Secure Multiparty Aggregation with Differential Privacy: A Comparative Study

Slawomir Goryczka
Emory University
sgorycz@emory.edu

Li Xiong
Emory University
lxiong@emory.edu

Vaidy Sunderam
Emory University
vss@emory.edu

ABSTRACT

This paper considers the problem of secure data aggregation in a distributed setting while preserving differential privacy for the aggregated data. In particular, we focus on the secure sum aggregation. Security is guaranteed by secure multiparty computation protocols using well known security schemes: Shamir's secret sharing, perturbation-based, and various encryption schemes. Differential privacy of the final result is achieved by distributed Laplace perturbation mechanism (DLPA). Partial random noise is generated by all participants, which draw random variables from Gamma or Gaussian distributions, such that the aggregated noise follows Laplace distribution to satisfy differential privacy. We also introduce a new efficient distributed noise generation scheme with partial noise drawn from Laplace distributions.

We compare the protocols with different privacy mechanisms and security schemes in terms of their complexity and security characteristics. More importantly, we implemented all protocols, and present an experimental comparison on their performance and scalability in a real distributed environment.

1. INTRODUCTION

Participatory sensing and data surveillance [6,22] are gradually integrated into an inseparable part of our society. In many applications, a data aggregator may wish to collect personal data from multiple individuals to study patterns or statistics over a population. *Data privacy and security* issues arise frequently and increasingly in such data surveillance systems [1, 27, 33, 41]. An important challenge is how to protect the privacy of the data subjects, especially when the data aggregator is untrusted or not present.

Example Applications. Many applications exist in a variety of domains, such as smart metering, sensor aggregation, traffic control, and crowd monitoring. We consider a following motivating scenario of intelligence data collection.

As recent events demonstrate, numerous situations exist, where intelligence gathering is performed in crowd settings both non-deliberately by the general public and by principals, who are anonymously embedded in the crowds. A canonical example is an uprising in a major city under hostile governmental control – the general public uses smart devices to report on various field data (*third party surveillance* [27]). There may also be agents among the crowd, reporting similar data using similar media (e.g. Twitter) to avoid identification. **Figure 1** illustrates the scenario where a city square is monitored by three data contributors (colored dots).

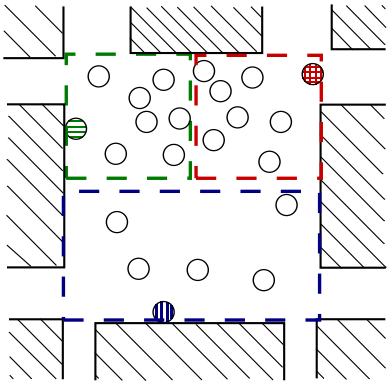


Figure 1: An example scenario: a town square divided into 3 zones monitored by data contributors (colored dots).

In order to aggregate information that contains personal data without involving a trusted aggregator, two important constraints must be fulfilled: 1) privacy of individuals or data subjects whose data are being collected, and 2) security of data contributors who need to protect their data from other contributors as well as the untrusted aggregator.

Differential privacy is the state-of-the-art privacy notion [16, 17, 28] that gives a strong and provable privacy guarantee for aggregated data. It requires that the output of an aggregation or computation should not change significantly, even if a data subject had opted out of the data collection. Therefore, this assures an individual that any privacy breach will not unveil participation of its record in the data collection. Such strong privacy guarantee is achieved only if we assume that user data are independent, i.e. deleting one record of data is equivalent to hiding evidence of its participation in the aggregated data, and further no deterministic statistics about the participating records have been previously released [28].

EDBT/ICDT '13 Workshops, March 18–22, 2013, Genoa, Italy.

A common way of achieving differential privacy is to perturb the aggregated statistics. When a trusted aggregator (TA) is available, it collects the data, aggregates them, and adds calibrated noise to the result. However, in many scenarios, the TA is not available (e.g. in the intelligence collection scenario). Even for scenarios with the TA, relying on the centralized entity makes it a single point of failure for the entire data collection. It is justified to always assume lack of the TA.

For such decentralized scenarios, all participating data contributors need to perform aggregations and perturbations among themselves, while ensuring that no sensitive information is disclosed. In the simplest and the most naive approach, each data contributor perturbs its own data to guarantee their differential privacy. However, this will result in an aggregated noise that significantly exceeds the required amount to ensure differential privacy for the aggregated result. In addition, data reported by contributors may still disclose sensitive information to the untrusted aggregator. This problem is typically formulated as a secure multiparty computation problem (SMC), in which parties wish to jointly compute a function of their private data inputs [2, 8, 9, 30, 36, 37]. Solutions for such problem are SMC protocols, which securely implement distributed computations. A protocol is *secure* if the parties learn only its final result and nothing else. SMC protocols that return differentially private results, which is achieved by privacy mechanisms, guarantee that privacy of data subjects and data contributors is protected at all time.

Contributions. In this paper, we review and present a comparative study of existing solutions for the problem of secure data aggregation with differential privacy. In particular, we focus on the secure sum aggregation. Among all secure computation schemes, we evaluate three representative groups: secret sharing [7, 39], homomorphic encryption [2, 26, 35, 40], and perturbation-based [11]. They represent different approaches of ensuring security: splitting data into shares (secret sharing), encrypting data (homomorphic encryption), and perturbing data with significant noise (perturbation-based). Among homomorphic encryption schemes we evaluate three of them, which are named after their authors: Paillier, Ács, and Shi.

To ensure differential privacy in a distributed setting without any trusted aggregator, we evaluate existing distributed perturbation mechanisms, which are named after distributions of partial noise: Gamma [2] and Gauss [37]. We also propose a new efficient mechanism that draws partial noise from Laplace distributions.

We compare complexities and security characteristics of protocols implemented in different schemes and using different privacy mechanisms. More importantly, we implemented all protocols, and present a comprehensive experimental comparison of their performance and scalability in a real distributed environment. Based on the results, we discuss most effective solutions in various settings.

2. RELATED WORK

Secure Multiparty Computation Protocols. Secure multiparty computations (SMC) has been defined by Yao, who presented a solution to the Millionaires' Problem, where two millionaires want to find out, who is richer without disclosing their actual wealth [44]. Security of multiparty computations has been formally defined in [25]. General SMC protocols, which can perform any computations, are computationally expensive. However, many specialized protocols have been developed for complex tasks, e.g., in data mining [11, 25, 30]. Among them are: a secure sum protocol, a secure union protocol, and a secure scalar product [11]. For computing sum, we use efficient techniques based on threshold homomorphic functions [12] and random shares [42].

Threshold variant of the Paillier cryptosystem has been introduced by Damgård *et al.* [13], and used to implement an electronic voting system. Hazay *et al.* presented a threshold Paillier encryption scheme for two-party setting, and a protocol for generating an RSA composite [26].

Pedersen *et al.* compared encryption-based and secret sharing schemes used in privacy preserving data mining [36]. They presented schemes from both groups, but their comparison does not include performance evaluations, and lacks the latest security schemes and privacy mechanisms.

Differential Privacy. Dwork *et al.* have developed distributed algorithms for random number generations, which shares are drawn from binomial and Poisson distributions [18]. Noise drawn from the binomial distribution (as an approximation of the Gaussian) or the Poisson distribution (as an approximation of the exponential) was used to perturb results of queries run against distributed databases. Perturbed results are δ-approximate α-indistinguishable for the former noise, and α-indistinguishable in the latter. However, generating noise requires from nodes exchanging many messages.

Secure Multiparty Computations with Differential Privacy. Several works studied SMC protocols while preserving differential privacy for the final results. Ács *et al.* applied privacy-preserving and secure data aggregation to the smart metering system [2]. Their scheme uses differential privacy model and a homomorphic properties of a modulo addition-based encryption scheme to ensure privacy of results and security of computations. Differential privacy is achieved by adding Laplace distributed noise to results. Noise is generated distributively, such that each meter generates one share of it, i.e., the difference between two Gamma distributed random variables. Below we name such mechanism Gamma DLPA, and evaluate it.

In a similar scenario individual users collect, and aggregate time-series data. PASTE is the system that securely aggregates such data while ensuring differential privacy of results [37]. Differential privacy is achieved by perturbing the most significant coefficients of the Discrete Fourier Transform of the query answers. Coefficients are perturbed by a Distributed Perturbation Laplace Algorithm (DLPA), which generates noise, by adding partial noise drawn by all contributors in a distributed way. To securely aggregate computed values, and generate noise, PASTE utilizes the threshold Paillier cryptosystem, which is additively homomorphic, i.e., allows to compute the encrypted sum of values encrypted using the same public key. Partial noise generated by each participant is a vector of four Gaussian random variables, which are used to securely calculate the final noise. We name such mechanism Gauss DLPA, and evaluate it.

Shi *et al.* also utilized a homomorphic encryption scheme, and minimized communication by generating an encryption key from the current round number and the originally established key [40]. Their privacy mechanism is distributed, but ensures approximate differential privacy with noise drawn

from the symmetric geometric distribution [24].

3. SYSTEM SETTING AND MODELS

In this section, we describe our problem setup and adversary model, formally present security and privacy models, and discuss reliability and performance requirements.

3.1 System Setting

We consider a dynamic set of *data contributors* or *nodes* (colored dots in **Figure 1**). They participate, and contribute their own data (self surveillance) or other data (third party surveillance) in a surveillance system. In our running example, nodes collect data from their areas independently, and aggregate them. We use *data subjects* to refer to the individuals represented in the collected data, which are the same as data contributors in the self surveillance case. We assume there is an untrusted application or an application run by an untrusted party for analysis and modeling (e.g. disease outbreak detection or intelligence analysis).

In a *centralized model with a trusted aggregator (TA)*, the TA (e.g. CDC offices in the syndromic surveillance scenario) collects the data, aggregates them, performs appropriate data perturbation, and outputs perturbed aggregates with privacy guarantee, which can be in turn used for modeling and predictive studies. In such setting, both security and privacy can be achieved easily, but for the price of making the TA a single point of failure for the entire system.

In a *decentralized setting without a TA* (e.g. in the intelligence collection scenario), the data contributors need to perform aggregations and perturbations among themselves through secure multiparty computation protocols, and submit the aggregated result to the untrusted aggregator or the application directly. Such settings are reliable, but are also more complex to design, and run. In this paper we analytically and experimentally compare existing distributed systems, and introduce new ones that achieve the same level of privacy and security with reliability.

3.2 Adversary Model

We assume that end users are curious, but not intrusive. Their main goal is to infer sensitive information about data subjects using final results, and any publicly known data.

We further assume that data contributors are *semi-honest*, i.e., they follow a protocol correctly, but may passively observe, and use any intermediate results to infer sensitive information of other data contributors or data subjects. In addition, a group of contributors may collude to increase their chances of successful attack.

3.3 Privacy Model

Privacy of *data subjects* is protected, if we can guarantee that data recipient will not learn anything about them including their participation in the data collection. Traditional approaches such as removing identifying attributes, generalizing, or perturbing individual attribute values, are susceptible to various attacks [21]. In our study we use the state-of-the-art differential privacy as our privacy model, which gives a strong and provable privacy guarantee [16, 17, 28]. Differential privacy requires that the result of an aggregation or computation should not change significantly, if any single data subject had opted out of the data collection. Therefore, this assures an individual that its participation in the data collection can be guessed only with negligible probability regardless a priori knowledge of an attacker. Informally, an attacker that knows all but one record, and the result of computations, shall not be able to find out, if the remaining record participates in the collection. Formally, differential privacy is defined as follows.

Definition 3.1 (α-Differential Privacy [15, 18, 28])

A randomized mechanism $\mathcal{A}$ satisfies α-differential privacy if for any neighboring databases D_1 and D_2, where D_1 can be obtained from D_2 by either adding or removing one record, and any possible output set S,

$$Pr[\mathcal{A}(D_1) \in S] \leq e^{\alpha} \cdot Pr[\mathcal{A}(D_2) \in S]$$

A common mechanism to achieve α-differential privacy is Laplace perturbation (LPA) [16]. LPA ensures that results of an aggregated query Q are α-differentially private, by returning $Q(D) + N$ in place of the original result $Q(D)$. N is a random noise drawn from the Laplace distribution $Lap(\Delta_Q/\alpha)$ with a probability density function $Pr(N = x) = \frac{\alpha}{2\Delta_Q} e^{-|x|\alpha/\Delta_Q}$, where Δ_Q is global sensitivity of Q.

Definition 3.2 (Global Sensitivity [19])

The sensitivity of any aggregate function $Q : D \rightarrow \mathbb{R}^d$, is equal to $\Delta_Q = \max_{D_1, D_2} ||\mathcal{A}_Q(D_1) - \mathcal{A}_Q(D_2)||_1$ for all D_1, D_2 differing in at most one record.

Note that global sensitivity does not depend on dataset, but only on query Q. Therefore, knowing its value is not helpful to breach privacy.

3.4 Security Model

The main security goal in distributed systems is to ensure that the data recipients or participating data contributors learn only the final result, and nothing else about the private data provided by other data contributors. Relying only on privacy mechanisms is not enough to meet our security goals. If we apply privacy mechanisms to ensure security, the final results will have very limited utility. At the same time satisfying only security requirements does not protect privacy of data subjects. Therefore, both security schemas and privacy mechanisms are necessary in our system.

We use the formal security notion to present the secure multiparty computation (SMC) protocols [25, 30, 44]. In an SMC protocol, parties jointly compute a function of their private data inputs. The protocol is *secure* if the parties learn only the result of the function and nothing else.

Definition 3.3 (Secure computation [25])

Let f be a function of n variables, and Π be an n-party protocol for computing f. The view of the i^{th} party during an execution of Π on $X = (x_1, \ldots, x_n)$ is denoted $\text{VIEW}_i^{\Pi}(X)$, and for a group of parties with indices $I \subseteq [n]$, we let $X_I = \{x_i \in X : i \in I\}$ and $\text{VIEW}_I^{\Pi}(X) = (I, \text{VIEW}_{i_1}^{\Pi}(X), \ldots, \text{VIEW}_{i_t}^{\Pi}(X))$. We say that Π securely computes f if there exists a probabilistic polynomial-time algorithm, denoted S, such that for every I, it holds that

$$\{S(I, X_I, f(X))\} \equiv \{\text{VIEW}_I^{\Pi}(X), \text{OUTPUT}^{\Pi}(X)\}$$

where $\text{OUTPUT}^{\Pi}(X)$ denotes the output sequence of all parties during the execution represented in $\text{VIEW}_I^{\Pi}(X)$.

Informally, an SMC protocol is secure, if all parties learn only what can be computed from its results, i.e., simulated

by S. Note that security of f does not guarantee privacy of $f(X)$, i.e., $f(X)$ may still disclose sensitive information. However, applying privacy mechanisms to computations of f preserves privacy of $f(X)$.

3.5 Additional Requirements

Besides the security and privacy requirements, we briefly discuss a few additional requirement and design goals for the applications we consider.

In real life scenarios, computation and communication resources are often limited. For example, participants that collect data may have only cell phones with minimal computation power. A very important limitation of many mobile devices is their battery life. This makes the power consumption for running a protocol an important evaluation metric. Power consumption is frequently correlated with computation and communication complexities, which we evaluate in this paper.

In addition, data contributors can be faulty or off-line. Ensuring secure and privacy preserving distributed computations with limited resources and in faulty environment is an additional challenge. We analyze fault tolerance of all presented security schemes.

Finally, communication channels among nodes may be heterogenous, and direct communication between any two nodes may not be always possible and trusted. Thus, securing these channels is an important requirement for a few presented solutions.

4. SECURE COMPUTATION SCHEMES

In this section we describe, and analytically compare several different secure computation schemes, which are used to implement secure aggregation protocols. Experimental comparisons are presented in Section 6.

4.1 Homomorphic Encryption

An encryption scheme is homomorphic if it allows computations to be carried out on ciphertext. Formally, for a given homomorphic encryption function E with respect to a function f, the encrypted result of f can be obtained by computing a function g over encrypted values of $x_1, \ldots, x_n$, i.e.,

$$E(f(x_1, \ldots, x_n)) \equiv g(E(x_1), \ldots, E(x_n)) \quad (1)$$

In a distributed settings each party encrypts its data, and sends them to a single party, which computes the function g on them. Fully homomorphic schemes allow secure multiparty computation of any function f. The price for such flexibility in choosing f is high complexity [23,43]. However, a few partially homomorphic schemes are efficient enough to achieve our security and performance goals, e.g., multiplicatively homomorphic ElGamal [20] and RSA [38] schemes. Examples of additively homomorphic schemes are: Paillier [35], Ács [2], and Shi [40].

Paillier Scheme. The Paillier scheme is a probabilistic public-key encryption scheme [13, 26, 35], which works as follows. A node i encrypts its private value x_i using the encryption function E and the public key. Then, all encrypted values are collected by a single node, which computes the encrypted sum, i.e., $g(E(x_1), \ldots, E(x_n))$. The sum is broadcasted, and partially decrypted by each node. One node collects all partially decrypted sums, and reveals the result.

The original protocol has been enhanced to a threshold scheme, in which shares of a private key are distributed, and any t out of n shares are sufficient to decrypt the ciphertext. Details of key generation, encryption, and decryption algorithms can be found in [13, 26].

Generation of public and private keys is a crucial task to ensure security of this scheme, because a party that has the private key is able to decrypt all ciphertexts. The public key and shares of the private key are generated by an entity that is not involved in computation, e.g., an administrator, a trusted data collector. Such an entity generates, distributes, and drops shares of the private key, which is part of the setup process. The distributed private key can be also generated by running a separate SMC protocol [14, 34], e.g., Diffie-Hellman key exchange protocol [2].

Ács Scheme. The Ács scheme is a modulo addition-based encryption scheme [2], i.e., addition is the encryption function. Encryption keys are generated in pairs by two nodes, such that their sum is equal to 0, e.g., by running Diffie-Hellman key exchange protocol. Each node has l encryption keys, and their inversions are held by l other nodes. Since in the final result all encryption keys are aggregated, they cancel out themselves, and explicit decryption is not necessary. Establishing l keys for each node is inefficient, if done for each aggregation round. Unfortunately, reusing the same key leads to potential security breaches.

Shi Scheme. In the Shi scheme removing perturbation is not necessary as well [40]. Similar as in the Ács scheme, initial encryption keys are established in the setup phase. Although all nodes need to participate to decrypt the result, they do not need to communicate in order to establish encryption keys for a new run. For each run, an encryption key is computed from the initially established key using a one-way function, i.e., inverting it is computationally impossible. This approach minimizes communication among nodes, but requires additional computational power.

4.2 Secret Sharing

Secret sharing schemes split a secret into multiple shares such that at least t shares are required to reconstruct the secret. Additionally, any set of fewer than t shares disclose nothing about the secret. The value of t also defines the minimal number of colluding nodes necessary to breach the security of computations. Shares are distributed among participants, and each node receives a few (usually one) shares.

We consider the Shamir secret sharing scheme, which has been proposed in [39]. In this scheme shares from different secrets can be summed up into shares of their sum, which can be then reconstructed. Other arithmetic and set operations are also possible, and implemented [7].

Shamir Scheme. The protocol works as follows. Each node i generates n shares of its local secret value x_i, by randomly generating coefficients of a polynomial w_i of order t, such that $w_i(0) = x_i$. For a set of selected and publicly known points $z_1, \ldots, z_n$ shares of x_i are generated using the polynomial such that $x_{i,j} = w_i(z_j)$ for $j = 1, \ldots, n$. Then, shares are distributed among nodes, such that $x_{i,j}$ is sent to node j. After exchanging all shares, node j computes a share at z_j, i.e., sums up received shares $\sum_i x_{i,j}$. Finally, one node collects at least t shares of the sum, interpolates a polynomial $\sum_i w_i$, and computes the sum $\sum_i x_i = \sum_i w_i(0)$.

Note that originally, Shamir scheme was introduced for

integer numbers. Adapting the scheme to floating point numbers requires implementing different arithmetic operation protocols [10], or encoding floating point numbers as integers, e.g., by multiplying them by the same power of 10.

Ensuring security of communication channels is crucial for the secret sharing scheme. Such security can be provided by encrypting communication channels, e.g., by using a Secure Sockets Layer (SSL). Since each participant sends at least t shares to others, the amount of communication for such schemes is relatively high.

Assuming that all communication channels are secure, Shamir scheme is information-theoretically secure, i.e., is secure against computationally unbounded attackers [4]. It is also immune to collusion of up to t parties. However, since efficient implementation of secure communication channels is done using encryption, the scheme is, in fact, computationally secure, i.e., using current computer technology an attacker is not able to break it. SEPIA is an example framework that implements secure arithmetic operations on shares generated by the Shamir scheme [7].

4.3 Perturbation-Based Protocols

Perturbation-based protocols are an efficient alternative for encryption-based protocols. They usually require certain topology of node connections. For example, in a secure sum protocol, nodes are connected in a ring [11]. The main idea behind these protocols is to perturb input data by adding some random noise, such that they become meaningless for an attacker, and then perform computations on the noisy data. This approach obfuscates all intermediate results, but is vulnerable to colluding nodes. In addition, it has low level of fault tolerance. Fault of any node requires rerunning the protocol by remaining active nodes.

As an example of perturbation-based protocols, we consider the secure sum protocol [11]. In this protocol all nodes are connected in a ring. Each node generates random noise, which is added to its private input. The starting node sends its perturbed value to its successor, which adds its own perturbed value, and passes it further. At the end of the first phase, the starting node gets the sum of perturbed values from all nodes. In the second phase, each node removes its noise from the perturbed sum, which, at the end, reveals the final sum.

Unfortunately, if two neighbors of a node collude, then they can easily compute the perturbation, and the participated value of that node. Many enhancements have been introduced to this security scheme to increase its collusion resistance, e.g., shuffling node positions in the ring, and dividing computations into multiple rounds.

4.4 Comparison

To compare analytically different types of protocols we present their complexity and security characteristics. Experimental performance comparisons are in Section 6.

A summary of the comparison is presented in **Table 1**. Among homomorphic encryptions Paillier and Shi schemes incur high computation overheads due to their encryption operations and length of keys. Therefore, these schemes may be suitable for scenarios where participating nodes have some computation power, and will scale well due to their linear communication complexity. The minimal amount of communication is generated by Shi scheme. In addition, this scheme is immune to $(n-2)$ colluding nodes, but will not be able to recover final results, if any node faults. In the Ács scheme the level of protection against colluding nodes can be set up as needed. Although, the scheme is fault tolerant, all encryption keys need to be regenerated in the beginning of each round, which causes exchanging tn additional messages. Among encryption schemes, there is no clear winner, but the Ács scheme could be used in majority of settings.

Scheme	**Communication complexity**	**Fault tolerance**	**Max. collusion (max. $n-2$)**
Paillier	$5n$	up to $(n-t)$	$t-1$
Ásc	$(2+t)n$	up to n	$t-1$
Shi	$2n$	0	$n-2$
Shamir	$3(n-1)^2$	up to $(n-t)$	$t-1$
Perturbation-based	$3n$	0	1

Table 1: Comparison of SMC schemes for n nodes.

Shamir secret sharing scheme does not require significant computational resources, and its fault tolerance level depends on t. However, the high communication complexity is a major drawback that limits efficiency and scalability of the scheme for scenarios with a large number of nodes. Additionally, all communication channels need to be secured.

Perturbation-based protocols depend on specific computations, and require participation of all nodes. Thus, their fault tolerance level is 0. In the presence of colluding nodes, schemes do not ensure sufficient security, which is their major limitation. However, they are suitable for settings, where nodes are not likely to fault, and have limited resources.

5. DISTRIBUTED DIFFERENTIAL PRIVACY MECHANISMS

The most common way of achieving differential privacy of aggregated data is by using Laplace mechanism, which perturbs the result with noise drawn from Laplace distribution $\mathcal{L}$. Since the trusted aggregator is not present in our scenarios, and not a single node should know the overall noise, it needs to be generated in a distributed manner.

Distributed Perturbation Algorithms. In distributed perturbation Laplace algorithms (DLPA), each node generates partial noise, such that the aggregated noise follows the Laplace distribution, which is enough to guarantee differential privacy. Due to infinite divisibility of Laplace distribution [29], a random variable with such distribution can be computed by summing up n other random variables. We utilize this property, and define a few algorithms named after distributions that are used to draw partial noise, i.e., Gamma, Gauss, and Laplace. Gamma and Gauss have been already introduced [2,37], while Laplace is introduced in this paper as an alternative for them. We describe, and compare all three DLPA mechanisms below. We denote $\mathcal{L}(\theta, s)$ as a random variable with the Laplace probability distribution having the mean equal to θ, and the scale equal to s.

5.1 *Gamma*

Gamma DLPA mechanism utilizes infinite divisibility of Laplace distribution, and generates partial noises by drawing random variables from the gamma distribution [2]. Formally, a Laplace distributed random variable can be simu-

lated by the sum of n random variables as follows,

$$\mathcal{L}(\theta, s) = \frac{\theta}{n} + \sum_{i=1}^{n}(G_i - G_i') \quad (2)$$

G_i and G_i' are gamma distributed random variables with densities following the formula ($x \geqslant 0$),

$$pdf(G_i) = pdf(G_i') = \frac{(1/s)^{1/n}}{\Gamma(1/n)} x^{1/n-1} e^{-x/s} \quad (3)$$

Γ is the gamma function, such that $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}\mathrm{d}x$.

Note that generating a random variable with gamma distribution requires drawing at least 2 random variables [3]. Assuming that all values of α are equally probable, the average number of generated uniformly distributed random numbers is equal to 2.54.

5.2 *Gauss*

A random variable with Laplace distribution can be also simulated by 4 random variables drawn from the normal distribution $\mathcal{N}(0, s/2)$ with variance $(s/2)$ [29] as follows,

$$\begin{aligned}\mathcal{L}(0, s) &= \mathcal{N}(0, s/2)^2 + \mathcal{N}(0, s/2)^2 \\ &\quad -\mathcal{N}(0, s/2)^2 - \mathcal{N}(0, s/2)^2 \end{aligned} \quad (4)$$

Since infinite divisibility of the normal distribution is stable, drawing a single random variable is simulated by the sum of n Gaussian random variables as follows,

$$\mathcal{N}(0, s/2) = \sum_{i=1}^{n} \mathcal{N}\left(0, \frac{s}{2n}\right) \quad (5)$$

The above formulas can be implemented in distributed settings, but secure computation of squares of random variables is a challenge. An SMC protocol implementing such computations has been introduced in [37], and uses homomorphic encryption, but has high complexity.

Generating a random variable from the Gaussian distribution requires, on average, drawing from uniform distribution only one random variable [5]. Additional approaches, which are more efficient on average but slow in the worst-case scenario, have been introduced in [31, 32].

5.3 *Laplace*

Infinite divisibility of the Laplace distribution is also stable [29]. Thus, drawing a random variable with Laplace distribution (with mean equal to zero) can be simulated using other random variables drawn from the same distribution, as defined by the following formula,

$$\mathcal{L}(0, s) = \sqrt{B_{n-1}} \sum_{i=1}^{n} \mathcal{L}(0, s) \quad (6)$$

B_{n-1} is a random variable drawn from beta distribution with parameters 1 and $(n-1)$, i.e., with the probability density function following the formula $(n-1)(1-x)^{n-2}$ for $x \in (0, 1)$. Note that B_{n-1} is a single random variable for all n shares, which is generated and distributed by one node.

Drawing a single random variable from the Laplace distribution requires running a random number generator only once. For each uniformly distributed random variable U taken from the range $(-0.5, 0.5)$, the following variable will follow the Laplace distribution,

$$\mathcal{L}(\theta, s) = \theta - \mathrm{sgn}(U) \cdot s \cdot \ln(1 - 2|U|) \quad (7)$$

Similarly, drawing a random variable B_{n-1} requires generating only one uniformly distributed random variable.

5.4 Comparison

All DLPA mechanisms guarantee the same level of differential privacy. However, drawing partial noise for each method has different computation cost.

To compare the complexities, we consider the number of random variables that each node generates for each method. For the Laplace mechanism, each node generates a single random variable, and one node generates 2 random variables, i.e., $1 + 1/n$ random variables per node, which makes it the most efficient mechanism. The implementation of gamma random number generator has an indeterministic number of steps, and requires generating at least 2 (on average 2.54) uniformly distributed random numbers [3]. The Gauss mechanism requires each node to generate 4 different random variables from the normal distribution. In order to draw a random variable from the normal distribution, it is sufficient to draw a single uniformly distributed number by methods such as the Ziggurat method [32].

6. EXPERIMENTS

In this section, we present a set of experimental evaluations of the various protocols. Since the security and privacy levels of the protocols have been formally analyzed above, we mainly focus on their performance. The questions we attempt to answer are: 1) How do the different distributed noise generation schemes compare with each other in terms of efficiency? 2) How do the different secure computation protocols perform in various settings, and how do they scale and compare with each other in terms of computation and communication cost?

6.1 Experiment Setup

All experiments have been run using JVM 1.6. We evaluated local computations including partial noise generation, and data preparation on three different platforms: 1) a *cluster* of 64 HP Z210 *nodes* with 2 quad-core CPUs, 8 GB of RAM each, running Linux Ubuntu system, 2) a *laptop* with Intel Core 2 Duo T5500 and 2 GB of memory running Windows XP, and 3) a shared *server* Sun Microsystems SunFire V880, with 8 CPUs and 16 GB of memory running SunOS 5.10. All protocols are evaluated in a distributed environment using the cluster of nodes, which are connected by the 100Mbit network. All reported results are averaged from 1000 runs.

Our main software framework is built on top of SEPIA [7], which uses Shamir's secret sharing scheme for secure distributed computations implemented in Java. We have extended SEPIA with other secure computation schemes and distributed noise generation mechanisms to achieve differential privacy of the final results. We chose implementation of the Paillier scheme from the UTD Paillier Threshold Encryption Toolbox[1]. Additionally, we use an HPC library Colt[2] for random number generation. All remaining schemes and mechanisms have been implemented by authors. Default values of experiment parameters are listed in the **Table 2**.

[1] `http://www.utdallas.edu/~mxk093120/paillier`
[2] `http://acs.lbl.gov/software/colt`

Name	Description	Default Value
n	Number of running nodes	32
k	Size of encryption keys in bits	128
γn	Number of trusted nodes	8
t	Shamir secret sharing threshold, i.e., the min number of shares required to reconstruct the secret	3
	The key size (in bits) of the AES encryption with RSA for SSL communication channels	128

Table 2: Default values of experiment parameters.

6.2 Privacy

The main goal of this experiment is to evaluate the overhead of all DLPA (Laplace, Gamma, and Gauss) privacy mechanisms (Section 5). All of them guarantee that the final results have the same level of differential privacy, i.e., noise added to the results is drawn from the same distribution. We compare the local computation time of the three noise generation schemes, as well as their impact on the overall protocol performance.

Noise share generation. In order to ensure that enough noise is added to the final result, each node adds its share of the noise. The average generation time of such shares for different mechanisms is shown in **Figure 2**.

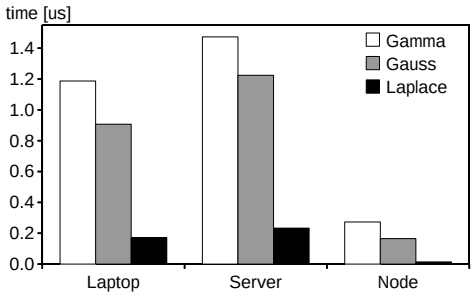


Figure 2: The average noise share generation times in microseconds for different mechanisms and platforms, taken from $1,000,000$ tries.

Generating a noise share for the Laplace DLPA is the most efficient, which confirms our expectations (Section 5.4). It only requires a single uniformly distributed random number and some simple arithmetics, in order to get a noise share. Gauss mechanism requires generating 4 normally distributed random numbers, while Gamma mechanism, on average, requires slightly over 5 uniformly distributed random numbers [3]. Note that Laplace DLPA requires also a random number drawn from the beta distribution. This number is generated by one node, and broadcasted to all nodes as part of the setup message, therefore it is not considered here.

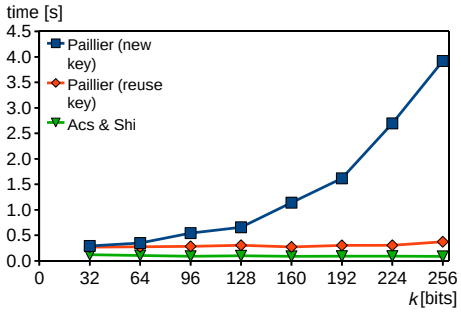
Impact on Protocol Performance. For devices with plenty of resources, e.g., desktop computers, the different privacy mechanisms have small impact on the overall runtime. However, for devices with limited computational resources (e.g. smart phones) running the aggregation protocol repeatedly, the amount of randomly generated numbers impacts energy consumption significantly. Thus, choosing the most efficient privacy mechanism, which also minimize redundant noise, is important in these settings.

6.3 Security

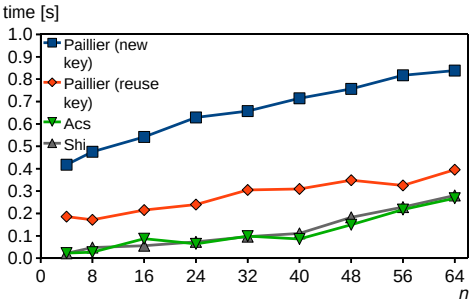
In this group of experiments we evaluate performance of distributed aggregation protocols for different security schemes. Security levels guaranteed by each scheme have been already discussed (Section 4).

6.3.1 Homomorphic Encryption

In this set of experiments we evaluate homomorphic encryption schemes in order to identify factors impacting their performance and security. In the setup phase encryption keys are generated, and distributed. To ensure maximal security of the Paillier and the Ács schemes each round is performed with a new set of encryption keys. Generation and distribution of encryption keys in the Ács scheme has negligible overhead, but in the Paillier scheme the overhead is significant. Thus, if we lower our security requirements, the same encryption key can be reused for a few runs. Therefore, we set up the Paillier scheme to be run in two settings named *new key* and *reuse key*. In the former scenario each distributed aggregation is performed using new encryption keys, while in the latter scenario the key generated in the first round is used in other computations as well.



(a)



(b)

Figure 3: The average runtimes of a protocol for different encryption key sizes k ($n = 32$) and different number of participants n ($k = 128$).

Figure 3 shows the average runtime of a single round for encryption keys of different sizes and different amounts of participants. Since results of Ács, and Shi schemes are very similar, we represent them as a single line, when evaluating schemes against different encryption key sizes. Generating and distributing a set of encryption keys in the Paillier scheme is a very time consuming process. Increasing the key

size k, significantly increases computation time for the *new key* scenario, but has a negligible overhead when one key is used all the time in the *reuse key* scenario.

Despite encryption overhead, the homomorphic encryption schemes scale well. Adding new nodes linearly increases the average runtime of all homomorphic encryption schemes.

6.3.2 Comparison

The goal of these experiments is to compare performance of presented security schemes. Since different privacy mechanisms have very little impact on the overall performance, we use Laplace LDPA in all runs.

Data Preparation Overhead. For all protocols majority of their computations are local and prior to any communication with other nodes. Therefore, before comparing protocols for different scenarios, we run an experiment to evaluate time needed by each node to prepare its data before sending them to other nodes.

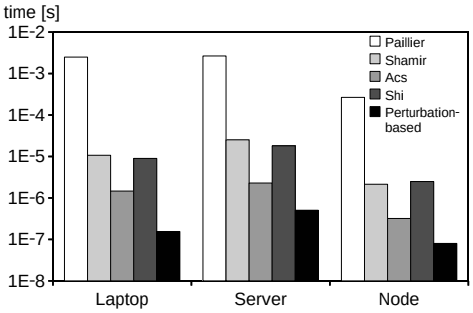


Figure 4: The average local computation time (logarithmic scale) for data preparation of different security schemes on different platforms.

Figure 4 shows the average local computation time (logarithmic scale) for data preparation of security schemes on different platforms. The runtime includes all random number generations, encryptions, and any other necessary computations. The Perturbation-based scheme outperforms others by at least 2 orders of magnitude. In this scheme each node generates at most one random number, which is a relatively easy task. Each node running Shamir, or Ács, or Shi scheme spends more time before communicating with others. However, it is still 2 orders of magnitude faster than for the Paillier scheme. Note that, we have measured only the encryption time, and skipped the time spent on generating 128-bit encryption keys.

Overall Protocol Performance. In this experiment we compare all security schemes for different numbers of nodes. Distributed noise is generated using Laplace DLPA mechanism for all protocols. **Figure 5** shows runtimes of the distributed aggregation protocols implemented in different security schemes. Note that the total runtime of the protocol includes the computation time for data preparation as well as all communication and subsequent computation time before the protocol completes.

Note that for security, nodes in both Paillier and Ács schemes reestablish their encryption keys in each round. As the number of nodes increases, both Perturbation-based and Shamir schemes do not scale well as the increasing communication cost becomes the dominant overhead.

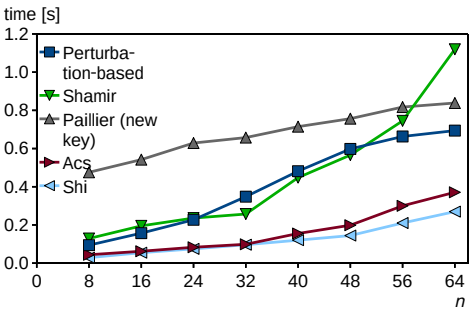


Figure 5: The average runtimes for different numbers of nodes and security schemes.

On the other hand, all homomorphic encryption schemes scale well due to their low communication costs. However, the Paillier scheme has a big computation overhead compared to others, which limits its scalability. Among encryption schemes, the Shi scheme is the fastest one. In each round of this scheme encryption keys do not have to be regenerated, and the encryption function has a relatively low complexity. Both of these factors allow computations to finish in a short time.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have described different ways of distributed data aggregation with security and privacy. Security schemes provide different levels of security with different overhead. There is no scheme that outperforms others in all settings. All evaluated privacy mechanisms ensure differential privacy, but with different overhead. The choice of privacy mechanism has relatively small impact on the performance. However, if one uses devices with limited power and computation resources, e.g., mobile devices, choosing the most efficient privacy mechanism is very important.

In future, we plan to evaluate security schemes and privacy mechanisms using devices with limited power and computation resources like smart phones. We will also evaluate new distributed privacy mechanisms that ensures approximated differential privacy and other privacy notions.

Acknowledgments. This research is supported by AFOSR under grant FA9550-12-1-0240. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

8. REFERENCES

[1] *Report of the August 2010 Multi-Agency Workshop on InfoSymbiotics/DDDAS, The Power of Dynamic Data Driven Applications Systems*. Workshop sponsored by: Air Force Office of Scientific Research and National Science Foundation.

[2] G. Ács and C. Castelluccia. I have a DREAM!: differentially private smart metering. In *Proc. of the 13th Intl Conf. on Information Hiding*, IH, pages 118–132, 2011.

[3] J. Ahrens and U. Dieter. Computer methods for sampling from gamma, beta, Poisson and bionomial distributions. *Computing*, 12:223–246, 1974.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[5] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[6] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In

Workshop on World-Sensor-Web (WSW): Mobile Device Centric Sensor Networks and Applications, 2006.

[7] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *19th USENIX Security Symposium*, August 2010.

[8] 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):20:1–20:36, June 2009.

[9] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *Proc. of the The 2nd Annual Intl Conf. on Mobile and Ubiquitous Systems: Networking and Services*, MOBIQUITOUS, pages 109–117, 2005.

[10] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Proc. of the 14th Intl Conf. on Financial Cryptography and Data Security*, FC, pages 35–50, 2010.

[11] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4:28–34, 2002.

[12] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[13] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proc. of the 4th Intl Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC, pages 119–136, 2001.

[14] I. Damgård and G. Mikkelsen. Efficient, robust and constant-round distributed rsa key generation. In D. Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 183–200, 2010.

[15] C. Dwork. Differential privacy. In *Proc. of the 33rd Intl Conf. on Automata, Languages and Programming - Volume Part II*, ICALP, pages 1–12, 2006.

[16] C. Dwork. Differential privacy: a survey of results. In *Proc. of the 5th Intl Conf. on Theory and Applications of Models of Computation*, TAMC, pages 1–19, 2008.

[17] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[18] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: privacy via distributed noise generation. In *Proc. of the 24th Annual Intl Conf. on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT, pages 486–503, 2006.

[19] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[20] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of CRYPTO 84 on Advances in Cryptology*, pages 10–18, 1985.

[21] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4):14:1–14:53, 2010.

[22] S. L. Garfinkel and M. D. Smith. Guest editors' introduction: Data surveillance. *IEEE Security & Privacy*, 4(6), 2006.

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

[24] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC*, pages 351–360, 2009.

[25] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[26] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. In *Proc. of the 12th Conf. on Topics in Cryptology*, CT-RSA, pages 313–331, 2012.

[27] J. Kang, K. Shilton, D. Estrin, J. Burke, and M. Hansen. Self-surveillance privacy. *Iowa Law Review*, 97, 2012.

[28] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD*, pages 193–204, 2011.

[29] S. Kotz, T. Kozubowski, and K. Podgórski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Progress in Mathematics Series. Birkhäuser Boston, 2001.

[30] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR Cryptology ePrint Archive*, 2008:197, 2008.

[31] G. Marsaglia and T. A. Bray. A convenient method for generating normal variables. *SIAM Review*, 6(3):260–264, 1964.

[32] G. Marsaglia and W. W. Tsang. The Ziggurat method for generating random variables. *J. Stat. Softw.*, 5(8):1–7, 10 2000.

[33] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. PEIR, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proc. of the 7th Intl Conf. on Mobile Systems, Applications, and Services*, MobiSys, 2009.

[34] T. Nishide and K. Sakurai. Distributed Paillier cryptosystem without trusted dealer. In Y. Chung and M. Yung, editors, *Information Security Applications*, volume 6513 of *Lecture Notes in Computer Science*, pages 44–60, 2011.

[35] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[36] T. B. Pedersen, Y. Saygın, and E. Savaş. Secret Sharing vs. Encryption-based Techniques For Privacy Preserving Data Mining. In *Proc. of UNECE/Eurostat Work Session on SDC*, 2007.

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

[38] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[39] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[40] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

[41] K. Shilton. Four billion little brothers?: privacy, mobile phones, and ubiquitous data collection. *CACM*, 52:48–53, 2009.

[42] J. Vaidya and C. Clifton. Privacy-preserving data mining: Why, how, and when. *IEEE Security & Privacy*, 2(6):19–27, 2004.

[43] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[44] A. C. Yao. How to generate and exchange secrets. In *Proc. of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.