypt-once model if any probabilistic polynomial-time (PPT) adversary has no more than negligible advantage with respect to a security parameter λ in the following security game:*
Setup *The challenger runs the Setup algorithm, returning any public parameters to the adversary.*
Queries *The adversary may make up to $poly(\lambda)$ of following types of queries adaptively:*
- Encrypt: The adversary may specify $(i, ts, x_{i,ts}, r_{i,ts})$ and ask for the ciphertext. The challenger returns the ciphertext $c_{i,ts} = Enc(s_i, ts, x_{i,ts}, r_{i,ts})$ to the adversary.

- Compromise: The adversary specifies a party $i \in [0, n) \cup \{\square\}$. If $i = \square$, the challenger returns the aggregator's decryption key s' to the adversary (i.e., the aggregator is compromised.). Otherwise, the challenger returns user i's secret key s_i, to the adversary (i.e., user i is compromised).
- Challenge: This query is only made once. The adversary specifies a set of participants U and a time ts, such that neither ts nor any $i \in U$ was previously argued to Compromise. For each user $i \in U$, the adversary chooses a pair of inputs (user input, along with noise if applicable) $(x^0_{i,ts}, r^0_{i,ts})$ and $(x^1_{i,ts}, r^1_{i,ts})$. The challenger then chooses a random bit b, and returns the ciphertexts $\{c_{i,ts} = Enc(s_i, ts, x^b_{i,ts}, r^b_{i,ts})\}_{i \in U}$ to the adversary.
***Guess** The adversary attempts to guess b.*
The adversary wins if they can guess the bit b, and if the aggregator was compromised, then $\sum x^0_{i,ts} + r^0_{i,ts} = \sum x^1_{i,ts} + r^1_{i,ts}$.

Aggregator obliviousness essentially states that nothing more leaks from the protocol's execution than what a collusion of parties can derive from their inputs and output [58].

3.3 Ring Learning With Errors

Many modern cryptographic constructions draw their hardness assumptions from the Ring Learning With Errors (RLWE) problem, due to its conjectured difficulty for quantum adversaries and convenient mathematical structure. We briefly summarize RLWE here; the reader is referred to other work for a more in-depth discussion of RLWE [50,33]. Consider the negacyclic ring $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ for a large number q and power-of-two N. We denote the modular reduction of x modulo q as $[x]_q$, which is applied coefficientwise to polynomials. For a desired security level $\lambda \in \mathbb{N}$, there exist standard choices for q and N to guarantee at least λ bits of security in solving the RLWE problem for these parameters [5,4]. We say that a distribution is B-bounded if the values drawn from it have an infinity norm bounded above by B with all but negligible probability.
The RLWE problem is as follows: let $\mathbf{s}$ be chosen randomly from R_q, and consider random distributions χ, ζ on R_q. In practice, the distribution ζ is often chosen to be 1-bounded [33], while χ is uniformly random on R_q. Summarized succinctly, the RLWE problem states that terms of the form $[\mathbf{A}_i \cdot \mathbf{s} + \mathbf{e}_i]_q$ or $[\mathbf{A}_i \cdot \mathbf{s} + t \cdot \mathbf{e}_i]_q$ are computationally indistinguishable from random when $gcd(q, t) = 1$, $\mathbf{A}_i \leftarrow \chi$ is publicly known, and $\mathbf{e}_i \leftarrow \zeta$ [16,20,57].
In RLWE-based cryptosystems, elements of R_q are large objects, using kilobytes or even megabytes of memory. RLWE-based cryptosystems thus superficially seem impractical for secure aggregation with resource-limited devices. Our key innovation is a novel strategy to allow using only small portions of these terms, keeping the guarantee of quantum security, while achieving the functionality of additive aggregation.

3.4 The Random-Oracle Model

In cryptography, it is often convenient to assume the existence of a "random-oracle" hash function. A random-oracle hash function operates as a black-box

functionality available to users, guaranteeing them random output for given inputs with the caveat that identical inputs will yield identical outputs. While the assumption of the existence of random-oracle hash functions was previously contentious [21], there is little practical evidence of any security risks from using random-oracle hashes [44].

4 Basic Construction

In this section, we review the previous state-of-the-art quantum-secure aggregation scheme, and show how to modify it for more efficient aggregation. We then show it satisfies the security definition of aggregator obliviousness. Guaranteeing user input privacy through the addition of differentially private noise is not our novel contribution, and is left to Section 5.1.

4.1 Prior State-of-the-Art RLWE-Based PSA

The LaPS protocol [15] brought several new developments in PSA, including post-quantum security, more generous plaintext spaces, and better efficiency. Their efficiency gains were demonstrated with their implementation and more thorough experimental results, as compared to previous work. However, LaPS left much room for improvement; it is overly complex, affecting both usability and practical performance. This was partially due to their use of FHE.

The SLAP scheme [61] improved upon these issues by eschewing the black-box approach to additive homomorphism used by LaPS. Instead, SLAP used purpose-built homomorphic lattice arithmetic in their scheme. This resulted in more efficient operations and much smaller ciphertexts. SLAP also found that noise-scaled message encoding is more efficient than message-scaled encoding; we follow their example by using noise-scaled encoding in TERSE. SLAP focused on practical throughput from message packing, but their latency was slightly greater than that of other state-of-the-art post-quantum PSA [32,65]. However, these schemes not using RLWE have disadvantages such as large key storage requirements or doubts on security [30], which leads us to focus on optimizing RLWE-based PSA.

SLAP and LaPS have some similarities. First, being related to RLWE or its A-LWE variant, public values $\mathbf{A}$ dependent on a timestamp were (or should have been) used. As is common to many PSA schemes [58,37], additively correlated secret keys are used. While SLAP represented a great leap forward in the state of the art, it still faced limitations in communication overhead. The ciphertexts of SLAP are ring polynomials in R_q, which may be as large as megabytes for common parameter settings. This makes it less practical for highly constrained users. SIMD batching still helps SLAP achieve high throughput, but the overall ciphertext size cannot be reduced. This necessitates either filling the remaining SIMD slots with junk data (greatly reducing throughput), or waiting for enough data to fill a ring polynomial, which may be undesirable in time-sensitive aggregations. These schemes consider security in the encrypt-once model, assuming each user will only produce a single encryption at a given timestamp – a reasonable security model for most applications.

Other work in quantum-secure PSA not using RLWE [65,32] is able to achieve much smaller ciphertexts. However, these schemes have some disadvantages. The security of the Learning With Rounding problem upon which [32] is based is of concern [30]. Using secret-sharing for aggregation [65] is not practical for large numbers of users, due to the quadratic growth in key storage needed. It is thus desirable to construct RLWE-based secure stream aggregation with smaller ciphertexts.

4.2 A More Performant Protocol: TERSE

We now show how to further break down lattice arithmetic for an even more efficient protocol, with smaller ciphertexts. Our path forward hinges on a few key ideas: first, with precomputation of user values, no expensive input-dependent polynomial multiplication is required, and the computation of these terms can be done ahead of time or prepared concurrently. (Phones and other limited devices can do this while plugged in and idling, or in a separate process, or they may outsource the precomputations to a synced computer.) This means that all input-dependent polynomial arithmetic is only addition, scalar multiplication, and base conversion, which can be done coefficient-wise. Second, transmitting elements of R_q one coefficient at a time does not give adversaries any additional information about users' secret data. Third, SLAP can apply a simple coefficient-wise SIMD batching to improve the throughput of their scheme. Packing the coefficients in this manner means that coefficients can be packed into a polynomial or extracted at any point in the addition-only computation, without affecting correctness.
Combining these insights gives us the core idea: we can perform aggregation and decryption *one polynomial coefficient at a time*, which does not impact security or correctness, and reduces the ciphertext size needed to send a single element of $\mathbb{Z}_t$ by a factor of N, which usually ranges from 2^{10} to 2^{16}. We can now describe TERSE, which applies the key ideas above. Essentially, we parse a timestamp into two parts, with one part used to index coefficients of polynomials in R_q. Then, we simply use coefficients at that index from the precomputed product of the user's secret key and the public hash.
Let $\lambda \in \mathbb{N}$ be the bits of guaranteed security, and let $\mathbf{A}_\theta = h(\theta)$ be a random-oracle hash function mapping the high bits θ of timestamps $ts = (\theta, \tau)$ to R_q. We consider a small error distribution ζ and a uniformly random distribution χ, drawing either polynomials or singleton values from R_q or $\mathbb{Z}_q$ as appropriate. Denote the i-th coefficient of a polynomial $\mathbf{x}$ as $\mathbf{x}[i]$. Differential privacy is an orthogonal extension, as discussed in Section 5.1, so we do not go into detail on the mechanisms of differentially private noise added through the terms $r_{i,ts}$. We describe TERSE as follows:

1. $TERSE.Setup(\lambda, t, n)$: For a plaintext space of $\mathbb{Z}_t$ and n users, choose the ciphertext modulus q such that $log_2(3) + log_2(n) + log_2(t) < log_2(q)$ and q, t are coprime. Choose N to ensure at least λ bits of security for the RLWE problem on R_q [4,5], and $H(\cdot)$ to be a random hash mapping timestamps to R_q. Choose users' secret keys $\mathbf{s}_0 \cdots \mathbf{s}_{n-1}$ randomly from χ. Finally, choose the additively correlated aggregator's key $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Users and the

aggregators parse timestamps into most significant and least significant bits as $ts = (\theta, \tau)$, with $\tau \in \mathbb{Z}_N$, i.e., τ is represented using up to $|N|$ bits.

2. $TERSE.Enc(\mathbf{s}_i \in R_q, ts = (\theta, \tau), x_{i,ts} \in \mathbb{Z}_t, r_{i,ts} \in \mathbb{Z}_t)$: Choose the user's RLWE error $e_{i,ts} \in \mathbb{Z}_t$ from ζ. Set $p_{i,ts} = (\mathbf{A}_\theta \cdot \mathbf{s}_i)[\tau]$. (Note that these steps can and should be precomputed before the user's time-series input $x_{i,ts}$ is available.) The user's ciphertext is $c_{i,ts} = [p_{i,ts} + t \cdot e_{i,ts} + [x_{i,ts} + r_{i,ts}]_t]_q$.
3. $TERSE.Agg(\mathbf{s}' \in R_q, ts = (\theta, \tau), c_{0,ts} \cdots c_{n-1,ts})$: Precompute $p'_{ts} = (\mathbf{A}_\theta \cdot \mathbf{s}'_{i,ts})[\tau]$. The sum of users' inputs $x_{i,ts}$ is $y_{ts} = [[p'_{ts} + \sum_{i=0}^{n-1} c_{i,ts}]_q]_t$.

Correctness is easy to see. Note that $p'_{ts} = -\sum p_{i,ts}$. Then $[p'_{ts} + \sum_{i=0}^{n-1} c_{i,ts}]_q = [\sum_{i=0}^{n-1} t \cdot e_{i,ts} + [x_{i,ts} + r_{i,ts}]_t]_q$. Reducing this modulo t removes the noise terms, and we avoid noise overflow so long as the bounds in $TERSE.Setup$ are observed. Note that the input-dependent portion of encryption in TERSE is extremely simple, requiring only base conversion from base t to base q, followed by one modular addition in base q. Similarly, the online portion of aggregation of TERSE only needs the additive aggregation of all user ciphertexts and p'_{ts}, followed by a base conversion. This simplicity leads to highly efficient operations for both the user and aggregator, as shown in Section 6.

Improving upon other work in RLWE-based PSA, TERSE achieves a relatively small ciphertext expansion - for a single input in $\mathbb{Z}_t$, a ciphertext is an element of $\mathbb{Z}_q$, so that the expansion factor is only q/t, and only $|q|$ bits are needed for ciphertexts – in practice, this is usually only 64 or 128 bits!

4.3 Proof of Security

Lemma 1. *In attempting to solve the RLWE problem (in either of the Search or Decision versions in Section 3.3), an adversary does not gain any advantage from seeing elements of R_q one coefficient at a time.*

Theorem 1. *TERSE is an aggregator oblivious PSA scheme.*

Proof. We follow previous work [15,3] and assume for simplicity that adversaries can choose the differentially private noise terms $r_{i,ts}$ during the Challenge phase. We will construct a reduction from RLWE to TERSE by showing that given an adversary $\mathcal{A}$ that can win the game of aggregator obliviousness (see Definition 1) in polynomial time, we can construct an adversary $\mathcal{B}$ able to distinguish RLWE terms from random in polynomial time, thus solving the Decisional version of RLWE. For simplicity, we consider a real-or-random version of the game of aggregator obliviousness, again following previous work [58,15,61]. As noted in Section 3.1, aggregator obliviouness does not protect against the case where all but one party is compromised, so we suppose that $\mathcal{A}$ will not attempt to make Compromise queries for n distinct parties.

First, consider a challenger $\mathcal{C}$ who tests the ability of $\mathcal{B}$ to attack RLWE. $\mathcal{B}$ will compute and return TERSE parameters including R_q, t, n for a given security level λ to $\mathcal{A}$ as a response to a Setup query from $\mathcal{A}$. R_q is the ring for which $\mathcal{B}$ will attempt to attack RLWE. $\mathcal{B}$ will then choose two distinct parties $j, k \in [0, n) \cup \{\square\}$, and draw secret keys $s_i \leftarrow \chi$ for $i \notin \{j, k\}$ (exactly as in $TERSE.Setup$).

As noted in previous work [15,61], $\mathcal{B}$'s choices j, k must be in the set of at least two users $\mathcal{A}$ will not attempt to compromise, which occurs with probability $\frac{1}{n^2}$. Next, $\mathcal{B}$ needs to prepare to match RLWE samples with the values it is able to send to $\mathcal{A}$. If $\mathcal{A}$ will make Encrypt queries for (up to) $\mathcal{P} = poly(\lambda)$ different timestamps, $\mathcal{B}$ will simply ask for $\mathcal{Q} = \lceil \frac{\mathcal{P}+1}{N} \rceil$ RLWE samples from $\mathcal{C}$. From this, $\mathcal{B}$ will receive a set of pairs $\mathcal{S} = \{(\mathbf{a}_\sigma, \mathbf{b}_\sigma)\}_{\sigma \in \mathbb{Z}_\mathcal{Q}}$. Note that θ and σ are both in $\mathbb{Z}_\mathcal{Q}$. Then, $\mathcal{B}$ will select $H(\cdot)$ such that for each of the $\mathcal{P}$ values $ts = (\theta, \tau)$, $H(ts) = \mathbf{a}_\theta[\tau]$, and distribute this function as part of $TERSE.Setup$.
When $\mathcal{A}$ makes an Encrypt query $(i, x_{i,ts}, r_{i,ts}, ts)$ to $\mathcal{B}$, if party i is not compromised and the pair i, ts has not been used in a previous Encrypt query, $\mathcal{B}$ will compute and return the TERSE encryption $NoisyEnc(\mathbf{s}_i, ts, x_{i,ts}, r_{i,ts})$ if $i \notin \{j, k\}$. For the parties j, k, $\mathcal{B}$ will eventually set j's secret key to be the secret RLWE value, and will (implicitly) let user k's secret key be the sum of all other users' keys. If $i = j$, with $ts = (\theta, \tau)$, then $\mathcal{B}$ finds the tuple $(\mathbf{a}_\tau, \mathbf{b}_\tau)$, and returns $\mathbf{b}_\theta[\tau] + (x_{j,ts} + r_{j,ts})$ to $\mathcal{A}$. If $i = k$, $\mathcal{B}$ again finds the appropriate value $\mathbf{b}_\theta[\tau]$, and returns $-\mathbf{b}_\theta[\tau] - (\mathbf{a}_\theta \cdot \sum_{\ell \notin \{j,k\}} \mathbf{s}_\ell)[\tau] + (x_{k,ts} + r_{k,ts})$ to $\mathcal{A}$.
When $\mathcal{A}$ makes a Compromise query for a party $i \in ([0, n) \cup \{\square\}) \setminus \{j, k\}$, $\mathcal{B}$ simply returns $\mathbf{s}_i$ to $\mathcal{A}$. We denote the set of never-compromised users as $K \subseteq [0, n) \cup \{\square\}$. If $i \in \{j, k\}$, i.e., $\mathcal{A}$ tried to compromise a user that $\mathcal{B}$ assumed would remain uncompromised, then $\mathcal{B}$ will simply abort.
When $\mathcal{A}$ makes a Challenge query, it will choose a set of uncompromised users $U \in K$, and will send input-noise pairs
$\{(x_{u,ts'}, r_{u,ts'})\}_{u \in U}$, where $ts' = (\theta', \tau')$ was not previously used in any Encrypt query. At this point, up to $\mathcal{Q} - 1$ timestamps have been used in Encrypt queries, leaving at least one unused value remaining. Then, $\mathcal{B}$ will compute the values $\mathbf{c}_{i,ts} = NoisyEnc(\mathbf{s}_i, ts', x_{i,ts}, r_{i,ts})$ for $i \in U \setminus \{j, k\}$, $c_{j,ts} = \mathbf{b}_{\theta'}[\tau'] + (x_{j,ts} + r_{j,ts})$, and $c_{k,ts} = -\mathbf{b}_{\theta'}[\tau'] - (\mathbf{a}_{\theta'} \cdot \sum_{\ell \notin \{j,k\}} \mathbf{s}_\ell)[\tau'] + (x_{k,ts} + r_{k,ts})$. Finally, $\mathcal{B}$ will return $\mathbf{c}_{i,ts}$ for $i \in [0, n) \cup \{\square\}$ to $\mathcal{A}$.
To make a decision on whether it was given real RLWE terms or random values, $\mathcal{B}$ will use the decision of $\mathcal{A}$. If $\mathcal{A}$ decides that it was given ciphertexts that are simply messages padded with random values, then $\mathcal{B}$ will decide that it was given random values. On the other hand, if $\mathcal{A}$ decides that it is in a real version of TERSE and had received TERSE ciphertexts, then $\mathcal{B}$ will decide that it received RLWE values from $\mathcal{C}$. Thus if $\mathcal{A}$ can achieve a greater-than-negligible advantage (i.e., a success rate non-negligibly better than a random guess) against the aggregator obliviousness of TERSE, then $\mathcal{B}$ can use this to gain a greater-than-negligible advantage against RLWE. This completes the reduction from RLWE to TERSE.

5 Extensions and Improvements

5.1 Differential Privacy

Previously, we have introduced TERSE and discussed its security in the context of aggregator obliviousness, where the aim is to provide security against external attackers and leak no additional information to an honest-but-curious aggregator

or user, or a collusion thereof. To construct a PSA protocol that is truly private, users should also have some notion of input privacy against the honest-but-curious aggregator. In particular, a user will want to avoid having the aggregator learn anything about its input. To this end, PSA schemes utilize differential privacy to obscure user inputs.

Differential privacy for PSA is well-known in the literature [61,15,58,32,65]. The exact mechanism of differential privacy (choosing values $r_{i,ts}$, based upon n, t, and the desired or acceptable accuracy and error) is an orthogonal issue to our work. We thus follow previous work in noting that differential privacy from adding noise to user inputs is preserved when executed in a PSA protocol [65,64]. Thus, TERSE is easily able to encapsulate both security and privacy, both of which are important concerns for users.

In practical use, it should be noted that implementing differential privacy into PSA (or any computation) can affect the accuracy of the computation. In RLWE-based PSA schemes with finite and possibly limited plaintext spaces, needing to account for noisy user input can significantly affect the practical parameter selection [15,61].

5.2 Network Faults or Disconnects

While previous work in quantum-secure PSA [15,61,32,65] has centered primarily upon efficient PSA construction, practical PSA should take fault tolerance into consideration. In several precursor works in PSA, user keys are correlated, such that the absence of a single user's input will result in failure of decryption [61,15,58,65]. As noted by Karl et al. [42], there are two prominent strategies of enhancing aggregation schemes with fault tolerance: recovering from faults by having a trusted party substitute missing inputs [12,13,47,2], or by having users provide redundant inputs as a precaution against future faults [14,24,25]. The Cryptonite protocol [42] uses trusted hardware, specifically Intel SGX, as a trusted third party for fault tolerance.

Intel SGX is a Trusted Execution Environment that provides confidentiality and integrity to a trusted portion of a program, which runs in an encrypted memory enclave that maintains integrity against even a malicious operating system [28]. While SGX provides strong guarantees of security to computations, it can be limited for computations at a very large scale, due to the practically limited size of its memory enclave and the overhead of encryption when paging memory in or out of the enclave [34,28,7,10,45].

Cryptonite used an aggregator-colocated SGX to read all user inputs and to generate encryptions of zero from missing users, using pre-received users' secret keys held securely in trusted memory. The more efficient variant of Cryptonite has the SGX only output encryptions of zero corresponding to missing users, so that the aggregation with ciphertexts from all users can be more efficiently performed in untrusted space. We note that for our model of honest-but-curious adversaries, it is reasonable for the SGX to trust that the aggregator will faithfully relay the set of faulted users. Then, the SGX does not need to take in $O(n)$ ciphertexts, but only a list of missing users, yielding a much smaller buffer that

is passed to the enclave. Further, we can have the SGX return only a single aggregated ciphertext from all missing users, greatly reducing the amount of data that needs to be passed out of the enclave.
We include Cryptonite with these optimizations in our implementation (see Section 6.1). While other methods of fault tolerance exist, we chose to study the novel integration of SGX-based fault tolerance and post-quantum PSA due to their non-interactive fault recovery without additional client work or interaction. This is the first work investigating the implementation and use of SGX-based fault tolerance and PSA featuring aggregation on order of microseconds.
There exists other work in PSA dealing with dynamic join/leave of users [22,39]. Dynamic user groups are outside the scope of this work, which is concerned with the basic primitive of efficient aggregation.

5.3 Optimizing Ring Arithmetic

In $R_q = \mathbb{Z}_q[x]/(x^N + 1)$, both N and q may be large - N commonly ranges from 2^{10} to 2^{16}, and q may be hundreds of bits. These large operands are an obstacle to efficient computation. Residue Number System (RNS) arithmetic can decompose an element of $\mathbb{Z}_q$ into a tuple of numbers modulo smaller coprimes, allowing the use of multiple single-precision operations instead of expensive multiprecision arithmetic [11,35]. The Number-Theoretic Transform (NTT) reduces the asymptotic complexity of polynomial multiplication from $O(N^2)$ to $O(N \cdot log(N))$, greatly improving the runtime of algorithms whose dominant operation is polynomial multiplication [48]. We use these optimizations in implementing TERSE. Due to the design of TERSE in minimizing input-dependent computation, both users and the aggregator will perform all of their polynomial multiplication ahead of time, so the benefits of NTT are seen in the runtime of precomputation.

6 Experimental Evaluation

6.1 Implementation and Environment

We implemented client and server programs to test the performance of TERSE. Both implementations use RNS and NTT as described in Section 5.3. Runtime tests generally report averages of at least 50 iterations; for longer-running tests at least 5 iterations were used. Both implementations were in C++.
We tested the client version of TERSE on a Google Pixel 4a with 6GB memory and a CPU running at up to 2.2 GHz. The client version assumes precomputation of users' values $p_{i,ts}$, but does not do so for the noise terms $t \cdot e_{i,ts}$. As described earlier, this is a reasonable assumption to make - these larger polynomial products can be outsourced to users' synced laptops for smartphone clients, or computed "out-of-band" in a separate process, because one polynomial generates thousands of $p_{i,ts}$ terms, and the key retrieval and computation of $p_{i,ts}$ only needs to occur once per N aggregations. Drawing error terms is a fast operation, and can done as input becomes available.
Our server tests were run on a computer with an Intel Xeon CPU running at 3.7GHz, with 128GB of RAM. Our server code integrates a modified version

Table 1: TERSE Parameter Settings and Precomputation Times for 128-bit RLWE security and 1000000 Aggregations

Users	Plaintext Space (bits)	Minimum Ciphertext Space (bits)	Cipher-text Moduli	RLWE Poly-nomial Modulus Degree	Secret Keys Generation (ms)	Derivation of $\mathbf{A}_\theta$ (ms)	Multiplicative Precomputation (ms)
100	32	41	1	2048	6.75889	28.0973	451.451
1000	32	44	1	2048	66.5254	28.049	450.521
10000	32	48	1	2048	666.651	28.1902	451.375
100000	32	51	1	2048	6650.29	28.0568	450.602
1000000	32	54	1	2048	66587.6	28.0712	450.984
10000000	32	58	2	4096	1715910	100.646	1128.82
100000000	32	61	2	4096	17164700	99.7437	1132.2
1000	1	13	1	1024	33.7708	22.1876	221.474
1000	2	14	1	1024	33.842	22.2518	222.798
1000	4	16	1	1024	33.7107	22.1595	221.613
1000	8	20	1	1024	33.7945	22.1702	221.553
1000	16	28	1	2048	66.5783	28.1416	450.641
1000	32	44	1	2048	66.5213	28.0544	450.474
1000	48	60	2	4096	171.914	99.7837	1128.89

of the Cryptonite protocol for fault-tolerance, as described in Section 5.2. Our client and server implementations are available at https://gitlab.com/jtakeshi/slap-iot-cryptonite-client and https://gitlab.com/jtakeshi/slap-iot-cryptonomial-server, respectively. Our profiling of precomputations (see Section 6.3) is included in the client implementation repository.

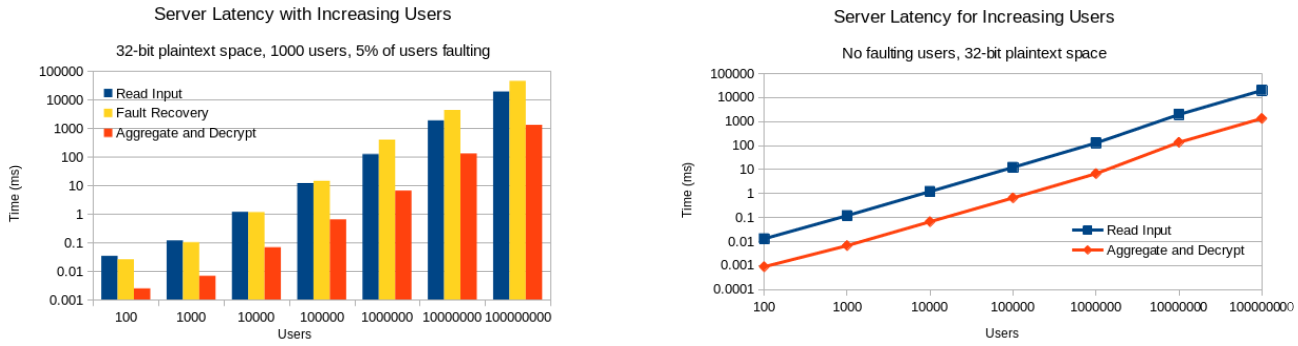
6.2 Parameters and Communication

A plaintext space of up to 48 bits is practical for a wide variety of practical uses, e.g., electronic voting for up to 2^{47} participants, or averaging patient ages for 2^{40} patients, or aggregating 65,536 users' 32-bit inputs for use in machine learning or data mining. Further, it allows us to keep TERSE ciphertexts small, and TERSE plaintexts within a single computer word. In our implementations of TERSE, we used standard RLWE parameters for 128-bit classical security [4,23]. For $|t| \leq 64$, only one or two RNS moduli represented in 64-bit words were required, making TERSE's communication overhead very lightweight. Parameter settings for our experiments are shown in Table 1.

6.3 Results

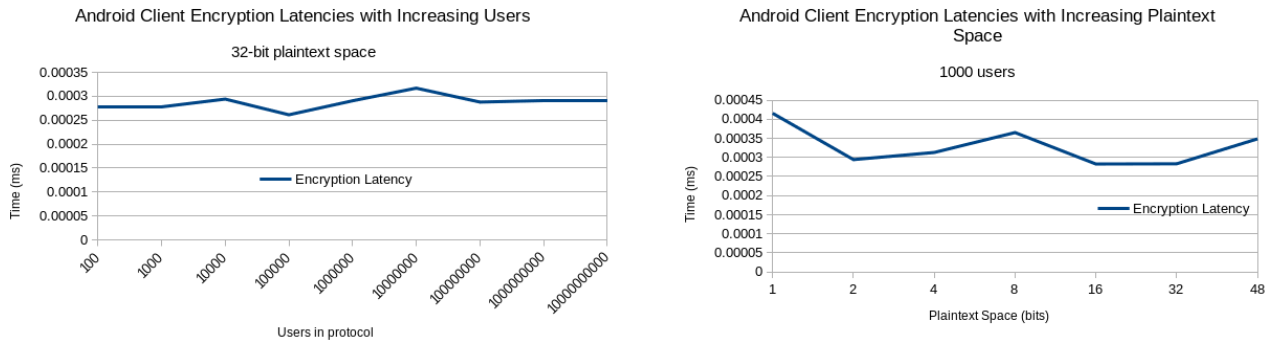
Impact of Aggregation Scale We first tested the impact of high scale and increasing users on our protocol's runtime. The aggregator's server-side performance is shown in Figure 1. The server achieves aggregation latency of 0.0067 ms for $n = 1000$ users, which is much more efficient than other state-of-the-art work in post-quantum PSA (see Section 6.4). The results from our Android user-side implementation are shown in Figure 2a. One of the strengths of TERSE as compared to aggregation schemes based upon secret sharing [65] is that users' computation (and memory to store keys) is not linearly dependent upon the number of other participants, making TERSE much more practical for deployment to users with limited devices such as smartphones or IoT devices. This is borne out by the minimal changes in users' encryption runtimes as the number

of users increases. Most notably, user-side encryption on an Android smartphone can take place in *less than 0.3 microseconds for 1 billion users*!



(a) (b)

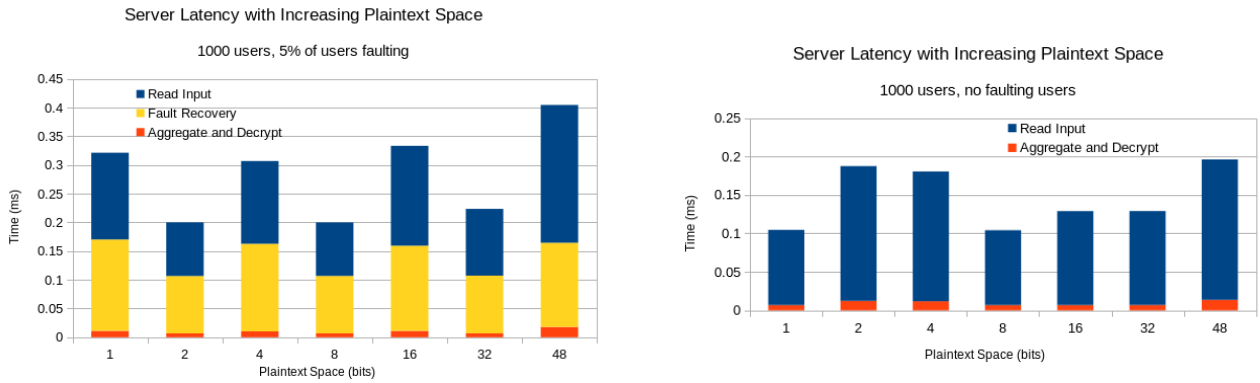
Fig. 1: Experimental Results for Server Performance With Increasing Users. (a) Without Faults (b) With Faults



(a) (b)

Fig. 2: Experimental Results for Client Performance. (a) Increasing Users (b) Increasing Plaintext Space

Impact of Input Size We investigated the impact of an increasing plaintext space upon runtime. As described in Section 6.2, we expect very little asymptotic effect from larger plaintext spaces up to the 64 bits used in our implementation, as only one or two ciphertext moduli are required in all cases. The server-side results from increasing the plaintext space are shown in Figure 3. In both cases, the runtime for the actual aggregation is very small, on order of 0.001 milliseconds (or 1 microsecond!). The runtime for reading inputs from file and fault



(a) (b)

Fig. 3: Experimental Results for Server Performance With Increasing Plaintext Space. (a) With Faults (b) Without Faults Space

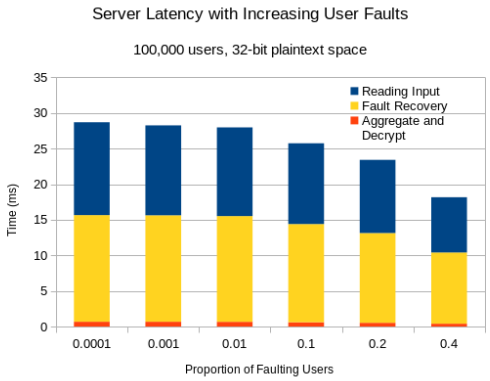


Fig. 4: Server Performance for Varying User Faults

recovery is much larger, and more variable. Our Android implementation's results are shown in Figure 2b, and again show that client encryption can run in less than a microsecond.

Impact of Fault Recovery We evaluate Cryptonite-based fault tolerance [42] as applied to TERSE with a few key differences. Instead of $O(n)$ inputs being passed into the SGX's secure memory enclave, in our setting we only need to pass in a list of the *faulting* users. This greatly reduces the paging overhead for calls into the enclave. We also only return a single ciphertext, reducing the paging overhead from returning to untrusted memory. Our experimental results for fault tolerance with TERSE are shown in Figure 4. As the proportion of faulting users increases, the time to aggregate and read user input from file decreases slightly. This is logical: with fewer user inputs, there is less work for these portions of the computation.

Table 2: Runtime in ms of TERSE vs. Reported Results from Other Work (1000 users and at least 16 bits plaintext space, unless noted)

Protocol	Encryption Runtime	Aggregation Runtime	Notes
TERSE	0.0003	0.006	
LaSS [65]	0.539	0.509	256-bit AES keys used. Encrypt-once model times shown here, many-time security much slower. Also does not consider differential privacy. Experiments used a laptop.
Ernst et al. [32]	0.913	0.875	Results from $\lambda = 114$ bits of security. Reimplements LaSS [65], reporting about a 2× speedup from the original. Experiments used a laptop.
LaPS [15]	3.722	1.964	Results from $\lambda = 80$ bits of security. Runtimes at 128 bits of security are an order of magnitude greater. Experiments used a laptop.
SLAP [61]	1.17	3.26	Ordinary latency reported, practical throughput may be improved. Differentially private noise included. Experiments used a server.
Lu et al. [49]	0.328	0.062	8-bit plaintext space, runtime slightly higher with differential privacy. Experiments used a laptop.
Zhuo et al. [67]	≈0.0001	6	Only $N = 100$ users, plaintext space not specified. Experiments used a desktop.
He et al. [36]	6.66	3433.4	Both figures only estimates, no implementation. Results estimated from a Pentium IV system.

Precomputation While TERSE's precomputations do not directly affect online latency, it is informative to observe their performance. To concretely observe the burden of precomputation, we tested the latency of the generation of secret keys (done by trusted setup), public hash derivation (done by both client and server), and finding multiplicative terms $\mathbf{A}_\theta \cdot \mathbf{s}_i$. Table 1 shows the runtimes of these operations when precomputing values for 1000000 aggregations. As expected, runtime for secret key generation increases linearly with the number of users. Deriving the public values $\mathbf{A}_\theta$ takes only hundreds of milliseconds at the most, and the per-aggregation burden is low when considering amortization for a million aggregations. Similarly, computing the terms $\mathbf{A}_\theta \cdot \mathbf{s}_i$ for a million aggregations takes only seconds at most. We note that the precomputations of TERSE need occur only once every N aggregations, while other post-quantum PSA need to calculate their public terms for every aggregation (while LaPS does not explicitly require this, it is needed for security [65]).

6.4 Comparison with Other Work

Differences in hardware platform, programming languages, and input types can make direct performance comparisons between different PSA protocols challenging. Still, we can make rough comparisons of protocol runtimes and communication overhead from reported experimental results. In Table 1, we report runtimes of TERSE and other schemes, using a plaintext space of at least 16 bits and at least 1000 users (unless otherwise noted). We note that these minimal bounds for scheme parameters are quite small, and TERSE is likely to perform even better relatively at scale; we chose smaller floors to show the best-possible runtime of other schemes. Against the 4 quantum-secure PSA schemes LaSS, Ernst et al., LaPS, and SLAP, TERSE shows improvements in latency of 1796× to 12406× for encryption and 848× to 5433× for aggregation.

7 Conclusion

In this paper, we presented TERSE, a quantum-secure PSA protocol uniquely well-suited for minimal latency. TERSE features highly efficient operations, minimally expansive ciphertexts, and a very simple and highly extensible design. Our experimental results show that TERSE achieves encryption latency of 0.0003 ms and aggregation latency of 0.0067 ms for 1000 users and a 16-bit plaintext space, with improvements of two to three orders of magnitude as compared to prior post-quantum PSA. The performance improvements of TERSE for client-side operations are especially important, as the client-side implementation of TERSE was tested on a smartphone as opposed to prior implementations of PSA on laptops or desktops. These microsecond-latency operations for users and aggregators make RLWE-based PSA truly practical for real-world deployments.

References

1. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. https://rb.gy/cbrasa. Accessed: 2021-10-15. 1
2. ppm-hda: Privacy-preserving and multifunctional health data aggregation with fault tolerance. 11
3. G. Ács and C. Castelluccia. I have a dream!(differentially private smart metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011. 9
4. 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. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. 6, 8, 13
5. M. R. Albrecht et al. Estimate all the {LWE, NTRU} schemes! In *SCN*, pages 351–367. Springer, 2018. 6, 8
6. P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *Theory of Cryptography Conference*, pages 28–57. Springer, 2020. 4
7. S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, et al. {SCONE}: Secure linux containers with intel {SGX}. In *12th USENIX OSDI)*, pages 689–703, 2016. 11
8. Y. N. Babuji, K. Chard, A. Gerow, and E. Duede. Cloud kotta: Enabling secure and scalable data analytics in the cloud. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 302–310. IEEE, 2016. 2
9. S. F. Bailey, M. K. Scheible, C. Williams, D. S. Silva, M. Hoggan, C. Eichman, and S. A. Faith. Secure and robust cloud computing for high-throughput forensic microsatellite sequence analysis and databasing. *Forensic Science International: Genetics*, 31:40–47, 2017. 2
10. M. Bailleu, J. Thalheim, P. Bhatotia, C. Fetzer, M. Honda, and K. Vaswani. {SPEICHER}: Securing lsm-based key-value stores using shielded execution. In *17th USENIX FAST*, pages 173–190, 2019. 11
11. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *SAC*, pages 423–442. Springer, 2016. 12
12. H. Bao and R. Lu. DDPFT: Secure data aggregation scheme with differential privacy and fault tolerance. In *2015 IEEE ICC*, pages 7240–7245. IEEE, 2015. 11
13. H. Bao and R. Lu. A new differentially private data aggregation with fault tolerance for smart grid communications. *IoT-J*, 2(3):248–258, 2015. 4, 11
14. H. Bao and R. Lu. A lightweight data aggregation scheme achieving privacy preservation and data integrity with differential privacy and fault tolerance. *Peer-to-Peer Networking and Applications*, 10(1):106–121, 2017. 11
15. D. Becker, J. Guajardo, and K.-H. Zimmermann. Revisiting Private Stream Aggregation: Lattice-Based PSA. In *NDSS*, 2018. 2, 3, 5, 7, 9, 10, 11, 16
16. I. Blanco-Chacón. On the RLWE/PLWE equivalence for cyclotomic number fields. *Applicable Algebra in Engineering, Communication and Computing*, pages 1–19, 2020. 6
17. D. Boneh, S. Eskandarian, and B. Fisch. Post-quantum epid signatures from symmetric primitives. In *Cryptographers' Track at the RSA Conference*, pages 251–271. Springer, 2019. 2

18. X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher. Quantum security analysis of aes. *IACR Transactions on Symmetric Cryptology*, 2019(2):55–93, 2019. 2
19. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014. 4
20. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011. 2, 3, 6
21. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004. 7
22. T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *FC*, pages 200–214. Springer, 2012. 2, 12
23. H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine. Simple encrypted arithmetic library v2. 3.0. *Microsoft Research, December*, 2017. 13
24. J. Chen, H. Ma, and D. Zhao. Private data aggregation with integrity assurance and fault tolerance for mobile crowd-sensing. *Wireless Networks*, 23(1):131–144, 2017. 3, 11
25. L. Chen, R. Lu, and Z. Cao. PDAFT: A privacy-preserving data aggregation scheme with fault tolerance for smart grid communications. *Peer-to-Peer networking and applications*, 8(6):1122–1132, 2015. 11
26. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Asiacrypt*, pages 409–437. Springer, 2017. 2
27. F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, et al. An iot endpoint system-on-chip for secure and energy-efficient near-sensor analytics. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2481–2494, 2017. 2
28. V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016. 2, 11
29. G. Danezis, C. Fournet, M. Kohlweiss, and S. Zanella-Béguelin. Smart meter aggregation via secret-sharing. In *ACM SEDAy*, pages 75–80, 2013. 2
30. J. Ding, X. Gao, T. Takagi, and Y. Wang. One sample ring-lwe with rounding and its application to key exchange. In *ACNS*, pages 323–343. Springer, 2019. 4, 7, 8
31. J. Du, C. Jiang, E. Gelenbe, L. Xu, J. Li, and Y. Ren. Distributed data privacy preservation in iot applications. *IEEE Wireless Communications*, 25(6):68–76, 2018. 1
32. J. Ernst and A. Koch. Private stream aggregation with labels in the standard model. *PETS*, 4:117–138, 2021. 2, 4, 5, 7, 8, 11, 16
33. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012. 2, 6
34. A. T. Gjerdrum, R. Pettersen, H. D. Johansen, and D. Johansen. Performance of Trusted Computing in Cloud Infrastructures with Intel SGX. In *CLOSER*, pages 668–675, 2017. 11
35. S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *CT-RSA*, pages 83–105. Springer, 2019. 12
36. D. He, N. Kumar, and J.-H. Lee. Privacy-preserving data aggregation scheme against internal attackers in smart grids. *Wireless Networks*, 22(2):491–502, 2016. 4, 16
37. M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *FC*, pages 111–125. Springer, 2013. 3, 7

38. T. Jung et al. Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation. In *2013 Proceedings IEEE INFOCOM*, pages 2634–2642. IEEE, 2013. 3
39. T. Jung, J. Han, and X.-Y. Li. PDA: semantically secure time-series data analytics with dynamic user groups. *TDSC*, 15(2):260–274, 2016. 12
40. T. Jung, X.-Y. Li, and M. Wan. Collusion-tolerable privacy-preserving sum and product calculation without secure channel. *TDSC*, 12(1):45–57, 2014. 3
41. R. Karl et al. Non-interactive MPC with trusted hardware secure against residual function attacks. In *SecureComm*, pages 425–439. Springer, 2019. 2
42. R. Karl et al. Cryptonite: A framework for flexible time-series secure aggregation with online fault tolerance. Cryptology ePrint Archive, Report 2020/1561, 2020. https://rb.gy/tdcsfs. 2, 11, 15
43. R. Karl et al. Cryptonomial: a framework for private time-series polynomial calculations. In *SecureComm*, pages 332–351. Springer, 2021. 2
44. N. Koblitz and A. J. Menezes. The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography*, 77(2):587–610, 2015. 7
45. R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer. Tensorscone: A secure tensorflow framework using Intel SGX. *arXiv preprint arXiv:1902.04413*, 2019. 11
46. C. Li, R. Lu, H. Li, L. Chen, and J. Chen. PDA: a privacy-preserving dual-functional aggregation scheme for smart grid communications. *Security and Communication Networks*, 8(15):2494–2506, 2015. 4
47. Q. Li and G. Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In *PETS*, pages 60–81. Springer, 2013. 11
48. 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*, pages 124–139. Springer, 2016. 12
49. R. Lu, K. Heung, A. H. Lashkari, and A. A. Ghorbani. A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot. *IEEE Access*, 5:3302–3312, 2017. 4, 16
50. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013. 2, 6
51. S. Mofrad, F. Zhang, S. Lu, and W. Shi. A comparison study of intel sgx and amd memory encryption technology. In *HASP*, pages 1–8, 2018. 2
52. P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key fhe. In *Eurocrypt*, pages 735–763. Springer, 2016. 4
53. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pages 223–238. Springer, 1999. 3
54. H. Park, S. Zhai, L. Lu, and F. X. Lin. {StreamBox-TZ}: Secure stream analytics at the edge with {TrustZone}. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 537–554, 2019. 2
55. Y. Pu, J. Luo, C. Hu, J. Yu, R. Zhao, H. Huang, and T. Xiang. Two secure privacy-preserving data aggregation schemes for iot. *Wireless Communications and Mobile Computing*, 2019, 2019. 1
56. V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD/PODS*, pages 735–746, 2010. 3
57. M. Rosca, D. Stehlé, and A. Wallet. On the ring-LWE and polynomial-LWE problems. In *Eurocrypt*, pages 146–173. Springer, 2018. 6
58. E. Shi, T. H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, volume 2, pages 1–17, 2011. 2, 3, 5, 6, 7, 9, 11

59. P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *FOCS*, pages 124–134. Ieee, 1994. 3
60. Z. Sui and H. de Meer. An efficient signcryption protocol for hop-by-hop data aggregations in smart grids. *IEEE Journal on Selected Areas in Communications*, 38(1):132–140, 2019. 3
61. J. Takeshita et al. SLAP: Simple Lattice-Based Private Stream Aggregation Protocol. *IACR Cryptol. ePrint Arch.*, 2020:1611, 2020. 2, 4, 5, 7, 9, 10, 11, 16
62. W. Tang, J. Ren, K. Deng, and Y. Zhang. Secure data aggregation of lightweight e-healthcare iot devices with fair incentives. *IoT-J*, 6(5):8714–8726, 2019. 1
63. S. Tonyali, K. Akkaya, N. Saputro, A. S. Uluagac, and M. Nojoumian. Privacy-preserving protocols for secure and reliable data aggregation in iot-enabled smart metering systems. *FGCS*, 78:547–557, 2018. 1
64. F. Valovich and F. Aldà. Computational differential privacy from lattice-based cryptography. In *NuTMiC*, pages 121–141. Springer, 2017. 11
65. H. Waldner, T. Marc, M. Stopar, and M. Abdalla. Private stream aggregation from labeled secret sharing schemes. *IACR Cryptol. ePrint Arch.*, 2021:81, 2021. 2, 4, 5, 7, 8, 11, 13, 16
66. X. Wang, Y. Liu, and K.-K. R. Choo. Fault-tolerant multisubset aggregation scheme for smart grid. *IEEE Transactions on Industrial Informatics*, 17(6):4065–4072, 2020. 3
67. G. Zhuo, Q. Jia, L. Guo, M. Li, and P. Li. Privacy-preserving verifiable data aggregation and analysis for cloud-assisted mobile crowdsourcing. In *INFOCOM*, pages 1–9. IEEE, 2016. 4, 16