Private Data Aggregation Over Selected Subsets of Users

Amit Datta[1], Marc Joye[2], and Nadia Fawaz

[1] Snap Inc., Santa Monica, CA, USA
amit.datta@snap.com
[2] OneSpan, Brussels, Belgium
marc.joye@onespan.com

Abstract. Aggregator-oblivious (AO) encryption allows the computation of aggregate statistics over sensitive data by an untrusted party, called aggregator. In this paper, we focus on exact aggregation, wherein the aggregator obtains the exact sum over the participants. We identify three major drawbacks for existing exact AO encryption schemes —no support for dynamic groups of users, the requirement of additional trusted third parties, and the need of additional communication channels among users. We present privacy-preserving aggregation schemes that do not require any third-party or communication channels among users and are exact and dynamic. The performance of our schemes is evaluated by presenting running times.

Keywords: Data aggregation · Privacy · Aggregator obliviousness

1 Introduction

Data aggregation is the process of compiling data from multiple entities or databases into one location with the intent of preparing a combined dataset. The entity performing the aggregation is referred to as the *aggregator*. Data aggregation has many applications in surveys, e-voting, data analytics, etc. However, a major concern with data aggregation is that the data handled in most of these settings are sensitive, or confidential, or may be correlated with sensitive information. This gives rise to data privacy concerns, particularly when the aggregator is *untrusted*.

Consider the simple example of a salary survey, where an aggregator is interested in computing the average salary of a group of users. A user may consider her individual salary as private information that she does not wish to reveal to the aggregator. However, if an aggregation protocol allows the aggregator to compute the average salary without requiring access to any individual's salaries in the clear, then the user may be willing to participate in the protocol.

Similarly, in the health sector, patient health information is protected by legal frameworks, such as Health Insurance Portability and Accountability Act (HIPAA), which regulates the use and disclosure of protected health information, including medical records and payment history, held by covered entities like

health-care providers and health insurances. A network of healthcare providers or hospitals may be interested in computing aggregate statistics over their joint populations of patients for research purposes, or for disease prevention and control. However legal frameworks may not allow them to share individual patient records in the clear. In such a scenario, the network would like to leverage a privacy-preserving aggregation protocol which would allow them to compute the aggregate statistics without access to data in the clear. Many businesses also leverage user data analytics, for operations management, marketing and sales campaigns, or to provide personalized services to their customer base, especially in the age of digital economy. Businesses that operate in different regions of the world, such as the USA and the European Union (EU), may face different privacy regulations in each location. Such businesses may see their ability to transfer user data across their different corporate locations or with their global partners restricted, as illustrated by the decision by the EU Court of Justice to invalidate the EU-US Safe Habor agreement, on which thousands of companies had relied for the transatlantic transfer of user data since 2000 [5]. Limitations on data transfer in turn generates the need for new privacy-preserving aggregation techniques to perform aggregate statistics over user data across locations, without transferring individual user data in the clear.

Aggregator Obliviousness. The notion of *aggregator obliviousness* (AO) [17] was introduced to allow an aggregator to receive encrypted data from users and compute the aggregate and *nothing else*. With an AO encryption scheme, the aggregator gains no additional knowledge other than what is evident from the aggregate itself. Formal definitions for AO are given in Section 2.

Suppose that there is a population of n users, denoted by $\{1, \ldots, n\}$, as well as a designated entity called the aggregator. Let $x_{i,\tau}$ denote the private data of user i, where $i \in \mathcal{S}$ for a subset $\mathcal{S} \subseteq \{1, \ldots, n\}$ of users, and with tag τ, where the tag is an identifier of the aggregation (τ may for example indicate the time period at which the aggregate is computed). The goal of the aggregator is to compute the aggregate value

$$\Sigma_\tau = \sum_{i \in \mathcal{S}} x_{i,\tau}$$

in an *oblivious way*, that is, without having access to the private data $x_{i,\tau}$ in the clear. For simplicity, we focus on the aggregate sum throughout the paper. Extensions to operations and statistics beyond the simple sum are presented in Section 5.

Certain aggregation schemes make use of additional communication channels among users to perform the aggregation. These channels are used to exchange information among users and can enhance schemes. However, such channels may not exist between users, and even when they do exist, such protocols have very high communication overheads. So, a protocol would ideally not make use of communication channels among users.

Several current schemes require the presence of one or more additional *third parties* to carry out the protocol. These third-parties are assumed to be non-colluding with the aggregator and their participation often adds desirable properties (like dynamism or fault tolerance) to the protocol. However, such third parties are often difficult to find in practice, thereby rendering such schemes less practical. For such settings, it is desirable to develop a scheme which does not require an additional third-party.

In many existing AO encryption schemes, the group of users that are aggregated is static. The scheme is setup for a given set of users and if any user joins or leaves, fresh keys need to be distributed to all the users. For many applications, it is desirable to have a *dynamic* scheme, which allows users to easily join or leave a group. With a dynamic scheme, the aggregator can carry out the aggregation over any subset or superset of users without much hassle; in particular, without having to redistribute keys to every user involved in the computation.

The lack of fault tolerance is another drawback present in many aggregation schemes. An aggregation scheme is said *fault-tolerant* when the aggregator is still able to compute an aggregate even if one or several users fail to report their share.

From the above discussion, it turns out that an ideal AO encryption scheme should be one where the users send their encrypted share in a single communication step and, at the same time, must enjoy the properties of being dynamic and fault-tolerant—without additional non-colluding third parties. No such scheme does exist as far are *exact* aggregates are concerned. This is not surprising as the above sought-after features are incompatible all together. In particular, such an exact AO encryption cannot be fault tolerant. Indeed, when there are no extra parties involved in the protocol, the users have to send their encrypted contributions directly to the aggregator. Hence, if the scheme is supposed to be fault-tolerant, the aggregator should be able to compute the aggregate sum in the clear over all participating users but also over all but one participating users, thereby obtaining the private data of the victim user by subtracting the two aggregate sums.

Our Contributions. This paper considers the weaker notion of *selective* fault-tolerance. In this setting, the subset of users over which the aggregate is to be computed can be dynamically chosen but must be known to the users beforehand. A useful application is the ability to leave out persistently failing users by making use of the dynamic nature of the scheme.

More specifically, we present two dynamic AO encryption schemes, which do not require any third party or communication among users. In these schemes, each user can participate in the protocol with the knowledge of the other users' identities. Just by knowing the identities, a user can derive a key to perform the encryption. This key has special properties that allow the aggregator to decrypt the aggregate when all users in a selected subset encrypt with their corresponding keys. We implement the schemes and evaluate their performance.

Related Work. A number of candidates for AO encryption schemes have been proposed, including [16,17,8,2,14,13,7,1,11,15]. However most of them are not dynamic. Among the ones that can support dynamic joins and leaves, [7] requires n^2 messages exchanged among users at each time period, [1,14] need each user to store n keys, which can be impractical for a large set of users, and [11,15,8,2] assume the presence of one or more third parties. Schemes like [16,17,11,1] provide a different notion of privacy: *Differential Privacy* (DP), which makes the final aggregate noisy.

In this paper, we focus on exact aggregation, without addition of any noise. However, our schemes can be modified to return noisy sums, for instance through differential-privacy techniques as was done, e.g., in [17]. Jawurek *et al.* [12] provide a fantastic survey for different techniques used for privacy-preserving aggregation.

2 Definitions

In this section, we formally define relevant notions for privacy-preserving aggregation. We first formally define Aggregator Obliviousness and then briefly describe the scheme by Shi *et al.*

2.1 Aggregator-oblivious encryption

The definition of *aggregator-oblivious encryption* was introduced in [17]. Here, we extend their definition to cover a broader class of encryption schemes and use cases.

Definition 1. *An* aggregator-oblivious encryption scheme *is a tuple of algorithms,* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{AggrDec})$*, defined as:*

$\mathsf{Setup}(1^\kappa)$ *On input a security parameter* κ*, the* Setup *algorithm generates the system parameters* param*, a master secret key* msk*, and the aggregation key* sk_0*. Parameters* param *are made public and key* sk_0 *is given to the aggregator.*

$\mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, i)$ *Given a user's identifier* i*, the* KeyGen *algorithm produces the private key* sk_i *for user* i *as* $\mathsf{sk}_i = \mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, i)$*.*

$\mathsf{Enc}(\mathsf{param}, \mathsf{sk}_i, \tau, \mathcal{S}_\tau, x_{i,\tau})$ *Given the private input* $x_{i,\tau}$ *with tag* τ *of user* $i \in \mathcal{S}_\tau$ *and the private key* sk_i*, user* i *applies this algorithm to produce the ciphertext* $c_{i,\tau} = \mathsf{Enc}(\mathsf{param}, \mathsf{sk}_i, \tau, \mathcal{S}_\tau, x_{i,\tau})$*.*

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, \tau, \mathcal{S}_\tau, \{c_{i,\tau}\}_{i\in\mathcal{S}_\tau})$ *Upon receiving the ciphertexts* $c_{i,\tau}$ $\forall i \in \mathcal{S}_\tau$ *associated with tag* τ*, the aggregator obtains the aggregate value*

$$\Sigma_\tau = \sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$$

as $\Sigma_\tau = \mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, \tau, \mathcal{S}_\tau, \{c_{i,\tau}\}_{i\in\mathcal{S}_\tau})$ *using the aggregation key* sk_0*.*

Remark 1. Three modifications were made to the original definition:

1. In the original definition, the aggregation is always computed over the entire set of users: $\mathcal{S}_\tau = \{1, \ldots, n\}$. This set is fixed and is part of the system parameters. In our case, we allow for subsets $\mathcal{S}_\tau$ of users that may change depending on the value of τ.
2. The setup algorithm is now divided in two sub-algorithms: the Setup algorithm itself that outputs the system parameters and the aggregation key and the KeyGen algorithm that returns the private keys for each user. The original definition corresponds to the case where the master secret key is the vector containing all user's private keys, $\mathsf{msk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, and the KeyGen algorithm simply returns the i-th component of msk as the private key of user i.
3. Finally, in the original definition, tag τ explicitly refers to a time period. We use the more generic term of tag, which serves as a unique identifier for the aggregation instance.

2.2 Aggregator obliviousness

The security notion of *aggregator obliviousness* (AO) requires that the aggregator cannot learn, *for each tag* τ, anything more than the aggregate value Σ_τ from the individual encrypted values. If there are corrupted users (i.e., users sharing their private information with the aggregator), the notion requires that the aggregator gets no additional information about the private values of the honest users beyond what is evident from the final aggregate value and the private values of the corrupted users. Furthermore, in our setting, we assume that each user encrypts only *one* value for a given tag.

More formally, AO security is defined by the following game between a challenger and an attacker.

Setup The challenger runs the Setup algorithm and gets param, msk and sk_0. It also runs KeyGen to obtain the encryption key sk_i of each user i. The challenger gives param to the attacker.

Queries In the first phase, the attacker can submit queries that are answered by the challenger. The attacker can make two types of queries:

1. Encryption queries: The attacker submits $(i, \tau, \mathcal{S}_\tau, x_{i,\tau})$ for a pair (i, τ) with $i \in \mathcal{S}_\tau$ and gets back the encryption of $x_{i,\tau}$ with tag τ under key sk_i;
2. Compromise queries: The attacker submits i and receives the private key sk_i of user i; if $i = 0$, the attacker receives the aggregation key sk_0.

Challenge The attacker chooses a target tag $\tau^\star$. Let $\mathcal{U}^\star \subseteq \{1, \ldots, n\}$ be the whole set of users for which, at the end of the game, no encryption queries associated to tag $\tau^\star$ have been made and no compromise queries have been made. The attacker chooses a subset $\mathcal{S}_{\tau^\star} \subseteq \mathcal{U}^\star$ and two different series of triples

$$\langle (i, \tau^\star, x^{(0)}_{i,\tau^\star}) \rangle_{i \in \mathcal{S}_{\tau^\star}} \quad \text{and} \quad \langle (i, \tau^\star, x^{(1)}_{i,\tau^\star}) \rangle_{i \in \mathcal{S}_{\tau^\star}} ,$$

that are given to the challenger. Further, if the aggregator capability sk_0 is compromised at the end of the game and $\mathcal{S}_{\tau^\star} = \mathcal{U}^\star$, it is required that

$$\sum_{i \in \mathcal{S}_{\tau^\star}} x_{i,\tau^\star}^{(0)} = \sum_{i \in \mathcal{S}_{\tau^\star}} x_{i,\tau^\star}^{(1)} \ .$$

Guess The challenger chooses at random a bit $b \in \{0,1\}$ and returns the encryption of $\langle x_{i,\tau^\star}^{(b)} \rangle_{i \in \mathcal{S}_{\tau^\star}}$ to the attacker.

More queries In the second phase, the attacker can make more encryption queries and compromise queries. Note that since $\mathcal{S}_{\tau^\star} \subseteq \mathcal{U}^\star$, the attacker cannot submit an encryption query $(i, \tau^\star, \cdot, \cdot)$ with $i \in \mathcal{S}_{\tau^\star}$ or a compromise query i with $i \in \mathcal{S}_{\tau^\star}$.

Outcome At the end of the game, the attacker outputs a bit b' and wins the game if and only if $b' = b$ (i.e., if it correctly guessed the bit b). As usual, $\mathcal{A}$'s advantage is defined to be

$$\mathbf{Adv}^{\mathsf{AO}}(\mathcal{A}) := 2 \left| \Pr[b' = b] - 1/2 \right|$$

where the probability is taken over the random coins of the game according to the distribution induced by Setup and over the random coins of the attacker.

Definition 2. *An encryption scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{AggrDec})$ *is said to meet the* AO security notion *if no probabilistic polynomial-time attacker* $\mathcal{A}$ *can win the above* AO *security game with an advantage* $\mathbf{Adv}^{\mathsf{AO}}(\mathcal{A})$ *that is non-negligible in the security parameter.*

2.3 Example scheme

As an illustration, we briefly review the scheme proposed by Shi *et al.* [17] for achieving aggregator-obliviousness. This notion is met under the DDH assumption [10,4] in the random oracle model. This scheme will serve as a building block for our final scheme.

$\mathsf{Setup}(1^\kappa)$ On input a security parameter κ, a trusted dealer generates a group $\mathbb{G} = \langle g \rangle$ of prime order q for which the DDH assumption holds. It also defines a cryptographic hash function $H \colon \{0,1\}^* \to \mathbb{G}$, viewed as a random oracle. Finally, it generates n random elements $s_1, \ldots, s_n \in \mathbb{Z}/q\mathbb{Z}$ and lets $s_0 = -\sum_{i=1}^{n} s_i \pmod{q}$, where n denotes the total number of users. The system parameters are $\mathsf{param} = \{\mathbb{G}, g, q, H\}$, the master secret key is $\mathsf{msk} = (s_1, \ldots, s_n)$, and the aggregation key is $\mathsf{sk}_0 = s_0$. Parameters param are made public and sk_0 is given to the aggregator.

$\mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, i)$ On input user's identifier i, the KeyGen algorithm returns the private key $\mathsf{sk}_i = s_i$ for user i, where s_i is the i-th component of msk.

$\mathsf{Enc}(\mathsf{param}, s_i, \tau, \{1, \ldots, n\}, x_{i,\tau})$ User $i \in \{1, \ldots, n\}$ encrypts a value $x_{i,\tau}$ with tag τ using the private key s_i to get the ciphertext

$$c_{i,\tau} = g^{x_{i,\tau}} H(\tau)^{s_i} \ .$$

$\mathsf{AggrDec}(\mathsf{param}, s_0, \tau, \{1,\ldots,n\}, \{c_{i,\tau}\}_{1\leqslant i\leqslant n})$ Upon receiving all the $c_{i,\tau}$'s (with $i \in \{1,\ldots,n\}$) associated with tag τ, the aggregator first computes

$$V_\tau = H(\tau)^{s_0} \prod_{1\leqslant i\leqslant n} c_{i,\tau}$$

and then obtains the aggregate value $\Sigma_\tau = \sum_{1\leqslant i\leqslant n} x_{i,\tau} \pmod q$ by computing the discrete logarithm of V_τ w.r.t. basis g.

The scheme works because the aggregation key is defined as $s_0 = -\sum_{1\leqslant i\leqslant n} s_i$. The aggregator is so able to remove the masking expression $H(\tau)^{\sum_{1\leqslant i\leqslant n} s_i} = H(\tau)^{-s_0}$ by using the aggregation key s_0. Indeed, the aggregation step computes $V_\tau = H(\tau)^{s_0} g^{\sum_{1\leqslant i\leqslant n} x_{i,\tau}} H(\tau)^{\sum_{1\leqslant i\leqslant n} s_i} = g^{\Sigma_\tau}$, the discrete logarithm of which yields Σ_τ.

3 Dynamic AO Encryption

In this section, we describe a dynamic aggregation scheme. With a dynamic scheme, the aggregator can carry out the aggregation over any subset or superset of users without needing to perform the full Setup. To aggregate over any subset of users, the aggregator only needs to convey information about the composition of the subset to all the users. To add a new user to the set $\mathcal{S}_\tau$, only KeyGen needs to be carried out for that user, after which the aggregation can be carried out as before.

To devise a dynamic AO scheme, we begin by building on top of the scheme due to Shi *et al.* As seen in Section 2.3, the key trick behind that scheme is $\mathsf{sk}_0 = -\sum_i \mathsf{sk}_i$. Shi *et al.* achieve this by having a trusted dealer generate and send keys satisfying this property to the users and the aggregator. To apply Shi *et al.*'s scheme, as it is, on a subset of users, the trusted dealer has to regenerate keys for each subset and send them over to the users. Thus, the scheme would require multiple interactions with the trusted dealer. In settings where the key is burned onto a hardware component (like set-top boxes), such a scheme becomes inapplicable.

In the proposed schemes, we give users the ability to compute such keys on their own. A trusted dealer provides all users and the aggregator with an initial secret key sk_i. If a user knows the identities of the other users are in the desired subset $\mathcal{S}_\tau$, then she can compute the subset-key $s_{i,\tau}$ for that subset such that $s_{0,\tau} = -\sum_{i\in\mathcal{S}_\tau} s_{i,\tau}$. We use a combination of pairings and identity-based encryption so that users can compute the subset key from only the identities of the other users in the given subset. There is no need to redistribute any keys. Kursawe *et al.* [14] use a similar technique, but they combine pairings and public-key encryption. The primary advantage in our schemes is that there is no need for a public-key infrastructure and users do not need to remember or verify public keys. Thus, we obtain more versatile aggregation schemes over dynamic subsets of users.

3.1 Key ingredient

To provide the ability to users to compute their own subset keys, we use identity-based encryption (IBE) and bilinear type-1 pairings. In an IBE scheme, the public key of a user is a unique identifying information about the user. This allows other users to send encrypted values with just the knowledge of the identity of the recipient. A bilinear type-1 pairing is a symmetric map $e\colon \hat{\mathbb{G}} \times \hat{\mathbb{G}} \to \hat{\mathbb{G}}_T$, where $\hat{\mathbb{G}}, \hat{\mathbb{G}}_T$ are cyclic groups of order p and the function e satisfies bilinearity: $e(\hat{g}^a, \hat{g}^b) = e(\hat{g}, \hat{g})^{ab}$, where $\langle \hat{g} \rangle = \hat{\mathbb{G}}$ and $a, b \in \mathbb{Z}/p\mathbb{Z}$.

In our setting, we assume that i is the unique identifier for user i. The trusted dealer sends the secret key $\mathsf{sk}_i = J(i)^{\mathsf{msk}}$ to user i, where msk is the master secret key of the trusted dealer and $J : \mathbb{Z} \to \hat{\mathbb{G}}$ is a publicly known cryptographic hash function. Given the identities of users in a subset $\mathcal{S}_\tau$, a user $i \in \mathcal{S}_\tau$ derives on her own the corresponding subset key from $\mathcal{S}_\tau$ and sk_i as

$$s_{i,\tau} = \sum_{\substack{k \in \mathcal{S}_\tau \cup \{0\} \\ k < i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k > i}} \mathfrak{H}(K_{i,k}) \pmod{T}$$

with

$$K_{i,k} = e(J(k), \mathsf{sk}_i)$$

and $\mathfrak{H} : \hat{\mathbb{G}}_T \to \mathbb{Z}/T\mathbb{Z}$ is hash function (typically, $T = q$ a prime, or $T = 2^\ell$).

We assume that the aggregator's identifier is $i = 0$. For a subset $\mathcal{S}_\tau$, the aggregator also derives the matching subset key $s_{0,\tau}$ in a similar manner, which results in

$$\begin{aligned} s_{0,\tau} &= \sum_{\substack{k \in \mathcal{S}_\tau \cup \{0\} \\ k < 0}} \mathfrak{H}(K_{0,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k > 0}} \mathfrak{H}(K_{0,k}) \pmod{T} \\ &= -\sum_{k \in \mathcal{S}_\tau} \mathfrak{H}(K_{0,k}) \pmod{T}. \end{aligned}$$

Property. We now show that the above construction satisfies the following property

$$\sum_{i \in \mathcal{S}_\tau} s_{i,\tau} = -s_{0,\tau} \pmod{T}$$

By construction, variable $K_{i,k}$ is symmetric. Indeed, we have

$$\begin{aligned} K_{i,k} &= e(J(k), \mathsf{sk}_i) = e(\mathsf{sk}_i, J(k)) = e(J(i)^{\mathsf{msk}}, J(k)) \\ &= e(J(i), J(k)^{\mathsf{msk}}) = e(J(i), \mathsf{sk}_k) = K_{k,i} \ . \end{aligned}$$

In a way similar to [9, Proposition 1], letting $m_\tau = \min_{i \in \mathcal{S}_\tau} i$ and $M_\tau = \max_{i \in \mathcal{S}_\tau} i$, it is then readily verified that:

$$\begin{aligned}
\sum_{i \in \mathcal{S}_\tau} s_{i,\tau} &= \sum_{i \in \mathcal{S}_\tau} \Bigg[\sum_{\substack{k \in \mathcal{S}_\tau \cup \{0\} \\ k<i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k>i}} \mathfrak{H}(K_{i,k}) \Bigg] \\
&= \sum_{i \in \mathcal{S}_\tau} \Bigg[\mathfrak{H}(K_{i,0}) + \sum_{\substack{k \in \mathcal{S}_\tau \\ k<i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k>i}} \mathfrak{H}(K_{k,i}) \Bigg] \\
&= -s_{0,\tau} + \sum_{\substack{i \in \mathcal{S}_\tau \\ i \neq m_\tau}} \sum_{\substack{k \in \mathcal{S}_\tau \\ k<i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{i \in \mathcal{S}_\tau \\ i \neq M_\tau}} \sum_{\substack{k \in \mathcal{S}_\tau \\ k>i}} \mathfrak{H}(K_{k,i}) \\
&= -s_{0,\tau} + \sum_{\substack{i \in \mathcal{S}_\tau \\ i \neq m_\tau}} \sum_{\substack{k \in \mathcal{S}_\tau \\ k<i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{k \in \mathcal{S} \\ k \neq m_\tau}} \sum_{\substack{i \in \mathcal{S}_\tau \\ i<k}} \mathfrak{H}(K_{k,i}) \\
&= -s_{0,\tau} \pmod{T} \ .
\end{aligned}$$

This was the key property used by Shi *et al.*'s scheme, but instead of having a key dealer generate keys which satisfy this property, we now have a mechanism to generate keys having this characteristic. With this building block in place, we are now ready to present our new schemes.

3.2 Proposed schemes

We detail the description of our schemes using the building block we describe in the previous section. The Enc phase now has an additional step wherein the subset keys are computed by users.

In both schemes, user i receives an initial secret key sk_i, while the aggregation key sk_0 is given to the aggregator. Given the identities of the users in the subset and the secret key, each user computes the subset key $s_{i,\tau}$, which satisfies the property $s_{0,\tau} = -\sum_{i \in \mathcal{S}_\tau} s_{i,\tau}$, where $s_{0,\tau}$ is the aggregator's subset key. With the subset key, each user encrypts her value and sends the ciphertext to the aggregator. Upon receiving ciphertexts from all the users in the subset, the aggregator uses the subset key $s_{0,\tau}$ to recover the aggregate value. The correctness of the schemes follows from the correctness of the building block since $s_{0,\tau} = -\sum_{i \in \mathcal{S}_\tau} s_{i,\tau}$. Details of both schemes are provided in the following sections.

Scheme I.

$\mathsf{Setup}(1^\kappa)$ On input a security parameter κ, a trusted dealer generates the system parameters $\mathsf{param} = \{\mathbb{G}, g, q, \hat{\mathbb{G}}, \hat{\mathbb{G}}_T, e, H, \mathfrak{H}, J\}$ where $\mathbb{G} = \langle g \rangle$ is a group of order q, $\hat{\mathbb{G}}$ and $\hat{\mathbb{G}}_T$ are two groups of order p with a bilinear type-1 pairing $e \colon \hat{\mathbb{G}} \times \hat{\mathbb{G}} \to \hat{\mathbb{G}}_T$, and $H \colon \{0,1\}^* \to \mathbb{G}$, $\mathfrak{H} : \hat{\mathbb{G}}_T \to \mathbb{Z}/q\mathbb{Z}$ and $J \colon \mathbb{Z} \to \hat{\mathbb{G}}$ are cryptographic hash functions. The trusted dealer also generates a master

secret key $\mathsf{msk} \in \mathbb{Z}/p\mathbb{Z}$ and computes the aggregation key $\mathsf{sk}_0 = [J(0)]^{\mathsf{msk}} \in \hat{\mathbb{G}}$. Parameters param are made public and key sk_0 is given to the aggregator.

$\mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, i)$ Given user's identifier i, the KeyGen algorithm returns $\mathsf{sk}_i = [J(i)]^{\mathsf{msk}} \in \hat{\mathbb{G}}$, where msk is the master secret key of the trusted dealer.

$\mathsf{Enc}(\mathsf{param}, \mathsf{sk}_i, \tau, \mathcal{S}_\tau, x_{i,\tau})$ $\mathcal{S}_\tau$ represents the subset of users among whom the aggregation is to be computed. For a private input $x_{i,\tau} \in \mathbb{Z}/q\mathbb{Z}$ with tag τ, provided that $i \in \mathcal{S}_\tau$, user i computes the ciphertext

$$c_{i,\tau} = g^{x_{i,\tau}} H(\tau)^{s_{i,\tau}} \ \ (\in \mathbb{G})$$

where

$$s_{i,\tau} = \sum_{\substack{k \in \mathcal{S}_\tau \cup \{0\} \\ k<i}} \mathfrak{H}(K_{i,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k>i}} \mathfrak{H}(K_{i,k}) \pmod q$$

with $K_{i,k} = e(J(k), \mathsf{sk}_i) \in \hat{\mathbb{G}}_T$. The user sends $c_{i,\tau}$ to the aggregator through *any* channel.

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, \tau, \mathcal{S}_\tau, \{c_{i,\tau}\}_{i \in \mathcal{S}_\tau})$ Upon receiving all the ciphertexts $c_{i,\tau}$ with $i \in \mathcal{S}_\tau$, all with tag τ, the aggregator computes in $\mathbb{G}$

$$V_\tau := H(\tau)^{s_{0,\tau}} \cdot \prod_{i \in \mathcal{S}_\tau} c_{i,\tau} = g^{\sum_{i \in \mathcal{S}_\tau} x_{i,\tau}}$$

where $s_{0,\tau} = -\sum_{k \in \mathcal{S}_\tau} \mathfrak{H}(K_{0,k}) \pmod q$ with $K_{0,k} = e(J(k), \mathsf{sk}_0)$. The aggregator obtains the sum $\Sigma_\tau := \sum_{i \in \mathcal{S}_\tau} x_{i,\tau} \pmod q$ by computing the discrete logarithm of V_τ with respect to g.

In [3], the authors show how the use of two hash functions lead to a much tighter security reduction. The same observation readily applies here.

Scheme II.

$\mathsf{Setup}(1^\kappa)$ On input a security parameter κ, a trusted dealer generates the system parameters $\mathsf{param} = \{\hat{\mathbb{G}}, \hat{\mathbb{G}}_T, e, \ell, \{\mathfrak{H}_\tau\}_\tau, J\}$ where $\hat{\mathbb{G}}$ and $\hat{\mathbb{G}}_T$ are two groups of order p with a bilinear type-1 pairing $e : \hat{\mathbb{G}} \times \hat{\mathbb{G}} \to \hat{\mathbb{G}}_T$, ℓ is a length upper-bounding the bit-length of the private inputs and their sums, $\{\mathfrak{H}_\tau\}_\tau \colon \hat{\mathbb{G}}_T \to \mathbb{Z}/2^\ell\mathbb{Z}$ is a family of keyed cryptographic hash functions, and $J \colon \mathbb{Z} \to \hat{\mathbb{G}}$ is a cryptographic hash function. The trusted dealer also generates a master secret key $\mathsf{msk} \in \mathbb{Z}/p\mathbb{Z}$ and computes the aggregation key $\mathsf{sk}_0 = [J(0)]^{\mathsf{msk}} \in \hat{\mathbb{G}}$. Parameters param are made public and key sk_0 is given to the aggregator.

$\mathsf{KeyGen}(\mathsf{param}, \mathsf{msk}, i)$ Given user's identifier i, the KeyGen algorithm returns $\mathsf{sk}_i = [J(i)]^{\mathsf{msk}} \in \hat{\mathbb{G}}$, where msk is the master secret key of the trusted dealer.

$\mathsf{Enc}(\mathsf{param}, \mathsf{sk}_i, \tau, \mathcal{S}_\tau, x_{i,\tau})$ $\mathcal{S}_\tau$ represents the subset of users among whom the aggregation is to be computed. For a private input $x_{i,\tau} \in \mathbb{Z}/2^\ell\mathbb{Z}$ with tag τ, provided that $i \in \mathcal{S}_\tau$, user i forms the ciphertext

$$c_{i,\tau} = x_{i,\tau} + s_{i,\tau} \ \ (\in \mathbb{Z}/2^\ell\mathbb{Z})$$

where

$$s_{i,\tau} = \sum_{\substack{k \in \mathcal{S}_\tau \cup \{0\} \\ k<i}} \mathfrak{H}_\tau(K_{i,k}) - \sum_{\substack{k \in \mathcal{S}_\tau \\ k>i}} \mathfrak{H}_\tau(K_{i,k}) \pmod{2^\ell}$$

with $K_{i,k} = e(J(k), \mathsf{sk}_i) \in \hat{\mathbb{G}}_T$. The user sends $c_{i,\tau}$ to the aggregator through *any* channel.

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, \tau, \mathcal{S}_\tau, \{c_{i,\tau}\}_{i \in \mathcal{S}_\tau})$ Upon receiving all the ciphertexts $c_{i,\tau}$ with $i \in \mathcal{S}_\tau$, all with tag τ, the aggregator computes in $\mathbb{Z}/2^\ell\mathbb{Z}$

$$\Sigma_\tau := s_{0,\tau} + \sum_{i \in \mathcal{S}_\tau} c_{i,\tau} = \sum_{i \in \mathcal{S}_\tau} x_{i,\tau} \pmod{2^\ell} \ .$$

4 Performance

In this section, we present performance results from implementations of both our schemes. The principal computational difference between our schemes and prior schemes is the additional time required for the computation of the subset key described in Section 3.1. This is the most expensive step computationally as it requires a user to compute n pairings, where n is the subset size.

For Scheme I, this computation of the subset key is required only when subsets change. As long as aggregates are computed over the same subsets, in spite of the aggregations being with different tags, there is no need to recompute subset-keys. However, for Scheme II, the subset keys cannot be reused in spite of having the same subsets. For every aggregation with a different tag τ, users must compute fresh subset keys. But if user i has access to the components $K_{i,k}$ for every $k \in \mathcal{S}_\tau$ (e.g., by storing them), this computation is very fast as it only involves evaluations of keyed hash functions.

For the second step of the encryption (i.e., the encryption itself) or for the decryption, Scheme II is far more efficient than Scheme I. This is expected since Scheme II does not require costly operations like exponentiations and discrete-log computations, instead relies solely on modular additions. The computational efficiency is clear from Table 1 of timing comparisons.

All computations described in this section were performed on a 2.6 GHz Intel Core i7 processor with 8 GB 1600 MHz DDR3 RAM.

Subset keys

For the computation of subset keys, we use Tate pairing with an embedding degree of 2 on a supersingular curve to compute pairings required for subset-key computation. The GF(p) elliptic curve is assumed to be of the form $y^2 = x^3 + Ax + B \mod p$, where $A = -3$ and $B = 0$. A low embedding degree (2 in our case) has an adverse impact on performance, but timings can be improved by using an embedding degree of 3 as shown by Teruya *et al.* [18].

To compute the subset key, a user has to compute a pairing for every other user in the subset. Thus, the time required for computing the subset key is

linear in the size of the subset. The rate of this linear increase (slope) depends on how big the pairings are. We compare the rate of increase in computation times using pairings of size 512, 1179, and 1536 bits. The slope is steeper as the size of pairing increases. With a 512-bit pairing, it takes nearly 1.5 seconds to compute the subset key when there are 400 users in a subset. Thus, depending on how often users have to compute the subset keys, the size of each subset can be decided. For example, if users are expected to report values once every few minutes, subsets can easily have about 16,000 users. However, if users are to report values every few seconds, then subsets need to be much smaller (less than 300). In Figure 1, we show how the computation time changes with the number of users and the size of the pairing.

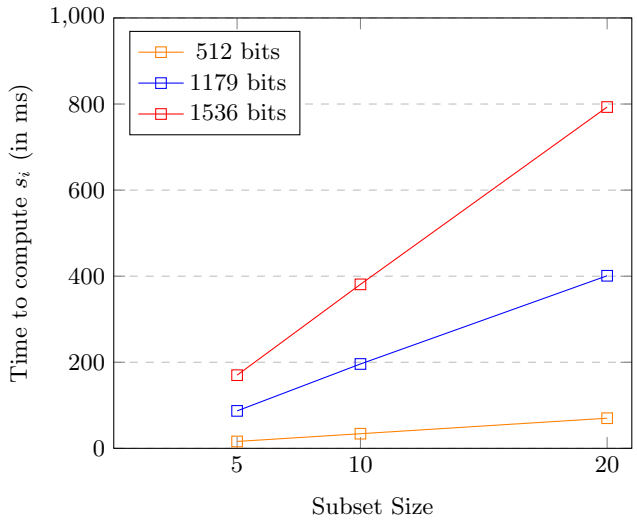

Fig. 1. Variation in subset key computation time with subset size and size of pairing. For a given pairing size, the time is linear in the subset size.

Schemes I & II

We present the computation times of both our schemes in Table 1. The running times are with a margin of error at 95% confidence, computed with 100 samples for 2^{12} users, for both the schemes. For the first scheme, we use a 200-bit curve similar to the one used in [3]. The timings presented here do not include the times required to compute the subset keys; we discuss them in the previous section. The computation time of AggrDec does not include the time to compute the subset key of the aggregator $s_{0,\tau}$. The hashing of tags ($H(\tau)$) is performed using SHA-512.

Table 1. Comparison of running times for Schemes I & II

	Enc		AggrDec	Result Recovery
	$H(\tau)$	$c_{i,\tau}$		
Scheme I	0.22(±0.01)	2.40(±0.06)	27.1(±0.38)	112(±11.73)
Scheme II	–	0.003(±0.0)	0.15(±0.01)	–

5 Extensions and Applications

The AO encryption schemes proposed in Section 3 can be extended to allow the aggregator to compute other statistics beyond the simple sum $\sum_i x_{i,\tau}$. In this section, we present extensions and example applications of dynamic AO encryption. These applications have in common the following: an aggregate value needs to be computed over a selected subset of a population, while not releasing individual user values in the clear. A major advantage of the schemes proposed in this paper is that they allow to compute aggregate values over any selected subsets of the population, without the need for redistributing keys for every new subset of selected users considered.

Hereafter, we will recurrently use the application example of surveys, polls, or electronic voting. In this case, the subset of users $\mathcal{S}_\tau$ on which the aggregate is computed is the set of users who are eligible to take the survey or poll, or to vote in the election. The tag τ is a unique identifier associated with each survey or election, that determines the subset of users over which the aggregate will be computed. For instance, the subset $\mathcal{S}_\tau$ can have associated eligibility criteria, such as demographic information (citizenship, age, location, ...). The individual user value $x_{i,\tau}$ represents the vote of user i, or her answers to the poll or to the survey. The dynamic schemes proposed in thispaper allow to conduct multiple surveys or elections over different subsets of the population without the need to redistribute keys each time.

5.1 Weighted sums

The proposed schemes can be extended to support weighted sums $\sum_i a_i\, x_{i,\tau}$ for some predetermined individual weights $a_i \in \mathbb{Z}$ known to each user. Particular cases of weighted sums include the simple sum $\sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$ where $a_i = 1$ for all $i \in \mathcal{S}_\tau$, and the sample mean $\bar{X}_\tau := \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$ where $a_i = \frac{1}{|\mathcal{S}_\tau|}$ for all $i \in \mathcal{S}_\tau$.

Application. Consider the simple example of a salary survey, where an aggregator is interested in computing the average salary of a subset $\mathcal{S}_\tau$ of the population. User $i \in \mathcal{S}_\tau$ may consider her individual salary $x_{i,\tau}$ as private sensitive information that she does not wish to disclose in the clear. However, she may be willing to take part in the survey provided only the average salary $\Sigma_\tau = \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$

over all users in $\mathcal{S}_\tau$ is released in the clear, but not the individual user values, which is the promise of AO encryption.

5.2 k-th order moments

If the aggregator wishes to evaluate the k-th order sample moment $m_{k,\tau} := \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau}(x_{i,\tau})^k$, each user encrypts $(x_{i,\tau})^k$. For instance, if the aggregator wishes to evaluate the sample mean (1st order moment) and the sample 2nd order moment, each user has to encrypt both $x_{i,\tau}$ and $(x_{i,\tau})^2$. Applying the basic scheme, the aggregator then gets $\sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$ and $\sum_{i\in\mathcal{S}_\tau}(x_{i,\tau})^2$, from which the sample mean and 2nd order moment are obtained as

$$\bar{X}_\tau := \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau} x_{i,\tau} \quad \text{and} \quad m_{2,\tau} := \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau}(x_{i,\tau})^2 \ .$$

Similarly, the sample variance (2nd order central moment) can be obtained as ${\sigma_\tau}^2 := \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau}\big((x_{i,\tau})^2 - {\bar{X}_\tau}^2\big)$. Note that the unbiased sample variance can be obtained using Bessel's correction, as ${\sigma_\tau}^2 := \frac{1}{|\mathcal{S}_\tau|-1}\sum_{i\in\mathcal{S}_\tau}\big((x_{i,\tau})^2 - {\bar{X}_\tau}^2\big)$.

5.3 Vector aggregation and histograms

The schemes can also be extended to support vector aggregation, where each user i holds a vector $x_{i,\tau}$ of some length L. Vector aggregation can be used to compute at once the aggregates of L different variables of interest, or to compute histograms. Using the schemes to encrypt the vectors and then perform vector aggregation is more efficient than encrypting and aggregating each of the L values of interest separately. This can be done via a standard batching technique, that is, by packing the L components of the vector into a single ciphertext.

Application. Consider the example of a survey, where an aggregator is interested in computing both the average height and weight of a subset $\mathcal{S}_\tau$ of the population. For each user i, $x_{i,\tau} = [h_i, w_i]$ is a vector of length $L = 2$ containing the user's height and weight. The vector aggregate $\bar{X}_\tau = \frac{1}{|\mathcal{S}_\tau|}\sum_{i\in\mathcal{S}_\tau} x_{i,\tau}$ produces a vector of the same length as $x_{i,\tau}$ which contains the average height and weight.

Application [Histogram]. Consider a survey or an election where users are asked to make a choice among several possible values. In that case, a user's vote or answer to a survey question can be encoded as a binary vector of length the number of possible values among which the user has to chose. Each entry of the vector is a 0 or a 1 indicating whether the user chose this value or not. For instance, if a user is asked to choose one candidate among 3 candidates for an election, then vector $x_{i,\tau}$ will be of length 3, the three entries of the vector representing candidates to the election. An example answer would be $x_{i,\tau} = [0, 1, 0]$, which means that the user voted for the second candidate. The result of the election can be obtained by aggregating the individual user vectors

over the set of eligible voters $\mathcal{S}_\tau$: $\Sigma_\tau = \sum_{i \in \mathcal{S}_\tau} x_{i,\tau}$ is a histogram of the votes, i.e., a vector of the same length as $x_{i,\tau}$ which contains the counts over each entries. In the previous election example, Σ_τ would be of length 3, where each entries contain the vote counts for each candidate.

In the more advanced case of surveys or election ballots with multiple questions, the vector binarization procedure presented in the previous example can be extended. A user's election ballot or answers to the full survey can be encoded as a binary vector of length [number of questions $\times$ number of possible answers to each question], and the position of each '1' in the vector indicates the user's selected answers to each question. For instance, if a user is asked to choose one candidate among 3 candidates for an election, and to answer a question with 5 choices, then vector $x_{i,\tau}$ will be of length $3 + 5 = 8$, the first three entries of the vector representing candidates to the election, and the next 5 entries representing the possible answers to the question. An example user ballot would be $x_{i,\tau} = [0, 1, 0, 1, 0, 0, 0, 0]$, which means that the user voted for the second candidate, and chose the first possible answer to the question. The result of the election can be obtained by aggregating the individual user vectors over the set of eligible voters $\mathcal{S}_\tau$: $\Sigma_\tau = \sum_{i \in \mathcal{S}_\tau} x_{i,\tau}$ is a vector of the same length as $x_{i,\tau}$ which contains the counts over each entries. In the previous election example, Σ_τ would be of length 8, the first 3 entries containing the vote counts for each candidate, and the next 5 entries containing the counts that teach possible answer to the question obtained. Note that questions allowing the choice of multiple simultaneous answers are possible, for instance $x_{i,\tau} = [0, 1, 0, 1, 0, 0, 1, 0]$ could be a possible ballot where answers 1 and 4 to the question were simultaneously selected.

Last, ensuring the validity of each individual user inputs $x_{i,\tau}$, to avoid that users game the election or survey, is a problem beyond the scope of this paper. It could be addressed by having a voting interface that restricts the format in which users provide individual ballot inputs rather than allowing a user to return any vector containing any values, or by using zero-knowledge proofs.

5.4 Statistics – Bootstrap sampling

The bootstrap is a general method for assessing statistical accuracy, that relies on sampling with replacement [6]. Consider a training dataset containing n datapoints $\{x_1, \ldots, x_n\}$ on which an analyst would like to fit a model, or from which he would like to compute an estimate. For instance, the analyst may be interested in computing an estimate of the population mean and assessing its accuracy. The analyst generates B subsets of users of size n, called bootstrap samples. Tag τ encodes each bootstrap sample. Then the analyst computes the sample mean for each bootstrap sample of size n using AO encryption without needing to redistribute keys for each bootstrap sample set. Using the B sample means, the analyst can assess the accuracy of the sample mean estimate.

References

1. Ács, G., Castelluccia, C.: I have a DREAM! (DiffeRentially privatE smArt Metering). In: Filler, T., et al. (eds.) Information Hiding (IH 2011). LNCS, vol. 6958, pp. 118–132. Springer (2011). https://doi.org/10.1007/978-3-642-24178-9_9
2. Barthe, G., Danezis, G., Grégoire, B., Kunz, C., Zanella-Béguelin, S.: Verified computational differential privacy with applications to smart metering. In: 26th IEEE Computer Security Foundations Symposium (CSF 2013). pp. 287–301. IEEE Press (2013). https://doi.org/10.1109/CSF.2013.26
3. Benhamouda, F., Joye, M., Libert, B.: A new framework for privacy-preserving aggregation of time-series data. ACM Transactions on Information and System Security **18**(3) (2016). https://doi.org/10.1145/2873069
4. Boneh, D.: The decision Diffie-Hellman problem. In: Buhler, J. (ed.) Algorithmic Number Theory (ANTS-III). LNCS, vol. 1423, pp. 48–63. Springer (1998). https://doi.org/10.1007/BFb0054851
5. Court of Justice of the European Union: The Court of Justice declares that the Commission's US Safe Harbour Decision is invalid. Press Release No 117/15, Judgment in Case C-362/14 Maximillian Schrems v Data Protection Commissioner (Oct 2015), http://curia.europa.eu/jcms/upload/docs/application/pdf/2015-10/cp150117en.pdf
6. Efron, B.: Bootstrap methods: Another look at the jackknife. The Annals of Statistics **7**(1), 1–26 (1979), http://www.jstor.org/stable/2958830
7. Erkin, Z., Tsudik, G.: Private computation of spatial and temporal power consumption with smart meters. In: Bao, F., Samarati, P., Zhou, J. (eds.) Applied Cryptography and Network Security (ACNS 2012). LNCS, vol. 7341, pp. 561–577. Springer (2012). https://doi.org/10.1007/978-3-642-31284-7_33
8. Garcia, F.D., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: Cuellar, J., et al. (eds.) Security and Trust Management (STM 2010). LNCS, vol. 6710, pp. 226–238. Springer (2011). https://doi.org/10.1007/978-3-642-22444-7_15
9. Hao, F., Zieliński, P.: A 2-round anonymous veto protocol. In: Christianson, B., et al. (eds.) Security Protocols. LNCS, vol. 5087, pp. 202–211. Springer (2009). https://doi.org/10.1007/978-3-642-04904-0_28
10. Hellman, M.E., Diffie, W.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
11. Jawurek, M., Kerschbaum, F.: Fault-tolerant privacy-preserving statistics. In: Fischer-Hübner, S., Wright, M. (eds.) Privacy Enhancing Technologies (PETS 2012). LNCS, vol. 7374, pp. 221–238. Springer (2012). https://doi.org/10.1007/978-3-642-31680-7_12
12. Jawurek, M., Kerschbaum, F., Danezis, G.: SoK: Privacy technologies for smart grids – A survey of options. Tech. Rep. MSR-TR-2012-119, Microsoft Research, Cambridge, UK (Nov 2012), https://www.microsoft.com/en-us/research/publication/privacy-technologies-for-smart-grids-a-survey-of-options/
13. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: Sadeghi, A.R. (ed.) Financial Cryptography and Data Security (FC 2013). LNCS, vol. 7879, pp. 111–125. Springer (2013). https://doi.org/10.1007/978-3-642-39884-1_10
14. Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly aggregation for the smart-grid. In: Fischer-Hübner, S., Hopper, N. (eds.) Privacy Enhancing

Technologies (PETS 2011). LNCS, vol. 6794, pp. 175–191. Springer (2011). https://doi.org/10.1007/978-3-642-22263-4_10
15. Leontiadis, I., Elkhiyaoui, K., Molva, R.: Private and dynamic times-series data aggregation with trust relaxation. In: Gritzalis, D., Kiayias, A., Askoxylakis, I.G. (eds.) Cryptology and Network Security (CANS 2014). LNCS, vol. 8813, pp. 305–320. Springer (2014). https://doi.org/10.1007/978-3-319-12280-9_20
16. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: Elmagarmid, A.K., Agrawal, D. (eds.) 2010 ACM SIGMOD International Conference on Management of Data. pp. 735–746. ACM Press (2010). https://doi.org/10.1145/1807167.1807247
17. Shi, E., Chan, T.H.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: Network and Distributed System Security Symposium (NDSS 2011). The Internet Society (2011), https://www.ndss-symposium.org/wp-content/uploads/2017/09/shi.pdf
18. Teruya, T., Saito, K., Kanayama, N., Kawahara, Y., Kobayashi, T., Okamoto, E.: Constructing symmetric pairings over supersingular elliptic curves with embedding degree three. In: Cao, Z., Zhang, F. (eds.) Pairing-Based Cryptography – Pairing 2013. LNCS, vol. 8365, pp. 305–320. Springer (2014). https://doi.org/10.1007/978-3-319-04873-4_6