An Application of Secure Data Aggregation for Privacy-Preserving Machine Learning on Mobile Devices

by

Chuyi Liu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Machine learning algorithms over big data have been widely used to make low-priced services better over the years, but they come with privacy as a major public concern. The European Union has made the General Data Protection Regulation (GDPR) enforceable recently, and the GDPR mainly focuses on giving citizens and residents more control over their personal data. On the other hand, with personal and collective data from users, companies can provide better experience for customers like customized news feeds and real time transportation systems. To solve this dilemma, many privacy-preserving schemes have been proposed such as homomorphic encryption and machine learning over encrypted data. However, many of them are not practical for the time being due to the high computational complexity. In 2017, Bonawitz *et al.* proposed a practical scheme for secure data aggregation from privacy-preserving machine learning, which comes with the affordable calculation and communication complexity that considers practical users' drop-out situations. However, the communication complexity of the scheme is not efficient enough because a mobile user needs to communicate with all the members in the network to establish a secure mutual key with each other.

In this thesis, by combining the Harn-Gong key establishment protocol and the mobile data aggregation scheme, we propose an efficient mobile data aggregation protocol with privacy-preserving by introducing a non-interactive key establishment protocol which reduces the communication complexity for pairwise key establishment of n users from $O(n^2)$ to a constant value. We correct the security proof of Harn-Gong key establishment protocol and provide a secure threshold of degree of polynomial according to Byzantine Problem. We implement *KDC* side Harn-Gong key establishment primitives and prepare a proof-of-concept Android mobile application to test our protocol's running time in masking private data. The result shows that our private data masking time is 1.5 to 3 times faster than the original one.

Acknowledgements

I would like to thank Prof. Gong for her supervision of my thesis. She is a sharped, energetic and caring professor who always encourage me to explore different research area, from whom I learned a lot.

I would like then give many thanks to my thesis readers, Prof. Menezes and Prof. Tripunitara. Prof. Menezes is very keen to the mathematical proof of the thesis and pointed out the error of security proof from the Harn-Gong protocol. His comments make the thesis much more solid.

Also I would like to thank Dr. Kalikinkar Mandal who have given me a lot of advises for the clarity of my statement, and he also promoted my experiment results of the implementation of the thesis.

Dedication

This is dedicated to my parents and my boyfriend for their endless love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last few years, the development and usage of smart phone applications have given much attention to data usage by cloud services. Since the computation and storage ability of a mobile device is limited, the data the user provides can be easily leaked. At the same time, cloud service providers can use their substantial resources to analyze users' data to provide better performance on their application. It is reasonable for users to want to maintain privacy of the data (*eg.* health data and financial data) while enjoying the cloud service at the same time. However, as the Internet Trends 2016 report [42] suggests, more and more users choose to avoid online usage because of privacy concerns. It is important to provide a communication and computationally efficient privacy scheme for today's big data usage.

There are numerous cloud services that a mobile user may need. For example, cloud storage, speech assistance, and recommendation systems. These technologies make use of users' data and can predict users' interests in order to make a profit. So it is with high probability that cloud service providers might be interested in probing users' data at the same time they respond to users' demands. This type of curious-but-honest threat model is a standard one for cloud service. Even when cloud service providers are not interested in probing users' privacy, the data format it stores in its server may also be prone to attacks, such as Facebook data breaches [59], which not only contributes to a company's loss, but also threatens users' privacy in other related areas.

There are many technologies available to address the above issues, such as homomorphic encryption. Homomorphic encryption is a cryptographic scheme by which a cloud service

provider can use operates on ciphertexts of users' data in the same way it operates on plaintext. When users get the operated data back, they can get the desired cloud computation results without leaking their private information. It is fair to say that homomorphic encryption makes data malleable in a way that does not leak the important information of users. The confidentiality of users' data is preserved while they enjoy cloud services. But there is still a long way for fully homomorphic encryption to be truly practical in daily use.

The basic business model of ML-based data analysis is a closed circle: the more the data is accumulated, the more accurately the model can be trained, which means more useful services are provided and more users are attracted to the services, and the process finally leads to more data being collected [62]. People who use machine learning methods tend to believe that when you cannot figure something out, you just give more data to the model and let it find an answer. Ubiquity of sensors via mobile and IoT devices has caused a surge in personal data generation and use while users' data collection can be important for the model's training. Recently, Bonawitz *et al.* [7] proposed a privacy preserving strategy to update the machine learning model with a new federate learning method. However, it needs to establish a secret key for each pair of users in the network. For a large scale of users (say 10^5), it needs 10^{10} pairwise keys to be established since the number of keys increases quadratically for the two party key agreement protocol.

It is widely accepted that a key agreement protocol can establish a safe communication channel for any two parties. However, when a group of members using two party key agreement protocols wants to start a secure communication, it is communication inefficient to use the standard protocol. It has been a big concern of how to establish a group key without communication overhead as the two party key agreement protocol does. In 1982, Blundo *et al.*[6] proposed an information theoretically secure one based on multivariate polynomials. Later in 2015, Harn and Gong proposed a storage efficient univariate polynomial product protocol [28] based on it. The Harn-Gong protocol is not only useful in establishing a secure key for a group communication, but also its non-communicative way to establish group keys helps to reduce communication overhead significantly in a scenario where a pairwise key for every pair of users in the network is needed.

Secure data aggregation is a process of securely collecting data from a group of users where each user has a private input, and the aggregation process does not reveal any private information about users' inputs. The secure data aggregation problem, particularly aggregate sum, has been studied in the literature in different contexts such as wireless sensor networks [16] and cloud database services [44]. The mobility nature of mobile users

introduces a new property called robustness or failure-robust where a mobile user is either alive in the system or drops out from the system.

In this work, we consider the problem of aggregating the sum of mobile users' data in a secure manner. Our contributions are as follows:

1) We propose an efficient mobile data aggregation protocol with privacy-preserving by introducing a non-interactive key establishment protocol that reduces pairwise key establishment of a group with $O(n^2)$ communication complexity to a constant value.

2) We correct the security proof of the Harn-Gong key establishment protocol and provide a secure threshold of degree of polynomial according to Byzantine Problem.

3) We implement *KDC* side Harn-Gong key establishment primitives and prepare a proof-of-concept Android mobile application to test our protocol's running time in masking private data. The result shows that our private data masking time is 1.5 to 3 times faster than the original one.

The rest of the thesis is organized as follows. In Chapter 2, we introduce some known methods for privacy issues related to machine learning algorithms, and introduce related key establishment protocols. We also discuss homomorphic encryption as a possible solution for cloud service providers to provide privacy-preserving services. In Chapter 3, we present terminologies that are used in the key establishment protocol and private data masking protocols. In Chapter 4, we present the detail of constructions of Harn-Gong key establishment protocol [28] and its security analysis as well as performance evaluation. In Chapter 5, we provide a detailed complexity analysis on complexity of privacy-preserving mobile data aggregation scheme in [7]. We present our new protocol in Chapter 6 and analyze its complexity. The implementation and analysis is in Chapter 7 and Chapter 8 is comprised of conclusions and some future work.

Chapter 2

Literature Survey

We introduce some machine learning algorithms and their privacy issues in Section 2.1. We then provide details of some key establishment protocols to build a secure communication channel in Section 2.2. Finally, we present the concept of the homomorphic encryption as a tool for addressing privacy issues in machine learning in Section 2.3, and state why it is not practical in current machine learning applications.

2.1 Machine Learning

Machine learning is a process that makes computer progressively improve performance on some specific tasks rather than being explicitly programmed. The term was first introduced by Samuel in 1959 [51] and it has become an essential method for problems of computer vision, speech detection, robotics, finances, etc. In general, machine learning algorithms can be divided into three paradigms:

1) Supervised learning

2) Unsupervised learning

3) Reinforcement learning

Supervised learning is a machine learning method by which labeled training data is given to the machine learning model, and the model maps input to output based on example pairs [43]. Many different forms of mappings exist, including decision trees [48],

decision forests [31], logistic regression [18], support vector machines [17], and neural networks [41]. One high-impact area of progress in supervised learning in recent years involves deep networks [4], which has multiple hidden layers between the input and output layers. Such deep learning systems can be trained over vast collections of data and can contain billions of parameters. The large-scale deep learning systems have had a major effect in computer vision [37] and speech recognition [27] in recent years.

Unsupervised learning is a machine learning method by which the model learns from the unlabeled training data. The instinct of unsupervised learning is to derive the hidden structure of the raw data [29]. Many different objectives of unsupervised learning have been proposed, such as discovering clusters in the input data [5], extracting features that characterize the input data more compactly [45], and uncovering non-accidental coincidences within the input data [63]. Given that the goal is to exploit the particularly large data sets, the concern with computational complexity is paramount in those methods.

Reinforcement learning is a machine learning method that explores training data between supervised and unsupervised learning [56]. Instead of giving exact correct answers, the training data in reinforcement learning provides only an indication as to whether an action is correct or not. The applications of reinforcement learning have increased in operations research [15] and control theory in the field of Markov decision processes [32].

Machine learning algorithms have been used in many online service providers with recommendation systems (Netflix, Spotify, Facebook, etc.), and those companies are prone to store and use a big number of user data. However, data breaches have been a big problem for those giant companies. The Facebook's 87 million users' data leakage is just one example of today's various company's data breaches [59]. With the introduction of the General Data Protection Regulation of European Union [60] enforceable in May 2018, data privacy of machine learning algorithms could be a big legal issue in the future.

2.2 Key Agreement Protocols

A key establishment protocol is a method to enable multiple users to share a secret key [28]. The key agreement protocol is a special type of key establishment protocols that enable all entities to influence the outcome collaboratively. The most widely-used key agreement

protocol is Diffie-Hellman key agreement protocol [20]. However, it can only allow two members to communicate efficiently. For a group of more than two members, $O(n^2)$ communication complexity is needed to establish a pairwise key for each pair of members in the group.

Many natural adaptations of the original Diffie-Hellman key agreement protocol [55, 14, 34] have then been proposed for group key agreement protocols, but their time efficiency and confidentiality have become great concerns. For example, Steiner *et al.* [55] proposed an adapted Diffie-Hellman protocol that needs $O(n)$ rounds communication among n participants, which cannot be scaled up to a large network.

In 1995, Burmester and Desmedt [14] proposed a generalized Diffie-Hellman group key establishment protocol with a constant round of communication for each user. However, it lacks a full proof of security because it only provides a security proof for an even number of participants. In 2007, Katz *et al.* [34] provided the first constant round, fully scalable and provably-secure protocol compiler and proved the Burmester-Desmedt protocol to be secure. In [40], the authors provide elliptic curve version of Burmester-Desmedt protocol and its authenticated variant. However, the digital signature of each user in the group for authentication of Diffie-Hellman public keys may also restrict its usage since the generation and verification of digital signature may take substantial computational resources.

In 1992, Blundo *et al.* [6] proposed a non-interactive k-secure m-conference protocol based on multivariate polynomials, where k-secure means any k users cannot collude to get other member's share information, and m-conference means that the protocol can establish a group key of up to m members. However, since the protocol requires each polynomial to have $(k+1)^{m-1}$ coefficients, the space complexity of the protocol is exponential in the number of users, which makes it impossible to use in a large number of users' network.

Based on the work of [6], in 2015, Harn and Gong [28] provide a protocol that uses constant time to achieve both authentication and confidentiality for the group communication in a public key network. The protocol uses a univariate polynomial product to reduce the space complexity of [6] and requires only $(k+1)$ coefficients for the group key computation. The protocol has space complexity linear in the number of users in the network. Its non-interactive and efficient storage properties make it attractive for many real applications.

Compared with the generalized Diffie-Hellman protocol and Harn-Gong protocol, it is

clear that Burmester-Desmedt protocol requires all parties to participate in the key establishment process whereas the Harn-Gong protocol is non-interactive. At the same time, the Harn-Gong protocol can easily accommodate the addition and deletion of members or subsets, while there is a significant computation cost for the user updates in the Burmester-Desmedt protocol.

2.3 Homomorphic Encryption

Homomorphic encryption (HE) is a type of encryption that allows computations on encrypted data. When the operated ciphertext is decrypted, the result matches operation as if the operation were performed on the plaintext. HE was first proposed in [49] in 1978 as a possible solution to the problem that most encrypted data needs to be decrypted if it needs complicated operation by a third party.

In order to perform homomorphic evaluation of an arbitrary function, it is sufficient for HE to allow only addition and multiplication operations because the two operations are functionally complete for any arithmetic operations over a field [49]. HE can be divided neatly into three categories, which are

1) Partially Homomorphic Encryption (PHE)

2) Somewhat Homomorphic Encryption (SWHE)

3) Fully Homomorphic Encryption (FHE)

PHE is a kind of HE that allows only one type of operation to be performed an unlimited number of times. For example, the basic RSA public key scheme [50] is a kind of PHE that achieves homomorphic multiplication. When given an RSA modulus N and public key e, since the encryption method for a message m is $E(m) = m^e \mod N$, for any two different messages m_1 and m_2, $E(m_1) * E(m_2) = m_1^e * m_2^e \pmod N = (m_1 m_2)^e \pmod N = E(m_1 m_2) \pmod N$. Besides the RSA public key scheme, there are many other useful PHE schemes such as Goldwasser and Micali [52], ElGamal [21], Benaloh [3] and Paillier [46]. Although PHE can only achieve one type of operation, these schemes were an important basis for subsequent HE research.

SWHE is a type of HE that allows a limited set of operations to be applied a limited number of times. An early attempt of computation over encrypted data is Yao's study in 1986 [61] which solves the Millionaire's problem by comparing the wealth of two rich people without revealing the exact amount to each other. However, the time and space complexity of Yao's algorithm is exponential. It was not until 2005 that Boneh-Goh-Nissim (BGN) introduced one of the most significant SWHE schemes [11]. BGN enables an arbitrary number of additions and one multiplication while keeping the ciphertext size constant. Although BGN only allows small message space and a limited number of multiplications, it was an important step towards FHE.

FHE allows an unlimited number of operations to be applied an unlimited number of times, and it is the ultimate objective of HE research. It was not until 2009 that Gentry proposed the first plausible FHE scheme in [24]. Gentry's scheme is based on ideal lattices and it led to other FHE designs [57, 13, 25]. One year after Gentry's work, Van Dijk *et al.* [57] constructed an FHE scheme over the integers from a "bootstrappable" SWHE by Gentry's method. Brakerski *et al.* [13] then proposed an FHE scheme based on learning with error (LWE) problem, which is considered as one of the hardest problems to solve even for even by quantum algorithms.

Homomorphic encryption is an ideal solution to data privacy problems as it allows cloud service providers to operate on encrypted data without decrypting it. After Gentry's work [24], many different FHE schemes were proposed. FHE still needs to be improved substantially to be practical because the current schemes are too computationally expensive for real-life big data applications.

Chapter 3

Preliminaries

In this chapter, to provide the ingredients of our data aggregation scheme, we introduce stream ciphers in Section 3.1, Shamir's secret sharing scheme in Section 3.2, and Diffie-Hellman key establishment protocol in Section 3.4. Section 3.3 describes Horner's method for evaluating polynomials. The notation we use in the protocol is presented in Section 3.5.

3.1 Stream Cipher

We first take a look at the One-time Pad (OTP) which has perfect secrecy [54] and simple encryption and decryption methods.

Definition 3.1.1 (One-time Pad)**.** A one-time pad is a cipher where key k, message m and ciphertext c are bit strings of the same length. The encryption method is $E(k, m) = k \oplus m$ and decryption method is $D(k, c) = k \oplus c$, where $\oplus$ denotes the exclusive-or operation.

The reason that the one-time pad can be encrypted/decrypted in such a way is that the exclusive-or of any message with itself is 0. The operations are as follows:

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

To achieve perfect secrecy, Shannon [54] proved that the key should be at least as long as the message. At the same time for a secure OTP scheme, the key cannot be used twice, which makes the OTP impractical in daily usage.

Stream ciphers overcome this problem by using a relatively short key to generate long keystream and then encrypt data in the OTP manner. The function to generate a long keystream from a short-length seed is called Pseudo-Random Number Generator (PRNG). Stream ciphers are widely used in modern technology because of their efficient encryption and decryption operations. AES Counter Mode, depicted in Figure 3.1, is a model for using the AES block cipher to construct a stream cipher.

3.2 Shamir's (k, n) Secret Sharing Scheme

Shamir's secret sharing scheme [53] is a method for distributing a secret over n members, each of whom is allocated a share of the secret, and the secret can only be reconstructed by up to k members. Shamir's secret sharing scheme is based on Lagrange interpolation, and the secret is information theoretically secure if fewer than k shares are collected [47].

Suppose a secret s is to be shared by a group of n members, in such a way that at least k shares are required to reconstruct s. Let the secret share be a y value, and each member's public identity be an x value. The shares and each member's identity can be treated as a set of points $(x_1, y_1), \cdots, (x_n, y_n)$ on a 2-dimensional plane.

According to polynomial interpolation, when provided k pairs of points $(x_i, y_i), 0 \leq i \leq k-1$, there is one and only one polynomial $p(x)$ of degree at most $k-1$ such that $p(x_i) = y_i$ for all i. After the polynomial is reconstructed, the secret can be selected as $p(0)$ which makes it reconstructable by all group members.

If we implement Shamir's secret sharing scheme in a finite field $\mathbb{Z}_q$, where q is of l bits, the advantage of an adversary to reconstruct the secret by knowing fewer that k shares is $\frac{1}{2^l}$. Generally, the usage of Shamir's secret sharing scheme can be divided into two steps:

1) Secret Share Generation.

2) Secret Reconstruction.

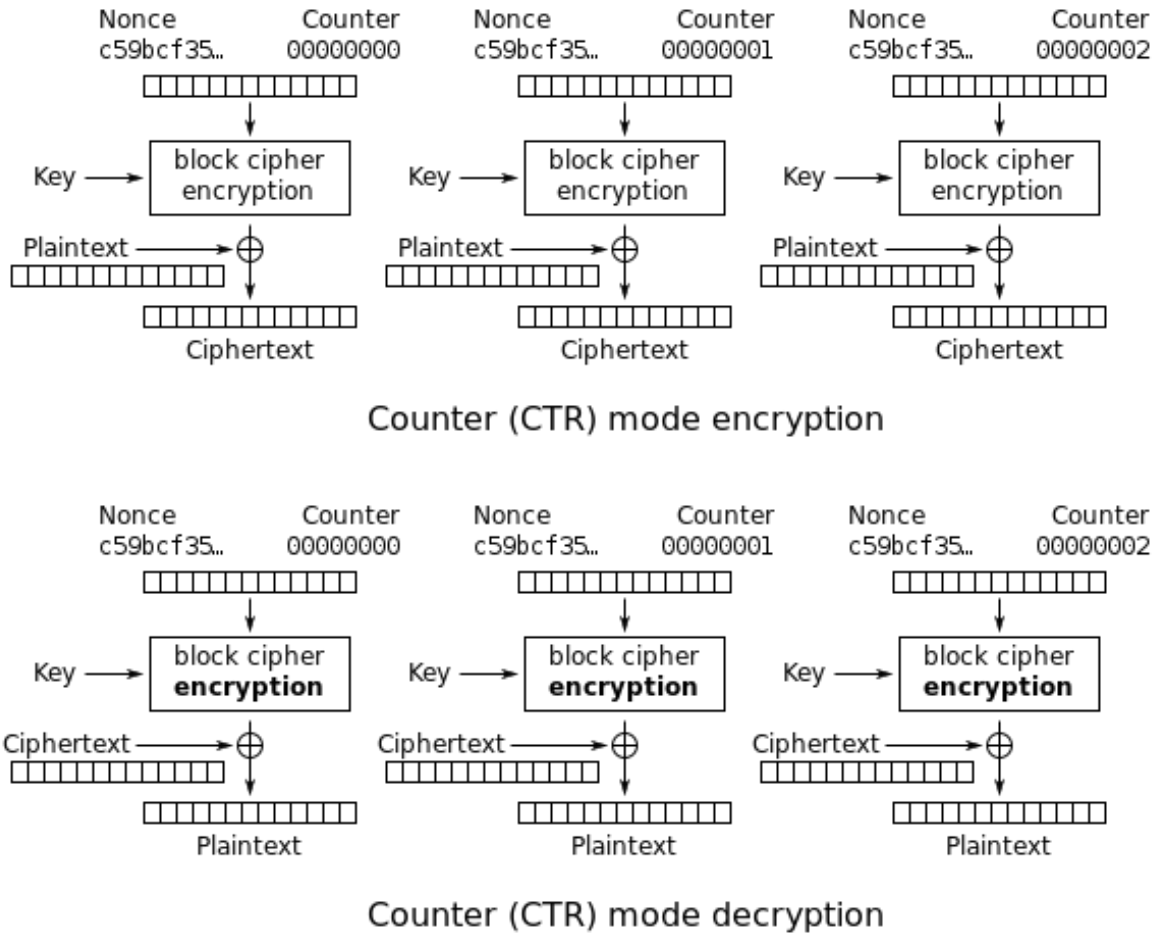


Figure 3.1: AES Counter Mode

3.2.1 Secret Share Generation

1. s is selected as a secret to be shared over field $\mathbb{Z}_q$.
2. For $1 \leq i \leq k-1, i \in \mathbb{N}$, generate $p_i \xleftarrow{R} \mathbb{Z}_q$.
3. Construct a polynomial $p(x) = s + p_1 x + p_2 x^2 + \cdots + p_{k-1} x^{k-1}$.
4. For $1 \leq i \leq n$, evaluate $y_i = p(x_i)$ over $\mathbb{Z}_q$, and distribute y_i to member i with identity $x_i \in \mathbb{Z}_q$.

3.2.2 Secret Reconstruction

1. k shares and corresponding member identities are collected which is denoted as $(x_1, y_1), \cdots, (x_k, y_k)$.

2. Construct a polynomial $p(x)$ where

$$p(x) = \sum_{i=1}^{k} \prod_{1 \le j \le k, j \ne i} \frac{x - x_j}{x_i - x_j} y_i$$

3. The secret s is the value of $p(0)$.

$$\begin{aligned} s &= p(0) \\ &= \sum_{i=1}^{k} \prod_{1 \le j \le k, j \ne i} \frac{-x_j}{x_i - x_j} y_i \end{aligned}$$

Lemma 3.2.1 (Shamir's Secret Sharing Reconstruction)**.** *For a (k, n) Shamir's secret sharing scheme with over k different shares, the secret polynomial $p(x)$ and secret $p(0)$ can be reconstructed as described above.*

3.3 Horner's Method

There are many algorithms to evaluate polynomials efficiently, among which Horner's method uses k multiplications and k additions to evaluate a polynomial of degree k. The technique is as follows:

1) Given a polynomial $p(x) = \sum_{i=0}^{k-1} a_i x^i$, where $a_0, \cdots, a_k \in \mathbb{Z}_q$ and $p(x_0)$ is the value to be evaluated.

2) A new sequence of constants are defined as follows:

$$\begin{aligned} b_k &= a_k \\ b_{k-1} &= a_{k-1} + b_k x_0 \\ &\vdots \\ b_0 &= a_0 + b_1 x_0 \end{aligned}$$

Then b_0 is the value of $p(x_0)$.

Note that the reason the scheme works is that the polynomial calculation can be written in the form

$$\begin{aligned} p(x_0) &= a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{k-1} + a_k x_0))) \\ &= a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(b_{k-1}))) \\ &\vdots \\ &= a_0 + x_0(b_1) \\ &= b_0 \end{aligned}$$

Clearly the complexity of the algorithm is k multiplications and k additions for calculation of k constants of b_i, where $0 \leq i \leq k-1$.

3.4 Diffie-Hellman Protocol

The Diffie-Hellman protocol is a cryptographic key agreement scheme proposed by Whitefield Diffie and Martin Hellman in 1976 [20]. The protocol is the earliest practical example of public key exchange and is widely used in information technology. Asides from the original version of Diffie-Hellman key agreement protocol, the elliptic curve version [1] is also widely used due to its efficiency at the same security level compared to the original one.

The basic protocol works like this. Suppose Alice and Bob want to establish a shared key over a group $\mathbb{G}$ of order q, where q is prime. Let g be a generator of $\mathbb{G}$, the Diffie-Hellman protocol against a passive adversary is in Figure 3.2.

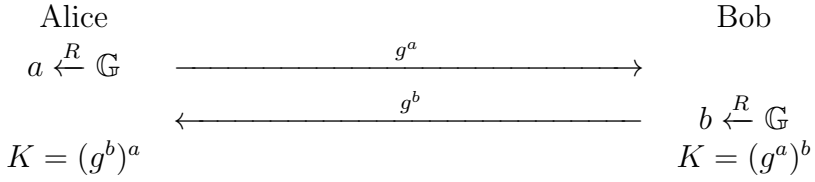


Figure 3.2: Standard Diffie-Hellman Protocol Against a Passive Adversary

Since the key $K = (g^b)^a = g^{ab} = (g^a)^b$, Alice and Bob can successfully establish a shared key . The security of Diffie-Hellman protocol is based on the assumption of Decision Diffie-Hellman problem (DDH) [9], which makes a passive adversary unable to differentiate a shared key established by Alice and Bob from a random element of $\mathbb{G}$.

Definition 3.4.1 (Decision Diffie-Hellman Assumption)**.** The two tuples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c), where g is a generator of group $\mathbb{G}$ of order q and $a, b, c \xleftarrow{R} \mathbb{G}$, are computationally indistinguishable.

The security level of the size of group and elliptic curve is listed in Table 3.1 from [1].

Table 3.1: Group Sizes for Different Security Levels

Security level	80	112	128
Bit Length of p when $\mathbb{G} = \mathbb{Z}_p*$	1024	2048	3072
Bit Length of p when $\mathbb{G} = E(\mathbb{Z}_p)$	160	224	256

3.5 Notation

In this section, we introduce the notation for protocol specification and computation complexity which will be used throughout the thesis.

3.5.1 Protocols

The notation in Table 3.2 is for the key agreement protocols, the encryption protocols, and the cryptographic primitives used in the protocol description.

3.5.2 Computation Complexity

The notation in Table 3.3 is for the complexity analysis using bit operations in a scheme under analysis.

Table 3.2: Notation for Protocol Description

Notation	Description
$KA.param(k) \rightarrow pp$	An algorithm that produces some public parameters over which the key agreement protocol is used
$KA.gen(pp) \rightarrow (s_u^{SK}, s_u^{PK})$	An algorithm that allows any party u to generate a private-public key pair
$KA.agree(s_u^{SK}, s_v^{PK}) \rightarrow s_{u,v}$	An algorithm that allows any user u to combine their private key s_u^{SK} with the public key s_v^{PK} for any v (generated using the same parameter pp), to obtain a private shared key $s_{u,v}$ between u and v
$SIG.sign(d^{SK}, m) \rightarrow \sigma$	An algorithm that takes as input the secret key d^{SK} and a message m and outputs a signature σ
$SIG.ver(d^{PK}, m, \sigma) \rightarrow \{0, 1\}$	An algorithm that takes as input a public key d^{SK}, a message m and a signature σ, and returns a bit indicating whether the signature should be considered valid or not
$PRNG(b) \rightarrow \mathbf{v}$	A pseudo-random number generater that generates a data vector $\mathbf{v}$ from a number b
$SS.share(s, t, \mathcal{U}) \rightarrow \{(u, s_u)\}_{u \in \mathcal{U}}$	An algorithm that takes as input a secret s, a set $\mathcal{U}$ of n field elements representing user IDs, and a threshold $t \leq \lvert\mathcal{U}\rvert$; it produces a set of shares s_u , each of which is associated with a different $u \in \mathcal{U}$
$SS.recons(\{(u, s_u)\}_{u \in \mathcal{V}}, t) \rightarrow s$	An algorithm that takes as input the threshold t and the shares corresponding to a subset $\mathcal{V} \subseteq \mathcal{U}$ such that $\lvert\mathcal{V}\rvert \geq t$, and outputs a field element s
$SS.reconp(\{(u, s_u)\}_{u \in \mathcal{V}}, t) \rightarrow p(x)$	An algorithm that takes as input the threshold t and the shares corresponding to a subset $\mathcal{V} \subseteq \mathcal{U}$ such that $\lvert\mathcal{V}\rvert \geq t$, and outputs a field polynomial $p(x)$ of degree $t-1$
$AE.enc(k, m) \rightarrow c$	An authenticated encryption algorithm that takes as input a key k and a message m and outputs a ciphertext c
$AE.dec(k, c) \rightarrow \{m, \perp\}$	An authenticated decryption algorithm that takes as input a ciphertext c and a key k and outputs either the original plaintext m, or a special error symbol $\perp$

Table 3.3: Notation for Complexity Analysis

Notation	Description
$Exp(l)$	The number of bit operations of the exponentiation of a number of bit-length l
$Inv(l)$	The number of bit operations of the inversion of a number of bit-length l
$Mul(l)$	The number of bit operations of the multiplication of two numbers of bit-length l
$Add(l)$	The number of bit operations of the addition of two numbers of bit-length l
$Sign(l)$	The number of bit operations of signing a digital signature of a message of bit-length l
$Enc(l)$	The number of bit operations of the encryption of a message of bit-length l
$Srand(l)$	The number of bit operations of a secure random number generation of bit-length l

3.5.3 Finite Fields

The notation in Table 3.4 is for the finite field over which a polynomial evaluation and a secure random number generation are performed.

Table 3.4: Notation for Finite Field

Notation	Description
$\mathbb{Z}_n$	The set $\{0, 1, \cdots, n-1\}$
$\mathbb{G}$	A finite group where Diffie-Hellman parameters are defined
$\mathbb{F}$	A finite field where secret share coefficients are defined
$\mathbb{R}$	A finite field where masking operation is executed
$s \xleftarrow{R} \mathbb{F}$	A random number s that is generated over field $\mathbb{F}$

Chapter 4

Harn-Gong Group Key Establishment Protocol and its Security Proof

In this chapter, we introduce a non-interactive protocol for establishing keys for each pair of users in a network. We present the protocol overview in Section 4.1. The details of the protocol description are presented in Section 4.2 and a new security proof is demonstrated in Section 4.3. Finally, we analyze the bit complexity of the protocol in Section 4.4.

4.1 Scheme Overview

A conference key is a secret shared by a group of communication members, the number of which is usually greater than or equal to two. Since Diffie and Hellman [20] described a new concept in cryptography called the public key distribution, two-party key agreement protocols have been well researched. Plenty of efficient and secure schemes have been proposed over the years [39, 19, 23]; however, to successfully generate a shared secret over a group in the network, massive communication overhead remains if we want to inherit the security properties of two-party protocols. For example, if we want to establish a conference key for six users, the communication overhead is demonstrated in Figure 4.1.

In 1992, Blundo *et al.* [6] proposed a non-interactive k-secure m-conference protocol based on multivariate polynomials, where k-secure meaning any k users cannot collude

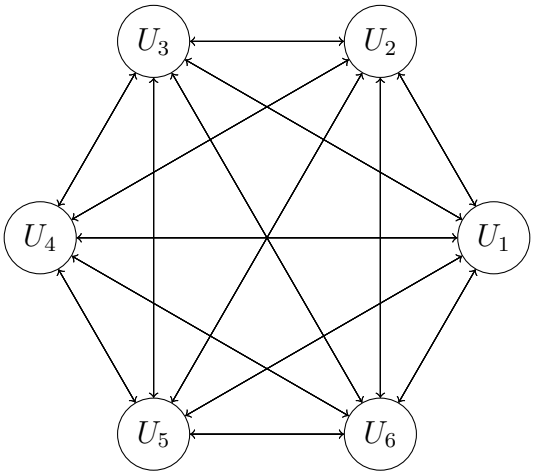


Figure 4.1: 6 Users Key Establishment Overhead Using 2 Party Key Agreement Protocol

to get other member's share information, and m-conference means that the protocol can establish a conference key for up to m users. However, since the number of coefficients of the polynomial is exponential in the number of users in the group, the protocol requires a large storage space. In 2015, Harn and Gong proposed a conference key establishment protocol [28] that significantly reduces the storage space of the polynomial coefficients, based on which we proposed our communication efficient data aggregation scheme.

A *Key Distribution Center (KDC)* is a component of a cryptosystem to reduce risks inherent in key exchange. The Harn-Gong protocol needs a *KDC* to compute each member's secret share polynomial first, and then sends it to each user secretly. The *KDC* can be offline once it successfully sends each user their secret share, which avoids the single point attack traditionally prevalent in a cryptosystem that employs a trusted third party. The use of *KDC* can significantly reduce communication overhead in the protocol. Also, the *KDC* can be implemented in a distributive way to avoid a single point trust problem, and the details of the approach are presented in [10]. For example, a group communication for 6 users employing KDC is presented in Figure 4.2.

Any group of up to m members can establish a group key by the following steps:

1) Univariate Polynomial Generation

2) Secret Share Polynomial Generation

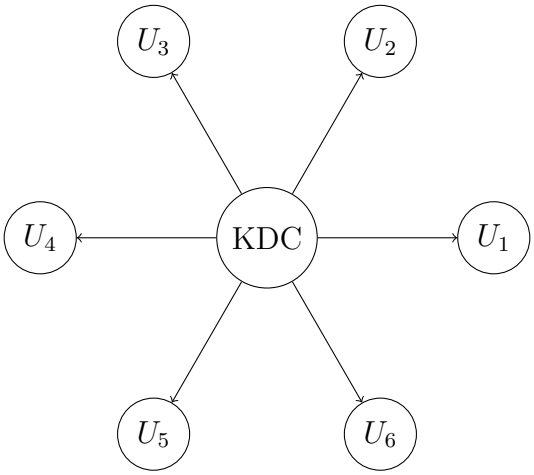


Figure 4.2: 6 Users Key Establishment Overhead Using Harn-Gong Protocol

3) Conference Key Generation

4.2 Conference Group Key Sharing Protocol

We use the Harn-Gong protocol to establish a k-secure m-members n-users conference key. Let k, m and n be some positive integers, where m is an even number and $m \leq n$. $\mathcal{U}_1$ denotes the network where the KDC sends secret polynomial to users, and $\mathcal{U}_2$ denotes the network where the group key is established. For a group consisting of n users, a k-secure m-members n-users conference key establishment protocol has the following properties:

1) The protocol can resist collusion by up to k users.

2) The protocol can establish a conference key for any r members where $r \leq m$.

The protocol is described in the following three subsections.

4.2.1 Univariate Polynomial Generation

KDC:

(a) Generate an RSA modulus N where N is a product of two large safe primes p and q. Keep p and q secret and broadcast N to the network $\mathcal{U}_1$.

(b) Select a suitable even number m such that $m \leq n$ where $|\mathcal{U}_1| = n$, and select a security parameter k as the degree of the secret univariate polynomial, $f(x)$.

(c) For $0 \leq i \leq k$, securely generate a random number $a_i \xleftarrow{R} \mathbb{Z}_N$ as the coefficient of polynomial $f(x) = a_k x^k + \cdots + a_1 x + a_0 \mod N$. We call $f(x)$ the master secret polynomial.

4.2.2 Secret Share Polynomial Generation

KDC:

(a) For each user $u \in \mathcal{U}_1$, compute $s_u(x) = f(u)^{\frac{1}{m-1}} f(x) \mod N$, where $u \in \mathbb{Z}_N$ is a public information associated with the user, and $\frac{1}{m-1}$ is the inverse of $m-1$ in $\mathbb{Z}_{\phi(N)}$ where ϕ is Euler's totient function.

(b) For each user $u \in \mathcal{U}_1$, send the secret share polynomial $s_u(x)$ to user u over a secure channel.

4.2.3 Conference Key Generation

User u:

(a) A group of users $\mathcal{U}_2$ including u wants to establish a conference key for group communication, where $|\mathcal{U}_2| = r \leq m$.

(b) Generate a group key $k_{\mathcal{U}_2}$ where

$$\begin{aligned} k_{\mathcal{U}_2} &= \prod_{v \in \mathcal{U}_2, v \neq u} s_u(v) \times s_u^{m-r}(0) \mod N \\ &= \prod_{v \in \mathcal{U}_2} f(v) \times f^{m-r}(0) \mod N \end{aligned}$$

4.3 Security Analysis

Lemma 4.3.1. *The inverse of $m-1$ in $\phi(N)$ always exists when m is an even number in the protocol.*

We first introduce the concept of a safe prime.

Definition 4.3.1 (Safe Prime)**.** A safe prime is a prime number p of the form $p = 2q + 1$, where q is also a prime.

Proof of Lemma 4.3.1. The Euler totient function of the modulus N is $\phi(N) = (p-1)(q-1) = 4p'q'$ where p' and q' are primes that makes $p = 2p' + 1$ and $q = 2q' + 1$. When m is even, the greatest common divisor (GCD) of $m - 1$ and $\phi(N)$ is 1 since $m - 1$ is an odd number, $\phi(N)$ is an even number, $m < p'$ and $m < q'$.

Since $GCD(m-1, \phi(N)) = 1$, from Euclidean algorithm there exists a number $z \in \mathbb{Z}_{\phi(N)}$ that

$$z \cdot (m-1) \equiv 1 \mod \phi(N)$$

Thus the inverse of $m - 1$ in $\phi(N)$ always exists when m is an even number in the protocol. □

Lemma 4.3.2. *The protocol can establish a secure conference key for any conference having r ($r \leq m$) members.*

Proof. From Section 4.2.3, suppose $\mathcal{U}_1 = \{i_1, i_2, \cdots, i_r\}$ and user $u = i_j$. Then the conference key $k_{\mathcal{U}_1}$ of the network $\mathcal{U}_1$ is

$$\begin{aligned}
k_{\mathcal{U}_1} &= \prod_{\substack{1 \leq l \leq r \\ l \neq j}} s_{i_j}(i_l) \times s_{i_j}^{m-r}(0) \mod N \\
&= \prod_{\substack{1 \leq l \leq r \\ l \neq j}} f(i_j)^{\frac{1}{m-1}} f(i_l) \times f(i_j)^{\frac{m-r}{m-1}} f^{m-r}(0) \mod N \\
&= \prod_{\substack{1 \leq l \leq r \\ l \neq j}} f(i_l) \times f(i_j)^{\frac{r-1}{m-1}} \times f(i_j)^{\frac{m-r}{m-1}} f^{m-r}(0) \mod N \\
&= \prod_{1 \leq l \leq r} f(i_l) \times f^{m-r}(0) \mod N
\end{aligned}$$

□

Definition 4.3.2 (RSA Assumption)**.** It is computationally infeasible to compute M given only the ciphertext $C = M^e \mod N$ and RSA public key (N, e)

In the following, we first introduce the result on security analysis of the protocol from [28].

Lemma 4.3.3 ([28]). *The proposed scheme is secure under the RSA assumption to resist the attack by a single user with one secret share polynomial to solve the master secret polynomial, $f(x)$, used to generate shares.*

We will present a proof of a different security claim, since the original proof has some flaws. In order to exhibit the flaw, in the following, we first present the security proof of Lemma 4.3.3 in [28].

Proof of Lemma 4.3.3 in [28]. Suppose there is an efficient algorithm, $\mathcal{A}$, that could factor a secret share polynomial $s_u(x)$ to gain KDC's master secret polynomial $f(x)$ such that $s_u(x) = f(u)^s f(x)$ where u and s are at his/her wish.

Then an attacker could use this algorithm to decrypt any ciphertext $C = M^e \mod N$ efficiently by the following steps where (N, e) is user u's RSA public key:

1. Let $C = (c_k, c_{k-1}, \ldots, c_0)_{10} \in \mathbb{Z}_N$ where $C = c_k 10^k + c_{k-1}10^{k-1} + \cdots + c_0 \mod N$.
2. Let $g(x)$ be a polynomial given by $g(x) = c_k x^k + c_{k-1}x^{k-1} + \cdots + c_0 \mod N$. Thus the ciphertext can be represented through the polynomial $g(x)$, i.e., $C = g(10)$.
3. Attacker models the RSA encryption in terms of the polynomial as $g(x) = f(u)^{e-1} f(x)$ where $u = 10$, then the adversary use algorithm $\mathcal{A}$ to factor $g(x)$ to get $f(x)$.
4. Attacker will have $g(10) = f(10)^e \pmod N = C = M^e \pmod N$, which implies $M = f(10) \pmod N$. Thus attacker could decrypt ciphertext C.

This contradicts the RSA assumption. Thus [28] concludes that such algorithm $\mathcal{A}$ does not exist. □

However, when using the contradiction to prove the lemma, there is a situation where there does not exist a polynomial $f(x)$ such that $g(x) = f(10)^{e-1} * f(x) \mod N$. For example, when $N = 187 = 11 * 17$, $e = 3$, and the ciphertext is 11, then we get $g(x) = 1 + x$. Nevertheless, there is no polynomial $f(x) = a + bx$ such that $g(x) = f(10)^2 * f(x) \mod N$. The problem in this case is that ciphertext 11 is a factor of RSA modulus ($187 = 11 * 17$). There is also no plaintext m such that makes $m^e = 11 \mod N$. Thus there is no way to

construct a polynomial that contradicts the RSA assumption.

In the original proof, knowing single share attack is an attack where the attacker is trying to acquire the master secret polynomial in the network. However, if the attacker can obtain other user's secret polynomial at his/her wish, the attacker can obtain any conference key in the network to compromise the security. As a result we change the definition of knowing single share attack to that the attacker is trying to acquire other user's secret polynomial but not necessarily the master secret polynomial.

Let (N, e) be an RSA public key where N is the RSA modulus and e is the public key. Let u, v be public information related to the users. Recall that $s_u(x) = f(u)^{\frac{1}{e}} f(x)$ and $s_v(x) = f(v)^{\frac{1}{e}} f(x)$. Let $KSSA$ denote a knowing single share attack problem where there is an algorithm $\mathcal{A}$ that takes input $(s_u(x), N, e, u, v)$ and output $s_v(x)$. Hence $\mathcal{A}(s_u(x), N, e, u, v) = s_v(x)$. Let $RSAP$ denote the RSA problem, and suppose there is an algorithm $\mathcal{A}'$ that takes input (N, e, M^e) and output M respectively where M is a random value from $\mathbb{Z}_N$. Hence $\mathcal{A}'(N, e, M^e) = M$. The new lemma and proof are as follows:

Lemma 4.3.4. *The proposed scheme is secure under the RSA assumption to resist the knowing single share attack by an attacker with one secret share polynomial $s_u(x)$ to obtain other user's secret polynomial $s_v(x)$ where v, u are related to users' public information and $v \neq u$.*

Proof. Let $KSSA$ denote a knowing single share attack problem where there is an algorithm $\mathcal{A}$ that takes input $(s_u(x), N, e, u, v)$ and output $s_v(x)$ where $s_u(x) = f(u)^{\frac{1}{e}} f(x)$, $s_v(x) = f(v)^{\frac{1}{e}} f(x)$, N is an RSA modulus and u, v are public information related to the users. Hence $\mathcal{A}(s_u(x), N, e, u, v) = s_v(x)$. Let $RSAP$ denote the RSA problem where there is an algorithm $\mathcal{A}'$ that takes input (N, e, M^e) and output M respectively. Hence $\mathcal{A}'(N, e, M^e) = M$.

We can reduce $KSSA$ problem to $RSAP$ problem by the following process:

1. Given $s_u(x), N, e, u, v$, let $\rho = s_u(v)/s_u(u) = (f(u)^{\frac{1}{e}} f(v))/(f(u)^{\frac{1}{e}} f(u)) = f(v)/f(u) \pmod N$.
2. Query $RSAP$ to calculate $\mathcal{A}'(N, e, \rho) = \rho^{\frac{1}{e}} \pmod N = f(v)^{\frac{1}{e}}/f(u)^{\frac{1}{e}} \pmod N$.
3. Obtain $s_v(x)$ by calculating $s_v(x) = \rho^{\frac{1}{e}} s_u(x) \pmod N$, since $\rho^{\frac{1}{e}} s_u(x) = (f(v)/f(u))^{\frac{1}{e}} f(u)^{\frac{1}{e}} f(x) = f(v)^{\frac{1}{e}} f(x) \pmod N$.

Since we can use algorithm $\mathcal{A}'$ to output $s_v(x)$ from $(s_u(x), N, e, u, v)$, we know that $KSSA \leq_p RSAP$.

Now we show how to reduce $RSAP$ to $KSSA$ by the following steps:

1. Given (N, e, C) where $C = M^e \pmod N$, let $m_0 \xleftarrow{R} \mathbb{Z}_N$ and $c_0 = m_0^e \pmod N$.
2. Let $c_1, \cdots, c_{t-1} \xleftarrow{R} \mathbb{Z}_N$, find c_t such that $\sum_{i=0}^{t} c_i = C \pmod N$.
3. Let $f(x) = \sum_{i=0}^{t} c_i x^i \pmod N$ such that $f(0) = c_0$ and $f(1) = C$. Let $s_0(x) = m_0 f(x) \pmod N$.
4. Query $KSSA$ to get $\mathcal{A}(s_0(x), N, e, 0, 1) = s_1(x) \pmod N$.
5. Since $s_1(x) = f(1)^{\frac{1}{e}} f(x) = C^{\frac{1}{e}} f(x) = Mf(x) \pmod N$, from $s_0(x)$ and $s_1(x)$, by setting $x = 0$, $\frac{s_v(0)}{s_u(0)} = \frac{Mf(0)}{m_0 f(0)} = \frac{M}{m_0} \pmod N$, then $M = m_0 \frac{s_v(0)}{s_u(0)} \pmod N$.

Since we can use algorithm $\mathcal{A}$ to output eth root of C from (N, e, M^e), we know that $RSAP \leq_p KSSA$.

From the two steps above, we conclude that $KSSA \approx RSAP$, which means that knowing a single share polynomial $(s_u(x), N, m, u, v)$, it is computationally infeasible to compute other user's secret polynomial $s_v(x)$ as long as $RSAP$ is a hard problem. □

From Lemma 4.3.4, we know that by knowing single secret polynomial one cannot use other user's secret polynomial to obtain any other conference keys. However, since the conference key is constructed by the production of the master secret polynomial $f(x)$, one can also obtain other conference keys by learning $f(x)$. We argue next that it is also computationally infeasible to determine $f(x)$ from a single secret polynomial.

Claim 4.3.1. By knowing a single secret polynomials $s_u(x)$ one cannot learn the master secret polynomial $f(x)$ used to generate conference keys.

Justification. Let $m - 1$ denoted as e. Suppose the user knows $f(u)$. Since $s_u(x) = f(u)^{\frac{1}{e}} f(x)$, if one can determine $f(x)$, then one can compute $f(u)^{\frac{1}{e}} \mod N$, which would be hard due to RSA assumption. Thus it is computationally infeasible to determine $f(x)$ from a single secret polynomial $s_u(x)$. □

Claim 4.3.2. By knowing t secret polynomials $s_{u_i}(x), 1 \leq i \leq t$ where $t < k+1$, one cannot learn the conference key $k_{\mathcal{U}}$ where $u_i \notin \mathcal{U}$ for all $1 \leq i \leq t$.

Justification. The proof is based on the well-known Lagrange interpolation formula. Suppose $\mathcal{U} = \{v_1, v_2, \cdots, v_r\}$ and $v_j \xleftarrow{R} \mathcal{U}$. Let $g(x) = \prod_{1 \le l \le r, l \ne j} f(v_l) \times f^{m-r}(0) f(x)$, then the conference key $k_{\mathcal{U}} = g(v_j)$.

By knowing a single secret polynomial $s_{u_i}(x)$, we can obtain the value of $g(u_i)$ by calculating $\prod_{1 \le l \le r, l \ne j} s_{u_i}(v_l) \times s_{u_i}^{m-r}(0) = \prod_{1 \le l \le r, l \ne j} f(v_l) \times f^{m-r}(0) f(u_i) = g(u_i) \mod N$. We can obtain t points $(u_i, g(u_i))$ by t secret polynomials $s_{u_i}(x), 1 \le i \le t$.

Since $g(x)$ is with degree k, according to the Lagrange interpolation formula, it needs at least $k+1$ or more than $k+1$ points to construct $g(v_j)$. By knowing t secret polynomials $s_{u_i}(x), 1 \le i \le t$ where $t < k+1$, it is computationally infeasible reconstruct $g(x)$. Hence one cannot learn the conference key $k_{\mathcal{U}}$ where $\forall u_i \notin \mathcal{U}, 1 \le i \le t$.

□

Claim 4.3.3. It is computationally infeasible for an adversary to know other conference keys by knowing fewer than $k+1$ secret polynomials.

Justification. From Lemma 4.3.4, we can conclude that it is computationally infeasible for an adversary to know other secret polynomials by knowing a single secret polynomial. From Claim 4.3.2, we also know that it is computationally infeasible to obtain other conference keys by knowing fewer than $k+1$ secret polynomials. Since one can neither obtain other conference keys by learning others' secret polynomials nor reconstructing conference keys by knowing fewer than $k+1$ secret polynomials, it is computationally infeasible for an adversary to know other conference keys by knowing less than $k+1$ secret polynomials. □

4.4 Complexity Analysis

Cryptographic primitives are expensive both in storage and computation, for example, the operation of cryptographic primitives might involve integers of 128 bytes or more (considering an RSA modulus of 1024 bits). To illustrate the time and space complexity in more detail, we analyze the bit complexity of computation and storage.

In the analysis, we use the notation from Tables 3.3 and 3.4. The l bits storage space can store a 2^l number. Since the operation of finding an inverse a number is equal to do an Euclidean algorithm to find the greatest common divisor of the number and modulus, $Inv(l)$ is proportional to $Mul(l)$.

4.4.1 Complexity Analysis of User

Time Complexity :

(a) In Sections 4.2.1 and 4.2.2 there is no computation needed.

(b) In Section 4.2.3, for an k-secure m-members conference key establishment process, each user needs to evaluate the secret polynomial m times and multiply evaluated value m times. With the secret polynomial of degree k, from Horner's method it takes $k(Mul(\log N) + Add(\log N))$ operations for evaluation. Thus the user needs $m(k(Mul(\log N) + Add(\log N))) + mMul(\log N)$ operations to establish a m-users conference key.

Space Complexity:

(a) In Sections 4.2.1 and 4.2.2 there is no storage needed.

(b) In Section 4.2.3, the user needs $(k+1)\log(N)$ space to store secret polynomial coefficients.

Communication Complexity:

(a) In Sections 4.2.1 and 4.2.3, there is no communication needed.

(b) In Section 4.2.2, the user needs to receive one message of secret polynomial.

4.4.2 Complexity Analysis of *KDC*

Time Complexity :

(a) In Section 4.2.1, KDC needs to generate $(k+1)$ polynomial coefficients which needs $(k+1)Srand(\log N)$ operations to generate a univariate polynomial.

(b) In Section 4.2.2 KDC needs to do n evaluations, n modulo inverse and $(k+1)$ multiplications for each user in $\mathcal{U}_1$. From Horner's Method we know that KDC needs $n(k(Mul(\log N) + Add(\log N)) + Inv(\log N)) + (k+1)Mul(\log N))$ operations in time.

(c) In Section 4.2.3 there is no computation needed.

Table 4.1: Complexity Analysis of Harn-Gong Key Establishment Protocol in Bit Operation

	Time Complexity	Space Complexity	Communication Complexity
KDC	$O(kSrand(\log N)+$ $nkMul(\log N)+$ $nkAdd(\log N))$	$O(k \log N)$	$O(n)$
user	$O(mkMul(\log N)+$ $mkAdd(\log N))$	$O(k \log N)$	$O(1)$

Space Complexity:

(a) In Section 4.2.1, the KDC needs $(k+1)\log(N)$ space to store $(k+1)$ polynomial coefficients.

(b) In Section 4.2.2, the KDC needs $(k+1)\log(N)$ space to store temporary secret polynomial for each user.

(c) In Section 4.2.3, there is no storage needed.

Communication Complexity:

(a) In Sections 4.2.1 and 4.2.3, there is no communication needed.

(b) In Section 4.2.2, the KDC needs to send the secret polynomial to each user, which means KDC needs to send n messages in total.

In conclusion, the time, space and communication complexity of the KDC and the user in bit operations are listed in Table 4.1.

Chapter 5

Mobile Data Aggregation for Privacy-Preserving Machine Learning

In this chapter, we present a data aggregation scheme for privacy-preserving machine learning introduced in [7]. We present the scheme overview in Section 5.1. By revising the adversarial model, the details of the scheme and its privacy analysis are presented in Section 5.2. We analyze the bit complexity in Section 5.3.

5.1 Scheme Overview

Machine learning is a field of artificial intelligence that uses statistical techniques to give computers the ability to "learn" [51]. Since the 1990s, machine learning has flourished in tackling solvable problems of a practical nature, such as object detection in computer vision, natural language processing and bioinformatics. Standard machine learning approaches require centralizing a significant amount of training data on one machine or in a data center. In 2017, Google described a new type of machine learning scheme, called Federated Learning [36], that distributes the training data across different mobile users.

The general idea of Federated Learning is that each user as a device holder downloads the current training model from the cloud machine learning provider, stores customized training data in his/her devices, improves the model by using the new individualized training data, and then summarizes the changes as a small and focused update. Federated Learning enables better user experience from machine learning without compromising

users' confidentiality, which is often a concern from the public generated by the standard centralized AI paradigm, where better algorithms always come at the cost of collecting more and more personal data [2].

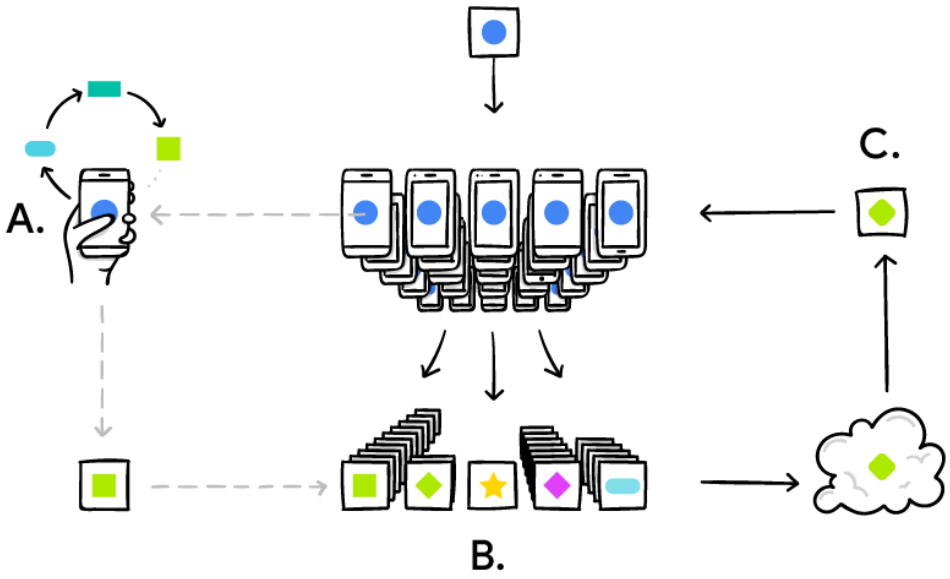


Figure 5.1: A user personalizes the model locally, based on customized usage (A). Many users' updates are aggregated (B) to form a consensus change. A shared model is distributed (C) after the consensus change. The procedure runs repeatedly.

A secure data aggregation protocol for Federated Learning has been proposed at the same time to reduce the risk of the cloud service provider learning the personal model update [7]. By the combination of cryptographic techniques and machine learning, the model enables the training algorithm update only by knowing the average data of 100s or 1000s users. With the evolution of other types of machine learning algorithms in the mobile device, this protocol can be applied to different situations as well.

For any mobile user who wants to send an update to the server, the process in [7] is as follows:

1) Diffie-Hellman Pairwise Key Generation

2) Secret Share Generation

3) Private Data Masking

4) User Drop-out Consistency Check

5) Unmasking Aggregate Data

5.2 Data Aggregation Protocol

The scheme provides an efficient way to calculate users' sums of vectors under the malicious model of a curious-but-honest server, and robustness to failures, which enables a practical and secure aggregation in mobile usage. However, the communication cost is expensive since drop-outs are common in mobile networks.

Also, the protocol in [7] is only guaranteed to be secure when the server is curious-but-honest and the adversary is only active in the communication channels of the network; this does not include a situation where the server is also an active adversary. Hence, in this chapter we consider the protocol in [7] against an active adversary as a protocol against a curious-but-honest adversary server and active users.

The requirements of the protocol are listed below:

1) All parties are given the security parameter k, the number of users n and a threshold value t, honestly generated $pp \leftarrow KA.param(k)$, parameters m and R such that $\mathbb{Z}_R^m$ is the space from which inputs are sampled, and a field $\mathbb{F}$ to be used for secret sharing.

2) All users also have a private authenticated channel with the server.

3) All users u receive their signing key d_u^{SK} from the trusted third party, together with verification keys d_v^{PK} bound to each user identity v.

If any of the below operations (assertion, signature verification, key agreement, encryption) fails, the protocol will abort. $\mathcal{U}_i, 1 \leq i \leq 4$ represents the set of online users in each round.

5.2.1 Diffie-Hellman Pairwise Key Generation

User u:

(1) Generate key pairs $(c_u^{PK}, c_u^{SK}) \leftarrow KA.gen(pp)$, $(s_u^{PK}, s_u^{SK}) \leftarrow KA.gen(pp)$, and generate $\sigma_u \leftarrow SIG.sign(d_u^{SK}, c_u^{PK}||s_u^{PK})$.

(2) Send $(c_u^{PK}||s_u^{PK}||\sigma_u)$ to the server through the private authenticated channel.

(3) Receive server's broadcast list $(v, c_v^{PK}, s_v^{PK}, \sigma_v)_{v\in\mathcal{U}_1}$. Assert that $|\mathcal{U}_1| \geq t$, that all the public key pairs are different, and that $\forall v \in \mathcal{U}_1, SIG.ver(d_v^{PK}, c_v^{PK}||s_v^{PK}, \sigma_v) = 1$.

(4) For each other user $v \in \mathcal{U}_1 \backslash \{u\}$, compute $k_{u,v} \leftarrow KA.agree(c_u^{SK}, c_v^{PK})$

(5) Store all messages received and values generated.

Server:

(1) Collect at least t messages from individual users in the previous round (denote with $\mathcal{U}_1$ this set of users). Otherwise, abort.

(2) Broadcast to all users in $\mathcal{U}_1$ the list $(v, c_v^{PK}, s_v^{PK}, \sigma_v)_{v\in\mathcal{U}_1}$.

5.2.2 Secret Share Generation

User u:

(1) Sample a random element $b_u \leftarrow \mathbb{F}$ to be used as a seed for a PRNG.

(2) Generate t-out-of-$|\mathcal{U}_1|$ shares of s_u^{SK}:$\{(v, s_{u,v}^{SK})\}_{v\in\mathcal{U}_1} \leftarrow SS.share(s_u^{SK}, t, \mathcal{U}_1)$

(3) Generate t-out-of-$|\mathcal{U}_1|$ shares of b_u: $\{(v, b_{u,v})\}_{v\in\mathcal{U}_1} \leftarrow SS.share(b_u, t, \mathcal{U}_1)$

(4) For each other user $v \in \mathcal{U}_1 \backslash \{u\}$, compute $e_{u,v} \leftarrow AE.enc(k_{u,v}, u||v||s_{u,v}^{SK}||b_{u,v})$

(5) Send all the ciphertexts $e_{u,v}$ to the server (each implicitly containing addressing information u, v as metadata).

(6) Store all messages received and values generated.

Server:

(1) Collect lists of ciphertexts from at least t users (denote with $\mathcal{U}_2 \subseteq \mathcal{U}_1$ this set of users).

(2) Sends to each user $u \in \mathcal{U}_2$ all ciphertexts encrypted for it: $\{e_{u,v}\}_{v \in \mathcal{U}_2}$.

5.2.3 Private Data Masking

User u:

(1) If the list of $e_{u,v}$ received from server is of size less than t, abort.

(2) For each other user $v \in \mathcal{U}_2 \backslash \{u\}$, expand $k_{u,v}$ using a PRNG into a random vector $\mathbf{p}_{u,v} = \Delta_{u,v} \cdot PRNG(k_{u,v})$, where $\Delta_{u,v} = 1$ when $u > v$, and $\Delta_{u,v} = -1$ when $u < v$. Additionally, define $\mathbf{p}_{u,u} = 0$.

(3) Compute the user's own private mask vector $\mathbf{b}_u = PRNG(b_u)$. Then, compute the masked input vector $\mathbf{y}_u \leftarrow \mathbf{x}_u + \mathbf{b}_u + \sum_{v \in \mathcal{U}_2} \mathbf{p}_{u,v} \pmod{)R}$

(4) Send $\mathbf{y}_u$ to the server.

Server:

(1) Collect $\mathbf{y}_u$ from at least t users (denote with $\mathcal{U}_3 \subseteq \mathcal{U}_2$ this set of users). Send to each user in $\mathcal{U}_3$ the list $\mathcal{U}_3$.

5.2.4 User Dropout Consistency Check

User u:

(1) Receive from the server a list $\mathcal{U}_3 \subseteq \mathcal{U}_2$ consisting of at least t users (including itself). If $\mathcal{U}_3$ is smaller than t, abort.

(2) Send to the server $\sigma'_u \leftarrow SIG.sign(d_u^{SK}, \mathcal{U}_3)$.

Server :

(1) Collect σ' from at least t users (denote with $\mathcal{U}_4 \subseteq \mathcal{U}_3$ this set of users). Send to each user in $\mathcal{U}_4$ the set $\{v, \sigma'_v\}_{v \in \mathcal{U}_4}$.

5.2.5 Unmasking Aggregate Data

User u:

(1) Receive from the server a list $\{v, \sigma'_v\}_{v \in \mathcal{U}_4}$. Verify that $\mathcal{U}_4 \subseteq \mathcal{U}_3$, that $|\mathcal{U}_4| \geq t$ and that $SIG.ver(d_u^{PK}, \mathcal{U}_3, \sigma') = 1$ for all $v \in \mathcal{U}_4$ (otherwise abort).

(2) For each other user v in $\mathcal{U}_2 \backslash \{u\}$, decrypt the ciphertext $v'||u'||s_{v,u}^{SK}||b_{v,u} \leftarrow AE.dec(k_{v,u}, e_{v,u})$ and assert that $u = u' \wedge v = v'$.

(3) Send a list of shares to the server, which consists of $s_{v,u}^{SK}$ for users $v \in \mathcal{U}_2 \backslash \mathcal{U}_4$ and $b_{v,u}$ for users in $v \in \mathcal{U}_4$.

Server:

(1) Collect from at least t users (denote with $\mathcal{U}_5$ this set of users).

(2) For each user in $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$, reconstruct $s_u^{SK} \leftarrow SS.recons(\{s_{u,v}^{SK}\}_{v \in \mathcal{U}_5}, t)$ and use it (together with the public keys received in Section 5.2.1) to recompute $\mathbf{p}_{v,u}$ for all $v \in \mathcal{U}_3$ using the PRNG.

(3) For each user $u \in \mathcal{U}_3$, reconstruct $b_u \leftarrow SS.recons(\{b_{u,v}\}_{v \in \mathcal{U}_5}, t)$ and then recompute $\mathbf{b}_u$ using the PRNG.

(4) Compute and output $\mathbf{z} = \sum_{u \in \mathcal{U}_3} \mathbf{x}_u$ as

$$\sum_{u \in \mathcal{U}_3} \mathbf{x}_u = \sum_{u \in \mathcal{U}_3} \mathbf{y}_u - \sum_{u \in \mathcal{U}_3} \mathbf{b}_u + \sum_{u \in \mathcal{U}_3, v \in \mathcal{U}_2 \backslash \mathcal{U}_3} \mathbf{p}_{v,u}$$

5.2.6 Security Analysis

The security proof of the protocol in [7] is based on the Decision Diffie-Hellman problem (Definition 3.4.1). We are going to illustrate why the privacy of the user is preserved in the curious-but-honest adversary model.

From Section 5.2.3 the server can receive the masking data $\mathbf{y}_u$ from every user $u \in \mathcal{U}_2$, however, it is impossible for the server to learn anything of user's private data $\mathbf{x}_u$ under a curious-but-honest adversary model. Since the server can collect information from at least t users in Section 5.2.5, we can select t users from $\mathcal{U}_5$ and denote those users as $u_1, u_2, \cdots, u_t$.

Lemma 5.2.1. *For any user $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$, the server can only get $\mathbf{x}_u + \mathbf{b}_u$ from $\mathbf{y}_u$ it received.*

Proof. From Section 5.2.5, the server can compute s_u^{SK} for each user $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$ by using Lagrange Interpolation from Lemma 3.2.1

$$s_u^{SK} = \sum_{i=1}^{t} \prod_{1 \le j \le t, j \ne i} \frac{0 - u_j}{u_i - u_j} ss_{u_i,u}^{SK}$$

The server can then calculate $k_{u,v}$ for each user $v \in \mathcal{U}_2$ by:

$$k_{u,v} \leftarrow KA.agree(s_u^{SK}, s_v^{PK})$$

$\mathbf{p}_{u,v}$ can be determined for each user $v \in \mathcal{U}_2$ by:

$$\mathbf{p}_{u,v} = \begin{cases} PRNG(k_{u,v}), u > v \\ 0, u = v \\ -PRNG(k_{u,v}), u < v \end{cases}$$

The sever can then calculate $\mathbf{x}_u + \mathbf{b}_u$ from $\mathbf{y}_u$ by:

$$\begin{aligned} \mathbf{y}_u - \sum_{v \in \mathcal{U}_2} \mathbf{p}_{u,v} &= \mathbf{x}_u + \mathbf{b}_u + \sum_{v \in \mathcal{U}_2} \mathbf{p}_{u,v} - \sum_{v \in \mathcal{U}_2} \mathbf{p}_{u,v} \mod \mathbb{R} \\ &= \mathbf{x}_u + \mathbf{b}_u \mod \mathbb{R} \end{aligned}$$

However, since in Section 5.2.5 the server will not receive information of $\mathbf{b}_u$ of any user $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$, the server can only get $\mathbf{x}_u + \mathbf{b}_u$ from $\mathbf{y}_u$ it received under a curious-but-honest adversary model. □

Lemma 5.2.2. *For any user $u \in \mathcal{U}_3$, the server can only get $\mathbf{x}_u + \sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v}$ from $\mathbf{y}_u$ it received.*

Proof. From Section 5.2.5, the server can compute b_u for each user $u \in \mathcal{U}_4$ by using Lagrange Interpolation from Lemma 3.2.1

$$b_u = \sum_{i=1}^{t} \prod_{1 \le j \le t, j \ne i} \frac{0 - u_j}{u_i - u_j} b_{u_i,u}$$

$\mathbf{p}_u$ can be determined by:

$$\mathbf{p}_u = PRNG(b_u)$$

The sever can then calculate $\mathbf{x}_u + \sum_{v \in \mathcal{U}_4} \mathbf{p}_{u,v}$ from $\mathbf{y}_u$ by:

$$\begin{aligned} \mathbf{y}_u - \mathbf{p}_u &= \mathbf{x}_u + \mathbf{b}_u + \sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v} - \mathbf{p}_u \pmod{\mathbb{R}} \\ &= \mathbf{x}_u + \sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v} \pmod{\mathbb{R}} \end{aligned}$$

However, in Section 5.2.5 the server will not receive information of $\mathbf{b}_u$ for any user $u \in \mathcal{U}_3$, under a curious-but-honest adversary model, the server can only get $\mathbf{x}_u + \sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v}$ from $\mathbf{y}_u$ it received. □

Lemma 5.2.3. *For any user $u \in \mathcal{U}_2$, the server cannot learn $\mathbf{x}_u$ from $\mathbf{y}_u$ it received.*

Proof. From Decision Diffie-Hellman problem we can think of $\sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v}$ as a variable that is computationally indistinguishable from random, thus $\sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v}$ is a random vector over field $\mathbb{R}$. Since b_u is a random variable, then the vector $\mathbf{b}_u$ which is generated by b_u and a pseudo-random generator function can also be computationally indistinguishable from a random vector.

Because the server can get either $\mathbf{x}_u + \mathbf{b}_u$ for each user $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$ and $\mathbf{x}_u + \sum_{v \in \mathcal{U}_3} \mathbf{p}_{u,v}$ from $\mathbf{y}_u$ for user $u \in \mathcal{U}_3$, then for all $\mathbf{y}_u, u \in \mathcal{U}_2$ which Sever get in Section 5.2.3, the value the server received is the sum of users' private data vector and a random vector. As a result, it is infeasible for the server to probe users' private information, which means the server cannot learn $\mathbf{x}_u$ from $\mathbf{y}_u$ it received. □

5.3 Complexity Analysis

We analyze the complexity of computation and storage in the same manner as Section 4.4 does. Given a random finite field $\mathbb{F}$, $\log \mathbb{F}$ is the bit length of the maximum value in $\mathbb{F}$. Since decryption scheme is an inverse operation of encryption scheme, we consider the operation of the decryption scheme used in the protocol whose input length is l bits is also $Enc(l)$. At the same time, since an operation of subtraction is the same as an operation of addition of the inverse number, the time of an operation of subtraction of l bits is $Add(l)$, too. We let the dimension of data vector to be r.

5.3.1 Complexity Analysis of the User

Time Complexity :

(a) In Section 5.2.1, the user needs to generate 2 Diffie-Hellman key pairs and 1 signature, which is $2(Srand(\log|\mathbb{G}|) + Exp(\log|\mathbb{G}|)) + Sign(\log|\mathbb{G}|)$ operations in total. A user then needs to compute n different Diffie-Hellman pairwise key, the computation time is $nExp(\log|\mathbb{G}|)$.

(b) In Section 5.2.2, the user needs to generate a random number using $Srand(\log|\mathbb{F}|)$ time. Then generate two $(t+1)$ coefficients polynomial using $2(t+1)Srand(\log|\mathbb{F}|)$ time.
Then the user needs to evaluate n times for the two polynomials. According to Horner's Method it needs $2nt(Mul(\log|\mathbb{F}|) + Add(\log|\mathbb{F}|))$ operations to evaluate.
In the end it needs to encrypt secret shares n times using $nEnc(\log|\mathbb{F}|)$ time.

(c) In 5.2.3, the user needs $rSrand(\log|\mathbb{R}|)$ operations to generate one $\mathbf{p}_{u,v}$, which in total is $rnSrand(\log|\mathbb{R}|)$ for n users.
Then $rSrand(\log|\mathbb{R}|)$ operations is needed to generate $\mathbf{b}_u$.
In the end $rnAdd(\log|\mathbb{R}|)$ operations is needed to generate $\mathbf{y}_u$.

(d) In Section 5.2.4, the user needs to sign a message which takes $Sign(\log|\mathbb{G}|)$ time.

(e) In Section 5.2.5, the user needs to do at least t signature verification which takes $tSign(\log|\mathbb{G}|)$ time, and at most $(n-t)$ decryption which take $(n-t)Enc(\log|\mathbb{F}|)$ time.

Space Complexity:

(a) In Section 5.2.1, the user needs $2(t+1)\log|\mathbb{F}|$ space to store coefficients, and $n\log|\mathbb{G}|$ space to store all the value received in this step.

(b) In Section 5.2.2, the user needs to store $n\log|\mathbb{F}|$ different messages for $\{e_{u,v}\}_{v\in\mathcal{U}_2}$.

(c) In Section 5.2.3, the user needs to store $4r\log|\mathbb{R}|$ space for $\mathbf{y}_u$,$\mathbf{x}_u$,$\mathbf{b}_u$ and $\sum_{v\in\mathcal{U}_2}\mathbf{p}_{u,v}$.

(d) In Section 5.2.4, the user needs at most $(n-t)\log n$ space to store the list.

(e) In Section 5.2.5, the user needs to spend at most $(n-t)\log|\mathbb{F}|$ space to store $b_{u,v}$.

Communication Complexity:

(a) In Section 5.2.1, the user needs to send 2 public keys and receive $2(n-1)$ public keys in communication.

(b) In Section 5.2.2, the user needs to send n messages of ciphertexts to the server.

(c) In Section 5.2.3, the user needs to send a message $\mathbf{y}_u$ with r dimension to the server.

(d) In Section 5.2.4, the user needs to receive at most n messages of signature and send 1 signature to the server.

(e) In Section 5.2.5, the user needs to send at most $n-t$ messages of secret shares to the server.

5.3.2 Complexity Analysis of the Server

Time Complexity :

(a) In Sections 5.2.1, 5.2.2, 5.2.3 and 5.2.4, there is no computation needed.

(b) In Section 5.2.5, the server needs $tMul(\log|\mathbb{F}|)$ to reconstruct s_u^{SK}, $Exp(\log|\mathbb{G}|)$ to reconstruct secret key, and at most $(n-t)$ times of operations to reconstruct users' secret polynomials. It takes $(n-t)(tMul(\log|\mathbb{F}|)+Exp(\log|\mathbb{G}|))$ in total.

Then, the server needs $tMul(\log|\mathbb{F}|)$ and $rSrand(\log|\mathbb{R}|)$ to reconstruct a $\mathbf{b}_u$ at most $n-t$ times, which takes $(n-t)(tMul(\log|\mathbb{F}|)+rSrand(\log|\mathbb{R}|))$ operations altogether.

At last server needs to do $3r$ additions to get aggregate value in $3rAdd(\log|\mathbb{R}|)$ time.

Space Complexity:

(a) In Section 5.2.1, server needs to store n messages from individual users which is $n\log|\mathbb{G}|$ space.

(b) In Section 5.2.2, server needs to store n^2 messages for $e_{v,u}$ which takes $n^2\log|\mathbb{F}|$.

(c) In Section 5.2.3, server needs to store at least $tr\log|\mathbb{R}|$ for $\mathbf{y}_u$ from at least t users.

(d) In Section 5.2.4, server needs to store at most n messages from individual users which is $n\log|\mathbb{G}|$ space.

(e) In Section 5.2.5, server needs $(t+1)\log|\mathbb{F}|$ to store secret share polynomial, and $\log|\mathbb{G}|$ to store secret key, which in total takes $(n-t)((t+1)\log|\mathbb{F}|+\log|\mathbb{G}|)$ space.

Then server needs $(n-t)(t\log|\mathbb{F}|+r\log|\mathbb{R}|))$ space to store values used in aggregating sum.

Since server can only store a running sum to calculate $\mathbf{y}_u$, at last it needs $r\log|\mathbb{R}|$ space to store running value.

Communication Complexity:

(a) In Section 5.2.1, server needs to broadcast a message of a n users' list to all user, which means server needs to send n^2 messages in communication.

(b) In Section 5.2.2, server needs to collect ciphertexts at most n times and send them to all users with the cost of n times, which is n^2 messages in total.

(c) In Section 5.2.3,server needs to send the list of users in $\mathcal{U}_3$ to all user in $\mathcal{U}_3$ at least n^2 times.

(d) In Section 5.2.4, server needs to send the list of users in $\mathcal{U}_4$ to all user in $\mathcal{U}_4$ at least n^2 times.

(e) In Section 5.2.5, server needs to receive at most $n-t$ shares to reconstruct $\mathbf{p}_{u,v}$ and b_u, and at most rn data vector to update running sum.

In conclusion, the time, space and communication bit complexity of the server and the user are given in Table 5.1.

Table 5.1: Bit Complexity Analysis of Privacy-Preserving Mobile Data Aggregation Protocol

	Time Complexity	Space Complexity	Communication Complexity
Server	$O(ntMul(\log \lvert\mathbb{F}\rvert)+ nExp(\log \lvert\mathbb{G}\rvert) + nrSrand(\log \lvert\mathbb{R}\rvert))$	$O(n^2 \log \lvert\mathbb{F}\rvert + nt \log \lvert\mathbb{G}\rvert + r \log \lvert\mathbb{R}\rvert))$	$O(n^2 + nr)$
User	$O(nSign(\log \lvert\mathbb{G}\rvert)+ nExp(\log \lvert\mathbb{G}\rvert) + rnSrand(\log \lvert\mathbb{F}\rvert) + ntMul(\log \lvert\mathbb{F}\rvert) + nEnc(\log \lvert\mathbb{F}\rvert))$	$O(n \log \lvert\mathbb{F}\rvert + r \log \lvert\mathbb{R}\rvert)$	$O(n + r)$

Chapter 6

A New Communication Efficient Data Aggregation Protocol

In this chapter, we introduce our new communication efficient and privacy-preserving data aggregation protocol. We present an overview of the protocol in Section 6.1. The details of the scheme are presented in Section 6.2. We present the adversarial model and the security analysis in Section 6.3. Finally, we analyze the bit complexity of the protocol in Section 6.4.

6.1 Scheme Overview

In the complexity analysis in Section 5.3, one can see that each user needs to generate a temporary pair of Diffie-Hellman public/private keys to mask private data. The inclusion of the mechanism ensures the private key reconstruction of drop-out users won't compromise the confidentiality of the users' private information. However, the pairwise keys' construction also involves n more times of Diffie-Hellman shared key exponentiation operations for the user and n^2 more communication rounds for the server. The construction is rather expensive considering that mobile device's resource is always limited. The original scheme requires each user to have a pair of public/private keys and know others' public keys, which would be rather expensive considering that as the number of mobile users scale, it is communication and space expensive to know every users' public key in the network.

Considering this issue, we combine the Harn-Gong key establishment protocol from Chapter 4 with the mobile data aggregation protocol from Chapter 5 to make it more communication efficient with privacy-preserving machine learning. We propose a detailed protocol in this chapter and the protocol improves the computation and communication efficiency. It also allows each user not to store and know every other user's public key in the network as long as they share the same master secret polynomial. A secure selection scheme of threshold t of master secret polynomial is also presented.

The trusted third party in Chapter 5 is assigned one more task to distribute the secret share polynomial for different users, which is a fair assumption considering that a trusted third party (in this case, the KDC) can issue each user a public/private keys certificate in the active adversary model of Chapter 5.

We also consider the network model where each user has the information of its neighboring nodes information in the new protocol. The numbers of neighboring users for all users are assumed to be equal, which is l, i.e., in terms of a graph representation, it is an l-regular undirected graph [8]. The l-regular communication network property is used in the data aggregation phase and the server will send the the information of neighboring nodes to each user.

The inclusion of l-regular communication network could reduce computation overhead significantly since the user does not need to mask his/her data with all the pairwise key in the network. For the drop-out situation in a mobile network in this chapter, we adopt the drop-out situation from Chapter 5, however, the computation overhead of drop-out situation can also be substantially reduced [33].

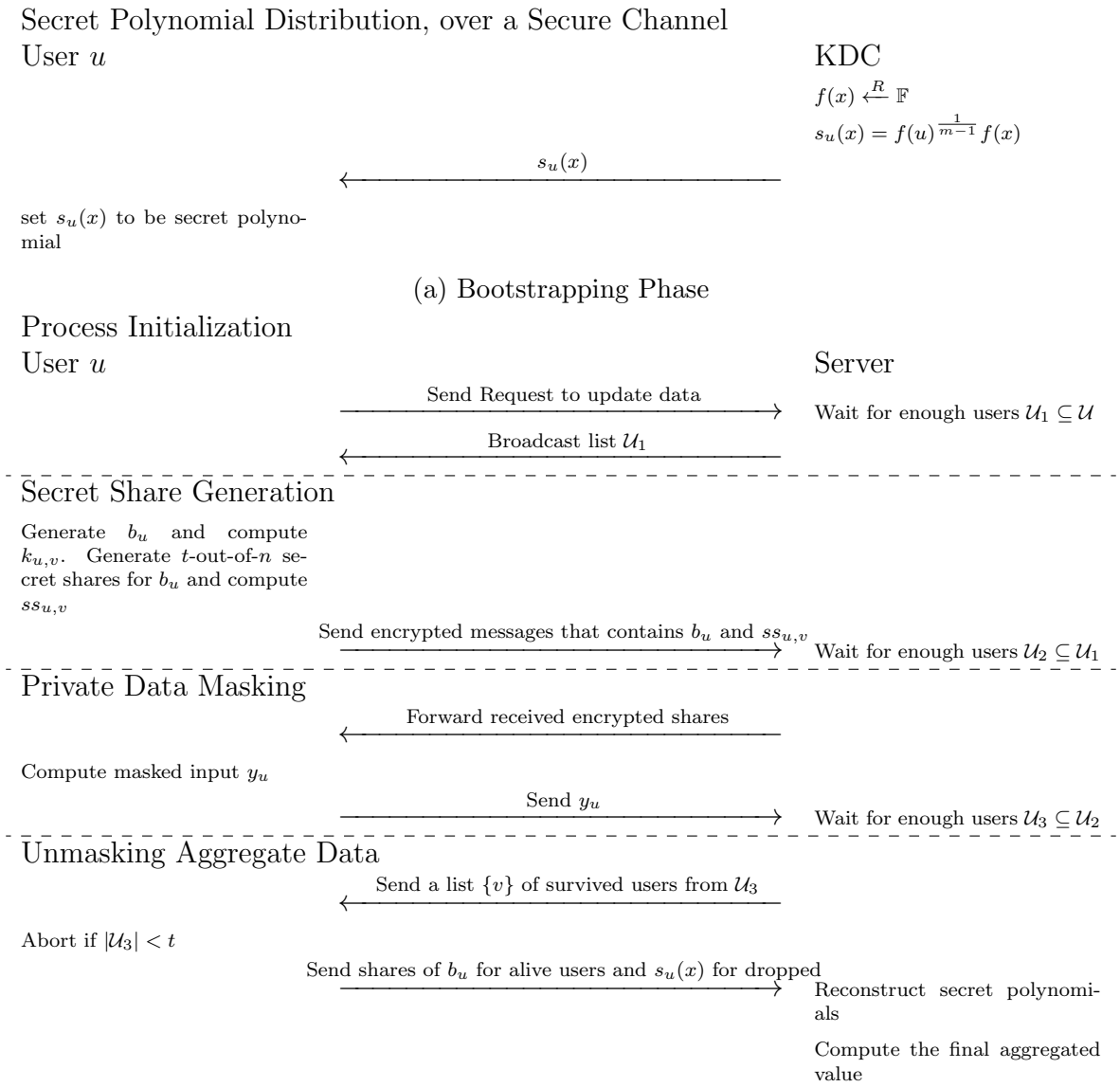
Property 6.1.1 (l-regular communication network)**.** An l-regular communication network is a logical network with a topology as an l-regular graph that is used in the pairwise key establishment phase.

Our scheme's high-level view is given in Figure 6.1.

Any mobile user who wants to send an update to the server can do the following:

1) Bootstrapping phase

   (a) Secret Share Polynomial Distribution



(a) Bootstrapping Phase

(b) Running Phase

Figure 6.1: High-level view of the protocol

2) Running phase

   (a) Process Initialization

   (b) Secret Share Generation

   (c) Private Data Masking

   (d) Unmasking Aggregate Data

6.2 Methodology

The setup for the protocol is the following, and the recommendation of the security parameters is given in Section 6.3:

1) All parties are given the security parameter k, the number of users n and a threshold value t, honestly generated $pp \leftarrow KA.gen(k)$, parameters m and R such that $\mathbb{Z}_R^m$ is the space from which inputs are sampled, and a field $\mathbb{F}$ to be used for secret sharing.

2) All users also have a private authenticated channel with the server.

3) Each user u receives his/her secret share polynomial $s_u(x) = f(u)^{\frac{1}{m-1}} f(x) \mod N$ from the trusted third party.

If any of the below operations (assertion, signature verification, key agreement, encryption) fails, the protocol will abort. $\mathcal{U}_i, 1 \leq i \leq 4$ represents the set of users in each round.

6.2.1 Process Initialization

User u:

(1) Request server to update data.

Server :

(1) Collect at least t messages from individual users in the previous round (denote by $\mathcal{U}_1$ this set of users). Otherwise, abort.

(2) Broadcast to all users in $\mathcal{U}_1$ the list $\{v\}_{v \in \mathcal{U}_1}$.

6.2.2 Secret Share Generation

User u:

(1) Receive the list $\{v\}_{v \in \mathcal{U}_1}$ broadcasted by the server. Verify that $|\mathcal{U}_1| \geq t$.

(2) Select $b_u \xleftarrow{R} \mathbb{F}$ to be used as a seed for a $PRNG$.

(3) For $v \in \mathcal{U}_1$, compute $ss_{u,v} = s_u(v) \pmod N$.

(4) Generate t-out-of-$|\mathcal{U}_1|$ shares of b_u: $\{(v, b_{u,v})\}_{v \in \mathcal{U}_1} \leftarrow SS.share(b_u, t, \mathcal{U}_1)$.

(5) For each user $v \in \mathcal{U}_1 \backslash \{u\}$, compute $e_{u,v} \leftarrow AE.enc(k_{u,v}, u||v||ss_{u,v}||b_{u,v})$, where $k_{u,v} = ss_{u,v} * s_u(0)^{m-2} \pmod N$.

(6) Send all the ciphertexts $e_{u,v}$ to the server (each implicitly containing addressing information u, v as metadata).

(7) Store all messages received and values generated.

Server:

(1) Collect lists of ciphertexts from at least t users (denote with $\mathcal{U}_2 \subseteq \mathcal{U}_1$ this set of users).

(2) Sends to each user $u \in \mathcal{U}_2$ all ciphertexts encrypted for it: $\{e_{u,v}\}_{v \in \mathcal{U}_2}$.

(3) Randomly selects the neighboring nodes $\mathcal{N}(u)$ of each user $u \in \mathcal{U}_2$. Sends $\mathcal{N}(u)$ to each user $u \in \mathcal{U}_2$.

6.2.3 Private Data Masking

User u:

(1) If the list of $e_{u,v}$ received from server is of size $< t$, abort.

(2) Suppose the data update to be sent is $\mathbf{x}_u$ in $\mathbb{R}$. For each other user $v \in \mathcal{N}(u)$, expand $k_{u,v}$ using a $PRNG$ into a random vector $\mathbf{p}_{u,v} = \Delta_{u,v} \cdot PRNG(k_{u,v})$, where $\Delta_{u,v} = 1$ when $u > v$, and $\Delta_{u,v} = -1$ when $u < v$. Additionally, define $\mathbf{p}_{u,u} = 0$.

(3) Compute the user's own private mask vector $\mathbf{b}_u = PRNG(b_u)$. Then, compute the masked input vector $\mathbf{y}_u \leftarrow \mathbf{x}_u + \mathbf{b}_u + \sum_{v \in \mathcal{N}(u)} \mathbf{p}_{u,v}$.

(4) Send $\mathbf{y}_u$ to the server

Server:

(1) Collect $\mathbf{y}_u$ from at least t users (denote by $\mathcal{U}_3 \subseteq \mathcal{U}_2$ this set of users). Send to each user in $\mathcal{U}_3$ the list $\mathcal{U}_3$.

6.2.4 Unmasking Aggregate Data

User u:

(1) Receive from the server a list $\{v\}_{v \in \mathcal{U}_3}$. Verify that $\mathcal{U}_3 \subseteq \mathcal{U}_2$, that $|\mathcal{U}_3| \geq t$ and $|\mathcal{U}_2| - |\mathcal{U}_3| \leq t$ (otherwise abort).

(2) For each user v in $\mathcal{U}_3 \backslash \{u\}$, decrypt the ciphertext $v'||u'||ss_{v,u}||b_{u,v} \leftarrow AE.dec(k_{v,u}, e_{v,u})$ and verify that $u = u'$ and $v = v'$.

(3) Send a list of shares to the server, which consists of $ss_{v,u}$ for users $v \in \mathcal{U}_2 \backslash \mathcal{U}_3$ and $b_{u,v}$ for users $v \in \mathcal{U}_3$.

Server:

(1) Collect the list of shares from at least t users (denote by $\mathcal{U}_4$ this set of users).

(2) For each user in $u \in \mathcal{U}_2 \backslash \mathcal{U}_3$, reconstruct $s_u(x) \leftarrow SS.reconp(\{ss_{u,v}\}_{v \in \mathcal{U}_4}, t)$ and use it (together with each user's public key) to recompute $\mathbf{p}_{v,u}$ for all $v \in \mathcal{U}_3$ using the PRNG.

(3) For each user $u \in \mathcal{U}_3$, reconstruct $b_u \leftarrow SS.recons(\{b_{u,v}\}_{v \in \mathcal{U}_4}, t)$ and then recompute $\mathbf{b}_u$ using the PRNG.

(4) Compute and output $\mathbf{z} = \sum_{u \in \mathcal{U}_3} \mathbf{x}_u$ as

$$\sum_{u \in \mathcal{U}_3} \mathbf{x}_u = \sum_{u \in \mathcal{U}_3} \mathbf{y}_u - \sum_{u \in \mathcal{U}_3} \mathbf{b}_u + \sum_{u \in \mathcal{U}_3, v \in \mathcal{U}_2 \backslash \mathcal{U}_3 \wedge v \in \mathcal{N}(u)} \mathbf{p}_{v,u}$$

.

6.3 Adversary Model and Security Analysis

We consider a curious-but-honest server that provides cloud services for machine learning. The server may be curious to inquire the user's secret, but it would not manipulate messages or act actively to acquire a user's secret or data. We assume there is a trusted third party who provides secret share polynomials and keeps $\phi(N)$ secret to make the secret share polynomial resist up to t users' collaboration. A secure threshold for t is provided to resist a passive adversary who wants to learn users' secret data to be updated by cloud server. The security of protocol is based on the RSA assumption and the Byzantine protocol.

Definition 6.3.1 (Byzantine Generals Problem)**.** A group of generals, each commanding a portion of the Byzantine army, encircle a city. These generals wish to formulate a plan for attacking the city. In its simplest form, the generals must decide only whether to attack or retreat. Some generals may prefer to attack, while others prefer to retreat.

Lemma 6.3.1. *For the multivariate polynomial key establishment protocol with m users and degree t in an open network, the protocol can reach a secure key agreement if $t \geq m/3$.*

Proof. We may assume the network of this protocol to be a situation where the Byzantine generals problem applied since no assumption of malicious nodes is made.

From [22] we know that there is a t resilient synchronous protocol without authentication which solves the strong Byzantine generals problems if and only if $t/n < 1/3$, where a t-resilient protocol automatically tolerates up to t processes and links failures and n is the number of processes. Thus we may assume the upper bound of malicious nodes to be $m/3$ for an m-members conference key protocol because there is no practical situation for this network to function securely if the upper bound of malicious node is over $m/3$.

We can then conclude that if the degree t of the polynomial to be greater than or equal to $m/3$ the security of the whole network could be based on other properties of the network without consideration of security parameter t. □

Lemma 6.3.2. *There exists an l-regular communication network for each user in the protocol.*

Proof. We first introduce the concept of an l-regular graph:

Definition 6.3.2 (l-regular graph)**.** A graph all of whose vertices have degree l is called an l-regular graph or regular graph of degree l.

For each user u that belongs to a network $\mathcal{U}$, it should be noticed that the user can communicate with any other user in the network directly or indirectly. Let $\mathcal{N}(u)$ denote a set of neighbors to which the user u can communicate directly. There exists an undirected edge between user u and each user $v \in \mathcal{N}(u)$. Since given two integers l and m, there exists an l-regular graph with m vertices if l or m is even, with a network with m members where m is even in the Harn-Gong protocol, the *KDC* can always maintain a network where each user has exactly l neighbors, i.e., $|\mathcal{N}(u)| = l$ for all $u \in \mathcal{U}$. □

Lemma 6.3.3. *From the new application of the data aggregation scheme, the cloud server can produce a correct aggregate sum in the final round.*

Proof. Since the server can collect information from at least t users' secret share, we can select t users' information and denote those users as $u_1, u_2, \cdots, u_t$. For each user $u_i, 1 \leq i \leq t$, the user's information consists of ss_{v,u_i} for users $v \in \mathcal{U}_2 \backslash \mathcal{U}_3$ and b_{v,u_i} for users $v \in \mathcal{U}_3$.

From Lemma 3.2.1, the server can construct the secret polynomial for all users $v \in \mathcal{U}_2 \backslash \mathcal{U}_3$ by the following calculation

$$s_v(x) = \sum_{i=1}^{t} \prod_{1 \leq j \leq t, j \neq i} \frac{x - u_j}{u_i - u_j} ss_{v,u_i}$$

As a result, for each user $v \in \mathcal{U}_2 \backslash \mathcal{U}_3 \wedge v \in \mathcal{N}(u)$, the server can get its $\sum_{u \in \mathcal{U}_2} \mathbf{p}_{v,u}$ where $\mathbf{p}_{v,u} = PRNG(k_{v,u})$ if $v > u$ and $\mathbf{p}_{v,u} = -PRNG(k_{v,u})$ if $v < u$. $k_{v,u}$ is pairwise key computed by $s_v(x)$.

By the same construction method, server can get b_v for all user $v \in \mathcal{U}_2$

$$b_v = \sum_{i=1}^{t} \prod_{1 \leq j \leq t, j \neq i} \frac{0 - u_j}{u_i - u_j} b_{v,u_i}$$

Thus for each user $v \in \mathcal{U}_2 \backslash \mathcal{U}_3$, the server can get its $\mathbf{p}_v = PRNG(b_v)$.

For every $u \in \mathcal{U}_3$, the server can get its $\mathbf{y}_u$ from Section 6.2.3, thus get its aggregate

sum $\sum_{u\in\mathcal{U}_3}\mathbf{x}_u$ by the following process:

$$\begin{aligned}
&\sum_{u\in\mathcal{U}_3}\mathbf{y}_u-\sum_{u\in\mathcal{U}_3}\mathbf{b}_u+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_2\setminus\mathcal{U}_3\wedge v\in\mathcal{N}(u)}\mathbf{p}_{v,u}\\
&=\sum_{u\in\mathcal{U}_3}(\mathbf{x}_u+\mathbf{b}_u+\sum_{v\in\mathcal{U}_2\wedge v\in\mathcal{N}(u)}\mathbf{p}_{u,v})-\sum_{u\in\mathcal{U}_3}\mathbf{b}_u+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_2\setminus\mathcal{U}_3\wedge v\in\mathcal{N}(u)}\mathbf{p}_{v,u}\\
&=\sum_{u\in\mathcal{U}_3}\mathbf{x}_u+\sum_{u\in\mathcal{U}_3}\mathbf{b}_u+\sum_{u\in\mathcal{U}_3}\sum_{v\in\mathcal{U}_2\wedge v\in\mathcal{N}(u)}\mathbf{p}_{u,v}-\sum_{u\in\mathcal{U}_3}\mathbf{b}_u+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_2\setminus\mathcal{U}_3\wedge v\in\mathcal{N}(u)}\mathbf{p}_{v,u}\\
&=\sum_{u\in\mathcal{U}_3}\mathbf{x}_u+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_2\wedge v\in\mathcal{N}(u)}\mathbf{p}_{u,v}+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_2\setminus\mathcal{U}_3\wedge v\in\mathcal{N}(u)}\mathbf{p}_{v,u}\\
&=\sum_{u\in\mathcal{U}_3}\mathbf{x}_u+\sum_{u\in\mathcal{U}_3,v\in\mathcal{U}_3\wedge v\in\mathcal{N}(u)}\mathbf{p}_{v,u}\\
&=\sum_{u\in\mathcal{U}_3}\mathbf{x}_u
\end{aligned}$$

□

Corollary 6.3.3.1. *For each pair of users u, v in $\mathcal{U}_1$, the established pairwise keys are the same, which means $k_{u,v}=k_{v,u}$.*

Proof. Since from the Harn-Gong protocol we know that

$$\begin{aligned}
k_{u,v}&=s_u(v)*s_u(0)^{m-2}\\
k_{v,u}&=s_v(u)*s_v(0)^{m-2}
\end{aligned}$$

And from the definition of the secret polynomial we know that

$$\begin{aligned}
s_u(x)&=f(u)^{\frac{1}{m-1}}f(x)\\
s_v(x)&=f(v)^{\frac{1}{m-1}}f(x)
\end{aligned}$$

Hence the established key is

$$\begin{aligned}
k_{u,v}&=s_u(v)*s_u(0)^{m-2}\\
&=f(u)^{\frac{1}{m-1}}f(v)*(f(u)^{\frac{1}{m-1}}f(0))^{m-2}\\
&=f(u)^{\frac{1}{m-1}}f(v)*(f(u)^{\frac{1}{m-1}})^{m-2}f(0)^{m-2}\\
&=f(u)^{\frac{1}{m-1}}f(v)*f(u)^{\frac{m-2}{m-1}}f(0)^{m-2}\\
&=f(u)f(v)f(0)^{m-2}
\end{aligned}$$

From the same process we can calculate that

$$\begin{aligned} k_{v,u} &= f(v)f(u)f(0)^{m-2} \\ &= k_{u,v} \end{aligned}$$

□

Corollary 6.3.3.2. *For any pair of users* $u, v \in \mathcal{U}_1$, $\mathbf{p}_{v,u}+\mathbf{p}_{u,v} = 0$ *and* $\sum_{u\in\mathcal{U},v\in\mathcal{U}\wedge v\in\mathcal{N}(u)} \mathbf{p}_{v,u} = 0$.

Proof. To prove that $\mathbf{p}_{v,u} + \mathbf{p}_{u,v} = 0$, we can assume that $v > u$. Since $v > u$, $\mathbf{p}_{v,u} = PRNG(k_{v,u})$ and $\mathbf{p}_{u,v} = -PRNG(k_{u,v})$

From Corollary 6.3 we know that $k_{v,u} = k_{u,v}$, thus

$$\begin{aligned} \mathbf{p}_{v,u} + \mathbf{p}_{u,v} &= PRNG(k_{u,v}) - PRNG(k_{v,u}) \\ &= 0 \end{aligned}$$

Since $\mathbf{p}_{v,u} + \mathbf{p}_{u,v} = 0$ and if $v \in \mathcal{N}(u)$ then $u \in \mathcal{N}(v)$, we can calculate that

$$\begin{aligned} \sum_{u\in\mathcal{U},v\in\mathcal{U}\wedge v\in\mathcal{N}(u)} \mathbf{p}_{v,u} &= \sum_{u\in\mathcal{U},v\in\mathcal{U}\wedge v\in\mathcal{N}(u),u<v} (\mathbf{p}_{v,u} + \mathbf{p}_{u,v}) \\ &= 0 \end{aligned}$$

□

Lemma 6.3.4. *For any user* $u \in \mathcal{U}_2$, *the server cannot learn* $\mathbf{x}_u$ *from* $\mathbf{y}_u$ *it received.*

Proof. From Claim 4.3.2 we know that by knowing fewer than $t + 1$ secret polynomials it is computationally infeasible to know other conference keys. For the curious-but-honest cloud server, since it only knows $|\mathcal{U}_2| - |\mathcal{U}_3|$ secret polynomials and $|\mathcal{U}_2| - |\mathcal{U}_3| < t + 1$, it is also computationally infeasible for the server to obtain other conference keys involving user $u \in \mathcal{U}_3$, which makes the server cannot learn $\mathbf{x}_u$ from $\mathbf{y}_u$ for each user $u \in \mathcal{U}_3$.

Since $\mathbf{b}_u$ is generated the same as the previous one does, from Lemma 5.2.6 we know that the server can neither learn $\mathbf{x}_u$ from $\mathbf{y}_u$ for each user $u \in \mathcal{U}_2\backslash\mathcal{U}_3$. Hence the server cannot learn $\mathbf{x}_u$ from $\mathbf{y}_u$ it received. □

Lemma 6.3.5. *The new protocol saves* $O(n^2)$ *communication complexity in the key establishment process compared to the scheme in Section 5.*

Proof. The mutual key establishment process in the protocol in Section 5 is that, every user in the network generates a Diffie-Hellman key pair and send that to the server, then the server sends each user in the network the Diffie-Hellman public keys' list of the network.

In the process above, the server needs to send a list of n Diffie-Hellman public keys to n users, which causes an $O(n^2)$ communication costs in the key establishment.

In the key establishment process of the new protocol, each user can establish a pairwise key non-interactively by Harn-Gong Protocol, thus the new protocol saves $O(n^2)$ communication complexity in the key establishment process compared to the scheme in Section 5. □

6.4 Complexity Analysis

We consider the bit complexity in the same manner as Section 5.3 does.

6.4.1 Complexity Analysis of the User

Time Complexity :

(a) In Section 6.2.1, no computation is needed.

(b) In Section 6.2.2, the user needs to generate a random number using $Srand(\log|\mathbb{F}|)$ time. Then generate a set of $(t+1)$ coefficients polynomial for b_u secret sharing using $(t+1)Srand(\log|\mathbb{F}|)$ time.
Then the user needs to evaluate n times for secret sharing polynomial of b_u and l times for Harn-Gong secret share polynomial. According to Horner's Method it needs $ntMul(\log|\mathbb{F}|)+ntAdd(\log|\mathbb{F}|)+ltMul(\log|\mathbb{G}|)+ltAdd(\log|\mathbb{G}|)$ operations to evaluate.
In the end it needs to encrypt n times using $nEnc(\log|\mathbb{G}|)$ time.

(c) In Section 6.2.3, the user needs $rSrand(\log|\mathbb{R}|)$ operations to generate $\mathbf{p}_{u,v}$ for each user v, which in total is $rnSrand(\log|\mathbb{R}|)$. $rSrand(\log|\mathbb{R}|)$ operations is needed to generate $\mathbf{b}_u$ and $rnAdd(\log|\mathbb{R}|)$ is required to generate $\mathbf{y}_u$.

(d) In Section 6.2.4, the user needs at most $(n-t)$ decryption which take $(n-t)Enc(\log|\mathbb{G}|)$ time.

Space Complexity:

(a) In Section 6.2.1, the user needs $(t+1)\log|\mathbb{G}|$ to store secret univariate product polynomial coefficients.

(b) In Section 6.2.2, the user needs to store $(t+1)\log|\mathbb{G}|$ to store secret share polynomial coefficients. The user also needs to store $n\log|\mathbb{F}|$ different messages for $\{e_{u,v}\}_{v\in\mathcal{U}_2}$.

(c) In Section 6.2.3, the user needs to store $4r\log|\mathbb{R}|$ space for $\mathbf{y}_u$,$\mathbf{x}_u$,$\mathbf{b}_u$ and $\sum_{v\in\mathcal{U}_2}\mathbf{p}_{u,v}$

(d) In Section 6.2.4, the user needs to spend at most $(n-t)\log|\mathbb{F}|$ space to store $b_{u,v}$.

Communication Complexity:

(a) In Section 6.2.1, the user needs to send one message to server in communication.

(b) In Section 6.2.2, the user needs to send n messages of ciphertext to the server.

(c) In Section 6.2.3, the user needs to send r dimension of message $\mathbf{y}_u$ to the server.

(d) In Section 6.2.4, the user needs to send at most $(n-t)$ messages of secret shares to the server.

6.4.2 Complexity Analysis of the Server

Time Complexity :

(a) In Sections 6.2.1, 6.2.2 and 6.2.3, there is no computation needed.

(b) In Section 6.2.4, server needs $t(Mul(\log|\mathbb{G}|)+Add(\log|\mathbb{G}|))$ to reconstruct $s_u(x)$, needs $t(Mul(\log|\mathbb{F}|)+Add(\log|\mathbb{F}|))$ to evaluate pairwise key, and needs to do evaluation at most $(n-t)$ times, which takes $(n-t)(t(Mul(\log|\mathbb{F}|)+Mul(\log|\mathbb{G}|)+Add(\log|\mathbb{F}|)+Add(\log|\mathbb{G}|)))$ time.
Then, server needs $t(Mul(\log|\mathbb{F}|)+Add(\log|\mathbb{F}|))$ and $rSrand(\log|\mathbb{R}|)$ to reconstruct a $\mathbf{b}_u$, at most $n-t$ times, which takes $(n-t)(tMul(\log|\mathbb{F}|)+Add(\log|\mathbb{F}|)+rSrand(\log|\mathbb{R}|))$ time.
At last server needs to do $3rAdd(\log|\mathbb{R}|)$ to get aggregate value.

Space Complexity:

(a) In Section 6.2.1, a list of users is stored in $n \log |\mathbb{G}|$ space.

(b) In Section 6.2.2, server needs to store n^2 messages for $e_{v,u}$ which takes $n^2 \log |\mathbb{G}|$.

(c) In Section 6.2.3, server needs to store $r \log |\mathbb{R}|$ for $\mathbf{y}_u$ from at least t users which costs $r \log |\mathbb{R}|$ space in total.

(d) In Section 6.2.4, server needs $(t+1) \log |\mathbb{G}|$ to store secret share polynomial, and $\log |\mathbb{F}|$ to store secret key, which in total takes $(n-t)((t+1) \log |\mathbb{G}|)$ space.

Since server can only store a running sum to calculate $\mathbf{y}_u$, at last it needs $r \log |\mathbb{R}|$ space to store running value.

Communication Complexity:

(a) In Section 6.2.1, server needs to broadcast a message of a n users' list to all user, which means server needs to send n^2 messages in communication.

(b) In Section 6.2.2, server needs to collect ciphertexts at most n times and send them to all users with the cost of n times, which is n^2 messages in total.

(c) In Section 6.2.3, server needs to send the list of users in $\mathcal{U}_3$ to all user in $\mathcal{U}_3$ at least n^2 times

(d) In Section 6.2.4, server needs to receive at most $n-t$ shares to reconstruct $\mathbf{p}_{u,v}$ and b_u, and at most rn data vector to update running sum.

In conclusion, the time, space and communication bit complexity of the server and the user are listed in Table 6.1.

Table 6.1: Bit Complexity Analysis of Privacy-Preserving Mobile Data Aggregation Protocol

	Time Complexity	Space Complexity	Communication Complexity
Server	$O(ntMul(\log \lvert\mathbb{F}\rvert)+ (nt + r)Add(\log \lvert\mathbb{F}\rvert) + nrSrand(\log \lvert\mathbb{R}\rvert))$	$O(n^2 \log \lvert\mathbb{G}\rvert + r \log \lvert\mathbb{R}\rvert))$	$O(n^2 + nr)$
User	$O(nrSrand(\log \lvert\mathbb{F}\rvert)+ (nr + lt)Add(\log \lvert\mathbb{R}\rvert) + ltMul(\log \lvert\mathbb{G}\rvert) + nEnc(\log \mathbb{G}))$	$O(n \log \lvert\mathbb{F}\rvert + r \log \lvert\mathbb{R}\rvert)$	$O(n + r)$

Chapter 7

Implementation of Mobile Data Aggregation

Since the protocol in [28] does not include implementation detail, we implement necessary primitives for Harn-Gong protocol in python and test its running time according to different users' number as well as different polynomials' degree. Moreover, because the implementation of [7] in mobile user side does not involve a real Android application but a Java client in desktop, we implement our new protocol an Android application and also compare its performance with [7] in the same settings. It shows that our protocol is 1.5 to 3 times faster than [7] due to the non-interactive pairwise key establishment and l-regular communication network properties.

In this Chapter, we introduce our implementation of Harn-Gong protocol and its performance in Section 7.1. Different versions of the implementation of our new scheme and performance comparison are demonstrated in Section 7.2.

7.1 Python Lib for *KDC* Generating Secret Share

The implementation of python lib is a python module based on secret sharing to achieve group key sharing. It contains 5 python files:

1. *conferenceKey.py* is main API for conference key construct and recover.
2. *polynomials.py* is used for Lagrange interpolation and modular arithmetic.

3. *primes.py* is used to generate safe prime number.

4. *sharing.py* is original secret sharing API, which is not used in this group key sharing implementation.

5. *test_secret.py* is a unit test file for safe prime generation and group key construction.

7.1.1 Sample Usage

Splitting Into Shareholders

```
>>> from secretsharing import conferenceKey
>>> A, shareholders = conferenceKey.share_generation(3, 4, 4, 16)
>>> A
>>> [1517141389, 635182453, 524693228, 3001449852]

>>> shareholders
>>> [[405405764L, 2986928440L, 1555085675L, 2683010161L],
    [1177434373L, 1662914492L, 1555645294L, 2047286792L],
    [2209032879L, 2932824304L, 1219761147L, 2514560514L],
    [1714642752L, 807232125L, 809625417L, 1082038406L]]
```

Constructing Group Key

```
>>> Shareholders[0].conference_key_construct(range(4))
>>> 2018829063L
>>> Shareholders[1].conference_key_construct(range(4))
>>> 2018829063L
```

Recovering Secret

```
>>> v = []
>>> for s in Shareholders:
        v.append(s.secret_value())
>>> mod = Shareholders[0].get_modulus()
>>> conferenceKey.secret_reconstruct(range(4), v, 4, 3, mod)
>>> 849991194L
>>> A[0]**4 % mod
>>> 849991194L
```

7.1.2 API Description

- conferenceKey.**share_generation**(k, m, n, l = None,pub_inf = None):

  Returns a tuple (A, Shareholders) which contains a random polynomial with degree k for *KDC*, and n shareholders' polynomials for each member.

  l is a long bit length prime which can be either decided by *KDC* or generated by program.

  pub_inf is users' public keys information which could either be transmitted or generated by program.

- share_Polynomial.**conference_key_construct**(self, shareholders, pub_inf = None):

  Returns group key of users whose id is in *shareholders*.

- conferenceKey.**secret_reconstruct**(shareholders, values, n,k, modulus, pub_inf = None)):

  Returns secret in *KDC*'s random polynomial, which is constructed by users in *shareholders* and secret value of each user in *values*.

7.1.3 Secret Share Polynomial Generation Time

The *KDC* primitives implementation is built and run on a macOS High Sierra system with an Intel Core i7 processor and 16 GB memory. The time needed for generating secret share polynomials for a different number of users is demonstrated. The degree of polynomials is selected to be k and the generation time increases with the degree increases. The generation time also increases linearly according to the increase in the number of users. The generation time of different degrees and users' number is in Figure 7.1.

7.2 Android App for Client Masking Private Data

The Android implementation of the user is built and run on a Nexus 5 x86 Android Studio Simulator on macOS High Sierra 10.13.3. For cryptographic primitives, we firstly used

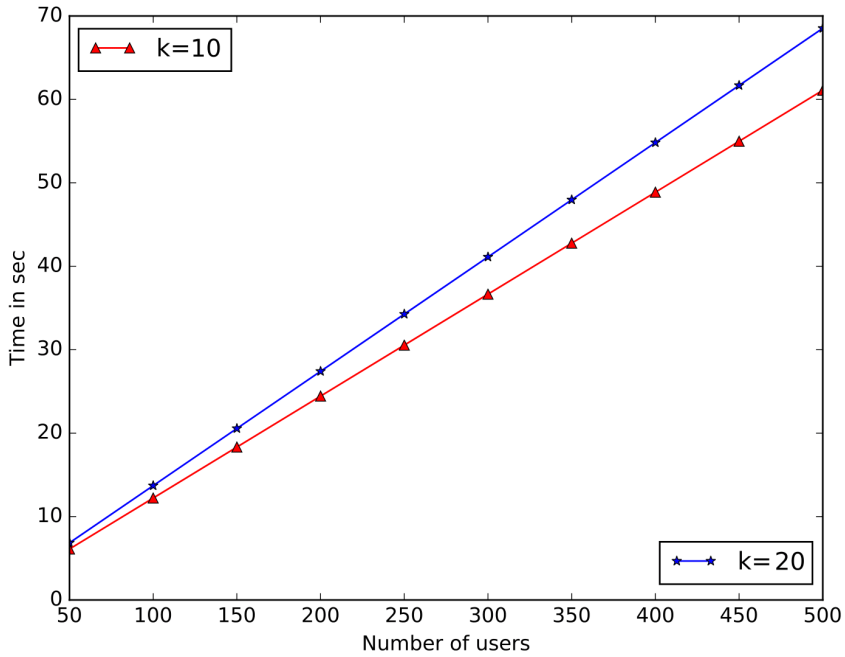


Figure 7.1: Secret Share Polynomial Generation Time in 128 Security Level

Java BigInteger class to evaluate the polynomial and calculated the application's masking time under different levels of security. The graphic user interface is shown in Figure 7.2. However, although we can get a rough result of masking time, we got a big memory overhead when evaluating polynomials using Horner's Method since the native BigInteger class is immutable.

We then compiled the OpenSSL library and use it as an Android native-lib to use a data structure like big integer and modulo arithmetic of C++ implementation. The C++ combined Android application makes it more time and memory efficient. We also changed our user interface to make the security level and number of neighbors more controllable by the user in Figure 7.3. Its masking time is demonstrated when clicking the button of "CALC" .

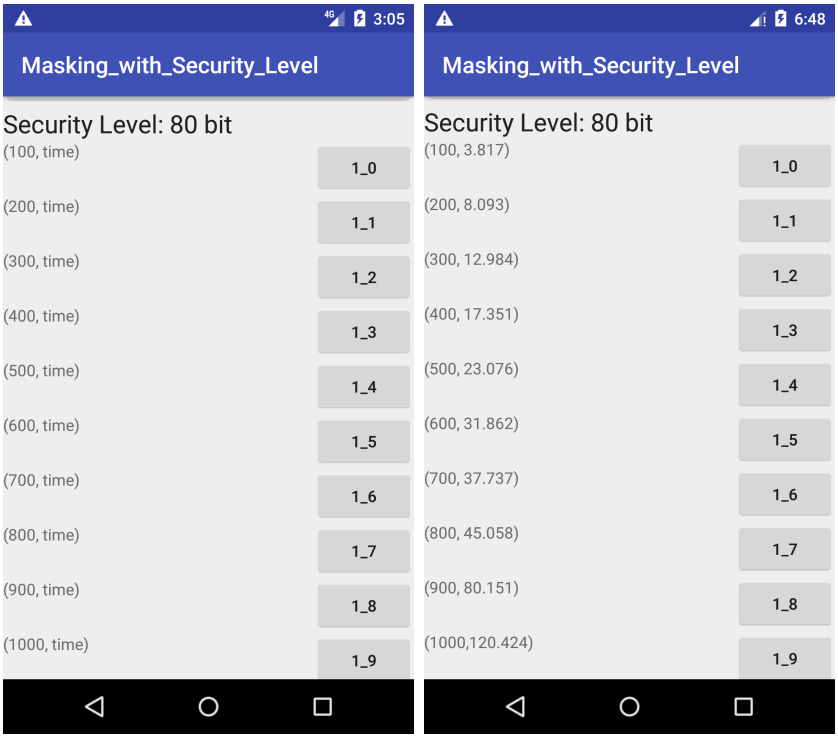


Figure 7.2: User selects security level locally using Java native BigInteger class. Left: Each button is displayed in the form of i_j, where j denotes how many neighbors user has in order and the label reveals number of neighbors and the time masking function uses. Right: After button is pressed, masking time is displayed in second by computation.

7.3 Execution Time

We first implemented a Java BigInteger class based masking app and collected its average execution time. However, a significant memory and time overhead is generated when evaluating different polynomials, and the app would even freeze when encountering big RSA modulus size and a large number of neighbors. The reason for it is that the Android app would inherently not collect garbage while the user is using the app, where a garbage collection would freeze the app and influence the user experience. For UX purpose it was fair, but since BigInteger class is immutable (like a String literal in general case), temporary BigInteger variables generated by Horner's method will not be cleared out, which causes a big memory space occupation of the app stack. Significant memory overhead is the reason why we can only get the security level for the number of users up to 1000.

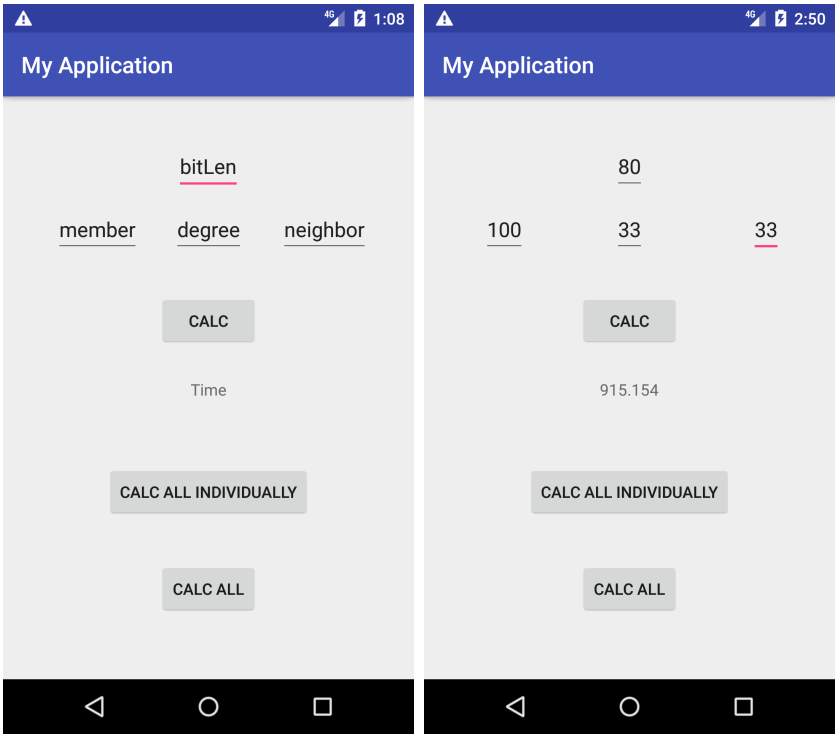


Figure 7.3: A user selects security level locally using native OpenSSL C++ library. Left: "CALC" button is clicked to calculate masking time and represent the result in "Time" label. Right: Masking time in millisecond for a user in a network of $m = 100$ users, with the secret polynomial of degree $t = 33$ and $l = 33$ neighbors.

Considering the essence of the developer not manipulating memory in Java language, we decided to compile a C++ library of OpenSSL into the Android system. The Java Native Interface is necessary to make the Android app call C++ function, so is a compiled C++ library based on the different underlying hardware system. After researching on several different programming languages' interaction, we compiled a C++ lib in macOS x86 Nexus 5 simulator and get a significant performance improvement. However, the hardware specific C++ library makes the app complicated to be transformed into a different Android phone and operating systems (ARM architecture, for example). Nevertheless, it is an excellent tool to illustrate how efficient our scheme could be.

7.3.1 Security Level Execution Time Comparison

We consider three security levels of 80, 112 and 128 bits. Its corresponding modulus sizes are provided in Table 7.1. The security level of RSA modulus is based on the difficulty of solving RSA problems of different modulus size, and the security level of Shamir's secret sharing (SS) modulus is raised to provide more security.

Table 7.1: Modulus Sizes for Different Security Levels

Security level	80	112	128
RSA modulus (N) size in bits	1024	2048	3072
SS modulus size in bits	160	224	256

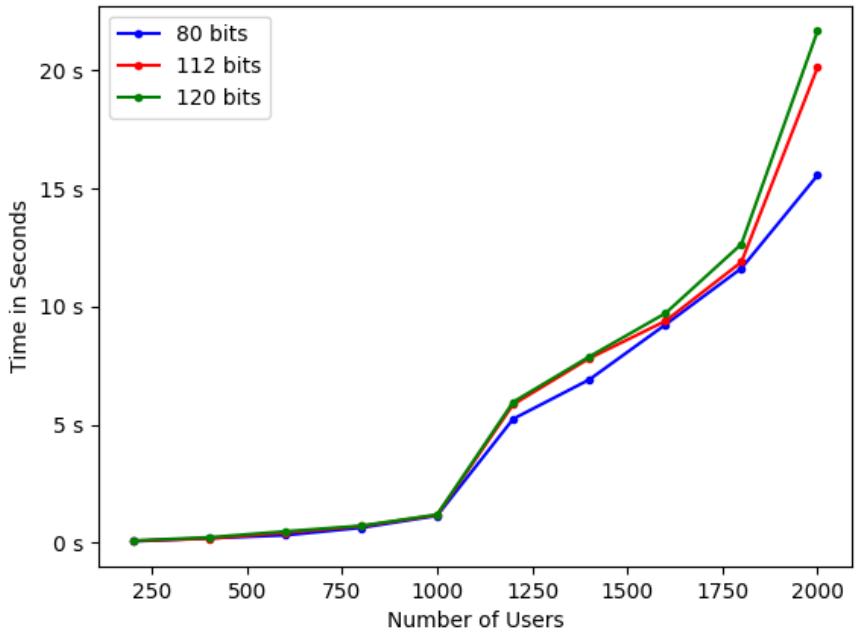


Figure 7.4: Different Security Level Masking Time Using Java BigInteger Class

Figure 7.4 shows that different security level influence little of performance time, the reason of which is that big overhead of the masking contributes from pseudo random num-

ber generation and masking. Such statement can also be found in [7].

7.3.2 Masking Schemes Execution Time Comparison

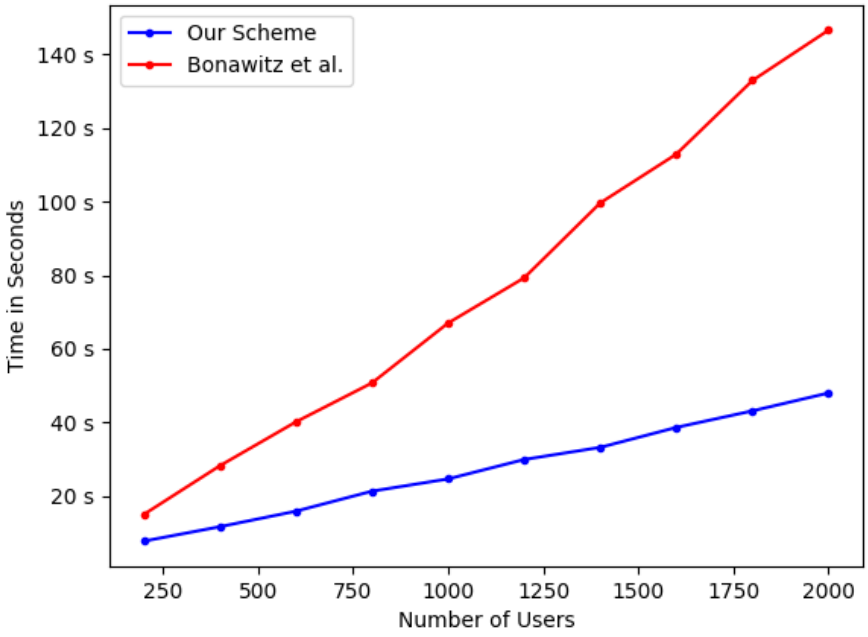


Figure 7.5: Masking Scheme Comparison Using C++ Native Lib

Figure 7.5 shows that our scheme and scheme in [7] are of the same increasing shape (which is the same in complexity analysis), but the running time of our scheme is significantly less than that of [7].

We implemented both schemes in the android app using C++ library, while the time comparison in [7] is a Java Client implemented in a desktop. The difference of computational resource from desktop and android phone might influence the running time of different schemes.

7.4 Power Consumption

Since power consumption of cryptographic operations can be a big issue for mobile devices, we test and provide the power consumption results of our Android application with OpenSSL and provide its comparison with previous work.

Figure 7.6 shows that our scheme is more power efficient in masking private data than

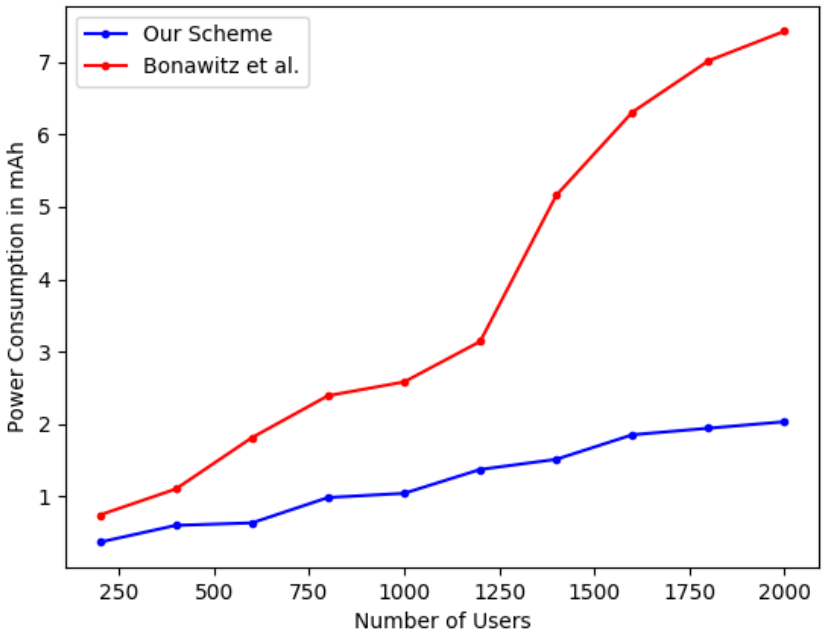


Figure 7.6: Masking Scheme Power Consumption Comparison

that of [7].

Chapter 8

Conclusions and Future Work

In this chapter, we present the conclusions in Section 8.1, and in Section 8.2 we list some directions for the future work.

8.1 Conclusions

With the growing use of Machine Learning as a Service (MLaaS), the cloud server collects data from mobile users to improve user experiences by applying learning algorithms. At the same time, providing privacy for mobile users data is challenging. In this thesis, we propose a communication-efficient scheme for a privacy-preserving data aggregation scheme, which aggregate users' data for machine learning services without revealing an individual's private information. The aggregate sum scheme is based on the one-time masking operation where the masking operation masks a user input with pairwise keys of its neighboring users specified by a communication network. Due to employing a non-interactive key generation mechanism for the masking operation, we improved Diffie-Hellman key exchange protocol[7] from $O(n^2)$ complexity to a constant value. The security and performance of the scheme is discussed.

In addition, we provided a new security proof of the Harn-Gong protocol and provided the security threshold of our new scheme. Furthermore, we implemented both the KDC python application for secret share polynomial distribution and the Android mobile application for masking user private data. We evaluated the feasibility of using the proposed scheme on smart phone devices by conducting experiments for 80, 112 and 128 bits security levels and various user and input sizes. The proof-of-concept implementation indicates

that our scheme is 1.5 to 3 times more efficient than that in [7].

8.2 Future Work

Since this thesis improves the communication efficiency of the privacy-preserving mobile data aggregation scheme, for the computation performance of the new protocol, our future work will aim to improve the multipoint evaluation efficiency. Since for Horner's Method, it is only ideal for one time evaluation, and there is a computation improving space for the multipoint evaluation for one polynomial using a divide-and-conquer algorithm.

Also, we adopt the passive adversary threat model from the privacy-preserving mobile data aggregation scheme and improves its efficiency. We would like to build a new efficient scheme against the active adversary threat model and provide its security proof.

References

[1] Elaine Barker, Lily Chen, Allen Roginsky, and Miles Smid. Recommendation for pairwise key establishment schemes using discrete logarithm cryptography. *NIST special publication*, 800:56A, 2013.

[2] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[3] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography SAC 1994*, pages 120–128, 1994.

[4] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[5] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[6] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptology — CRYPTO' 92*, pages 471–486. Springer, 1992.

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[8] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*. 1976.

[9] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium ANTS 1998*, pages 48–63. Springer, 1998.

[10] Dan Boneh and Matthew Franklin. Efficient generation of shared rsa keys. In *Advances in Cryptology — CRYPTO '97*, pages 425–439. Springer, 1997.

[11] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference TCC 2005*, pages 325–341. Springer, 2005.

[12] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[13] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual cryptology conference CRYPTO 2011*, pages 505–524. Springer, 2011.

[14] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in cryptology—EUROCRYPT'94*, pages 275–286. Springer, 1995.

[15] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010.

[16] Claude Castelluccia, Aldar CF Chan, Einar Mykletun, and Gene Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):20, 2009.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[18] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[19] Tim Dierks. The transport layer security (tls) protocol version 1.2. Technical Report 5426, RFC Editor, 2008.

[20] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[21] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[22] Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International Conference on Fundamentals of Computation Theory*, pages 127–140. Springer, 1983.

[23] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. Technical report, RFC Editor, 2011.

[24] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*. Number 09. Stanford University Stanford, 2009.

[25] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Advances in cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

[26] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[28] Lein Harn and Guang Gong. Conference key establishment protocol using a multivariate polynomial and its applications. *Security and Communication Networks SCN 2015*, 8(9):1794–1800, 2015.

[29] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009.

[30] Geoffrey E Hinton, Terrence Joseph Sejnowski, and Tomaso A Poggio. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[31] Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995.

[32] Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in neural information processing systems*, pages 345–352, 1995.

[33] Mandal Kalikinkar, Gong Guang, and Liu Chuyi. Nike-based fast privacy-preserving multi-dimensional data aggregation for mobile devices. submitted, forthcomming.

[34] Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, 2007.

[35] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[36] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[38] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[39] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[40] Zhu Liehuang, Liao Lejian, Li Wenzhuo, and Zhang Zijian. An authenticated constant round group key agreement protocol based on elliptic curve cryptography. *IJCSNS International Journal of Computer Science and Network Security*, 6(8 B):131–134, 2006.

[41] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[42] Meeker M. 2016 internet trends report, 2016.

[43] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[44] Einar Mykletun and Gene Tsudik. Aggregation queries in the database-as-a-service model. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 89–103. Springer, 2006.

[45] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

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

[47] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, pages 129–140. Springer, 1991.

[48] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[49] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

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

[51] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[52] Goldwasser Shafi and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[53] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[54] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[55] Michael Steiner, Gene Tsudik, and Michael Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, 2000.

[56] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[57] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[58] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[59] Wikipedia contributors. Facebook–cambridge analytica data scandal — Wikipedia, the free encyclopedia, 2018. [Online; accessed 9-July-2018].

[60] Wikipedia contributors. General data protection regulation — Wikipedia, the free encyclopedia, 2018. [Online; accessed 9-July-2018].

[61] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[62] Jianxin Zhao, Richard Mortier, Jon Crowcroft, and Liang Wang. Privacy-preserving machine learning based data analytics on edge devices. In *AAAI/ACM conference on Artificial Intelligence, Ethics, and Society*, 2018.

[63] Long Leo Zhu, Chenxi Lin, Haoda Huang, Yuanhao Chen, and Alan Yuille. Unsupervised structure learning: Hierarchical recursive composition, suspicious coincidence and competitive exclusion. In *European Conference on Computer Vision*, pages 759–773. Springer, 2008.

APPENDICES

Appendix A

Fast Multipoint Evaluation Algorithm

For arbitrary points $x_0, x_1, \cdots, x_{n-1}$, multipoint evaluation can be done with $O(n^2)$ operations by using Horner's Method n times. It is reasonable to think that n evaluation of polynomial needs at least n^2 multiplication, however, we found a faster way from [58]. The idea is to divide the point set $\{x_0, x_1, \cdots, x_{n-1}\}$ into two halves and solve the problem locally. This method leads to a binary tree of depth $\log n$.

From a set of algebraic operations we can get for any polynomial $p(x)$ for which we want to evaluate an arbitrary point x_0 in a ring R, when $p(x) \mod (x - x_0)$ is denoted as $r(x)$

$$p(x_0) = r(x_0) \mod \mathbb{R}$$

Similarly, for a set of arbitrary points $x_0, x_1, \cdots, x_{n-1}$, if we denote $r(x) = p(x) \mod (x - x_0)(x - x_1) \cdots (x - x_{n-1}) \mod \mathbb{R}$, then

$$(p(x_0), p(x_1), \cdots, p(x_{n-1})) = (r(x_0), r(x_1), \cdots, r(x_{n-1})) \mod \mathbb{R}$$

We let $m_i = x - x_j$ and define

$$\begin{aligned} M_{i,j} &= m_{j \cdot 2^i} m_{j \cdot 2^i + 1} \cdots m_{j \cdot 2^i + 2^i - 1} \\ &= \prod_{0 \le l < 2^i} m_{j \cdot 2^i + l} \\ &= (x - u_{j \cdot 2^i})(x - u_{j \cdot 2^i + 1}) \cdots (x - u_{j \cdot 2^i + 2^i - 1)}) \end{aligned}$$

for $0 \leq i \leq k = \log n$ and $0 \leq j \leq 2^{k-i}$.

Then we have

$$\begin{aligned} M_{0,j} &= (x - u_j) \\ &= m_j \end{aligned}$$

and

$$\begin{aligned} M_{i+1,j} &= \prod_{0 \leq l < 2^{i+1}} m_{j \cdot 2^{i+1}+l} \\ &= \prod_{0 \leq l < 2^{i}} m_{j \cdot 2^{i+1}+l} \cdot \prod_{0 \leq l < 2^{i}} m_{j \cdot (2^{i+1}+2^{i}+l)} \\ &= \prod_{0 \leq l < 2^{i}} m_{2j \cdot 2^{i}+l} \cdot \prod_{0 \leq l < 2^{i}} m_{(2j+1) \cdot (2^{i}+l)} \\ &= M_{i,2j} \cdot M_{i,2j+1} \end{aligned}$$

We can produce subtree of $M_{i,j}$ as in algorithm 1.

Algorithm 1: Building up the subproduct tree

input : $x_0, x_1, \cdots, x_{n-1}$, where $n = 2^k$ for some $k \in \mathbb{N}$
output: The polynomials $M_{i,j}$ for $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$

```
foreach j = 0, ··· , n − 1 do
    M_{0,j} ← m_j
foreach i = 1, ··· , k do
    foreach j = 0, ··· , 2^{k−i} − 1 do
        M_{i,j} ← M_{i−1,2j} · M_{i−1,2j+1}
return
```

We now have a divide-and-conquer algorithm that, given all subproducts $M_{i,j}$, we can proceed our evaluation process in the algorithm 2.

It can be proved in [58] that the complexity of algorithm 2 is $O(n \log n)$.

Algorithm 2: Going down the subproduct tree

input : p(x) of degree less than $n = 2^k$ for some $k \in \mathbb{N}$, $x_0, x_1, \cdots, x_{n-1} \in \mathbb{R}$, and the subproducts $M_{i,j}$ from Algorithm 1

output: $p(x_0), p(x_1), \cdots, p(x_{n-1}) \in \mathbb{R}$

if *n = 1* **then**
  return $p(x_0)$
$r_0(x) \leftarrow p(x) \mod M_{k-1,0}$, $r_1(x) \leftarrow p(x) \mod M_{k-1,1}$
call the algorithm recursively to compute $r_0(u_0), \cdots, r_0(u_{n/2-1})$
call the algorithm recursively to compute $r_1(u_{n/2}), \cdots, r_0(u_{n-1})$
return $r_0(u_0), \cdots, r_0(u_{n/2-1}), r_1(u_{n/2}), \cdots, r_0(u_{n-1})$