Differentially Private Queries in a Practical Systems Setting
CS261 Project Report

Neil Agarwal
University of California, Berkeley

Jack Sullivan
University of California, Berkeley

Abstract

The increasing amount of sensitive user data that is being generated can provide value in creating sophisticated inferences about people and their surroundings. Although aggregation statistics can be extremely useful, users do not trust any single third-party aggregator to see their data in plaintext. We contribute a practical system for privacy preserving queries using hardware enclaves. Our design provides trust in an untrusted online service provider infrastructure, flexible aggregate queries, maintenance of differential privacy, and does not require user clients to be online for query computations. We designed a proof-of-concept implementation and analyzed the scalability of query latencies with increasing numbers of users. With individual privacy preserved, we hope users with sensitive data will be much more willing to participate in statistical databases that have widely applied benefits.

1 Introduction

The increasing ubiquity of smart phones are equipped with a rich set of embedded sensors such as microphones, cameras, accelerometers, gyroscope, and GPS. Such sensors enable the collection of streams of time-series data. The data generated by these sensors provides opportunities to make sophisticated inferences about not only people (e.g. human activity, health, location, social event) but also their surroundings (e.g. pollution, noise, weather, oxygen level), and thus can help improve peoples health as well as lives. This technology and individual participation help drive domains such as environmental monitoring, traffic monitoring, and health care.

Although aggregation statistics computed from time-series data can be extremely useful, the data from individual users are often privacy-sensitive, and users do not trust any single third-party aggregator to see their data in plaintext. Projects such as E-mission, a research project at the University of California, Berkeley that aggregates crowd-sourced activity and behavior data to enable personal and structural shifts to sustainable transportation, collects sensitive location data and would greatly benefit from differentially private secure aggregation. To encourage user participation in aggregated queries, it is important to address such privacy concerns by considering how an untrusted data aggregator can learn desired statistics over multiple participants data without compromising each individual's privacy. While existing literature provides solutions for differentially private aggregation for mobile sensing data, they fail to achieve a solution that ensures a secure threat model that supports a flexible set of queries, is scalable with a large number of users, and does not require any user involvement.

In this project, we present a method for privacy preserving and differentially private queries in a practical systems setting. We use propagating remote attestation of hardware enclaves to ensure trust and privacy across untrusted infrastructure. Each user has their own enclave which acts as their attestable proxy in the cloud; the user enclaves communicate with an aggregate enclave who they attest. Hence, the propagating or chaining of attestation. The aggregate enclave performs the final aggregation computation of intermediate values computed by user enclaves. Our model provides for a separation of clients (data sources) and query computations; users are only required to contact the system when they upload data. This paper serves as a comprehensive description of our project, detailing the context and motivation, related work, design, and a proof-of-concept implementation.

To test the scalability and practicality of this system, we created a Python implementation of the aggregate query computation component of our proposed system. The enclaves are implemented as Docker containers that run SCONE (Secure Linux Containers on Intel SGX) [1]. A simple sum query experiment was run in order to observe the effect of using hardware enclaves on overall query latencies. We notice that there is considerable overhead for using SGX (about 3.12 seconds) and the total query latency for 50 users is high when using SGX secure hardware. Thus, future work hopes to tackle the potential scalability limitations by exploring partitioning user enclaves onto different machines.

2 Background

The goal of our system is to be able to produce privacy-preserving queries that additionally ensure individual privacy. Privacy-preserving queries ensure that the querier can learn only the final query result and not any intermediate results. A stronger notion of privacy-preserving queries is differential privacy.

2.1 Differential Privacy

2.1.1 Definition & Overview

The concept of differential privacy was first introduced by C. Dwork [5] and ensures that an individual is not at increased risk of privacy when they participate in a statistical database and their associated queries. Essentially, a differentially private query result should be approximately the same whether any participant is included or excluded from the query. The traditional setting involves a trusted aggregator adding statistical noise to the query result before releasing it to the data analyst. Our approach uses the same strategy, but we rely on attestable enclaves to add the noise to the query result.

In this paper we consider ε-differential privacy. Let ε denote a positive real number and A denote a randomized algorithm that takes a dataset representing the actions that can alter the data as input. The algorithm A is said to provide ε-differential privacy if, for all datasets D_1 and D_2 that differ on a single element and for all subsets S of $Range(A)$:

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

where the probability is taken over the randomness used by the algorithm A.

Adding statistical noise to each query result is not sufficient in ensuring differential privacy if a querier repeats a query and averages the results as this will lead to a more precise query answer. To mitigate this, privacy budgets, the ε in the definition above, represents the maximum privacy loss. We can think of each query as a privacy expense which incurs an incremental privacy loss and once the privacy budget is depleted, access to that data is blocked. Privacy budget can be replenished if new data is added.

2.1.2 Query Noise Procedure

We can ensure ε-differential privacy by adding statistical noise such as Laplace noise [6]. The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function:

$$Lap(x|b) = \frac{1}{2b} exp(-\frac{|x|}{b})$$

To ensure ε-differential privacy our value for b is $\frac{\triangle f}{\varepsilon}$ [6]. $\triangle f$ represents the global sensitivity of the query. Sensitivity is defined as the maximum query result difference between any two databases that only differ by one element (i.e. an individual being included or not included in a database). The formal definition of sensitivity is:

$$\triangle f = max||f(D_1) - f(D_2)||_1$$

where the maximum is over all pairs of datasets D_1 and D_2 from the space of all possible databases differing by at most one element, and $||\cdot||_1$ denotes the l_1 norm.

Therefore ensuring ε-differential privacy involves adding $Lap(\frac{\triangle f}{\varepsilon})$ noise to the query result.

3 Related Work

3.1 Cryptographic Protocols

Our differentially private approach shares similarities with distributed differentially privacy related work. Rastogi and Nath [14] proposed the first differentially private aggregation algorithm for distributed time-series data that offered good practical utility without a trusted central server. This solution had two main drawbacks: an extra round of interaction between the aggregator and users for decrypting the sum, leading to high communication costs, and the lack of fault tolerance (i.e. requires all users to be online until decryption is completed, which may not be practical in many mobile sensing scenarios due to user mobility and the heterogeneity of user connectivity). To overcome the extra round of interaction, E. Shi, T-H.H. Chan, E. Rieffle, R. Chow and D. Song [16] proposed a Diffie-Hellman-based encryption scheme; however, this was not fault tolerant as it still relied on all users being online to contribute their encrypted data and noise.

Fault tolerance was addressed a year later with a new scheme by T-H.H. Chan, E. Shi, and D. Song [3]. They achieved fault tolerance by constructing a binary interval tree over the n users, allowing the aggregator to estimate the sum of contiguous intervals of users as represented by nodes in the interval tree. This is used to handle user failures, joins, and leaves with small logarithmic communication and estimation error. However, since each node's data is aggregated into multiple sums, a large noise is added to each node's data to provide differential privacy which leads to high aggregation error on large systems relative to related works.

Both [16] and [3] require the decryption to traverse the possible plaintext space of the aggregated value, which is very expensive for a large system with large plaintext space. Especially in the case of mobile sensing, the plaintext space of some applications can be large. The approach of Q. Li and G. Cao [9] retains the same data perturbation strategy as [16] but improves that encryption scheme by using PRF (Pseudo-Random Function) computations over modular exponentiation.

Instead of having a key dealer to distribute keys to all users that they would use for subsequent queries, Jawurek et al. [7] uses a key authority replaces the key dealer and users encrypt their data and randomness using the public key of the key authority. The aggregator and key authority then bi-directionally communicate to decrypt the noisy sum. This scheme handles fault tolerance and dynamic joins/leaves, but still allows for collusion between the aggregator and key authority. The scheme of Rane et al. [13] involves only the aggregator and users with no other system components while ensuring fault tolerance for any number of users through Shamir's secret sharing. The drawback is this paper did not incorporate differential privacy, aggregator and user communication is bi-directional, and the secret sharing scheme is relatively inefficient ($O(n^2)$ where n is the number of total users). Bindschaedler et al. [2] uses the same scheme but incorporates differential privacy at the cost of efficiency.

3.2 Practical Systems

There has also been prior work in constructing practical privacy-preserving systems. PrivStats [12] is a system for computing aggregate statistics over location data in a privacy-preserving manner. However, PrivStats relies on the use of an anonymization network and any aggregation functions must be able to run on homomorphically encrypted data. [15] proposes a data dissemination platform that supports untrusted infrastructures. However, this system does not support persistent user data – aggregate computations are performed on live streaming data from clients. In Prio [4], another privacy-preserving system for the collection of aggregate statistics, all queries must be performed on Affine-Aggregatable Encodings (AFEs); however, we avoid this limitation by performing all queries on *plaintext* values. [10] provides a new communication efficient and privacy-preserving data aggregation protocol but relies on the use of a trusted third party for key setup. Chorus [8] presents the first differentially private practical approach for SQL queries; this work focuses on relational databases and does not involve the data collection aspect.

3.3 Contribution

In this project, we contribute a practical system for privacy preserving queries that: (1) does not depend on clients (data sources) to be constantly online/available, (2) allows for data analysts and OSP infrastructure to be untrusted, (3) allows any aggregate function to be computed over data (from simple queries to more complex machine learning models) and (4) incorporates differential privacy.

4 Threat Model

We can split up the entities of this protocol into three main components: users (mobile devices uploading data), the online service provider, and data analysts (clients who would like to perform aggregate queries on user data). We take on the assumption that a majority of distributed users are honest (prevention against data pollution attacks are discussed in Section 5.7.3). We consider an untrusted threat model for the online service provider (a tangible example of this is E-mission) that consists of the machines that store the user enclaves, the shared encrypted and persistent database, and the machine that hosts the aggregate enclave). We also consider the data analyst/querier untrusted. Overall, this is a strong threat model that assumes trust only in a majority of users.

5 Design

5.1 Online Service Provider Infrastructure

For an online service provider to make use of our protocol, the infrastructure would need to augmented by the following: an encrypted and persistent database containing all user data, a set of secure hardware enclaves (one per data source/user) and a separate *aggregate* secure hardware enclave.

5.2 Overview

At a high level, the protocol can be broken up into three main steps. In the first step, users join the online service provider's infrastructure, engage in remote attestation of the user enclave reserved for them, and send their generated secret key to the enclave so that the enclave can decrypt user data and thus participate in queries. The code in the user enclave also attests code in the aggregate enclave when computing a query - this propagated aggregation provides a level of trust amidst an untrusted infrastructure. In the second step, users upload data to the shared encrypted database and this is the only time the user has to be online. The third step involves a data analyst submitting a query and the query being computed throughout the OSP infrastructure. The aggregate enclave broadcasts the query and its parameters to the user enclaves machines which notify the user enclaves to participate in the query. These enclaves decrypt and perform the query function over their dedicated user's data from the database and forward the intermediate results to the aggregate enclave. The aggregate enclave performs the final aggregation computation, and will add noise to the final result based on certain privacy parameters which will be covered in more detail later.

5.3 User Privacy Budgets

To provide differential privacy for user data, we allocate a privacy budget $\varepsilon_{i,m}$ for each user u_i independently for a different segments of times (in this case, months m). Privacy budgets are tied to users as opposed to queriers because multiple queriers could collude to learn more information than they would learn alone. Each query provided by the data analyst includes an ε_q that represents the differential privacy budget for that query. A user u_i's data is only included in the results of a query if they have data related to the query and if $\varepsilon_{i,m} - \varepsilon_q > 0$; if included, their privacy budget decrements by the query privacy budget $\varepsilon_{i,m} \leftarrow \varepsilon_{i,m} - \varepsilon_q$. As a result, a data analyst can specify a large value for ε_q providing a more accurate query result, but this reduces the number of times that a data analyst can query that data.

5.4 Database

All the user data is stored in a shared, encrypted, and persistent database. The primary database index is the user's unique id followed by a month. Note the choice of *month* as the granularity of data storage: this decision was based off a comparison of efficiency and user privacy. Using more granular time leads to greater visibility of user activity. The benefit of a more granular time is that less data would have to be streamed to a user enclave during query computations. On the other hand, a more coarse time (e.g. a year) means that all user data for a year would have to be sent to the user enclave upon a query during that year.

5.5 Query

A query Q is submitted by the data analyst and comprises a number of parameters. These parameters are currently the query type, time range, and location range. A differential privacy budget ε_q is also included in the query. Our system supports any query that ensures differential privacy by adding noise which includes simple aggregation and machine learning queries. This is because the function is calculated over plaintext (obviating any restrictions of computing over encrypted data).

5.6 Enclaves

There are two main benefits of enclaves: secure computation and remote attestation. Enclaves provide a secure execution environment, meaning that the enclave memory space is encrypted so any outsider including the kernel and hypervisor can not infer anything about the computation. Remote attestation of an enclave allows an outsider to verify the identity of a program being run by the enclave [11]. Once attested, the enclave can no longer be modified. This characteristic of enclaves allows for secure offline computation of user data. Attestation of user enclaves is done by users; attestation of the aggregate enclave is done by the user enclaves. We call this propagated attestation.

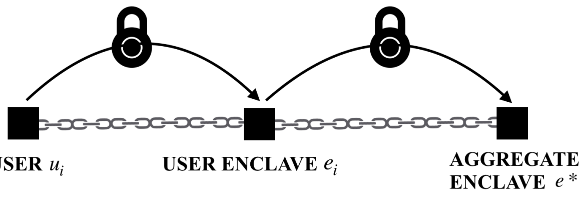

Figure 2: This represents propagated or chained attestation - our fundamental mechanism of ensuring trust in an untrusted infrastructure.

5.7 Protocol

Our protocol supports three main system operations: User Join, User Upload Data, and Aggregate Query Computation.

5.7.1 User Join

In this phase, users join the online service provider infrastructure and engage in steps to set up their respective enclave.

1. User u_i sends a `JOIN` message to the OSP.
2. The OSP allocates an enclave e_i for u_i and load e_i with the appropriate user enclave code pages.
3. The OSP sends contact information for enclave e_i to user u_i.
4. User u_i engages in remote attestation of e_i, generates a secret key sk_i, and then sends sk_i to the now attested e_i which will be stored in enclave memory.

The symmetric secret key the user generated in this step is used for encrypting and decrypting their sensitive data. Once the user's respective enclave has this key, their enclave is able to decrypt all of that user's data; this provides a proxy for users to participate in query computations. This setup allows for users to be offline during the query computation process.

Also note the remote attestation of the user enclave – this begins the propagation of ensuring trust throughout the OSP infrastructure. The user enclave's code attests the aggregate enclave's code before sending it sensitive intermediate query results. This attestation is necessary as the OSP could replace aggregate enclave code with malicious code that could leak sensitive data.

In the attestation process, the user compares the hash provided by the user enclave with a hash from an attestation service, say Intel Attestation Service (ISA).

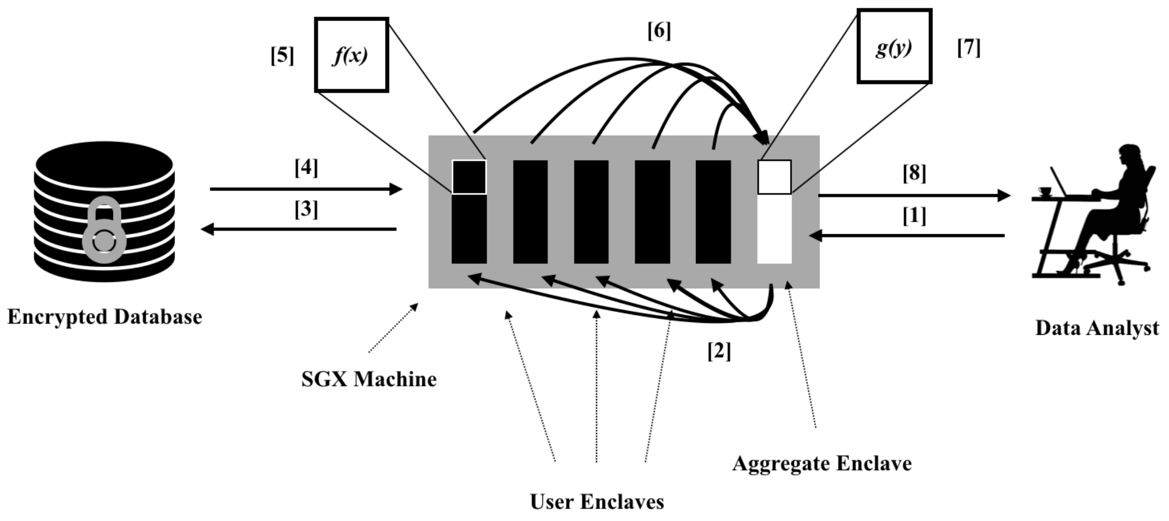

Figure 1: Aggregate Query Computation: In this phase, an aggregate computation is conducted over user data. Recall that this phase does not require user involvement - the decryption of data is done by user enclaves.

5.7.2 User Upload Data

In this phase, users upload their encrypted data to the OSP database where it is indexed and stored for later access.

1. User u_i sends $month : E_{sk_i}(data)$ to the OSP.
2. The OSP appends the data in an encrypted database, indexing by user i's user id and the month.
3. The OSP replenishes user u_i's privacy budget for that month.

Note that the month and user id are not encrypted. This is so the database can index the *encrypted data* accordingly.

5.7.3 Aggregate Query Computation

In this phase, a data analyst submits an aggregate query to be calculated on user data using the OSP infrastructure. See Figure 1 for a graphical representation of this phase.

1. Data analyst A sends Q, ε_q to the aggregate enclave $e*$ where Q is composed of a query type, time range, and location range. ε_q is the differential privacy budget.
2. $e*$ sends Q, ε_q to all user enclaves.
3. For all users u_i, user enclave e_i request from the database the data for the relevant months based on the time range of the query Q.
4. Database streams encrypted data from database to enclaves.
5. For all users u_i, user enclave e_i decrypts data with symmetric key and computes an intermediate query function over the relevant data.
6. For all users u_i, user enclave e_i attests $e*$ and if verified, then it sends the unencrypted result to aggregate enclave $e*$.
7. The aggregate enclave $e*$ performs the final query function over this data, only including data if the $\varepsilon_q >$ user's privacy budget $\varepsilon_{i,m}$ for month m. If the user's data is included in that query, $\varepsilon_{i,m} \leftarrow \varepsilon_{i,m} - \varepsilon_q$.
8. The aggregate enclave $e*$ adds $Lap(\frac{\triangle f}{\varepsilon_q})$ noise to the final query result and sends to the data analyst.

Note that the amount of data sent from the database to the user enclaves is dependent on the granularity of the database indexing as well as the time range given in the query. Because of the variable length of this data and the use of lightweight enclaves, data is streamed into the user enclave. As a result, streaming algorithms must be used for computation.

In the case that a subset of users are potentially malicious, the code inside user enclaves can perform data validation filtering and the aggregate enclave $\varepsilon*$ can additionally attest user enclaves to ensure that this filtering takes place.

6 Evaluation

6.1 Implementation

We provide a Python implementation of the aggregate query computation component of the system. The enclaves are implemented as Docker containers that run SCONE [1]. SCONE runs programs inside secure enclaves, can transparently encrypt files and network traffic, attest programs to ensure that only the correct, unmodified programs are executing, and is compatible with Docker. Users have privacy budgets stored in their user enclaves and decrements them by the query privacy budget if a user has enough privacy budget and has data satisfying the query.

A Flask server is set up to handle all queries which are modeled by sending a POST request containing the query to the server. Upon receiving a query, the aggregate enclave broadcasts the query and its parameters to all of the controllers of the user enclave machines. A controller is responsible for initialization and management of enclaves within a given machine. The controllers then "unpause" the user enclaves and forward the query to the user enclaves. The user enclaves compute the intermediate query function and send the result to the aggregate enclave. Once this is done, the controllers "pause" the user enclaves. The aggregate enclave does the final computation and returns the result to the querier. The code is made available at: https://github.com/jesbu1/scone_sgx_scripts.

6.2 Experiment Setup

For our evaluation, we analyze the scalability of increasing number of users on the overall query latency. Various experiment details are listed below:

- Query Details:
  - Type: Summation
  - One time period
  - One location area
- All user enclaves are grouped in a single Intel SGX machine.
- The aggregate enclave shares the same machine as the user enclaves.
- Query latencies were recorded for every multiple of 5 users from 5 to 50.
- Experiment was ran in both simulation (no SGX) and hardware (with SGX) modes in order to observe the overhead for using enclaves.
- Users enclaves had immediate access to their raw data via a JSON file inside their enclave (no streaming or decrypting from an encrypted, persistent database).

For this experiment, we toggle between the different execution modes: simulation or hardware. We instantiate the aggregate enclave Docker container. For the set number of users, we also instantiate user enclave containers. We start a timer, send the query, and measure the amount of time until the final computation is sent back to the querier. After the query is completed, the user enclave Docker containers are removed and the aggregate enclave Docker container is removed once the experiment is complete.

6.3 Experiment Results

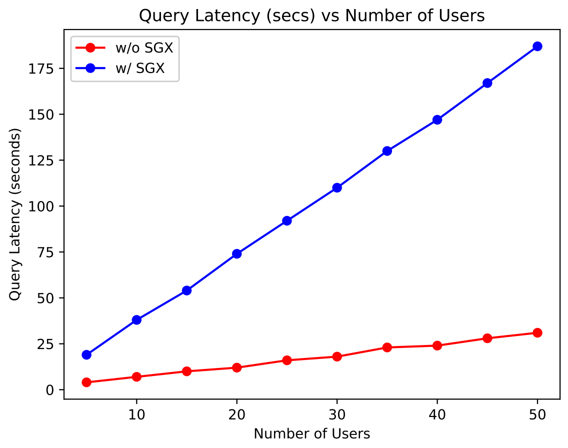

Figure 3: This graph demonstrates the effects of increasing the number of users on overall query latency and displays the overhead of using SGX.

The linear relationship of both curves in Figure 3 are expected as each user has their own enclave and each user must observe their data to determine if they can compute the query. The key takeaways are how the latencies scale for a certain amount of users sharing one machine and the enclave overhead. The enclave overhead latency can be observed from the difference between the two lines. From this experiment, we can see the average latency for running a query in simulation mode is 0.62 seconds while using SGX amounts to an average query latency of 3.74 seconds. Thus the average SGX overhead is 3.12 seconds. This is an indication that we need to split users across more machines (in the case of SGX).

7 Conclusion and Future Work

In this project, we contribute a practical system for privacy preserving queries using hardware enclaves. Our design allows for offline computation, an untrusted online service provider infrastructure, flexible aggregate queries, and differentially private queries. We also provide a proof-of-concept implementation of the query computation using SGX.

Our future work hopes to tackle the potential scalability limitations, possibly exploring partitioning user enclaves onto different machines. Moreover, as our system streams all of the data for a user for the queried month, further analysis is needed to measure the effect of the amount of data on the overall efficiency and speed of the query computation. Furthermore, there is a possibility of network traffic from database to user enclaves revealing private information – more exploration into the obliviousness of network communication is needed. Also, we aim to support more query types such as origin-destination pairs as well as examine ways to replenish user privacy budgets. Additionally, in order to prevent malicious queriers from depleting the privacy budgets of the users, it is important to include querier access control and provide users with the ability to specify which queriers they want to answer queries to. We are excited to continue working on our design and incorporating it with data collection and aggregation platforms such as E-mission.

Research Project Information, Collaborators, and Acknowledgments

This work is part of a research project (Jack Sullivan's 5th-year M.S. thesis); distributed differential privacy related work was reviewed before starting the semester. This work was conducted with the help of Jesse Zhang (an undergraduate researcher at the Rise Lab) and under the mentorship of Ph.D. student K. Shankari and Prof. David Culler. Jesse helped tremendously with the proof-of-concept implementation. Also, thank you to the BETS lab for letting us run experiments on their Intel SGX machine. Finally, a big thank you goes to Prof. Raluca Ada Popa for a great class and for providing valuable input on our system design in office hours.

References

[1] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. Stillwell, et al. Scone: Secure linux containers with intel sgx. In *OSDI*, volume 16, pages 689–703, 2016.

[2] V. Bindschaedler, S. Rane, A. B. . V. Rao, and E. Uzun. "Achieving Differential Privacy in Secure Multiparty Data Aggregation Protocols on Star Networks". *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY 17), 2017.*, 2017.

[3] T.-H. Chan, E. Shi, and D. Song. "Privacy-Preserving Stream Aggregation with Fault Tolerance". *Proc. Sixth Int'l Conf. Financial Cryptography and Data Security (FC '12)*, 2012.

[4] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, pages 259–282, 2017.

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

[6] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. In *Foundations and Trends in Theoretical Computer Science*, 2014.

[7] M. Jawurek and F. Kerschbaum. "Fault-Tolerant Privacy-Preserving Statistics". *The 12th Privacy Enhancing Technologies Symposium (PETS)*, 2012.

[8] N. M. Johnson, J. P. Near, and D. X. Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[9] Q. Li, G. Cao, and T. F. L. Porta. "Efficient and Privacy Preserving Stream Aggregation in Mobile Sensing with Low Aggregation Error". *Technical Report, The Pennsylvania State University*, 2013.

[10] Liu, Chuyi. An application of secure data aggregation for privacy-preserving machine learning on mobile devices. Master's thesis, 2018.

[11] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. *HASP@ ISCA*, 10, 2013.

[12] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 653–666, New York, NY, USA, 2011. ACM.

[13] S. Rane, J. Freudiger, A. E. Brito, and E. Uzun. "Privacy, Efficiency Fault Tolerance in Aggregate Computations on Massive Star Networks". *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2015.

[14] Rastogi and S. Nath. "Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption". *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2010.

[15] L. Sampaio, F. Silva, A. Souza, A. Brito, and P. Felber. Secure and privacy-aware data dissemination for cloud-based applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17, pages 47–56, New York, NY, USA, 2017. ACM.

[16] E. Shi, T. Chan, E. Rieffel, R. Chow, and D. Song. ”Privacy-Preserving Aggregation of Time-series Data”. *Proc. Network and Distributed System Security Symp. (NDSS ’11)*, 2011.