Cryptonomial: A Framework for Private Time-Series Polynomial Calculations

Ryan Karl, Jonathan Takeshita, Alamin Mohammed, Aaron Striegel, and Taeho Jung

University of Notre Dame, Notre Dame, IN 46556, USA
{rkarl,jtakeshi,amohamm2,striegel,tjung}@nd.edu

Abstract. In modern times, data collected from multi-user distributed applications must be analyzed on a massive scale to support critical business objectives. While analytics often requires the use of personal data, it may compromise user privacy expectations if this analysis is conducted over plaintext data. Private Stream Aggregation (PSA) allows for the aggregation of time-series data, while still providing strong privacy guarantees, and is significantly more efficient over a network than related techniques (e.g. homomorphic encryption, secure multiparty computation, etc.) due to its asynchronous and efficient protocols. However, PSA protocols face limitations and can only compute basic functions, such as sum, average, etc.. We present Cryptonomial, a framework for converting any PSA scheme amenable to a complex canonical embedding into a secure computation protocol that can compute any function over time-series data that can be written as a multivariate polynomial, by combining PSA and a Trusted Execution Environment. This design allows us to compute the parallelizable sections of our protocol outside the TEE using advanced hardware, that can take better advantage of parallelism. We show that Cryptonomial inherits the security requirements of PSA, and supports fully malicious security. We implement our scheme, and show that our techniques enable performance that is orders of magnitude faster than similar work supporting polynomial calculations.

Keywords: Private Multivariate Polynomial Evaluation · Trusted Execution Environment · Secure Aggregation

1 Introduction

Third-party analysis on personal records is becoming increasingly important due to widespread data collection in the modern world. However, this data often contains private information about users such that its publication could seriously compromise their privacy, and a number of studies have shown that significant precautions must be taken to protect such data from malicious actors [6]. Accordingly, it would be beneficial to have a technology that allows a third-party aggregator to learn the result of the analysis performed on users' private datasets over a network, but nothing else. Many such distributed analyses can be done by or approximated by multivariate polynomial calculations. Common

machine learning (ML) tasks such as linear regression, support vector machines (SVMs), activation functions, etc., can be formulated as a multivariate polynomial function over users' private inputs. In recent times, the COVID-19 crisis has led to a renewed interest in applying ML to disease detection and diagnosis, and a number of highly successful techniques [9, 29] have been developed to assist medical researchers in combating the virus. For such pressing demands, we consider the problem of allowing a set of users in S to privately compute a polynomial function over their collected time-series data such that an untrusted aggregator only learns the final result, and no individual honest user's data is revealed. More formally, we aim at supporting polynomial evaluation over users' time-series input data in the following format of a *general multivariate polynomial*: $f(\{x_{i,j}\}_{i\in S, j=1,\ldots,z}) = \sum_{j=1}^{z} c_j(\prod_{i\in S} \mathbf{m}_{i,j,ts}^{e_{i,j}})$, where z is the number of product terms in the polynomial, c_j and $e_{i,j}$ are public parameters, and $\mathbf{m}_{i,j,ts}$ are secret data from the i-th user at time stamp ts.

There are relatively few practical techniques that can be utilized in this setting, where maintaining the privacy of patients' medical data is of critical (and due to HIPPA, legal) importance. Fully Homomorphic Encryption (FHE), Differential Privacy (DP), Secure Multiparty Computation (MPC), or Oblivious Polynomial Evaluation (OPE) might be used individually as black boxes to solve this problem, but each have significant constraints that negatively impact their practical deployment in the real world. FHE's high computational overhead leads to significant slowdown that makes it impractical in large-scale settings. DP adds noise to the final output of the function, and the resulting accuracy loss can greatly harm the predictive power of any ML analysis. MPC requires participants to send multiple messages during protocol execution, which can seriously degrade overall runtime. OPE also requires multiple messages to be sent, and is focused on the two party setting, which limits its applicability in large scale data analysis.

Private Stream Aggregation (PSA) is a form of distributed secure computing that is promising for achieving this functionality. With this technique, users independently encrypt their input data and send it to an aggregator in a way that allows the aggregator to efficiently learn the aggregation results of time-series data without being able to infer individual data. PSA is generally superior to other types of secure computation paradigms (e.g., MPC, FHE) in large-scale applications involving time-series data because of its extremely low overhead and the ease of key management [17, 30]. Notably, PSA is non-interactive (i.e., users send their time-series data in a "stream" and only one message is sent per time interval) and asynchronous (i.e., users can leave after submitting their inputs), making it more efficient in communication than most existing alternative techniques [36]. Although PSA is a mature field of study, prior work in this field is mostly limited to simple aggregation (sum, average, etc.). Due to these limitations, it is challenging for even the most advanced PSA protocols to be deployed in real-world applications for computing stream polynomial evaluation over users' time-series data.

To overcome this limitation with existing works, we developed the Cryptonomial framework, which can convert any PSA scheme amenable to a complex canon-

ical embedding (CCE) [8], an isometric ring homomorphism between complex numbers and integral polynomials, into a privacy-preserving stream polynomial evaluation scheme, that supports additional functionality beyond an additive sum, up to general stream polynomial evaluation. With the use of a Trusted Execution Environment (TEE), we avoid sacrificing security or performance, and can build a highly scalable protocol that is capable of computing richer statistics with high efficiency and throughput. Our framework intelligently combines/tweaks traditional quantum-secure PSA, the CCE, and a TEE to efficiently support stream polynomial evaluation without incurring the drawbacks of simply using a TEE alone or a PSA scheme alone to directly do so. Although it may seem more efficient to simply send plaintext data to an SGX enclave to be computed over, it is known that some TEEs (e.g., Intel SGX) have difficulties exploiting multi-threading [35] due to the lack of common synchronization primitives often found on traditional operating systems, and leveraging threading within TEEs can introduce security vulnerabilities [37] which compromise data privacy. Overall performance can be improved by outsourcing the computationally expensive steps to an untrusted space in an encrypted form, so we can leverage more robust forms of parallel computing especially on high performance hardware such as GPUs, which is not possible with the approaches entirely based on TEEs.

Cryptonomial combines additive PSA with a complex canonical embedding to develop a multivariate polynomial PSA, where single product terms are leaked to the aggregator in a basic design. This leakage is prevented by integrating a TEE into the design, where only a small constant amount of computations are outsourced due to the nature of our design. These techniques allow for significant performance improvements over the current state-of-the-art protocols for privacy-preserving polynomial calculations by multiple orders of magnitude. It is noteworthy that our framework is compatible with state-of-the-art techniques in computational differential privacy [2,31,36] which prevent adversarial inference from the outcomes of aggregation. Cryptonomial contributes to the development of secure ecosystems of collection and analysis involving user-generated datasets, by increasing the utility of the data gathered for data aggregators, while still ensuring the privacy of users with a strong set of guarantees. Note that tolerance of online/offline faults and input poisoning (when malicious users send false inputs to poison the final function output) are orthogonal problems to our work, and this paper focuses on expanding the versatility of PSA. Existing solutions towards these problems [18, 20] can be incrementally deployed on top of ours if either of these properties are needed.

In summary, our contributions are as follows:

- We present a new framework to support for the first time PSA-based general stream polynomial evaluation.
- We demonstrate the strong provable privacy guarantees of our instantiated protocol by presenting a formal proof of security.
- We provide an implementation in order to evaluate the performance when compared to existing work and verify the improved efficiency over existing work by multiple orders of magnitude. Our implementation is open source and

available at an anonymous repository `https://anonymous.4open.science/r/ea7619a6-3c77-483f-86c0-2ba60068ea54/`.

2 Potential Applications of Cryptonomial

Regression Analysis: Regression analysis (i.e. polynomial regression and ridge regression) is a statistical process for estimating the relationship among multiple variables, with numerous applications in finance, medical research, and a number of other domains [28]. In this type of data analysis each user i 's data record is described as a feature vector $\mathbf{x}$ and a dependent variable y_i, and training a regression model is to find $\mathbf{p}$ which minimizes $MSE(\mathbf{p}) = \sum_i (y_i - \mathbf{p}\mathbf{x}_i)^2$, i.e., the linear predictor who predicts users' dependent variable vector $\mathbf{y}$ using their feature matrix $\mathbf{X}$ with minimum mean squared error. Since $MSE(\mathbf{p})$ is convex, it is minimized if and only if $\mathbf{A}\mathbf{p} = \mathbf{b}$ where $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and $\mathbf{b} = \mathbf{X}^T\mathbf{y}$, such that $\mathbf{A} = \sum_i \mathbf{x}_i\mathbf{x}_i^T$ and $\mathbf{b} = \sum_i y_i\mathbf{x}_i$. By using our technique, the aggregator can obliviously evaluate any polynomial regression model.

Support Vector Machines: This protocol can be useful for modeling/predicting diseases in individuals, by supporting a variety of privacy-preserving ML techniques, such as support vector machines (SVM). Supervised machine learning methods have high performance in solving classification problems in many biomedical fields, particularly the SVM [38]. Because the SVM approach is data-driven and model-free, it has discriminative power for classification, especially in cases where sample sizes are small and there are large numbers of variables. This technique has recently been used to develop automated classification and detection of diseases in the clinical setting [24, 33], but in all of these cases, participants' privacy was not preserved, and participants either forfeited their data or signed legal agreements that their data would not be shared. In many instances this level of privacy protection may not be sufficient, and we seek to design a system that protects the privacy of each individual data point from public health authorities.

3 Related Work

There are six primary techniques that can be leveraged to achieve traditional secure aggregation or secure polynomial evaluation: 1) FHE, which suffers from high computational overhead, 2) DP, which introduces noise that negatively impacts accuracy, 3) MPC, which increases the communication complexity (number of communication rounds) compared to other techniques, 4) PSA, which overcomes most of the communication and computational overhead constraints of the previous approaches, but is limited to computing simple functions, 5) OPE which supports polynomials but suffers from high communication complexity and is primarily focused on the two party setting, and 6) Secure/Privacy-preserving Polynomial Evaluation, which also suffers from high communication complexity and scalability issues. We discuss these in more detail below:

Fully Homomorphic Encryption: FHE [8] can be applied to evaluate a multivariate polynomial securely. However, the key management is nontrivial.

The aggregator must be trusted to not decrypt ciphertexts pre-aggregation, or the duties of aggregation and decryption should be separated between two servers. Furthermore, there can be significant computational overhead when using FHE. For instance, existing work leveraging FHE [21] to construct secure protocols for aggregation reports an overall computation time of approximately 15 minutes when computing over only 40 thousand data points. As a result, FHE is often impractical in large-scale aggregations.

Differential Privacy: Modified DP has been used in existing works [27, 30] to achieve O(1) error, while using generic differential privacy techniques alone would result in at least Ω(N) error. Note, [27] also considers periodic aggregation of the sum statistic in the presence of an untrusted aggregator. Their work does not present a formal security definition and requires that the aggregator engage in an extra round of interaction with the participants to decrypt the sum for every time interval. While these techniques can be useful, we are interested in better techniques that do not seriously impact the final accuracy of the aggregation. High amounts of noise or accuracy loss in the secure aggregation can greatly harm the predictive power of future data analysis, and we are primarily focused on the case where users do not apply differentially private noise to their inputs.

Secure Multiparty Computation: MPC protocols allow a set of parties to securely compute an arbitrary function over their inputs [3]. While it is feasible to evaluate a polynomial with MPC, MPC protocols require multiple messages be sent each time interval (round) between users, limiting scalability. All users must wait on the slowest user, and the runtime of each round is determined by that of the slowest user. In cases when MPC is conducted over the Internet, communication round complexity is often the primary bottleneck [3, 19], since network latency slows the delivery of packets necessary for continuing to execute the protocol. This problem becomes significantly worse when parties are geographically distant and the communication latency of each message is high.

Private Stream Aggregation: PSA was first studied by Rastogi et al. [27] and Shi et al. [30]. There have been many papers that build on these works, but the vast majority focus on sum aggregation, and not on more complicated functions, such as those based on polynomials. The most extensively studied domain is the pre-quantum PSA based on the Decisional Diffie-Hellman (DDH) assumption and/or the Decisional Composite Residuosity (DCR) assumption [16]. These PSA schemes are vulnerable against quantum computers. Some post-quantum PSA schemes are superior to pre-quantum PSA schemes in overall throughput due to the smaller parameters enabled by the quantum-secure constructions and the various algorithmic optimizations available in quantum-proof cryptography. Early work in quantum-secure PSA [2] employed existing lattice-based encryption schemes as black-box building blocks and was disadvantaged in performance due to complex designs. More recent work in quantum-secure PSA [31] has used a white-box approach to reduce the complexity and overhead in both computation and communication, but is still limited to a single additive aggregation.

Oblivious Polynomial Evaluation: OPE is a protocol involving two parties, a sender whose input is a polynomial $\mathcal{P}$, and a receiver whose input is a value α. At

Paper	Supports Polynom.	Computation Complexity	Comm. Complexity	Comm. Rounds	Users	Tools and Techniques	Security Remarks
[2]	No	$\mathcal{O}(n)$	$\mathcal{O}(n)$	1	n	PSA, FHE	Q, AO
[16]	No	$\mathcal{O}(n)$	$\mathcal{O}(n)$	1	n	ECC, PSA	AO
[26]	No	$\mathcal{O}(n)$	$\mathcal{O}(n)$	1	n	ECC, PSA	DP
[30]	No	$\mathcal{O}(n+\sqrt{n\Delta})$	$\mathcal{O}(n)$	1	n	ECC, PSA	AO
[4]	Yes	$\mathcal{O}(l \cdot n)$	$\mathcal{O}(l \log l)$	Multiple	n	MPC	SH
[13]	Yes	$\mathcal{O}(\lambda(l+x))$	$\mathcal{O}((\lambda + dn^{1/d})n^2)$	Multiple	n	MPC	FM
[18]	Yes	$\mathcal{O}(n)$	$\mathcal{O}(kn^2\chi + xk)$	Multiple	n	ECC, PSA	Sem
[10]	Yes	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Multiple	n	MPC	SH
[25]	Yes	$\mathcal{O}((\lambda d+1)on)$	$\mathcal{O}((\lambda d+1)n)$	Multiple	n	OPE, MPC	SH
[24]	Yes	$\mathcal{O}((\lambda d+1)o)$	$\mathcal{O}(\lambda d+1)$	Multiple	2	OPE	FM
[34]	Yes	$\mathcal{O}((\lambda d+1)o)$	$\mathcal{O}(\lambda d+1)$	Multiple	2	OPE	FM
[14]	Yes	$\mathcal{O}(\lvert X\rvert + \lvert Y\rvert \log \lvert X\rvert)$	$\mathcal{O}(\lvert X\rvert + \lvert Y\rvert)$	Multiple	2	OPE	FM
[7]	Yes	$\mathcal{O}(dl+\lambda)$	$\mathcal{O}(dl(dl+\lambda))$	Multiple	2	OPE	FM
[39]	Yes	$\mathcal{O}((\lambda d+1)o)$	$\mathcal{O}(\lambda d+1)$	Multiple	2	OPE	FM
[12]	Yes	$\mathcal{O}(dDrn)$	$\mathcal{O}(d(10D+1)(\sum_{j=1}^{n}\sum_{i=1}^{b_j}\log\alpha_{j,i}+1))$	Multiple	2	OPE	FM
Ours	**Yes**	$\mathcal{O}(n)$	$\mathcal{O}(n)$	**1**	**n**	**PSA,TEE**	**Q, AO**

Fig. 1. Comparison of Existing Work and Cryptonomial; s is the number of secret shares, n is the number of users, w is the plaintext modulus, Δ is the range of inputs, l is the bit length of the inputs, λ is the security parameter, d is the degree of the polynomial, x is the number of terms in the polynomial, χ is the bitlength of the safe prime numbers, t is the minimum threshold of participants in the aggregation, o is the number of points defined on the polynomial, a is the length of the RSA modulus, X and Y are sets of elements, D is the sum of the logarithms of the variable degrees for polynomials consisting of r monomials, b is the number of inputs for each user, α is the degree of the inputs, and k is a parameter where $k \leq n$. Q indicates quantum security, AO indicates aggregator obliviousness, SH/FM indicates security against semi-honest and fully malicious adversaries respectively, Sem indicates semantic security, DP indicates differential privacy and TA indicates a trusted aggregator is required.

the end of the protocol the receiver learns $\mathcal{P}(\alpha)$ and the sender learns nothing [25]. There are many interesting applications of this idea, including private comparison of data, mutually authenticated key exchange, and anonymous coupons. Many have built on top of this idea, to support operations over floating point numbers for use training neural networks [7], to allow verifiable outsourcing of polynomial calculations to enable secure set intersection [14], to have tighter bounds on computational and/or communication efficiency [34, 39].These techniques can be powerful, but a major drawback is that they only consider the two party setting and generally require multiple rounds of communication (i.e. multiple messages must be sent each time-interval).

Secure/Privacy-preserving Stream Polynomial Evaluation: There is existing work that supports private polynomial calculations [17, 18]. However, individual product terms are disclosed to the aggregator in some work [17], and they generally rely on the DDH assumption and are not quantum secure. Also, existing works have limited scalability, and their polynomial degree is limited to a constant [13], otherwise the communication overhead is prohibitively large. Some approaches also suffer from high communication round complexity and low scalability [10, 12, 26]. A more general approach exists [4], however it is also an interactive protocol with high communication round complexity.

Our work is more general than standard PSA, and can be used to compute any function that can be written as a polynomial. It avoids the drawbacks of the previous approaches by combining PSA and a TEE, to maximize efficiency while supporting polynomial evaluation. We summarize our findings in Figure 1.

4 Preliminaries

PSA Adversary Model: In general, PSA schemes are designed to allow an untrusted third party (the aggregator) to perform aggregation computation while providing semantic security to data sent by users. We consider a slightly different adversary model than what is standard in PSA. The users have the same role as before, and send ciphertexts to an aggregator, but in our work the aggregator is equipped with a TEE. We assume that all users may collude with each other and/or the aggregator, although the TEE is trusted. We want to guarantee that the aggregator cannot learn any individual input from any honest user (this implies if an aggregator corrupts or colludes with a user to learn their input, this does not impact the privacy of the honest users). All the aggregator can learn is the output of the function. Standard aggregator obliviousness [2,16,18,30], which states the aggregator and colluders learn only the final aggregation outcome and what can be inferred from their inputs, is guaranteed. More specifically, we consider the case of a set of n users and a single aggregator A. Each user $u_i \in S$ where $0 < i \leq n-1$ possesses a piece of data $x_{i,ts}$, corresponding to some timestamp ts. The users wish to calculate an aggregation function f over the private values they send. PSA is 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'.
- $Enc(parms, x_{i,ts}, s_i, ts, \cdots)$: 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. 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 ts, and the n time-series ciphertexts from the users (with timestamp ts). Returns $y_{ts} = x_{0,ts} + x_{1,ts} + \cdots + x_{n-1,ts}$.

Users will run Enc on their data, and send their results $c_{i,ts}$ to the aggregator. Then the aggregator calls Agg on the ciphertexts $c_{0,ts}, \cdots c_{n-1,ts}$ it has collected to learn the aggregation result y_{ts}. Note we sometimes omit the timestamp notation moving forward for clarity when the context is clear. In PSA schemes, the algorithm $Setup$ is run in a trusted manner [2], via the use of an additional trusted third party, secure hardware, or secure multiparty computation. Informally, we wish to require that an adversary able to compromise the aggregator and any number of other users is unable to learn any new information about uncompromised users' data. This idea is known as *aggregator obliviousness*, and the standard definition [2,30] is stated below:

Definition 1. *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 π is aggregator oblivious [2,30] 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 $Enc(parms, sk_i, ts, x_{i,ts}, r_{i,ts})$ to the adversary.*

***Compromise**: The adversary argues $i \in [0, n) \cup \{\zeta\}$. If $i = \zeta$, 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 $c_{i,ts} = Enc(parms, s_i, ts, x_{i,ts}, r_{i,ts})$ 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. The adversary wins if they can correctly guess bit b chosen during the Challenge.*

Trusted Execution Environment: Note that our framework can work with any form of Trusted Execution Environment (TEE), but we chose the Intel SGX for our concrete instantiation. Intel SGX is a set of new CPU instructions that can be used by applications to set aside private regions of code and data. It allows developers to (among other things) protect sensitive data from unauthorized access or modification by malicious software that may be running at superior privilege levels. To do this, the CPU protects an isolated region of memory called Processor Reserved Memory (PRM) against other non-enclave memory accesses, including the kernel, hypervisor, etc.. Sensitive code and data is encrypted and stored as 4KB pages in the Enclave Page Cache (EPC), a region inside the PRM. Even though EPC pages are allocated and mapped to frames by the OS kernel, page-level encryption guarantees confidentiality and integrity. In addition, to provide access protection to the EPC pages, the CPU maintains an Enclave Page Cache Map (EPCM) that stores security attributes and metadata associated with EPC pages. This allows for strong privacy and integrity guarantees if applications can be written in a two part model [11, 20].

Applications must be split into a secure part and a non-secure part. The application can then launch an enclave, which is placed in protected memory, that allows user-level code to define private segments of memory, whose contents are protected and unable to be read or saved by any process outside the enclave. Enclave entry points are defined during compilation. The secure execution environment is part of the host process, and the application contains its own code, data, and the enclave, but the enclave contains its own code and data too. An enclave can access its application's memory, but not vice versa, due to a

combination of software and hardware cryptographic primitives. Only the code within the enclave can access its data, and external accesses are always denied. The enclave is decrypted "on the fly" only within the CPU itself, and only for code and data running from within the enclave itself. This is supported by an autonomous piece of hardware called the Memory Encryption Engine (MEE) that protects the confidentiality and integrity of the CPU-DRAM traffic over a specified memory range. Code running within the enclave is therefore protected from being "spied on" by other code. Although the enclave is trusted, no process outside it needs to be trusted, including the operating system [11, 20]. Before performing computation on a remote platform, a user can verify the authenticity of the trusted environment. By using the attestation mechanism, users can establish that software is running on an Intel SGX enabled device inside an enclave.

Lattice-Based Cryptography: Our framework utilizes the complex canonical embedding (CCE) [8], to support privacy-preserving polynomial evaluation on floating-point data. The CCE and the inverse of it allows one to map a polynomial ring element to a vector of complex numbers and vice versa, and this mapping is an isometric ring homomorphism, making it possible to encode complex numbers into a quotient ring of polynomials. Thus, it is frequently used in the lattice-based cryptography using polynomial rings, as complex-number inputs can be encrypted with the CCE. As such, we anticipate our framework will be most useful in conjunction with lattice-based PSA schemes. In general, lattice-based cryptography has recently generated significant interest among cryptography researchers, as it is quantum secure and generally faster than more traditional approaches (RSA, etc.) due to its shorter operands and other recent optimizations. With large coefficients, Residue Number System (RNS) representations can be used to break large numbers down into smaller components. Using Single Instruction Multiple Data (SIMD) optimizations allows multiple plaintexts to be encoded into a single ciphertext. Large polynomial degrees can make polynomial multiplication very expensive, and to mitigate this the Number-Theoretic Transform (NTT) can be used to decrease the theoretical complexity [8, 31]. Full-RNS variants of lattice-based cryptosystems reduce the complexity of the cryptosystems' most expensive operations to the complexity of the NTT [8]. The Ring Learning with Errors (RLWE) problem is frequently used as a hardness assumption when designing lattice-based cryptosystems, and we give an overview of it below. Note that we use boldface lowercase letters to denote elements of rings. Consider two coprime numbers q, p, with $q \gg p$, and let $\mathbf{s}$ be a random element of R_q with coefficients bounded by b (b is often 1), where R is the quotient ring of $\mathbb{Z}[X]/\Phi(X)$, and $\Phi(X)$ is the $M = 2N$-th cyclotomic polynomial with degree $N = 2^d$ for some positive integer d, such that $R_t = \mathbb{Z}_t[X]/\Phi(X)$, is the ring with all coefficients in $\mathbb{Z}_t$. We 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 $\mathbb{Z}_t = [\frac{-t}{2}, \frac{t}{2}) \cap \mathbb{Z}$; when centered modular reduction is applied coefficientwise to ring elements we write $[\mathbf{a}]_t \in R_t$. 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.

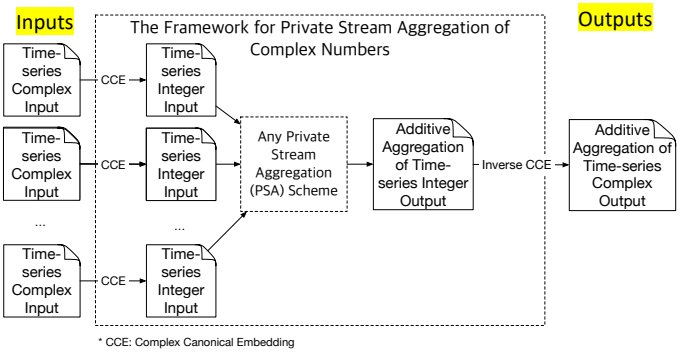

Fig. 2. The Framework for Complex-Number PSA

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. $\mathbf{b}_i = [\mathbf{a}_i \cdot \mathbf{s}_i + p'\mathbf{e}_i]_q$ with $p' \in \{1, p\}$, 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, without any knowledge of $\mathbf{s}_i$ or $\mathbf{e}_i$. The RLWE problem is believed to be intractable for quantum computers; its difficulty comes from reduction to the Shortest Vector Problem [22]. The difficulty of the RLWE problem is parameterized by q and N. Larger values of q provide more utility for RLWE-based cryptosystems, but decreases the difficulty of the RLWE problem. Note increasing N also increases the difficulty of the RLWE problem and thus the overall security.

5 Our Framework

We enable the stream polynomial evaluation via a composition of additive PSA and CCE, and address its partial leakages with a TEE. It is extremely challenging to apply an approach that uses RLWE terms as the computationally indistinguishable random elements to design multiplicative PSA with exact aggregation [2,31]. One reason among others is that the RLWE term is inherently additive, i.e., the error term $\mathbf{e}_{i,ts}$ is added instead of multiplied in the term $\mathbf{a}_{ts}\mathbf{s}_i + \mathbf{e}_{i,ts}$, making it challenging to cancel out the random terms if the ciphertexts are multiplied together at the aggregator's side. It should be said that traditional PSA schemes are defined over integers, but by leveraging the CCE [8] we can transform any PSA taking quotient rings of polynomials as the plaintext space (e.g. any lattice-based PSA schemes based on the RLWE problem [2, 23, 31]) into PSA defined over floating point numbers. We summarize our framework for transforming any PSA scheme operating over integers to operate over floating point numbers in Figure 2.

The nature of PSA makes it possible to execute only a small constant amount of computations inside a TEE, which minimizes the performance impact. We achieve PSA for the multivariate polynomial f by composing the additive PSA with a TEE so that each user i provides a ciphertext corresponding to their private data $\{\mathbf{m}_{i,j,ts}^{e_{i,j}}\}_{j=1}^{z}$ and the aggregator can multiply each product term $\prod_{i \in S} \mathbf{m}_{i,j,ts}^{e_{i,j}}$

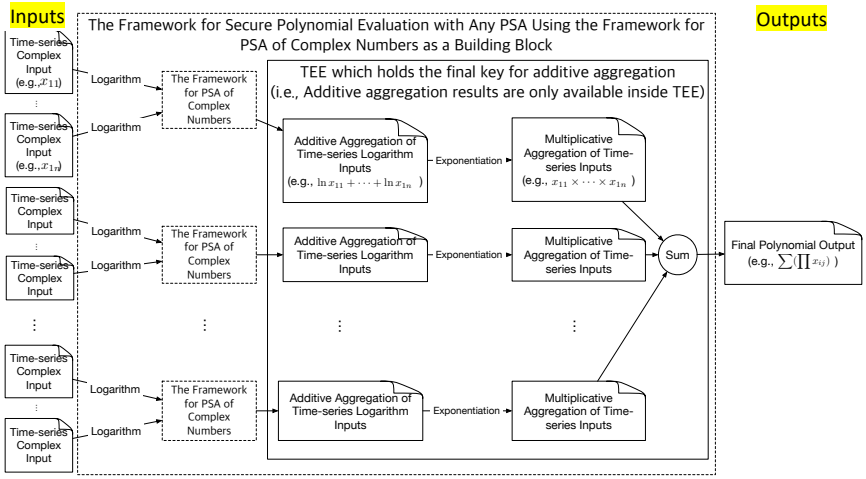

Fig. 3. The Framework of Polynomial Evaluation with Complex-Number PSA (using the Framework for PSA of Complex Numbers, Figure 2, as a building block)

for $j = 1$ to z. Note that, due to the SIMD technique, each user can pack all z input values for the z product terms into one plaintext polynomial, and the aggregator only needs to perform one multiplicative aggregation to get the outcome of z individual products. More specifically, we rely on the following technique to build our multiplicative PSA: For each input $\mathbf{m}_{i,j,ts} \in \mathbb{C}$ of i-th user, let the user calculate the natural logarithm of the input and encode it into a polynomial as $\mathbf{m}'_{i,ts} = \mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))$ using the complex canonical embedding [8]. Then, the nearly-exact additive PSA is leveraged to let the aggregator compute $\sum_{i \in S} \mathbf{m}'_{i,ts}$ with negligible error terms. We then undo the complex canonical embedding to recover $\sum_{i \in S} \ln(\mathbf{m}_{i,j,ts}) = \ln(\prod_{i \in S} \mathbf{m}_{i,j,ts})$, and a natural exponential function can be computed to get $\prod_{i \in S} \mathbf{m}_{i,j,ts}$. Due to the limitation of the multiplicative PSA, we are limited to nearly-exact aggregation only, i.e., the outcome is exact up to the pre-defined precision only. Then, the aggregator can locally calculate $f(\{\mathbf{m}_{i,j,ts}\}_{i \in S, j=1,\ldots,z})$ using the public parameters c_j, where S is the set of users whose ciphertexts are received by the aggregator. Such an approach guarantees correct aggregation up to the precision of the outcome, however, the aggregator learns all individual product terms which may not be acceptable especially when the product terms are correlated.

Although such additional knowledge does not always lead to complete disclosure of individuals' inputs, the search space can be reduced by leveraging such knowledge. Thus, the proposed PSA above fails to achieve the *aggregator obliviousness* [2, 30] that states the aggregator should learn only the final output. To address this, we adopt the idea of a *one-time program* [15] that leverages trusted hardware implementations to prevent the leakages similar to the one above. Namely, we let the aggregator deploy a TEE, e.g,. Intel SGX and leverage its secure functionality to prevent the aggregator from receiving more information than the final result. The memory encryption and isolation of the TEE guarantees that operating systems cannot view or change the program/data within the TEE.

A naïve way to prevent the aforementioned leakages is to let the aggregator perform the aggregation within the TEE. Then, even though the multiple aggregation results are calculated for many different subsets, the final outcome resides inside the TEE only, and the program running in it (which is verified by all users through remote attestation) can decide to output the appropriate result(s) to the outside of the TEE, i.e., the aggregator. In the case of stream polynomial evaluation, the outcome of the multiplicative aggregation, i.e., the individual product terms, resides in the TEE, and the remotely verified program running in the TEE computes and returns only the sum of the product terms to the aggregator. Though being secure, such a method is more complicated. One can simply let users set up secure communication channels with the aggregator's TEE (by exchanging the keys) and let users send their input data to the TEE who performs arbitrary aggregation within the TEE securely.

We design a method to integrate the TEE into the PSA such that users benefit from the security guarantees of TEE while the overhead at the aggregator's end is much smaller than the overhead of the entire raw data being sent to the TEE and aggregated inside the TEE. Note that if only one or a few constant number of user ciphertexts are sent to the TEE and the rest, which are sent to the aggregator, are aggregated outside the TEE, the aggregator only observes the incomplete aggregation results which are indistinguishable from random elements due to the security of the PSA (e.g., randomness of the RLWE terms [2, 31]). After the aggregation of the ciphertexts outside the TEE is finished, the aggregator can send the aggregated incomplete results into the TEE who continues the aggregation inside the TEE, at which point only a constant number of operations need to be performed since only a few operations are needed inside the TEE. Considering that the TEE introduces the extra overhead of memory encryption for every communication between the CPU and the DRAM, the PSA with our optimization has higher throughput than the plain aggregation performed entirely within the TEE especially when the scale of the aggregation is large. Recall it is known that some TEEs (e.g., Intel SGX) have difficulties exploiting multi-threading [35] due to the lack of common synchronization primitives, and leveraging threading within TEEs can introduce security vulnerabilities [37]. Also, TEEs have been shown to run common functionalities over an order of magnitude slower than what can be achieved on comparable untrusted hardware, due to the overhead of computing within the enclave [35], and performing a large number of context switches to send each user's data into the TEE can add serious overhead, especially in a big data setting. Overall performance can be improved if we minimize the number of context switches and outsource computationally expensive steps to an untrusted space that can better leverage parallel computing. More specifically, we aggregate all of the users' ciphertexts outside the enclave, and only perform a single context switch to send this intermediate result into the enclave, where we add the aggregator's secret key to recover the product terms of the polynomial. We later calculate the sum of the calculated products, so we learn the final output inside the TEE. We summarize the data flow in our framework

for transforming any PSA scheme into a secure stream polynomial evaluation scheme in Figure 3, and we formalize it with the following 3 algorithms:

- $Cryptonomial.Setup(\lambda, \cdots)$: All users perform attestation on the aggregator's TEE, and input to the TEE 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. The TEE returns a set of parameters $parms$, users' secret keys $k_i, i \in [0, n)$ (over a secure channel), and the aggregation key k'.
- $Cryptonomial.Enc(parms, \mathbf{m}_{i,ts}, k_i, ts, \cdots)$: Takes the scheme's parameters, and a user's secret key k_i and vector of time-series input $\mathbf{m}_{i,j,ts}$, along with a timestamp ts. Returns $\mathbf{c}_{i,ts}$, an additively homomorphic encryption of $\mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))$, the natural logarithm of the user's vector of noisy inputs encrypted under their secret key, where the natural log is taken component-wise over the vector, and **CCE** is the complex canonical embedding function.
- $Cryptonomial.Agg(parms, k', ts, \mathbf{c}_{0,ts}, \cdots \mathbf{c}_{n-1,ts})$: Takes the scheme's parameters, the aggregation key, a timestamp, and the n time-series ciphertexts from the users (with timestamp ts). In the untrusted space compute $\mathbf{y}_{ts} = \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}$ via homomorphic addition. Then send $\mathbf{y}_{ts}$ into the TEE and add in the aggregation key k' as appropriate based on the underlying PSA scheme. Then within the TEE take the inverse of the **CCE** of this as $\sum_{i=0}^{n-1} \mathbf{CCE}^{-1}(\mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))) = \sum_{i=0}^{n-1} \ln(\mathbf{m}_{i,j,ts}) = \ln(\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts})$. They then take the exponential to recover $\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts}$ and recover $\sum_{j=1}^{z} c_j(\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts}^{e_{i,j}})$, where z is the number of product terms in the polynomial, and c_j's and $e_{i,j}$'s are public parameters.

6 Framework Instantiation with Existing PSA

PSA chosen for instantiation: Our scheme can leverage any PSA amenable to a complex canonical embedding as a building block. There are several such schemes [1, 2], but we chose the noise-scaled variant of SLAP (i.e. $SLAP_{NS}$, [31]) as a building block for its simplicity and open-source implementation. Before describing our protocol, we review the SLAP protocol below. Note, in the scheme operands are ring elements, not matrices or vectors. We denote the plaintext domain as the ring R_t and the ciphertext domain as the ring R_q, with $q \gg t$ and an appropriate value of the polynomial modulus degree N to allow for the necessary security. Secret keys and error terms are drawn from distributions χ, ζ (1-bounded in practice) on R_q. The scheme is defined as follows:

- $SLAP_{NS}.Setup(\lambda, t, n)$: 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 $\mathbb{Z}_q$. Choose a set of public keys $\{\mathbf{a}_{ts}\}$ uniformly at random, or a method of generating keys indistinguishable from such. Choose users' secret keys $\mathbf{s}_0 \cdots \mathbf{s}_{n-1}$ from χ. Construct the aggregator's key

as $\mathbf{s}' = -[\sum_{i=0}^{n-1} \mathbf{s}_i]_q$. Return $parms = (R_q, t, n, \{\mathbf{a}_{ts}\})$, the users' secret keys $\mathbf{s}_i$, and the aggregation key $\mathbf{s}'$.
- $SLAP_{NS}.Enc(parms, \mathbf{s}_i, \mathbf{m}_{i,ts} \in R_t, ts)$: Choose the user's error $\mathbf{e}_{i,ts}$ from ζ. Return the user's ciphertext $\mathbf{c}_{i,ts} = [\mathbf{a}_{ts} \cdot \mathbf{s}_i + t\mathbf{e}_{i,ts} + \mathbf{m}_{i,ts}]_q$ (based upon the secret key, the user's input, a small random error, and the timestamp ts).
- $SLAP_{NS}.Agg(parms, \mathbf{s}', ts, \mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts})$: If any of $\mathbf{c}_{0,ts} \cdots \mathbf{c}_{n-1,ts}$ are absent or not well-formed (i.e., an element of R_q), then abort. Otherwise, compute and return $\mathbf{y}_{ts} = [[\mathbf{a}_{ts} \cdot \mathbf{s}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q]_t$

Our Instantiated Protocol (τ): We now present the concrete instantiation of our scheme. Let **CCE** be the complex canonical embedding described in [8], which is an isometric ring homomorphism that preserves the magnitude of the elements, to encode complex numbers into polynomials. We assume the set of users u_i perform remote attestation with the aggregator A's TEE, and the polynomial function is agreed upon beforehand. We model our system in Figure 3. The protocol instantiated with our framework and the building block SLAP is denoted as τ and defined as follows:

- $\tau_{Setup}(\lambda, t, n)$: Inside the TEE, call $SLAP_{NS}.Setup(\lambda, t, n)$. The secret keys $\mathbf{k}_0 \cdots \mathbf{k}_{n-1}$ and the relevant parameters are then distributed to their owners over secure channels, and the aggregator's key $\mathbf{k}'$ remains inside the TEE.
- $\tau_{Enc}(parms, \mathbf{k}_i, \mathbf{m}_{i,j,ts}, ts)$: Note in this functionality each user determines their private values $\mathbf{m}_{i,j,ts} \in R_t$ they wish to send for a given time stamp ts, and encrypt it as follows: First, take the natural logarithm of their inputs as $\ln(\mathbf{m}_{i,j,ts})$, and apply the complex canonical embedding over this as $\mathbf{m}'_{i,ts} = \mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))$. Finally they encrypt this as $\mathbf{c}_{i,ts} = SLAP_{NS}.Enc(parms, \mathbf{s}_i, \mathbf{m}'_{i,ts}, ts)$. Then each u_i sends their $\mathbf{c}_{i,ts}$ to A.
- $\tau_{Agg}(parms, \mathbf{k}', ts, \mathbf{c}_{i,ts} \cdots \mathbf{c}_{n-1,ts})$: In the untrusted space A computes $\mathbf{y}_{ts} = [\sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q$. Then they send $\mathbf{y}_{ts}$ into the TEE, and inside they compute $[[\mathbf{y}_{ts} + \mathbf{a}_{ts} \cdot \mathbf{k}']_q]_t = \sum_{i=0}^{n-1} \mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))$. Then within the TEE they take the inverse of the **CCE** of this as $\sum_{i=0}^{n-1} \mathbf{CCE}^{-1}(\mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts}))) = \sum_{i=0}^{n-1} \ln(\mathbf{m}_{i,j,ts}) = \ln(\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts})$. They then take the exponential to recover $\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts}$ and compute $\sum_{j=1}^{z} c_j(\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts}^{e_{i,j}})$, where z is the number of product terms in the polynomial, and c_j's and $e_{i,j}$'s are public parameters.

Correctness: This protocol is correct, since we know that when adding n ciphertexts $\mathbf{c}_{i,ts}$, we find $[\mathbf{a}_{ts} \cdot \mathbf{k}' + \sum_{i=0}^{n-1} \mathbf{c}_{i,ts}]_q = [\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts})))]_q$. 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 as long as $\frac{3 \cdot n \cdot t}{2} < \frac{q}{2}$, $\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{CCE}(\ln(\mathbf{m}_{i,j,ts})))$ does not overflow modulo q, guaranteeing correctness. Then reducing $\sum_{i=0}^{n-1}(t\mathbf{e}_{i,ts} + \mathbf{m}_{i,j,ts})$ modulo t removes the error terms, leaving us with the sum of the users' inputs modulo t. Note $\sum_{i=0}^{n-1} \ln(\mathbf{m}_{i,j,ts}) = \ln(\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts})$, so exponentiating recovers $\prod_{i=0}^{n-1} \mathbf{m}_{i,j,ts}$.

Security: Although a formal proof of aggregator obliviousness is in the appendix for completeness, note by the underlying security of SLAP [31], the RLWE problem [22], and the TEE, the protocol is secure. Even if the untrusted aggregator colludes with some malicious users, although they can learn the individual inputs of the malicious users, since they only receive the aggregated function output as a final result, they cannot learn which honest user inputted which value, provided there is more than one honest user. Similarly, since each term in the polynomial is calculated inside the TEE, there is no partial leakage. Post-quantum security follows from SLAP [31], and from the underlying post-quantum security of AES encryption [11], which is used by the Intel SGX to encrypt data in the enclave. Note we assume quantum secure signatures are used during attestation, a forthcoming future feature of Intel SGX [5]. We can easily guarantee differential privacy for our protocol using existing techniques if necessary [2, 31].

Parallel-Friendliness: Note that the computation of the product terms is perfectly parallelizable (except for one operation, adding the aggregator's secret key, which must be done inside the enclave) and thus can be outsourced to many-core hardware. Existing work [32] notes the DDR4 specification gives a peak data transfer rate of 25,600 MB/s, which gives $70\mu s$ seconds per ciphertext transfer time from DRAM to the hardware used for parallelization. In practice, the overhead from data transfer can be significantly less, due to pipelining and interleaving of execution and data transfer. They estimate $21.2\mu s$ seconds per ciphertext transfer time from DRAM to the hardware after observing the time difference between operating upon ciphertexts that were/were not resident in cache memory [32].

7 Experimental Evaluation

To better understand the improvements gained in performance we implemented our scheme using C++11, and version 2.10 of the Intel SGX SDK and present our results below (code available at: `https://anonymous.4open.science/r/ea7619a6-3c77-483f-86c0-2ba60068ea54/`). For our PSA backend we used the open-source implementation of SLAP's noise-scaled variant [31], which uses optimizations including RNS, SIMD batching, and NTT, which are discussed in Chapter 4. We used SLAP's default parameters; security parameter $\lambda = 128$, polynomial modulus degree $N = 1024$, and ciphertext modulus q with 56 bits. Our experiments were run on a computer running Ubuntu 18.04 with an Intel(R) Xeon(R) W-1290P 3.70GHz CPU with 10 cores, 20 threads, 128 GB of memory, and Intel SGX support. We did not leverage GPUs/FPGAs because we did not have access to computers equipped with both Intel SGX and GPUS/FPGAs. Our tests took average runtimes of 5 trials.

Benchmarks: To benchmark our protocol, we computed polynomials of the form $\sum_{j=1}^{2} \prod_{i=1-j+1}^{n-j+1} \mathbf{m}_{i,j,ts}$ and report the time for each step below. Achieving accurate timings for operations within the enclave is difficult, because the SGX primitive `sgx_get_trusted_time` only supports second level precision, but many operations can be computed at millisecond precision (Intel is committed to

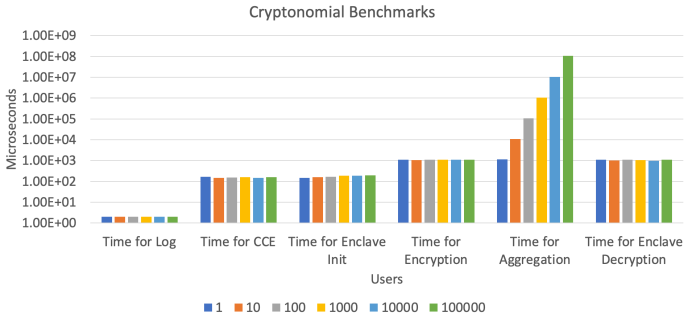

Fig. 4. Benchmarks of Cryptonomial using SLAP as the PSA

providing better timing support in future releases). To measure benchmarks, we used C++ `std::chrono`, which supports microsecond precision, and timed the overall time to compute ECALLs within the untrusted component of the program. We report how these times scale as we increase the number of users in Figure 4. In general, we find that preprocessing time is linear in the number of users, and is ≤ 1 millisecond for 100,000 users (excluding network latency). Also, encryption and decryption time is linear in the number of users, and is approximately 1 millisecond per user. Aggregation time is logarithmic in the number of users, but is still practical in large scale computation.

Case Study: To better understand how our technique performs in a real world setting, we implemented multiple linear regression analysis using Cryptonomial, and compared it to the performance reported by the current state-of-the-art privacy-preserving polynomial evaluation technique known as PDA [18]. The linear regression model consists of one equation of linearly increasing variables (also called parameters or features) along with a coefficient estimation algorithm called least squares, which attempts to determine the best possible coefficient given a variable. Multiple linear regression is a model that can capture the linear relationship between multiple variables and features, assuming that there is one. The multiple linear regression formula is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_i x_i + \varepsilon$, where β_0 is known as the intercept, β_1 to β_i are known as coefficients, x_1 to x_i are the features of the dataset, and ε are the residual terms.

We can also represent the formula for linear regression in vector notation. Linear least squares (LLS) is the main algorithm for estimating coefficients of the formula just shown. We use the most popular variant called ordinary least squares (OLS). The OLS algorithm minimizes the sum of squares of residuals. The following formula ensures that the resulting coefficients define a minimum for the normal equation, which means that the result is the minimized total sum of squared residual: $\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$. Here $\hat{\boldsymbol{\beta}}$ is a vector containing all of the coefficients that can be used to make predictions by using the formula presented in the beginning for multiple linear regression. We simulated training a linear regression model over the datasets in a privacy-preserving manner using our scheme (Section 4.5) with data from the UCI Machine Learning Database as done in PDA [18]. We measured the time to complete the training in a local computer and our times are reported in Table 1. By utilizing batching and other

Table 1. Cryptonomial OLS Performance on UCI Dataset

Datasets	Records	Features	Our Time	PDA Time	Speedup
Census	48,842	14	1.74 s	355 s	204x
Bank	45,211	17	1.63 s	341 s	209x
Insurance	9,822	14	0.42 s	74 s	176x
White wine	4,898	11	0.25 s	33 s	132x
Red wine	1,599	11	0.14 s	12 s	85x

optimizations in SLAP, data for all features was encoded into a single ciphertext, significantly reducing computation and communication overhead.

We note that our technique is always the fastest by at least an order of magnitude. This makes sense as our lattice-based PSA cryptographic primitive combined with a TEE as discussed in our framework is considerably less computationally expensive than the ECC-based techniques of PDA. Also, PDA makes use of an interesting but expensive ECC-based encoding procedure that allows for a form of fault tolerance, where users can be dynamically added or dropped from the system. This encoding negatively impacts the overall run time, and this trend continues as we increase the number of records in the final aggregation calculation. We note that although this paper does not consider fault tolerance, there are preexisting techniques that leverage a TEE to transform any traditional PSA scheme into a fault tolerant PSA scheme [20], and such techniques can be incrementally deployed on our solution. Thus, we can conclude that in secure polynomial evaluation scenarios where aggregation times greatly impact the overall performance, our method offers the best efficiency.

8 Conclusion

We presented Cryptonomial, a framework for converting any PSA scheme amenable to a CCE into a secure computation protocol that can compute any function that can be written as a polynomial, by combining PSA and a TEE. We showed that Cryptonomial meets the security and privacy requirements of PSA, and supports strong security guarantees. Simulations show our scheme's performance is orders of magnitude faster than similar work supporting polynomial calculations.

Acknowledgements

This work was supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via contract #2020-20082700002. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

1. Abdallah, A., Shen, X.S.: A lightweight lattice-based homomorphic privacy-preserving data aggregation scheme for smart grid. IEEE Transactions on Smart Grid **9**(1), 396–405 (2016)

2. Becker, D., Guajardo, J., Zimmermann, K.H.: Revisiting private stream aggregation: Lattice-based psa. In: NDSS (2018)
3. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: CCS. pp. 578–590. ACM (2016)
4. Blanton, M.: Achieving full security in privacy-preserving data mining. In: SocialCom. pp. 925–934. IEEE (2011)
5. Boneh, D., Gueron, S.: Surnaming schemes, fast verification, and applications to sgx technology. In: CT-RSA. pp. 149–164. Springer (2017)
6. Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: IEEE Symposium on Security and Privacy. pp. 538–552. IEEE (2012)
7. Chang, Y.C., Lu, C.J.: Oblivious polynomial evaluation and oblivious neural learning. In: Asiacrypt. pp. 369–384. Springer (2001)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Floating-point homomorphic encryption. IACR Cryptology ePrint Archive **2016**, 421 (2016)
9. Chowdhury, M.E., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M.A., Mahbub, Z.B., Islam, K.R., Khan, M.S., Iqbal, A., Al Emadi, N., et al.: Can ai help in screening viral and covid-19 pneumonia? IEEE Access **8**, 132665–132676 (2020)
10. Cianciullo, L., Ghodosi, H.: Efficient information theoretic multi-party computation from oblivious linear evaluation. In: IFIP. pp. 78–90. Springer (2018)
11. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)
12. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Secure efficient multiparty computing of multivariate polynomials and applications. In: ACNS. pp. 130–146. Springer (2011)
13. Franklin, M., Mohassel, P.: Efficient and secure evaluation of multivariate polynomials and applications. In: ACNS. pp. 236–254. Springer (2010)
14. Hazay, C.: Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. Journal of Cryptology **31**(2), 537–586 (2018)
15. Järvinen, K., Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In: CHES. pp. 383–397. Springer (2010)
16. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: ICFCDS. pp. 111–125. Springer, Berlin, Germany (2013)
17. Jung, T., Mao, X., Li, X., Tang, S., Gong, W., Zhang, L.: Privacy-preserving data aggregation without secure channel: multivariate polynomial evaluation. In: IEEE INFOCOM (2013)
18. Jung, T., Han, J., Li, X.Y.: Pda: Semantically secure time-series data analytics with dynamic user groups. TDSC **15**(2), 260–274 (March 2018)
19. Karl, R., Burchfield, T., Takeshita, J., Jung, T.: Non-interactive mpc with trusted hardware secure against residual function attacks. In: SecureComm. pp. 425–439. Springer (2019)
20. Karl, R., Takeshita, J., Jung, T.: Cryptonite: A framework for flexible time-series secure aggregation with online fault tolerance. IACR Cryptol. ePrint Arch. **2020**, 1561 (2020)
21. Lu, W., Kawasaki, S., Sakuma, J.: Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. IACR Cryptol. ePrint Arch. **2016**, 1163 (2016)
22. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 1–35 (2013)
23. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. In: Eurocrypt. pp. 35–54. Springer (2013)
24. Maglogiannis, I., Loukis, E., Zafiropoulos, E., Stasis, A.: Support vectors machine-based identification of heart valve diseases using heart sounds. Computer methods and programs in biomedicine **95**(1), 47–61 (2009)
25. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM Journal on Computing **35**(5), 1254–1281 (2006)

26. Özarar, M., Özgit, A.: Secure multiparty computation via oblivious polynomial evaluation. In: Theory and Practice of Cryptography Solutions for Secure Information Systems, pp. 253–278. IGI Global (2013)
27. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: ACM SIGMOD ICM. pp. 735–746 (2010)
28. Sen, A., Srivastava, M.: Regression analysis: theory, methods, and applications. Springer Science & Business Media (2012)
29. Sethy, P.K., Behera, S.K., Ratha, P.K., Biswas, P.: Detection of coronavirus disease (covid-19) based on deep features and support vector machine. Arxiv Preprint (2020)
30. Shi, E., Chan, T.H., Rieffel, E., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: Proc. NDSS. vol. 2, pp. 1–17. Citeseer (2011)
31. Takeshita, J., Karl, R., Gong, T., Jung, T.: Slap: Simple lattice-based private stream aggregation protocol. Arxiv Preprint (2020)
32. Takeshita, J., Reis, D., Gong, T., Niemier, M., Hu, X.S., Jung, T.: Algorithmic acceleration of b/fv-like somewhat homomorphic encryption for compute-enabled ram. In: SAC. Springer (2020)
33. Thurston, R.C., Matthews, K.A., Hernandez, J., De La Torre, F.: Improving the performance of physiologic hot flash measures with support vector machines. Psychophysiology **46**(2), 285–292 (2009)
34. Tonicelli, R., Nascimento, A.C., Dowsley, R., Müller-Quade, J., Imai, H., Hanaoka, G., Otsuka, A.: Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. IJIS **14**(1), 73–84 (2015)
35. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. ICLR (2018)
36. Valovich, F., Aldà, F.: Computational differential privacy from lattice-based cryptography. In: NutMiC. pp. 121–141. Springer (2017)
37. Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R.: Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves (2016)
38. Yu, W., Liu, T., Valdez, R., Gwinn, M., Khoury, M.J.: Application of support vector machine modeling for prediction of common diseases: the case of diabetes and pre-diabetes. BMC medical informatics and decision making **10**(1), 16 (2010)
39. Zhu, H., Bao, F.: Augmented oblivious polynomial evaluation protocol and its applications. In: ESORICS. pp. 222–230. Springer (2005)

A Proof

We adapt existing proofs [2, 30] for our own protocol.

Theorem 1. *(Aggregator Obliviousness Security): Let the output of Cryptonomial.Enc be indistinguishable from random. Then the instantiation of our framework is secure under aggregator obliviousness.*

Proof. Our goal is 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 our protocol.

A Slightly Modified Game: For the proof, we modify the game of aggregator obliviousness as follows: First, we change any Encrypt query to be 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. However, in our proof the adversary chooses

one plaintext ($\mathbf{x}_i$) and they must distinguish between valid encryptions of ($\mathbf{x}_i$) or random values. Note that any adversary with greater than negligible advantage in winning this modified game would also win the original game of aggregator obliviousness with more than negligible advantage [2, 30]. Thus, we must show that 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, to solve the decisional RLWE problem.

For simplicity, we consider the protocol's operation at a single timestamp ts, and use $\mathbf{A}$ to denote an element of R_t chosen uniformly at random, or via a method of generating keys indistinguishable from such. We also omit the timestamp identifier of plaintexts, ciphertexts, and other variables. Recall that aggregator obliviousness acknowledges the case where the adversary compromises all participants but one, and allows that an adversary may inevitably learn the secret key of that participant and therefore distinguish between valid encryptions and random values. Because of this, the definition requires that adversaries 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 the $SLAP_{NS}$ cryptosystem, $\mathcal{B}$ 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 $\mathbf{s}^*$ 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 $j, k \in [0, n) \cup \{\zeta\}$. Note the probability $\mathcal{A}$ will not select these parties to be compromised is $\frac{1}{n^2}$. 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 know either of $\mathbf{s}_k, \mathbf{s}_j$, which are the aggregation scheme's secret keys for users j, k. After this, $\mathcal{B}$ 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 $\mathcal{A}$, otherwise we abort. Also, if $i = \zeta$, then $\mathbf{s}'$ will be returned to $\mathcal{A}$.

In the Challenge phase, $\mathcal{A}$ will choose a set of uncompromised users $U \subseteq [0, n) \setminus \{j, k\}$, to send plaintexts $\{(\mathbf{x}_i)\}$ 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 $\{\mathbf{c}_i = \tau_{Enc}(parms, \mathbf{s}_i, \mathbf{x}_i)\}$ for $i \in U \setminus \{j, k\}$.

Now $\mathcal{B}$ enters the Distinguish phase and sends $\mathbf{m}_k = [\mathbf{x}_k]_q$ 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 plaintexts, with $\mathbf{v} = [\sum_{i \in U} \mathbf{x}_i]_q$ and $\mathbf{c}_j = FHE.Enc(parms, \mathbf{A}, \mathbf{v})$, where FHE is the backend cryptosystem used by $SLAP$. 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 $\mathbf{x}_k$ 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 $SLAP_{NS}$ scheme being used. Therefore, the scheme achieves aggregator obliviousness and all the adversary can learn is what can be inferred based on the output of the the final polynomial calculation and inputs they control, since the ciphertexts seen outside the enclave are semantically secure and the final decryption and aggregation done in the enclave is secure against eavesdropping due the TEE's strong isolation. This completes the proof. □