CryptoGram: Fast Private Calculations of Histograms over Multiple Users' Inputs

Ryan Karl, Jonathan Takeshita, Alamin Mohammed, Aaron Striegel, Taeho Jung
Department of Computer Science and Engineering, University of Notre Dame
Notre Dame, Indiana, USA
{rkarl,jtakeshi,amohamm2,striegel,tjung}@nd.edu

***Abstract*—Histograms have a large variety of useful applications in data analysis, e.g., tracking the spread of diseases and analyzing public health issues. However, most data analysis techniques used in practice operate over plaintext data, putting the privacy of users' data at risk. We consider the problem of allowing an untrusted aggregator to privately compute a histogram over multiple users' private inputs (e.g., number of contacts at a place) without learning anything other than the final histogram. This is a challenging problem to solve when the aggregators and the users may be malicious and collude with each other to infer others' private inputs, as existing black box techniques incur high communication and computational overhead that limit scalability. We address these concerns by building a novel, efficient, and scalable protocol that intelligently combines a Trusted Execution Environment (TEE) and the Durstenfeld-Knuth uniformly random shuffling algorithm to update a mapping between buckets and keys by using a deterministic cryptographically secure pseudorandom number generator. In addition to being provably secure, experimental evaluations of our technique indicate that it generally outperforms existing work by several orders of magnitude, and can achieve performance that is within one order of magnitude of protocols operating over plaintexts that do not offer any security.**

***Index Terms*—Histogram, Trusted Hardware, Cryptography**

I. Introduction

The collection and aggregation of time-series user-generated datasets in distributed sensor networks is becoming important to various industries (e.g., IoT, CPS, health industries) due to the massive increase in data gathering and analysis over the past decade. However, the data security and individual privacy of users can be compromised during data gathering, and numerous studies and high profile breaches have shown that significant precautions must be taken to protect user data from malicious parties [1]–[3]. As a result, there is an increased need for technology that allows for privacy-preserving data gathering and analysis. Specifically, many applications based on statistical analysis (e.g., crowd sensing, advertising and marketing, health care analysis) often involve the collection of user-generated data from sensor networks periodically for calculating aggregate functions such as the histogram distribution, sum, average, standard deviation, etc.. Due to data security and individual privacy concerns involved in the collection of consumer datasets, it is desirable to apply privacy-preserving techniques to allow the third-party aggregators to learn only the final outcomes.

In light of the COVID-19 crisis, the use of histograms in epidemiology to learn about the distribution of data points has taken on a new importance. In intervention epidemiology, histograms are often used to convey the distribution of onsets of the disease across discrete time intervals. This is called an *epidemic curve*, and is a powerful tool to statistically visualize the progress of an outbreak [4]. In the past, this technique has been used to help identify the disease's mode of transmission, and can also show the disease's magnitude, whether cases are clustered, if there are individual case outliers, the disease's trend over time, and its incubation period [5]. Additionally, the tool has assisted outbreak investigators in determining whether an outbreak is more likely to be from a point source (i.e. a food handler), a continuous common source (via continuing contamination), or a propagated source (such as primarily between people spreading the disease to each other) [6]. There are many applications for secure histogram calculation in the context of IoT and networked sensor systems, such as smart healthcare, crowd sensing, and combating pandemics (we discuss these in more detail in Section II).

Despite the versatility and usefulness of histograms, research into privacy-preserving histogram calculations has been largely ignored by the broader privacy-preserving computation research community. It should be said that in this context there are important differences between supporting location privacy, which requires hiding the location of the user participating in the protocol and input data privacy, which requires hiding the input value against the aggregator. It is possible to protect location privacy by uploading dummy data to hide the user's location, but protecting input value privacy from an aggregator is far more difficult, as one must build custom privacy-preserving cryptographic techniques to guarantee each user's data is secure. In our scenario, running privacy-preserving computations in a distributed setting is a challenging task, since protocols operating over massive amounts of data must be optimized for storage efficiency. Also, solutions must validate the authenticity and enforce access control over the endpoints involved in the calculations, ideally in real-time, and in spite of frequent user faults. There are existing techniques, such as secure multiparty computation (MPC) [7], fully homomorphic encryption (FHE) [8], and private stream aggregation (PSA) [9], that have been studied extensively in the past and might be used as black boxes to privately compute a histogram over users' data. However, prior work in these fields is not ideal for private histogram calculation on a massive scale. MPC often requires multiple rounds of input-dependent communication to be

performed, giving it high communication overhead [10]. FHE protocols are known to be far more computationally expensive than other private computation paradigms [11], making them impractical for private histogram calculations. Existing work in PSA, while having lower overhead [9], [12], is not ideally suited to histogram calculation, as it (i) is predominantly focused on pre-quantum cryptography, (ii) has limited scalability of users due to the open problem of handling online faults, and (iii) is often limited to simple aggregation (e.g., sum) only.

In contrast to existing works, our scheme CryptoGram is a highly scalable and accurate protocol, with low computation and communication overhead, that is capable of efficiently computing private histogram calculations over a set of users' data. In particular, CryptoGram is a non-interactive scheme with high efficiency and throughput, that supports tolerance against online faults without trusted third parties in the recovery, extra interactions among users, or approximation in the outcome, by leveraging a Trusted Execution Environment (TEE). Note that TEEs have their own constraints in communication/computation/storage efficiency that prevent them from being used naively as a black box to efficiently compute histograms, therefore naïvely using TEEs does not solve the problem successfully. Our protocol consists of specialized algorithms that minimize the amount of data sent into and computed over inside the TEE, to reduce overhead and support a practical runtime. CryptoGram contributes to the development of secure ecosystems of collection and statistical analysis involving user-generated datasets, by increasing the utility of the data gathered to data aggregators, while ensuring the privacy of users with a strong set of guarantees.

More specifically, we consider the problem of allowing a set of users to privately compute a histogram over their collected data such that an untrusted aggregator only learns the final result, and no individual honest user's data is revealed. In our approach, users communicate exclusively with the untrusted aggregator. We work in the malicious model and assume that all users may collude with each other and/or the untrusted aggregator. We want to guarantee that the untrusted aggregator cannot learn any individual input from any honest user (this implies if an untrusted aggregator corrupts or colludes with a user to learn their input, this does not impact the privacy of the honest users). All the untrusted aggregator can learn is the output of the function, i.e. the number of users that have a value within a certain histogram bucket. Further, we can prove the messages users send to the aggregator are indistinguishable from random. The main idea of our protocol is to leverage the Durstenfeld-Knuth uniformly random shuffling algorithm [13] to update the mapping between buckets and keys by using a deterministic cryptographically secure pseudorandom number generator similar to Rivest's [14]. We have implemented a proof-of-concept system and our experimental results indicate CryptoGram generally outperforms existing work by several orders of magnitude while achieving performance similar to protocols operating over plaintexts that do not offer any security.

Our contributions are as follows:

1) We present a novel technique that combines uniformly random shuffling with trusted hardware to minimize I/O overhead and support private histogram calculations with improved efficiency and strong privacy guarantees.

2) Our private histogram calculation scheme achieves a strong cryptographic security that is provably secure. The formal security proof is available in the appendix.

3) We experimentally evaluate our scheme (code available at: https://github.com/RyanKarl/PrivateHistogram) and demonstrate it is generally several orders of magnitude faster than existing techniques despite the guarantee of provable security.

II. Background and Applications

CryptoGram has several significant applications towards smart healthcare and combating epidemics, such as public health risk analysis, patient monitoring, and pharmaceutical supply chain management. For example, this protocol can be useful in determining the risk of a COVID-19 super spreader event [15]. Informally, this is an event where there are a large number of people that are in close physical proximity (i.e. less than 6 feet) for a prolonged amount of time. Super spreader events pose a threat to disease containment, as a single person can rapidly infect a large number of other people. By calculating a private histogram we can better understand the risk while protecting privacy. For instance, if we assume the untrusted aggregator is a public health authority (PHA) and users represent people at a particular location, we can have users calculate the number of people they come into contact with for a prolonged period of time with less than 6 feet of social distancing over discrete time intervals via Bluetooth. Then, the users can report the number of contacts to the PHA, which can learn the distribution of the contacts without being able to discover the number of contacts each individual user was exposed to. In this way, a PHA can learn which locations pose more of a risk of super spreader events, and perhaps devote more time and resources at these locations, instead of areas of comparatively lower risk.

Many hospitals utilize wireless body area networks, to monitor patients' health status in an efficient manner, by collecting and monitoring different physiology parameters including blood pressure, electrocardiography and temperature [16]. In addition, partial/spacial aggregate histogram data (which is the aggregation of multiple users' data at the same time point, i.e. the distribution of cholesterol scores among a group of people in the same region) is needed by pharmaceutical researchers to support production processes. Furthermore, temporal aggregate data (which is the aggregation of the same user's data at different time points, i.e. the highest cholesterol score for a given week) is needed by certified hospitals to monitor the health condition of outpatients and provide feedback in a short time span [17]. CryptoGram can be used to support the necessary privacy-preserving computation for these scenarios in a way that does not require the communication power, computation power, and storage resources of other approaches.

There are several other potential applications that can benefit from this work, such as private marketing statistics calculations (e.g., A/B testing), smart grid calculations, privacy-preserving

machine learning, etc.. Also, the underlying approach we discuss here to compute histograms can be easily extended to other functions, such as max, min, median, average, standard deviation, percentile aggregation, ANOVA, etc. [17]. We leave a formal discussion of this for future work.

III. Related Work

Fully homomorphic encryption (FHE) is an encryption technique that enables an analytical function to be computed over encrypted data while outputting the same results in an encrypted form as if the function was computed over plaintexts. This technique is powerful, but it is known to be considerably slower than other cryptographic methods [9]. To the best of our knowledge, the only work investigating the direct application of FHE to compute private histogram calculation [18] leverages the BGV [8] cryptosystem to construct secure protocols for a variety of basic statistical calculations, such as linear regression, counting, histogram, etc.. While their work is very interesting from a theoretical standpoint, it may have limited usability in practice, as they report overall computation time of approximately 15 minutes when computing a histogram over only 40 thousand data points.

Secure multi-party computation (MPC) is a technique in cryptography that allows for a group parties to jointly compute a function over their inputs while keeping those inputs private. Although it has been suggested that MPC can be leveraged [7] to efficiently compute histograms, we found no formal work specifically investigating this use case. Nevertheless, it is well known that MPC is generally subject to high communication delay in real life implementations, due to its underlying complexity and multiple required communication rounds [9].

Private stream aggregation (PSA) is a paradigm of distributed secure computing in which users independently encrypt their input data and an aggregator learns the final aggregation result of the private data but cannot infer individual data from the final output. PSA is superior to other types of secure computation paradigms especially in large-scale applications due to its extremely low overhead, the ease of key management, and because only a single round of input-dependent communication must be performed per aggregation [9], [10], [12]. To the best of our knowledge, there is only one existing PSA scheme designed to support histogram calculation known as PPM-HDA [17]. This system can calculate histograms by leveraging the BGN [19] cryptoscheme and a data filtering method based on numericalized prefixes, which negatively impacts run time.

In general, the respective techniques described above suffer from high computational overhead and communication complexity. Our own work can be thought of as a special purpose type of encryption that avoids the drawbacks of the previous approaches by leveraging trusted hardware (TEE).

Recently, there has been an interest in leveraging differential privacy (DP) to support private histogram publishing where the central publisher is a trusted entity [20]. In this problem, solutions are proposed to prevent data inference from the published histograms. By adding a carefully chosen distribution of noise [21], one can prevent the public from inferring the internal dataset from which the histograms are calculated, which increases privacy at the expense of accuracy. This problem and the technique are orthogonal to our own work, as we focus on the security of the computation against untrusted aggregator and not the privacy of the final result. That said, DP-based approaches can complement our approach to prevent the aggregator from inferring the distribution of users' inputs. We plan to investigate this in future work.

IV. Preliminaries

Below we describe some preliminary information regarding trusted hardware and TEEs useful for understanding our work since it is an important building block in our framework. Trusted hardware is a broad term used to describe any hardware that can be certified to perform according to a specific set of requirements, often in an adversarial scenario. One of the most prevalent in modern computing is Intel SGX, 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 running at superior privilege levels [22]. To allow 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 privacy 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 guarantees if applications are written in a two part model. Our framework can work with any form of TEE in the domain of trusted hardware, but we chose the Intel SGX for our concrete instantiation.

Applications must be split into a secure part and a non-secure part. The application can then launch an enclave, that is placed in protected memory, which allows user-level code to define private segments of memory, whose contents are protected and unable to be either 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. When it returns, enclave data stays in the protected memory [22]. 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 enabled by an autonomous hardware unit called the Memory Encryption Engine (MEE) that protects the confidentiality and integrity of the CPU-DRAM traffic over a specified memory range. The codes running within the enclave

is thus protected from being "spied on" by other code. Although the enclave is trusted, no process outside it needs to be trusted. Note that Intel SGX utilizes AES encryption, which is known to be quantum secure [23]. Before performing computation on a remote platform, a user can verify the authenticity of the trusted environment. Via the attestation mechanism, a third party can establish that correct software is running on an Intel SGX enabled device and within an enclave [22].

V. Definitions and Protocol Description

A. Adversary Model

Our scheme is designed to allow a third party (referred to as the aggregator) to perform the histogram calculation while providing strong data security guarantees. In our adversary model, the users $u_i \in \mathbb{U}$ send ciphertexts to an untrusted histogram aggregator that is equipped with a TEE. We work in the malicious model, and assume that all users may collude with each other and/or the untrusted aggregator, although the TEE is trusted. Under such adversary models, we want to guarantee that the untrusted aggregator cannot learn any individual input from any honest user (this implies if an untrusted aggregator corrupts or colludes with a user to learn their input, this does not impact the privacy of the honest users). All the untrusted aggregator can learn is the output of the histogram, i.e. the number of users that have a value within a certain range. Standard aggregator obliviousness [11], [12], which states the aggregator and colluders should learn only the final result 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 untrusted aggregator $\mathbb{A}$. Each user i where $0 < i \leq n-1$ possesses a piece of data $x_{i,t}$, corresponding to some timestamp t. The aggregator wishes to calculate the histogram y_t over the distribution of the private values users send. Our scheme provides privacy to individual users, preventing the aggregator from learning their individual data even when compromising or colluding with other users.

B. Protocol Definition and User/Aggregator Behavior

Our protocol is composed of three algorithms.

- $Setup(\lambda, \cdots)$: Takes a security parameter λ as input, along with any other required parameters (the number of users n, etc.). Returns a set of parameters $parms$, users' secret keys $s_i, i \in [0, n-1]$, and the randomness generated from the secret keys $r_{i,t}, i \in [0, n-1]$.
- $Enc(parms, x_{i,t}, s_i, r_{i,t}, t \cdots)$: Takes the scheme's parameters, a user's secret key s_i, the randomness generated with s_i for the timestamp t denoted $r_{i,t}$, and time-series input $x_{i,t}$, along with a timestamp t, and any other required parameters. Returns an encryption $c_{i,t}$ of the user's input under the randomness generated from their secret key.
- $Agg(parms, t, c_{0,t}, \cdots, c_{n-1,t})$: Takes the scheme's parameters, a timestamp t, and the n time-series ciphertexts from the users (with timestamp t). Returns the final histogram for timestamp t as $y_t = f(x_{0,t}, x_{1,t}, \cdots, x_{n-1,t})$, where f is the function that computes the histogram.

Users will run Enc on their data, and send their results $c_{i,t}$ to the aggregator. Then the aggregator calls Agg on the ciphertexts $c_{0,t}, \cdots c_{n-1,t}$ it has collected to learn the histogram result y_t. In our scheme, the algorithm $Setup$ is run in a trusted manner via additional trusted third party, secure hardware, etc..

C. Security Definition

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 [11], [12]). We define this below:

Definition 1 (Aggregator Obliviousness): Suppose we have a set of n users, who wish to compute a histogram over their data at a time point denoted timestamp t. A scheme π is aggregator oblivious if no polynomially bounded adversary has more than negligible advantage in the security parameter λ in winning the game below:

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 sends $(i, x_{i,t}, s_i, r_{i,t}, t)$ to the challenger and receives back $Enc(parms, s_i, t, x_{i,t}, r_{i,t})$.

Compromise: The adversary argues $i \in [0, n)$. 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 plaintexts $(x_{i,t}), (\tilde{x}_{i,t})$ and sends them to the challenger. The challenger then chooses a random bit b. If $b = 0$, the challenger computes $c_{i,t} = Enc(parms, s_i, t, x_{i,t}, r_{i,t})$ for every $i \in S$. If $b = 1$, the challenger computes $c_{i,t} = Enc(parms, s_i, t, \tilde{x}_{i,t}, r_{i,t})$ for every $i \in S$. The challenger returns $\{c_{i,t}\}_{i \in S}$ to the adversary.

The adversary wins the game if they can correctly guess the bit b chosen during the Challenge.

At a high level, the above definition says the aggregator obliviousness holds if an adversary cannot distinguish between $Enc(parms, s_i, t, x_{i,t}, r_{i,t})$ and $Enc(parms, s_i, t, \tilde{x}_{i,t}, r_{i,t})$ for any input values $x_{i,t}, \tilde{x}_{i,t}$. Our goal is to design a scheme that achieves the aggregator obliviousness.

Note, an adversary can learn information about a single user's data, by compromising the aggregator along with all other users, and then decrypting the compromised users' data and computing the difference of the aggregator's histogram and the histogram of the compromised users' data. Such situations are inherent in many scenarios with powerful adversaries, and are managed by building the security definition to require that no *additional* information is learned in such a case [11].

D. Our Concrete Protocol Design

When using TEEs, sending data into and out of the enclave can greatly increase overhead due to memory isolation. Also, performing standard encryption operations (i.e. AES) on the fly on user's devices, or similar decryption operations inside

Algorithm 1: Durstenfeld-Knuth Shuffle

Result: Shuffled list $\mathbb{L}$
Input: A list of values $\mathbb{L}$;
for *(var i = $\mathbb{L}$.length – 1; i > 0; i – –)* **do**
    index = random % (i + 1);
    current = $\mathbb{L}$[i];
    swap = $\mathbb{L}$[index];
    $\mathbb{L}$[i] = swap;
    $\mathbb{L}$[index] = current;
end

Algorithm 2: Cryptographically Secure PRNG

Result: Random number in range [0,q]
Input: A hash digest $\mathbb{H}$;
for *(var i = 0; i ≤ $\mathbb{H}$.length; i++)* **do**
    If (result[i] < 255 - (255 mod q)){return result[i]};
    If (i ≥ $\mathbb{H}$.length){$\mathbb{H}$ = $HASH(\mathbb{H})$};
end

the enclave, can be expensive, especially in the context of IoT. To overcome these challenges, we leverage the Durstenfeld-Knuth uniformly random shuffling algorithm [13] to update the mapping between buckets and keys by using a deterministic cryptographically secure pseudorandom number generator similar to that of Rivest [14]. This allows us to move most of the computational work to the offline phase, so we can efficiently process histogram calculations during the online phase, while minimizing the amount of data that must be passed into the TEE. Note, our technique is faster and more space efficient than simply encrypting inputs and sending them directly to the enclave (for Elliptic Curve Cryptography 64 bytes vs 1 byte for ours per user). Also, shuffling is better than storing all possible mappings locally, as the space complexity grows at a factorial rate (10 buckets means 10! possible mappings). Our novel approach allows us to efficiently pipeline the SGX enclave and user computation, significantly increasing performance.

We need the following functionalities to build our protocol:

Setup$(\lambda) \rightarrow \mathbb{K}_{i,t}, \mathbb{S}, \mathbb{B}$: In this subroutine, after inputting the security parameter λ, each user first performs attestation with the aggregator's Intel SGX, to verify it will faithfully execute the protocol (this is a one time process). The TEE generates a set of cryptographically secure random numbers $s_i \in \mathbb{S}$ for each user $u_i \in U$. Also, the mapping $\mathbb{K}_{i,t}$ associated with each user u_i between their keys $c_{i,j,t}$ where $1 \leq j \leq k$ and buckets $\mathbb{B} = \{b_1, b_2, \ldots, b_k\}$ is initialized (it is not strictly necessary that the number of keys and buckets be the same).

Enc$(\mathbb{K}_{i,t}, x_{i,t}, t) \rightarrow c_{i,j,t}$: In this subroutine, we inspect a mapping table $\mathbb{K}_{i,t}$, for the value $x_{i,t}$ at timestamp t and output the corresponding key $c_{i,j,t}$.

Shuffle$(\mathbb{K}_{i,t}, t, \mathbb{R}) \rightarrow \hat{\mathbb{K}}_{i,t+1}$: In this subroutine, we perform an unbiased random shuffle over a mapping table $\mathbb{K}_{i,t}$ for a time stamp t, given a set of random numbers $\mathbb{R}$ that has at least many elements as the length of the table $\mathbb{K}_{i,t}$. We output the newly shuffled mapping table $\hat{\mathbb{K}}_{i,t+1}$.

Random$(s_i) \rightarrow \mathbb{R}$: In this subroutine, given a seed s_i, we generate a cryptographically secure set of random numbers $\mathbb{R}$.

AggrDec$(\mathbb{K}_{i,t}, t, \mathbb{C}_t, \mathbb{B}_t) \rightarrow \hat{\mathbb{B}}_t$: In this subroutine, for each $c_{i,j,t} \in \mathbb{C}_t$ we inspect the related mapping table $\mathbb{K}_{i,t}$ associated with time stamp t, and using the provided key $c_{i,j,t}$, determine the corresponding input $x_{i,t}$. We then increment the associated bucket $b_j \in \mathbb{B}_t$ where $1 \leq j \leq k$ where the final set of buckets after all $c_{i,j,t} \in \mathbb{C}_t$ have been processed is denoted $\hat{\mathbb{B}}_t$.

We formally describe our protocol in Figure 1 for a single location for simplicity, but can easily extend it to support multiple locations, as long as we can synchronize the updates to the mapping. We provide a high level discussion of our protocol below, and omit timestamps to simplify notation.

Let $\mathbb{M} = \{x_1, x_2, \ldots, x_n\}$ denote all n users' private health data, and $\mathbb{B} = \{b_1, b_2, \ldots, b_k\}$ denote the buckets of the histogram, where k is the number of buckets (we assume the histogram's number of buckets is agreed upon beforehand). After the users attest the aggregator's TEE is running properly, the aggregator's TEE generates a cryptographically secure random number s_i for each user which will serve as the seed for generating future random numbers.

This s_i is sent over a secure channel to each user. On the user end and internally inside the TEE, the mapping $\mathbb{K}_i$ associated with each user u_i between their keys $c_{i,j}$ where $1 \leq j \leq k$ and buckets $\mathbb{B} = \{b_1, b_2, \ldots, b_k\}$ is initialized. On the user end and internally inside the TEE, we apply the Durstenfeld-Knuth shuffle over each mapping $\mathbb{K}_i$ to shuffle the mapping in an unbiased way. This algorithm is described in Algorithm 1. To ensure this shuffling is done in a cryptographically secure manner, we can use a method similar to Rivest's [14] secure deterministic PRNG to shuffle that mapping in a way that is indistinguishable from random. Our random number generator is described in Algorithm 2.

Essentially this technique computes the cryptographic hash (i.e. SHA) of the seed as $HASH(s_i)$ to generate a random number. To generate the next random number, we compute the hash again as $HASH(HASH(s_i))$. Note that we do not use the full SHA for each shuffle/swap, since in practice we have a small number of buckets. With less than 256 buckets, one byte of hash is sufficient for a swap. If we have less that 16 buckets, we can also shift the bits before continuing to iterate over the byte array. Because both the TEE and each user start with the same seed they will generate the same synchronized random stream, and as a result, will generate the same new random mapping after each run of the Durstenfeld-Knuth shuffle. This means that for each round (time stamp) of private histogram calculations that we want to precompute, we should run one instance of the shuffle and store the updated mapping. In practice, this step might occur overnight during a period of down time when user's devices are connected to a power source. After computing the private value x_i they wish to send (number of close contacts in a time interval), each user inspects their mapping table $\mathbb{K}_i$ to determine which key represents the bucket to which their private value belongs. Each user sends the appropriate key to the TEE for the given timestamp. After receiving each user's key c_i the TEE can use its internal mapping table to determine which bucket u_i's private value belongs to and increment that bucket's value accordingly. After completing this for each user in $\mathbb{U}$, the TEE

Protocol CryptoGram

Setup Phase): The TEE runs **Setup** and generates a cryptographically secure random number s_i for each user which will serve as the seed for generating future random numbers. This is sent over a secure channel to each user $u_i \in \mathbb{U}$. On the user end and internally inside the TEE, the mapping $\mathbb{K}_{i,t}$ associated with each user u_i between their keys $c_{i,j,t}$ where $1 \leq j \leq k$ and buckets $\mathbb{B}_t = \{b_1, b_2, \ldots, b_k\}$ is initialized.
Offline Phase): The TEE and each user run the **Random** subroutine using their seeds s_i to generate their set of random numbers. They then run **Shuffle** over their mapping $\mathbb{K}_{i,t}$ to precompute the mapping for each time stamp t as $\mathbb{K}_{i,t}$ for which they wish to precompute a mapping table.
Online Phase): After computing the private value $x_{i,t}$ they wish to send, each user $u_i \in \mathbb{U}$ runs **Enc**, and inspects their mapping table $\mathbb{K}_{i,t}$ for the current time stamp t to determine which key which represents the bucket to which their private value belongs. Each user sends the appropriate key $c_{i,j,t}$ to the TEE for the given round (timestamp).
Evaluation Phase): After receiving each user's key $c_{i,j,t}$ the TEE can run **AggDec** and use its internal mapping table to determine which bucket u_i's private value belongs to and increment that bucket's value accordingly. After completing this for each $u_i \in \mathbb{U}$, the TEE outputs the final histogram to the untrusted aggregator $\mathbb{A}$.

Fig. 1. Our Protocol

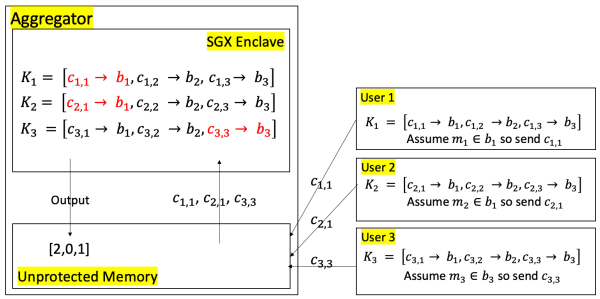


Fig. 2. Toy Example

outputs the final histogram to the untrusted aggregator $\mathbb{A}$.

E. Toy Example

Our toy example is in Figure 2. With three buckets and three users, the initial mapping is $\mathbb{K}_1 = [c_{1,1} \rightarrow b_1, c_{1,2} \rightarrow b_2, c_{1,3} \rightarrow b_3]$, $\mathbb{K}_2 = [c_{2,1} \rightarrow b_1, c_{2,2} \rightarrow b_2, c_{2,3} \rightarrow b_3]$, $\mathbb{K}_3 = [c_{3,1} \rightarrow b_1, c_{3,2} \rightarrow b_2, c_{3,3} \rightarrow b_3]$). Users 1, 2, and 3 decide to send inputs 1, 1, and 3 respectively, and after looking up the correct value in their local mapping tables, send $c_{1,1}, c_{2,1}$, and $c_{3,3}$ to the aggregator. The aggregator can send these into the TEE, to decode these and increment the appropriate histogram buckets accordingly. After completing the histogram calculation, the users and aggregator will run the shuffle algorithm over their mapping tables, and might generate the following new tables for use during the next calculation: $\mathbb{K}_1 = [c_{1,1} \rightarrow b_2, c_{1,2} \rightarrow b_3, c_{1,3} \rightarrow b_1]$, $\mathbb{K}_2 = [c_{2,1} \rightarrow b_3, c_{2,2} \rightarrow b_2, c_{2,3} \rightarrow b_1]$, $\mathbb{K}_3 = [c_{3,1} \rightarrow b_1, c_{3,2} \rightarrow b_3, c_{3,3} \rightarrow b_2]$).

F. Proof of Security

We provide a formal proof of security in the appendix, and provide a proof sketch here for completeness. Essentially, by the security of the cryptographic hash function and the security of generating a truly random seed, the random streams generated by each user are indistinguishable from random. Because these numbers are used to perform the shuffling algorithm, the uniformly shuffled mapping is also indistinguishable from random, and the adversary cannot deduce which key corresponds to which input for each user for each round. The semantic security of the ciphertexts $c_{i,j,t}$ generated in our scheme follows directly from Theorem 1. Post-quantum security follows from Theorem 1, and from the underlying post-quantum security of AES encryption [22]–[24], which is used by the Intel SGX Memory Encryption Engine to encrypt data in the enclave (note we assume quantum secure signatures are used during attestation, a forthcoming but not yet implemented future feature of Intel SGX). Thus, the protocol is secure. This protocol is naturally fault tolerant, as if a user fails to respond this does not impact the calculation of the histogram over the remaining users' data. In practice, we can assign a dummy value for users to send so that we can mask their location, if we are calculating histograms for multiple locations simultaneously.

VI. Experimental Evaluation

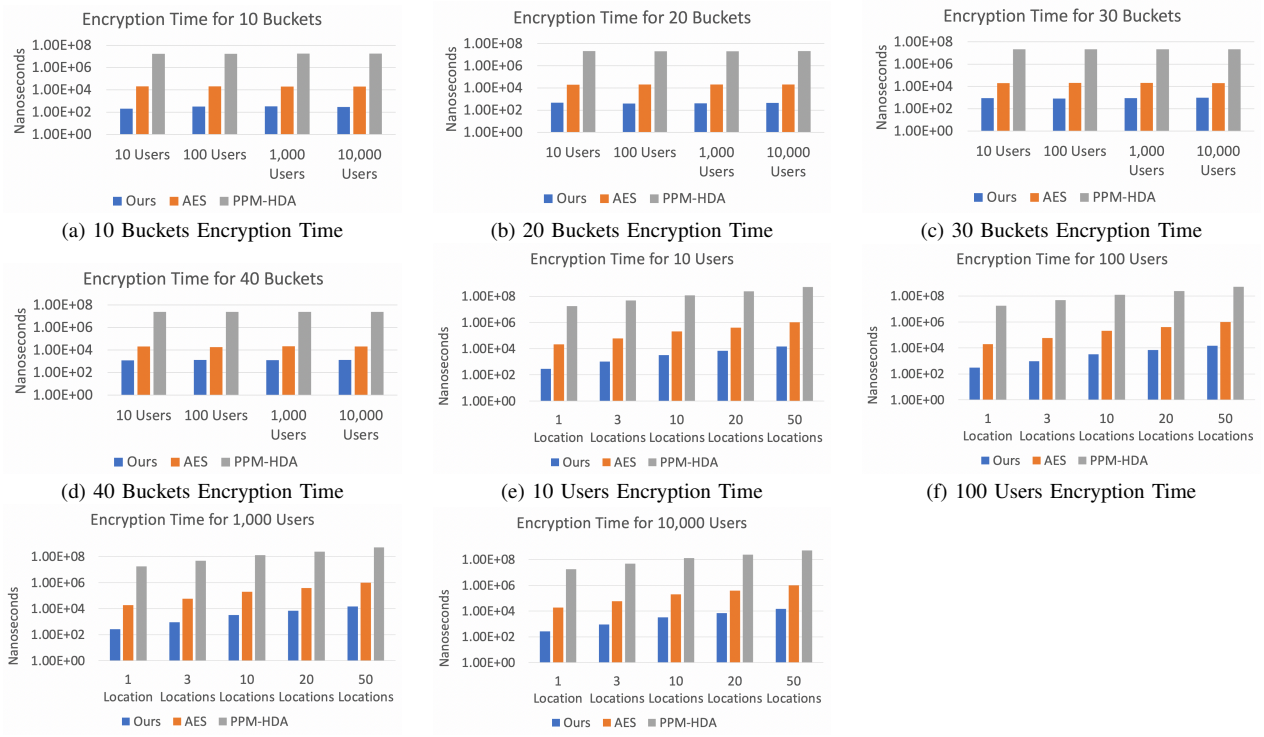
A. Experiment Setting

To better understand the improvements gained in performance, we experimentally evaluated our work when compared to several baseline techniques. This is the best we can do because a secure computation implementation for histogram calculation does not exist in the literature. The vast majority of existing work focus on the differential privacy in the histograms disclosed by a trusted central server, which is orthogonal to the problem in this paper where the aggregator is untrusted.

We ran experiments using data from the Tesserae project, [25], which tracked the close (less than 6 feet) contact of 757 participants in office settings (concentrated around four major organizations) across the USA by utilizing Bluetooth beacons and sensors embedded in personal phones and wearables over the course of a year [1]. We compared our protocol to simple plain histogram aggregation with no security guarantees and to simply encrypting all user plaintexts using OpenSSL's implementation of AES with a 128-bit security level (which is common in traditional PSA schemes [12]) and sending the corresponding ciphertexts in to the Enclave to be aggregated into a histogram. We also ran other simulations to compare our scheme to the state of the art work in private histogram calculation [17].

We implemented our experiments using C++11, and version 2.10 of the Intel SGX Software Development Kit. Our experiments were run on a computer running Ubuntu 18.04 with an Intel(R) Xeon(R) W-1290P 3.70GHz CPU with 128 GB of memory. We report the average runtimes over 50 trials of the experiment in question. In general, our technique performs the best in all scenarios, often by several orders of magnitude.

[1]We oversampled randomly from this dataset for experiments using large numbers of users and/or locations.



(a) 10 Buckets Encryption Time

(b) 20 Buckets Encryption Time

(c) 30 Buckets Encryption Time

(d) 40 Buckets Encryption Time

(e) 10 Users Encryption Time

(f) 100 Users Encryption Time

(g) 1,000 Users Encryption Time

(h) 10,000 Users Encryption Time

Fig. 3. Encryption Experimental Results

B. Comparison of Encryption

We first compare the encryption time of the respective protocols and vary the number of buckets in the histogram. We report results in Figure 3. We note that our technique is always the fastest by at least multiple orders of magnitude. This makes sense as our one time pad based method is considerably less computationally expensive than AES encryption and the ECC based techniques of PPM-HDA. Also, PPM-HDA makes use of an interesting but expensive binary prefix encoding procedure, which negatively impacts the overall run time. This trend continues as we increase the number of locations participating in the histogram calculation. Thus, we can conclude that in secure histogram aggregation scenarios where encryption times greatly impact the overall performance, our method offers the best efficiency.

C. Comparison of Aggregation

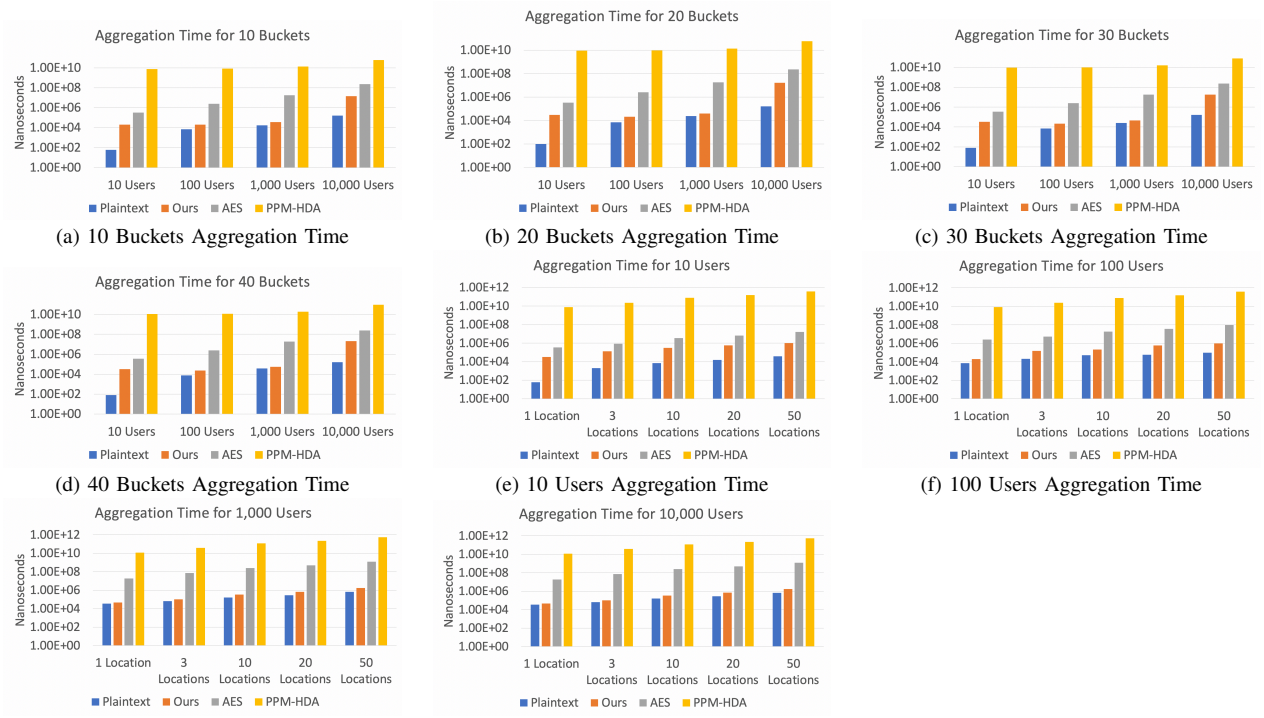
We also compare the aggregation time of the respective protocols, along with a plaintext implementation of aggregation that offers no privacy guarantees, and vary the number of buckets in the histogram. We report results in Figure 4. Interestingly, in the best case, our method is within one order of magnitude of the plaintext run time. This is not surprising, as it has been documented [26] that code running inside Intel SGX enclaves can achieve very high throughput, on par with code running outside enclaves as long as the number of expensive context switches between trusted and untrusted space is minimized. We note that although the plaintext aggregation technique is always the fastest, our method is consistently the most efficient of the three privacy-preserving schemes, often by several orders of magnitude. Again, this makes sense, as our one time pad based method is considerably less computationally expensive than AES encryption and the ECC based techniques of PPM-HDA, which require the aggregator to solve the discrete logarithm problem to successfully decrypt. Again, PPM-HDA's binary prefix encoding procedure negatively impacts performance. This trend continues as we increase the number of locations participating in the histogram calculation. We conclude that in secure histogram aggregation scenarios where aggregation times greatly impact the overall performance (i.e. large number of users and locations), our method offers the best efficiency.

VII. Conclusion

We considered the problem of allowing an untrusted party to privately compute a histogram over multiple users' data such that the aggregator only learns the final result, and no individual honest user's data is revealed in the malicious adversary model. In addition to being provably secure, experimental evaluations of our technique indicate that it generally outperforms existing methods by several orders of magnitude, and can achieve performance that is within one order of magnitude of protocols operating over plaintexts that offer no security guarantees.

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



(a) 10 Buckets Aggregation Time

(b) 20 Buckets Aggregation Time

(c) 30 Buckets Aggregation Time

(d) 40 Buckets Aggregation Time

(e) 10 Users Aggregation Time

(f) 100 Users Aggregation Time

(g) 1,000 Users Aggregation Time

(h) 10,000 Users Aggregation Time

Fig. 4. Aggregation Experimental Results

expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] V. Mayer-Schönberger and K. Cukier, *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[2] J. Bonneau, J. Anderson, and G. Danezis, "Prying data out of a social network," in *ASONAM*. IEEE, 2009, pp. 249–254.

[3] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *arXiv preprint cs/0610105*, 2006.

[4] M. Torok, "Epidemic curves ahead," *Focus on Field Epidemiology*, vol. 1, no. 5, 2019.

[5] D. M. Dwyer, C. Groves, and D. Blythe, "Outbreak epidemiology," *Infectious Disease Epidemiology: Theory and Practice. Jones & Bartlett Publishers*, pp. 105–127, 2014.

[6] D. J. Weber, L. B. Menajovsky, and R. Wenzel, "Investigation of outbreaks," *Epidemiologic methods for the study of infectious diseases. NY, NY: Oxford University Press, Inc*, pp. 291–310, 2001.

[7] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases—real-world applications of secure multi-party computation," *The Computer Journal*, vol. 61, no. 12, pp. 1749–1771, 2018.

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

[9] T. Jung, J. Han, and X.-Y. Li, "PDA: Semantically secure time-series data analytics with dynamic subgroups," *TDSC*, vol. PP, no. 99, pp. 1–1, 2016.

[10] F. Valovich and F. Aldà, "Computational differential privacy from lattice-based cryptography," in *NuTMiC*. Springer, 2017, pp. 121–141.

[11] D. Becker, J. Guajardo, and K.-H. Zimmermann, "Revisiting private stream aggregation: Lattice-based psa." in *NDSS*, 2018.

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

[13] R. Durstenfeld, "Algorithm 235: Random permutation," *Commun. ACM*, vol. 7, no. 7, p. 420, Jul. 1964.

[14] R. Rivest, "Publicly verifiable nominations committee (nomcom) random selection." The Internet Society, 2011.

[15] T. R. Frieden and C. T. Lee, "Identifying and interrupting superspreading events—implications for control of severe acute respiratory syndrome coronavirus," in *CDC Publishing*, 2020.

[16] U. Mitra, B. A. Emken, S. Lee, M. Li, V. Rozgic, G. Thatte, H. Vathsangam, D.-S. Zois, M. Annavaram, S. Narayanan *et al.*, "Knowme: a case study in wireless body area sensor network design," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 116–125, 2012.

[17] S. Han, S. Zhao, Q. Li, C.-H. Ju, and W. Zhou, "PPM-HDA: privacy-preserving and multifunctional health data aggregation with fault tolerance," *TIFS*, vol. 11, no. 9, pp. 1940–1955, 2015.

[18] W. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data." *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1163, 2016.

[19] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *TCC*. Springer, 2005, pp. 325–341.

[20] X. Liu and S. Li, "Histogram publishing method based on differential privacy," *DEStech TCSE*, no. csse, 2018.

[21] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[22] V. Costan and S. Devadas, "Intel SGX explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[23] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum security analysis of AES," *IACR ToSC*, 2019.

[24] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[25] S. M. Mattingly, J. M. Gregg, P. Audia, A. E. Bayraktaroglu, A. T. Campbell, N. V. Chawla, V. Das Swain, M. De Choudhury, S. K. D'Mello, A. K. Dey *et al.*, "The Tesserae project: Large-scale, longitudinal, in situ, multimodal sensing of information workers," in *CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–8.

[26] D. Harnik, "Impressions of intel SGX performance," 2017. [Online]. Available: https://medium.com/@danny_harnik/impressions-of-intel-sgx-performance-22442093595a

APPENDIX A
PROOF

Recall that semantic security is a concept within the field of cryptography to capture the following functionality: Suppose an adversary is allowed to choose between two plaintext messages, p_0 and p_1, and they receive an encryption of either one of them. An encryption scheme is considered to be semantically secure, if an adversary cannot guess with better probability than $\frac{1}{2}$ whether the given ciphertext is an encryption of p_0 or p_1. The definition describing this requirement is equivalent to requiring the indistinguishability of encryptions.

For our proof of semantic security, we will first make note of the additive stream cipher (one time pad) for completeness. Privacy of user data in our approach is ensured via an approach similar to symmetric cryptography (an additive stream cipher). Note that for an additive stream cipher, if we let F_z be a field, we can have the TEE generate a stream of pseudorandom elements $r_1, r_2, ...r_j, ... \in F_z$ in an offline phase. Later, the remote client i can send an input $c_i = x_i \oplus r_j$ during an online phase, and the TEE can compute $x_i = c_i \oplus r_j$, where $\oplus$ is a mapping (addition) operation to recover the x_i inside of the TEE, while keeping this hidden from the untrusted aggregator.

We recall Shannon's definition of perfect secrecy [24]:

Definition 2 (Perfect Secrecy): An encryption scheme satisfies perfect secrecy if for all messages m_1, m_2 in message space $\mathcal{M}$ and all ciphertexts $c \in \mathcal{C}$, we have

$$\Pr_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_1) = c] = \Pr_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_2) = c]$$

where both probabilities are taken over the choice of K in $\mathcal{K}$ and over the coin tosses of the probabilistic algorithm Enc.

More simply, for a given ciphertext, every message in the message space is equally as likely to be the actual plaintext message, i.e. the plaintext is independent of the ciphertext. This means that this definition of secrecy implies that an attacker learns nothing about the underlying plaintext. It is well known that the Additive Stream Cipher satisfies this definition, as long as the length of the message is the same as the length of the key. This is because conducting a brute force search gives the attacker no extra information and every ciphertext could correspond to any message since the total keyspace is as large as the message space. Below is a proof of this [24] for completeness.

Theorem 1: The Additive Stream Cipher satisfies Perfect Secrecy as defined in Def. 2.

Proof 1:

Proof: Take any $m \in \mathcal{M}$ and $c \in \mathcal{C}$, and let $k^* = m \oplus c$. Note that: $\mathrm{Prob}_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m) = c] = \mathrm{Prob}_{K \leftarrow \mathcal{K}}[(K \oplus m) = c] = \mathrm{Prob}_{K \leftarrow \mathcal{K}}[K = c \oplus m] = \mathrm{Prob}_{K \leftarrow \mathcal{K}}[K = k^*] = \frac{1}{2^\ell}$ Noting that the above equation holds for each $m \in \mathcal{M}$, this means for every $m_1, m_2 \in \mathcal{M}$ we have $\mathrm{Prob}_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_1) = c] = \frac{1}{2^\ell}$, and $\mathrm{Prob}_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_2) = c] = \frac{1}{2^\ell}$, thus $\mathrm{Prob}_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_1) = c] = \mathrm{Prob}_{K \leftarrow \mathcal{K}}[\mathrm{Enc}(K, m_2) = c]$.

We now prove the security of our scheme in the following theorem:

Theorem 2: Assuming the TEE is secure, the random number generator is cryptogrpahically secure, and trusted setup infrastructure is secure, our protocol described in Figure 1 securely computes the final histogram result via the histogram calculation function f in the presence of malicious adversaries.

Proof 2:

Without loss of generality, assume the adversary controls the first $m < n$ users, where n is the total number of users. We show that a probabilistic polynomial time simulator can generate an entire simulated view, given $\hat{\mathbb{B}}$ and $x_1, \cdots x_m$ for an adversary indistinguishable from the view an adversary sees in a real execution of the protocol, where x_i denotes the input value from user i, f denotes the histogram calculation function, and y denotes the final histogram. Note the simulator is able to find $x'_{m+1}, \cdots, x'_n$ such that $y = f(x_1, \cdots, x_m, x'_{m+1}, \cdots, x'_n)$. Besides this, it follows the protocol as described in Figure 1, pretending to be the honest players. Because the trusted setup was successful and identical in both the real and ideal worlds, each user (adversary controlled or otherwise) has secret key s_i or s'_i that they can use to generate a cryptographcally secure stream of numbers as $r_i \in \mathbb{R}$ or $r'_i \in \mathbb{R}'$ in an offline phase. Note in both worlds, when the users run **Shuffle**, because the algorithm is perfectly unbiased [13] and executed using an infinite stream (for practical purposes) of cryptographically secure random numbers via [14], the ordering of the bucket key mapping $\hat{\mathbb{K}}_{i,t+1}$ is indistinguishable from a truly random sequence within the bucket range. Furthermore, we can continue running the **Shuffle** algorithm to generate an infinite number of mappings that are indistinguishable from a truly random sequence within the bucket range.

Now when users in each world run **Enc** over their messages x_i or x'_i to compute the associated c_i or c'_i, this is equivalent to computing the additive stream cipher calculation $c_i = x_i \oplus r_j$, since the unique mapping $\hat{\mathbb{K}}_{i,t+1}$ is indistingusiable from the random r_j. By the perfect secrecy of the additive stream cipher, the messages sent in both worlds by all users are semantically secure, and indistinguisable from each other, even though the message length and key are short (less than 16 bits) for each message sent. Once the aggregator receives all the given ciphertexts for the current timestep, no information about the corresponding plaintexts for the honest users can be inferred at this point in the protocol, since this encryption is semantically secure, and each user (honest or otherwise) used their own unique cryptogrpahically secure random seed to generate their bucket key mapping $\hat{\mathbb{K}}_{i,t+1}$. From here, the adversary sends the ciphertexts into the TEE, where they isolated from the adversary (and encrypted via AES with Intel SGX), and the trusted TEE can safely run **AggrDec** over the ciphertexts to recover the set of plaintexts for either world. Following this, the final histogram $\hat{\mathbb{B}}$ or $\hat{\mathbb{B}}'$ is outputted to the aggregator, while the plaintexts remain in the TEE and are kept secret. Now since the plaintexts used are identical in both worlds, $\hat{\mathbb{B}}$ and $\hat{\mathbb{B}}'$ are identical too. Note it is known these techniques are quantum secure [23], [24].

We now must show the protocol is aggregator oblivious. Note we use techniques that are very similar to existing proofs

[11], [12], and we adapt these for our own protocol. Our goal is to show 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 the ciphertexts in our scheme, before they are sent into the TEE to compute the histogram. We modify the game of aggregator obliviousness as follows:

1) We change any Encrypt query as a Compromise query from the adversary (note this strengthens the adversary), and we change the Challenge phase to a real-or-random version.

2) In the original game of aggregator obliviousness, the adversary must specify two plaintexts $(x_i), (\tilde{x}_i)$ and then to distinguish between encryptions of either of them. Here, we let the adversary choose one (x_i) and have them distinguish between valid encryptions of (x_i) or a random value v_i.

Any adversary with more than negligible advantage in winning this modified game will also be able to win the original aggregator obliviousness security game with more than negligible advantage. Thus, it is enough to 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 our ciphertexts from random. For simplicity, we consider the protocol's operation at a single timestamp t. We also omit the timestamp identifier of plaintexts, ciphertexts, and other variables. A basic property of aggregator obliviousness is that it tolerates the case when the adversary compromises all but one participant, and allows that they can learn the secret key of that participant and can therefore distinguish between valid encryptions and random values. To account for this, the definition requires that they do not learn any additional information about that particular participant.

First, we define a game for $\mathcal{B}$ that describes their ability to break the ciphertexts' semantic security. Suppose $\mathcal{B}$ receives the parameters $parms$. Then with a challenger $\mathcal{C}$ testing the ability of $\mathcal{B}$ to break the encryption, $\mathcal{B}$ will play the modified game described above (the TEE is somewhat similar to challenger $\mathcal{C}$). $\mathcal{B}$ can make Sample queries by arguing x_i to $\mathcal{C}$ and will receive back the ciphertext c_i, which is an encryption using the random stream associated with i. Later, during the Distinguish phase, $\mathcal{B}$ argues x_i to $\mathcal{C}$. Based on a random bit b chosen by $\mathcal{C}$, $\mathcal{C}$ will choose c_i either as an encryption of x_i (if $b = 0$) or a random element v_i of $\mathbf{F}_k$ (if $b = 1$). Then $\mathcal{B}$ must guess the value of b, winning if correct. We note that clearly by Theorem 1 it is impossible for $\mathcal{B}$ to be able to win the game with more than negligible advantage, as the ciphertexts c_i are indistinguishable from random.

$\mathcal{B}$ can simulate the modified game of aggregator obliviousness to $\mathcal{A}$ as follows. In the Setup phase, $\mathcal{B}$ will first choose distinct $l, q \in [0, n-1)$. Then, $\mathcal{B}$ sets $s_q = b_q$, $s_l = b_l$, where b_q and b_l are random values, and selects secret keys s_i for all $i \neq l, q$. Note that $\mathcal{B}$ does not actually know either of s_q, s_l, stored by $\mathcal{C}$ which are the aggregation scheme's secret keys for users l, q. In the Compromise phase, $\mathcal{A}$ will send a query i to $\mathcal{B}$. If $i \notin \{l, q\}$, then $\mathcal{B}$ returns s_i to $\mathcal{A}$, otherwise we abort. In the Challenge phase, $\mathcal{A}$ will choose a set of uncompromised users $\mathcal{U} \subseteq [0, n-1) \setminus \{l, q\}$. They will then send plaintexts $\{(x_i)\}$ with $i \in \mathcal{U}$ to $\mathcal{B}$. Because we chose earlier to abort if a query was for either of l, q, we know that $l, q \in \mathcal{U}$. Then, $\mathcal{B}$ computes $\{c_i = Enc(parms, s_i, x_i, r_i)\}$ for $i \in \mathcal{U} \setminus \{l, q\}$. We then enter the Distinguish phase and $\mathcal{B}$ sends $(parms, s_{q,l}, x_{q,l}, r_{q,l})$ to $\mathcal{C}$, who returns $c_{q,l} = Enc(parms, s_{q,l}, x_{q,l}, r_{q,l})$. Then $\mathcal{B}$ computes an encryption of the plaintext, with $c_{q,l} = Enc(parms, x_{q,l}, r_{q,l}, s_{q,l})$. Now $\mathcal{B}$ has c_i for $i \in U$, including c_l and c_q. $\mathcal{B}$ then returns these values to $\mathcal{A}$. We now move to the Guess phase. If $\mathcal{A}$ has more than negligible advantage in winning the security game, they can distinguish the ciphertexts from random. Specifically, if $c_{q,l}$ is a valid encryption of $x_{q,l}$ 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 the ciphertexts from random and break the semantic security of the scheme and violate Theorem 1. Thus it is impossible for them to be able to win the game with more than negligible advantage, and this completes the proof.

Therefore, all the adversary can learn is what can be inferred based on the output of the the final histogram and inputs they control. As we know this is aggregator oblivious from the proof above, we know the adversary cannot distinguish between real and simulated executions and our protocol securely computes the histogram over the participants data.