Privacy-Preserving Aggregation of Time-Series Data

Elaine Shi
PARC/UC Berkeley
elaines@eecs.berkeley.edu

T-H. Hubert Chan
The University of Hong Kong
hubert@cs.hku.hk

Eleanor Rieffel
FxPal
rieffel@fxpal.com

Richard Chow
PARC
rchow@parc.com

Dawn Song
UC Berkeley
dawnsong@cs.berkeley.edu

Abstract

We consider how an untrusted data aggregator can learn desired statistics over multiple participants' data, without compromising each individual's privacy. We propose a construction that allows a group of participants to periodically upload encrypted values to a data aggregator, such that the aggregator is able to compute the sum of all participants' values in every time period, but is unable to learn anything else. We achieve strong privacy guarantees using two main techniques. First, we show how to utilize applied cryptographic techniques to allow the aggregator to decrypt the sum from multiple ciphertexts encrypted under different user keys. Second, we describe a distributed data randomization procedure that guarantees the differential privacy of the outcome statistic, even when a subset of participants might be compromised.

1 Introduction

In many practical applications, a data aggregator wishes to mine data coming from multiple organizations or individuals, to study patterns or statistics over a population. An important challenge in these applications is how to protect the privacy of the participants, especially when the data aggregator is untrusted.

This paper describes novel Private Stream Aggregation (PSA) algorithms which allow users to upload a stream of encrypted data to an untrusted aggregator, and allow the aggregator to decrypt (approximate) aggregate statistics for each time interval with an appropriate capability. We guarantee a strong notion of privacy. First, our aggregation scheme is *aggregator oblivious*, meaning that the aggregator is unable to learn any unintended information other than what it can deduce from its auxiliary knowledge and the desired statistics. Second, we guarantee *distributed differential privacy* for each individual participant, in the sense that the statistic revealed to the aggregator will not be swayed too much by whether or not a specific individual participates. Therefore, users may safely contribute their encrypted data, as presence in the system will not lead to increased risk of privacy breach. Our privacy guarantees hold even when the aggregator has arbitrary auxiliary information about an individual's inputs (but has not compromised her secret key). Such auxiliary information may be obtained from publicly available datasets, personal knowledge about an individual participant, or through collusion with a small subset of corrupted participants.

The proposed privacy mechanisms represent a promising approach to ensuring user privacy in numerous application, including cloud services, medical privacy, sensor network aggregation, and smart metering.

1.1 Contributions

Formulation of a privacy model. One important contribution we make is the formulation of a notion of privacy. A good way to understand our contributions is to compare our notion of privacy with differential privacy [5]. The differential privacy literature assumes the presence of a trusted data aggregator who wishes to publish statistics about a population. The trusted data aggregator is entitled to see all participants' data in the clear. Our privacy model is stronger in the sense that we do not trust the data aggregator. We ensure that the data aggregator is able to learn only the intended statistics and no additional information. Furthermore, the statistics revealed to the data aggregator satisfies differential privacy guarantees. Our scheme protects each individual participant's privacy even when the aggregator has arbitrary auxiliary information about an individual's data (but has not compromised her secret key), or colludes with a subset of corrupted participants.

Computing the sum statistic for time-series data. We propose novel constructions that allow an untrusted data aggregator to compute the sum statistic for time-series data. Imagine a data aggregator who wishes to keep track of the total sales revenue of n companies every week. Our scheme allows each individual company to upload a noisy encryption of their revenue every week to the data aggregator. With an appropriate capability, the data aggregator is able to decrypt the noisy sum of all companys' revenues, but is unable to infer additional information about an individual company.

Our effort is a first step towards this new notion of privacy. In Section 8, we propose several open problems that remain to be addressed for this new direction. We hope that our effort will inspire future research in this area to address these interesting challenges.

1.2 Applications

Sensor network aggregation. Sensor networks are being widely-deployed to monitor the safety of buildings, measure traffic flows, or track environmental pollutants. In a typical setting, deployed sensor nodes periodically send their readings to a base station, which mines the data for some pattern or statistic. In many scenarios, the readings from each individual sensor may be privacy sensitive, especially if the sensors are deployed across multiple organizations. Our construction may provide a promising approach to address privacy issues arising in sensor network aggregation.

Smart metering. Another example is the advent of the electrical "smart grid" and "smart metering" [4]. Smart meters read electrical usage at a much finer granularity than traditional meters; smart meters might read usage every 15 minutes as opposed to once a month. See [15] for a sampling of what might be gleaned from your fine-grained electrical usage. For instance, one can deduce the number of individuals in the household and their sleep/work habits, as well as their use of common household appliances. These privacy concerns could be much reduced if household usage information were only released in the aggregate. Membership in the aggregation groups would be flexible and open, and these aggregate statistics would still be enough for the smart grid operators to do much of their monitoring and price optimization.

Public health and clinical research. Medical research benefits greatly from medical data, but privacy concerns restrict the extent to which this data is collected and disseminated. Molina *et al.* [14] use multiparty computation by caregivers to answer researcher aggregation queries, especially for medical telemetry data. PSA algorithms would enable researchers to obtain those statistics, and only those statistics, from data uploaded continually by the caregivers or the telemetry devices, without need for further interaction.

Population monitoring and sensing. There are many examples of population polling, sensing, and monitoring spurring privacy concerns. As one example, Rieffel *et al.* [17] use data from cameras, wifi, and computer activity to estimate a user's availability, and help co-workers identify the best means for communication with that user. A lot of information about a user's work habits can be deduced from her communication availability. For this reason, users were reluctant to have any long-term data stored for fear that it would be misused by their managers. As they used the system, however, users became interested in sharing information with selected individuals, and were open to allowing managers to learn statistical data across the group. Rieffel *et al.*'s work addressed the oblivious aggregation problem, but was susceptible to collusion.

Cloud services. As cloud computing gains popularity, individuals and organizations will store an increasing amount of data on third-party cloud services. Cloud service providers wish to compute useful statistics over this data, to realize various social and economic goals. Unfortunately, companies cite concerns about the security and privacy of their data as a top reason for not making more use of cloud services. Our constructions represent a promising approach in cloud applications, especially when the cloud services wishes to track some aggregate statistics from multiple users over time.

2 Related Work

To understand our contributions, it helps to know the relationship of this paper with well-known privacy techniques such as differential privacy, homomorphic encryption and secure multi-party computation.

Differential privacy. The differential privacy notion was first formulated by Dwork *et al.* [5, 7]. Differential privacy ensures that a user is not at increased risk of privacy when she participants in a certain statistical database. Previous work on differential privacy considers a trusted data aggregator who has access to all users' data. The trusted aggregator typically adds appropriate noise to some statistics before releasing them. In comparison, our work provides stronger privacy guarantees as we ensure the privacy of individual participants even against the data aggregator itself. This is

valuable in many real-world settings (e.g., cloud applications) where users may not entrust the data aggregator or storage server with their data. Dwork *et al.* [6] have also considered distributed randomness among participants in order to achieve differential privacy. However, their scheme involves interactions among all users.

Homomorphic encryption. Most previous work on homomorphic encryption considers homomorphic operations on ciphertexts encrypted under the same key [2, 8]. These schemes do not directly apply in our case, since if participants encrypted their data under the aggregator's public key, the aggregator would not only be able to decrypt the aggregate statistics, but also each individual's values. By contrast, our cryptographic construction allows additive homomorphic operations over ciphertexts encrypted under different users' secret keys.

Castelluccia *et al.* [3] designed a symmetric-key homomorphic encryption scheme that allows an aggregator to efficiently decrypt the mean and variance of encrypted sensor measurements. However, they also assume a trusted aggregator who is allowed to decrypt each individual sensor's values. Yang *et al.* [19] designed an encryption scheme that allows an aggregator to compute the sum over encrypted data from multiple participants. As pointed out by Magkos *et al.* [11], their construction only supports a single time step, and an expensive re-keying operation is required to support multiple time steps.

Secure multi-party computation. Secure multi-party computation (SMC) [10] is a well-known cryptographic technique allowing n parties with inputs $\mathbf{x} = (x_1, x_2, \ldots x_n)$ respectively to privately compute functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})$. At the end of the protocol, party i learns the value of $f_i(\mathbf{x})$ but nothing more. In a sense, SMC is orthogonal and complementary to differential privacy techniques. SMC does not address potential privacy leaks through harmful inferences from the outcomes of the computation, but one can potentially build differential privacy techniques into a multi-party protocol to address such concerns.

Most SMC constructions are interactive. Therefore, directly employing SMC in our setting would require participants to interact with each other whenever an aggregate statistic needs to be computed. Such a multi-way interaction model may not be desirable in practical settings, especially in a client-server computation model as often seen in cloud computing applications.

Most closely related work. To the best of our knowledge, Rastogi *et al.* [16] and Rieffel *et al.* [17] were the first ones to consider the problem of privately aggregating sums over multiple time periods.

Rastogi and Nath [16] also consider periodic aggregation of the sum statistic in the presence of an untrusted aggregator. Their work differs from our work in several aspects. (1) One of our contributions is to present a formal security definition. Rastogi *et al.* prove that the aggregator cannot compute linear combinations of the users' values other than the sum – and this implicit security defintion they adopted is incomplete in some sense. Note that although we define security specifically for the sum statistic in this paper, our security definitions can potentially be generalized for the aggregation of general statistics. (2) The construction by Rastogi *et al.* requires that the aggregator engage in an extra round of interaction with the participants to decrypt the sum for every time interval. In our scheme, the aggregator need not interact with the participants to decrypt the sum. (3) Rastogi *et al.* also consider how the users can jointly contribute randomness to achieve differential privacy. Interestingly, they propose a noise generation algorithm different from ours. In contrast to Rastogi *et al.*, our scheme and privacy analysis explicitly address the issue of rounding when the underlying algebraic structure of the encryption scheme supports only finite discrete values.

Rieffel *et al.* consider an application scenario where a manager would like to periodically decrypt a summary statistic across a set of users, while not being able to decrypt individual values [17]. Their construction does not provide distributed differential privacy guarantees, and is not fully resistant against collusions, i.e., users and the manager may collude to decrypt a victim user's value. They raise the question whether it is possible to design a scheme fully resistant against collusion. This paper gives explicit formalizations of the privacy model implicit in their work, and provides a positive answer to the question they raised.

3 Problem Definition and Overview

Suppose we have one data aggregator and n participants. For notational convenience, we number the participants $1, \ldots, n$, and we number the data aggregator 0. Let $[n] := \{1, 2, \ldots, n\}$. In every time period $t \in \mathbb{N}$, each participant $i \in [n]$ has a value $x_{i,t} \in \mathcal{D}$ from a certain domain $\mathcal{D}$. When the context is clear, we omit the subscript t and write x_i instead. Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{D}^n$ denote the vector of values from all participants in some time period. The aggregator would like to compute some aggregate statistics represented by the function $f : \mathcal{D}^n \to \mathcal{O}$. The function $f(\mathbf{x})$ produces some value from the some range $\mathcal{O}$, representing the desired statistics.

To achieve strong privacy guarantees when the aggregator may have arbitrary auxiliary information about users' inputs, each participant generates independent random noise from some sample space Ω, represented by $\mathbf{r} := (r_1, \ldots, r_n) \in \Omega^n$. Let $\chi : \mathcal{D} \times \Omega \to \mathcal{D}$ denote some randomization function allowing each participant to compute a noisy version of her data $\widehat{x}_i := \chi(x_i, r_i)$ before encrypting and uploading it to the aggregator. From the encrypted values of $\widehat{\mathbf{x}} := (\widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_n)$, the aggregator computes a noisy statistic $f(\widehat{\mathbf{x}})$, which should be close to the desired statistic $f(\mathbf{x})$. Throughout the remainder of the paper, we use hatted variables to denote the randomized versions of a participants' data (associated with some random $\mathbf{r}$ and randomization function χ), and we use non-hatted versions to denote the original data.

Remark 1 (More general setting)**.** *In general, each participant i can generate noise r_i according to her own distribution. Moreover, each pariticipant i can apply a different randomization function $\chi_i(x_i, r_i)$ to her data x_i. In an even more general setting, each participant may encrypt her data x_i and her randomness r_i separately before sending to the aggregator, who computes a randomized aggregate function $\widehat{f} : \mathcal{D}^n \times \Omega^n \to \mathcal{O}$ on the encrypted inputs.*

For simplicity, this paper considers the special case when $\widehat{f}(\mathbf{x}, \mathbf{r}) = f(\widehat{\mathbf{x}})$. Furthermore, we assume that each participant applies the same randomization χ function before encrypting her data.

Our goal is to design a privacy mechanism such that for every time period, the aggregator is able to learn some aggregate statistic $f(\widehat{\mathbf{x}})$, but not each individual's value even when it has arbitrary auxiliary information. We call a scheme that meets the above requirements a Private Stream Aggregation (PSA) mechanism. More formally, a Private Stream Aggregation scheme consists of the following algorithms.

$\mathsf{Setup}(1^\lambda)$: Takes in a security parameter λ, and outputs public parameters param, a private key sk_i for each participant, as well as a aggregator capability sk_0 needed for decryption of aggregate statistics in each time period. Each participant i obtains the private key sk_i, and the data aggregator obtains the capability sk_0.

$\mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_i, t, x, r)$: During time step t, each participant calls the $\mathsf{NoisyEnc}$ algorithm to encode its data x with noise r. The result is a noisy encryption of x randomized with the noise r. Without risk of ambiguity, we sometimes write $\mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_i, t, \widehat{x})$ where $\widehat{x} := \chi(x, r)$ is the noisy version of the participant's data, and χ is some underlying randomization function.

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, t, c_1, c_2, \ldots, c_n)$ The decryption algorithm takes in the public parameters param, a capability sk_0, and ciphertexts $c_1, c_2, \ldots, c_2$ for the same time period t. For each $i \in [n]$, let $c_i = \mathsf{NoisyEnc}(\mathsf{sk}_i, t, \widehat{x}_i)$, where each $\widehat{x}_i := \chi(x_i, r_i)$. Let $\mathbf{x} := (x_1, \ldots, x_n)$ and $\widehat{\mathbf{x}} := (\widehat{x}_1, \ldots, \widehat{x}_n)$. The decryption algorithm outputs $f(\widehat{\mathbf{x}})$ which is a noisy version of the targeted statistics $f(\mathbf{x})$.

3.1 The Case for Summation: Overview of Our Solution

In this paper, we mainly consider a simple but common statistic: summation. Each participant's data x_i comes from $\mathbb{Z}_p$ for some prime p. Define the aggregating function $\mathsf{sum}(\mathbf{x}) := \sum_{i=1}^{n} x_i$. Moreover, each participant generates noise r_i from the set of integers and applies the randomization function $\chi(x_i, r_i) := x_i + r_i \mod p$, i.e., a participant incorporates additive noise before encrypting her data.

Figure 3.1 gives a high-level overview of our construction. In the remainder of the paper, we first formalize the privacy notions in Section 4. We then describe the two building blocks in our solution in following two sections: 1) Section 5 describes a cryptographic construction to ensure that the aggregator learns nothing but the noisy sum; 2) Section 6 describes how each participant should choose her noise distribution so that the differential privacy of an individual participant is protected even when a subset of the participants may be compromised.

4 Formal Privacy Notions

We consider an untrusted aggregator who may have arbitrary auxiliary information. For example, the aggregator may collude with a set of corrupted partipants. The corrupted participants can reveal their data and noise values to the aggregator. Such information leaked from corrupted participants can be considered as a form of auxiliary information. Auxiliary information about participants can also be obtained in other ways, such as from public datasets on the web, or through personal knowledge about a specific participant.

Our goal is to guarantee the privacy of each individual's data against an untrusted aggregator, even when the aggregator has arbitrary auxiliary information. At a high level, our formal privacy notions consists of two properties:

- **Aggregator oblivious.** Suppose that an aggregator has auxiliary information aux. With an appropriate capability, the aggregator learns a noisy aggregate statistics $f(\widehat{\mathbf{x}})$ at the end of a time period. We

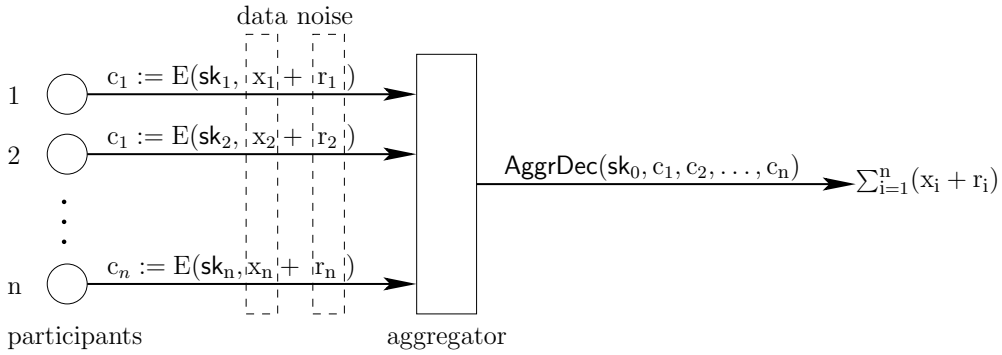


Figure 1: **Overview of our construction.** In every time period, each participant adds noise r_i to her value x_i before encrypting it. The aggregator uses the capability sk_0 to decrypt a noisy sum, but learns nothing more. The noisy sum output by this distributed mechanism ensures each participant's differential privacy.

would like to guarantee that the aggregator learns nothing other than what can be inferred from aux and the revealed statistics $f(\widehat{\mathbf{x}})$. In addition, we require that a party without an appropriate aggregator capability learns nothing.

- **Distributed differential privacy.** We require that a user's participation in the system leaks only negligible information about herself. In other words, the aggregate statistic $f(\widehat{\mathbf{x}})$ revealed is roughly the same whether or not a specific user participates in the system. To achieve this goal, we adopt a privacy model similar to the differential privacy notion first introduced by Dwork *et al.* [5, 7]. While traditional differential privacy considers a trusted aggregator who sees the participants' data in the clear and is trusted to add noise to the published statistics, in our model, the participants need not trust the data aggregator or other participants. The final noise in the revealed statistic is collected from each individual participant. Our privacy guarantee is strong, even when the aggregator may have arbitrary auxiliary information, or collude with a small subset of corrupted participants.

Malicious participants can also perform a data pollution attack where they lie about their values in an attempt to sway the final output. Although data pollution attacks are outside the scope of the paper, we would like to mention that one possible defense is for each participant to use a non-interactive zero-knowledge proof to prove that her encrypted data lies within a valid range, e.g., $\{0, 1, \ldots, \Delta\}$. In this way, each participants's influence is bounded.

4.1 Aggregator Oblivious

For simplicity, we define the aggregator oblivious notion of security only for the sum statistic. Intuitively, we would like to capture the following security notions:

- The aggregator can learn only the noisy sum for each time period, and nothing more. For example, the aggregator cannot learn any partial information from a proper subset of all participants' ciphertexts.
- Without knowing the aggregator capability, one learns nothing about the encrypted data, even if several participants form a coalition against the remaining users.
- If the aggregator colludes with a subset of the participants, or if a subset of the encrypted data has been leaked, then the aggregator can inevitably learn the sum of the remaining participants. We require that in this case, the aggregator learns no additional information about the remaining participants' data.

We describe the following Aggregator Oblivious (AO) security game. We assume that each participant incorporates additive noise (which we see later is to ensure privacy) to their data before encrypting them.

Setup. Challenger runs the Setup algorithm, and returns the public parameters param to the adversary.

Queries. The adversary makes the following types of queries adaptively. As described later, certain constraints must be met for these queries.

- **Encrypt.** The adversary may specify (i, t, x, r), and ask for the ciphertext. The challenger returns the ciphertext $\mathsf{NoisyEnc}(\mathsf{sk}_i, t, x, r)$ to the adversary.

- **Compromise.** The adversary specifies an integer $i \in \{0, \ldots, n\}$. If $i = 0$, the challenger returns the aggregator capability sk_0 to the adversary. If $i \neq 0$, the challenger returns sk_i, the secret key for the i^{th} participant, to the adversary.
- **Challenge.** This query can be made only once throughout the game. The adversary specifies a set of participants U and a time t^*. Any $i \in U$ must not have been compromised at the end of the game.

  For each user $i \in U$, the adversary chooses two plaintext-randomness pairs $(x_i, r_i), (x'_i, r'_i)$. The challenger flips a random bit b. If $b = 0$, the challenger computes $\forall i \in U : \mathsf{NoisyEnc}(sk_i, t, x_i, r_i)$, and returns the ciphertexts to the adversary. If $b = 1$, the challenger computes and returns the ciphertexts $\forall i \in U : \mathsf{NoisyEnc}(sk_i, t, x'_i, r'_i)$ instead.

Guess. The adversary outputs a guess of whether b is 0 or 1.

We say that the adversary wins the game if she correctly guesses b and the following condition holds. Let $K \subseteq [n]$ denote the set of compromised participants at the end of the game (not including the aggregator). Let $Q \subseteq [n]$ denote the set of participants for whom an **Encrypt** query has been made on time t^* by the end of the game. Let $U \subseteq [n]$ denote the set of (uncompromised) participants specified in the **Challenge** phase. If $U = \overline{K \cup Q} := [n] \backslash (K \cup Q)$, and the adversary has compromised the aggregator capability, the following condition must be met:

$$\sum_{i \in U} \widehat{x}_i = \sum_{i \in U} \widehat{x}'_i. \tag{1}$$

Definition 1 (Aggregator oblivious security)**.** *A PSA scheme is aggregator oblivious, if no probabilistic polynomial-time adversary has more than negligible advantage in winning the above security game.*

Explanation. Suppose that the adversary has compromised the aggregator capability sk_0. In addition, for every participant $i \notin U$, the adversary knows a ciphertext c_i for the time t^* as well as the corresponding randomized plaintext $\widehat{x}_i$. Such an adversary is able to use the AggrDec function to learn the sum of all participants in time period t^*. From this sum, the adversary is able to infer the partial sum over the subset U. Note that the the adversary may be able to learn a plaintext and ciphertext pair for $i \notin U$ in two ways. The adversary can either make an **Encrypt** query for $i \notin U$ and time t^*, or compromise the secret key of participant i so that it is able to produce the ciphertexts on its own. Therefore, when $U = \overline{K \cup Q}$ and the aggregator capability has been compromised, we require that apart from the sum over the subset U, the adversary is unable to infer additional information about the honest participants in U. This means that the adversary in the above security game is unable to distinguish which plaintext vector $\widehat{\mathbf{x}}_U := \{\widehat{x}_i | i \in U\}$ or $\widehat{\mathbf{x}}'_U := \{\widehat{x}'_i | i \in U\}$ the challenger encrypted, as long as $\widehat{\mathbf{x}}_U$ and $\widehat{\mathbf{x}}'_U$ are equivalent with respect to summation.

On the other hand, under the following conditions, the adversary learns nothing from the challenge ciphertexts corresponding to the set U of participants. 1) The adversary has not compromised the aggregator capability; or 2) $U \neq \overline{K \cup Q}$, i.e., there exists at least one $i \notin U$ for whom the adversary does not know a ciphertext for time period t^*. Under these situations, for arbitrary choices of $\widehat{\mathbf{x}}_U$ and $\widehat{\mathbf{x}}'_U$ that the adversary submits in the **Challenge** phase, the adversary is unable to distinguish which one the challenger encrypted.

Remark 2 (General statistic)**.** *The notion of aggregator oblivious security may be extended to general statistics other than sum. Extra care must be taken, however. If an adversary has compromised the set $K \subseteq [n]$ of participants, she is able to encrypt anything on behalf of these participants. Therefore, she can plug in any plaintext vector $\widehat{\mathbf{x}}_K = \{\widehat{x}_i | i \in K\}$ of her choice for the set K, encrypt them, and then call the* AggrDec *function to decrypt the aggregate statistics conditioned on $\widehat{\mathbf{x}}_K$. Such attacks are inherent in the problem definition and cannot be avoided. The security definition must reflect the fact that this is the best and only strategy for the adversary, i.e., the adversary is unable to learn extra information other than information gleaned from this attack. For the sum statistic, this requirement boils down to Equation 1. Basically, as long as the two challenge plaintexts are equivalent with respect to sum, the adversary is unable to distinguish which one the challenger encrypted. This condition is more tricky to state for general queries. For simplicity, this paper defines the aggregator oblivious security game specifically for the sum statistic.*

Encrypt-once security. Our construction makes one additional assumption that each honest participant only encrypts once in each time period. Formally, this condition is reflected in the game as follows.

Definition 2 (Encrypt-once security)**.** *We say that a PSA scheme is aggregator oblivious in the "encrypt-once" model, if no probabilistic polynomial-time adversary has more than negligible advantage in the above security game, and in addition, the following constraint holds:*

$\forall i \in U$, $\forall (x, r) \in \mathcal{D} \times \Omega$: *the tuple* (i, t^*, x, r) *must not have appeared in any* ***Encrypt*** *query.*

4.2 Distributed Differential Privacy

Previous differential privacy literature assumes that all users send their data to the aggregator in the clear. In this case, if the users wish to guarantee their privacy against an untrusted aggregator, each participant must add sufficient noise to her value to ensure her differential privacy. As a result, the aggregate noisy statistic may accumulate too much noise, and the resulting $f(\widehat{\mathbf{x}})$ may have a huge error. In contrast, we guarantee that the aggregator learns only the noisy statistic, but not each individual's values. In this way, each individual may add less noise to her data. As long as the final statistic $f(\widehat{\mathbf{x}})$ has accumulated sufficient randomness, each individual's privacy is guaranteed. We also consider the case when a certain fraction of participants are compromised. The compromised participants can collude with the data aggregator and reveal their data or randomness to the aggregator. In this case, we would like to ensure that the remaining uncompromised participants' randomness is sufficient to protect their privacy.

We referred to the above notion of privacy as Distributed Differential Privacy (DD-Privacy), to reflect the fact that the noise in the the released statistic is collected from all participants. We formalize this notion of distributed differential privacy below.

Recall that the aggregator evaluates a function $f : \mathcal{D}^n \to \mathcal{O}$ on randomized data $\widehat{\mathbf{x}} \in \mathcal{D}^n$ of n participants, which are generated in the following way. Each participant generates independent randomness $r_i \in \Omega$ according to some distribution, and apply some randomization function $\chi : \mathcal{D} \times \Omega \to \mathcal{D}$ on her data x_i to produce $\widehat{x}_i := \chi(x_i, r_i)$. Given $\mathbf{x} \in \mathcal{D}^n$ and $\mathbf{r} \in \Omega^n$, we use the notation $\widehat{\mathbf{x}} = \widehat{\mathbf{x}}(\mathbf{r}) := (\chi(x_1, r_1), \chi(x_2, r_2), \ldots, \chi(x_n, r_n))$, i.e., the dependence of $\widehat{\mathbf{x}}$ on $\mathbf{r}$ is implicit.

Given a subset K of participants, we let $\mathbf{r}_K := \{r_i : i \in K\}$ and $\overline{K}$ be the complement of K, i.e., $\overline{K} = \{1, 2, \ldots, n\} \setminus K$.

We require that the following notion of distributed differential privacy applies to *every time period* $t \in \mathbb{N}$.

Definition 3 ((ϵ, δ)-DD-Privacy)**.** *Suppose* $\epsilon > 0$, $0 \leq \delta < 1$ *and* $0 < \gamma \leq 1$. *We say that the data randomization procedure, given by the joint distribution* $\mathbf{r} := (r_1, \ldots, r_n)$ *and the randomization function* χ *achieves* (ϵ, δ)-distributed differential privacy *(DD-privacy) with respect to the function* f *and under* γ *fraction of uncompromised participants if the following condition holds. For any neighboring vectors* $\mathbf{x}, \mathbf{y} \in \mathcal{D}^n$*, for any subset* $S \subseteq \mathcal{O}$*, and for any subset* $\overline{K}$ *of uncompromised participants of size at least* γn*,*

$$Pr[f(\widehat{\mathbf{x}}) \in S | \mathbf{r}_K] \leq \exp(\epsilon) \cdot Pr[f(\widehat{\mathbf{y}}) \in S | \mathbf{r}_K] + \delta. \tag{2}$$

In the above definition, two vectors $\mathbf{x}, \mathbf{y} \in \mathcal{D}^n$ are said to be *neighbors* or *neighboring vectors* if they differ in exactly one coordinate. This corresponds to the scenario when exactly one user changes her data.

When K is the set of compromised nodes, the above definition requires that the remaining honest participants' randomness be sufficient to ensure differential privacy. Therefore, the probability is conditioned on the randomness $\mathbf{r}_K$ from compromised participants. In other words, the probability is taken over the randomness $\mathbf{r}_{\overline{K}}$ from honest participants. The definition of DD-privacy requires that for any set $\overline{K}$ of uncompromised participants, as long as $|\overline{K}| \geq \gamma n$, Equation 2 holds.

Strictly speaking, we achieve differential privacy against polynomial-time adversaries, as our constructoin relies on an encryption scheme that is secure against polynomial-time adversaries. Computational differential privacy was first introduced by Mironov *et al.* [13]. In fact, it is possible to define a computational version of the above DD-privacy notion as well, and prove our scheme secure under the computational differential privacy model. We leave the computational definition and proofs to the expanded journal version.

5 Achieving Aggregator Oblivious Security

In this section, we describe a cryptographic construction that allows us to achieve aggregator oblivious security. For simplicity, in this section, we assume that each participant incorporates additive noise r to her data x before encrypting it. To avoid writing the plaintext and noise terms separately, we use the notation $\widehat{x}_{i,t} := x_{i,t} + r_{i,t}$ to denote participant i's noisy plaintext in time t. When the context is clear, we omit one or more of the subscripts and write $\widehat{x}_i$ or $\widehat{x}$ instead.

5.1 Intuition

One challenge we face when designing the mechanism is how to minimize the necessary communication between the participants and the data aggregator. If one allows the participants and the aggregator to engage in an interactive multiple-party protocol in every time period, then standard Secure Multi-Party Computation [10] techniques can be used to ensure that the data aggregator learns only the sum. However, the requirement that all participants must be simultaneously online and interact with each other perodically renders many

applications impractical, especially large-scale cloud applications. In contrast, in our solution, after a trusted setup phase between all participants and the data aggregator, no further interaction is required except for uploading a noisy encryption to the data aggregator in each time period. The trusted setup may be performed by a trusted third-party or through a standard Secure Multi-Party protocol.

We now explain the intuition behind our construction. Suppose that for every time period $t \in \mathbb{N}$, the participants and the aggregator had means of determining $n+1$ random shares of 0. In other words, they generate $\rho_{0,t}, \rho_{1,t}, \ldots, \rho_{n,t} \in \mathbb{Z}_p$, such that

$$\sum_{i=0}^{n} \rho_{i,t} = 0.$$

Specifically, $\rho_{0,t}$ is the aggregator's capability for time t, and participants 1 through n obtain $\rho_{1,t}$ through $\rho_{n,t}$ respectively. Then the following simple idea allows the aggregator to decrypt the sum of all participants for all time periods, without learning each individual's values.

NoisyEnc. To encrypt the value $\widehat{x}_{i,t} := x_{i,t} + r_{i,t}$ in time period t, participant i simply computes the following ciphertext

$$c_{i,t} = \widehat{x}_{i,t} + \rho_{i,t}.$$

AggrDec. At time $t \in \mathbb{N}$, the aggregator receives $c_{1,t}, \ldots, c_{n,t}$. The aggregator may obtain the plaintext simply by summing up these ciphertexts and its capability $\rho_{0,t}$.

$$V \leftarrow \rho_{0,t} + \sum_{i=1}^{n} c_{i,t}.$$

Since $\sum_{i=0}^{n} \rho_{i,t} = 0$, the aggregator obtains $V = \sum_{i=1}^{t} \widehat{x}_i$ as the desired sum.

The question is how participants and the aggregator can obtain random shares of 0 without having to interact with each other in every time period. Our scheme relies on a trusted setup phase during which each participant obtains a secret key sk_i where $i \in [n]$, and the aggregator obtains a capability sk_0. Moreover, $\sum_{i=0}^{n} \mathsf{sk}_i = 0$. Let H denote a hash function (modeled as a random oracle) that maps an integer to an appropriate mathematical group. In every time period t, each participant computes $R_{i,t} = H(t)^{\mathsf{sk}_i}$ for $i \in [n]$, and the aggregator computes $R_{0,t} = H(t)^{\mathsf{sk}_0}$. Since the sk_i sum to zero, $\prod_{i=0}^{n} R_{i,t} = 1$. We leverage this property to construct a scheme in which the participants never have to communicate with each other after the trusted setup phase. Furthermore, if Decisional Diffie-Hellman is hard in the mathematical group in question, we prove that the numbers $R_{i,t}$ are "seemingly" random under the random oracle model.

5.2 Basic Construction

Let $\mathbb{G}$ denote a cyclic group of prime order p for which Decisional Diffie-Hellman is hard. Let $H : \mathbb{Z} \to \mathbb{G}$ denote a hash function modelled as a random oracle.

$\mathsf{Setup}(1^\lambda)$. A trusted dealer chooses a random generator $g \in \mathbb{G}$, and $n+1$ random secrets $s_0, s_1, \ldots, s_n \in \mathbb{Z}_p$ such that $s_0 + s_1 + s_2 + \ldots + s_n = 0$. The public parameters $\mathsf{param} := g$. The data aggregator obtains the capability $\mathsf{sk}_0 := s_0$, and participant i obtains the secret key $\mathsf{sk}_i := s_i$.

$\mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_i, t, \widehat{x})$. For participant i to encrypt a value $\widehat{x} \in \mathbb{Z}_p$ for time step t, she computes the following ciphertext:

$$c \leftarrow g^{\widehat{x}} \cdot H(t)^{sk_i}$$

Becuase we assume that each participant adds noise to her data before encryption, we use the term $\widehat{x} := x + r \mod p$ to denote the randomized plaintext.

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, t, c_1, c_2, \ldots, c_n)$. Compute

$$V \leftarrow H(t)^{\mathsf{sk}_0} \prod_{i=1}^{n} c_i.$$

Suppose $c_i = \mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_0, t, \widehat{x}_i)$ for $i \in [n]$. It is not hard to see that V is of the form

$$V = g^{\sum_{i=1}^{n} \widehat{x}_i}.$$

To decrypt the sum $\sum_{i=1}^{n} \widehat{x}_i$, it suffices to compute the discrete log of V base g. When the plaintext space is small, decryption can be achieved through a brute-force search. A better approach is to use Pollard's lambda method [12] which requires decryption time roughly square root in the plaintext space. For example, suppose each participant's input is in the range $\{0, 1, \ldots, \Delta\}$. Then the sum of the participants fall within the range $\{0, 1, \ldots, n\Delta\}$. In this case, decryption would require $\sqrt{n\Delta}$ time using Pollard's method. In other words, we require that $n\Delta$ is polynomial in the security parameter λ to ensure successful decryption in polynomial time. Note that the limitation of small plaintext space is in fact typical of Diffie-Hellman-based encryption schemes when used as additively homomorphic encryption schemes, e.g,

El Gamal encryption and the BGN homomorphic encryption scheme [2].

The careful reader may now question how each participant picks noise to add to her data. In Section 6, we show that it is possible to pick an appropriate distribution of noise that guarantees differential privacy and meanwhile ensures the ability to decrypt with high probability.

Theorem 1. *Assuming that the Decisional Diffie-Hellman problem is hard in the group $\mathbb{G}$ and that the hash function H is a random oracle, then the above construction satisfies aggregator oblivious security in the "encrypt-once" model.*

We present the proof of Theorem 1 in Appendix A.

Practical performance. In the proposed cryptographic construction, encryption consists of a hash operation (e.g., SHA-256), two modular exponentiations and one multiplication in a Diffie-Hellman group. The running time is dominated by the two modular exponentiations, as the time for computing the hash function and group multiplication are much smaller in comparison with the time for an expenentiation. According to benchmarking numbers reported by the eBACS project [1], on a modern 64-bit desktop PC, it takes roughly 3 *ms* to compute a modular exponentiation using a classic Diffie-Hellman group modular a 1024-bit prime. Using high-speed elliptic curves such as "curve25519", it takes only 0.3 *ms* to compute a modular exponentiation. Therefore, encryption can be done in roughly 0.6 *ms* on a modern computer. Decryption of the aggregate statistics requires taking a discrete log, and if one uses the brute-force method, it takes one modular exponentiation, that is 0.3 *ms* to try each possible plaintext. Therefore, our scheme is practical in situations where the plaintext space is small. For example, in the application described by Rieffel *et al.* [17], each participant's plaintext is a bit indicating her availability for communication. In this case, with roughly 1000 participants, decryption can be done in about 0.3 *s* using the brute-force approach. We can have a further speed-up if we adopt Pollard's lambda method for decryption, reducing the running time to about $\sqrt{n\Delta}$, where n is the number of participants, and assuming each participant's value comes from $\{0, 1, \ldots, \Delta\}$.

6 Achieving Distributed Differential Privacy

6.1 Inituition

The cryptographic construction of Section 5 ensures that the aggregator learns nothing other than what it already knows and the noisy statistic revealed during each time period. Therefore, the aggregator has no direct access to each individual's data. Individual privacy can be violated indirectly, however, as the revealed statistic may enable deductions about an individual's data. In this section, we show how to build a guarantee of (ϵ, δ)-differential privacy into the cryptographic construction.

In previous differential privacy literature, a trusted aggregator is responsible for releasing statistics. The trusted aggregator has access to all data, and is charged with meeting privacy guarantees when releasing data. A standard procedure for ensuring differential privacy is for the aggregator to add an appropriate magnitude of noise before publishing the desired statistic.

In our case, the participants do not trust the aggregator. Therefore, we cannot reveal the true statistic to the aggregator. Instead, we must add noise before the aggregator is able to decrypt the statistic. Our approach is to let the participants be responsible for ensuring the differential privacy of their own data. Each participant would add noise to their data before encrypting them. We need to address the following two challenges when designing a differentially private mechanism:

- **Compromised participants.** To ensure the differential privacy for participating individuals, the revealed statistic must contain random noise r of an appropriate magnitude. One naive solution is to rely on a single participant to add an appropriate magnitude of noise r to her data before submission. However, this solution is problematic, because this designated user knows the noise and hence can deduce from the output the true aggregated value. In real-world settings, participants may not trust each other. In particular, a subset of the participants may be compromised and collude with the data aggregator. In the worst case, if every participant believes that the other $n - 1$ participants may be compromised and collude with the aggregator, each participant would need to add sufficient noise to ensure the privacy of her own data. The resulting statistic would accumulate a big error.

  If at least γ fraction of the participants are honest and not compromised, then we can distribute the noise generation task amongst these participants. Each participant may add less noise, and as long

as the noise in the final statistic is large enough, individual privacy is protected. Our scheme assumes that the participants have an apriori estimate on the lower bound for γ. However, they need not know exactly which participants are compromised. Each participant is supposed to generate noise from a distribution that depends on γ. Honest participants will follow this protocol, but the compromised participants may reveal their noise to the data aggregator or choose not to add noise. Our construction guarantees that, with high probability, the revealed statistic will accumulate sufficient noise from the honest participants, while keeping the error of the final statistic small.

- **Algebraic constraints.** Another challenge is to work within the algebraic constraints induced by the cryptographic construction. Most encryption schemes require that the plaintext be picked from a group comprised of discrete elements. Therefore, we need to be able to encode our data and noise values in a discrete group. Moreover, the cryptographic construction proposed in Section 5 imposes one more constraint, that the plaintext space must be small. To work with discrete groups, we choose to use a symmetric geometric distribution instead of the more commonly used Laplace distribution.

  The symmetric geometric distribution is unbounded, so it may overflow the size of the group, or the size of the plaintext space. Our construction ensures that the probability of an overflow is small, so that the aggregator can successfully decrypt the noisy statistics with high probability.

Next, we introduce some preliminaries on differential privacy, and then detail our construction.

6.2 Differential Privacy Preliminaries

The differential privacy literature commonly adds noise sampled from a Laplace distribution to the true output to ensure individual privacy. However, as pointed out earlier, because the encryption scheme uses discrete groups, we need a discrete distribution instead. We use symmetric geometric distribution, which can be regarded as a discrete approximation to the Laplace distribution. The use of geometric distribution for the noise was pioneered by Ghosh *et al.* [9]. We now provide some background on the geometric distribution.

Definition 4 (Geometric Distribution)**.** *Let $\alpha > 1$. We denote by $\mathsf{Geom}(\alpha)$ the* symmetric geometric distribution *that takes integer values such that the probability mass function at k is $\frac{\alpha-1}{\alpha+1} \cdot \alpha^{-|k|}$.*

We denote by $\mathsf{Geom}^{+}(\alpha)$ the one-sided geometric distribution *that takes positive integer values such that the probability mass function at k is $(\alpha-1)\alpha^{-k}$.*

The symmetric geometric distribution $\mathsf{Geom}(\alpha)$ can be viewed as a discrete version of the Laplace distribution $\mathsf{Lap}(b)$ (where $\alpha \approx \exp(\frac{1}{b})$), whose probability density function is $x \mapsto \frac{1}{2b}\exp(-\frac{|x|}{b})$. The following property of Geom distribution is useful for designing differentially private mechanisms that output integer values.

Fact 1. *Let $\epsilon > 0$. Suppose u and v are two integers such that $|u - v| \leq \Delta$. Let r be a random variable having distribution $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$. Then, for any integer k, $Pr[u + r = k] \leq \exp(\epsilon) \cdot Pr[v + r = k]$.*

Fact 1 suggests that if the targeted statistic $f(\mathbf{x})$ has sensitivity Δ, then adding geometric noise with magnitude proportional to Δ is sufficient to achieve differential privacy. As mentioned before, participants do not trust the aggregator or each other. As a result, we cannot entrust the aggregator with the task of noise generation, since revealing the true statistic to the aggregator clearly violates differential privacy. Neither can we entrust any single participant with this task, since otherwise, this designated participant would be able to learn true statistic as well.

6.3 Achieving DD-Privacy for Summation

Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{D}^n$ and $\mathbf{r} = (r_1, \ldots r_n) \in \Omega^n$ represent the data and noise values respectively from all participants in a certain time period. Here we have $\mathcal{D} = \mathcal{O} = \mathbb{Z}_p$, the cyclic group equipped with addition modulo p, and $\Omega = \mathbb{Z}$. We consider the aggregating function $\mathsf{sum} : \mathcal{D}^n \to \mathcal{O}$, with $\mathsf{sum}(\mathbf{x}) = \sum_{i=1}^{n} x_i \mod p$. Each participant uses the same randomization function $\chi(x_i, r_i) := x_i + r_i \mod p$.

For any two elements $u, v \in \mathbb{Z}_p$, we define $|u - v|$ to be the smallest non-negative integer s such that $u = v + s \mod p$ or $v = u + s \mod p$. Moreover, when we add an integer to an element in $\mathbb{Z}_p$, we assume that addition is performed modulo p.

We assume that each participant's original data falls within the domain $\{0, 1, \ldots, \Delta\}$, and hence the sensitivity of sum is Δ with respect to one participant's change. In other words, if a single participant changes her data, the sum changes by at most Δ. Recall from Fact 1 that if a $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$ noise is incorporated into the output, then ϵ-differential privacy is achieved. In our case, the participants jointly generate the noise in the final output. Our goal is to ensure that if at least γn participants are honest and uncompromised, we will accumulate noise of a similar magnitude. In this way, we not only guarantee

Algorithm 1: DD-Private Data Randomization Procedure.

Let $\alpha := \exp(\frac{\epsilon}{\Delta})$ and $\beta := \frac{1}{\gamma n}\log\frac{1}{\delta}$.

Let $\mathbf{x} = (x_1, \ldots x_n)$ denote all participants' data in a certain time period.

foreach participant $i \in [n]$ **do**

Sample noise r_i according to the following distribution.

$$r_i \leftarrow \begin{cases} \mathsf{Geom}(\alpha) & \text{with probability } \beta \\ 0 & \text{with probability } 1-\beta \end{cases}$$

Randomize data by computing $\widehat{x}_i \leftarrow x_i + r_i \mod p$.

differential privacy, but also ensure that the accumulated noise is bounded in the final output so that the error is small.

Informally, our mechanism guarantees (ϵ, δ)-DD-privacy, and meanwhile ensures small error of roughly $O(\frac{\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}})$ magnitude. As long as a constant fraction γ of participants are honest, the error term is independent of n, the number of participants. In fact, our result is nearly optimal, since an accumulated noise of magnitude $\Theta(\frac{\Delta}{\epsilon})$ is necessary to ensure differential privacy. Furthermore, consider the extreme case when $\gamma = O(\frac{1}{n})$, i.e., each participant believes that all other participants may be compromised, or only a constant number of them are honest. Then, our accumulated noise would be $O(\frac{\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}}) = O(\frac{\Delta}{\epsilon}\sqrt{n})$. This agrees with our intuition as well, since each participant must add a symmetric noise of magnitude $\Theta(\frac{\Delta}{\epsilon})$ in this case to ensure her privacy. It is not hard to show that the sum of n independent symmetric noises of magnitude $\Theta(\frac{\Delta}{\epsilon})$ results in a final noise of magnitude $O(\frac{\Delta}{\epsilon}\sqrt{n})$ with high probability.

Below, we first state the main theorem of this section, and then describe our construction.

Theorem 2 (DD-Private Procedure with Low Error)**.** *Let $\epsilon > 0$ and $0 < \delta < 1$. Suppose each participant's data comes from integers inside an interval of width Δ in $\mathbb{Z}_p$, where $\Delta \geq \frac{\epsilon}{3}$. Suppose at least γ fraction of the n participants are uncompromised such that $\gamma \geq \frac{1}{n}\log\frac{1}{\delta}$. Then, there is a randomized procedure to generate $\mathbf{r} = (r_1, \ldots, r_n)$ that is (ϵ, δ)-DD-private with respect to* sum*. Moreover, for all $\mathbf{x} \in (\mathbb{Z}_p)^n$, for all $0 < \eta < 1$ such that $\log\frac{2}{\eta} \leq \frac{1}{\gamma}\log\frac{1}{\delta}$, with probability at least $1-\eta$ over the random choice of $\mathbf{r}$, $|\mathsf{sum}(\mathbf{x}) - \mathsf{sum}(\widehat{\mathbf{x}})| \leq \frac{4\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}\log\frac{1}{\delta}\log\frac{2}{\eta}}$, where $\widehat{\mathbf{x}} := \widehat{\mathbf{x}}(\mathbf{r}) := (x_1 + r_1, x_2 + r_2, \ldots, x_n + r_n) \mod p$.*

Algorithm 1 describes a procedure that achieves the guarantee in Theorem 2. We give the analysis in Lemma 1 and Theorem 3.

Lemma 1. *Let $\epsilon > 0$ and $0 < \delta < 1$. Suppose at least γ fraction of participants are uncompromised. Then, the above randomization procedure achieves (ϵ, δ)-DD-privacy with respect to* sum*, for $\beta = \min\{\frac{1}{\gamma n}\log\frac{1}{\delta}, 1\}$.*

Proof. Let $\overline{K}$ be a set of uncompromised users of size at least γn. Let B be the bad event that none of the users from $\overline{K}$ samples from $\mathsf{Geom}(\alpha)$. Observe that the event B is independent from the random variables $\mathbf{r}_K$. It follows that $Pr[B] \leq (1-\beta)^{\gamma n} \leq \exp(-\beta\gamma n)$. We use the inequality that $1 - t \leq \exp(-t)$ for all reals t. By the choice of β, this probability is at most δ.

Let $\mathbf{x}, \mathbf{y}$ be neighboring vectors, with each coordinate having range at most Δ. Let S be some subset of $\mathbb{Z}_p$. Conditioning on the good event $\overline{B}$ and $\mathbf{r}_K$, we know there is at least one independent copy of $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$ incorporated into the final sum, hence, by Fact 1, we have $Pr[\mathsf{sum}(\widehat{\mathbf{x}}) \in S|\mathbf{r}_K, \overline{B}] \leq \exp(\epsilon) \cdot Pr[\mathsf{sum}(\widehat{\mathbf{y}}) \in S|\mathbf{r}_K, \overline{B}]$. Finally, we have

$$\begin{aligned}
& Pr[\mathsf{sum}(\widehat{\mathbf{x}}) \in S|\mathbf{r}_K] \\
= \; & Pr[\mathsf{sum}(\widehat{\mathbf{x}}) \in S \cap \overline{B}|\mathbf{r}_K] + Pr[\mathsf{sum}(\widehat{\mathbf{x}}) \in S \cap B|\mathbf{r}_K] \\
\leq \; & Pr[\overline{B}|\mathbf{r}_K] \cdot Pr[\mathsf{sum}(\widehat{\mathbf{x}}) \in S|\mathbf{r}_K, \overline{B}] + Pr[B|\mathbf{r}_K] \\
\leq \; & Pr[\overline{B}|\mathbf{r}_K] \cdot \exp(\epsilon) \cdot Pr[\mathsf{sum}(\widehat{\mathbf{y}}) \in S|\mathbf{r}_K, \overline{B}] + Pr[B] \\
= \; & \exp(\epsilon) \cdot Pr[\mathsf{sum}(\widehat{\mathbf{y}}) \in S \cap \overline{B}|\mathbf{r}_K] + Pr[B] \\
\leq \; & \exp(\epsilon) \cdot Pr[\mathsf{sum}(\widehat{\mathbf{y}}) \in S|\mathbf{r}_K] + \delta,
\end{aligned}$$

hence proving the DD-privacy of the randomization procedure. □

6.4 Analyzing Utility

We next analyze how much the noisy statistic $\mathsf{sum}(\widehat{\mathbf{x}})$ deviates from the true output $\mathsf{sum}(\mathbf{x})$. Since $|\mathsf{sum}(\widehat{\mathbf{x}}) - \mathsf{sum}(\mathbf{x})| \leq |\sum_{i=1}^n r_i|$, it suffices to bound the magnitude of $Z := \sum_{i=1}^n r_i$.

Theorem 3 (Bounding Error)**.** *Let $\epsilon > 0$ and $0 < \delta < 1$. Suppose each participant's data comes from integers inside an interval of width Δ, where $\Delta \geq \frac{\epsilon}{3}$. Suppose at least $\gamma \geq \frac{1}{n}\log\frac{1}{\delta}$ fraction of the n participants are uncompromised. Suppose the randomized procedure described in Section 6.3 is run to produce $\mathbf{r} :=$*

$(r_1, \ldots, r_n)$ *with* $\alpha := \exp(\frac{\epsilon}{\Delta})$ *and* $\beta := \frac{1}{\gamma n} \log \frac{1}{\delta} \leq 1$. *Then, for all* $0 < \eta < 1$ *such that* $\log \frac{2}{\eta} \leq \frac{1}{\gamma} \log \frac{1}{\delta}$*, with probability at least* $1 - \eta$*,*

$$|\sum_{i=1}^{n} r_i| \leq 4\sqrt{\frac{1}{\gamma} \log \frac{1}{\delta} \log \frac{2}{\eta}} \cdot \frac{\sqrt{\alpha}}{\alpha - 1} \leq \frac{4\Delta}{\epsilon} \sqrt{\frac{1}{\gamma} \log \frac{1}{\delta} \log \frac{2}{\eta}}.$$

According to Theorem 3, the accumulated error is bounded by $O(\frac{\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}})$ with high probability. Suppose each participant's value is picked from the domain $\mathcal{D} = \{0, \ldots, \Delta\}$. Then, the aggregator simply has to try to decrypt the sum within the range $[-O(\frac{\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}}), n\Delta + O(\frac{\Delta}{\epsilon}\sqrt{\frac{1}{\gamma}})] \mod p$, where p is the size of the mathematical group in use. Decryption will succeed with high probability.

Theorem 3 is a measure concentration result, and we prove it by analyzing the moment generating function of each r_i. Observe that as long as there is a constant fraction γ of uncompromised participants, the error bound is independent of n. Because the variance of $\mathsf{Geom}(\alpha)$ distribution is $\frac{2\alpha}{(\alpha-1)^2}$, with high probability the error is at most a constant factor worse than adding one copy of $\mathsf{Geom}(\alpha)$ to the final answer, which is, in the sense described in [9], the minimal amount of noise required to ensure ϵ-differential privacy. We provide a proof of Theorem 3 in Appendix B.

Empiricial error. Figure 6.4 shows simulation result where we compare our scheme against a naive scheme. In the simulation, we assume that each participant's input is a bit from $\{0, 1\}$. The naive scheme is where each participant adds independent geometric noise to her input, and uploads the perturbed data to the aggregator. The naive scheme ensures differential privacy, but not aggregator obliviousness. We varied the number of participants n, and compare the utility of our scheme and the naive scheme under fixed privacy parameters ($\epsilon = 0.1$ and $\epsilon = 0.5$ respectively). For each value of n, we compute the mean and standard deviation over 200 runs. γ is set to be 1 in both plots, i.e., assuming no compromised users. The simulation shows that the error of our scheme is independent of the number of participants, thereby confirming our theoretic analysis.

7 Extensions and Variants

Evaluating distributions. Analysts often would like to study distributions over a population. Our scheme can be extended to allow the aggregator to periodically evaluate the (approximate) distribution of n participants' data. For example, suppose that the distribution is known to be a Gaussian, then it suffices for each participant to encrypt the orginal value as well as its square. It is not hard to see that the aggregator can then recover the distribution through the mean and the variance (or second moment). For other distributions, the participants may need to encrypt higher moments as well. In general, the more moments each participant encrypts, the better the aggregator is able to estimate the distribution.

Public access to a statistic. A slight variant on our scheme enables public access to the sum, but not to individual values. In this variant, we simply set the aggregator capability $\mathsf{sk}_0 = 0$, essentially making this capability public. The n participants receive values $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ that add up to zero. Encryption and decryption of aggregate statistcs are done as before. To obtain the aggregate sum, a discrete log must be computed, so again the plaintext space must be small.

Multiple-level hierarchies. The protocol can be nested in order to support access control hierarchies, as described in Rieffel *et al.* [17], in which entities at higher levels have access only to statistics pooled over all leaf nodes under them. In the setup phase, an entity at level $j > 1$ is given the sum of the secrets of the entities at the level below. (For $j = 1$, each entity above the leaf nodes is given the negative of the sum of the secrets of the participants below it, as is done in the basic construction.)

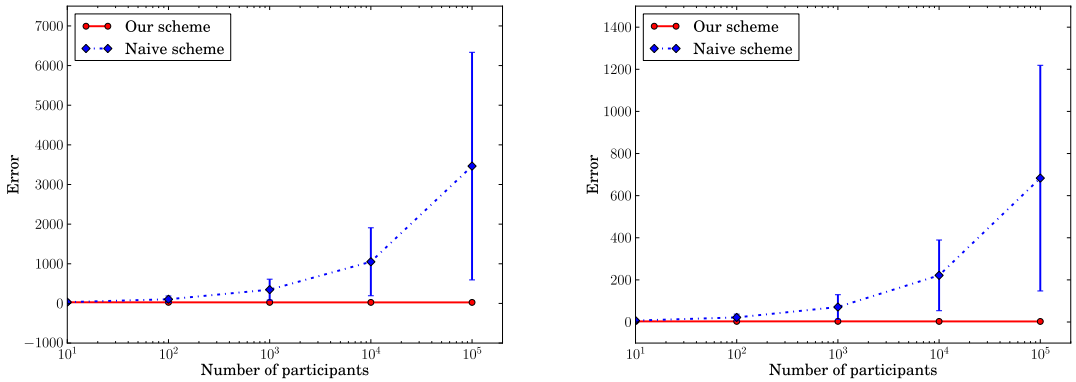
Product, rather than sum. The basic construction can be easily modified to support oblivious computation of a product instead of a sum. Simply encrypt χ as $c \leftarrow \chi \cdot H(t)^{sk_i}$. Because the plaintext is no longer in the exponent, this scheme for products does not suffer from the small plaintext restriction.

8 Open Research Challenges

This paper defined a new problem – how an untrusted data aggregator can compute aggregate statistics over ciphertexts from multiple sources, while preserving each individual's privacy in a strong sense. We formally defined privacy notions, and demonstrated a construction allowing the aggregator to compute the sum statistic for time series data.

This paper leaves open several problems that are both intriguing and challenging. We hope to inspire future research efforts in this new direction, especially in solving the challenges stated below.

Construction supporting large plaintext space. One limitation of our cryptographic construction is that it



(a) $\epsilon = 0.1, \delta = 0.001$ (b) $\epsilon = 0.5, \delta = 0.1$

Figure 2: **Empirical error of our scheme and the naive scheme.** The x-axis is the number of participants, and the y-axis shows the mean and standard deviation of error (in absolute value). Each participant's data is a bit from $\{0, 1\}$. The naive scheme is where each participant adds independent geometric noise to her input and uploads the perturbed data to the aggregator.

supports only polynomial-sized plaintext spaces for computing sums. However, when our construction is modified to support the product statistic, it can readily support large plaintext spaces. One interesting challenge, therefore, is to design a scheme that supports large plaintext spaces for computing sums. A promising direction is to try other algebraic primitives such as bilinear groups, Paillier groups, lattices, etc.

Richer statistics. An important goal in this research direction is the ability to support rich statistics. In general, we can imagine that the aggregator may wish to evaluate an arbitrary polynomial-time function f on the participants inputs. While recent advances on fully-homomorphic encryption schemes [8, 18] are encouraging, they do not directly apply in our setting, as their homomorphic operations are performed over ciphertexts encrypted under the same key. In general, it is interesting and challenging to consider expressive homomorphic encryption schemes over ciphertexts encrypted under multiple user keys.

Dynamic joins and leaves. In our scheme, whenever a participant dynamically joins or leaves the system, we need to perform the trusted setup phase again. This makes our scheme more suitable for scenarios where the set of participants is relatively static over time. An important open problem is to provide better support for dynamic joins and leaves, a capability that is valuable in systems with high churn.

Node failures. One or more participants may fail to upload their encrypted values in a certain time period. Malicious participants may also fail to respond in the form of a Denial-of-Service attack. When this happens, our scheme requires that the aggregator is unable to decrypt any partial information about the remaining participants. This requirement is inherent in our security definition – had we allowed the aggregator to decrypt partial information about a subset of the users, then even when all participants are functioning, a malicious aggregator is free to use inputs from only a subset of the participants. We acknowledge that this mode of failure may not be desirable in some practical settings. Therefore, one open problem is how to support graceful degradation in the face of failures. This question is challenging to answer, as it remains unclear what security notion one might employ to deal with node failures. The security definition must somehow reconcile two conflicting goals, the requirement of graceful degradation in the presence of failures, and the requirement that the aggregator should not learn anything from a subset of the inputs.

Acknowledgments

We gratefully thank the anonymous reviewers for insightful comments and feedback.

Dawn Song is partially supported by the National Science Foundation under Grants No. 0716230, 0448452 and CCF-0424422, and by the Office of Naval Research under MURI Grant No. N000140911081. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, or the Office of Naval Research.

References

[1] D. J. Bernstein and T. L. (editors). eBACS: ECRYPT benchmarking of cryptographic systems. `http://bench.cr.yp.to`, accessed 7 March 2011.

[2] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, 2005.

[3] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(3):1–36, 2009.

[4] A. Cavoukian, J. Polonetsky, and C. Wolf. SmartPrivacy for the smart grid: embedding privacy into the design of electricity conservation. *Identity in the Information Society*, 3(2):275–294, August 2010.

[5] C. Dwork. Differential privacy. Invited talk at *ICALP*, 2006.

[6] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, 2006.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[8] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[9] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, 2009.

[10] O. Goldreich. Secure multi-party computation. `http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps`.

[11] E. Magkos, M. Maragoudakis, V. Chrissikopoulos, and S. Gritzalis. Accurate and large-scale privacy-preserving data mining using the election paradigm. *Data & Knowledge Engineering*, 2009.

[12] J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. Handbook of applied cryptography. CRC Press, 1997.

[13] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *CRYPTO*, 2009.

[14] A. D. Molina, M. Salajegheh, and K. Fu. HICCUPS: health information collaborative collection using privacy and security. In *SPIMACS'09*, pages 21–30, 2009.

[15] E. L. Quinn. Privacy and the new energy infrastructure. *SSRN*, Feb 2009.

[16] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD 2010*, pages 735–746, 2010.

[17] E. G. Rieffel, J. Biehl, W. van Melle, and A. J. Lee. Secured histories: computing group statistics on encrypted data while preserving individual privacy. In submission, 2010.

[18] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, 2010.

[19] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *In SIAM SDM*, 2005.

A Proof of Aggregator Oblivious Security

First, prove that the following intermediate game is difficult to win, given that Decisional Diffie-Hellman is hard. Let $\mathbb{G}$ be a group of prime order p.

Setup. The challenger picks random generators $g, h \in \mathbb{G}$, and random $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$ such that $\sum_i \alpha_i = 0$. The challenger gives the adversary $g, h, g^{\alpha_0}, g^{\alpha_2}, \ldots, g^{\alpha_n}$.

Queries. The adversary can make "compromise" queries adaptively and ask for the value of α_i. The challenger returns α_i to the adversary when asked.

Challenge. The adversary speficies an uncompromised set $U \subseteq \{0, \ldots, n\}$. The challenger flips a random coin b. If $b = 0$, the challenger returns to the adversary $\{h^{\alpha_i} | i \in U\}$. If $b = 1$, the challenger picks $|U|$ random elements $h'_1, \ldots, h'_{|U|}$ from the group $\mathbb{G}$, such that

$$\prod_{i \in U} h'_i = \prod_{i \in U} h^{\alpha_i} \tag{3}$$

The challenger returns $h'_1, \ldots, h'_{|U|}$ to the adversary.

More Queries. The adversary can make more "compromise" queries, as in the previous query stage.

Guess. The adversary guesses either $b = 0$ or $b = 1$.

The adversary wins the game if she has not asked for any α_i for $i \in U$, and if she successfully guesses b. We also require that $|U| \geq 2$, since otherwise, the distributions of the outputs of the challenger when $b = 0$ and $b = 1$ are trivially indistinguishable.

Lemma 2. *The above game is difficult for computationally bounded adversaries assuming Decisional Diffie-Hellman is hard for group* $\mathbb{G}$.

Proof. By the hybrid argument. Define the following sequence of hybrid games. Assume that the set U specified by the adversary in the challenge stage is $U = \{i_1, i_2, \ldots, i_m\}$. For simplicity, we write $(\beta_1, \ldots, \beta_m) := (\alpha_{i_1}, \ldots, \alpha_{i_m})$. In Game_d, the challenger reveals the following to the adversary:

$$R_1, R_2, \ldots, R_d, h^{\beta_{d+1}}, \ldots, h^{\beta_m}$$

Here, each $R_i (i \in [d])$ means an independent fresh random number, and the following condition holds:

$$\prod_{1 \leq i \leq d} R_i = \prod_{1 \leq i \leq d} h^{\beta_i}$$

It is not hard to see that Game_1 is equivalent to the case when $b = 0$. Moreover, Game_{m-1} is equivalent to the case when $b = 1$.

Due to the hybrid argument, it suffices to show that adjacent games Game_{d-1} and Game_d are computationally indistinguishable. To demonstrate this, we show that if, for some d, there exists a polynomial-time adversary $\mathcal{A}$ who can distinguish between Game_{d-1} and Game_d, we can then construct an algorithm $\mathcal{B}$ which can solve the DDH problem.

Suppose $\mathcal{B}$ obtains a DDH tuple (g, g^x, g^y, T). $\mathcal{B}$'s task is to decide whether $T = g^{xy}$ or whether T is a random element from $\mathbb{G}$. Now $\mathcal{B}$ randomly guesses two indices j and k to be the d^{th} and the $(d+1)^{\text{th}}$ values of the set U specified by the adversary in the challenge phase. The guess is correct with probability $\frac{1}{n^2}$, and in case the guess turns out to be wrong later, the algorithm $\mathcal{B}$ simply aborts.

Now $\mathcal{B}$ picks random exponents $\{\alpha_i\}_{i \neq j, i \neq k}$. $\mathcal{B}$ implicitly sets $\alpha_k = x$ and $\alpha_j = -\sum_{i \neq j} \alpha_i$. Notice that $\mathcal{B}$ does not know the values of α_j and α_k, however, it knows or can compute the values of $g^{\alpha_k} = g^x$ and $g^{\alpha_j} = (\prod_{i \neq j} g^{\alpha_i})^{-1} = (g^x)^{-1} \cdot \prod_{i \neq j, i \neq k} g^{\alpha_i}$. $\mathcal{B}$ gives $\mathcal{A}$ the tuple $(g, h = g^y, g^{\alpha_1}, \ldots, g^{\alpha_n})$.

When $\mathcal{A}$ asks for any exponent except α_j and α_k, $\mathcal{B}$ simply returns the corresponding α_i value to $\mathcal{A}$. If $\mathcal{A}$ asks for α_j or α_k, the algorithm $\mathcal{B}$ aborts.

In the challenge phase, $\mathcal{A}$ submits a set $U = \{i_1, i_2, \ldots i_m\}$. If j and k are not the d^{th} and the $(d+1)^{\text{th}}$ values of the set U, i.e., if $i_d \neq j$ or $i_{d+1} \neq k$, the algorithm $\mathcal{B}$ aborts.

If $i_d = j$ and $i_{d+1} = k$, the algorithm $\mathcal{B}$ returns the following tuple to $\mathcal{A}$.

$$R_1, R_2, \ldots, R_{d-1},$$
$$(\textstyle\prod_{i \notin \{i_1, \ldots, i_{d+1}\}} (g^y)^{\alpha_i} \cdot \prod_{i=1}^{d-1} R_i \cdot T)^{-1}, \quad T,$$
$$(g^y)^{\alpha_{i_{d+2}}}, \ldots, (g^y)^{\alpha_{i_m}}$$

It is not hard to see that if $T = g^{xy}$, then the above game is equivalent to Game_{d-1}. Otherwise, if $T \in_R \mathbb{G}$, then the above game is equivalent to Game_d. Therefore, if the adversary $\mathcal{A}$ has a non-negligible advantage in guessing whether it is playing game Game_{d-1} or Game_d with $\mathcal{B}$, then the algorithm $\mathcal{B}$ would be able to solve the DDH problem with non-negligible advantage as well. □

Proof of Theorem 1: First, we will make a small modification to the aggregator oblivious security game. In the **Encrypt** queries, if the adversary submits a request for some tuple (i, x, t^*) where t^* is the time step specified in the **Challenge** phase, the challenger treats this as a **Compromise** query, and simply returns the sk_i to the adversary. Given sk_i, the adversary can compute the requested ciphertext herself. Therefore, this modification actually gives more power to the adversary. From now on, we will assume that the adversary does not make any **Encrypt** queries for the time t^*.

We divide our security game into two cases. Let $K \subseteq [n]$ denote the set of compromised participants (not including the aggregator). Let $\overline{K} := [n] \backslash K$ denote the set of uncompromised participants.

- **Case 1.** $U \neq \overline{K}$ or the aggregator capability has not been compromised. In other words, either there exists an uncompromised participant or the aggregator capability has not beem compromised. In this case, it suffices to show that the adversary cannot distinguish between "real" or "random", that is, whether the challenger returns a faithful encryption of the plaintext submitted in the challenge stage, or a random tuple picked from the appropriate group.

- **Case 2.** $U = \overline{K}$ and the aggregator capability has been compromised. In this case, we show that the adversary cannot distinguish whether the challenger returns a faithful encryption of the plaintext submitted in the challenge stage, or a random tuple *with the same product*.

Given an adversary $\mathcal{A}$ who can break the PSA game with non-negligible probability, we construct an algorithm $\mathcal{B}$ who can solve the above intermediate problem with non-negligible probability.

Setup. $\mathcal{B}$ obtains from its challenger $\mathcal{C}$ the following tuple $g, h, g^{\alpha_0}, g^{\alpha_1}, \ldots, g^{\alpha_n}$. $\mathcal{B}$ implicitly sets α_0 to be the data aggregator's capability, and $\alpha_1, \ldots, \alpha_n$ to be the secret keys of participants 1 through n respectively. The public params is g.

The algorithm $\mathcal{B}$ makes a random guess as to whether Case 1 or Case 2 will happen, and if the guess turns out to be wrong, the simulator simply aborts. Moreover, if $\mathcal{B}$ guesses Case 1, then $\mathcal{B}$ will randomly guess a participant (or aggregator) $j^* \in (\overline{K} \backslash U) \cup \{0\}$ that remains uncompromised at the end of the game. If the guess turns out to be wrong later, $\mathcal{B}$ aborts.

Let q_H denote the total number of oracle queries made by the adversary $\mathcal{A}$ and by the algorithm $\mathcal{B}$ itself. $\mathcal{B}$ guesses at random an index $k \in [q_H]$. Suppose the input to the k^{th} random oracle query is t^*. The algorithm $\mathcal{B}$ assumes that t^* will be the challenge time step. If the guess turns out to be wrong later, $\mathcal{B}$ simply aborts.

Hash Function Simulation. The adversary submits a hash query for the integer t. $\mathcal{B}$ first checks the list $\mathcal{L}$ to see if t has appeared in any entry (t, z). If so, $\mathcal{B}$ returns g^z to the adversary. Otherwise, if this is not the k^{th} query, $\mathcal{B}$ picks a random exponent z and returns g^z to the adversary, and saves (t, z) to a list $\mathcal{L}$. For the k^{th} query, $\mathcal{B}$ returns h.

Queries.

- **Encrypt.** The adversary $\mathcal{A}$ submits an **Encrypt** query for the tuple (i, x, t). As mentioned above, in the modified version of the game, we ensure that $t \neq t^*$, since otherwise, we simply treat it as a **Compromise** query. $\mathcal{B}$ checks if a hash query has been made on t. If not, $\mathcal{B}$ makes a hash oracle query on t. As a result, $\mathcal{B}$ knows the discrete log of $H(t)$. Let $H(t) = g^z$, then $\mathcal{B}$ knows z. Since $\mathcal{B}$ also knows g^{α_i}, $\mathcal{B}$ can compute the ciphertext $g^x \cdot (g^z)^{\alpha_i}$ as $g^x \cdot (g^{\alpha_i})^z$.
- **Compromise.** $\mathcal{B}$ forwards $\mathcal{A}$'s query to its own challenger $\mathcal{C}$, and forwards the answer α_i to $\mathcal{A}$.

Challenge. The adversary $\mathcal{A}$ submits a set U and a time t^*, as well as plaintexts $\{x_i | i \in U\}$. (We consider the real-or-random version of the security game.) If t^* does not agree with the value submitted in the k^{th} hash query, then $\mathcal{B}$ aborts.

If $\mathcal{B}$ has guessed Case 1 at the beginning of the game, then it submits the set $U \cup \{j^*\}$ in a **Challenge** query to its own challenger $\mathcal{C}$. As a result, it obtains a tuple $\{T_i\}_{i \in U}, T_{j^*}$.

If $\mathcal{B}$ has guessed Case 2, then it simply submits the set U in a **Challenge** query to its own challenger. As a result, it obtains a tuple $\{T_i\}_{i \in U}$.

In both cases, the challenger returns the following ciphertexts to the adversary:

$$\forall i \in U : g^{x_i} \cdot T_i$$

More queries. Same as the **Query** stage.

Guess. If the adversary $\mathcal{A}$ guesses that $\mathcal{B}$ has returned a random tuple then $\mathcal{B}$ guesses $b' = 1$. Otherwise, $\mathcal{B}$ guesses that $b' = 0$.

- **Case 1.** If the challenger $\mathcal{C}$ returns to $\mathcal{B}$ a faithful Diffie-Hellman tuple $\forall i \in U : T_i = h^{\alpha_i}$, and $T_{j^*} = h^{\alpha_{j^*}}$, then the ciphertext returned to the adversary $\mathcal{A}$ is a faithful encryption of the plaintext submitted by the adversary. Otherwise, if the challenger returns to $\mathcal{B}$ a random tuple under the product constraint, then the ciphertext returned to $\mathcal{A}$ is a random tuple.
- **Case 2.** If the challenger $\mathcal{C}$ returns gives $\mathcal{B}$ a faithful Diffie-Hellman tuple $\forall i \in U : T_i = h^{\alpha_i}$, then the ciphertext returned to the adversary $\mathcal{A}$ is a faithful encryption of the plaintext submitted by the adversary. Otherwise, if the challenger returns to $\mathcal{B}$ a random tuple under the product constraint, then the ciphertext returned to $\mathcal{A}$ is a random tuple under the product constraint.

□

B Proof of Utility

Lemma 3 (Moment Generating Function)**.** *Suppose* $1 < \alpha \leq 22$. *For each* i, $E[\exp(hr_i)] \leq \exp(\frac{4\alpha\beta}{(\alpha-1)^2} \cdot h^2)$, *for* $|h| < \frac{\alpha-1}{2\sqrt{\alpha}}$.

Proof. Let $\alpha > 1$. Let G be a random variable having distribution $\mathsf{Geom}(\alpha)$ and W be a random variable having distribution $\mathsf{Geom}^+(\alpha)$. It follows that r_i and G can be sampled in the following way.

$$r_i := \begin{cases} 0 & \text{with probability } 1-\beta \\ G & \text{with probability } \beta \end{cases}$$

$$G := \begin{cases} 0 & \text{with probability } \frac{\alpha-1}{\alpha+1} \\ W & \text{with probability } \frac{1}{\alpha+1} \\ -W & \text{with probability } \frac{1}{\alpha+1} \end{cases}$$

By standard computation, for $h < \ln \alpha$, $E[\exp(hW)] = \frac{(\alpha-1)e^h}{\alpha - e^h}$.

Hence, for $|h| < \ln \alpha$,

$$\begin{aligned} & E[\exp(hG)] \\ = \; & \frac{\alpha-1}{\alpha+1} \cdot e^0 + \frac{1}{\alpha+1} \cdot E[\exp(hW)] + \frac{1}{\alpha+1} \cdot E[\exp(-hW)] \\ = \; & \frac{\alpha-1}{\alpha+1} + \frac{1}{\alpha+1} \cdot \frac{(\alpha-1)e^h}{\alpha - e^h} + \frac{1}{\alpha+1} \cdot \frac{(\alpha-1)e^{-h}}{\alpha - e^{-h}} \\ = \; & \frac{(\alpha-1)^2}{\alpha^2 + 1 - \alpha(e^h + e^{-h})} \leq \frac{(\alpha-1)^2}{(\alpha-1)^2 - 2\alpha h^2}, \end{aligned}$$

where in the last inequality, we assume $|h| \leq 2.5$ so that $e^h + e^{-h} \leq 2 + 2h^2$, and also $|h| < \frac{\alpha-1}{\sqrt{2\alpha}}$ so that the denominator remains positive. We next further assume $|h| \leq \frac{\alpha-1}{2\sqrt{\alpha}}$ and use the inequality $(1-u)^{-1} \leq 1 + 2u$ for $0 \leq u \leq \frac{1}{2}$ to obtain the following bound.

For $|h| \leq \min\{\ln \alpha, \frac{\alpha-1}{2\sqrt{\alpha}}, 2.5\} = \frac{\alpha-1}{2\sqrt{\alpha}}$ (for $1 < \alpha \leq 22$), $E[\exp(hG)] \leq 1 + \frac{4\alpha}{(\alpha-1)^2} \cdot h^2$.

Hence, for the same bound on h, we have

$$\begin{aligned} E[\exp(hr_i)] &= (1-\beta) \cdot e^0 + \beta E[\exp(hG)] \\ &\leq 1 - \beta + \beta(1 + \frac{4\alpha}{(\alpha-1)^2} \cdot h^2) \\ &\leq \exp(\frac{4\alpha\beta}{(\alpha-1)^2} \cdot h^2), \end{aligned}$$

where in the last inequality, we use $1 + u \leq \exp(u)$, for all reals u. □

Proof of Theorem 3: Let $\alpha := \exp(\frac{\epsilon}{\Delta})$. From assumption, $\frac{\epsilon}{\Delta} \leq 3$ and hence we have $1 < \alpha \leq 22$.

Let $Z := \sum_{i=1}^{n} r_i$ and $\lambda := 4\sqrt{\log \frac{2}{\eta}} \cdot \sqrt{n\beta} \cdot \frac{\sqrt{\alpha}}{\alpha-1}$.

For positive h satisfying the bound in Lemma 3, we have

$$\begin{aligned} & Pr[Z \geq \lambda] \\ \leq \; & \exp(-\lambda h) E[\exp(hZ)] \\ \leq \; & \exp(-\lambda h + \frac{4\alpha\beta n}{(\alpha-1)^2} \cdot h^2) \end{aligned}$$

The first inequality comes from a standard application of Markov inequality. The second inequality comes from the independence of the r_i's and Lemma 3. Set $h := \frac{\lambda(\alpha-1)^2}{8\alpha\beta n} = \frac{\sqrt{\log \frac{2}{\eta}}}{\sqrt{n\beta}} \cdot \frac{\alpha-1}{2\sqrt{\alpha}} \leq \frac{\alpha-1}{2\sqrt{\alpha}}$, where the last inequality follows from the choice of η.

Hence, it follows that $Pr[Z \geq \lambda] \leq \exp(-\frac{1}{2}\lambda h) = \frac{\eta}{2}$. Since Z is symmetric, $Pr[|Z| \geq \lambda] \leq \eta$.

Hence, it follows that with probability at least $1 - \eta$, $|Z| \leq \lambda = 4\sqrt{\frac{1}{\gamma} \log \frac{1}{\delta} \log \frac{2}{\eta}} \cdot \frac{\sqrt{\alpha}}{\alpha-1}$.

Finally, the result follows by observing that for all $\alpha > 1$, $\frac{\sqrt{\alpha}}{\alpha-1} \leq \frac{1}{\ln \alpha} = \frac{\Delta}{\epsilon}$. □