Privacy-Preserving Stream Aggregation with Fault Tolerance

T-H. Hubert Chan[1], Elaine Shi[2], and Dawn Song[2]

[1] The University of Hong Kong
[2] UC Berkeley

Abstract. We consider applications where an *untrusted aggregator* would like to collect privacy sensitive data from users, and compute aggregate statistics periodically. For example, imagine a smart grid operator who wishes to aggregate the total power consumption of a neighborhood every ten minutes; or a market researcher who wishes to track the fraction of population watching ESPN on an hourly basis.

We design novel mechanisms that allow an aggregator to accurately estimate such statistics, while offering provable guarantees of user privacy against the untrusted aggregator. Our constructions are resilient to user failure and compromise, and can efficiently support dynamic joins and leaves. Our constructions also exemplify the clear advantage of combining applied cryptography and differential privacy techniques.

1 Introduction

Many real-world applications have benefitted tremendously from the ability to collect and mine data coming from multiple individuals and organizations. These applications have also spurred numerous concerns over the privacy of user data. In this paper, we study how an *untrusted aggregator* can gather information and learn aggregate statistics from a population without harming individual privacy. For example, consider a smart grid operator who wishes to track the total electricity consumption of a neighborhood every 15 minutes, for scheduling and optimization purposes. Since such power consumption data can reveal sensitive information about individual's presence and activities, we wish to perform such aggregation in a privacy-preserving manner.

More generally, we consider the *periodic distributed stream aggregation* model. Imagine a group of n users. In every time period, each user has some data point within a certain range $(-\Delta, +\Delta)$. An untrusted aggregator wishes to compute the sum of all users' values in each time period. Each user considers her data as sensitive, and does not want to reveal the bit to the untrusted aggregator. How can we allow an untrusted aggregator to periodically learn aggregate information about a group of users, while preserving each individual's privacy?

The problem of privacy-preserving stream aggregation was first studied by Rastogi *et al.* [13] and Shi *et al.* [14]. These two works demonstrate how to combine cryptog-

raphy with differential privacy and achieve $O(1)$ error, while using differential privacy techniques alone would result in at least $\Omega(\sqrt{N})$ error [3] in this setting[3].

Specifically, these two works [13, 14] both employ special encryption schemes which work as follows: in each aggregation period, each user encrypts its (perturbed) data value and sends the encrypted value to the aggregator. The aggregator has a cryptographic capability allowing it to decrypt the sum of all users' values, but learn nothing else. In constructing such encryptions schemes, both works [13, 14] rely on the following key idea: each user would incorporate a random value into their ciphertext; and the aggregator's capability also incorporates a random value. All of these random values sum up to 0, and would cancel out in the decryption step, such that the aggregator can recover the sum of all users' values, but learn nothing else.

One major drawback of these earlier works [13, 14] is that these schemes are not tolerant of user failures. Even if a single user fails to respond in a certain aggregation round, the server would not be able to learn anything. This can be a big concern in real-world applications where failures may be unavoidable. For example, in a smart sensing applications, where data is collected from multiple distributed sensors, it is quite likely that some sensor might be malfunctioning at some point, and fails to respond. Failure tolerance is an important challenge left open by Shi *et al.* [14] and Rastogi *et al.* [13].

Summary of contributions. Our main contribution is to introduce a novel technique to achieve *fault tolerance*, while incurring only a very small (logarithmic or poly-logarithmic) penalty in terms of communication overhead and estimation error (see Table 1). In our construction, the aggregator is still able to estimate the sum over the remaining users when an arbitrary subset of users (unknown in advance) fail.

As a by-product of the fault tolerance technique, our scheme also supports *dynamic joins and leaves*, which is another problem left open by previous work [13, 14]. Specifically, our scheme supports dynamic joins and leaves without having to perform costly rekeying operations with every join and leave.

Apart from failure tolerance and dynamic joins/leaves, our scheme has another desirable feature in that it requires only *a single round of client-to-server communication*. On a very high level, our construction works as follows: in every time period, each user uploads an encrypted and perturbed version of her data, and then the aggregator can compute the noisy sum by using a cryptographic capability obtained during an initial one-time setup phase.

Techniques. Our main technique for achieving failure tolerance may be of independent interest. Specifically, we build a *binary interval tree* over n users, and allow the aggregator to estimate the sum of contiguous intervals of users as represented by nodes in the interval tree. In comparison with Shi et al. [14], the binary-tree technique allows us to handle user failures, joins and leaves, with a small logarithmic (or polylog) cost in terms of communication and estimation error.

More applications. Apart from the smart grid example mentioned earlier, the distributed stream aggregation problem is also widely applicable in a variety of problem

[3] The lower bound holds when the aggregator sees all the messages in the protocol, for example, in the case where each user communicates only with the aggregator.

Scheme	Total comm.	Avg comm. per user	Error	Fail-safe/Dynamic joins & leaves	Security Model	Comm. model
Naive	$O(n)$	$O(1)$	$O(\sqrt{n})$	Yes	DP	C $\rightarrow$ S
Rastogi *et al.* [13]	$O(n)$	$O(1)$	$O(1)$	No	CDP AO	C $\Leftrightarrow$ S
Shi *et al.* [14]	$O(n)$	$O(1)$	$O(1)$	No	CDP AO	C $\rightarrow$ S
This paper:						
Sampling (Online TR)	$O(\frac{1}{\rho^2})$	$O(\frac{1}{\rho^2 n})$	$O(\rho n)$	Yes	DP	C $\Leftrightarrow$ S
Binary	$O(n \log n)$	$O(\log n)$	$\tilde{O}((\log n)^{\frac{3}{2}})$	Yes	CDP	C$\rightarrow$ S

DP: Differential Privacy CDP: Computational Differential Privacy AO: Aggregator Obliviousness (explanations in Section 1.1) $C \rightarrow S$: client-to-server uni-directional $C \Leftrightarrow S$: interactive between client and server

Table 1. Comparison between existing schemes and our contributions. The asympototic bounds hide the privacy parameters ϵ and δ. The parameter ρ denotes any constant between 0 and 1. The $\tilde{O}(\cdot)$ notation hides a $\log \log n$ factor.
In our full online technical report [2], we also propose two variants of sampling-based constructions, in which a random subset of users respond by sending a perturbed version of their data. The sampling constructions can be useful in applications where bandwidth efficiency is a major concern. In particular, for arbitrarily small ρ between 0 and 1, we can achieve error $O(\rho n)$ with $O(\frac{1}{\rho^2})$ words of total communication.

domains, such as distributed hot item identification, sensing and monitoring, as well as medical research. We elaborate more on these applications in the online full version [2].

1.1 Related Work

Differential privacy [1, 5, 6, 8] was traditionally studied in a setting where a trusted curator, with access to the entire database in the clear, wishes to release statistics in a way that preserves each individual's privacy. The trusted curator is responsible for introducing appropriate perturbations prior to releasing any statistic. This setting is particularly useful when a company or a government agency, in the possession of a dataset, would like to share it with the public.

In real-world applications, however, users may not trust the aggregator. A recent survey by Microsoft [15] found that "...58% of the public and 86% of business leaders are excited about the possibilities of cloud computing. But, more than 90% of them are worried about security, availability, and privacy of their data as it rests in the cloud."

Recently, the research community has started to consider how to guarantee differential privacy in the presence of an untrusted aggregator [13, 14]. Rastogi *et al.* [13] and Shi *et al.* [14] proposed novel algorithms that allow an untrusted aggregator to periodically estimate the sum of n users' values, without harming each individual's privacy. In addition to (computational) differential privacy, these two schemes also provide *ag-*

gregator obliviousness, meaning that the aggregator only learns the noisy sum, but no intermediate results.

Both of these schemes [13, 14], however, would suffer in the face of user failures, thereby leaving resilience to node failure as one of the most important open challenges in this area. Our Binary Protocol utilizes Shi *et al.*'s encryption scheme as a building block, and we successfully solve the node failure problem.

Dwork *et al.* [7] study distributed noise generation, however, their scheme requires interactions among all users.

The use of a binary tree in our construction may be reminiscent of Dwork *et al.* [9] and Chan *et al.* [4], where they use a binary-tree-like construction for a completely different purpose, i.e., to achieve high utility when releasing statistics continually in a trusted aggregator setting.

2 Problem Definition and Assumptions

For simplicity, consider a group of n users each holding a private bit $x_i \in \{0,1\}$ – although our approach can be trivially adapted to the case where each user has a data point within a certain discrete range. We use the notation $\mathbf{x} := (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ to denote the vector of all users' bits, also referred to as an *input configuration*. An aggregator $\mathcal{A}$ wishes to estimate the count, denoted $\mathsf{sum}(\mathbf{x}) := \sum_{i\in[n]} x_i$.

Periodic aggregation. We are particularly interested in the case of periodic aggregation. For example, a market researcher may wish to track the fraction of the population watching ESPN during different hours of the day. In general, in each time period $t \in \mathbb{N}$, we have a vector $\mathbf{x}^{(t)} \in \{0,1\}^n$, e.g., indicating whether each of the surveyed users is currently watching ESPN. The aggregator wishes to evaluate $\mathsf{sum}(\mathbf{x}^{(t)}) := \sum_{i\in[n]} x_i^{(t)}$ in every time period. For ease of exposition, we often focus our attention on the aggregation algorithm in one time step, and as a result, omit the superscript t.

Failure tolerance. When a user fails, it stops participating in the protocol. A protocol is *failure tolerant*, if for any subset of failed users, the aggregator can still make an estimate on the sum of the bits from the remaining functioning users.

Communication model. In real-world applications, peer-to-peer communication is undesirable as it requires all users to be online simultaneously and interact with each other. This paper will focus on schemes that requires no user-to-user communication, i.e., all communication takes place between an aggregator and a user.

2.1 Assumptions and Privacy Definitions

Trust Model. We consider the scenario when the aggregator is untrusted. We think of the aggregator as the adversary from whom we wish to protect the users' privacy. The aggregator does not have access to the users' bits directly, but may have arbitrary auxiliary information a priori. Such auxiliary information can be harvested in a variety of ways, e.g., from public datasets online, or through personal knowledge about a user. Our constructions ensure individual privacy even when the aggregator may have arbitrary auxiliary information.

Compromise. We assume a semi-honest model, where *compromised* users can collude with the aggregator by revealing their input bits or random noises to the aggregator. However, we assume that all users honestly use their inputs in the aggregation. The *data pollution* attack, where users inflate or deflate their input values, is out of the scope of this paper, and can be solved using orthogonal techniques such as [12]. In this paper, we assume a slightly relaxed model of compromise, where the compromised nodes are chosen independently from the randomness used in the algorithm (more details in Section 4).

Key distribution. We assume that any cryptographic keys or privacy parameters required are already distributed to the aggregator and users in a separate setup phase ahead of time. The setup phase needs to be performed only once at system initialization, and need not be repeated during the periodic aggregation rounds.

We define a transcript π to be the sequence of all messages sent by the users and the aggregator at the end of the protocol. As we consider protocols with no peer-to-peer communication, i.e., all communication takes place between the users and the aggregator, the view of the aggregator during the protocol is essentially the transcript π.

Users (and the aggregator) may contribute randomness to the protocol, for example, users will add noise to perturb their input bits. Therefore, we can define a distribution on the transcripts. Formally, we use the notation Π to denote a randomized protocol, and use $\Pi(\mathbf{x})$ to denote the random transcript when the input configuration is $\mathbf{x}$.

In this paper, we consider the computational version of differential privacy, as in practice it suffices to secure the protocol against computationally-bounded adversaries. We now define computational differential privacy (CDP), similar to the CDP notion originally proposed by Mironov *et al.* [11].

In addition to the users' data $\mathbf{x}$, the protocol Π also takes a security parameter $\lambda \in \mathbb{N}$. We use the notation $\Pi(\lambda, \mathbf{x})$ to denote the distribution of the transcript when the security parameter is λ and the input configuration is $\mathbf{x}$.

Definition 1 (Computational Differential Privacy Against Compromise). *Suppose the users are compromised by some underlying randomized process $\mathcal{C}$, and we use C to denote the information obtained by the adversary from the compromised users. Let $\epsilon, \delta > 0$. A (randomized) protocol Π preserves computational (ϵ, δ)-differential privacy (against the compromise process $\mathcal{C}$) if there exists a negligible function $\eta : \mathbb{N} \to \mathbb{R}^+$ such that for all $\lambda \in \mathbb{N}$, for all $i \in [n]$, for all vectors $\mathbf{x}$ and $\mathbf{y}$ in $\{0,1\}^n$ that differ only at position i, for all probabilistic polynomial-time Turing machines $\mathcal{A}$, for any output $b \in \{0,1\}$,*

$$\Pr_{\mathcal{C}_i}[\mathcal{A}(\Pi(\lambda, \mathbf{x}), C) = b] \leq e^{\epsilon} \cdot \Pr_{\mathcal{C}_i}[\mathcal{A}(\Pi(\lambda, \mathbf{y}), C) = b] + \delta + \eta(\lambda),$$

where the probability is taken over the randomness of $\mathcal{A}$, Π and $\mathcal{C}_i$, which denotes the underlying compromise process conditioning on the event that user i is uncompromised.

A protocol Π preserves computational ϵ-differential privacy if it preserves computational $(\epsilon, 0)$-differential privacy.

3 Preliminaries

3.1 Tool: Geometric Distribution

Two noise distributions are commonly used to perturb the data and ensure differential privacy, the Laplace distribution [8], and the Geometric distribution [10]. The advantage of using the geometric distribution over the Laplace distribution is that we can keep working in the domain of integers. The geometric distribution is particularly useful when used in combination with a crypto-system, e.g., our Binary Protocol described in Section 4. Most crypto-systems work in discrete mathematical structures, and are not designed to work with (truly) real numbers.

We now define the *symmetric* geometric distribution.

Definition 2 (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 $\frac{\alpha-1}{\alpha+1} \cdot \alpha^{-|k|}$.*

The following property of Geom distribution is useful for designing differentially private mechanisms that output integer values.

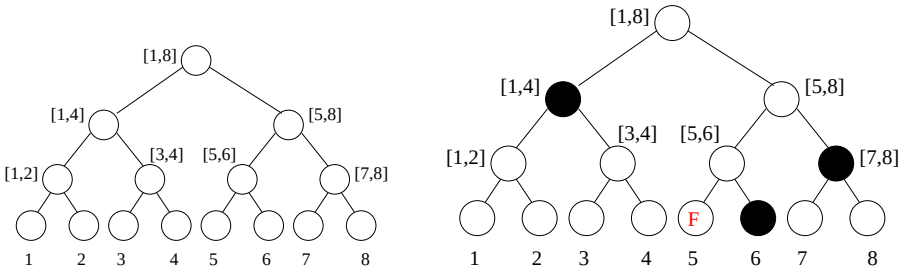
Fact 1 *Let $\epsilon > 0$. Suppose u and v are two integers such that $|u - v| \leq \Delta$. Let r be a random variable having distribution $\mathsf{Geom}(\exp(\frac{\epsilon}{\Delta}))$. Then, for any integer k, $Pr[u + r = k] \leq \exp(\epsilon) \cdot Pr[v + r = k]$.*

In our setting, changing 1 bit can only affect the sum by at most 1. Hence, it suffices to consider $\mathsf{Geom}(\alpha)$ with $\alpha = e^{\epsilon}$. Observe that $\mathsf{Geom}(\alpha)$ has variance $\frac{2\alpha}{(\alpha-1)^2}$. Since $\frac{\sqrt{\alpha}}{\alpha-1} \leq \frac{1}{\ln \alpha} = \frac{1}{\epsilon}$, the magnitude of the error added is $O(\frac{1}{\epsilon})$. The following diluted geometric distributions is useful in the description of our protocols.

Definition 3 (Diluted Geometric Distribution). *Let $0 < \beta \leq 1$, $\alpha > 1$. A random variable has β-diluted Geometric distribution $\mathsf{Geom}^{\beta}(\alpha)$ if with probability β it is sampled from $\mathsf{Geom}(\alpha)$, and with probability $1 - \beta$ is set to 0.*

3.2 Naive Scheme

We describe a simple scheme as a warm-up, and as a baseline of comparison. In the Naive Scheme, each user generates an independent $\mathsf{Geom}(e^{\epsilon})$ noise, which is added to her bit. Each user sends her perturbed bit to the aggregator, who computes the sum of all the noisy bits. As each user adds one copy of independent noise to her data, n copies of noises would accumulate in the sum. As some positive and negative noises may cancel out, the accumulated noise is $O(\frac{\sqrt{n}}{\epsilon})$ with high probability. Notice that if we employs the information-theoretic (as opposed to computational) differential privacy notion, the naive scheme is in some sense the best one can do. Chan *et al.* [3] show in a recent work that in a setting with n users and one aggregator, any (information theoretically) differential private summation protocol with no peer-to-peer interaction must result in an error of $\Omega(\sqrt{N})$.



(a) The aggregator obtains block estimates corresponding to all nodes appearing in the binary interval tree.

(b) When user 5 fails, the aggregator sums up the block estimates corresponding to the black nodes.

Fig. 1. Intuition for the Binary Protocol.

4 Binary Protocol: Achieving Failure Tolerance

4.1 Intuition

Consider the periodic aggregation scheme proposed by Shi *et al.* [14], henceforth referred to as the Block Aggregation (BA) scheme. In the BA scheme, every time period, each user sends a perturbed and encrypted version of her data to the aggregator. The aggregator has a cryptographic capability to decrypt the sum of all encrypted values, but can learn nothing else. The BA scheme achieves $O(1)$ error. and guarantees all users' differential privacy against polynomial-time adversaries.

Unfortunately, the use of cryptography in the BA scheme introduces the all-or-nothing decryption model. Therefore, the aggregator learns nothing if a single user fails.

The challenge. On one hand, we have the naive scheme which achieves $O(\sqrt{n})$ error, and is failure tolerant On the other hand, we have the BA Scheme which achieves $O(1)$ error (by combining cryptography with differential privacy), but is unfortunately not failure tolerant. Can we seek middle-ground between these approaches, such that we can obtain the best of both worlds, i.e., achieve both fault tolerance and small error?

Binary tree idea. One idea is to form user groups (henceforth referred to as *blocks*), and run the BA Scheme for each block. The aggregator is then able to estimate the sum for each block. If a subset of the users fail, we must be able to find a set of disjoint blocks to cover the functioning users. In this way, the aggregator can estimate the sum of the functioning users. The challenge is how to achieve this with only a small number of groups.

As depicted in Figure 1, our construction is based on a binary interval tree, hence the name Binary Protocol. For ease of exposition, assume for now that n is a power of 2. Each leaf node is tagged with a number in $[n]$. Each internal node in the tree represents a contiguous interval covering all leaf nodes in its subtree. As a special case, we can think of the leaf nodes as representing intervals of size 1. For each node in the tree, we also use the term *block* to refer to the contiguous interval represented by the node.

Intuitively, the aggregator and users would simultaneously perform the BA Scheme for every interval (or block) appearing in the binary tree. Hence, the aggregator would

obtain an estimated sum for each of these blocks. Normally, when n is a power of 2, the aggregator could simply output the block estimate for the entire range $[1, n]$. However, imagine if a user i fails to respond, the aggregator would then fail to obtain block estimates for any block containing i, including the block estimate for the entire range $[1, n]$.

Fortunately, observe that any contiguous interval within $[n]$ can be covered by $O(\log n)$ nodes in the binary interval tree. If κ users have failed, the numbers 1 through n would be divided into $\kappa + 1$ contiguous intervals, each of which can be covered by $O(\log n)$ nodes. This means that the aggregator can estimate the sum of the remaining users by summing up $O((\kappa + 1) \log n)$ block estimates.

Example. For convenience, we use the notation $\mathsf{sum}[i..j]$ (where $1 \leq i \leq j \leq n$) to denote the estimated sum for the block $x_i, x_{i+1}, \ldots, x_j$ of user inputs. Figure 1 depicts a binary tree of size $n = 8$. When all users are active, the aggregator can obtain block estimates corresponding to all nodes in the tree. Therefore, the aggregator can simply output block estimate $\mathsf{sum}[1..8]$. Figure 1 illustrates the case when user 5 has failed. When this happens, the aggregator fails to obtain the block estimates $\mathsf{sum}[5..5]$, $\mathsf{sum}[5..6]$, $\mathsf{sum}[5..8]$, and $\mathsf{sum}[1..8]$, since these blocks contain user 5. However, the aggregator can still estimate the sum of the remaining users by summing up the block estimates corresponding to the black nodes in the tree, namely, $\mathsf{sum}[1..4]$, $\mathsf{sum}[6..6]$, and $\mathsf{sum}[7..8]$.

Privacy-utility tradeoff. We now give a quick and informal analysis of the privacy-utility tradeoff. It is not hard to see that each user is contained in at most $O(\log n)$ blocks. This means that if a user's bit is flipped, $O(\log n)$ blocks would be influenced. Roughly speaking, to satisfy ϵ-differential privacy, it suffices to add noise proportional to $O(\frac{\log n}{\epsilon})$ to each block.

If κ users fail, we would be left with $\kappa + 1$ intervals. Each interval can be covered by $O(\log n)$ nodes in the binary tree. Therefore, the final estimate would consist of $O((\kappa + 1) \log n)$ block estimates. Since each block estimate contains $O(\frac{\log n}{\epsilon})$ noise, the final estimate would contain $O((\kappa + 1) \log n)$ copies of such noises. As some positive and negative noises cancel out, the final estimate would contain noise of roughly $O(\frac{(\log n)^{1.5}\sqrt{\kappa+1}}{\epsilon})$ magnitude.

In the remainder of the section, we first give a formal description of the BA Scheme [14] used as a building block of the Binary Protocol. Then we formally describe the Binary Protocol and state the theorems on the privacy and utility tradeoff.

4.2 Background: Basic Block Construction

We will use the BA Scheme [14] as a building block to aggregate the sum for each block (or subset) $B \subseteq [n]$ of users. We now explain at a high level how the BA scheme works. Note that in place of the BA scheme by Shi *et al.* [14], the binary tree framework also readily applies on top of the scheme by Rastogi *et al.* [13]. The tradeoffs are discussed later in Section 5.

Encryption scheme. The BA Scheme leverages an encryption scheme that *allows an aggregator to decrypt the sum of all users' encrypted values (with an appropriate cryp-*

tographic capability), but learn nothing more. The encryption scheme has three (possibly randomized) algorithms.

- $\mathsf{Setup}(m, \lambda)$: A one-time setup algorithm, run by a trusted dealer, takes the number of users m, and a security parameter λ as inputs. It outputs the following:

$$(\mathsf{params}, \mathsf{cap}, \{\mathsf{sk}_i\}_{i\in[m]}),$$

  where params are system parameters, e.g., a description of the selected algebraic group. Capability cap is distributed to the aggregator, and sk_i ($i \in [m]$) is a secret key distributed to user i. The users will later use their secret keys to encrypt, and the aggregator will use its capability to decrypt the sum, in each aggregation period. The setup algorithm is performed *only once at system initialization*, and need not be repeated for each periodic aggregation round.
- $\mathsf{Encrypt}(\mathsf{sk}_i, x_i, t)$: During time step t, user i uses sk_i to encrypt its (possibly perturbed) data x_i. The user uploads the outcome ciphertext c_i to the aggregator.
- $\mathsf{Decrypt}(\mathsf{cap}, \{c_i\}_{i\in[m]}, t)$: During time step t, after the aggregator collects all users' ciphertexts $\{c_i\}_{i\in[m]}$, it calls the decryption algorithm $\mathsf{Decrypt}$ to retrieve the sum $\sum_{i\in[m]} x_i$. Apart from this sum, the aggregator is unable to learn anything else.

The BA scheme relies on the following key idea. In the Setup phase, each user i ($1 \le i \le m$) obtains a secret-key which incorporates a random value r_i. The aggregator obtains a capability which incorporates a random value r. Furthermore, the condition $r + \sum_{i=1}^{m} r_i = 0$ is satisfied. In every aggregation period, each user incorporates its random value r_i into its ciphertext. After collecting all ciphertexts from users, the aggregator can homomorphically "sum up" all ciphertexts, such that the random values r, $r_1, \ldots, r_m$ cancel out, and the aggregator can thus decrypt the sum of all users' encrypted values. The above is a grossly simplied view of the BA scheme intended to capture the intuition. The full construction is more sophisticated, and requires additional techniques to allow the random values distributed in the Setup phase to be reusable in multiple aggregation phases, while still maintaining security.

Input perturbation. Revealing the exact sum to the aggregator can still harm an individual's differential privacy. To guarantee differential privacy, each user adds some noise to her data before encrypting it.

Recall that in the naive scheme, each user must add one copy of geometric noise to guarantee its own differential privacy. In the BA Scheme, however, the aggregator can only decrypt the sum, and cannot learn each individual's perturbed values. Therefore, as long as the all users' noises sum up to roughly one copy of geometric noise, each user's differential privacy can be guaranteed. This is why the BA Scheme construction can guarantee $O(1)$ error.

Let ϵ, δ denote the privacy parameters. In every time step t, each user i generates an independent r_i from the diluted geometric distribution $\mathsf{Geom}^{\beta}(\alpha)$ and computes $\widehat{x}_i := x_i + r_i$. In other words, with probability β, the noise r_i is generated from the geometric distribution $\mathsf{Geom}(\alpha)$, and with probability $1-\beta$, r_i is set to 0. Specifically, we choose $\alpha := e^{\epsilon}$, and $\beta := \min\{\frac{1}{m} \ln \frac{1}{\delta}, 1\}$. This ensures that with high probability, at least one user has added $\mathsf{Geom}(e^{\epsilon})$ noise. More generally, if $1-\gamma$ fraction of the users are compromised, then we set $\beta := \min(\frac{1}{\gamma m} \ln \frac{1}{\delta}, 1)$.

SETUP$(n, \lambda, \epsilon, \delta)$: *# run by a trusted dealer*

$K \leftarrow \lfloor \log_2 n \rfloor + 1$
$\epsilon_0 \leftarrow \frac{\epsilon}{K}, \delta_0 \leftarrow \frac{\delta}{K}$
Give ϵ_0 and δ_0 to all users.
for $B \in \mathcal{T}(n)$ **do**
  $(\mathsf{params}, \mathsf{cap}_B, \{\mathsf{sk}_{i,B}\}_{i \in B}) \leftarrow \mathsf{BA.Setup}(|B|, \lambda)$
  Give params to aggregator and all users.
  Give cap_B to the aggregator.
  Give $\mathsf{sk}_{i,B}$ and $|B|$ to each user $i \in B$.
end for

ALGUSER$\big(x_i, t, \mathcal{B}(i), \{\mathsf{sk}_{i,B}\}_{i \in \mathcal{B}(i)}, \epsilon_0, \delta_0\big)$:

 # Periodic aggregation – user i's algorithm

for $B \in \mathcal{B}(i)$ **do**
  $\beta \leftarrow \min(\frac{1}{|B|} \ln \frac{1}{\delta_0}, 1)$
  $r \leftarrow \mathsf{Geom}^{\beta}(e^{\epsilon_0})$
  $\widehat{x}_{i,B} \leftarrow x_i + r$
  $c_{i,B} \leftarrow \mathsf{BA.Encrypt}(\mathsf{sk}_{i,B}, \widehat{x}_{i,B}, t)$
  Send $c_{i,B}$ to aggregator.
end for

ALGAGGR$\big(S, \{\mathsf{cap}_B\}_{B \in \mathcal{T}(n)}, \{c_{i,B}\}_{i \in S, B \in \mathcal{B}(i)}\big)$:

 # Periodic aggregation – the aggregator's algorithm

Find a set of blocks $\mathcal{B}$ to uniquely cover S.
$s \leftarrow 0$
for $B \in \mathcal{B}$ **do**
  $s_B \leftarrow \mathsf{BA.Decrypt}(\mathsf{cap}_B, \{c_{i,B}\}_{i \in B})$
  $s \leftarrow s + s_B$
end for
return the estimated sum s

λ	security parameter
n	total number of users
t	current round
x_i	user i's data in current round
$\mathcal{T}(n)$	set of all blocks corresponding to nodes in a binary tree of size n
ϵ, δ	privacy parameters
$\mathsf{sk}_{i,B}$	user i's secret key for block B where $i \in B$
cap_B	aggregator's capability for block B
S	set of functioning users in current round
$\mathcal{B}(i)$	$\mathcal{B}(i) := \{B \mid B \in \mathcal{T}(n) \text{ and } i \in B\}$ set of blocks containing user i

Fig. 2. The Binary Protocol.

The user then computes the ciphertext $c_i := \mathsf{Encrypt}(\mathsf{sk}_i, \widehat{x}_i, t)$, where $\widehat{x}_i$ is the purtubed data, and uploads the ciphertext to the aggregator.

Theorem 1 (Computational Differential Privacy of BA). *Let $\epsilon > 0$, $0 < \delta < 1$ and $\beta := \min\{\frac{1}{\gamma m} \ln \frac{1}{\delta}, 1\}$, where γ is the probability that each user remains uncompromised. If each user adds diluted Geometric noise $\mathsf{Geom}^{\beta}(\alpha)$ (where $\alpha = e^{\epsilon}$), then at each time step, the Block Aggregation Scheme is computationally (ϵ, δ)-differentially private against compromised users.*

4.3 Binary Protocol: Construction

The Binary Protocol consists of running the BA Scheme over a collection of blocks simultaneously. Specifically, if n is a power of 2, then one can build a binary interval tree of the n users such as in Figure 1(a). Each node in the tree represents a contiguous interval, which we call a block. The aggregator and users would run the BA Scheme for all blocks depicted in the interval tree. It is not hard to see that each user i is contained in at most $K := \lfloor \log_2 n \rfloor + 1$ blocks, represented by nodes on the path from the i-th leaf node to the root of the tree.

We now state the above description more formally. Given integers $k \geq 0$ and $j \geq 1$, the jth block of rank k is the subset $B_j^k := \{2^k(j-1)+l : 1 \leq l \leq 2^k\}$ of integers. If there are n users, we only need to consider the blocks B_j^k such that $B_j^k \subseteq [n]$. Define $\mathcal{T}(n)$ to be the set of all relevant blocks when there are n users.

$$\mathcal{T}(n) := \{B_j^k | k \geq 0, j \geq 1, B_j^k \subseteq [n]\}$$

Specifically, when n is a power of 2, $\mathcal{T}(n)$ basically corresponds to the collection of all nodes in the binary interval tree with n leaf nodes. It is not hard to see that the total number of blocks is at most $2n$. The following observations will be important in the design of the Binary Protocol.

Observation 1 *Each user $i \in [n]$ is contained in at most $K := \lfloor \log_2 n \rfloor + 1$ blocks. In particular, each user is in at most one block of rank k.*

Setup phase. Like in the BA Scheme, a one-time trusted setup is performed at system initialization. A trusted dealer distributes $O(\log n)$ secret keys to each user. In particular, each user $i \in [n]$ obtains one secret key corresponding to each block containing the user (i.e., the path from the i-th leaf node to the root). We use the notation $\mathsf{sk}_{i,B}$ to denote user i's secret key corresponding to the block B.

For each block $B \in \mathcal{T}(n)$, the trusted dealer issues a capability cap_B to the aggregator. The aggregator thus receives $O(n)$ capabilities. The parties also agree on other system parameters including the privacy parameters (ϵ, δ).

Periodic aggregation: user algorithm. In each time step $t \in [n]$, each user i performs the following:

For each block B containing the user i, the user generates a fresh random noise r from the diluted geometric distribution $\mathsf{Geom}^{\beta}(e^{\epsilon_0})$, where the choice of parameters β and ϵ_0 will be explained later. The user adds the noise $r_{i,B}$ to her input bit x_i, and obtains $\widehat{x}_{i,B} := x_i + r_{i,B}$. The user then encrypts $\widehat{x}_{i,B}$ using $\mathsf{sk}_{i,B}$, i.e., her secret key corresponding to the block B. Specifically, user i computes

$$c_{i,B} := \mathsf{BA.Encrypt}(\mathsf{sk}_{i,B}, \widehat{x}_{i,B}, t)$$

The final ciphertext c_i uploaded to the aggregator is the collection of all ciphertexts, one corresponding to each block containing the user i.

$$c_i := \{c_{i,B} | B \in \mathcal{T}(n), i \in B\}$$

As each user is contained in $O(\log n)$ blocks, the ciphertext size is also $O(\log n)$.

Parameter choices. Suppose we wish to guarantee computational (ϵ, δ)-differential privacy for the Binary Protocol, where (ϵ, δ) are parameters agreed upon by all parties in the setup phase. Each user needs to determine the parameters ϵ_0 and β when generating a noise from the diluted geometric distribution $\mathsf{Geom}^{\beta}(e^{\epsilon_0})$. Specifically, each user chooses $\epsilon_0 := \frac{\epsilon}{K}$, where $K := \lfloor \log_2 n \rfloor + 1$. When selecting noise for a block B of size $|B|$, the user selects an appropriate $\beta := \min\{\frac{1}{|B|} \ln \frac{1}{\delta_0}, 1\}$, where $\delta_0 = \frac{\delta}{K}$. Notice that due to Theorem 1, the above choice of ϵ_0 and β ensures that each separate copy of the BA Scheme satisfies computational (ϵ_0, δ_0)-differential privacy. This fact is used later to analyze the differential privacy of the entire Binary Protocol.

Intuitively, using the diluted geometric distribution, each user effectively adds a geometric noise with probability β, and adds 0 noise with probability $1-\beta$. Notice that β is smaller if the block size is bigger, since we wish to guarantee that at least one user added a real geometric noise.

More generally, if each user may be compromised with independent probability $1-\gamma$, then each (uncompromised) user would choose $\epsilon_0 := \frac{\epsilon}{K}$, and $\beta := \frac{1}{\gamma|B|} \ln \frac{1}{\delta_0}$ for a block B whose size is $|B|$, where $\delta_0 := \frac{\delta}{K}$.

Periodic aggregation: aggregator algorithm. Suppose $0 \leq \kappa < n$ users have failed to respond. Then the entire range $[n]$ would be divided up into $\kappa + 1$ contiguous interval. The aggregator will recover the noisy sum for each of these intervals, and the sum of these will be the estimate of the total sum.

It suffices to describe how to recover the noisy sum for each of these contiguous intervals. An important observation is that each contiguous interval within $[n]$ can be covered uniquely by $O(\log_2 n)$ blocks. This is stated more formally in the following observation.

Observation 2 (Unique cover for a contiguous interval.) *Let $[s,t]$ denote a contiguous interval of integers within $[n]$, where $1 \leq s \leq t \leq n$. We say that $[s,t]$ can be covered uniquely by a set of blocks $\mathcal{B} \subseteq \mathcal{T}(n)$, if every integer in $[s,t]$ appears in exactly one block in $\mathcal{B}$. For any interval $[s,t] \subseteq [n]$, it is computationally easy to find set of at most $2\lceil \log_2 n \rceil + 1$ blocks that uniquely cover $[s,t]$.*

Therefore, to recover the noisy sum for an interval $[s,t] \subseteq [n]$, the aggregator first finds a set of blocks $\mathcal{B}$ to uniquely cover $[s,t]$. Then, the aggregator decrypts the noisy sum for each block $B \in \mathcal{B}$ by calling the decryption algorithm: $\mathsf{BA.Decrypt}(\mathsf{cap}_B, \{c_{i,B}\}_{i \in B})$. The sum of all these block estimates is an estimate of the total sum.

One possible optimization for decryption is to leverage the homomorphic property of the BA Scheme [14]. Instead of decrypting each individual block estimates, the aggregator can rely on the homomorphic property to compute an encryption of the sum of all block estimates. In this way, only one decryption operation is required to decrypt the estimated sum. As mentioned in Section 4.7 decryption takes $O(n)$ time using the brute-force approach, and $O(\sqrt{n})$ time using Pollard's Rho method.

This concludes the description of our Binary Protocol. Earlier in Section 4.1, we explained the intuition of the above Binary Protocol with a small-sized example. In the remainder of this section, we will focus on the privacy and utility analysis.

4.4 Theoretic Guarantees

Theorem 2 below states that our Binary Protocol satisfies computational (ϵ, δ)-differential privacy, and achieves an error bound of $\tilde{O}(\frac{(\log n)^{1.5}}{\epsilon}\sqrt{\frac{\kappa+1}{\gamma}})$ with high probability (hiding a $\log\log n$ factor and δ, η parameters). Here κ is the number of failed users, and γ is the fraction of users that remain uncompromised.

The intuition behind the proof was explained earlier in Section 4.1. Due to space constraint, we defer the full proof of this theorem to the online full version [2].

Theorem 2 (Error Bound with κ-Failed Users). *Let $\epsilon > 0$ and $0 < \delta < 1$. Suppose each of the n users remains uncompromised independently with probability γ. Then, the Binary Protocol can be run such that it is computationally (ϵ, δ)-differentially private. Moreover, when there are κ failed users, for $0 < \eta < 1$ subject to some technical condition*[4]*, with probability at least $1 - \eta$, the aggregator can estimate the sum of the participating users' bits with additive error at most $O(\frac{(\log n)^{1.5}}{\epsilon} \cdot \sqrt{\frac{\kappa+1}{\gamma}} \cdot \sqrt{(\log\log n + \log\frac{1}{\delta})\log\frac{1}{\eta}})$.*

4.5 Dynamic Joins

First, imagine that the system knows beforehand an upper-bound $n = 2^K$ on the total number of users – if n is not a power of 2, assume we round it up to the nearest power of 2. We will later discuss the case when more than n users actually join. In this case, when a new user i joins, it needs to contact the trusted dealer and obtain a secret key $\mathsf{sk}_{i,B}$ for every block $B \in \mathcal{T}(n)$ that contains i. However, existing users need not be notified. In this case, the trusted dealer must be available to register newly joined users, but need not be online for the periodic aggregation phases. The trusted dealer may permanently erase a user's secret key (or the aggregator's capability) after its issuance.

What happens if more users join than the anticipated number $n = 2^K$? We propose 2 strategies below.

Key updates at every power of two. When the number of users exceeds the budget $n = 2^K$, the trusted dealer sets the new budget to be $n' := 2^{K+1}$, and issues new keys and capabilities to the users and aggregator as follows. For every new block B that forms in $\mathcal{T}(n')$ but is not in $\mathcal{T}(n)$, a new secret key (or capaiblity) needs to be issued to every user contained in B (and the aggregator). Notice that the secret keys for existing blocks in $\mathcal{T}(n)$ need not be updated. In this way, each existing user obtains one additional secret key, the newly joined user obtains $O(\log n)$ secret keys, and the aggregator obtains $O(n)$ capabilities. Notice that such key updates happen fairly infrequently, i.e., every time the number of users reach the next power of 2.

Allocate a new tree. When the number of users reach the next power 2^K of two, the trusted dealer allocates a new tree of size 2^K. For every block in the new tree, the trusted dealer issues a capability to the aggregator corresponding to that block. For the next 2^K users that join the system, each user is issued $O(K)$ secret keys corresponding to blocks in the new tree. Hence, the sizes of the trees are $1, 1, 2, 4, 8, ...$ and so on.

When the aggregator estimates the sum, it will simply sum up the estimate corresponding to each tree. Suppose the number of current users is n. Then, there are $O(\log n)$ such trees. A straightforward calculation shows that the additive error made by the aggregator will be $\tilde{O}(\frac{(\log n)^3}{\epsilon})$ with high probability.

The advantage of this approach is that only the aggregator needs to be notified when the number of users changes. The existing users need not be notified. Therefore, this

[4] The following condition is satisfied certainly when n is large enough: $\frac{(\kappa+1)\log_2 n}{\gamma} \ln\frac{\log_2 n}{\delta} \geq \exp(\frac{\epsilon}{\log_2 n}) \ln\frac{2}{\eta}$.

approach is particularly suited when making push notifications to users may be difficult (e.g., when users are frequently offline).

4.6 Dynamic Leaves

When a user leaves, that user can be treated as permanently failed. As mentioned in Theorem 2, the estimation error grows only sub-linearly in the number of absent users.

For reduced error and higher utility, sometimes we may consider repeating the setup phase when too many users have left. The application designer can make this choice to fit the characteristics and requirements of the specific application.

4.7 Practical Performance

Consider a scenario with $n \simeq 10,000$ users. The Binary Protocol leverages the BA Scheme scheme proposed by Shi *et al.* [14]. According to their performance estimates [14], each encryption takes about 0.6 *ms* on a modern computer, when we use high-speed elliptic curves such as "curve25519". When $n \simeq 10,000$, each user needs to perform roughly $\lfloor \log_2 n \rfloor + 1 = 14$ encryptions. Therefore, a user's computation overhead is roughly $8 \sim 9$ *ms* on a modern computer.

Decryption of the underlying BA Scheme requires taking a discrete logarithm. The brute-force method involves enumerating the plaintext space. It takes one modular exponentiation, roughly 0.3 *ms* to try each possible plaintext. With $n = 10,000$ users, our simulation shows that the additive error is under 500 with $> 99\%$ probability (when $\epsilon = 0.5$, $\delta = 0.05$, and in the absence of failures). Therefore, the brute-force method takes on average 1.5 seconds to decrypt the sum. We can speed up decryption significantly using one of the following methods: 1) Use Pollard's Rho method, which reduces the decryption overhead to about $\sqrt{n + o(n)}$. 2) Exploit parallelism. The brute-force method is trivially parallelizable, and particularly suited for modern clusters such as MapReduce or Hadoop.

5 Discussions

Faster decryption. One limitation of the proposed scheme is that the decryption time is $O(\sqrt{n})$ using Pollard's Rho method. As a result, we need the plaintext space to be polynomially sized. While Sections 4.3 and 4.7 have proposed some methods to make decryption faster in practice, we also point out that another method would be to replace the encryption scheme entirely with the encryption scheme used by Rastogi *et al.* [13]. Basically, the binary tree method can be regarded as a generic approach which can be applied on top of both the works by Rastogi *et al.* [13] and Shi *et al.* [14]. If we use the scheme by Rastogi *et al.* [13] as a building block, we remove the constraint of polynomially-sized plaintext space, at the cost of introducing interactions between the users and the server (however, still, no peer-to-peer interaction would be needed).

Operations in an algebraic group. Due to the use of cryptography, integer additions are in fact performed in a discrete mathematical group of prime order p, which is needed by the encryption algorithm in the BA Scheme. Our error analysis also guarantees that with high probability, no integer overflow or underflow will happen.

6 Conclusion

We investigated how an untrusted aggregator can learn aggregate statistics about a group of users without harming each individual's privacy. Our construction addresses fault tolerance, a question left open by earlier works in this area [13,14]. Our construction is desirable in the sense that it requires no peer-to-peer communication (unlike the traditional approach of Secure Multi-Party Computation), and achieves high utility guarantees.

Acknowledgments

This work 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. A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *STOC*, 2008.
2. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. Full online technical report, `http://eprint.iacr.org/2011/722.pdf`, 2011.
3. H. Chan, E. Shi, and D. Song. Tight lower bounds for distributed private data analysis. In submission, 2011.
4. T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. In *ICALP*, 2010.
5. C. Dwork. Differential privacy. Invited talk at *ICALP*, 2006.
6. C. Dwork. A firm foundation for private data analysis. In *Communications of the ACM*, 2010.
7. C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, 2006.
8. C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
9. C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.
10. A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC*, 2009.
11. I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *CRYPTO*, 2009.
12. B. Przydatek, D. Song, and A. Perrig. Sia: secure information aggregation in sensor networks. In *ACM Sensys*, 2003.
13. V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD 2010*, pages 735–746, 2010.
14. E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.
15. L. Whitney. Microsoft urges laws to boost trust in the cloud. `http://news.cnet.com/8301-1009_3-10437844-83.html`.