Protecting Identity and Location Privacy in Online Environment

by

Xinxin Zhao

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved August 2015 by the
Graduate Supervisory Committee:

Guoliang Xue, Chair
Gail-Joon Ahn
Dijiang Huang
Yanchao Zhang

ARIZONA STATE UNIVERSITY

August 2015

ABSTRACT

The recent years have witnessed a rapid development of mobile devices and smart devices. As more and more people are getting involved in the online environment, privacy issues are becoming increasingly important. People's privacy in the digital world is much easier to leak than in the real world, because every action people take online would leave a trail of information which could be recorded, collected and used by malicious attackers. Besides, service providers might collect users' information and analyze them, which also leads to a privacy breach. Therefore, preserving people's privacy is very important in the online environment.

In this dissertation, I study the problems of preserving people's identity privacy and location privacy in the online environment. Specifically, I study four topics: identity privacy in online social networks (OSNs), identity privacy in anonymous message submission, location privacy in location based social networks (LBSNs), and location privacy in location based reminders. In the first topic, I propose a system which can hide users' identity and data from untrusted storage site where the OSN provider puts users' data. I also design a fine grained access control mechanism which prevents unauthorized users from accessing the data. Based on the secret sharing scheme, I construct a shuffle protocol that disconnects the relationship between members' identities and their submitted messages in the topic of identity privacy in anonymous message submission. The message is encrypted on the member side and decrypted on the message collector side. The collector eventually gets all of the messages but does not know who submitted which message. In the third topic, I propose a framework that hides users' check-in information from the LBSN. Considering the limited computation resources on smart devices, I propose a delegatable pseudo random function to outsource computations to the much more powerful server while preserving privacy. I also implement efficient revocations. In the topic of location privacy in location

based reminders, I propose a system to hide users' reminder locations from an untrusted cloud server. I propose a cross based approach and an improved bar based approach, respectively, to represent a reminder area. The reminder location and reminder message are encrypted before uploading to the cloud server, which then can determine whether the distance between the user's current location and the reminder location is within the reminder distance without knowing anything about the user's location information and the content of the reminder message.

Dedicated to my husband Lingjun and my family

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Guoliang Xue for his continuous support and guidance of my study and research at Arizona State University, and for his patience, motivation, enthusiasm, and immense knowledge. The guidance, encouragement, and academic freedom that he has given me help me all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my Ph. D. study.

I would like to thank my dissertation committee members, Dr. Gail-Joon Ahn, Dr. Dijiang Huang, and Dr. Yanchao Zhang for their insightful advise and comments. I have been honored to co-author with, Lingjun Li, Dr. Gail-Joon Ahn, Dr. Huan Liu, Huiji Gao, and Gabriel Silva, Their unselfish help, insights, and feedbacks helped me improve my knowledge in the area. I am also grateful to my colleagues and friends: Dejun Yang, Xi Fang, Xinhui Hu, Xiang Zhang, Vishnu Kilari, Ruozhou Yu, Ziming Zhao, Lei Liu, Yan Wu, Yiming Jing, Qiang Zhang, and Yashu Liu for their friendship and care, which helped me survive through these years.

Finally and most importantly, none of my achievement would have been possible without the love and patience of my huaband and my family. My most sincere thanks go to my huaband Lingjun Li for his endless support and love during the past six years. I am very thankful to my parents, Xia Chen and Xuewu Zhao, for their dedication and unconditional support throughout my life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

Introduction

Privacy has always been a very important societal issue. As more and more people involve in the digital world, this issue is becoming increasingly important. In the online environment, people are increasingly concerned about their privacy, since every action people take could leave a trail of information that could be recorded and used in the future. For example, people like to keep their opinions and daily thinkings on Blogs or online social networks; also, people are getting used to location based services on smart phones, while these services can obtain and accumulate people's location information. These information could be collected, accumulated, and analyzed by service providers. Trusting these providers are risky and they may abuse these sensitive information for profits, which could bring unwanted troubles to people. However, the existing digital world is built on top of these services and cannot live without them. Thus, preserving people's privacy is important and challenging in current online environment.

In this dissertation, we study the problems of protecting people's identity privacy and location privacy in the online environment. We focus on four topics: identity privacy in online social networks (OSNs), identity privacy in anonymous message submission, location privacy in location based social networks (LBSNs), and location privacy in location based reminders. Current OSNs give people a chance to share personal information (e.g. their thoughts, photos) with their family and friends. Most people don't know to properly protect their privacy, although they are not willing to share their private information to strangers who can view or use their information. Most OSN providers would thus provide some

sort of access control mechanism to restrict access from unwanted users. However, this protection only defends against the third party attacker but fails on first or second party attack. For example, many OSN providers will store user's data on a partnering storage site, which may access users' data. Also, the OSN provider itself can always access to user's data within this mechanism. When these first and second parties are not trusted, we need a new and strong mechanism to protect users' data. Another useful application of protecting identity privacy is anonymous online message submission. The participants might be willing to provide useful information but do not want anyone link their identity to the submitted information, since the information they provide might be their health information, their feedbacks about their supervisors, etc. Again, we should not rely on the information collector to provide the anonymity since the collector itself may be the one you want to defend against. An example is a company's performance evaluation, where the evaluation collection system may be eventually accessible to the boss of the company. Based on the secret sharing scheme, we construct a shuffle protocol. All members' messages are shuffled to bury their submitting order. After all messages have been collected, the message collector does not know who submitted which message.

The smart phone market has grown very fast in recent years. In addition to traditional functionalities (e.g., text messaging and making phone call), current smart phones are equipped with various sensors. According to [87], most smart phones are equipped with high-precision localization sensors, usually based on GPS receivers, access points or triangulation with nearby base stations. These localization sensors enable smart phones to locate themselves at any time. Thus, many location based applications emerge to the market, including location based social networks, location based reminders, etc. While people are enjoying the convenience these applications have brought, most of them are worried about the breach of location privacy. The leakage of people's location might lead to illegal

stalking, or even physical criminals. These worries may also impede the development of the location based services. Thus, the protection of location privacy is essential for both users and location based services.

1.1 Overview and Contributions

This dissertation focuses on the privacy issues in the online environment. Particularly, this dissertation studies two essential parts of privacy – identity privacy and location privacy. In the digital world, users grant a server or service provider to use their information to provide a certain service. Although we may assume that the service provider will protect user's information due to reputation pressure, this assumption is not always true, especially to small business providers. Also, malicious attackers are always interested in people's identity and location privacy. They will try every possibility to obtain the information, like hacking into the system or promising high benefits for private information. Throughout this dissertation, we assume the server is **NOT** fully trusted, and under this assumption, we construct secure mechanisms to protect users' identity privacy and location privacy.

1.1.1 Identity Privacy in Online Social Networks

Users of OSNs form new relationships and share their information in the networks. However, for some private information, the user might only be willing to share them with their family and friends rather than strangers. Therefore, access control is a basic feature in OSNs. A user assigns her online friends to one of several groups (e.g. family, colleagues, roommates) and grants different authorizations to her personal data. If a user accepts a friend request from another user, she grants this user with corresponding credentials to access her personal information. All users' data are stored on a storage site provided by the OSN provider. The storage site is untrusted. In our design, all of the uploaded data are en-

crypted to prevent the storage site from unauthorized access while the storage site can still retrieve the correct information when a user gets access to it. In addition, we hide users' identity when they access the storage site.

In summary, our contributions are as follows.

- Only the authorized user can access the encrypted data stored in the storage site. Users' identity is hidden from the storage site when she accesses the data. We use zero knowledge proof of knowledge to construct oblivious transfer which enables the storage site to verify the identity and credentials of the user.
- We design a fine-grained access control mechanism to prevent users from unauthorized access. Only users with valid credentials can access the data.

1.1.2 Identity Privacy in Anonymous Message Submission

In the real world, anonymous message submission is easy to realize. People just cast their ballots into a locked and opaque box. The box is then shaken to destroy the casting order. After that, all the ballot papers are collected and submitted to a collector. If we assume that the collector cannot tell the hand writings of each participant, the collector will not be able to infer the relationship between the ballots and the identities of the participants. However, in the digital world, we cannot realize anonymous message submission in a similar way as in the real world, because a secure ballot box trusted by all of the participants does not exist in the digital world. Besides, the shaking process has to be done under the supervision of all the participants and it is difficult to reproduce in a distributed network.

In this dissertation, we propose a protocol to enable participants to anonymously submit their messages online. We propose a technique which can secretly aggregate users' mes-

sages into a message vector. In the final vector, each participant does not know the position of the other members' message and the message collector cannot connect each participant's identity to their message. We use secret sharing scheme to construct the protocol. Compared to the work of [21], the use of the secret sharing scheme makes the protocol efficient towards to a practical size group.

In summary, our contributions are as follows.

- We propose an efficient anonymous message submission protocol named AMS for groups of practical size. Our protocol executes in time bounded by a polynomial in the group size, and has a constant communication rounds.
- We theoretically prove the anonymity of our AMS protocol. It is collusion resistant under malicious attacks. The execution of the AMS protocol does not rely on a trusted third party.
- We propose a technique to let each member acquire a position in the submitted message sequence such that every member does not know other members' positions. Specifically, we use a vector to represent the sequence. A position in the sequence is mapped to an index in the vector. This technique is implemented using a probability algorithm and has a high success probability.
- We analyze our protocol from the aspects of security and efficiency. We conduct comprehensive simulations to show that our protocol is efficient compared to previous works.

1.1.3 Location Privacy in Location Based Social Networks

Location based social network (LBSN) applications are now enjoying a great deal of attention because of its rapid development on smart phones. Many companies are rushing to have LBSN included in their services. In LBSNs, users ”check in” at a venue and leave “tips” which serves as their personal suggestions for this venue. In this dissertation, we call all the information a user shares at a venue as her ”check-in”, which includes her tips about this venue, the check-in time, etc. Another feature of LBSNs is that users can search a venue and see their friends' check-ins when they gets to a new place.

As more and more people involve in the LBSN applications on smart phones, the privacy issues are becoming urgent. Currently, users need to transmit their location information, e.g., GPS location, to the LBSN server when they check in at a venue, share their information with friends, or search within the LBSN for information. The breach of users' location privacy might lead to unexpected troubles.

In this dissertation, we propose a framework that can hide users' location from untrusted LBSN server while the functionalities in the LBSN can still be performed. Technically, in this framework, the user encrypts the data and generates a check-in trapdoor before uploading the check-in records to the LBSN server. The LBSN server converts the check-in trapdoor to a search trapdoor, which is available to the user's friends. If any of the user's friends wants to search her check-in records, this friend generates corresponding search trapdoor and sends it to the LBSN server. Using the search trapdoor, the server searches corresponding encrypted record and sends the results back to the user's friend.

In summary, our contributions are as follows.

- We propose a framework that can hides users' location from the LBSN server. We

also propose a method which can reduce the processing time of the same check-ins frequently.

- We propose a delegatable pseudo random function which outsources the user side computation to the server side to save the limited computation resources of users' smart devices.

- Since revocation happens more frequently in LBSNs, the re-encryption of every record causes a large computation overhead for users after each revocation happens. To solve this issue, we propose a hash chain based session key generation scheme so that users do not need to re-encrypt all of the records after each revocation takes place.

1.1.4 Location Privacy in Location Based Reminders

Reminder applications are essential applications in mobile devices. In traditional reminder applications, users set up a future time to remind her things to do at that time. The mobile device will alert the user at the reminder time. However, in this dissertation, our focus is location based reminder applications rather than time based reminder applications. Location based reminder applications are developed with the development of the high precision localization sensor equipped in smart devices. In the location based reminder application, the user marks a reminder location and sets up a reminder distance in the reminder application. The smart device will alert the user when her current location is within the reminder distance of the reminder location. Recently, the development of cloud service has attracted many attentions, many companies now use the cloud to synchronize personal data, including reminders. In other words, users' reminder locations are stored in the cloud. Therefore, we propose the topic of preserving users' location privacy from untrusted cloud server.

In this dissertation, we construct a secure cloud-assisted location based reminder system. The system hides a user's reminder message, reminder location, and her current location from the cloud server, while the cloud server is still able to determine whether the distance between the user's current location and the reminder location is smaller than the reminder distance. We propose a novel method to represent location points. Technically speaking, we divide the earth surface into small squares. We use each square to represent the infinite location points contained in this square. We use a set of squares to represent a reminder area. Using searchable symmetric encryption (SSE), the square information and the reminder message is encrypted and uploaded to the cloud server. The use of SSE enables the server to search the encrypted data without knowing the content of the data.

In summary, our contributions are as follows.

- We propose a secure cloud-assisted location based reminder system. The system hides users' location information and reminder message from the untrusted cloud server. We prove the security of our system and did extensively simulations to show its efficiency on Motorola Droid phone.
- We propose a cross based area representation approach and an improved bar based area representation approach, respectively, to represent a reminder area. They are easy to construct and efficient in space overhead. The implementation of these approaches also saves a lot of search time in the cloud server.

Part I

IDENTITY PRIVACY

CHAPTER 2

Identity Privacy in Online Social Networks

In today's society, people spend more and more time in online social networks (OSNs), getting news, sharing photos and discovering new relationships. It is reported that facebook, which is the most popular OSN, has more than 500 million active users. 700 million minutes are spent each month on facebook [48]. An OSN operator usually rents storage facilities from untrusted storage sites to meet the increasing need of storing users' personal data. Therefore, the protection of personal data from unauthorized access is becoming a critical concern in OSNs.

Until now, most of the existing works about privacy preservation focus on protecting data privacy. However, identity privacy is of the same importance as data privacy. For example, if the ciphertext of some personal data is stored in a third party storage site, even though adversaries cannot decrypt the data, since they do not have private keys, they know who has accessed it. Gradually, they could obtain most of the relation graph of the OSN. From this graph they can identify other nodes as long as several nodes' identities are leaked. Unfortunately, to the best of our knowledge, few works deal with the identity privacy issue in the OSN.

Camenisch presented a protocol [24] for anonymous access to the records stored in an untrusted database and each record has an access control list. They used zero-knowledge proofs of knowledge to construct oblivious transfer. Oblivious transfer, first introduced by Rabin[84], and later extended by Even and Goldreich [47][20], is described as follows [26]: There is a sender with N records $(R_1, \cdots, R_N)$ and a receiver with a secret choice

$\sigma \in (1, \cdots, N)$, the receiver requests the record R_σ from the sender in such a way that the sender learns nothing about σ and the receiver learns nothing about the records except R_σ. However, their work on access control is static. Once people obtain the credentials, they can always access all records associated with these credentials and will not be revoked. While this is suitable in a database environment, it cannot be applied to an OSN environment.

In this dissertation, we propose a system which can hide users' identities from adversaries when they visit the data stored in an untrusted storage site. We also design a fine-grained access control mechanism to prevent authorized access. It contains two lists: an access list (AL) and a revocation list (RL). AL allows a user to define which group of friends can access her information, while RL allows a user to revoke individuals from future accessing. In the OSN, usually users in the same group have the same authorization to access the data owner's information. However, sometimes the data owner may want to hide certain information from some users in the group. For example, Alice is a friend of Bob, and she belongs to his "college" group. Bob wants to plan a birthday party for Alice and hide this information from Alice until the party is carried out. To do so, Bob adds Alice to the RL. In this way, all the "colleague" group members except Alice could access the data associated with the group.

We make the following contributions:

- The authorized users can anonymously access the data in the storage site such that the storage site is oblivious to either the visitor's identity or the index of the accessed data stored in the storage site. Users do not need to show their identities in access processes and zero-knowledge proofs based oblivious transfer are used to conceal the users' identities as well as the index of the data.

- We design a fine-grained access control mechanism which allows users to prevent

unauthorized access. It allows the data owner to set group-based access control policy, i.e., the data owner can set which group of friends can access the data, and revoke individuals from future accessing.

- We prove the security of our system within the universal composability (UC) framework [27]. This framework allows our system executing composably with other UC secure systems without sacrificing any security guarantee. More importantly, UC security guarantees that our system can be concurrently executed, in which an adversary cannot get advantages by running several instances of our system in parallel. This property is especially useful for the Internet circumstance because different users probably execute instances of our protocols at the same time.

2.1 Related Work

As far as we know, current study on OSN security mainly focuses on anonymizing and de-anonymizing social structure [60] [99], defense on sibyl attack [44] [95] [104], and privacy-preserving computing [72] [80]. Although privacy issues in OSNs, especially anonymity issues, are significant to their development, to the best of our knowledge, there are few works discussing these issues. Sun *et al.* [93] proposed a privacy-preserving scheme for OSNs. In their work, the authors presented a scheme to protect data privacy with regard to the data stored in third party storage site. Public key encryption with key word search scheme and identity-based encryption are used to enable a user with proper role to efficiently retrieve the data stored in the storage site. However, their work did not provide an identity concealing mechanism. Baden *et al.* designed a scheme called Persona which allows a user to define fine-grained privacy policy and authorize her friends to access the users' records depends on which group the friend belongs to [6]. They used attribute based encryption to achieve the access control. However, they did not deal with the identity con-

cealing and revocation issues. Domingo-Ferrer *et al.* [45] utilized homomorphic encryption algorithms to enable resource access without a trusted mediating party. This scheme prevented the resource owner from learning the relationship and trust level of the parties who collaborated to access the resource. In [91], Squicciarini *et al.* leveraged game theory to achieve cooperative privacy management on shared data. In [24], Camenisch introduced a protocol which could hide users' identity from database. As we said in the previous part, their access control policy is static, which is not suitable for the OSN circumstance.

2.2 Problem Formulation

In this section, we present the system model, the threat model, and define the security objectives in the context of OSNs.

2.2.1 System Model

There are three roles in our system: data owners, visitors and the storage site.

Data owners (denoted as DO) are the ones who manage their homepages and friend list. They classify their friends into several groups (e.g., classmates, family and colleagues) and issue corresponding credentials to each group. When some data is uploaded, the data owner generates AL and RL for the data.

Visitors (denoted as V) are the ones who visit other users' homepages in OSNs. They belong to a certain group when they become friends with the data owner. When they visit other user's homepage, they may not want any third party to know the relationship between them. In fact, both data owners and visitors are the users of the OSN. The difference lies into their different actions. When the visitor manages her data and friend list, she is the data owner. When she visits other user's homepage, she becomes the visitor.

The storage site (denoted as *DB*) is an untrustworthy third storage provider. It signs a contract with the OSN operator and provides storage service for all OSN users.

2.2.2 Threat Model

The adversary considered in this dissertation is fully malicious and can arbitrarily deviate from the protocol specification. Furthermore, the adversary controls all the channels in the system such that they can eavesdrop, block or tamper any message transmitted between two honest participants. The adversary can also replay or deliver out-of-order messages to an honest participant. In addition, the adversary is able to control any participant in our system. However, we do not consider the trivial scenario that the adversary controls all the participants in the system. We assume the adversary is computationally bounded, i.e., it could be any probabilistic polynomial time algorithm. It is reasonable because it is infeasible for an adversary to obtain an unbounded computation resource.

2.2.3 Security Objectives

Our system is designed to achieve the following security objectives.

- Access Control: Visitors who do not have credentials or are revoked cannot access the data.
- Identity Hiding: The storage site can verify a visitor's credentials without knowing the identity of the visitor and the index of the data when the visitor accesses the storage site.
- Collusion Resistance: If the database and the visitor or the data owner and the visitor collude, they cannot benefit from the collusion, i.e., they cannot get more from the

collusion than what they already have.

The security of our system is proved by indistinguishability of a real world and an ideal world in the UC framework. In the real world, all honest parties follow the protocol execution. The parties who are corrupted by the adversary can arbitrarily deviate from the protocol specification. In the ideal world there is an incorruptible trusted party T to implement all the functionalities of our protocols. All parties communicate through T. There are ideal data owners DO', ideal storage site DB', and ideal visitors V' in the ideal world. Upon receiving each party's message, T responses properly. In both of the ideal world and the real world, there exists an environment ξ who observes all honest parties' inputs, outputs, and the view of corrupted parties. The security of a system is defined as follows [27]:

Definition 1. *A system is said to be securely implement a functionality if for an environment ξ and any adversary $\mathscr{A}$, we can construct a simulator Sim in the ideal world, such that ξ cannot distinguish whether it is communicating with Sim in the ideal world, or it is communicating with the adversary $\mathscr{A}$ in the real world.* □

The trusted party T holds the tuple (D, AL, RL) which represents the data, its corresponding AL and RL, and maintains a group list $\mathscr{G}_{ID}$ for each visitor V who selects an identity number ID.

- Initially, T sets its internal database $DB_T \leftarrow \perp$.
- Upon receiving $(Upload, D, AL, RL)$ from DO', T adds (AL, RL) and (AL, RL, D) to DB' and DB_T, respectively.
- Upon receiving $(Join, ID, \mathscr{G}_i, DO')$ from V', T sends $(Join, ID, \mathscr{G}_i)$ to DO'. If DO' sends back $b = 1$ then T adds $\mathscr{G}_i$ to $\mathscr{G}_{ID}$, otherwise T just returns b to V'.

- Upon receiving $(Retrieve, \sigma)$ from V', where σ is the index of the data, T processes as follows. If $DB_T \neq \perp$, T sends $Retrieve$ to DB', then DB' sends back a bit b. If $b = 1 \wedge AL_\sigma \subseteq \mathscr{G}_{ID} \wedge ID \notin RL_\sigma$, T sends D_σ to V', otherwise it just returns $\perp$ to V'.

2.3 System Overview

Our system consists of four protocols: (1) the storage site and data owner initialization protocol, (2) joining in a group protocol, (3) uploading data protocol, and (4) retrieving data protocol. We briefly review the construction of our system.

Initialization: In the initialization protocol, the whole system is initialized. The storage site and the data owner generate their public/private key pairs.

Joining in a group: In this protocol, a visitor sends a friend request to a data owner. If the data owner approves the visitor's request, she then decides which group she wants the visitor belongs to by issuing a corresponding credential and private keys. The credential is a kind of the data owner's signature on the group the visitor joins. We assume that the communications between the data owner and the visitor are authenticated.

Uploading Data: Each time of executing the uploading data protocol, the data owner uploads one piece of data D. Then the data owner generates AL and RL for data D and sends (AL, RL) to the storage site. The storage site signs the AL, and returns the signature to the data owner. On the data owner side, she encrypts the data and uploads the ciphertext to the storage site. The storage site then publishes the ciphertext, its AL and RL, i.e., by posting them on the website.

Retrieving Data: If a visitor wants to retrieve some data in the storage site, she should prove to the storage site that she has credentials. Zero-knowledge proofs guarantee that the storage site can verify whether the visitor has valid credentials or not without knowing the

identity of the visitor or the index of the data. After confirming that the visitor has valid credentials, the storage site returns a random number R for decrypting the data. Only if the visitor's identity number $ID \notin RL$, she can get the plaintext using R and her private keys issued by the data owner.

2.4 Technical Preliminaries

In this section, we introduce the technical preliminaries of our system.

2.4.1 Bilinear Map

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic multiplicative groups with prime order p. A bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$ is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathbb{T}$, such that for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u,v)^{ab}$.

Bilinear map has the following properties.

Bilinear: $e(u^a, v^b) = e(u,v)^{ab}$;

Non-degenerate: $e(g_1, g_2) \neq 1$, where g_1, g_2 are respective generators of groups $\mathbb{G}_1, \mathbb{G}_2$;

Computable: It can be efficiently computed.

Usually we regard $\mathbb{G}_1$ and $\mathbb{G}_2$ as the same group, which is denoted as $\mathbb{G}$.

2.4.2 Complexity Assumptions

q-Decisional Multi-Exponent Bilinear Diffie-Hellman Assumption (q-MEBDH) [70]: Given a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with a prime order $p > 2^k$, we say that the q-MEBDH problem

holds if for all polynomial-time adversaries $\mathscr{A}$, the advantage $Adv^{q\text{-MEBDH}}_{\mathbb{G},\mathbb{G}_T}$ given by

$$Pr[\mathscr{A}(J, T = (e(g,g)^{\alpha z}) = 1)] - Pr[\mathscr{A}(J, T = num) = 1]$$

is a negligible function in k, where $num \xleftarrow{\$} \mathbb{G}_T$, J is a tuple of

$$\Big(g, g^z, e(g,g)^{\alpha},$$
$$\forall 1 \le i, j \le q, (g^{a_i}, g^{a_i z}, g^{a_i a_j}, g^{\alpha/a_i^2})$$
$$\forall 1 \le i, j, k \le q, i \ne j, (g^{a_i a_j z}, g^{\alpha a_j/a_i^2}, g^{\alpha a_i a_j/a_k^2}, g^{\alpha a_i^2/a_j^2})\Big),$$

and $z, \alpha, a_1, \cdots, a_q \xleftarrow{\$} \mathbb{G}$. Here, $\alpha \xleftarrow{\$} \mathbb{G}$ denotes α is randomly chosen from $\mathbb{G}$. In other words, given J, the adversary cannot distinguish a random number $num \in \mathbb{G}_T$ from $e(g,g)^{\alpha z}$.

q-Strong Diffie-Hellman Assumption (q-SDH) [14]: Given a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with a prime order $p > 2^k$, g is a generator of group $\mathbb{G}$. q-SDH problem can be stated as follows. The advantage for adversary $\mathscr{A}$

$$Adv^{q\text{-SDH}}_{\mathbb{G},\mathbb{G}_T} = Pr[\mathscr{A}(g, g^x, \cdots, g^{x^q}) = (c, g^{1/(x+c)})]$$

is a negligible function in k, where $x, c \xleftarrow{\$} \mathbb{Z}_p$.

q-Bilinear Diffie-Hellman Exponent Assumption (q-BDHE) [16]: Given a bilinear group $(\mathbb{G}, \mathbb{G}_{\mathbb{T}})$ of prime order $p > 2^k$, g, g_T are their generators respectively. q-BDHE problem can be stated as follows. The advantage for adversary $\mathscr{A}$ to solve q-BDHE problem in polynomial time

$$Adv^{q\text{-BDHE}}_{\mathbb{G},\mathbb{G}_T} =$$
$$Pr[\mathscr{A}(g, g_T, g^a, \cdots, g^{a^{q-1}}, g^{a^{q+1}}, \cdots, g^{a^{2q}}, e(g,g_T)^{a^q}) = 1]$$
$$- Pr[\mathscr{A}(g, g_T, g^a, \cdots, g^{a^{q-1}}, g^{a^{q+1}}, \cdots, g^{a^{2q}}, S) = 1]$$

is a negligible function in k, where $S \xleftarrow{\$} \mathbb{G}_T$ and $a \xleftarrow{\$} \mathbb{Z}_p$. That is to say, given g, g_T, g^a, $\cdots$, $g^{a^{q-1}}$, $g^{a^{q+1}}$, $\cdots$, $g^{a^{2q}}$, the probability that $\mathscr{A}$ can distinguish $e(g,g_T)^{a^q}$ with a random number $S \in \mathbb{G}_T$ is negligible.

2.4.3 Broadcast Encryption

Broadcast encryption was first proposed by Fiat [49]. It can encrypt broadcasting content in such a way that only intended users can decrypt the content. Users who are not authorized to assess the content cannot get it even if they have the private key. Broadcast encryption has several applications, including file systems, group communication, satellite TV subscription services, and DVD content protection [16] [70]. In all of the applications, the concept of revocation is very important. For example, in a group communication, if one user's private key has been comprised, we should revoke him from future broadcast.

In our system, we integrate the protocol by Lewko and Sahai [70] into our system to revoke a visitor from future accessing. In summary, the broadcast encryption algorithm consists of four steps.

Setup: A public key *pk* and a master private key *sk* are generated by the algorithm;

Key Generation: The algorithm takes *sk* and the user's *ID* as input to generate each user's private keys (P_0, P_1, P_2);

Encryption: Using *RL*, *pk* and *D* as input, the data *D* is encrypted into a broadcast ciphertext.

Decryption: For a user whose identity number $ID \notin RL$ can use her private keys to decrypt the ciphertext.

2.4.4 Signature Scheme

In our protocol, we use two signature schemes, one for AL signature and one for credential signature.

2.4.4.1 AL Signatures Scheme

We use the modification of Boneh's signature scheme [14] for our AL signature. Given a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with a prime order $p \geq 2^k$. g is a generator of group $\mathbb{G}$. We choose $x_e, x_1, \cdots, x_l \xleftarrow{\$} \mathbb{Z}_p$, and compute $y_e = g^{x_e}, y_1 = g^{x_1}, \cdots, y_l = g^{x_l}$. The private keys are $(x_e, x_1, \cdots, x_l)$, the public keys are $(g, y_e, y_1, \cdots, y_l)$. Suppose the messages are $(r_m, \mathscr{G}_1, \cdots, \mathscr{G}_i)$, where $i \leq l$. The signature on the message tuple is $E \leftarrow g^{\frac{1}{x_e + r_m + \Sigma_{j=1}^{i} x_j \mathscr{G}_j}}$. It can be verified by checking whether the equation $e(E, y_e g^{r_m} \prod_{j=1}^{i} y_j^{\mathscr{G}_j}) = e(g, g)$ holds. This signature scheme is said to be secure against weak chosen-message attack [24].

2.4.4.2 Credential Signature Scheme

We utilize the signature scheme proposed by Au *et al.* [5] for our credential signature. Given a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with a prime order $p \geq 2^k$. g, h_0, h_1, h_2 are random generators of $\mathbb{G}$. We choose $x \xleftarrow{\$} \mathbb{Z}_p$, and compute $y = g^x$. The secret key is x. The public keys are (y, g, h_0, h_1, h_2). Suppose the messages to be signed are (c_1, c_2). We choose $s, r \xleftarrow{\$} \mathbb{Z}_p$. The signature is $A \leftarrow (g h_0^{c_1} h_1^{c_2} h_2^r)^{\frac{1}{x+s}}$. The signature can be verified by checking whether $e(A, g^s y) = e(g h_0^{c_1} h_1^{c_2} h_2^r, g)$ holds.

2.4.5 Zero-Knowledge Proof

A zero-knowledge proof is to let others verify the proof of knowledge in a zero-knowledge way, i.e., the proof leaks nothing about the knowledge itself. Using zero-knowledge proofs, we are able to prove 1) a discrete logarithm modulo a prime [85], 2) the equality of some discrete logarithms [32] and 3) the conjunction of the previous two [40]. Given a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with prime order p, we follow the notation introduced in [24] for constructing a zero-knowledge proof. For example, $PK\{(\alpha, \beta) : Y = g^\alpha h^\beta\}$ denotes a zero-

knowledge proof of α, β, where $Y = g^{\alpha} h^{\beta}$. Here, $Y \in \mathbb{G}$, $\alpha, \beta \in \mathbb{Z}_p$ and $\mathbb{G} = \langle g \rangle = \langle h \rangle$. α and β are secrets for the prover to prove, g, h and Y are public parameters. Given this notation, one can easily derive an efficient protocol [23] [25] to realize the proof.

Now we construct a zero-knowledge proof of the credential signature. Assume we are given a signature (A, s, r) on (c_1, c_2). We prove that we indeed possess such a signature without leaking the signer's information. For this purpose, we need to randomize the public keys by choosing values w_1, w_2, w_3, w_4, w_5, w_6, $w_7 \stackrel{\$}{\leftarrow} \mathbb{G}$ such that $\log_g w_1$, $\log_g w_2$, $\log_g w_3$, $\log_g w_4$, $\log_g w_5$, $\log_g w_6$, $\log_g w_7$ are not known. This can be done by choosing random values $t_1, t_2, t_3, t_4, t_5, t_6, \gamma, \delta, \eta, \rho, \phi, \psi \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, computing $\tilde{A} \leftarrow Au^{\gamma}$, $\tilde{g} \leftarrow gu^{\delta}$, $\tilde{y} \leftarrow yu^{\eta}$, $\tilde{h_0} = h_0 u^{\rho}$, $\tilde{h_1} = h_1 u^{\phi}$, $\tilde{h_2} = h_2 u^{\psi}$, $B = w_2^{\delta} w_1^{t_1}$, $C = w_3^{\gamma} w_1^{t_2}$, $S = w_4^{\psi} w_1^{t_3}$, $T = w_5^{z_1} w_1^{t_4}$, $W = w_6^{z_7} w_1^{t_5}$, $Z = w_7^{z_5} w_1^{t_6}$, and executing the following proofs of knowledge:

$$PK\{(c_1, c_2, s, r, t_1, t_2, t_3, t_4, t_5, t_6, \gamma, \delta, \eta, \rho, \phi, \psi) : \quad Proof(c_1, c_2, s, r, A, g, y, h_0, h_1, h_2)\}.$$

Here, $Proof(c_1, c_2, s, r, A, g, y, h_0, h_1, h_2)$ represents

$$\begin{aligned}
&B = w_2^{\delta} w_1^{t_1} \wedge 1 = B^{-s} w_2^{z_1} w_1^{z_2} \wedge 1 = B^{-c_1} w_2^{z_3} w_1^{z_4} \wedge \\
&1 = B^{-c_2} w_2^{z_5} w_1^{z_6} \wedge 1 = B^{-r} w_2^{z_7} w_1^{z_8} \wedge 1 = B^{-\delta} w_2^{z_9} w_1^{z_{10}} \wedge \\
&1 = B^{-\rho} w_2^{z_{11}} w_1^{z_{12}} \wedge C = w_3^{\gamma} w_1^{t_2} \wedge 1 = C^{-s} w_3^{z_{13}} w_1^{z_{14}} \wedge \\
&1 = C^{-\eta} w_3^{z_{15}} w_1^{z_{16}} \wedge S = w_4^{\psi} w_1^{t_3} \wedge 1 = S^{-r} w_4^{z_{17}} w_1^{z_{18}} \wedge \\
&T = w_5^{z_1} w_1^{t_4} \wedge 1 = T^{-\gamma} w_5^{z_{19}} w_1^{z_{20}} \wedge W = w_6^{z_7} w_1^{t_5} \wedge
\end{aligned}$$

$$1 = T^{-\psi} w_5^{z_{21}} w_1^{z_{22}} \wedge Z = w_7^{z_5} w_1^{t_6} \wedge 1 = Z^{-\phi} w_7^{z_{23}} w_1^{z_{24}} \wedge$$

$$\frac{e(\tilde{A},\tilde{y})}{e(\tilde{g},\tilde{g})} = e(\tilde{A},w_1)^{\eta} e(\tilde{A},w_1)^{z_1} e(\tilde{A},\tilde{g})^{-s} e(\tilde{y},w_1)^{\gamma}$$

$$e(\tilde{h_0},\tilde{g})^{c_1} e(\tilde{h_0},w_1)^{-z_3} e(\tilde{h_1},\tilde{g})^{c_2} e(\tilde{h_1},w_1)^{-z_5} e(\tilde{h_2},\tilde{g})^{r}$$

$$e(\tilde{h},w_1)^{-z_7} e(\tilde{g},w_1)^{z_{13}} e(\tilde{g},w_1)^{-z_{17}} e(\tilde{g},w_1)^{-\rho} e(\tilde{g},w_1)^{-\phi}$$

$$e(\tilde{g},w_1)^{-\delta} e(\tilde{g},w_1)^{-\delta} e(w_1,w_1)^{z_9} e(w_1,w_1)^{-z_{15}}$$

$$e(w_1,w_1)^{-z_{19}} e(w_1,w_1)^{z_{21}} e(w_1,w_1)^{z_{11}} e(w_1,w_1)^{z_{23}},$$

where $z_1 = s\delta, z_2 = st_1, z_3 = \delta c_1, z_4 = t_1 c_1, z_5 = \delta c_2, z_6 = t_1 c_2, z_7 = r\delta, z_8 = t_1 r, z_9 = \delta\delta, z_{10} = t_1\delta, z_{11} = \delta\rho, z_{12} = t_1\rho, z_{13} = s\gamma, z_{14} = st_2, z_{15} = \eta\gamma, z_{16} = \eta t_2, z_{17} = \psi r, z_{18} = t_3 r, z_{19} = z_1\gamma = s\delta\gamma, z_{20} = t_4\gamma, z_{21} = z_7\psi = r\delta\psi, z_{22} = t_5\psi, z_{23} = z_5\phi = \delta c_2\phi$ and $z_24 = t_6\phi$.

2.5 Construction of System

In this section, we describe the details of our system.

The initialization protocol consists of two parts: the storage site initialization and the data owner initialization. The protocol initializes the whole system and enables the storage site and the data owner to generate their public/private key pairs.

Figure 2.1 shows the details of the initialization protocol. In Figure 2.1a, the storage site utilizes a security parameter k to generate a bilinear group pair $(\mathbb{G}, \mathbb{G}_T)$ with a prime order $p > 2^k$. Then it generates its public/private key pair (pk_{DB}, sk_{DB}). g, h, g_B and h_B are random generators of $\mathbb{G}$. In Figure 2.1b, using p as input, the data owner generates her bilinear group pair $(\tilde{\mathbb{G}}, \tilde{\mathbb{G}_T})$ with the prime order p. The public/private key pair (pk_{DO}, sk_{DO}) is generated from the bilinear group. g_1, g_2, h_0, h_1 and h_2 are random generators of $\tilde{\mathbb{G}}$.

Figure 2.2 shows the joining in a group protocol. In the protocol, if a visitor's friend request is approved by a data owner, this visitor gets a credential on the group she joins and private

Function: Storage site initialization.
$(\mathbb{G}, \mathbb{G}_T, p) \xleftarrow{\$} \mathsf{Pg}(1^k)$; $g, h, g_B, h_B \xleftarrow{\$} \mathbb{G}$; $M \leftarrow e(g,h)$; $x_e \xleftarrow{\$} \mathbb{Z}_p$; $y_e \leftarrow g^{x_e}$
For $i = 1, \cdots, l$ do $x_i \xleftarrow{\$} \mathbb{Z}_p$; $y_i \leftarrow g^{x_i}$
$sk_{DB} \leftarrow (x_e, h, (x_i)_{i=1}^l)$; $pk_{DB} \leftarrow (g, y_e, M, (y_i)_{i=1}^l, g_B, h_B, p)$
Return (sk_{DB}, pk_{DB})

(a) Storage Site Initialization

Function: Data owner initialization.
$(\tilde{\mathbb{G}}, \tilde{\mathbb{G}_T}) \xleftarrow{\$} \mathsf{Pg}(p)$; $g_1, g_2, h_0, h_1, h_2 \xleftarrow{\$} \tilde{\mathbb{G}}$; $x_{DO}, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_p$; $y_{DO} \leftarrow g_1^{x_{DO}}$
$sk_{DO} \leftarrow (x_{DO}, \alpha, \beta)$; $pk_{DO} \leftarrow (y_{DO}, g_1, h_0, h_1, h_2, g_2^{\alpha}, g_2^{\beta})$
Return (sk_{DO}, pk_{DO})

(b) Data Owner Initialization

Figure 2.1: Initialization Protocol

Function: A visitor V sends a request to join a group. The data owner DO sends V corresponding credentials.

1. If it is the first time the visitor V joins the OSN, she picks $z_V \xleftarrow{\$} \mathbb{Z}_p$, computes $ID \leftarrow h_0^{z_V}$, and parses her state as $st_V \leftarrow (z_V, ID, 0, \emptyset, \emptyset, \emptyset)$; otherwise the visitor parses her state as $(z_V, ID, count, \mathscr{G}, Cred, P)$.
2. The visitor V sends ID to the data owner DO, and does a zero knowledge proof of $PK\{(z_V) : ID = h_0^{z_V}\}$ to DO.
3. If ID already exists or is not accepted, DO returns $\perp$; otherwise DO computes $X \leftarrow g_2^{\alpha}$ and $Y \leftarrow g_2^{\beta}$, and does a zero zero knowledge proof of $PK\{(\alpha, \beta) : X = g_2^{\alpha} \wedge Y = g_2^{\beta}\}$ to V.
4. The data owner DO chooses $s_{\mathscr{G}_i}, r_{\mathscr{G}_i}, t \xleftarrow{\$} \mathbb{Z}_p$, computes $A \leftarrow (g_1 ID h_1^{\mathscr{G}_i} h_2^{r_{\mathscr{G}_i}})^{\frac{1}{x_{DO}+s_{\mathscr{G}_i}}}$, $P_0 = g_B^{\alpha} g_B^{\beta^2 t}$, $P_1 = (g_B^{\beta ID} h_B)^t$;, and $P_2 = g_B^{-t}$, and sends $(r_{\mathscr{G}_i}, s_{\mathscr{G}_i}, A_{\mathscr{G}_i}) || (P_0, P_1, P_2)$ to V.
5. The visitor V verifies whether $\overline{e}(A, g_1^{s_{\mathscr{G}_i}} y_{DO}) = \overline{e}(g_1 ID h_1^{\mathscr{G}_i} h_2^{r_{\mathscr{G}_i}}, g_1)$ holds. If it is, V makes records $\mathscr{G} \leftarrow \mathscr{G} \cup \mathscr{G}_i$, $P \leftarrow P \cup (P_0, P_1, P_2)$, $Cred \leftarrow Cred \cup (A_{\mathscr{G}_i}, s_{\mathscr{G}_i}, r_{\mathscr{G}_i})$

Figure 2.2: Joining in A Group Protocol

keys for decrypting the data associated with this group.

The visitor's inputs are $(\mathscr{G}_i, st_V, pk_{DO})$, where $st_V = (z_V, ID, count, \mathscr{G}, Cred, P)$ is the visitor's state which consists of the visitor's secret, her ID, a bit *count* showing whether she has visited the storage site or not, all groups she has joined, the credentials and the private

Function: A data owner *DO* uploads data to the storage site *DB*.

1. The data owner *DO* generates *AL* and *RL* for the data *D* and sends (AL, RL) to *DB*.
2. If it is the first time *DB* receives *D*, it sets $index \leftarrow 1$. *DB* parses *AL* as $(\mathscr{G}_1, \cdots, \mathscr{G}_{l'})$, computes $E_{index} \leftarrow g^{\frac{1}{x_e + index + \sum_{j=1}^{l'} x_j^{\mathscr{G}_j}}}$ and $F_{index} \leftarrow e(h, E_{index})$, and sends (E_{index}, F_{index}) to *DO*.
3. The data owner *DO* chooses $s \xleftarrow{\$} \mathbb{Z}_p$. For $i = 1, \cdots, r$, *DO* randomly chooses $s_1, \cdots, s_{r-1}$, such that $s_r = s - s_1 - \cdots - s_{r-1}$. *DO* computes $F_1 = F_{index} e(g_B, g_B)^{\alpha s} D$ and $F_2 = g_B^s$. For each $j = 1, 2, \cdots, r$, *DO* computes $C_{j,1} = g_B^{\beta s_j}$ and $C_{j,2} = (g_B^{\beta^2 ID_j} h_B^{\beta})^{s_j}$, and sends $\{F_1, F_2, (C_{j,1}, C_{j,2})_{j=1}^r\}$ to *DB*.
4. The storage site *DB* sets $index \leftarrow index + 1$, and publishes $L_{index} = (E_{index}, F_1, F_2, (C_{j,1}, C_{j,2})_{j=1}^r, AL, RL)$.

Figure 2.3: Uploading Data Protocol

keys she has acquired. If it is the first time the visitor joins the OSN, her state is set to be $st_V = \perp$. The visitor chooses her secret z_V randomly from $\mathbb{Z}_p$, computes $ID \leftarrow h_0^{z_V}$, sends *ID* to the data owner, and proves to the data owner that she indeed has z_V in a zero-knowledge way. After that, the visitor's state is set to be $(z_V, ID, 0, \emptyset, \emptyset, \emptyset)$. If *ID* is not accepted, the data owner returns $\perp$ to the visitor. Otherwise, the data owner proves to the visitor that she has private keys α and β in a zero-knowledge way; The data owner chooses random $s_{\mathscr{G}_i}, r_{\mathscr{G}_i}, t \xleftarrow{\$} \mathbb{Z}_p$, computes the signature $A \leftarrow (g_1 ID h_1^{\mathscr{G}_i} h_2^{r_{\mathscr{G}_i}})^{\frac{1}{x_{DO} + s_{\mathscr{G}_i}}}$ and the private keys P_0, P_1 and P_2, and then sends them to the visitor. The resulting signature is the credential for the visitor to access the data associated with group $\mathscr{G}_i$. (P_0, P_1, P_2) are private keys for the visitor to decrypt the data. Each time of executing this protocol, the visitor acquires one credential. If he needs more than one credentials to access the data, the protocol needs to be executed several times.

Figure 2.3 presents the uploading data protocol. In this protocol, the data owner uploads the ciphertexts of the data *D* with its *AL* and *RL* to the storage site. The input of the data owner is (D, pk_{DO}, sk_{DO}). The input of the storage site is (AL, pk_{DB}, sk_{DB}). Upon each

executing of the protocol, one piece of data D is uploaded.

First the data owner generates the AL and RL for D, sends (AL, RL) to the storage site. The AL is a list of $(\{\mathscr{G}_i\}_{i=1}^{l'})$, and the RL takes the form of $(\{ID_i\}_{i=1}^{r})$, where $\{ID_i\}_{i=1}^{r} \subseteq \{\mathscr{G}_i\}_{i=1}^{l'}$. $\{\mathscr{G}_{t_i}\}_{i=1}^{l'}$ is a l'-subset of the total l groups $(\mathscr{G}_1, \cdots, \mathscr{G}_l)$, $\{ID_i\}_{i=1}^{r}$ is a r-subset of the total N visitors $(ID_1, \cdots, ID_N)$ on the data owner's friend list. r is the number of revoked visitors. l' is the number of groups the visitor needs to join to access D. On the storage site side, if it was the first time the storage site receives a piece of data, the global *index* is set to 1. The storage site generates the signature E_{index} on AL using the signature scheme introduced in section 2.4.4.1, sends the pair (E_{index}, F_{index}) to the data owner. Note that the AL contains l' groups, where $l' \leq l$. The storage site only need to use the first l' private keys $(x_i)_{i=1}^{l'}$ to sign the AL. On the data owner side, he chooses a random number $s \in \mathbb{Z}_p$, and divides it into r random shares $s_1, s_2, \cdots, s_r$ such that $s_1 + s_2 + \cdots, +s_r = s$. The data D is encrypted as $F_1 = F_{index} e(g_B, g_B)^{\alpha s} D$. Note that before the encryption, we should first map D from 0, 1 bits to an element in group $\mathbb{G}_T$ using collision resistant hash functions. After receiving the encryption of D, the storage site increases *index* by 1 and publishes $L_{index} = (E_{index}, F_1, F_2, (C_{j,1}, C_{j,2})_{j=1}^{r}, AL, RL)$.

Figure 2.4 shows the retrieving data protocol. This protocol enables the visitor to anonymously get the data while the storage site is oblivious to the identity of the visitor and the index of the data.

First the visitor parses her state st_V as $(z_V, P, count, \mathscr{G}, Cred, P)$. Then the visitor checks whether she has credentials to access the data. If $AL \nsubseteq \mathscr{G}$ or $ID \in RL$, then she aborts the protocol. Otherwise, the visitor chooses the index of the data that he wants to access, $index = \sigma$, picks up a random number $k' \in \mathbb{Z}_p$, computes $K \leftarrow (E_{index})^{k'}$, and sends $(K, count)$ to the storage site. Note that K is the randomization of E_σ. The purpose of this randomization is to let the storage site not know which data the visitor has chosen.

Function: A visitor V retrieves data from the storage site DB.

1. The visitor V parses st_V as $(z_V, P, count, \mathscr{G}, Cred, P)$. If $AL \nsubseteq \mathscr{G}$ or $ID \in RL$, V aborts the protocol; otherwise V chooses $k' \xleftarrow{\$} \mathbb{Z}_p$, computes $K = (E_\sigma)^{k'}$, and sends $(K, count)$ to DB.
2. If $count = 0$ holds, DB does a zero knowledge proof of $PK\{(h) : M = e(g,h)\}$ to V.
3. The visitor V sets $count \leftarrow 1$, parses AL as $\{\mathscr{G}_1, \cdots, \mathscr{G}_{l'}\}$, and chooses $u \xleftarrow{\$} \tilde{\mathbb{G}}$. For $i = 1, \cdots, l'$, V chooses $t_{1_i}, t_{2_i}, t_{3_i}, t_{4_i}, t_{5_i}, t_{6_i} \xleftarrow{\$} \mathbb{Z}_p$, $\gamma_i, \delta_i, \eta_i, \rho_i, \phi_i, \psi_i \xleftarrow{\$} \mathbb{Z}_p$, computes $\tilde{A}_i \leftarrow A_{\mathscr{G}_i} u^{\gamma_i}$, $\tilde{g}_{1_i} \leftarrow g_1 u^{\delta_i}$, $\tilde{y}_i \leftarrow y_{DO} u^{\eta_i}$; $\tilde{h}_{0_i} = h_0 u^{\rho_i}$, $\tilde{h}_{1_i} = h_1 u^{\phi_i}$, and $\tilde{h}_{2_i} = h_2 u^{\psi_i}$, and sends $(\tilde{A}_i, \tilde{g}_{1_i}, \tilde{y}_i, \tilde{h}_{0_i}, \tilde{h}_{1_i}, \tilde{h}_{2_i})_{i=1,\cdots,l'}$ with a zero knowledge proof ZKP to DB.
4. The storage site DB computes $R = e(h,K)$, and sends R together with a zero knowledge proof of $PK\{(h) : M = e(g,h) \bigwedge R = e(h,K)\}$ to V.
5. The visitor V computes $D = \frac{F_{1,\sigma} \bar{e}(P_1, \prod_{i=1}^{r} C_{i,1}^{\frac{1}{ID - ID_i}}) \bar{e}(P_2, \prod_{i=1}^{r} C_{i,2}^{\frac{1}{ID - ID_i}})}{R^{\frac{1}{k'}} \bar{e}(F_{2,\sigma}, P_0)}$

Figure 2.4: Retrieving Data Protocol

If it's the first time the visitor accesses the storage site, the storage site should prove to the visitor that it has the secret key h. This is realized by a zero-knowledge proof of $PK\{(h) : M = e(g,h)\}$. Then the visitor updates $count \leftarrow 1$, executes a zero-knowledge proof to prove that K is indeed the randomization of E_σ, that he has the secret key z_V as he claimed and has valid credentials to access the data. ZKP in Figure 2.4 denotes $PK\{(\sigma, k', z_V, (\mathscr{G}_i, s_{\mathscr{G}_i}, r_{\mathscr{G}_i}, \gamma_i, \delta_i, \eta_i, \rho_i, \phi_i, \psi_i)_{i=1,\cdots,l'}) : e(K, y_{DB}) e(K,g)^\sigma \prod_{i=1}^{l} e(K, y_i)^{\mathscr{G}_i}$ $= e(g,g)^{k'} \bigwedge_{i=1}^{l'} Proof(z_V, \mathscr{G}_i, s_{\mathscr{G}_i}, r_{\mathscr{G}_i}, \tilde{A}_i, \tilde{g}_{1_i}, \tilde{y}_i, \tilde{h}_{0_i}, \tilde{h}_{1_i}, \tilde{h}_{2_i})\}$. After the proof is verified, the storage site computes $R \leftarrow e(h,K)$ and returns it to the visitor. Using R and the private keys (P_0, P_1, P_2), the visitor can decrypt the ciphertext.

2.6 Security Analysis

In this section, we prove the security of our system using UC framework introduced in section 2.2.3. The indistinguishability of the real world and ideal world is proved by defining

a sequence of hybrid games Game-0, $\cdots$, Game-n. In each proof, for an adversary $\mathscr{A}$ in the real world, we construct a corresponding simulator *Sim* in the ideal world. $\mathscr{A}$ runs as a subroutine in *Sim*, which emulates the honest parties interacting with $\mathscr{A}$ and provides the entire view of $\mathscr{A}$ to ξ. We denote Game-0 equivalent to the environment of the real world, and Game-n equivalent to the environment of the ideal world. We define that $Real_{\xi,\mathscr{A}}(k)$ is the probability that ξ output 1 given the view of the adversary and outputs of the honest parties in the real world, $Ideal_{\xi,Sim}(k)$ is the probability that ξ output 1 given the view of *Sim* and outputs of the honest parties in the ideal world, and $Hybrid_{\xi,Sim_i}(k)$ is the probability that ξ outputs 1 given the view of Sim_i and outputs of the honest parties in Game-i.

We start from Game-0, then change some parameter in each game, and bound the difference between two adjacent games. Summing up all the differences, we can get an upper bound of differences between the real world and the ideal world according to union bound. We prove that this upper bound is a negligible function in k. Finally, we come to the conclusion that it is indistinguishable between the real world and the ideal world.

Theorem 1. *Provided that the assumptions q-MEBDH, q-SDH, and q-BDHE hold, our proposed system securely realizes the functionalities in Section 2.2.3. In other words, for an environment ξ and any adversary $\mathscr{A}$, there exists an ideal world simulator Sim such that*

$$Real_{\xi,A}(k) - Ideal_{\xi,Sim}(k) \leq \varepsilon(k),$$

where $\varepsilon(k)$ is a negligible function in k. □

We prove Theorem 1 by proving the following lemmas.

Lemma 2.6.1. *For environment ξ and any real world adversary $\mathscr{A}$ controlling part of the data owners and the storage site, there exists an ideal world simulator Sim such that*

$$Real_{\xi,A}(k) - Ideal_{\xi,Sim}(k) \leq 2^{-k}$$

□

Proof. In this proof, we assume there are multiple of data owners and visitors, one storage site.

Game-1: Simulator Sim_1, in the joining in a group protocol, runs the extractor for the proof of knowledge $PK\{(\alpha,\beta) : X = g_2^{\alpha} \wedge Y = g_2^{\beta}\}$ to extract the secret α and β such that $g_2^{\alpha} = X$ and $g_2^{\beta} = Y$. If the extractor fails, it just outputs $\perp$ to ξ; otherwise, it continues to run interacting with the adversary on behalf of the honest visitor. The difference between *Game-0* and *Game*-1 is the knowledge error of the proof of knowledge, which is

$$Real_{\xi,A}(k) - Hybrid_{\xi,Sim_1}(k) \leq 2^{-k}$$

Game-2: Simulator Sim_2 runs exactly like that in *Game-1*, except that in the retrieving data protocol, runs the extractor for the proof of knowledge $PK\{(h) : PK\{(h) : M = e(g,h)\}\}$ to extract the secrete h such that $e(g,h) = M$. If the extractor fails, it just output $\perp$ to ξ; otherwise, it continues to run interacting with the adversary on behalf of the honest visitor. The difference between *Game-1* and *Game-2* is the knowledge error of the proof of knowledge, or,

$$Hybrid_{\xi,Sim_1}(k) - Hybrid_{\xi,Sim_2}(k) \leq 2^{-k}$$

Game-3: Simulator Sim_3 runs exactly like the above game, except that during the retrieving data protocol, instead for querying D_{σ}, where σ is the *index* of D, it uses randomly chosen

data D_j which it has necessary credentials and runs a simulated proof for $PK\{\sigma, k', z_V, \cdots\}$. We state that

$$Hybrid_{\xi, Sim_2}(k) = Hybrid_{\xi, Sim_3}(k)$$

The statement follows a perfect zero knowledge proof of $PK\{(\sigma, k', z_V, \cdots)\}$. Now we construct based on the real world adversary $\mathscr{A}$, a simulator *Sim* in the ideal world, and combines all the steps from the above games. *Sim* runs $\mathscr{A}$ to get the public key of the data owner and the storage site, pk_{DO} and pk_{DB}, as well as the encrypted data item L_{index}. Upon receiving $(Join, ID, \mathscr{G}_i, DO)$ from T, *Sim* runs the knowledge extractor to exact α, β from $\mathscr{A}$ and computes $X = g_B^{\alpha}$, $Y = g_B^{\beta}$. From $X = g_B^{\alpha}$, *Sim* can compute $\mathscr{K} = e(g_B^{\alpha}, g_B^{s}) = e(g_B, g_B)^{\alpha s}$. After that, *Sim* executes the visitor side algorithm and maintains necessary state. If the resulting credential is valid, *Sim* returns $b = 1$ to T, otherwise it returns $\perp$. The first time *Sim* receives $(Retrieve, \sigma)$ from T, it runs knowledge extractor to extract h from $\mathscr{A}$ and computes $F_{index} = e(h, E_{index})$. *Sim* then uses $\mathscr{K}$ and F_{index} to decrypt the records. It then uses D_j chosen among those she has necessary credentials to query the storage site. If the transfer succeeds, *Sim* sends $b = 1$ to T, otherwise it sends $\perp$ to T. The following execution of protocols are just like the above, except the removing of knowledge extractor in retrieving data protocol.

We can see that *Sim* provides exactly the same environment to $\mathscr{A}$ as Sim_3 does. We have

$$Ideal_{\xi, Sim}(k) = Hybrid_{\xi, Sim_3}(k)$$

Summing up all the statements above, we obtain lemma 2.6.1. □

Lemma 2.6.2. *For the environment ξ and any real world adversary $\mathscr{A}$ controlling the storage site, there exists an ideal world simulator Sim such that*

$$Real_{\xi,A}(k) - Ideal_{\xi,S}(k) \leq Adv^{q_V-MEBDH}_{\mathbb{G},\mathbb{G}_T}(k)$$

where q_V is the maximum number of revoked visitors. □

Proof. For proof simplicity, we assume there is one data owner, one storage site and one visitor.

Game-1: Upon receiving $(Upload, D, ACL, RL)$ from T, simulator Sim_1 executes the data owner side algorithm. Instead of computing $e(g_B, g_B)^{\alpha s}D$, the simulator uses numbers D' chosen randomly from $\mathbb{G}_T$ as a substitution. The adversary can not distinguish $e(g_B, g_B)^{\alpha s}D$ from D'. Unless he has an algorithm to solve instances of q_V-MEBDH problem in polynomial time with non-negligible probability. We state

$$Real_{\xi,A}(k) - Hybrid_{\xi,Sim_1}(k) \leq Adv^{q_V\text{-MEBDH}}_{\mathbb{G},\mathbb{G}_\mathbb{T}}$$

Game-2: Simulator Sim_2 runs exactly like Sim_1, except that during the retrieving data protocol, Sim_2 executes the visitor side algorithm. It lets the visitor algorithm query the random number D' encrypted during *Game-1* instead of querying the real data D. Then it uses the simulated proof for a zero-knowledge proof of $PK\{(\sigma, k', z_V, \cdots)\}$. Therefore,

$$Hybrid_{\xi,Sim_1}(k) = Hybrid_{\xi,Sim_2}(k)$$

Now based on the real world adversary $\mathscr{A}$, we construct a simulator *Sim* in the ideal world. $\mathscr{A}$ runs as a subroutine in *Sim*. *Sim* plays the role of the storage site, and combines all the steps from the above games. *Sim* runs $\mathscr{A}$ to get the storage site public key pk_{DB}. Upon receiving $(Upload, D, ACL, RL)$ from T, *Sim* executes the data owner side algorithm with $\mathscr{A}$, it chooses numbers D' randomly from $\mathbb{G}_T$ and encrypts it using F_{index} returned by $\mathscr{A}$,

then uploads the encrypted data, *ACL* and *RL* to $\mathscr{A}$. Upon receiving $(Retrieve, \sigma)$ from T, *Sim* executes the visitor side algorithm and queries the random number D'. Then it runs a simulated proof for a zero-knowledge proof $PK\{(\sigma, k, z_V, \cdots)\}$.

We can see that *Sim* provides exactly the same environment to $\mathscr{A}$ as Sim_2 does. We have

$$Ideal_{\xi,Sim}(k) = Hybrid_{\xi,Sim_2}(k)$$

Summing up all the above statements, we get lemma 2.6.2. □

Lemma 2.6.3. *For the environment ξ and any real world adversary A controlling part of the visitors, there exists an ideal world simulator Sim such that*

$$Real_{\xi,A}(k) - Ideal_{\xi,Sim}(k) \leq 2^{-k} q_{tri} + Adv^{q_{cred}\text{-}SDH}_{\mathbb{G},\mathbb{G}_T}(k)$$
$$+ Adv^{(N+1)SDH}_{\mathbb{G},\mathbb{G}_T}(k) + (N+1) Adv^{(N+2)BDHE}_{\mathbb{G},\mathbb{G}_T}(k)$$

where q_{tri} is the total number of retrieve queries, q_V is the number of credential queries, N is the total number of data items in the storage site. □

Proof. In this proof, we assume there are multiple of visitors, one data owner and one storage site.

Game-1: In uploading data protocol, simulator Sim_1 executes the storage site side algorithm and instead of publishing encrypted data F_i, it publishes F_i' chosen randomly from $\mathbb{G}_T$. $\mathscr{A}$ can not distinguish these two numbers. Unless it has a polynomial time algorithm to solve instances of BDHE problem. The difference between *Game-0* and *Game-1* is bounded by $Adv^{(N+2)\text{BDHE}}_{\mathbb{G},\mathbb{G}_T}(k)$ [24]. We have

$$Real_{\xi,A}(k) - Hybrid_{\xi,Sim_1}(k) \leq (N+1) Adv^{(N+2)\text{BDHE}}_{\mathbb{G},\mathbb{G}_T}(k)$$

Game-2: Simulator Sim_2 runs exactly like Sim_1, except that during retrieving data protocol, Sim_3 uses the knowledge extractor to exact $(\sigma, k', z_V, \cdots)$ from the zero knowledge proof of knowledge. If the extractor fails, it output $\perp$ to ξ. If it succeed, it continues to run honest storage site algorithm with visitors. The difference between *Game-1* and *Game-2* is the knowledge error of knowledge proof. We have that

$$Hybrid_{\xi,Sim_2}(k) - Hybrid_{\xi,Sim_1}(k) \leq q_{tri} 2^{-k}$$

where q_{tri} is the number of zero-knowledge proofs.

Game-3: Simulator Sim_3 runs exactly like Sim_2, except that during retrieving data protocol, when at least one of the extracted A_i was not issued by the data owner to the visitor with the secret key z_V, Sim_4 output $\perp$ to ξ. This means the visitor was trying to forge a credential signature. Note that this case also include corrupted visitors pooling their credentials. The difference between *Game-3* and *Game-2* is given by the security of the signature scheme, which is

$$Hybrid_{\xi,Sim_3}(k) - Hybrid_{\xi,Sim_2}(k) \leq Adv^{q_{cred}\text{-SDH}}_{\mathbb{G},\mathbb{G}_T}(k)$$

where q_{cred} is the number of credential queries the adversary made.

Game-4: Simulator Sim_4 runs exactly like Sim_3, except that during the retrieving data protocol, when the extracted $\sigma \notin (1, \cdots, N)$ or $\mathscr{G}_i$ does not match ACL_σ during any retrieve, then Sim_4 output $\perp$ to ξ. In this case, we can say that $K^{1/k'}$ is a forged signature on D_σ. The difference between *Game-4* and *Game-3* is bounded by $Adv^{(N+1)-SDH}_{\mathbb{G},\mathbb{G}_T}(k)$ [24]. We have

$$Hybrid_{\xi,Sim_4}(k) - Hybrid_{\xi,Sim_3}(k) \leq Adv^{(N+1)SDH}_{\mathbb{G},\mathbb{G}_T}(k)$$

Game-5: Simulator Sim_5 runs exactly like Sim_4, except that during the retrieving data protocol, the value R is computed as $R \leftarrow \{\frac{F_{1,\sigma}\overline{e}(P_1,\prod_{i=1}^{r} C_{i,1}^{\frac{1}{ID-ID_i}})\overline{e}(P_2,\prod_{i=1}^{r} C_{i,2}^{\frac{1}{ID-ID_i}})}{R\overline{e}(F_{2,\sigma},P_0)}\}^{k'}$. And then the simulator runs a simulated proof of h. We have

$$Hybrid_{\xi,Sim_5}(k) = Hybrid_{\xi,Sim_4}(k)$$

We now construct based on the real world adversary $\mathscr{A}$, simulator *Sim* in ideal world. $\mathscr{A}$ runs as a subroutine in *Sim*. *Sim* plays the role of corrupted visitors, and executes all the steps in the above games. In the *Retrieve* stage, after *Sim* exacts σ from $\mathscr{A}$, it request record R_σ from T. If A does not has the authorization to access R_σ, T will return $\perp$ to *Sim* and *Sim* simply returns it to ξ, otherwise T returns R_σ to *Sim*, *Sim* computes $R = \{\frac{F_{1,\sigma}\overline{e}(P_1,\prod_{i=1}^{r} C_{i,1}^{\frac{1}{ID-ID_i}})\overline{e}(P_2,\prod_{i=1}^{r} C_{i,2}^{\frac{1}{ID-ID_i}})}{R\overline{e}(F_{2,\sigma},P_0)}\}^{k'}$ and runs a simulated proof of h. We can see that *Sim* provides an environment exactly the same as Sim_5 does. Summing up all the above statements, we can prove Lemma 2.6.3. □

Lemma 2.6.4. *For the environment ξ and any real world adversary A controlling part of the visitors and the data owners, there exists an ideal world simulator Sim such that*

$$Real_{\xi,A}(k) - Ideal_{\xi,Sim}(k) \leq 2^{-k} q_{tri}$$
$$+ Adv_{\mathbb{G},\mathbb{G}_T}^{(N+1)SDH}(k) + (N+1)Adv_{\mathbb{G},\mathbb{G}_T}^{(N+2)BDHE}(k)$$

□

Proof. In the proof, we assume there are multiple of visitors and data owners, and one storage site.

Game-1: In uploading data protocol, simulator Sim_1 executes the storage site side algorithm and instead of publishing encrypted data F_i, it publishes F_i' chosen randomly from $\mathbb{G}_T$. $\mathscr{A}$ can not distinguish these two numbers. Unless it has a polynomial time algorithm to solve instances of BDHE problem. The difference between *Game-0* and *Game-1* is bounded by $Adv_{\mathbb{G},\mathbb{G}_T}^{(N+2)\text{BDHE}}(k)$ [24]. We have

$$Real_{\xi,A}(k) - Hybrid_{\xi,Sim_1}(k) \leq (N+1)Adv_{\mathbb{G},\mathbb{G}_T}^{(N+2)\text{BDHE}}(k)$$

Game-2: Simulator Sim_2 runs exactly like Sim_1, except that during retrieving data protocol, Sim_2 uses the knowledge extractor to extract $(\sigma, k', z_V, \cdots)$ from the zero knowledge proof of knowledge. If the extractor fails, it output $\perp$ to ξ. If it succeed, it continues to run honest storage site algorithm with the visitors. The difference between *Game-1* and *Game-2* is the knowledge error of knowledge proof. We have that

$$Hybrid_{\xi,Sim_2}(k) - Hybrid_{\xi,Sim_1}(k) \leq q_{tri}2^{-k}$$

where q_{tri} is the number of zero-knowledge proofs.

Game-3: Simulator Sim_3 runs exactly like Sim_2, except that during the retrieving data protocol, when the extracted $\sigma \notin (1,\cdots,N)$ or $\mathscr{G}_i$ does not match ACL_σ during any retrieve, then Sim_3 output $\perp$ to ξ. In this case, we can say that $K^{1/k'}$ is a forged signature on R_σ. The difference between *Game-3* and *Game-2* is bounded by $Adv_{\mathbb{G},\mathbb{G}_T}^{(N+1)\text{-SDH}}(k)$ [24]. We have

$$Hybrid_{\xi,Sim_4}(k) - Hybrid_{\xi,Sim_3}(k) \leq Adv_{\mathbb{G},\mathbb{G}_T}^{(N+1)\text{SDH}}(k)$$

Game-4: Simulator Sim_4 runs exactly like Sim_3, except that during the retrieving data protocol, the value D is computed as $D \leftarrow \{\frac{F_{1,\sigma}\bar{e}(P_1,\prod_{i=1}^{r} C_{i,1}^{\frac{1}{ID-ID_i}})\bar{e}(P_2,\prod_{i=1}^{r} C_{i,2}^{\frac{1}{ID-ID_i}})}{R\bar{e}(F_{2,\sigma},P_0)}\}^{k'}$. And then the simulator runs a simulated proof of h. We state

$$Hybrid_{\xi,Sim_5}(k) = Hybrid_{\xi,Sim_4}(k)$$

Summing up all the statement, we can prove lemma 2.6.4. □

2.7 Performance Evaluation

We implemented our protocols on a laptop with dual 2.40GHz CPUs and 2G memory. In order to generate pairing groups and bilinear map, our implementation used a C++ wrapper [66] of Pairing-Based Cryptography (PBC) library, developed by Lynn [77]. PBC is a well-known C library, efficiently implementing elliptic curve generation, elliptic curve arithmetic and pairing computation. Our evaluation exactly followed the description of the protocols.

Since our protocols mainly consist of broadcast encryption, ACL signature, credential signature and zero-knowledge proofs, the purpose of our experiments was to test the performance of these four primitives. The underlying elliptical curve used in our implementation is $y = x^3 + x$ with an embedding degree of 2. We chose the group size k, equalling to $32, 64, 128, 256$ and 512 bits respectively, and tested the running time needed for each primitive. Figure 2.5 shows the running time of the four primitives. In each figure, the lines with different marker styles represent the running time under different group sizes.

Figure 2.5a is the running time of broadcast encryption algorithm in uploading data protocol. The X-coordinate corresponds to r in our protocols, representing the number of revoked visitors. In our implementation, this number is from 16 to 256 with increment of 16. From the figure, we can see that if the number of the revoked visitors is 50 ($x = 50$), the running time of the algorithm with variant group size is bellow 5 seconds. This is fast for such a large number, since in real life the data owner usually does not add so many people

in a revocation list.

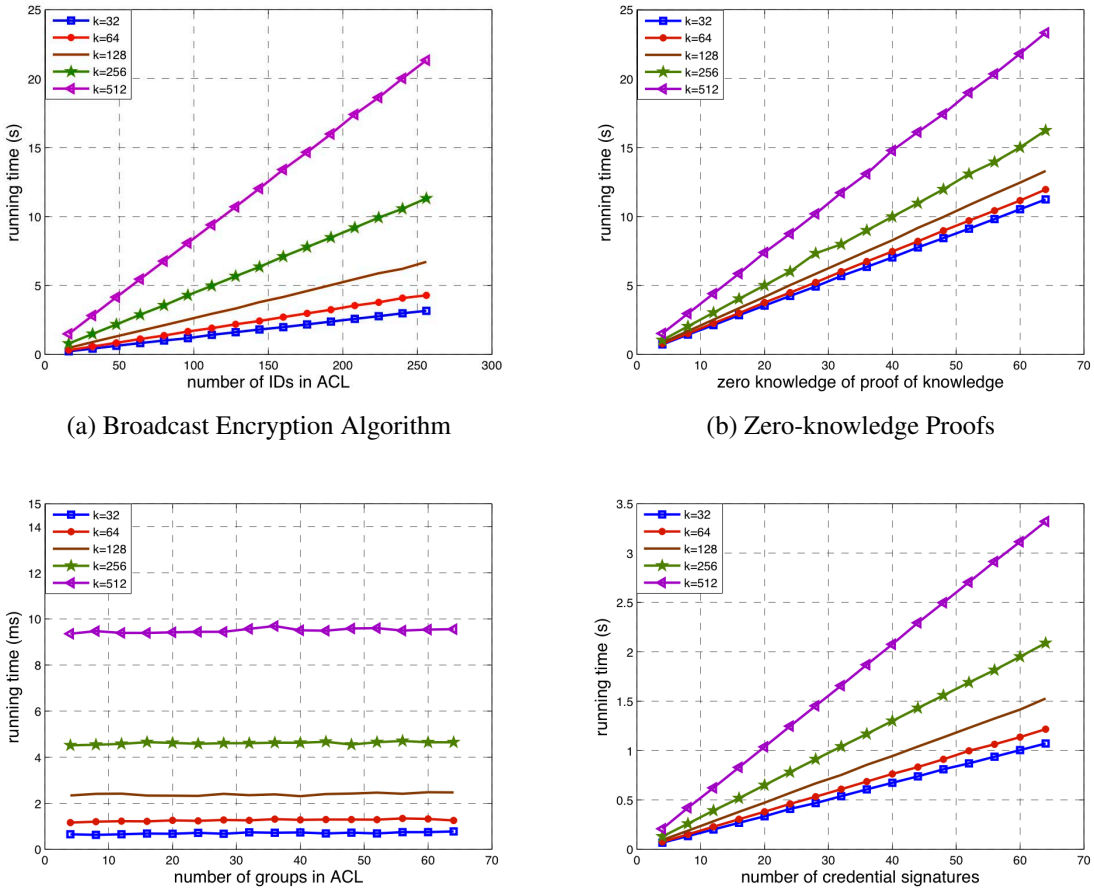
Figure 2.5b shows the running time of zero-knowledge proofs for proving the knowledge of credentials in data retrieving protocol. The X-coordinate corresponds to l' in our protocol, representing the number of credentials a visitor needs to prove in order to access the data. The range is from 4 to 64 with increment of 4. From the figure we can see that when $k = 512$, it takes about 15 seconds to prove 40 credentials.

Figure 2.5c shows the running time of ACL signature in uploading data protocol. From the figure we can see that when the value k keeps constant, the running time does not change so much under different group size. When the number of groups increases by 1 in ACL, extra computation is cost on computing a large E_{index} with more groups. However, the computation time of multiplication is much less than those of the pairing operations and the computation of discrete logarithm module a prime. This extra computation does not give a remarkable increment in the running time.

Figure 2.5d shows the running time of credential signatures. The X-coordinate represents the number of credentials the data owner needs to sign one time for accessing the data. In our implementation, the range is from 4 to 64 with increment of 4. From this figure, we can see that when $k = 512$, it takes less than 3.5 seconds to sign 64 credentials at one time. Since one credential corresponds to a group in our scenario, 64 groups is more than a user's common use in the OSN.

2.8 Conclusions

In this dissertation, we have proposed a system which can hide visitors' identities and the index of the data when they visit the storage site. Using zero-knowledge proof based oblivious transfer, the storage site can verify visitors' credentials without leaking the private



(a) Broadcast Encryption Algorithm

(b) Zero-knowledge Proofs

(c) ACL Signature

(d) Credential Signature

Figure 2.5: Running Time

information of the signer. We have designed a group-based ACL for the data owner to freely determine which group of friends can access the data. In addition, we have designed an RL which enables the data owner to revoke individuals from future accessing. In this way, our access control policy can be generalized to individual visitors. We have proved the security of our system by indistinguishability of the real world and the ideal world in UC framework. Our evaluation results have shown the efficiency of our system.

CHAPTER 3

Identity Privacy in Anonymous Message Submission

In the real world, anonymity has always been an important societal issue. As more and more people discover the digital world and find the need for anonymous participation, this issue is becoming increasingly important. In addition, the new wave of online social networks has created an unprecedented demand for anonymity in the digital world.

Consider a health care product company that wants to do an online survey about its products in an online social network, such as Facebook. The company wants to receive accurate feedback from its users and avoid redundant responses from the same person. Hence, the company would like the participants to log in their social network accounts. This can be easily achieved by having, for example, a "Login with Facebook" button on the survey web page. In this way, the company could get the participants' demographic data and reject repeated submissions. However, the participants may not feel good about this approach, which links the answers to their social identities. Particularly, people would rather not participate in the survey when the answers contain any private information, such as their health conditions. Another example is that some online career social networks, such as LinkedIn, often conduct surveys like "how do you like your current company". The result of this kind of survey is very helpful to other people's career development. However, people would not like to answer or may not give their truthful thinking if they are worried about that their negative answers may link to their real identities and backfire on themselves.

To release the worries mentioned in the above, it is necessary to construct an anonymous message submission mechanism that hides the connection between a submitted message

and its provider's identity. Before we get to the details of our work, it is helpful to study how people do anonymous message submissions in real life. Consider secret ballot which is an example of "anonymous message submission system". Participants cast their ballots into a locked and non-transparent ballot box. The box is then shaken to randomize the order in which people cast their ballots. At last, the ballot papers are taken out and handed to the collector. Aside from hand writing differences, all ballot papers are similar, or indistinguishable, to one another. Assuming the collector cannot tell people's hand writing differences, he would not know the connections between people's identities and their ballots. The submission order connects a ballot to its participant but has been broken by the shaking process. However, it is challenging to migrate this simple real-life system to a public and untrusted network, because a secure ballot box trusted by all participants is difficult to find in the digital world. Besides, the shaking process is supervised simultaneously by all participants and difficult to reproduce when participants are connected by separate connections (e.g., TCP).

One may suggest to use existing anonymous networks, such as Onion routing [30, 94], to deliver a participant's message to the collector. This solution does not fit our scenario, which needs to verify participants' identities before message submission happens. The only thing to be protected in our scenario is the connection between a message and the participant's identity. However, anonymous networks also hide participants' identities. In other words, the collector does not know who participates in the message submission.

Anonymous message submission has attracted wide attentions [21, 38, 101]. In [101], Yang *et al.* proposed the first construction for online anonymous message submission. Bickell and Shmatikov revisited this problem and proposed a collusion resistant anonymous data collection protocol in [21]. Recently, Corrigan-Gibbs and Ford improved the protocol in [21] to make it accountable and added the functionality of sending variable length messages

[38]. Anonymity in the above protocols is guaranteed by a "shuffle" process, which is collaboratively done by all group members. The existing solutions [21] [38] are based on two rounds of IND-CCA2 secure encryption that are done by group members one by one. This makes the protocol's communication rounds linear in the group size. Their solutions can only handle groups of small sizes. For example, the system in [38] can handle up to 44 members.

The motivation of this work is to construct an efficient and collusion resistant anonymous message submission protocol for a group of a practical scale (e.g., hundreds of group members). We propose a novel technique which secretly aggregates the group members' messages into a message vector with each message being a component in the vector. Each member only knows his selected component index and is unaware of other members' choices. Our work aims to promote the practical application of message submission with strong anonymity guarantees, which will increase privacy protections and anonymity in the digital world.

Our main contributions are as follows:

1. We propose a novel anonymous position application technique, which enables each group member to obtain a position in the final message submission sequence in a manner that each member is oblivious to other members' positions. Technically, we use a vector to represent a sequence, and a position in the sequence is mapped to the corresponding component index in the vector. This process can be done in a few communication rounds with a high success probability (2 rounds with 95% success probability).

2. We propose an efficient anonymous message submission protocol, AMS, for groups of practical scales. Our protocol runs in time bounded by a polynomial in the group

size, and has constant communication rounds.

3. We prove that the AMS protocol is anonymous and collusion resistant under malicious attacks. The AMS protocol does not rely on any trusted third party. The security of our protocol is based on the security of the secret sharing scheme and the symmetric key cryptosystem.

4. We analyze the performance of our protocol from the aspects of security and efficiency. We conduct comprehensive simulations showing that our protocol is more efficient than existing ones , especially when the group size is large.

To achieve accountability and handle variable length messages, we adopt the signature based audit protocol and the DC-net based message transmission protocol proposed in [38].

3.1 Related Work

In this section, we review the closely related work on anonymous message submission and secure multi-party computation (SMC).

The anonymous message submission problem was first introduced by Yang *et al.* [101], who proposed a shuffle technique to solve it. In their protocol, t leaders are chosen out of the N group members. Each member's message is encrypted with all leaders' public keys. Each leader randomly shuffles the messages and proves that the shuffle is correct using zero-knowledge proof. However, if the last leader colludes with the collector, the anonymity will be violated. In [21], Brickell and Shmatikov proposed a collusion resistant anonymous data collection protocol. In their protocol, each member generates his primary and secondary public/private key pairs. The users first encrypt their messages using the collector's public key, then encrypt the resulting ciphertexts using each member's secondary public key, and finally encrypt the resulting ciphertexts using each member's primary pub-

lic key. Next, the ciphertexts are randomly shuffled by each member who also strips off one layer of the encryption during the shuffle. Finally, the ciphertexts are sent to the collector in a random order. The collector decrypts the ciphertexts using the secondary private keys sent by the members and his own private key. In [38], Corrigan-Gibbs and Ford extended the shuffle protocol and proposed an accountable anonymous message submission protocol, called Dissent. Dissent consists of two protocols: shuffle and bulk. They still use the shuffle technique in [21] in their shuffle protocol. In addition, each member in the protocol of [38] creates a log file and updates its state during the protocol execution. If at some point the protocol terminates abnormally, all members execute the protocol again according to their log files so that they can expose the member who is responsible for the abnormality. The bulk protocol enables the members to send variable length messages. It requires $2N+7$ communication rounds and $O(N^2)$ total computations.

Other anonymous data submission schemes, such as Mix-networks [30] [94] and DC-nets [31] [57] [88] [96], also achieve strong anonymity. However, they seem to be a poor match for our scenario, since the collector may need to know who the group members are. In this case, complicated techniques are needed to enable a well-defined group to submit their messages to the collector, and one or more nodes in the network may be compromised, which breaks the anonymity. Our protocol can provide strong anonymity even when k $(k \leq N-2)$ out of the N group members are compromised.

SMC was first proposed by Yao [102] and focused on a limited set of problems [46], such as privacy-preserving database query [100] [28], privacy-preserving geometric computation [80] and privacy-preserving data mining [3] [73]. While these problems are interesting, they are different from what we focus on in this dissertation. In SMC, there are several parties, and each of them has an input. They collaborate to compute the function on their inputs without revealing their inputs to the other parties. While in our problem, the group

members do want the collector to learn their inputs. While secret sharing scheme is extensively used in SMC [81] [46] [72], to our best knowledge, we are the first to use secret sharing scheme in anonymous message submission.

To make strong anonymous communications scalable, Corrigan-Gibbs *et al.* recently proposed a client/server architecture in [98] and then developed it in [39]. In their architecture, clients are divided into several groups and each group is assigned to a server, which is called the group's "upstream" server. Every client only communicates with its upstream server during the entire communication session. An upstream server collects message ciphertexts from clients and communicates with other servers as its downstream clients' agent. Servers then collaboratively put messages together and hand down to their downstream clients. The system's anonymity relies on DC-nets and the assumption that at least one upstream server is trusted. Different from their approaches, the AMS protocol proposed in this dissertation is a peer-to-peer protocol and does not assume the existence of a trusted third party.

3.2 Problem Formulation

In this section, we present the network model, the threat model, and the security objectives of our work.

3.2.1 Network Model

Our network consists of $N+1$ parties: a set $\mathscr{M}$ consisting of N group members, and a collector C who collects all group members' messages. Each group member has a unique group ID. Without loss of generality, we assume that an ID is a number from $\{1,2,\cdots,N\}$. The group members want to submit their messages to the collector without exposing their identities. The initiator of the protocol could be either the collector or any group member.

We assume that there are at least two honest group members, since it is impossible to ensure anonymity if there is only one honest member.

We **do not** assume the existence of a trusted third party during our protocol execution. Our protocol runs in a completely distributed manner. All the communications are carried out on authenticated and credential channels, i.e., secure channels. We assume that each member has a public/private key pair and the public key is public. The secure channels can be set up by using a public key cryptosystem. We also assume that all members keep connecting to the network during the protocol execution.

3.2.2 Threat Model

There are two types of adversaries: *semi-honest* adversaries and *malicious* adversaries [51] [56]. Semi-honest adversaries honestly follow the protocol execution but are curious about people's private information. They will do their best to collect all messages that they can obtain, analyze them, and infer private information. Malicious adversaries do not necessarily follow the protocol and may eavesdrop the communications, modify, replay, or inject messages. Adversaries could be multiple parties or a single party (e.g., the collector). If multiple parties collude, we consider they are controlled by one adversary so that we only need to consider a single adversary hereafter.

In this dissertation, we consider the security of our protocol in the presence of a malicious adversary. Since a party can always abort from a protocol execution, we do not claim that our protocol can successfully deliver messages under any circumstance, which is the same as in [21] and [38]. Instead, we prove that the security properties are always preserved when the protocol terminates.

3.2.3 Security Objectives

The security objectives of our work are stated as follows.

- **Anonymity:** If k $(k \leq N-2)$ out of the N $(N \geq 3)$ group members collude with the collector, they cannot infer the provider's identity for a given message. Note that anonymity cannot be guaranteed when there is only one honest group member. Having all $N-1$ messages and identities, the adversary can easily link the remaining message to the honest member.
- **Integrity:** When the protocol terminates, the collector should either receive the honest members' messages or be notified that the messages are modified. In the latter case, the culprit should be exposed.
- **Confidentiality:** If the collector is honest, k $(k \leq N-1)$ dishonest members cannot learn the content of the honest members' messages, even by colluding.
- **Accountability:** At least one malicious member should finally be exposed by the group members if the protocol execution is broken.

We use an *anonymization game* [21] to formalize our notion of anonymity for the anonymous message submission protocol. The anonymization game is played between an adversary and an oracle with a security parameter 1^{λ}, where λ is an integer. Suppose that k out of the N group members and the collector are dishonest. The adversary plays the roles of the collector and the k dishonest members, while the oracle plays the roles of the honest members. A protocol is said to be *anonymous* if the adversary can win the game with only a negligible probability. Prior to the game, the adversary chooses $N-k$ messages and gives them to the oracle, who then participates in the anonymous message submission

protocol and submits these messages to the collector on behalf of the honest members. The adversary may repeat this process for a polynomial number of times. Formally, the anonymization game is defined as follows.

1. The adversary chooses two honest members α and β, and two messages d_0 and d_1 in plaintext. He also assigns a message d_i, in plaintext, to each remaining honest member i. The adversary gives these messages to the oracle.

2. The oracle selects a bit $b \in \{0,1\}$ uniformly at random, and sets $d_\alpha = d_b$ and $d_\beta = d_{\bar{b}}$, where $\bar{b}$ denotes the negation of b.

3. The oracle participates in the anonymous message submission protocol. When a message is needed for an honest member i, the oracle responds with the message d_i.

4. After observing the protocol execution, the adversary outputs his guess about b.

Let $\mathscr{D}$ be a probabilistic polynomial time adversary. Then

$$Pr[\mathscr{D}(1^\lambda, \alpha, \beta, d_0, d_1, 0) = 1]$$

is the probability that $\mathscr{D}$ outputs 1 when $b = 0$, and

$$Pr[\mathscr{D}(1^\lambda, \alpha, \beta, d_0, d_1, 1) = 1]$$

is the probability that $\mathscr{D}$ outputs 1 when $b = 1$. The adversary's advantage is

$$\begin{aligned} Adv_{\mathscr{D}} = & Pr[\mathscr{D}(1^\lambda, \alpha, \beta, d_0, d_1, 1) = 1] \\ & - Pr[\mathscr{D}(1^\lambda, \alpha, \beta, d_0, d_1, 0) = 1]. \end{aligned}$$

Definition 2. *A message submission protocol is anonymous if, for any probabilistic polynomial time adversary $\mathscr{D}$, its advantage in the anonymization game is a negligible function in λ.* □

Again, we remark that this definition is valid only when there are at least two honest group members, i.e. $k \leq N-2$.

3.3 Secret Sharing Schemes

A (t,N)-secret sharing ((t,N)-SS) scheme is an efficient scheme that shares a secret among N parties. As stated in [86], the scheme divides the secret into N shares and gives the i-th share $[s]_i^{(t,N)}$ to the i-th party. At least t parties are required to reconstruct s. The notation $[s]_i^{(t,N)}$ denotes the share specifically for the i-th party.

When $t = N$, there is a simplified (N,N)-SS scheme which can achieve linear time complexity [92]. Suppose there is a secret $s \in \mathbb{Z}_m$ that is to be shared among N parties. First, we secretly choose (independently at random) $N-1$ elements $s_1, s_2, \cdots, s_{N-1}$ from $\mathbb{Z}_m$, compute $s_N = s - s_1 - \cdots - s_{N-1} \mod m$, and give s_i to the i-th party. In order to reconstruct s, N parties expose their shares, and compute $s = s_1 + s_2 + \cdots + s_N$. In this dissertation, we use (N,N)-SS in our protocol construction. Hence, we use the notation $[s]_i$ to denote the share for the i-th party.

The aforementioned (N,N)-SS is additive homomorphic [12]. Particularly, given two secrets a_0 and a_1, we have $[a_0 + a_1]_i = [a_0]_i + [a_1]_i$, where $[a_0 + a_1]_i$, $[a_0]_i$, and $[a_1]_i$ are the i-th shares of $a_0 + a_1$, a_0, and a_1, respectively.

We say that an (N,N)-SS is *indistinguishable* if it is impossible to learn any information about a secret from any of its $N-1$ shares. The indistinguishability of an (N,N)-SS is defined by the following *distinguishing game*, which is played between an adversary and

an oracle.

1. The adversary splits the secret using the (N,N)-SS, and performs any operation on the shares.

2. The adversary submits two distinct secrets a_0 and a_1 to the oracle.

3. The oracle selects a bit $b \in \{0,1\}$ uniformly at random, splits $a_b, a_{\bar{b}}$ using the (N,N)-SS, and returns $N-1$ shares of a_b followed by $N-1$ shares of $a_{\bar{b}}$. The indices of returned shares are specified by the adversary. Without loss of generality, we assume that the returned shares are $[a_b]_1, \cdots, [a_b]_{N-1}$ and $[a_{\bar{b}}]_1, \cdots, [a_{\bar{b}}]_{N-1}$, respectively.

4. The adversary is free to perform any operation on the returned shares, and, finally, outputs a guess for the value of b.

Let $\mathscr{A}$ be an adversary algorithm. Then

$$Pr[\mathscr{A}(a_0, a_1, [a_b]_1, \cdots, [a_b]_{N-1}, [a_{\bar{b}}]_1, \cdots, [a_{\bar{b}}]_{N-1}, 0) = 1]$$

is the probability that $\mathscr{A}$ outputs 1 when $b = 0$, and

$$Pr[\mathscr{A}(a_0, a_1, [a_b]_1, \cdots, [a_b]_{N-1}, [a_{\bar{b}}]_1, \cdots, [a_{\bar{b}}]_{N-1}, 1) = 1]$$

is the probability that $\mathscr{A}$ outputs 1 when $b = 1$. The adversary's advantage is

$$\begin{aligned} &Adv_{\mathscr{A}} = \\ &Pr[\mathscr{A}(a_0, a_1, [a_b]_1, \cdots, [a_b]_{N-1}, [a_{\bar{b}}]_1, \cdots, [a_{\bar{b}}]_{N-1}, 1) = 1] - \\ &Pr[\mathscr{A}(a_0, a_1, [a_b]_1, \cdots, [a_b]_{N-1}, [a_{\bar{b}}]_1, \cdots, [a_{\bar{b}}]_{N-1}, 0) = 1]. \end{aligned}$$

Definition 3. *An (N,N)-secret sharing scheme is said to be unconditionally indistinguishable if for any two secrets, a_0 and a_1, the advantage of any algorithm $\mathscr{A}$ in the distinguishing game is* 0. □

Theorem 2. *The simplified (N,N)-secret sharing scheme is unconditionally indistinguishable.* □

Proof. Suppose $\mathscr{A}$ gets $N-1$ shares $[a_0]_1,\cdots,[a_0]_{N-1}$ generated by the aforementioned simplified (N,N)-SS, where $a_0 \in \mathbb{Z}_m$. Then for any $a_0' \in \mathbb{Z}_m$, there is a unique $[a_0']_N \in \mathbb{Z}_m$ such that $[a_0]_1+\cdots+[a_0]_{N-1}+[a_0']_N = a_0'$. Since a_0' is distributed uniformly over $\mathbb{Z}_m$, the construction of $[a_0']_N$ is equally likely. Therefore, $[a_0]_1,\cdots,[a_0]_{N-1}$ could be $N-1$ shares of any number $a_0' \in \mathbb{Z}_m$ with equal probability. It is true for any $N-1$ shares of a secret in $\mathbb{Z}_m$. We conclude that the distribution of the $N-1$ shares of a_0 and a_1 is identical. Observing the $N-1$ shares, the adversary gets no information about the secret. Therefore, $Adv_{\mathscr{A}} = 0$. □

3.4 Preludes to Protocol Construction

3.4.1 Protocol Overview

Our protocol consists of two sub-protocols: an anonymous message submission (AMS) protocol and a bulk protocol. The bulk protocol is used to send variable length messages and is proposed by Corrigan-Gibbs and Ford in [38]. We adopt it as a building block for our protocol. Our contribution is an efficient peer-to-peer anonymous message submission protocol for scalable size of groups. It is comparable to the anonymous shuffle protocol [21]. The purpose of AMS is to submit a sequence of messages to a designated collector without disclosing the message senders' identities.

AMS consists of six phases: application, encryption, anonymization, verification, decryption, and blame. Compared with existing shuffle protocol, AMS saves the computation time in the following aspects: 1) The core of AMS relies on two light-weight operations – a simplified (N,N)-SS and a symmetric key encryption (i.e., AES); and 2) A member's operations in each phase are independent of that of other members. However, AMS requires the messages to be of the same length, which is a common weakness of existing shuffle protocols. We briefly introduce the six phases of the AMS protocol in the following.

- **Application**: All the messages will be eventually submitted in a sequence. The application phase is to make every member chooses a position in the final sequence but be oblivious to other members' choices. When the application phase ends, every member i obtains a unique position π_i $(1 \leq \pi_i \leq N)$. Here, $[\pi_1, \cdots, \pi_N]$ is a random permutation of $[1, \cdots, N]$.

- **Encryption**: Each member i encrypts his data d_i using his symmetric key k_i and gets the ciphertext $e_i = Enc_{k_i}(d_i)$ with equal length of r bits.

- **Anonymization**: Each member i constructs a data vector $\vec{e}$ such that the π_i-th component is e_i and the rest components are 0. All members split the data vector using the (N,N)-SS, keep their own shares confidential, and send out the remaining shares.

- **Verification**: Each member i reconstructs $\vec{e}$ and checks whether his e_i is the π_i-th component of $\vec{e}$. If any member's message has been altered, an alert message is broadcasted and all members go to the blame phase; otherwise, all members encrypt their symmetric key using a distributed ElGamal encryption [71] and anonymously broadcast the ciphertext via the same technique as in the anonymization phase. Finally, all members obtain a key ciphertext vector $\vec{k'}$ with π_i-th component being ElGamal encryption key k'_i $(1 \leq i \leq N)$.

- **Decryption**: After checking the integrity of key ciphertexts, every member retrieves the key ciphertexts from $\vec{k'}$, decrypts them, and uses the keys to decrypt the corresponding messages.

- **Blame**: All members publishes their secret choices of their positions, the message ciphertexts, the key ciphertexts, the messages received from and sent to other members, and their message logs. All members then replay the protocol execution and every member's behaviors to expose at least one culprit.

We use the bulk protocol [38] to send variable length messages. To execute the bulk protocol, each member generates a ciphertext of his data by XORing the data pseudo random numbers generated by himself. Next, the member constructs a message extractor that helps other members to reproduce the corresponding pseudo random numbers. All members' message extractors are of equal length. Hence, we transform variable length messages into equal length ones. All members then execute the AMS protocol to anonymously broadcast the message extractors. Using the message extractors, the collector can reconstruct the messages by XORing necessary pseudo random numbers.

3.4.2 Anonymous Data Aggregation

A technique used in our protocol is to aggregate data held by group members into a data vector $\vec{v} = [v_{\sigma^{-1}(1)}, v_{\sigma^{-1}(2)}, \cdots, v_{\sigma^{-1}(N)}]$ in a way that a member's position is only known to himself. Here, v_i is the data held by a member i and σ is the permutation

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & N \\ \pi_1 & \pi_2 & \cdots & \pi_N \end{pmatrix}.$$

Each member i obtains the position π_i in the application phase. To achieve this, each member i constructs an individual vector $\vec{v}_i$ such that the π_i-th component is v_i and the

remaining components are 0. Each member i generates N shares for $\vec{v}_i$ by splitting each component of $\vec{v}_i$ using (N,N)-SS. The t-th share of the vector is a vector of all components' t-th shares. Next, all members send their j-th share to member j and keep their own share of the secret. Hence, each member receives $N-1$ shares from other members. Each member i sums up the received $N-1$ shares and his own share to form one share of vector $\vec{v}$, since (N,N)-SS is additive homomorphic. The group members together can reconstruct $\vec{v}_i$.

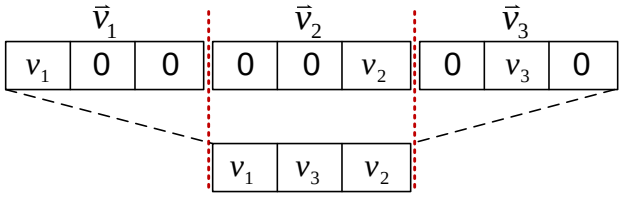


Figure 3.1: Data Aggregation

Fig. 3.1 illustrates the process of the data aggregation. There are three members 1, 2, and 3. In this example, $N = 3$. The positions they obtained are $\pi_1 = 1$, $\pi_2 = 3$, and $\pi_3 = 2$. Members 1, 2, and 3 construct individual vectors $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ as shown in Fig. 3.1. Using a $(3,3)$-SS, members $1,2$ and 3 share $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ with the other two members, respectively. In this way, each member gets two shares from the other members. Due to the homomorphic property of (N,N)-SS, each member sums up the two received shares and the share kept by himself, and gets a share of $\vec{v}$.

3.5 Anonymous Message Submission Protocol

In this section, we present the details of each phase in our AMS protocol. Before the protocol execution, each member i generates a signature key pair (u_i, v_i), where u_i is the signing key and v_i is the verifying key. The signature of a message m is denoted as $Sig_{u_i}(m)$. For the purpose of accountability, every group member keeps a log file during each protocol execution. Given a step in a protocol phase, a member's log file is all his previous incom-

ing and outgoing messages along with the member's generated secrets (e.g., secret shares, random numbers used in encryption) during the execution up to the current step. For each protocol step, the hash value of a member's log file is sent out for the commitment purpose. We use $h_i^{\phi.k}$ to denote the hash value of member i's k-th log file in the phase ϕ. All members agree on a nonce n_R to uniquely identify one protocol execution and a cryptographic hash function, denoted as $h(\cdot)$. They also agree on a symmetric key encryption system and generate their individual encryption key k_i $(1 \leq i \leq N)$. In addition, they agree on a cyclic multiplicative group $\mathbb{G}$ of prime order p and one of its generators g.

Fig. 3.2 shows the application phase. The idea of the application phase is that each member randomly selects a component in the position vector $\vec{p}$, adds 1 to the component, and keeps the component index secret. Eventually, all members exam $\vec{p}$ to see if every member obtains a unique position. A component bigger than 1 indicates a collision. Since each member operates independently, collision is most likely to happen when $\vec{p}$ is N-dimensional. In order to solve this problem, we extend the dimension of $\vec{p}$ to a larger value M. The intuition is that selecting N out of M candidates will have fewer collisions if M is large enough. We will prove in Section 4.5 that if $M = \max(361+N, 2N^2-2N)$, one execution of the application phase achieves no collision with a probability no less than 95%. At the end of the application phase, the members remove all 0 components and shrink the selected position indices to the range between 1 and N.

Initially, every member i $(1 \leq i \leq N)$ constructs an M-dimensional individual vector $\vec{p}_i$ and initializes every component to 0. Member i randomly picks a component p_{ia} in $\vec{p}_i$ and sets it to 1. Next, every member splits the individual vector into N shares using the (N,N)-SS and sends the j-th share to member j. At the same time, member i also receives shares from other members. Utilizing the additive homomorphic property of the (N,N)-SS, i gets one share of $\vec{p}$: $[\vec{p}]_i = (\sum_{a=1}^{N}[p_{a1}]_i \cdots \sum_{a=1}^{N}[p_{aM}]_i)$. With the shares, the group members

Phase 1: Application
Round 1:
Each member $i(1 \leq i \leq N)$ executes the following:

1. Initially, i initializes an individual position vector $\vec{p}_i = [p_{i1}, \cdots, p_{iM}]$, where $M = max\{361+N, 2N^2-2N\}$. He randomly chooses a component, say $p_{i\alpha}$, and sets $p_{i\alpha}$ to 1 and the rest components to 0, indicating that he wants to occupy index α.

2. Member i splits vector $\vec{p}_i$ by dividing each component into N shares. For another member $l(l \neq i)$, i constructs a message $m_l = ([p_{i1}]_l||[p_{i2}]_l||\cdots||[p_{iM}]_l||h_i^{1.1}||n_R)$, and sends $m_l||Sig_{u_i}(m_l)$ to member l.

3. Upon receiving $m_i||Sig_{u_j}(m_i)$ from all other members $j(1 \leq j \leq N, j \neq i)$, i computes $[\vec{p'}]_i = [\sum_{a=1}^{N}[p_{a1}]_i, \sum_{a=1}^{N}[p_{a2}]_i, \cdots, \sum_{a=1}^{N}[p_{aM}]_i]$, constructs the message $m_{i\to *} = [\vec{p'}]_i||h_i^{1.2}||n_R$ and broadcasts $m_{i\to *}||Sig_{u_i}(m_{i\to *})$ to all other $N-1$ members.

4. Upon receiving other members' shares $m_{j\to *}||Sig_{u_j}(m_{j\to *})(j \neq i)$, member i now has all N shares $\{[\vec{p'}]_j\}_{j=1}^{N}$ and reconstructs $\vec{p'} = [\sum_{b=1}^{N}\sum_{a=1}^{N}[p_{a1}]_b, \cdots, \sum_{b=1}^{N}\sum_{a=1}^{N}[p_{aM}]_b]$.

5. Every member i counts the collision headcount in $\vec{p'}$, i.e. $c = \sum_{p'_a>1} p'_a$. If $c > 6$, this round fails and they repeat Round 1 again. After two failures of running Round 1, all members directly go to Phase 6. If $0 < c \leq 6$, they go to Round 2, otherwise, they go to Step 10.

Round 2:
Each member $i(1 \leq i \leq N)$ executes the following:

6. Since every member has $\vec{p'}$, they know which components are occupied and which ones have collisions. In vector $\vec{p'}$, if the value at index π_i is 1, then i does not change his individual vector $\vec{p}_i$; otherwise i resets all vector components to 0, randomly chooses another index except the occupied ones, and sets it to 1.

7. Member i divides each component $p_{ia}(1 \leq a \leq M)$ into N shares, constructs the message $m_l = ([p_{i1}]_l||\cdots||[p_{iM}]_l||h_i^{1.3}||n_R)(l \neq i)$, and sends $m_l||Sig_{u_i}(m_l)$ to member l.

8. Repeating Step 3 to Step 4 in Round 1, i can reconstruct another final vector $\vec{p''}$.

9. i checks the number if there is any collision in $\vec{p''}$. If there is, all members repeat Phase 1. After the second failure of running Phase 1, all members go to Phase 6. Otherwise they go to the next step.

10. Assuming member i selects the α-th $(1 \leq \alpha \leq M)$ index of $\vec{p'}$ (or $\vec{p''}$), member i's final component index in the position vector is $\pi_i = \sum_{t=1}^{\alpha}, p''_t$. In this way, every member obtains a unique component index which is between 1 and N.

Figure 3.2: Application Phase

together reconstruct a vector $\vec{p'}$ and check if there is any collision. There are three cases:

- If $\sum_{p'_i>1} p'_i = 0$, all members directly go to Step 10.
- If $\sum_{p'_i>1} p'_i > 6$, all members repeat Round 1 again.
- If $\sum_{p'_i>1} p'_i \leq 6$, all members go to Round 2.

In Round 2, the members who do not collide with others do not change their positions. Other members randomly pick their positions again except those that have been occupied. All the members execute Step 7 and Step 8, and reconstruct a vector $\vec{p''}$. If no collision exists, all components in $\vec{p''}$ should be either 1 or 0. In the end, member i computes the selected component index which is the α-th index using the formula: $\pi_i = \sum_{t=1}^{\alpha} p''_t$. Note that in Phase 1-Round 1, each component is of $\lceil \log N \rceil$ bits in case that all members select the same component. However, each component can be shrunk to 3 bits in Phase 1-Round 2 so as to save spaces.

Now, one might think of a jamming attack, in which an adversary always fills every component of $\vec{p}$ such that the application phase cannot be ended with a non-collision vector. This is the reason why we need a blame phase, which is to expose at least one faulty member. The success probability of finding a non-collision vector is greater than 99.75% after two runs of Phase 1. Therefore, if the members still cannot find a non-collision vector, it is quite possible that some members deviate from the protocol specification. In this case, all members go to the blame phase to find out the culprit.

Phase 2: Encryption
Each member $i (1 \leq i \leq N)$ executes the following:

1. i encrypts d_i using the symmetric key encryption scheme, and gets $e_i = Enc_{k_i}(d_i)$.

Figure 3.3: Encryption Phase

Phase 3: Anonymization
Each member $i(1 \leq i \leq N)$ executes the following:

1. i constructs an individual vector $\vec{e}_i$ by putting e_i at index π_i and 0 at others.
2. Member i divides each component of $\vec{e}_i$ into N shares. For each member l, i constructs the message $m_l = [e_{i1}]_l||\cdots||[e_{iN}]_l||h_i^{3.1}||n_R$, and sends $m_l||Sig_{u_i}(m_l)$ to member l.
3. Upon receiving $m_i||Sig_{u_j}(m_i)(1 \leq j \leq N, j \neq i)$ from all other members, i locally computes $[\vec{e}]_i = (\sum_{a=1}^{N}[e_{a1}]_i, \sum_{a=1}^{N}[e_{a2}]_i, \cdots, \sum_{a=1}^{N}[e_{aN}]_i)$, generates message $m_{i\to *} = [\vec{e}]_i||h_i^{3.2}||n_R$ and sends $m_{i\to *}||Sig_{u_i}(m_{i\to *})$ to all other $N-1$ members.

Figure 3.4: Anonymization Phase

Phase 4: Verification
Each member $i(1 \leq i \leq N)$ executes the following:

1. Upon receiving $m_{j\to *}||Sig_{u_j}(m_{j\to *})(1 \leq j \leq N, j \neq i)$ from the other members, member i locally reconstructs $\vec{e} = (\sum_{b=1}^{N}\sum_{a=1}^{N}[e_{a1}]_b, \cdots, \sum_{b=1}^{N}\sum_{a=1}^{N}[e_{aN}]_b)$.
2. i checks if π_i-th component of $\vec{e}$ is his ciphertext e_i. If not, i broadcasts a message $b = \perp$ and all members directly go to Phase 6; otherwise, the members go to next step.
3. Each member i picks a random integer $x_i \leftarrow_R \mathbb{Z}_p^*$ and keeps it secret, where p is the order of group $\mathbb{G}$. Member i calculates $y_i = g^{x_i}$ and broadcasts y_i. The group members calculate a common public key $y = \prod_{i=1}^{N} y_i$.
4. Each member i computes an ElGamal encryption of his symmetric key k_i': $k_i' = g^{\gamma}||k_i y^{\gamma}$, where $\gamma \leftarrow_R \mathbb{Z}_p^*$.
5. Member i constructs an N dimensional vector $\vec{k_i'}$ with π_i-th component being k_i' and the rests being 0. Member i divides each component of $\vec{k_i'}$ into N shares, constructs the message $m_l = [k_{i1}']_l||[k_{i2}']_l||\cdots||[k_{iN}']_l||h_i^{4.1}||n_R$, for each other member l, and sends $m_l||Sig_{u_i}(m_l)$ to member l.
6. Upon receiving $m_i||Sig_{u_j}(m_i)(i \neq j, 1 \leq j \leq N)$ from all other group members, i locally computes $[\vec{k'}]_i = (\sum_{a=1}^{N}[k_{a1}']_i, \sum_{a=1}^{N}[k_{a2}']_i, \cdots, \sum_{a=1}^{N}[k_{aN}']_i)$, generates the message $m_{i\to *} = [\vec{k'}]_i||h_i^{4.2}||n_R$, and sends $m_{i\to *}||Sig_{u_i}(m_{i\to *})$ to the other $N-1$ group members.

Figure 3.5: Verification phase

Fig. 3.3 shows the encryption phase. In the encryption phase, each member i encrypts his data d_i using the symmetric key encryption scheme. The ciphertext is $e_i = Enc_{k_i}(d_i)$.

Fig. 3.4 shows the anonymization phase. In this phase, each member i first constructs

Phase 5: Decryption
Each member i executes the following:

1. Upon receiving $m_{j\to *}||Sig_{u_j}(m_{j\to *})(1 \le j \le N, j \ne i)$ from other group members, member i locally reconstructs $\vec{k'} = (\sum_{b=1}^{N}\sum_{a=1}^{N}[k'_{a1}]_b, \cdots, \sum_{b=1}^{N}\sum_{a=1}^{N}[k'_{aN}]_b)$.
2. Member i checks if the π_i-th component of $\vec{k'}$ is k'_i. If not, he will send an alarm message and all members go to Phase 6, otherwise, he broadcasts x_i.
3. Upon receiving $x_j(1 \le j \le N, j \ne i)$, member i checks its validity by checking whether the equation $g^{x_j} = y_j$ holds. If any x_j is invalid, member i reports member j as a faulty member. Having all valid x_j's, the distributed ElGamal decryption key is $x = \sum_{j=1}^{N} x_j \bmod p$. Member i retrieves a secret key $k'_j = g^{\gamma}||k_j y^{\gamma}$ from $\vec{k'}$, decrypts it, and obtains $k_j = k_j y^{\gamma}/(g^{\gamma})^{x}$. Having k_j, member i decrypts the corresponding ciphertext $d_{\sigma^{-1}(j)}$ from $\vec{e}$.
4. Member i sends a copy of $(d_{\sigma^{-1}(1)}, \cdots, d_{\sigma^{-1}(N)})$ to the collector.
5. The collector checks the validity of the sequence $(d_{\sigma^{-1}(1)}, \cdots, d_{\sigma^{-1}(N)})$. If one or more sequences are different from the others, all members go to Phase 6.

Figure 3.6: Decryption phase

an N-dimensional message vector $\vec{e}$ with the π_i-th component being the ciphertext of e_i and the rest being 0. All members split their message vectors and send shares to other corresponding members. Therefore, each member i receives $N-1$ shares. Every member i sums up those $N-1$ shares together with his own share and obtains one share of $\vec{e}$.

Fig. 3.5 shows the verification phase. After the reconstruction of $\vec{e}$, each member i checks whether the π_i-th component is equal to his ciphertext e_i or not. If not, the member i broadcasts an alarm message $b = \perp$. If there is a $b = \perp$ message, all members go to the blame phase, otherwise all members encrypt their individual symmetric key using a distributed ElGamal encryption system [71]. With the distributed ElGamal encryption, the message is encrypted by a common public key and the decryption requires all members' secret keys. This encryption is necessary because the members will not expose their key in the blame phase if the key transmission is broken by a malicious adversary. We do not

Phase 6: Blame
The members may enter the blame phase from the application, verification, or decryption phase. There should be a corresponding blame for each different entrance. However, the descriptions are similar and we put them together to save space. We use *entrance phase* to denote the phase from which the members enter the blame phase. When they enter this phase, members publish as much secret information as they could and replay the protocol execution as far as they could go.
Each member executes the following:

1. If the entrance phase is Decryption, every member destroys his own secret key x_i and symmetric key k_i.

2. Each member i reveals his own secret shares when he splits vectors before entrance phase, i.e., $[\vec{p}_i]_i$, $[\vec{e}_i]_i$, and $[\vec{k'}_i]_i$. These secret shares are never sent out during the previous phases. Members also reveal all incoming and outgoing messages along with the signatures up to the entrance phase. If the entrance phase is after the encryption phase, members also need to reveal the randomness used in the encryption phase, e.g. initialization vector for AES-CTR. With these information, member i can replay any other member's behaviors and reconstruct their individual vectors.

3. Member i exposes j as faulty if

   - The hash value of any replayed message does not match the corresponding hash value in the log file;
   - Any of the replayed messages does not match the corresponding messages received in the previous phase;
   - In the application phase, the vector $\vec{p}_j$ is ill-formed, i.e., not in the form of one component being 1 and the rests being 0;
   - In the anonymization phase, the ciphertext vector $\vec{e}_j$ is ill-formed, i.e., $\exists n \in \{1, \cdots, N\}\ s.t.\ n \neq j \wedge e_n \neq 0$;
   - In the decryption phase, member j's key vector $\vec{k'}_j$ is ill-formed;
   - If the entrance phase is Anonymization or Decryption, everything is verified to be correct, but j claims that his ciphertext or key is corrupted.

Figure 3.7: Blame phase

require the ElGamal encryption to be IND secure. An encryption scheme that can maintain confidentiality is sufficiently suitable for our protocol. The reason is that the plaintext, i.e., the secret key, is a random number and the resulting ciphertext is thus indistinguishable from a random number. The members secretly construct an N-dimensional key vector $\vec{k'}$

such that the π_i-th component is k'_i. To achieve this, they use the same data aggregation technique that is introduced in Section 3.4.2.

Fig. 3.6 shows the decryption phase. After the key vector is reconstructed, all members retrieve each key ciphertext k'_i from $\vec{k}'$. If any ciphertext is not corrupted, all members publish their secret x_i so that the distributed ElGamal decryption key can be constructed to decrypt the ciphertexts of the secret keys. Finally, the obtained secret keys are used to decrypt the ciphertexts of the messages.

Fig. 3.7 presents the blame phase. Group members publish all secret information, including the secret choice of component index, all messages received from and sent to other members, their secret shares, message ciphertexts, and key ciphertexts, and replay the protocol execution again to find the culprit.

3.6 Analysis of AMS Protocol

This section analyzes the security and efficiency of AMS protocol.

3.6.1 Security

In this section, we prove that the security properties defined in Section 3.2.3 can be preserved under malicious attacks.

3.6.1.1 Anonymity

We prove the anonymity (Definition 2) of our protocol in two aspects. First, we prove that if the collector and some members behave dishonestly and learn some associations between the identities and the ciphertexts, they cannot pass the verification phase and thus they cannot learn the plaintexts. Second, we prove that if the collector and the dishonest

members behave honestly, they pass the verification phase and learn the final decrypted plaintexts, but they will not learn the associations between the identities and the plaintexts.

Theorem 3. *In the AMS protocol, if the collector colludes with no more than $N-2$ group members and the symmetric encryption is IND-CPA secure, the collector has only a negligible probability to get the associations between the messages and the identities of the honest group members.* □

Proof. Our proof is done in two parts. First, we show that in the verification phase, either there is exactly one copy of the ciphertext for each honest member, or the deviation from the protocol could be detected by the members before the collector gets the secret keys. Second, we show that an adversary who can win the anonymization game while maintaining the security properties can also win the distinguishing game, which is a contradiction because the (N,N)-SS is unconditionally indistinguishable and the underlying encryption scheme is IND-CPA secure.

Part 1: The honest members' messages cannot appear more than once due to the indistinguishable property of (N,N)-SS. If the adversary can reproduce the honest members' messages, then the adversary can break the (N,N)-SS scheme from less than N shares, which is a contradiction to the property of the (N,N)-SS.

Now we show that if the adversary modifies the honest members' messages, this modification will be detected in the verification phase, since the honest members do not find their ciphertexts in the message vector. Hence, the protocol is abort and the adversary cannot get the secret keys of the ciphertexts. In this case it is infeasible for the adversary to learn the plaintexts without the secret keys since the underlying encryption scheme is semantically secure.

Part 2: Now suppose that the adversary honestly handles all ciphertexts belonging to the

honest members. If there is a probabilistic polynomial time algorithm $\mathscr{D}$ that allows this adversary to win the anonymization game with a non-negligible probability, we show how to use $\mathscr{D}$ as a subroutine to the algorithm $\mathscr{A}$ that wins the distinguishing game with a non-negligible probability. Because the underlying (N,N)-SS is unconditionally indistinguishable, this is a contradiction, and we conclude that no such $\mathscr{D}$ exists.

Let the set of $N-k$ honest group members in the anonymization game be $\mathscr{K} = \{1, \cdots, N-k\}$. Let $\mathscr{D}$ be an algorithm that allows the adversary to win the anonymization game with a non-negligible probability. Then there exist honest group members α and β such that for a negligible function in λ, $\varepsilon(\lambda)$, the advantage

$$Adv_{\mathscr{D}} > \varepsilon(\lambda).$$

To apply $\mathscr{D}$, $\mathscr{A}$ must simulate the oracle in the anonymization game to reproduce the view of the adversary. We show how $\mathscr{A}$ is able to achieve this.

Algorithm $\mathscr{A}$ begins by applying $\mathscr{D}$ to learn its choices in Step 1 of the anonymization game. $\mathscr{A}$ therefore learns the following:

- two honest participants α and β, and two plaintext messages d_0 and d_1; and
- a message d_i for each honest member i.

Then, for each honest member i, $\mathscr{A}$ chooses a secret key, k_i. $\mathscr{A}$ selects plaintext messages d_0 and d_1 for α and β, respectively.

$\mathscr{A}$ is now ready to play the role of the oracle in the anonymization game by simulating the messages of the honest members in the protocol execution. For each phase of the protocol, we explain how $\mathscr{A}$ is able to reproduce the messages sent in that phase.

Phase 1: $\mathscr{A}$ has all the necessary information to exactly reproduce this phase.

Phase 2: For all honest members other than α and β, $\mathscr{A}$ encrypts d_i using k_i, which results in the ciphertext e_i. $\mathscr{A}$ then encrypts d_0 and d_1 using the keys k_0 and k_1, respectively, getting $e_0 = E_{k_0}(d_0)$ and $e_1 = E_{k_1}(d_1)$.

Phase 3: $\mathscr{A}$ gives e_0 and e_1 to the distinguishing game oracle, getting back $[e_b]_1, \cdots,$ $[e_b]_{N-1}$ and $[e_{\bar{b}}]_1, \cdots, [e_{\bar{b}}]_{N-1}$.

Suppose that in the application phase, α and β obtain positions π_α and π_β, respectively. $\mathscr{A}$ now constructs $N-1$ shares of $\vec{e}_\alpha$ as follows. The components at positions other than π_α are set to 0 and their shares can be easily constructed. The π_α-th component of $N-1$ share vectors are respectively filled by $N-1$ shares of e_b obtained from the distinguishing game oracle. $N-1$ shares of $\vec{e}_\beta$ are constructed in the same way except that $e_{\bar{b}}$ is used instead of e_b.

At the end of Phase 3, $\mathscr{A}$ needs to compute a share of the message vector $\vec{e}$. $\mathscr{A}$ does not have the N-th share of e_b or $e_{\bar{b}}$. But he has the original ciphertexts e_0 and e_1. With any $N-1$ shares and e_0 (resp. e_1), $\mathscr{A}$ could compute the last share such that all shares are reconstructed as e_0 (resp. e_1). At this moment, $\mathscr{A}$ randomly picks a ciphertext from e_0 and e_1, computes the last share, and puts it on the π_α-th component for a member α. The other ciphertext is used for a member β. We argue that this does not change $\mathscr{D}$'s view when it does the reconstruction. Because all honest group members' positions are chosen randomly and the position sequence is a random permutation on the set $\{1, \cdots, N\}$, the probability that e_0 appears in π_α and e_1 appears in π_β is equal to the probability that e_1 appears in π_α and e_0 appears in π_β. Therefore, randomly allocating e_0 and e_1 on π_α and π_β does not change $\mathscr{D}$'s view.

Phase 4 and Phase 5: $\mathscr{A}$ has all the necessary information to complete the phases.

Now $\mathscr{A}$ simulates the view of the adversary and applies $\mathscr{D}$ to the view. If $\mathscr{D}$ outputs 1, then

$\mathscr{A}$ outputs 1; if $\mathscr{D}$ outputs 0, $\mathscr{A}$ outputs 0.

We now analyze the probabilities of $\mathscr{A}$ outputting 1 when the distinguishing oracle chose $b = 0$ and when the distinguishing oracle chose $b = 1$. If $b = 0$, then the view of the adversary is $(\alpha, \beta, d_0, d_1, 0)$. If $b = 1$, then the view of the adversary is $(\alpha, \beta, d_0, d_1, 1)$. Based on our assumption that $\mathscr{D}$ wins the anonymization game, we have that $Adv_{\mathscr{D}} > \varepsilon(\lambda)$. Now we make a simple substitution,

$$Adv_{\mathscr{A}} = Adv_{\mathscr{D}} > \varepsilon(\lambda).$$

We conclude that $\mathscr{A}$ can win the distinguishing game with a non-negligible probability, which contradicts with the unconditionally indistinguishable property of the (N,N)-SS. □

3.6.1.2 Integrity

In the verification phase, the honest members verify whether their ciphertexts appear in the message vector $\vec{e}$. If the verification fails, i.e., the ciphertext of at least one honest member has been altered, the protocol aborts. So the adversary cannot get the secret keys. If the verification succeeds, the honest members send their secret keys to the collector who can then decrypt the ciphertexts. We conclude that the authentication defined in Section 3.2.3 is preserved.

3.6.1.3 Confidentiality

In Phase 3, all members send out $N-1$ shares of the individual vector to the other members except for one share. Because the (N,N)-SS is unconditionally indistinguishable, even if $N-1$ group members collude, they will not get any useful information about the honest messages. Therefore, we conclude that the confidentiality defined in Section 3.2.3 is preserved.

3.6.1.4 Accountability

Accountability of an anonymous messaging system is first proposed and studied in [38]. The proof of our system's accountability follows the similar philosophy.

A member i *cannot* expose another honest member j unless i obtains evidence of j's misbehaviors that are verifiable to a third member. Accountability is maintained if

Condition I: no member can accuse an honest member;

Condition II: at the end of a protocol execution, either (a) the protocol completes successfully; (b) the protocol fails in application phase, which is not caused by malicious attacks; or (c) at least one faulty member is exposed.

Condition I is shown as follows. An evidence of member j's misbehavior consists of some "incorrect" data content d_j signed by j at the step stp of some phase, together with all his log messages and all members' generated secret shares up to a particular step stp. Member j will be exposed as faulty only if he signed some incorrect messages or his behavior deviates from the protocol. For example, a member j has two non-zero elements in his individual encryption vector $\vec{e}_j$ to block another member's message. However, this contradicts the assumption that the member j is honest. A member i may collude with another member k to accuse an honest member j by forging a message signed by the member k prior to the step stp and the message is different from the one received and used in generating j's message m_j. The message received by j must be valid because j will check against k's signature. However, since every message is signed and contains a unique nonce, two valid messages at the same step will also expose k as faulty, which weaken i's accusation against j.

Since the application phase cannot guarantee to always find each member a unique position, our AMS protocol may fail even when all members are honest. However, as shown in the following section, the failure probability of two application executions is no more than 0.25%, which is small enough to omit in practice. When the application fails, members still enter the blame phase to check if it is a failure caused by malicious attacks or not. If it is, a faulty member will be exposed.

We now show that our AMS protocol satisfies condition II(c), i.e., a faulty member is exposed in the blame phase. A member enters the blame phase if a) the application phase fails twice; b) some members report that his ciphertext is manipulated in the verification phase; or c) some members report that his encryption key is manipulated in the decryption phase. In case a), the blame phase requires members to recover their secret shares and log messages in the previous two attempts. This enables a member to reconstruct every member's position vector and a faulty member will be immediately exposed. In cases b) and c), all members recover their secret shares and log messages up to the current step. Also it will expose a faulty member or a member i himself if he untruthfully reports that his information is manipulated.

3.6.2 Efficiency

In this section, we analyze the success probability of finding a non-collision vector in Phase 1, and the communication rounds.

3.6.2.1 The Success Probability in Phase 1

Theorem 4. *In the application phase, at the end of round 2, the success probability of finding a non-collision vector $\vec{p}$ is greater than 95%.* □

Proof. Let η_{ij} denote the event that members i and j choose the same position, leading to a collision. Let ϕ_i^w denote the event that a member i chooses position $w(1 \leq w \leq M)$. Because all members choose their positions independently at random, the probability $p(\phi_i^w) = 1/M$. Using Bayes' theorem, we have

$$p[\eta_{ij}] = \sum_{w=1}^{M} p[\phi_i^w | \phi_j^w] p[\phi_j^w] = \sum_{w=1}^{M} \frac{1}{M} p[\phi_j^w] = \frac{1}{M}.$$

We define an indicator random variable X_{ij} such that $X_{ij} = 1$ if the event η_{ij} happens and $X_{ij} = 0$ if η_{ij} does not happen. Since X_{ij} only takes the value 0 or 1, it is a Bernoulli variable. We define $\mathscr{X}$ to be the number of collisions, i.e. $\mathscr{X} = \sum_{i<j} X_{ij}$. Then we compute

$$\begin{aligned} E[\mathscr{X}] &= E[\sum_{i<j} X_{ij}] = \sum_{i<j} E[X_{ij}] \\ &= \sum_{i<j} p[X_{ij} = 1] = \sum_{i<j} \frac{1}{M} = \frac{1}{M} \binom{N}{2}. \end{aligned}$$

Assume that $E[\mathscr{X}] = \mu$, from the above equation, we get $\mu = N(N-1)/2M$. If we choose $M = 2N^2 - 2N$, then $\mu = 1/4$. Utilizing the Chernoff bound [33], we have

$$p[\mathscr{X} \geq (1+\delta)\mu] \leq (\frac{e^{\delta}}{(1+\delta)^{1+\delta}})^{\mu}$$

for any $\delta > 0$. Let $\delta = 11$, we get $(1+\delta)\mu = 3$ and $p[\mathscr{X} \geq 3] \leq (\frac{e^{11}}{(1+11)^{1+11}})^{0.25} \approx 0.009$. Hence, at the end of Phase 1-Round 1, the probability that there exist no less than 6 collisions is not greater than 0.9%. In Phase 1-Round 2, since there are at most 6 members who collide with each other, we need to have enough positions for the colliding members to choose from. Suppose the positions left at the end of round 1 is M_l, the probability that 6 members' positions do not collide is $p = \frac{M_l(M_l-1)(M_l-2)\cdots(M_l-5)}{M_l^6}$. We let $p > 0.96$, and get the smallest integer which is 361. So the lower bound of M is $361 + N$. We define ρ to be the event that the members find a non-collision vector in Phase 1. $p[\rho] \geq p[\text{succeed in round 1} \wedge \text{succeed in round 2}] \geq (1 - 0.009) \times 0.96 = 0.95136$. The theorem is proved. □

We define the total number of execution times of Phase 1 to T. The expectation $E[T] = \sum_{i=1}^{\infty} i(1-p[\rho])^{i-1}p[\rho] = 1/p[\rho]$. Since $p[\rho] > 95\%$, the average number of executions needed to find a non-collision vector is $\lceil 1/0.95 \rceil = 2$. In fact, the failure probability is less than $5\% \times 5\% = 0.25\%$ after two executions of the application phase.

3.6.2.2 Communication Rounds

All phases are parallelizable. In Phase 1, the number of communication rounds is 4. There is no communication in Phase 2. Phase 3 needs 2 communication rounds. Phase 4 needs 4 communication rounds. Finally, there are 2 communication rounds in Phase 5. The expected total number of communication rounds is $4/p[\rho]+8$, which is approximately 12 and independent of the group size. In contrast, in Brickell and Shmatikov's [21] protocol, the total communication rounds is $2N+7$. Note that our protocol is probabilistic, while the protocol in [21] is deterministic.

3.7 Bulk Protocol

The bulk protocol is to handle variable length messages, which was first proposed in [38]. The anonymity property of the bulk protocol is guaranteed by the shuffle protocol, which is replaced by AMS protocol in this dissertation. We use bulk as a built-in block and give its description here. More analysis of the bulk protocol can be found in [38].

All members 1, 2, $\cdots$, N hold messages $msg_i (1 \leq i \leq N)$ of variable lengths L_1, L_2, $\cdots$, L_N, respectively. Each member i has a key pair (u_i, v_i) for signature and another key pair (x_i, y_i) for encryption. All members know other members' public keys, and have agreed upon session nonce n_R and an order of the members.

There are five phases in the bulk protocol: message extractor generation phase, message extractor distribution phase, data transmission phase, message reconstruction phase, and

blame phase.

Phase 1: Message extractor generation
Each member i executes the following:

1. Member i chooses a random number generator seed s_{ij} for each other member j and generates an L_i-bit random number $C_{ij} = RAND(L_i, s_{ij})$ $(i \neq j)$.
2. Member i XORs his message with all C_{ij}'s and obtains C_{ii}, $C_{ii} = C_{i1} \oplus \cdots \oplus C_{i(i-1)} \oplus msg_i \oplus C_{i(i+1)} \oplus \cdots \oplus C_{iN}$.
3. Member i computes $H_{ij} = h(C_{ij})$, encrypts each seed s_{ij}, $S_{ij} = Enc_{y_i}(s_{ij}, r_{ij})$, using member j's public key and a secret random number r_{ij}. Member i sets S_{ii} to a random number.
4. Member i collects all hash values and seed ciphertexts to form two vectors $\vec{H}_i = \{H_{i1}, \cdots, H_{iN}\}$, $\vec{S}_i = \{S_{i1}, \cdots, S_{iN}\}$, and gets the message extractor d_i, $d_i = \{L_i, h(msg_i), \vec{H}_i, \vec{S}_i\}$.

Figure 3.8: Message extractor generation phase

In the message extractor generation phase (Fig. 3.8), each member i generates a message extractor $d_i = (L_i, h(msg_i), \vec{H}_i, \vec{S}_i)$ which is used to recover the message. Here, $\vec{H}_i$ is the vector of hash values generated from $h(C_{ij})(1 \leq j \leq N, j \neq i)$, where C_{ij} is the pseudo random number used to XOR member i's message. $\vec{S}_i$ is a vector of the ciphertexts of the pseudo random generator seeds. The message extractor consists of all the information needed to recover the original message.

Phase 2: Message extractor distribution
The group of members run AMS protocol introduced in Section 3.5:

1. Each member i submits the message extractor d_i to AMS protocol.
2. All members collaboratively run the AMS protocol.
3. Each member i gets a sequence of message extractors with d_i appearing at π_i-th position.

Figure 3.9: Message extractor distribution phase

In the message extractor distribution phase (Fig. 3.9), all members submit their message extractors to the AMS protocol, the output of which is a randomly permuted sequence of

message extractors. The anonymity property of the AMS protocol ensures that the message extractors cannot be traced to their providers.

Phase 3: Data transmission

1. Each member j now holds a message extractor sequence $D = (d_{\sigma^{-1}(1)}, \cdots, d_{\sigma^{-1}(N)})$, where $(d_{\sigma^{-1}(1)}, \cdots, d_{\sigma^{-1}(N)})$ is a permutation of $(d_1, \cdots, d_N)$.
2. Member j recognizes his own message extractor d_j in D, and sets $C'_{\sigma(j)j} = C_{jj}$. For each $d_i \in D$ $(i \neq \sigma(j))$, member j extracts j-th component S_{ij} from $\vec{S}_i$ and decrypts S_{ij} using his private key x_j obtaining the seed s_{ij}.

   Member j computes the pseudo random number $C_{ij} = RAND(L_i, s_{ij})$ and checks $h(C_{ij})$ against H_{ij}. If the check has passed, member j sets $C'_{ij} = C_{ij}$; member j sets $C'_{ij} = NULL$ otherwise.
3. Member j assembles all C'_{ij}'s in the received order, i.e. putting C'_{ij} at the i-th position, forms the message $m_j = C'_{1j}||\cdots||C'_{Nj}||n_R||h_j^{3\cdot 1}$, and sends $m_j||Sig_{u_j}(m_j)$ to the collector.

Figure 3.10: Data transmission phase

Phase 4: Message reconstruction

1. The collector checks the hash value of each C'_{ij} from member j against H_{ij} from $d_i \in D$. If C'_{ij} is $NULL$ or $h(C'_{ij}) \neq H_{ij}$, the message on the i-th position is corrupted and ignored.
2. For each uncorrupted slot, the collector reconstructs the data by computing $d_i = C'_{i1} \oplus \cdots \oplus C'_{iN}$.

Figure 3.11: Message reconstruction phase

Phase 5: Blame

1. If C'_{ij} is corrupted in Phase 4, all members run AMS protocol again, during which member i sends an audit message $A_i = \{j, S_{ij}, s_{ij}, r_{ij}\}$. All members check the validity of the audit message: $S_{ij} = Enc_{y_j}(s_{ij}, r_{ij})$ and $H_{ij} = h(RAND(L_i, s_i j))$. If the audit message is valid, member j is exposed as faulty.

Figure 3.12: Blame phase

In the data transmission phase (Fig. 3.10), all members have N message extractors. A member j extracts the pseudorandom number generator seeds prepared for him by extracting the j-th components from each extractor's seed vector and decrypting it. Each member

j generates the pseudo random numbers $C'_{ij}(1 \leq i \leq N)$, assembles them to form the j-th row in the message submission matrix, and sends the row to the collector. The collector receives all the C'_{ij}'s $(1 \leq i, j \leq N)$, forms a matrix by putting $C'_{ij}(1 \leq i \leq N)$ in a row, and recovers a message by XORing one column of the matrix (Fig. 3.11).

If any restored message is meaningless, all members go to the blame phase to find the culprit (Fig. 3.12).

3.8 Performance Evaluation

We made a proof-of-concept implementation of our protocol, and compared it with the protocols in [21] and [38]. All the experiments were carried out on an Intel® Core™2 Duo CPU P8600 @ 2.40GHz computer. We used Crypto++ [41] as our underlying cryptographic library.

Dissent [38] uses the same technique as the one in [21] in its shuffle protocol, which requires an IND-CCA2 secure cryptosystem. Since Crypto++ does not have any IND-CCA2 secure algorithm in the standard model, we implemented the Cramer-Shoup cryptosystem by using SHA-1 and a $3,072$-bit integer group, MODP-15 [69]. To compare with the implementation in [38], we also implemented a shuffle based on RSA-OAEP [10], which is CCA-2 secure only in the random oracle model. The modulus for RSA is of $3,072$ bits. The symmetric key cryptosystem in AMS protocol was implemented by using AES-128 in counter (CTR) mode, which is CPA secure in the standard model [9]. The distributed ElGamal encryption used in AMS protocol is also built on top of MODP-15 group. All the modules were implemented in C++ and compiled by g++ 4.4.3 with `-O3` level optimization.

First we tested the number of rounds needed in Phase 1. We increased N from 100 to

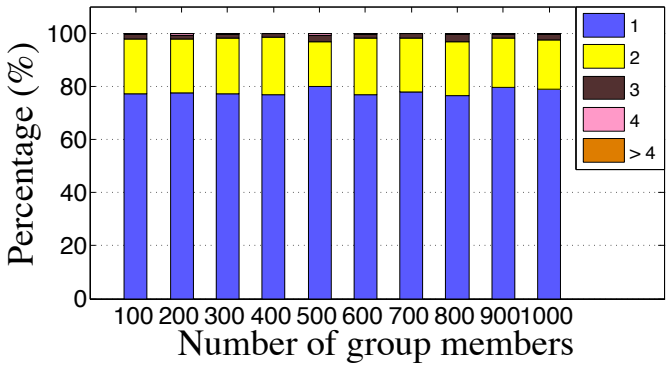
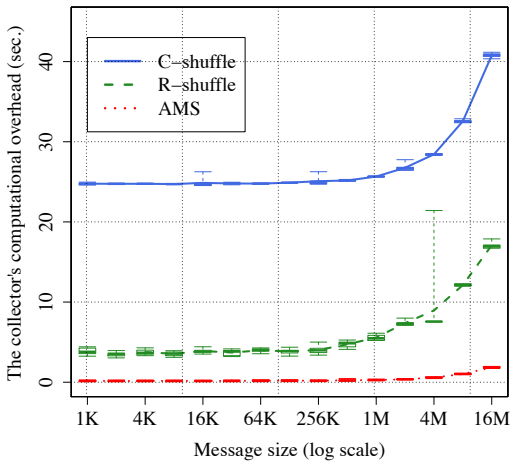


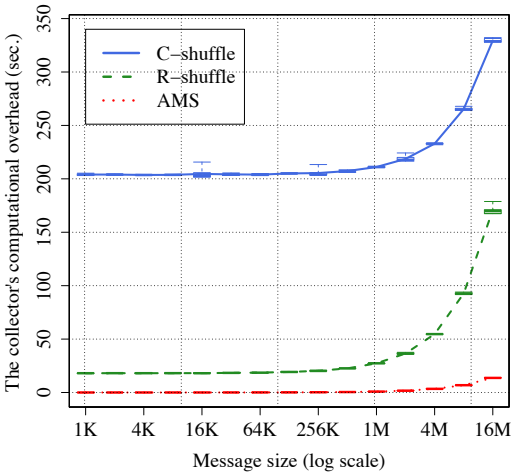
Figure 3.13: The distribution of needed rounds in Application

1,000 using an increment of 100. For each value of N, we ran Phase 1 for 1,000 times and recorded the number of rounds needed to find a non-collision vector $\vec{p}$. We remark that one Application execution contains two rounds. Fig. 3.13 shows the result. We observe that at least 77% of the tests were completed in 1 round no matter what N is. Furthermore, 97% of them were completed within 2 rounds, finding a non-collision vector $\vec{p}$.

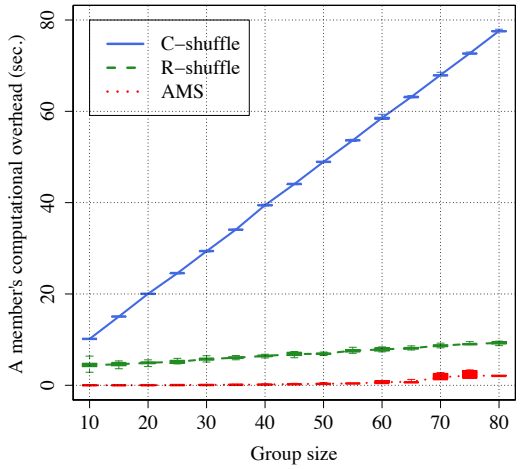
Next, we compared the computational overhead of our protocol to that of the shuffle protocol in [21]. We used "C-shuffle" and "R-shuffle" to denote the Cramer-Shoup based and the RSA based shuffle implementations, respectively. To obtain a point in a plot, we ran the corresponding simulation for 10 times and calculated the average over the 10 runs. Also, we used box plot to visualize the distribution of the 10 results. The top and bottom bars of a box plot represent the max and min values in the collection. A squeezed box denotes that the data points are close to the average value. Fig. 3.14a-3.14d show the computational overheads for the collector and the members under different group sizes N and message sizes l. Note that the little bar on each line is a squeezed box plot. First we fixed $N = 40$, changed l from 1KB to 16MB, and compared the computation time needed for each member and the collector in different anonymous message systems. Note that in Fig. 3.14b, the computation time of the collector in our protocol is much shorter than that in C-shuffle and



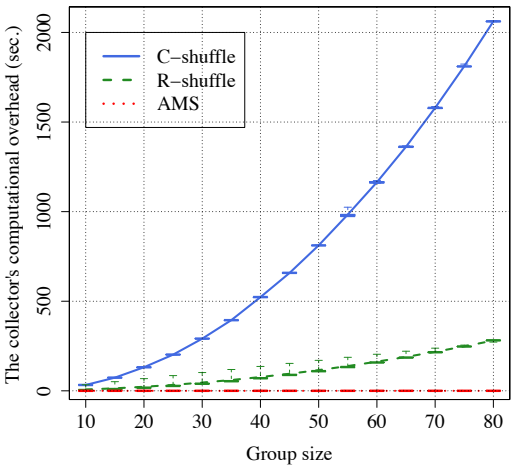
(a) A member's computational overhead vs. message size



(b) Collector's computational overhead vs. message size



(c) A member's computational overhead vs. group size



(d) Collector's computational overhead vs. group size

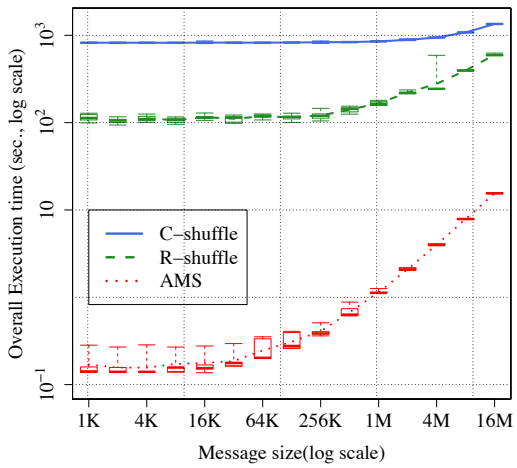
Figure 3.14: Protocol computational overhead

R-shuffle. This is because the decryption time of an IND-CCA2 secure cryptosystem is much longer than that of a symmetric key decryption algorithm. In addition, the shuffle protocol needs $N^2 + N$ decryptions on the collector, while the collector in our protocol only needs to decrypt N ciphertexts. From Fig. 3.14a and 3.14b, we observe that AMS performs better than the two shuffle implementations, especially when the message size goes big. In Fig. 3.14c and 3.14d, we fixed l to 1KB, and changed N from 10 to 80 at an increment of

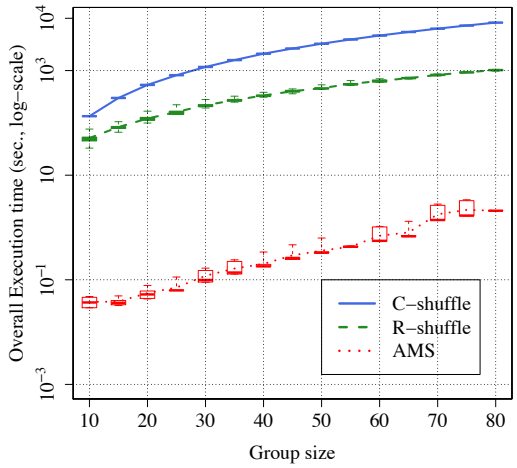
10. Since the number of communication rounds depends on N in the shuffle protocol, the execution time of each member increases with N, while the execution time of each member in our protocol increases slowly. In both sub-figures, for the shuffle implementations, the execution time increases polynomially in N. However, the execution time of the collector in our protocol is almost constant, which shows that the addition and symmetric key decryption is so fast that we cannot tell the time difference when the group size is small. We remark that a party's computational overhead sometimes is different from the protocol's execution time. In the AMS system, a member's computation is in parallel with other members, and the protocol is finished within constant rounds while the shuffle protocol needs $O(N)$ communication rounds.

From Fig. 3.15a and Fig. 3.15b, we observe that the execution time of AMS is much shorter than that of the shuffle implementations. We also ran the simulation of AMS under larger group sizes, from 100 to 400. The results are plotted in Fig. 3.15c. The blue and green lines in the plot denote the execution time of C-shuffle under a 15-member group and R-shuffle under a 30-member group, respectively. We observe that the execution time of AMS was much faster even when the group size was increased to 400. The longest time cost by AMS in the tests when the group size was 400 is still better than the average time of R-shuffle when the group size was 35.

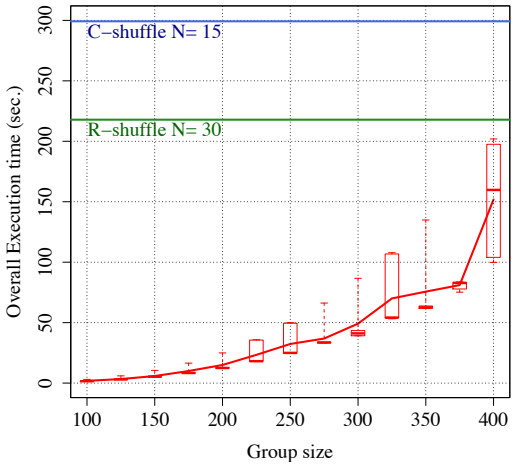
When dealing with variable length messages, we adopted Bulk protocol [38] by using AMS, instead of the shuffle protocol, to send message extractors. We implemented Bulk protocol on top of AMS, C-shuffle, and R-shuffle, respectively, and compared their execution times. The comparison result is shown in Fig. 3.15d. To do the comparison, we fixed the group size as 40 and let each member randomly pick a message size from $[l-1K, l+1K]$, where $l = 1K, 2K, 4K, \cdots, 8M, 16M$. The message size is in terms of bytes. After all members obtained their message size, they executed bulk protocol to submit the



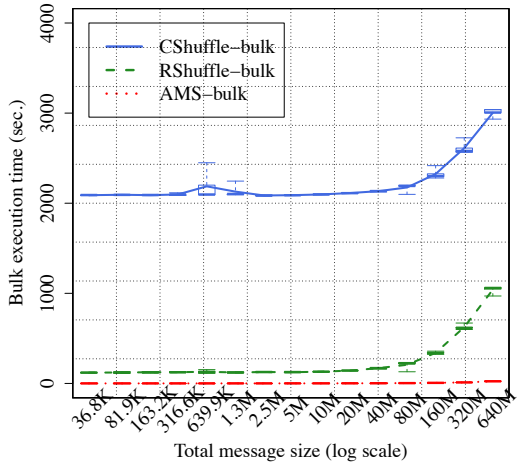
(a) System execution time vs. message size



(b) System execution time vs. group size



(c) AMS Execution time for scalable groups

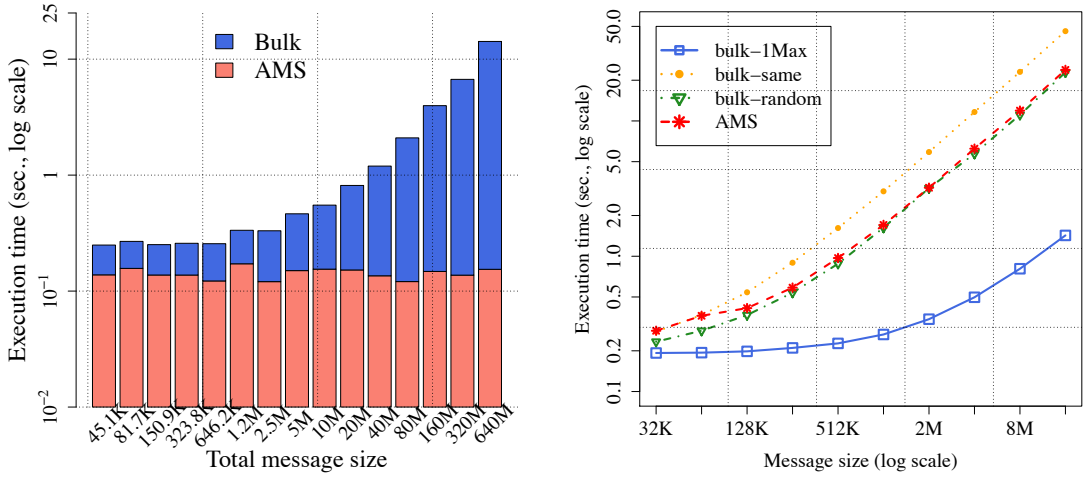


(d) Bulk execution time vs. message size.

Figure 3.15: Protocol execution time

messages and repeated the protocol execution for 10 times. In Fig. 3.15d, the x-axis represents the total message size and the y-axis represents the bulk execution time. We observe that AMS performs significantly better than the shuffle protocols. For example, to handle a total message size of 640MB, the average running time is 3003.97471 seconds for C-shuffle, 1040.87252 seconds for R-shuffle, and 23.89172 seconds for AMS. We also note that the difference between AMS and R-shuffle is not remarkably big when the transmitted message is of small size. This is because the data transmission in bulk will cost more time

than shuffling message extractors when the transmitting data size is small. Hence, we are curious about the time cost on sending message extractors and data transmission by using AMS as a sub-protocol. Again, we fixed group size as 40 and let each member randomly pick their message size in the same way as in the previous test. We separately recorded the



(a) Time cost on AMS and Data Transmission

(b) Comparison between AMS and Bulk

Figure 3.16: Bulk performance using AMS

time cost on AMS and bulk data transmission and show the result in Fig. 3.16a. The y-axis represents the time cost and is in log-scale. First, we observe that the time cost on AMS is almost the same, indicating that transmitted message extractors size is independent of the transmitted total message size. Besides, we note that the time cost on AMS is nearly 0.1 seconds while that on bulk data transmission increases to over 500 seconds.

We introduced the bulk protocol because AMS itself cannot handle variable length message transmissions. If we want to use AMS as a stand-alone system to transmit variable length messages, we have to pad the messages to a fixed maximum length over all to-be-sent messages. Since AMS is so fast, we are curious about its performance against the bulk protocol. To this end, we set up another experiment in the following way. First, we fixed the group size as 40 and set $l = 32K, 64K, 128K, \cdots, 8M, 16M$. For each l, we ran

bulk protocol, using AMS as a sub-protocol, under 3 different settings: 1Max, same-length, and random-length. In the 1Max setting, one member set the message length as l and the remaining members set their message length as $l/1024$; in the same-length setting, each member set his message length as l; in the random-length, each member set his message length as a random number *rand* such that $0 < rand \leq l$. Each member set the message size as l when we ran AMS as a stand-alone system. The result is shown in Fig. 3.16b. From the figure, we observe that AMS alone took shorter time than that of the bulk protocol under the same-length setting. We also note that AMS (represented by the red dash line) cost almost the same time as that of the bulk protocol under the random-length setting (represented by the blue doted line). At last, it is obvious that AMS cost more time than that of the bulk protocol under the 1Max setting. This is reasonable and expected because under the 1Max setting, it is actually a special circumstance favoring the bulk protocol. In this case, the AMS system has to do $N-1$ paddings and transmitting more unnecessary contents. Therefore, we suggest to use AMS directly when the sizes of submitted messages are close to one another. This situation is common in the real world life. Online surveys introduced in our introduction section is one of such examples. Answers to a set of questionnaires usually have similar lengths because a set of questionnaires consist of either multiple-choice questions or short comments.

3.9 Conclusions

In this dissertation, we have proposed an efficient online anonymous message submission protocol. Utilizing the simplified secret sharing scheme, we have designed a novel position application technique, in which all members secretly select their positions in a position vector, such that a member knows nothing about the other members' message positions. We have introduced a data aggregation technique, in which all members aggregate their

messages into a message vector and submit it to the collector without exposing their identities. We have theoretically proved that our protocol is anonymous under malicious attacks. We have also demonstrated the efficiency of our protocol through rigorous evaluations and experiments.

Part II

LOCATION PRIVACY

CHAPTER 4

Location Privacy in Location Based Social Networks

Location based social networks (LBSNs) are now enjoying a great deal of attentions. Many companies, such as Foursquare, Google Latitude, Facebook, etc., are rushing to have LBSNs included in their services or products. In LBSNs, users ***"check in"*** at a venue and leave ***"tips"*** which serve as their personal suggestions for what to do, see or eat at a location. Users share the tips with their friends. In this dissertation, we call all the information a user wants to share on a specific venue as her ***"check-in"***, which includes the tips, the check-in time, etc. Besides, users can search a place and see their friends' check-ins when they adventure into a new place.

The gap between the reality and the virtual world is being narrowed by the LBSN, which naturally gives rise to people's more concerns on its privacy issues. Currently, users have to provide their location information, such as GPS coordinates, to the LBSN server when they check in at a venue, search places, and store the check-in records on the server, which may be against people's willing and leads to a privacy breach. For example, a user may check in at a hospital and share the doctor reviews with her family and very close friends. She does not want anyone else to know her location and access the information, which expose her health condition. In addition, by obtaining the location information, the server may spam users with unwanted advertisements or learn about users' unpopular political or religious views, which can be inferred from checking in at entertainment clubs or political events. Besides, an attacker may break in the server and obtain those information, making users suffer from more losses. Foursquare has admitted an unauthorized crawling of users'

private check-in records, although they claimed the problem has been solved in a short time [50]. Without any solid privacy guarantee, some people would rather not check in at some venues, if they are worried about their location information being abused. These worries definitely become obstacles to the progress of current LBSNs.

The purpose of this work is to design a framework for current LBSNs to safeguard users' private locations and check-in records. Towards this goal, we face the following challenges. The first one is how to hide a user's check-in location from the server, while making it searchable for her friends. Besides the searchability, an efficient online search is also important in LBSNs. Second, social networks are dynamic in the sense that a user often accepts new friends or revokes old ones. Therefore, the revoked users should not be allowed to access new check-ins and the new friends should be able to retrieve the user's check-in history. Finally, current LBSN client applications are usually running on mobile devices, mostly smartphones with limited computational resources. Therefore, the framework should not put heavy computational tasks on the client side.

Overcoming the above challenges, we propose a privacy preserving framework for LBSNs. On a high level, our framework works as follows. When a user checks in at a venue, she encrypts the check-in record and generates a check-in trapdoor using a pseudo random function. The server converts the trapdoor to all her friends' search trapdoors and pushes the necessary information to her friends. If the user's friends want to search, they generate the search trapdoor and send a search query to the server. The server then returns the search result without any knowledge of which place has been searched. In addition, our framework enables a user to efficiently revoke a friend and add a new friend.

The contributions of this work are as follows.

1. We propose the privacy preserving problem in LBSNs and present the goals that a

privacy preserving LBSN framework should achieve.

2. We propose a framework that hides users' check-in and search locations from the LBSN server. We also design a novel approach which reduces the processing time when users check in at the same venue frequently.

3. Considering the limited computational resources of the mobile devices, we propose a delegatable pseudo random function so that a user can outsource the generation of search trapdoors to the server.

4. In order to revoke a friend, a user needs to update the session key and re-encrypts all the ciphertexts. Since the revocation happens more often in social networks, we propose a hash chain based session key generation scheme so that a user does not need to re-encrypt all the ciphertexts every time the revocation takes place.

4.1 Related Work

In this section, we discuss two areas that are related to our work: location privacy and searchable encryption.

4.1.1 Location Privacy

Prior studies on location privacy mainly focused on traditional location based service, which is not combined with social networks. A large body of works in location privacy are based on the idea of K-anonymity or location cloaking [7, 58, 79]. In this approach, a trusted third party blurs a user's location request by combining several nearby users' location requests together in an area and sending the area to the untrusted server. In the process, users have to fully trust the third party which may become a security weakness of the system. Kalnis *et al.* found there were certain scenarios in which the private location

information of users was leaked to malicious entities [64]. In addition, these approaches employ complex server query processing techniques and entail the transmission of large quantities of intermediate results between users and the trusted third party.

To avoid some shortcomings of cloaking-based techniques, a new class of transformation based approaches were proposed. More recently, Yin *et al.* proposed a framework called SpaceTwist [103] to blind an untrusted location server by incrementally retrieving nearest neighbour based on their ascending distance from a fake point which is different from the user's actual location. In [67], Khoshgozaran and Shahabi proposed an approach to encode the query and the object space using a one-way transformation and evaluating the query in the transformed space to preserve users' location privacy. Their approaches do not rely on a trusted party, which makes the proposed system more practical, but could not be applied in scenarios which require exact results.

Other works proposed to use private information retrieval (PIR) to preserve the location privacy. Ghinita *et al.* used computational PIR to enable private evaluation of the first nearest neighbour queries [53]. The heavy computational overhead of the underlying PIR scheme makes the approach unsuitable for a mobile LBSN. Hengartner proposed an architecture which utilizes PIR and trusted computing to protect user locations from the untrusted server [61]. However, the proposed architecture is not implemented yet. Recently, Khoshgozaran *et al.* proposed a fast PIR based location privacy scheme [68]. The efficiency of their scheme depends on a tamper-resistant trusted hardware, required to be installed close to the server.

4.1.2 Searchable Encryption

In order to make protected check-in venue IDs searchable, we propose to use the idea of searchable encryption. Searchable encryption [8] is widely studied since it was first proposed by Song *et al.* [90]. Since then, progress has been made on its efficiency and security definition formalization. Goh [55], and Chang and Mitzenmacher [29] proposed searchable index schemes, which reduce the search cost proportional to the number of files. In [42], Curtmola *et al.* summarized previous studies and gave a new formal security definition on searchable symmetric encryption. Recently, Wang *et al.* [97] proposed a conjunctive similarity keyword search achieving non-adaptive semantic security defined in [42]. In the public key setting, Boneh *et al.* gave the first construction on searchable encryption, where anyone with the public key can write to the data stored in the server but only the ones with the private key can search on the data [15]. Conjunctive key word search, subset query, and range query over encrypted data [18] were also proposed in the public key setting.

In the studies of searchable encryption, all data files are usually ready when the user uploads them to the server and the search request was usually compared against one user's file collection. However, the LBSN is dynamic in the sense that the check-in record is uploaded one by one, without any fixed time intervals. The search request is compared against all the data items of users followed by the request sender. On top of the searchable encryption, the establishment and revocation of a social relationship are also considered and handled by our framework. These features special to LBSNs differentiate our work from the traditional searchable encryption.

4.2 Problem Formulation

In this section, we present our system model, threat model, and design goals.

4.2.1 System Model and Threat Model

Our framework consists of a LBSN server S, and a set of M users, $\mathscr{U} = \{U_1, \cdots, U_M\}$. In our framework, the relation graph between users is uni-directional, i.e., a user U_i has to *follow* another user U_j in order to see U_j's check-ins. Meanwhile, U_i is *followed* by N_i users, denoted by $\{f_1^i, \cdots, f_{N_i}^i\} \subseteq \mathscr{U}$. When a user U_b requests to join U_i's follower group, it is U_i's responsibility to authenticate U_b and grant her secret information. Hence, we assume that f_a^i is fully trusted by U_i. We also assume the communications between users and the server are secure. Each user U_i has a public/private key pair (pk_{U_i}, sk_{U_i}). The authentication between users and the server is out of the scope of this work.

Each venue has a unique venue ID *vid* assigned by the server. U_i checks in at a venue *vid*, and leaves tips which are stored on the server. When her followers search *vid*, they are able to pull out all of U_i's check-ins at *vid*. During the whole process, the server has no knowledge of *vid* or the contents of the tips.

Throughout the work, we consider an honest but curious (HBC) server, which is used by many privacy preserving schemes [42, 52, 97]. To protect its reputation, the server would honestly follow the protocol specification, but is curious about users' private information. It may keep all messages which it sends to or receives from users, analyze them, and try to get additional information. The server may be broken in by malicious attackers. But we assume a malicious attacker cannot stay in the server for a long period, e.g., a few hours, without being detected. Our framework does not rely on a trusted third party.

4.2.2 Design Goals

In general, a privacy preserving LBSN framework should satisfy the following requirements.

- Location Privacy: The server does not know the location of the user when she checks in at a venue. Also, the server cannot infer the location of the user when she searches a place and tries to pull out the check-in records.
- Check-in Confidentiality: Only the authorized user can access the check-ins belonging to users who are followed by her.
- Forward Security: If a follower is revoked from U_i's follower group, she will no longer be able to access U_i's new check-ins.
- History Retrievability: When U_i accepts U_b as her follower, the new follower should be able to retrieve U_i's check-in history uploaded before and after U_b joins.
- User Side Efficiency: The computation should be efficient on the user side, since the mobile devices used by users are resource limited.

4.3 Technical Preliminaries

In this section, we review the main techniques used in this dissertation.

4.3.1 Pseudo Random Functions

A pseudo random function (PRF) is an efficiently computable function $F : K \times X \rightarrow Y$ where K is the key space, X is called the domain, and Y is called the range [17]. Generally speaking, given a key k, a function F would map an input $x \in X$ to an element $y \in Y$,

such that y looks like being randomly chosen from Y. Formally, we have the following definition.

Definition 4. *A PRF $F : K \times X \rightarrow Y$ is secure if for all efficient adversaries $\mathscr{A}$ the advantage*

$$Adv_F^{prf}(\mathscr{A}) = Pr[Real_F^{\mathscr{A}} \Rightarrow 1] - Pr[Rand_R^{\mathscr{A}} \Rightarrow 1]$$

is a negligible function.

Here, $Real_F^{\mathscr{A}}$ (resp. $Rand_R^{\mathscr{A}}$) denotes a game, in which the adversary interacts with a PRF (resp. real random function) [11]. $Real_F^{\mathscr{A}} \Rightarrow 1$ and $Rand_R^{\mathscr{A}} \Rightarrow 1$ denote that the adversary thinks she is interacting with a real random function. In other words, this definition means that no adversary with polynomial time computational resources can distinguish the output of a PRF from the output of a real random function [11].

In this dissertation, we use two PRFs defined as $F : \mathbb{Z}_q \times \{0,1\}^* \rightarrow \mathbb{G}$ and $F' : \mathbb{Z}_{q'} \times \{0,1\}^* \rightarrow \mathbb{G}'$, where $\mathbb{G}$ and $\mathbb{G}'$ are two multiplicative groups of prime orders q and q', respectively. F is used to generate search trapdoors and F' is used to generate symmetric encryption keys (Sec. 4.4.3). A construction of F is given in [62]: $F(k,x) = H_1(x)^k$ which is secure under the decisional Diffie-Hellman (DDH) assumption in the random oracle model (ROM). Here, $H_1(\cdot)$ is a cryptographic hash function hashing a string to an element in $\mathbb{G}$: $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$. F' can be constructed in a similar way by using a hash function $H_1'(\cdot)$ hashing $\{0,1\}^*$ to an element in the group $\mathbb{G}'$.

4.3.2 Session Key Management Scheme

In our framework, we define a ***session*** to be a time period between two continuous revocations. A ***session key***, which is denoted by *sek*, is a symmetric key for encryption and decryption during a communication session. We assign a session ID *sid* to each session.

The initial session is assigned the session ID $sid = 1$. Each time a member is revoked, U_i updates sid to a new session ID $sid' = sid + 1$ and generates a new session key $sek_{sid'}$. Since a new follower should be able to access U_i's previous check-ins, there is no need to update the session key when a new member joins the follower group. Instead, U_i just sends the current session key to the new follower via a secure communication channel.

We use modified key tree based session key management scheme [35], which uses proxy re-encryption, for our session key management. We use it as a building block in our framework. One advantage of using this scheme is that it can efficiently handle off-line users. Another advantage is that the public bulletin only needs to store the latest public re-encryption keys.

4.4 Construction of Our Framework

In this section, we describe our framework construction. First we give our framework overview, followed by novel designs specially for LBSNs, and then show the details of our framework construction.

4.4.1 Overview of Our Framework

Table 4.1: Main notations

$\mathscr{U} = \{U_1, \cdots, U_M\}; S$	user set; the LBSN server
$\{f_1^i, f_2^i, \cdots, f_{N_i}^i\}$	U_i's follower group
vid; sek_{sid}, sid	venue ID; session key, session ID
k'; k; $\|\|$	check-in key; search key; concatenation
HI; $headerK$	header table index; header table symmetric key
$headerC$; NI	header table ciphertext; node index
$nodeK$; $nodeC$	node symmetric encryption key; node ciphertext
chk	symmetric encryption key for check-ins
PRN'; PRN	check-in trapdoor; search trapdoor

Our framework consists of four protocols: check-in, search, revocation, and join. Table 4.1 shows the main notations.

Check-in: Before uploading the check-in record on *vid*, U_i first encrypts it. Then U_i and S collaboratively generate the search trapdoor for each U_i's follower when necessary. S pushes the index of the check-in ciphertext to every follower, so that the follower can search the check-in record.

Search: U_i picks a venue ID *vid* she is interested in, generates a search trapdoor, and sends it to S. S returns all the check-in ciphertexts uploaded on *vid* by users who are followed by U_i. Using the corresponding decryption keys, U_i decrypts all check-in ciphertexts.

Revocation: U_i revokes members from her follower group and updates the session key.

Join: $f^i_{a'}$ is approved to join U_i's follower group. U_i sends the session key sek_{sid} and *sid* to $f^i_{a'}$. $f^i_{a'}$ sends her search key $k_{f^i_{a'}}$ to U_i, who then computes the public delegation key. Using the delegation key, S pushes the index of U_i's previous check-ins to $f^i_{a'}$.

In our framework, every user U_i is assigned a searchable data structure by the server for searching check-ins. While one can use any data structure that supports search operations in our framework, we choose AVL tree [2] for efficient insertion and search. The AVL tree is constructed by inserting each node using the user's search trapdoor as keyword, and storing the check-in index information on the nodes. U_i's check-in trapdoor PRN'_{U_i} and search trapdoor PRN_{U_i} are computed from $F(k'_{U_i}, vid)$ and $F(k_{U_i}, vid)$, respectively. To save a follower's search time, our framework prepares, in check-in protocol, every U_i's follower's search trapdoor and insert the index of U_i's check-in into her follower's AVL tree according to the search trapdoor. Using the following delegatable PRF technique, we let the server do the preparation task and save the user's computation time.

4.4.2 Delegatable Pseudo Random Function

When U_i checks in at *vid*, she computes $PRN'_{U_i} = F(k'_{U_i}, vid)$ which is then converted to $PRN_{f_a^i} = F(k_{f_a^i}, vid)$ for all f_a^i's $(1 \leq a \leq N_i)$. We propose an approach which delegates the conversion task to S. We call the key for conversion as delegation key *Dkey*. Formally, we design the following probabilistic polynomial time algorithms:

1. GrpGen(1^λ): Given a security parameter λ, the algorithm outputs a multiplicative group $\mathbb{G}$ of a prime order q;
2. KeyGen($\mathbb{G}, q$): On inputs $\mathbb{G}$ and q, the algorithm outputs a key k randomly chosen from $\mathbb{Z}_q$;
3. ComFunc(k, x): On inputs k and x, the algorithm outputs $F(k,x) = H_1(x)^k$;
4. DelKeyGen(k_i, k_j): On inputs k_i and k_j, the algorithm outputs the delegation key $Dkey_{i \to j} = k_j / k_i \mod q$;
5. ReComFunc($F(k_i,x), Dkey_{i \to j}$): On inputs $F(k_i,x)$ and $Dkey_{i \to j}$, the algorithm outputs $F(k_j,x) = (F(k_i,x))^{Dkey_{i \to j}}$. The correctness can be verified by: $F(k_j,x) = (H_1(x)^{k_i})^{k_j/k_i} = (F(k_i,x))^{Dkey_{i \to j}}$.

After receiving a user U_i's check-in trapdoor PRN'_{U_i}, S can compute $PRN_{f_a^i}$ for each f_a^i $(1 \leq a \leq N_i)$, using the public delegation key $Dkey_{U_i \to f_a^i}$. We prove in section 4.5.1 that the conversion result is still secure in the ROM.

4.4.3 Index Structure

Most previous studies on searchable encryption assumed that all data items were ready. However, in the LBSN framework, the data items are dynamically uploaded. Therefore,

traditional searchable encryption approaches do not fit in our scenario. If U_i checks in at the same *vid* frequently, S should frequently update the followers' AVL trees using traditional approaches, which causes a heavy computational overhead.

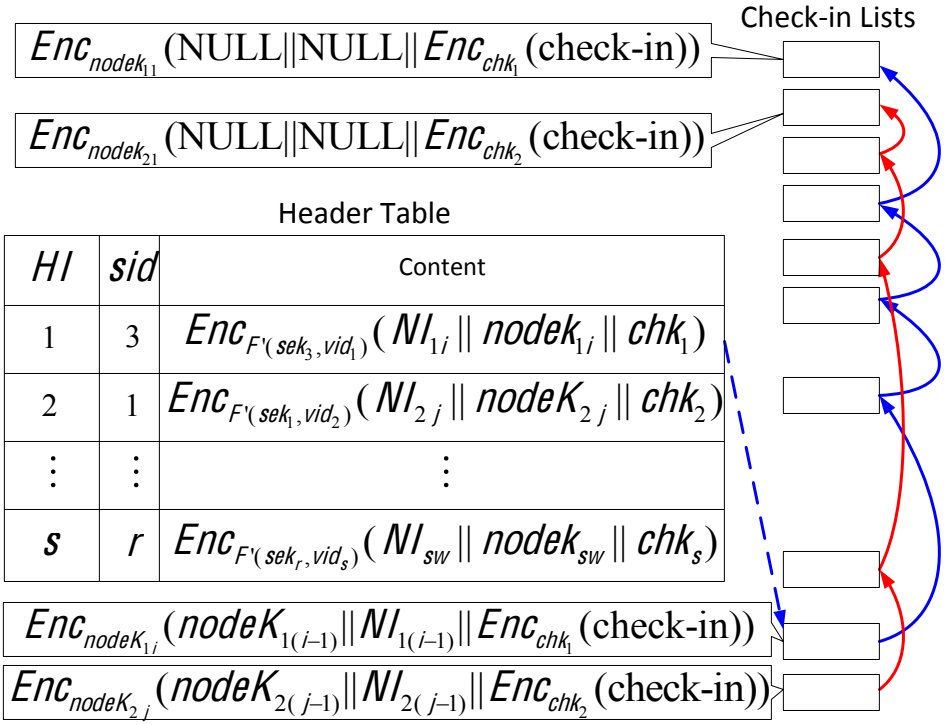


Figure 4.1: Index structure

We propose a novel index structure (Fig. 4.1) for efficiently processing U_i's check-ins. The index structure consists of a header table and check-in lists. The check-in lists are a set of linked lists. For each *vid* that U_i has checked in, there is a unique corresponding list in the check-in lists. Each time U_i checks in at a *vid*, a new node is added to the list corresponding to *vid*. Each node is a ciphertext (denoted as *nodeC*) under a random symmetric key. The node contains, in its plaintext, the encryption key and the index of the next node, and the encryption of the check-in record. Therefore, given the encryption key and the index of the header node, one can decrypt the whole list sequentially. The check-in records in the same list are encrypted using the same symmetric key, so that a user can

decrypt all check-in ciphertexts at one time. We use a header table to associate each *vid* with its check-in list. Each line of the header table stores the header table index HI, the session ID *sid*, and a ciphertext (denoted as $headerC$) of the header node index, the header node encryption key, and the encryption key for the check-in records. The encryption key of $headerC$ is $F'(sek_{sid}, vid)$, where sek_{sid} is the current session key, and *vid* is the venue ID corresponding to the list.

Fig. 4.1 shows an example of the index structure. The nodes linked by blue lines and red lines store the check-in records on two different venues vid_1 and vid_2. From the header node index and encryption key stored in the header table, one can locate and decrypt the corresponding header node (see the dashed blue line in Fig. 4.1). The header node stores the newest check-in and the last node stores the oldest check-in. We note that the value of *sid* in the header table is not sequentially increased. This is because a line i is only updated when the user checks in at vid_i. For example, if vid_1 has never been checked in since session 3, the symmetric key for line 1 is still $F'(sek_3, vid_1)$, even if the current session key is sek_r other than sek_3. We do not store the venue ID in the header table.

U_i locally stores a list mapping the *vid*'s she has checked in to the corresponding header table index. When U_i checks in at a *vid*, she first looks up in her local list to see if it has been checked in before. If it has, she finds the header table index s, retrieves the session ID and the ciphertext $headerC_s$ from her header table, decrypts $headerC_s$, and gets the header node index NI_{sw} and decryption key $nodeK_{sw}$. Then U_i generates a new node, stores NI_{sw} and $nodeK_{sw}$ in it, encrypts it using a new key $nodeK_{s(w+1)}$, asks S to add it in the check-in lists, and gets the node index $NI_{s(w+1)}$ returned by S. Finally, U_i substitutes $nodeK_{sw}$ and NI_{sw} in the header table with $nodeK_{s(w+1)}$ and $NI_{s(w+1)}$, respectively. If *vid* has never been checked in before, U_i increases the header table size by 1 (denote the increased one as s'), and creates a new node $Enc_{nodeK_{s'1}}(NULL||NULL||Enc_{chk_{s'}}(\text{check-in}))$, where $nodeK_{s'1}$

and $chk_{s'}$ are two random keys for encrypting the new node and the check-in record. S adds a new node in U_i's check-in lists, and returns the index $NI_{s'1}$. U_i computes $headerC_{s'} = Enc_{F'(sek_{sid},vid)}(NI_{s'1}||nodeK_{s'1}||chk_{s'})$ and sends $s'||sid||headerC_{s'}$ to S, which adds them to a new line s' in U_i's header table. Here, sek_{sid} is the current session key. Finally, U_i adds vid and the index s' in her local list.

The advantage of the index structure is that S does not need to update N followers' AVL trees upon U_i checking in at an old venue. Another advantage is that if a member f_a^i has been revoked, she cannot access U_i's new check-ins any more. Because each list in the check-in lists is linked from the newest check-in to the oldest check-in, even if f_a^i has the decryption keys for the check-in ciphertexts, she cannot get the decryption keys for new nodes, which store the ciphertexts of new check-in records.

4.4.4 Hash Chain based Session Key Generation Scheme

A LBSN framework should enable the new followers to access all U_i's check-ins and disable the revoked ones to continue accessing her new check-ins. A traditional approach is to re-encrypt the whole header table using a new session key every time a user updates the session key, which brings heavy computational overhead to the user. We propose a hash chain based session key generation scheme, in which the new followers can efficiently compute previous session keys from the current session key, while the revoked ones cannot infer new session keys from old ones. In this scheme, the header table need not to be re-encrypted frequently. We define a re-encryption window of length L. Within the window, a user does not need to re-encrypt the whole header table every time. Given an initial value val and a cryptographic hash function $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$, a user can generate L session keys by computing $H_2(val)$, $H_2(H_2(val))$, $\cdots$, $H_2^L(val)$, where we use $H_2^L(\cdot)$ to denote L compositions of H_2. We use the t-th output $H_2^t(val)$ as the $(L-t+1)$-th session key sek_{L-t+1} and set the

session ID $sid = L - t + 1$. If all L session keys have been used, the user needs to generate a new hash chain and re-encrypts the whole header table using the first new session key.

For a new follower $f^i_{a'}$, she can calculate all the previous session keys only if she gets the current session key and the session ID sid. For example, if $f^i_{a'}$ joins U_i's follower group in session t, the session key she acquires is sek_t. For each previous session l $(1 \leq l \leq t-1)$, she can compute the session key $sek_l = H_2^{t-l}(sek_t)$. If she needs to decrypt line s which contains $sid = r$ and corresponds to vid_s in U_i's header table, she computes $sek_r = H_2^{t-r}(sek_t)$ and the encryption key $headerK_s = F'(sek_r, vid_s)$.

4.4.5 The Construction of Our Framework

In this section, we present the details of our framework.

In the initialization of the framework, we let S generate multiplicative groups $\mathbb{G}$ and $\mathbb{G}'$ of prime orders q and q', respectively, and publish $\mathbb{G}$ and $\mathbb{G}'$. Each user U_i runs the algorithm KeyGen$(\mathbb{G}, q)$ twice to generate her check-in key k'_{U_i} and search key k_{U_i}.

Fig. 4.2 shows the check-in protocol. The user U_i checks in at a venue vid. First, she looks up in her local list to see whether it is the first time to check in at vid. If not, this means that there already exists a check-in list corresponding to vid. U_i only needs to add a new node in the list, and updates the corresponding line in her header table. In order to do this, U_i gets the header table index s from her local list, retrieves $sid'||headerC_s$ from her header table, where sid' is the session ID in which U_i checked in at vid last time. U_i computes $headerK_s$, uses $headerK_s$ to decrypt $headerC_s$, and gets the header node index NI_{sw}, header node key $nodeK_{sw}$, and chK_s. In step 2, U_i computes the node ciphertext $nodeC_{s(w+1)}$ and sends it to S. S returns the new node index $NI_{s(w+1)}$. Upon receiving $NI_{s(w+1)}$, U_i computes the current header table key for line s, $headerK_s = F'(sek_{sid}, vid)$, where sid is the current session ID,

When U_i checks in at a venue *vid*, she looks up in her local list:
if it is not the first time to check in at *vid*:

1. U_i gets the header table index s corresponding to *vid* from her local list, retrieves $sid'||headerC_s$ from her header table, computes $headerK_s = F'(sek_{sid'}, vid)$, decrypts $headerC_s$ using $headerK_s$, and gets $NI_{sw}||nodeK_{sw}||chk_s$;
2. U_i generates a random key $nodeK_{s(w+1)}$, computes $c_{s(w+1)} = Enc_{chK_s}$(check-in) and $nodeC_{s(w+1)}$=$Enc_{nodeK_{s(w+1)}}$ $(NI_{sw}||nodeK_{sw}||c_{s(w+1)})$, and sends $nodeC_{s(w+1)}$ to S;
3. Upon receiving $nodeC_{s(w+1)}$, S adds a new node in U_i's check-in lists and sends the node index $NI_{s(w+1)}$ to U_i;
4. Upon receiving $NI_{s(w+1)}$, U_i computes $headerK_s = F'(sek_{sid}, vid)$ and $headerC_s = Enc_{headerK_s}(NI_{s(w+1)}||nodeK_{s(w+1)}||chk_s)$, and sends $s||sid||headerC_s$ to S;
5. S Updates the content of line s upon receiving $s||sid||headerC_s$;

Otherwise:

6. U_i increases the size of her header table index by 1. The increased size is denoted as s'.
7. U_i generates two random symmetric keys $nodeK_{s'1}$ and $chK_{s'}$, computes $c_{s'1} = Enc_{chK_{s'}}$(check-in) and $nodeC_{s'1}$=$Enc_{nodeK_{s'1}}(NULL||NULL||c_{s'1})$, and sends $nodeC_{s'1}$ to S;
8. Upon receiving $nodeC_{s'1}$, S adds a new node in U_i's check-in lists and sends the node index $NI_{s'1}$ to U_i;
9. U_i computes $headerK_{s'} = F'(sek_{sid}, vid)$, $PRN'_{U_i} = F(k'_{U_i}, vid)$, and $headerC_{s'} = Enc_{headerK_{s'}}(NI_{s'1}||nodeK_{s'1}||chK_{s'})$, sends PRN'_{U_i} and $s'||sid||headerC_{s'}$ to S;
10. S adds a new line $s'||sid||headerC_{s'}$ in U_i's header table, computes $PRN_{f_a^i} = F(k_{f_a^i}, vid) = ReComFunc(F(k'_{U_i}, vid), Dkey_{U_i \to f_a^i})$ $(1 \le a \le N_i)$, and inserts $Enc_{pk_{f_a^i}}(U_i||s')$ into f_a^i's AVL trees using $PRN_{f_a^i}$;
11. U_i adds $vid||s'$ into her local list;

Figure 4.2: Check-in protocol

uses $headerK_s$ to generate the ciphertext $headerC_s$, and sends $s||sid||headerC_s$ to S, which updates the content of line s.

If it is the first time that U_i checks in at *vid*, U_i increases the header table size by 1. We denote the increased size as s'. U_i generates two random keys $nodeK_{s'1}$ and $chK_{s'}$ to encrypt

a new node and the check-in record, computes the ciphertext $nodeC_{s'1}$, and sends $nodeC_{s'1}$ to S. Upon receiving $nodeC_{s'1}$, S adds a new node in U_i's check-in lists, and sends back the node index $NI_{s'1}$. U_i computes the header table symmetric key, $headerK_{s'}$ and the ciphertext $headerC_{s'}$. Then she computes the check-in trapdoor FNR'_{U_i}, sends FNR'_{U_i} and $s'||sid\ ||headerC_{s'}$ to S which adds $s'||sid||headerC_{s'}$ to a new line in U_i's header table. Using FNR'_{U_i} and the public delegation key $Dkey_{U_i \to f_a^i}$, S computes the search trapdoor $PRN_{f_a^i}$ for each follower f_a^i, and inserts $Enc_{pk_{f_a^i}}(U_i||s')$ into f_a^i's AVL tree using $PRN_{f_a^i}$. In the end, U_i adds $vid||s'$ into her local list.

For a *vid* of U_i's interest:

1. U_i computes $PRN_{U_i} = F(k_{U_i}, vid)$, and sends PRN_{U_i} to S;
2. Upon receiving PRN_{U_i}, S searches U_i's AVL tree, and returns $\perp$ if nothing is found;

If any value has been returned, say $Enc_{pk_{U_i}}(U_j||s)$:

3. U_i decrypts $Enc_{pk_{U_i}}(U_j||s)$, and sends $U_j||s$ to S;
4. S retrieves $s||sid||headerC_s$ from U_j's header table, and sends it to U_i;
5. U_i computes $headerK_s = F'(sek_{sid}, vid)$, decrypts $headerC_s$, gets $NI_{sw}||nodeK_{sw}||chK_s$, and sends $NI_{sw}||nodeK_{sw}$ to S;
6. S sequentially decrypt the whole list after decrypting the header node NI_{sw} from U_j's check-in lists, and sends the ciphertexts $\{Enc_{chK_s}(\text{check-in})\}_v$ $(1 \le v \le w)$ to U_i;
7. U_i decrypts $\{Enc_{chK_s}(\text{check-in})\}_v$ $(1 \le v \le w)$ and gets the check-in records on *vid*.

Figure 4.3: Search protocol

Figure 4.3 shows the search protocol. The functionality of the protocol is that U_i picks a *vid* of interest, and generates a search trapdoor PRN_{U_i} on *vid*. S uses the search trapdoor to search the index of check-in records, without revealing the location of U_i. The challenge is that how to let U_i know the venue ID *vid*, since when U_i gets to a new place, she may be unfamiliar with the neighbourhood, let alone all *vid*'s in the neighbourhood.

We provide an offline approach and an online approach for U_i to acquire all *vid*'s that she

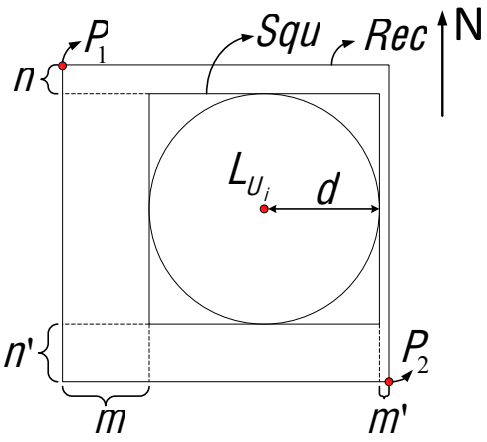


Figure 4.4: Constructing an online search

is interested in. In the offline approach, the user first downloads the map of the intended area into her mobile device. After arriving at the area, she locally finds the venue ID *vid* , computes the search trapdoor, and sends it to the server S which searches U_i's AVL tree. Therefore, S would have no idea about the location of U_i or the *vid* she searches. However, some users may be unwilling or forget to download the map before they go to some place. We propose an alternative online approach. Let L_{U_i} be the location of U_i, let d be the search distance defined by U_i, let *Cir* be the circle centered in L_{U_i} with a radius of d, and let *Squ* be a square circumscribed about *Cir*, as shown in Fig. 4.4. U_i randomly selects m, m', n, and n' from the range $[0, R']$, where R' is defined by the user. The larger R' is, the better protection it provides, but the more bandwidth it consumes. Then U_i extends *Squ* to the east by m, to the west by m', to the north by n, and to the south by n', and sends the locations of the northwest point P_1 and the southeast point P_2 to S. Using the locations of L_{P_1} and L_{P_2}, S can reconstruct a rectangle *Rec*. It retrieves all *vid*'s within the area of *Rec*, and sends *vid*'s back to U_i.

After acquiring all *vid*'s within the area, U_i chooses a *vid* of her interest (see Fig. 4.3), computes the search trapdoor PRN_{U_i}, and sends PRN_{U_i} to S. Upon receiving PRN_{U_i}, S

searches U_i's AVL tree. If nothing is found, S returns $\perp$ to U_i. Otherwise, S sends the search result to U_i. Note that a node of a user's AVL tree may store multiple items. For example, if U_j and $U_{j'}$ both have checked in at the same *vid* and are followed by U_i, the items $U_j||HI$ and $U_{j'}||HI'$ will both be stored in the same node of U_i's AVL tree, the search will return the ciphertexts of $Enc_{pk_{U_i}}(U_j||HI)$ and $Enc_{pk_{U_i}}(U_{j'}||HI')$. In our protocol, for ease of description, we assume one value $Enc_{pk_{U_i}}(U_j||s)$ is returned. U_i decrypts the returned value and sends $U_j||s$ to S. S retrieves $s||sid||headerC_s$ from U_j's header table, and sends it back to U_i. Here, *sid* is the session ID in which U_j checked in at *vid* last time. U_i decrypts $headerC_s$ using $headerK_s$, gets $NI_{sw}||nodeK_{sw}||chK_s$, and sends $NI_{sw}||nodeK_{sw}$ to S. Using NI_{sw} and $nodeK_{sw}$, S decrypts the list corresponding to *vid* in U_j's check-in lists sequentially, and sends back all check-in ciphertexts $\{Enc_{chK_s}(\text{check-in})\}_v$ $(1 \leq v \leq w)$ to U_i. Using chK_s, U_i can decrypt all of the ciphertexts.

U_i wants to revoke f_a^i, she checks the current session ID *sid*:
if $sid = L$**:**

1. U_i randomly chooses a number *val*, computes a new hash chain, sets the session ID $sid = 1$ and the session key $sek_1 = H_2^L(val)$, re-encrypts her header table using sek_1, and distributes sek_1 using the scheme introduced in Section 4.3.2;

Otherwise, if $sid = t < L$**, and the current hash chain initial value is** val'**:**

2. U_i sets the session ID $sid' = t + 1$, computes the new session key, $sek_{sid'} = H_2^{L-t}(val')$, and distributes $sek_{sid'}$ using the scheme introduced in Section 4.3.2.

Figure 4.5: Revocation protocol

Fig. 4.5 shows the revocation protocol. If U_i wants to revoke a follower f_a^i, she checks the current session ID *sid*. If $sid = L$, it means that all keys computed from the hash chain have been used. U_i randomly chooses a number *val*, computes a new hash chain, sets the session ID $sid = 1$ and the session key $sek_1 = H_2^L(val)$. Then U_i re-encrypts her whole header table using sek_1, distributes sek_1 to her followers except f_a^i using the scheme introduced in Section 4.3.2. If the current session ID $sid = t < L$ and the current hash chain

initial value is val', U_i set the new session ID $sid' = t+1$, computes the new session key $sek_{sid'} = H_2^{L-t}(val')$, and distributes $sek_{sid'}$ to her followers except f_a^i as in the previous case.

We note that there would be some idle items in the revoked follower's AVL tree. Although it somewhat affects the usability of our framework, it does not affect the security, because the revoked follower is not able to decrypt U_i's new check-ins. In order to improve this, we let each user select a nonce for each *vid* she has checked in and the server insert the ciphertext of $U_i||HI$ along with the nonce into the follower's AVL trees. Now, when U_i revokes a follower, she sends the nonces to the server. The server can delete the idle items from the revoked follower's AVL tree using the nonce and a copy of the search trapdoor it has computed. Due to space limitation, we leave the technique details to our future work.

$f_{a'}^i$ is approved to join U_i's follower group in session t:

1. U_i sends $sid = t$ and sek_{sid} to $f_{a'}^i$;
2. $f_{a'}^i$ sends her search key $k_{f_{a'}^i}$ to U_i;
3. U_i computes the public delegation key $Dkey_{U_i \to f_{a'}^i} = k_{f_{a'}^i}/k'_{U_i}$, and sends $Dkey_{U_i \to f_{a'}^i}$ to S.
4. Using $Dkey_{U_i \to f_{a'}^i}$ and PRN'_{U_i}, S computes the search trapdoors for all *vid*'s U_i has checked in, and inserts the corresponding ciphertext of $U_i||HI$ into $f_{a'}^i$'s AVL tree.

Figure 4.6: Join protocol

Fig. 4.6 shows the join protocol. If a user $f_{a'}^i$ is approved to join U_i's follower group, U_i sends the current session ID *sid* and session key sek_{sid} to $f_{a'}^i$. Using *sid* and sek_{sid}, $f_{a'}^i$ can compute all previous session keys which is introduced in section 4.4.4. At the same time, U_i receives $f_{a'}^i$'s search key, which is used to compute the delegation key *Dkey*. In step 4, S computes the search trapdoor and inserts U_i's previous check-in indexes into $f_{a'}^i$'s AVL tree. We note that each time after U_i computes the check-in trapdoor PRN'_{U_i}, she sends it to S, which then convert PRN'_{U_i} to U_i's followers' search trapdoors. Therefore, we let S keep

a copy of PRN'_{U_i} and the corresponding header table index HI.

4.5 Analysis of Our Framework

4.5.1 Location Privacy

We first need to guarantee that the server cannot learn any location information from the computation of the delegatable PRF. We have the following theorem.

Theorem 5. *Our delegatable PRF is semantic secure under the DDH assumption in the ROM.*

The idea of the proof is as follows. We only give the division k_j/k_i to the server, which cannot recover either k_i or k_j. It has been proven in [62] that, in the ROM, the output of $H_1(x)^{k_i}$ cannot be distinguished with a real random number as long as k_i is kept secret. Therefore, the calculated number $H_1(x)^{k_j}$ still seems random to the server.

For our framework, some patterns, such as check-in record size or the check-in lists length, are unavoidably to be released. Our framework guarantees that an HBC server cannot learn anything beyond these patterns. The security definition of our framework is the same as the non-adaptive semantic security of a searchable encryption scheme in [42]. We first introduce some auxiliary notions which are used in the security proof.

History H: History is a request sequence $\{(\text{type}, U, \text{content})_p\}_{p=1}^{P}$ starting from the beginning of the protocols, where U is the request user's ID, P is the number of requests in the sequence. The type is check-in, search, join, or revocation. The corresponding content is: 1), the venue ID and check-in record for the check-in type, 2), the venue ID for the search type, 3), the revoked user ID for the revocation type, or 4), the user ID for the join type. History determines the interactions between users and the server.

View $V(H)$: The server can only see an encrypted version of history. Specifically, it includes: the check-in trapdoors of the venue IDs, the encryption of the check-in records, the encrypted head table, the encrypted check-in nodes, the search trapdoors of the requested venue IDs.

Trace $Tr(H)$: Given a history H, trace is the information that our framework releases to the server, including the check-in pattern, search pattern, revocation pattern, and join pattern. For each request in H, the type, the request user ID, and the request order in the sequence are known to the server. The revocation pattern and the join pattern contain the user ID that is to be revoked or followed, respectively. Therefore, the server knows the whole relation graph. The check-in pattern contains the size of the check-in record, the length of a user U's every check-in list, and the size of the header table. For each check-in request, the check-in pattern also contains the index of updated or added line of the header table and the set of the user's followers whose AVL trees need to be updated, $\{(td, f^U)_a\}_{a=1}^{N_U}$, where td_a denotes the new inserted or updated node in f_a^U's AVL tree. The set could be empty if the user checks in at an old venue. The search pattern includes the nodes found on the AVL tree.

Theorem 6. *Our framework is semantic secure under the DDH assumption in the ROM, given the condition that the symmetric encryption algorithm is semantic secure.*

Proof. We give the sketch of the proof as follows. Our task is to construct a simulator $\mathscr{S}$ such that given only $Tr(H)$, it can simulate a view V' indistinguishable from the server's view in the real protocol execution with a probability $1-\varepsilon(\lambda)$, where λ is a system security parameter and $\varepsilon(\lambda)$ is a negligible function in λ.

Setup: Given a multiplicative group $\mathbb{G}$ with a prime order q, for each user U, $\mathscr{S}$ randomly selects two numbers $\hat{k}', \hat{k}$ from $\mathbb{Z}_q$ as the user's check-in key and search key. $\mathscr{S}$ also gener-

ates a random value as the user's hash chain initial value. Having all the check-in requests, $\mathscr{S}$ constructs a graph G as follows. Suppose there are h check-in requests, $\mathscr{S}$ constructs h nodes with the label (i,U_i) on the i-th node. U_i is the user ID in i-th check-in request. For the i-th check-in request, $\mathscr{S}$ is given the updated information $\mathscr{F}_i = \{(td,f)_a\}_{a=1}^{N_{U_i}}$. $\mathscr{S}$ connects every two nodes (i,U_i) and (j,U_j) if $\mathscr{F}_i \cap \mathscr{F}_j \neq \emptyset$. Then for each connected component, $\mathscr{S}$ selects a number r' randomly from $\mathbb{G}$ and assigns it to the component. This construction can be done in polynomial time by using depth-first search to find the connected components.

Now we construct the simulator for each request.

Join and **Revocation:** The simulator $\mathscr{S}$ interacts with the server as in the real Join/Revocation protocol by using the corresponding initial value, check-in key, and search key. We note that the delegation key now is calculated as $Dkey_{U_i \to U_j} = \hat{k}_{U_j}/\hat{k}'_{U_i}$ in the join protocol. The server will not distinguish the simulated join and revocation protocols because the hash chain initial value, the check-in key, and the search key are also randomly selected by each user.

Check-in: $\mathscr{S}$ first finds the connected component containing this check-in request in G, gets the assigned random number r', and calculates $(r')^{\hat{k}'_U}$ as the check-in trapdoor. Here, $\hat{k}'_U$ is the request user's check-in key simulated by $\mathscr{S}$. Then $\mathscr{S}$ chooses $\hat{c} \leftarrow_R \{0,1\}^{s'}$ as the check-in ciphertext, where s' is the size of the check-in record in the request obtained from $Tr(H)$. Using the simulated check-in trapdoor and ciphertext, $\mathscr{S}$ interacts with the server as in the real check-in protocol. We remark that, if two users, U_i, U_j, check in at the same venue in two requests and they share at least one follower, then the two requests are in the same connected component and the server will generate the same search trapdoor for each shared follower f'_a due to the equation $((r')^{\hat{k}'_{U_i}})^{\hat{k}_{f'_a}/\hat{k}'_{U_i}} = ((r')^{\hat{k}'_{U_j}})^{\hat{k}_{f'_a}/\hat{k}'_{U_j}}$. We claim that the server cannot distinguish the simulated check-in instance and the real one with

a non-negligible probability. It is because that the semantic security of the PRF and the symmetric encryption algorithm guarantee that the server cannot distinguish real random strings with the generated trapdoor and the ciphertext with a non-negligible probability.

Search a venue *vid*: $\mathscr{S}$ knows the request user U and the searched node td, if (td, U) is not empty. the simulator locates a connected component in G which has (td, U) in one node's updated follower set. Then the simulator uses r^{k_U} as the search trapdoor, where r is the number assigned to the component. If no node is found in the trace, r will be a random number other than the assigned ones. Using this trapdoor, $\mathscr{S}$ interacts with the server as in the real search protocol. Due to the semantic security of the PRF, the server cannot distinguish the views with a non-negligible probability. And the correctness of the search is guaranteed by the simulator set up and the simulated check-in protocol. □

4.5.2 Other Attacks

In our framework, the header table and the nodes on the AVL tree are encrypted. The check-in list nodes and the link structure are also encrypted. If an attacker breaks in the server, she cannot decrypt any of the items due to the semantic security of the encryption algorithms and the PRF used in our framework. If an attacker stays in the server for a long period, she will be able to see the increasing of users' check-in lists. However, hacking into a system and staying for a long period is usually impossible in the real world. We remark that even in this worst case, the user's exact location information cannot be obtained because it is randomized before it has been uploaded to the server.

A revoked user can only access the check-in records that were uploaded before she was revoked. Because the check-in list is linked from the newest check-in to oldest check-in, even if the revoked user has the key for the check-in records, she cannot decrypt the nodes

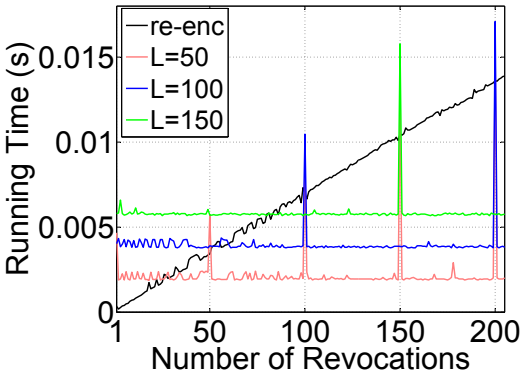
storing new check-ins. Note that the new added lines in the header table are encrypted with new session keys. Hence, the revoked user cannot decrypt them to get the head nodes of the check-in lists.

The items stored on U_i's AVL tree are all encrypted under her public key. Hence, attackers cannot get the user's friend IDs stored on the tree even if they obtain the whole tree. In addition, if U_j is followed by U_i and gets U_i's AVL tree, she cannot infer other user IDs stored on U_i's tree even if U_j can generate U_i's search trapdoors. Therefore, the location privacy of U_i's friends is preserved against unauthorized users and malicious attackers.

4.6 Evaluation

We made a proof-of-concept implementation of our framework to evaluate its performance. The server side simulation was carried out on a laptop with Intel Core 2 Duo 2.53GHz, 4GB memory, and 250GHz hard driver. The client side simulation was carried out on a Motorola Droid 1, which has a 550MHz ARM A8 processor, 256MB memory, a 16GB SD card, and the Android 2.2 OS. In order to speed up the execution, we coded our client application in Android native code and cross compiled it using ndk-r7c tool chain. Since Android does not provide any cryptographic library in native library, we ported Crypo++ library [41] to Android and embedded it in our implementation. The underlying group $\mathbb{G}$ was a 160 bits elliptic curve group, which achieves 80 bits security level and is enough for common security demands. For all hash functions used in the scheme, we used SHA1 with 160 bits output. We used AES with 128 bits key size for the symmetric key encryption and elliptic curve ElGamal as the public key encryption algorithm. In our implementation, we used the same PRF for F' and F. We truncated the 128 least significant bits of the output of F' to match with the key size used in AES.

Fig. 4.7a shows the running time of the hash chain based session key generation scheme



(a) Hash chain vs. re-encryption

L	Time (s)
1000	0.046
2000	0.089
3000	0.126
4000	0.171
4500	0.192
5000	0.21

(b) Compuation time

Figure 4.7: Performance of the hash chain

compared with traditional re-encryption approach. In this experiment, the user checked in at a new venue every two time units and revoked a follower at a random time. The revocation time interval obeys Poisson distribution with a mean value 10.

In Fig. 4.7a, when the window size $L = 50$, after about 30 revocations, our scheme outperformed the traditional re-encryption one, except that when all keys of a hash chain were used, a new hash chain needed to be generated and the header table should be re-encrypted. In Fig. 4.7a, when L was smaller, it took less time to compute the keys on the chain. However, the re-encryption frequency was higher than that with a larger L. Hence, a real implementation should consider the trade off between the re-encryption frequency and the larger computation time. In fact, the time for computing a hash chain was very fast in our experiments. We listed the computation time of hash chains with different lengths in Fig. 4.7b and could see that the computation of a hash chain with 1000 session keys only took 0.046s.

One of our contributions is to use delegatable PRF to save computation time on the client device. In Fig. 4.8, we show the computation time of the server and the client when they used delegatable PRF, compared with the time when delegatable PRF was not used. We

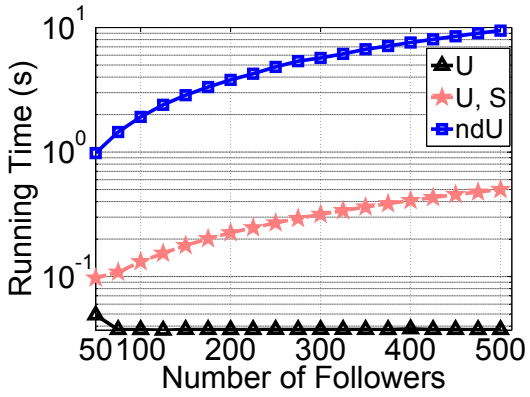
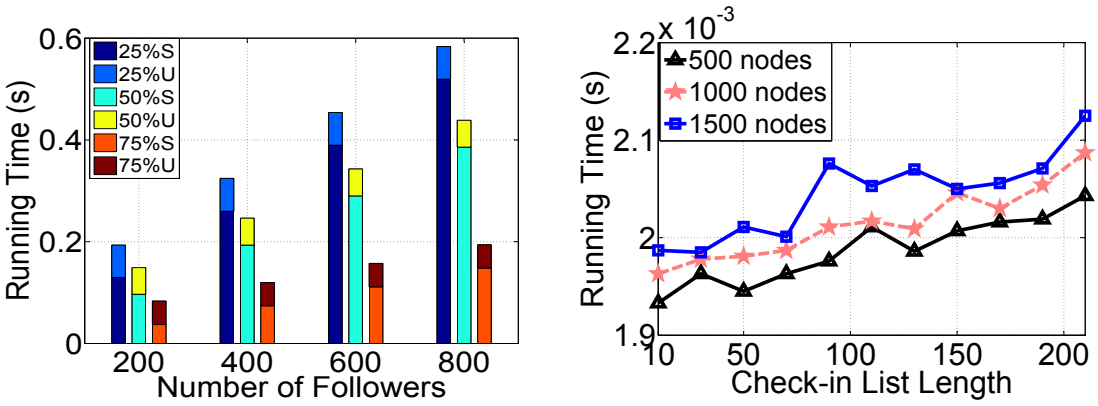


Figure 4.8: Delegatable PRF

increased the number of followers from 50 to 500 and recorded the running time. The lines with stars and rectangles represent the total running time of the client and the server, and the running time of the client itself, respectively, when delegatable PRF was used. The remaining line represents the running time of the client without using delegatable PRF. Without using delegatable PRF, the client needs to compute all the follower's search trapdoors and sends them to the server. Therefore, there is no computational cost on the server side. In Fig. 4.8, we can see that the client side computational overhead are dramatically reduced using delegatable PRF.



(a) Running time of the check-in protocol

(b) Running time of the search protocol

Figure 4.9: Performance of the framework

Fig. 4.9a and Fig. 4.9b show the running time of the check-in protocol and search protocol, respectively. In our simulation, we did not consider the communication time. Instead, we only counted the computation time needed on the client side and the server side. In the search protocol, the user only needs to compute a pseudo random number, decrypt one line in each searched user's header table, and decrypt the check-in ciphertexts. Due to the small size of a line in the header table, the user's execution time is dominated by decrypting the check-in ciphertexts, which is actually independent of the framework and determined by the encryption algorithm.

The running time on the server side is more interesting in the check-in protocol. We considered three kinds of users according to their check-in habits: 1) users often check in at new venues; 2) users check in at new and old venues with equal probability; 3) users often check in at old venues. In our simulation, we assumed the three type of users checking in at an old venue with probabilities 25%, 50%, and 75%, respectively. In Fig. 4.9a, we show the total running time of the server and user with different number of followers. We could see that for users who frequently check in at old venues, our proposed framework could greatly reduce both the server and user's computation time. Also, for users who usually check in at new venues, our framework could reduce the client's computational overhead.

Without loss of generality, we generated AVL trees with 500, 1000, and 1500 nodes, respectively. Using a search trapdoor, the server searched the AVL tree, and used the returned value to decrypt the check-in list. We show the time the server used to search the AVL tree and decrypt the check-in list in Fig. 4.9b. The server's computation time increased with the increase of the AVL tree size and the check-in list length.

4.7 Conclusions

In this dissertation, we have proposed an efficient location privacy preserving framework for LBSNs. We have proposed novel designs to reduce the computational overhead of users and the server. We have theoretically proved that the location privacy of users has been preserved and demonstrated that our framework is secure against several attacks. Extensive simulations have been carried out to demonstrated the efficiency of our framework.

CHAPTER 5

Location Privacy in Location Based Reminders

Reminder applications are becoming a crucial applications in mobile devices. The reminder applications help users to memorize something important to do in the future and remind user at specific occasion. Traditional reminder applications are time based reminders, i.e., the user enters the reminder message and set up the reminder time, which could be one time or periodically repeated times, the application displays the reminder message on the screen at the setting time and alert the user. Recent years have witnessed a rapid development of mobile devices. Most mobile devices are now equipped with accurate localization capabilities, based for example on GPS receivers, access points, or triangulation with nearby base stations [87].

Thanks to the localization sensors, more and more reminder applications on smart devices now have an additional functionality, which allows users to mark a location during the setting of a reminder. We call this location a *reminder location*. The smart device will alert a user when the user's current location is close to the reminder location. The extent of closeness is pre-set by the user and is called the *reminder distance*. For example, a user can add a location based reminder such that the smart device can remind her to buy groceries when she is near Walmart, or to refill medicines when she passes by CVS or Walgreens.

One approach to implement the location base reminder system is based on a single smart device. All the reminder messages and the associated reminder locations are stored locally on a single device. The smart device periodically queries the localization sensor to detect the user's current location. If the user is near the reminder location, the smart device

will alert her. This approach is simple and secure in the sense that it does not leave the user's location information out of the smart device. However, using this method, it is hard to synchronize reminders on one device with other smart devices. Since more and more users tend to have more than one smart device, synchronization is an important issue. For example, a user may want to add a reminder using her tablet while she is browsing Internet at home, but she may want this reminder to automatically appear on her smartphone when she is out.

Due to the development of cloud service, many companies now use the cloud to synchronize personal data, including reminders, among a user's multiple smart devices. A typical example is iCloud, which can synchronize multiple smart devices with the same account [89]. A user can add a reminder on her iPhone, which then uploads an encrypted copy to iCloud automatically. iCloud pushes the reminder to the user's other iOS devices, such as iPad or iTouch. In this way, the user can get the reminder from whichever device she brings with her. The disadvantage of this approach is that multiple copies exist among a user's different smart devices, which causes efficiency issues during synchronization if the size of a reminder message is large, since current reminder messages may contain not only texts but also multimedia, such as audios [78] or images [1]. If a user has many multimedia reminders and goes out with a device that has not been synchronized for a period of time, the synchronization latency may be large and the user may even miss some reminders. At the same time, copying reminders to every device unnecessarily cost a user's cellular data usage and local storage of the smart devices.

It is desirable to keep only one copy of reminders on the cloud server, which pushes the reminder to a user when she is near the reminder location. We call this kind of system *cloud-assisted location based reminder system*, which is illustrated in Figure 5.1. As shown in Figure 5.1, the *reminder area* with respect to a reminder location is a disc centered at the

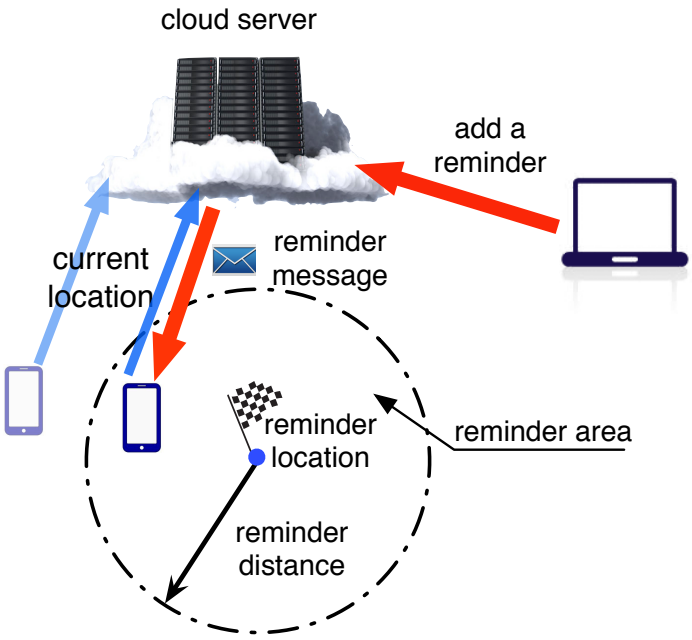


Figure 5.1: Cloud Assisted Location Based Reminder System

reminder location with a radius of the reminder distance. A user can add a reminder using one of her smart devices. The reminder is then uploaded to the cloud server, which synchronizes the user's multiple smart devices, including her smartphone. When she is away from home with her smartphone, the smartphone periodically sends the current location to the cloud server. If the user appears in any reminder area, the corresponding reminder message is pushed to the user. In addition to saving local storage space, responsiveness is another advantage of the cloud-assisted location based reminder system. A user can access her reminders on any of her smart devices if they share the same account without synchronization and only the triggered reminders are pushed from the cloud server. If the smart device is not privately owned, the user would not need to worry about extra reminder messages besides necessary ones pushing to the device.

A big challenge of such a system that we need to overcome is to protect users' location privacy and message confidentiality. If there is no privacy and confidentiality guarantee,

users may not want to use the system, worrying about their privacy being exposed. For example, a user may not want to expose the reminder message and the reminder location if she wants to remind herself to refill medicines when she passes by a pharmacy, which might expose her health condition. Users may also not want to expose their current location, which leads to illegal tracking.

The goal of this work is to construct a secure cloud-assisted location based reminder system, which protects users' location privacy and the confidentiality of reminder messages. The system hides a user's reminder messages, reminder locations, and current location from the cloud server. However, the system still enables the cloud server to decide whether a user's current location is within any reminder area. Specifically, the system divides the surface of earth into small squares (forming a grid), represents a reminder area by a set of squares, and associates them with the encrypted reminder message using searchable symmetric encryption. The smart device represents the current area in the same way and lets the cloud server search nearby reminder locations without exposing any location information and reminder message. We propose a cross based area representation approach and a bar based area representation approach, respectively, to implement the system. The cross based area representation approach enables us to reduce an inequality testing, i.e. distance comparison, to an equality testing, i.e. finding set intersections. The implemention of the bar based area representation approach enables the user to accurately compute the current distance. It also greatly reduces the storage overhead on the cloud server side.

The contributions of this work are as follows.

- We propose a secure cloud-assisted location based reminder system. which is based on cross based area representation. We use two different approaches, cross based approach and improved bar based approach, to implement the systme. We prove

the security of the system constructed based cross based approach under the decisional Diffie-Hellman (DDH) assumption [92] in the random oracle model (ROM) and demonstrate its efficiency through simulations on a Motorola Droid phone. The implemention of the bar based approach saves a large storage space for the cloud server.

- We propose a novel method to represent the reminder area, which reduces the time complexity for checking whether a user is in a particular reminder area. The representation is easy to construct and lightweight in space overhead.

- Based on our novel representation approach, we propose to use searchable symmetric encryption to construct a secure cloud-assisted location based reminder system. Considering the computational limitations, we propose to use a lightweight pseudo random function [63], which is proved to be secure in the random oracle model.

5.1 Related Work

In this section, we review two security areas that are related to our work: location privacy and searchable encryption. We also differentiate our work with prior works.

5.1.1 Location Privacy

Prior studies on location privacy mainly focused on privacy issues in the traditional location based service (LBS), in which a user sends the LBS server a location query and the server returns the local information around the query location. An extensive literature on location privacy utilizes the idea of K-anonymity or location cloaking [7, 58, 79]. In this approach, a user's query is relayed by a trusted third party to the LBS server. The trusted third party does not send queries to the LBS server until K different user queries from the same

area are collected. In this way, the LBS server cannot tell which user queries a specific location. The assumption that the third party is fully trusted by users became a security weakness of this approach. Kalnis *et al.* [64] pointed out that there exists certain scenarios in which K-anonymity approach leaks private location information to malicious entities. In addition, these approaches employ complex server query processing techniques and entail the transmission of large quantities of intermediate results between users and the trusted third party.

To overcome the disadvantages of cloaking-based techniques, a new class of location privacy protection approaches were proposed, which are known as transformation based approaches. More recently, Yin *et al.* proposed a framework called SpaceTwist [103] to deceive an untrusted location server by incrementally querying nearest neighbor based on their ascending distance from a fake point which is different from the user's actual location. In [67], Khoshgozaran and Shahabi proposed to use an one-way transformation to encode the location query and the object space. The query is evaluated in the transformed space such that the users' location privacy is preserved. Their approaches do not rely on a trusted party, but may have errors in scenarios that require exact result.

Some other works were proposed to preserve the location privacy by using private information retrieval (PIR) [36]. Hengartner proposed an architecture which utilizes PIR and trusted computing to protect user location privacy [61]. However, the proposed architecture is not implemented yet. Ghinita *et al.* used computational PIR to enable private evaluation of the nearest neighbour queries [53]. However, the heavy computational overhead of the underlying PIR scheme makes the approach unsuitable for a smart device. A fast PIR based location privacy scheme was proposed by Khoshgozaran *et al.* [68]. Their scheme requires a tamper-resistant trusted hardware installed close to the server, which makes the scheme less practical. Recently, Albanese *et al.* [4] proposed a novel approach to detect

an attacker's location by using information of detected malicious nodes when the attacker tries to locate a victim in the Mobile Ad Hoc Networks.

Different from the above works, we study the location privacy issue in a cloud-assisted location based reminder system, in which a user not only provides a location query, i.e., the current location, but also sets a reminder location and both locations are oblivious to the cloud server. In addition, the server is able to test whether the current location appears in a reminder area even if it does not know the current locations and the reminder locations.

5.1.2 Searchable Encryption

In order to enable the square IDs to be searchable, we propose to use the idea of searchable symmetric encryption. To the best of our knowledge, searchable encryption was first proposed by Song *et al.* [90] and has received extensive attentions [8] . The earlier progress has been made on its efficiency and security definition formalization. The searchable index schemes proposed by Goh [55], and Chang and Mitzenmacher [29] reduce the search cost to one proportional to the number of files. In [42], Curtmola *et al.* summarized previous studies and gave a new formal security definition on searchable symmetric encryption. Recently, a conjunctive similarity keyword search over encrypted data is proposed by Wang *et al.* [97]. Their scheme is proved to be non-adaptive semantic secure defined in [42]. Regarding the public key setting, Boneh *et al.* gave the first construction on searchable asymmetric encryption, where anyone with the public key can write to the data stored in the server but only the ones with the private key can search on the data [15]. Conjunctive keyword search, subset query, and range query over encrypted data [18] were also proposed in the public key setting.

5.2 Problem Formulation

In this section, we present the system model, threat model, and design goals.

5.2.1 System Model and Threat Model

Our system consists of a cloud server s and multiple users. Since all the interactions are between a user and the cloud server, we use u to denote the user intersecting with s. The user u may have multiple smart devices (e.g., tablet, smartphone, etc.) with the same account. Since our system does not need data synchronization, the system interactions are independent of devices and only specific to a user account. Therefore, we use a smart device to denote any one being used by the user. When user u adds a reminder to her smart device, u enters a reminder time or mark a reminder location, and enters a reminder message, which could be text, audio, or image. We focus on protecting users' location privacy in this dissertation and will omit reminder time setting in the system construction. Extending our work to include a reminder time can be simply achieved by storing the reminder time in ciphertext on the cloud server.

When a user u enrolls in the system for the first time, she sets a reminder distance d_u and a parameter a_u. The reminder area is a disc centered around the reminder location with the radius d_u. According to the reminder setting, if a user u enters or leaves the reminder area, the smart phone should be able to retrieve the reminder message from the cloud server and alert the user. The user may have multiple smart devices and multiple reminders on the cloud server . She may take any of her devices when she is out. Therefore, the cloud server should be able to synchronize reminders between different devices and test whether any smart device is in the reminder area or not, and send corresponding reminder to the smart device when it is in the reminder area.

We consider an honest but curious cloud server, which is consistent with most privacy preserving systems [42, 65, 97]. This adversary model is also acceptable in our application scenario. A cloud server will follow the protocol specification and not intentionally destroy users' reminder service in order to maintain its business reputation. However, the server may be curious to learn users' reminder contents and location privacy. To this end, the server may collect any information beyond the protocol specification and perform arbitrary computation on the information. It may keep records of all messages transmitted between the user and the cloud server, analyze them, and try to infer the user's location privacy. Our system does not rely on a trusted third party so as to simplify the system architecture and enhance the system performance. We assume that the user's smart device is clean with regard to malicious and vulnerable applications [83]. In this dissertation, we focus on protecting a user's location privacy against an untrusted server. Privacy breach via malicious applications is a topic studied by malicious application detection [74–76, 83] and is out of the scope of this work.

5.2.2 Design Goals

In the following, we summarize the goals that a secure cloud-assisted location based reminder system should achieve.

- Location privacy: The cloud server should not know a user's reminder locations and her current location;
- Confidentiality: The server should not be able to learn about a user's reminder message.
- Workability: The cloud server should be able to check whether a user's current location is in a reminder area and send the associated reminder message to the user.

- Efficiency: The system should be efficient in computation, data usage, and server space consumption.
- Responsiveness: A user's device should not need to synchronize to catch up reminders from the cloud server, even when the device has been offline for a long time.

5.3 Technical Preliminaries

In this section, we review the main technical preliminaries used in this dissertation.

5.3.1 Pseudo Random Function

Informally speaking, a pseudo random function (PRF) is a cryptographic primitive whose output looks like a random number. A PRF is an efficient computable function $F : K \times X \rightarrow Y$, where K is the key space, X is the domain, and Y is the range [17]. Given a key k, a function F maps an input $x \in X$ to an element $y \in Y$. Formally, we have the following definition.

Definition 1. *A PRF $F : K \times X \rightarrow Y$ is secure if for all probabilistic polynomial-time adversaries $\mathscr{A}$, the adversary's advantage*

$$Adv_F^{prf}(\mathscr{A}) = Pr[Real_F^{\mathscr{A}} \Rightarrow 1] - Pr[Rand_R^{\mathscr{A}} \Rightarrow 1]$$

is a negligible function.

Here, $Real_F^{\mathscr{A}}$ (resp. $Rand_R^{\mathscr{A}}$) denotes a game, in which the adversary interacts with a PRF (resp. real random function) [11]. The symbol $\Rightarrow 1$ denotes that the adversary thinks she is interacting with a real random function. In other words, this definition states that

no probabilistic polynomial time adversary can distinguish the output of a PRF from the output of a real random function with a non-negligible probability [11].

Considering the limited computational resources on smart devices, we use a construction of PRF given in [63], which is based on the cryptographic hash function. In this dissertation, we use two PRFs defined as $F : \mathbb{Z}_q \times \{0,1\}^* \rightarrow \mathbb{G}$ and $F' : \mathbb{Z}_{q'} \times \{0,1\}^* \rightarrow \mathbb{G}'$, where $\mathbb{G}$ and $\mathbb{G}'$ are two multiplicative groups of prime orders q and q', respectively. F is used to generate search trapdoors and F' is used to generate symmetric encryption keys. The construction of F is : $F(k,x) = H_1(x)^k$, where $H_1(\cdot)$ is a cryptographic hash function hashing an input to an element in $\mathbb{G}$: $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$. This pseudo random function is secure under the DDH assumption in the ROM. F' can be constructed in a similar way by using a hash function $H_1'(\cdot)$ which hashes $\{0,1\}^*$ to an element in group $\mathbb{G}'$.

5.3.2 Hashing onto A Group

A hash function usually hashes an input to a fixed length of binary string, such as SHA-1. In the construction of PRFs, we utilize a special type of hash functions, H_1 and H_1', which hash the input to an element in groups $\mathbb{G}$ and $\mathbb{G}'$, respectively. Here, we only introduce the construction of H_1. H_1' is constructed in a similar way. H_1 is constructed from a common cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{l_1}$, where $l_1 = \log q$ and q is the order of $\mathbb{G}$. For an input x, if the output of H is an element t in $\mathbb{Z}_q$, the desired hash value can be obtained by calculating g^t, where g is a generator of $\mathbb{G}$. The challenge is that the output of H may exceed the range of $\mathbb{Z}_q$. A standard and simple solution is to re-hash $H(x||1)$ and verify whether the new hash value is in $\mathbb{Z}_q$ or not, where $x||1$ denotes the concatenation of x with 1. The re-hash is repeated k times ($(H(x||k)$ for k-th re-hash) until the output is in $\mathbb{Z}_q$. Since the probability that the output exceeds $\mathbb{Z}_q$ is less than $1/2$, the failure probability decreases exponentially with repeat times. In the simulation, we select $k = 9$ and obtain an

overall failure probability less than 0.1%.

5.3.3 Searchable Symmetric Encryption

Searchable symmetric encryption (SSE) allows a user to encrypt her data in such a way that the data can still be searched by the cloud server using predefined keywords while the cloud server cannot learn about the keywords [65].

SSE was initially proposed to search over encrypted documents on an untrusted database. A construction of SSE can be found in [97] and we summarize its basic scheme here. Given a document D, a set of keywords $id_1, id_2, \cdots id_N$ associated with it, and two pseudo random functions f and g, we can build an SSE scheme as follows.

- SetUp: The user generates two keys k_1 and k_2 for the pseudo random functions, f and g, respectively, and a symmetric encryption key k for D.
- TrapdoorGen: For each given keyword id_i $(1 \leq i \leq N)$, the user generates a trapdoor $T_i = f(k_1, id_i)$.
- BuildIndex: For each keyword id_i, the user generates an index as follows. She takes the key k_2, id_i and the trapdoor T_i as inputs, and constructs the index $(T_i, Enc_{g(k_2,id_i)}(k))$. The user uploads $Enc_k(D)$ associated with $\{(T_i, Enc_{g(k_2,id_i)}(k))\}_{i=1}^{N}$ to the server.
- Search: For a given keyword id, the user generates $(f(k_1, id),\ g(k_2, id))$ and sends it to the server. The server looks up $f(k_1, id)$ against trapdoors in the database. If any match is found, the associated $Enc_k(D)$ is decrypted. Here, the key k is obtained by decrypting $Enc_{g(k_2,id)}(k)$. The server decrypts $Enc_k(D)$ and returns D to the user.

If there are multiple documents, each one is processed as above and uploaded to the server.

We implement SSE in the system constructed based on the cross based approach . The user encrypts her reminder messages and uploads the encrypted messages along with a reminder location to the cloud server. The keyword in our case is the reminder location or its representations. Different from traditional SSE, we do not want the server to learn the message content. Therefore, the user does not upload $g(k_2, id)$ to the server. The server returns $Enc_k(D)$ and $Enc_{g(k_2,id_i)}(k)$ to the user. The reminder message decryption is then performed locally on the smart device. The details of our construction can be found in Section 5.6.2.

5.4 Tessellation on the Surface of the Earth

To represent a proximity area, Narayanan *et al.* [80] proposed to tessellate the earth surface by hexagons. To deal with the curved earth surface, they divided the earth surface into strips by latitude, one strip per degree of latitude, and tessellated each strip separately. Since the curvature of the earth surface within a single strip can be ignored, the small cells can thus be viewed as approximately the same everywhere. Following the same idea, we tessellate the earth surface by squares, instead of hexagons. The tessellation forms a grid on the surface of earth. We note that the reminder area is usually small and the curvature of the earth surface passing through the area is too small to ignore. Let G_u denote the tessellation on the plane, which is defined by user u and consists of squares with the side length a_u. Each square has a unique identifier and we use $id_{(x,y)}$ to denote the square identifier that the location point (x, y) locates.

5.5 Toward Private Location Search

In the desired secure cloud-assisted location based reminder system, a user u stores reminder locations on the untrusted server and wants to retrieve the reminder content when

she appears within the reminder area. We note that the problem studied in this dissertation is different from the private proximity testing problem [80], in which two parties know their own locations and try to hide them from each other. In our problem, one party, the cloud server, only provides storage and search services and cannot access the information that belongs to the user. For such a blind search service, one possible construction is to use homomorphic cryptographic primitives [19] to construct a protocol such that a user provides fully homomorphic encryptions of the reminder location, the reminder distance, and the current location, respectively; the cloud server blindly calculates the distance between the two locations and compares with the reminder distance using a secure comparison protocol [43]. This approach is not suitable for mobile devices because homomorphic encryption schemes are usually computationally expensive.

We propose to use searchable symmetric encryption [90] to construct a secure cloud-assisted location based reminder system. As introduced previously, the searchable symmetric encryption allows a user to associate an encrypted content with a set of keywords. Since a reminder area is a disc which is composed of uncountable number of points, the biggest challenge here is to find an efficient approach to represent the disc using a finite number of location points.

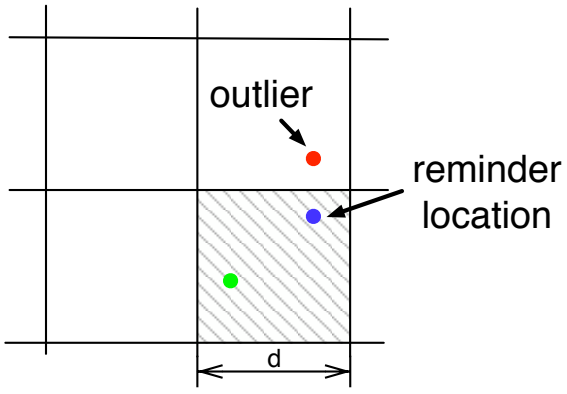


Figure 5.2: Using Grid to Represent the Reminder Area

Inspired by the work of Narayanan *et al.* [80], a straightforward idea of the area representation is to use a square in the tessellation that is introduced previously to represent a reminder area, as shown in Figure 5.2. When a user wants to mark a reminder location, such as the blue point (x, y) in Figure 5.2, she uses the square identifier $id_{(x,y)}$ as a keyword associated with an encrypted reminder message and stores them in the cloud server. The user will then use the square identifier of her current location as a keyword to search on the cloud server. If the two identifiers are identical, e.g., as shown in Figure 5.2, the blue point and the green point have the same square identifier, the reminder message will be retrieved, and pushed to the user's smart device. This approach is spatially and computationally lightweight, but suffers from a large inaccuracy. When the reminder location and the current location are close but in different squares, such as the red point and the blue point in Figure 5.2, even if their distance is smaller than d_u, this approach cannot remind the user when she is at the red point, because the current square is different from the square where the reminder location is.

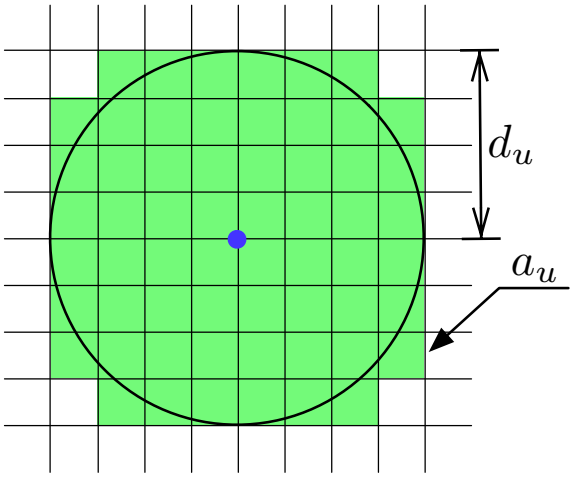


Figure 5.3: Using Small Squares to Approximate the Reminder Area

Although it fails in some cases, the previous approach shows that it is a good way for representing the reminder area using the squares on the tessellation. This observation leads

to the second approach. Again, we divide the earth surface into squares, the side length of which is smaller than d_u, as shown in Figure 5.3. The idea is to use small squares to approximate the disc-shaped reminder area. On the grid, we draw a disc centered at the point (x,y) with radius of d_u. Every square that is fully or partially covered by the disc is a valid square, as shown in the green squares in Figure 5.3. When a user marks the point (x,y), she stores all valid square identifiers and the associated reminder message in the cloud server. Every time, the user searches the cloud server using the square identifier of her current location. If she is at any green area (see Figure 5.3), the cloud server will find a matching square identifier in the storage and the corresponding reminder message will be pushed to the user's screen. The problem of this approach is that it consumes a lot of cloud server storage space and user's cellular data during uploading trapdoors of the square identifiers.

5.6 Cross based Approach

In this section, we describe the system constructed based on the cross based area representation approach.

5.6.1 Cross Based Area Representation

We try to find an area representation method that reduces the space consumption of the second approach while still keeping the inaccuracy small. Our idea is to use an equal-armed cross to represent a reminder area, as shown in Figure 5.4. The cross consists of multiple small squares in the tessellation and is centered at the square where the interested location (a reminder location or a current location) locates. Each arm of the cross contains $\lceil d_u/a_u \rceil$ squares. Suppose there is a point (x,y), e.g., the blue point in Figure 5.4, we let $id^{i,*}_{(x,y)}$, $i = \pm 1, \pm 2, \cdots, \pm(\lceil d_u/a_u \rceil)$ denote i-th square identifier to the east/west of square

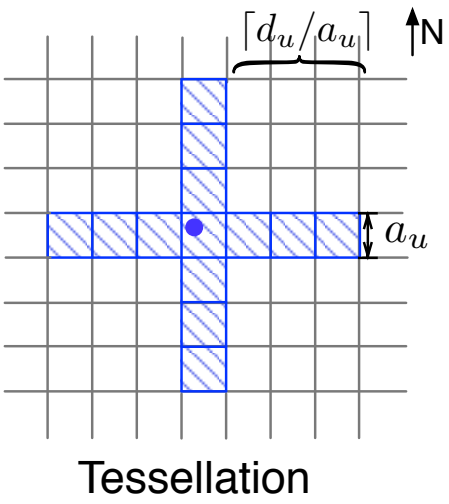


Figure 5.4: Using Cross to Represent An Area

$id_{(x,y)}$ (negative numbers denote the west direction), and $id^{*,i}_{(x,y)}$ $i = \pm 1, \pm 2, \cdots, \pm(\lceil d_u/a_u \rceil)$ denote the i-th square identifier to the north/south of square $id_{(x,y)}$ (negative numbers denote the south direction). All the squares in the cross, i.e., shaded squares in Figure 5.4, are called *valid* squares with respect to (x,y).

Procedure 1: CreateCross(x, y, d_u, a_u)

Data: location (x,y), reminder distance d_u, square side length a_u
Result: valid square identifier set $\mathscr{ID}_{(x,y)}$

1 $\mathscr{ID}_{(x,y)} \leftarrow \emptyset$;
2 $\mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID} \cup \{id_{(x,y)}\}$;
3 **for** $i \leftarrow 1$ *to* $\lceil d_u/a_u \rceil$ **do**
4 $\quad \mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID}_{(x,y)} \cup \{id^{*,i}_{(x,y)}, id^{*,-i}_{(x,y)}, id^{i,*}_{(x,y)}, id^{-i,*}_{(x,y)}\}$;
5 **end**

Given a location (x,y), a distance d_u, and a square side length a_u, Procedure 1 shows how to construct a valid square identifier set $\mathscr{ID}_{(x,y)}$. Using the valid square identifier set, we can test whether the distance between the two points (x',y') and (x,y) is smaller than d_u or not. We calculate two valid square identifier sets $\mathscr{ID}_{(x',y')}$ and $\mathscr{ID}_{(x,y)}$ with respect to (x',y') and (x,y), respectively, and test whether they have intersections. If they do not

have any intersection, it indicates that their distance is longer than d_u, as shown in Figure 5.5a. In Figure 5.5, we use the blue point and the red point to denote (x,y) and (x',y'), respectively. If they have intersections, we can further estimate the distance between the two locations using the intersected squares. If there are two interactions on two different directions, as shown in Figure 5.5d, the user knows that the distance between the two locations is no longer than the longest distance between any two points in the two squares. Therefore, this distance can be used to bound the exact distance between the two locations. If the intersected squares are on the same line, the user can also estimate the distance. An example is shown in Figure 5.5b, there is only one intersected square and the user knows that the distance between the two locations is no longer than $2d_u - a_u$. If the number of interacted squares is more than $\lceil d_u/a_u \rceil + 1$, e.g. Figure. 5.5c, the distance between the two locations is definitely smaller than d_u.

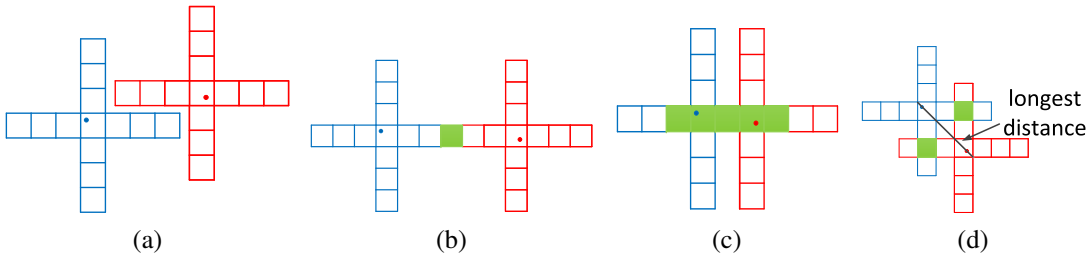


Figure 5.5: Four Cases of Cross Intersection

The original location testing is an inequality testing, in which the user needs to compute the distance between the two locations and compare it with the reminder distance. Our novel representation reduces this inequality testing to an equality testing, in which users test if there are intersections between the valid square identifier sets of the two locations. This reduction allows us to do location testing even when the reminder location and the user's current location is transformed by a one-way trapdoor function.

5.6.2 Constructions

This approach consists of three protocols: user enrollment protocol, location marking protocol, and location search protocol.

Function: A user gets enrolled in the system.

1. The user sends request to the cloud server s.
2. s sets up a new user account and assigns a user ID, e.g. u, to the user.
3. User u sets up her reminder distance d_u and square side length a_u. d_u and a_u are securely stored on the user's local storage. u also generates two pseudo random function keys k_1, k_2, a symmetric encryption key k, and keeps them secret in the local storage.

Figure 5.6: User Enrollment

The user enrollment protocol is shown in Figure 5.6. When a user is enrolled in the system, she selects her reminder distance d_u and square side length a_u. A smaller square side length gives smaller inaccuracy but consumes more server storage space and larger uploading data volume. In Section 5.6.3 we will discuss the selection of a_u. We note that a user may have multiple smart devices. If the user wants to bind one more device to her account, she needs to have the keys k_1 and k_2 stored in the device either. The user can transfer the keys from a smart device which is already bound to her account to the newly added device through a secure out-of-band channel, such as near field communication (NFC) channel or blue tooth.

Figure 5.7 presents the location marking protocol. Our system needs a semantic secure symmetric encryption scheme, in which $Enc_k(\cdot)$ and $Dec_k(\cdot)$ are the encryption and decryption algorithm under the key k, respectively. In the protocol, a user constructs the valid square identifier set with respect to the reminder location. The user encrypts the reminder message with a random key and then generates an index set for the reminder location. In each index tuple $t_{id} = <f(k_1, id), Enc_{g(k_2,id)}(k)>$, the first element is a pseudo random

Function: User u marks location (x, y) as a reminder location with a reminder message m.

1. u randomly selects a symmetric encryption key k and encrypts the reminder message as $C = Enc_k(m)$.
2. u gets the valid square identifier set $\mathscr{ID}_{(x,y)} \leftarrow$CreateCross ($x$, y, d_u, a_u). For each $id \in \mathscr{ID}_{(x,y)}$, u calculates an index tuple $t_{id} = <f(k_1, id), Enc_{g(k_2,id)}(k)>$ and puts t_{id} into an index set $\mathscr{T}$. Finally, user u shuffles the elements in $\mathscr{T}$ and submits $<\mathscr{T}, C>$ to the cloud server s
3. Cloud server s maintains a sorted index set $\mathscr{IND}_u$ for each user u. The index set is sorted by the first trapdoor value $f(k_1, id)$ in the index tuple t_{id}. Upon receiving $<\mathscr{T}, C>$ from u, s stores C, obtains a reference r to C, and updates $\mathscr{IND}_u$ as $\mathscr{IND}_u = \mathscr{IND}_u \cup \{<b, c, r> | <b, c> \in \mathscr{T}\}$

Figure 5.7: Location Marking

number generated from the square identifier id, and the second one is a ciphertext using another pseudo random number derived from id as the encryption key. The first element in the tuple is used for search. If a pseudo random number $f(k_1, id')$ matches $f(k_1, id)$, it is the case that $id' = id$. The user can use the second pseudo random function to derive the key k for decrypting $Enc_{g(k_2,id)}(k)$. The user submits the index set and the ciphertext of the reminder message to the cloud server, which merges the submission to the user's $\mathscr{IND}_u$.

After marking locations in her reminder, a user periodically checks whether her current location is within any reminder area. This process is specified by the location search protocol in our system, which is presented in Figure 5.8.

Figure 5.8 shows the location search protocol. Generally speaking, in the location search protocol, user u checks whether the cross centered at the current location intersects with any cross stored on the cloud server and retrieves the corresponding reminder messages if the condition in Step 3 is satisfied. Specifically, if the two crosses intersect at a square id, the trapdoor $ind_{id} = f(k_1, id)$ calculated in the location marking protocol and the location search protocol are identical. Therefore, if any non-empty $\mathscr{IND}$ is returned by s in Step

Function: Check whether user u's current location (x, y) is within any reminder area.

1. u gets the valid square identifier set $\mathscr{ID}_{(x,y)} \leftarrow$ CreateCross (x, y, d_u, a_u). For each $id \in \mathscr{ID}_{(x,y)}$, u calculates trapdoor $ind_{id} = f(k_1, id)$ and puts it in set $\mathscr{I}$. u shuffles set $\mathscr{I}$ and submits it to the cloud server s. u also keeps all calculated $<ind_{id}, id>$'s in a table T.

2. Upon receiving $\mathscr{I}$ from u, cloud server s searches the user's index set $\mathscr{I}\mathscr{N}\mathscr{D}_u$ against each element in $\mathscr{I}$ and returns $\mathscr{I}\mathscr{N}\mathscr{T} = \{<b,c,r> | b \in \mathscr{I} \wedge <b,c,r> \in \mathscr{I}\mathscr{N}\mathscr{D}_u\}$ to the user.

3. Upon receiving $\mathscr{I}\mathscr{N}\mathscr{T}$, the user u constructs a request set $\mathscr{R}$ as follows:

   - If a reference r appears for no less than $\lceil d_u/a_u \rceil + 1$ times in $\mathscr{I}\mathscr{N}\mathscr{T}$, add it in $\mathscr{R}$;
   - If a reference r appears for less than $\lceil d_u/a_u \rceil + 1$ times in $\mathscr{I}\mathscr{N}\mathscr{T}$, find the corresponding intersection squares. If they are on the same line, drop r and continue to check the next available reference. Otherwise, assume the two square identifiers are $id^{i,*}_{(x,y)}$ and $id^{*,j}_{(x,y)}$. If ${a_u}^2 \cdot (|i|+1)^2 + {a_u}^2 \cdot (|j|+1)^2 \leq {d_u}^2$, then u adds r in $\mathscr{R}$.

   u sends $\mathscr{R}$ to s.

4. Upon receiving a request set $\mathscr{R}$ from user u, s sends a ciphertext set $\mathscr{C} = \{<C,r> | r \in \mathscr{R}\}$ to the user.

5. Upon receiving $\mathscr{C}$ from the server, the user decrypts the ciphertexts. For each $<C_i, r_i> \in \mathscr{C}$, u finds a square identifier id from table T such that $<b,c,r>_j \in \mathscr{I}\mathscr{N}\mathscr{T}$, $r_i = r_j$, and $ind_{id} = b_j$. User u decrypts C_i using the decryption key $k_i = Dec_{g(k_2,id)}(c_j)$ to obtain the reminder message m_i.

6. After obtaining message m_i, user u may want to remove it from the cloud server. User u sends a delete request and the corresponding reference r_i to the cloud server s. The cloud server s removes C_i and all elements containing r_i in $\mathscr{I}\mathscr{N}\mathscr{D}_u$.

Figure 5.8: Location Search

2, u knows not only that the cross centered at the current location intersects with the cross centered at a certain reminder location but also the exact intersected squares by looking up the trapdoors in table T. In Step 3, the user filters out the out-of-range reminder message

references, because it is possible that the distance between the two locations is longer than d_u even if their crosses are intersected (see Figure 5.5b and Figure 5.5d). For the remaining references, the user retrieves the encrypted reminder messages and decrypts them to obtain the plaintexts. The key point here is that the symmetric key for encrypting the reminder message is stored in the ciphertexts encrypted by $g(k_2, id)$, where k_2 is known to the user and *id* is any square identifier in the cross, including the identifier of the intersected square. Since the user has the intersected square identifier *id*, she can derive the key $g(k_2, id)$ and then obtain the symmetric key to decrypt the ciphertext of the reminder message.

5.6.3 Late Reminding

There are two types of errors in a location based reminder system, *false reminding* and *late reminding*. A false reminding happens if a location point outside a reminder area is recognized as an one in the reminder area. A late reminding happens if the current location is already inside a reminder area but the system does not remind the user. We note that a false reminding is more unacceptable than a late reminding in our application scenario.

As shown in Figure 5.9, our system calculates the longest possible distance d' between the current location point and the reminder location and reminds the user when $d' \leq d_u$. Therefore, our system does not have the false reminding error but may cause late reminding. The latency of the late reminding is dependent on the distance inaccuracy when we try to use the longest distance between two squares to approximate the real distance between the two locations. Assume the reminder location, the blue point in Figure 5.9, is (x,y) and the intersected square identifiers are $id^{i,*}_{(x,y)}$ and $id^{*,j}_{(x,y)}$, the distance between the reminder location and the current location could be any distance between $a_u\sqrt{(N_h-2)^2+(N_v-2)^2}$ and $a_u\sqrt{N_h^2+N_v^2}$, where N_h and N_v are respectively the number of horizontal squares and vertical squares to the cross center, e.g., $N_h = |i|+1$ and

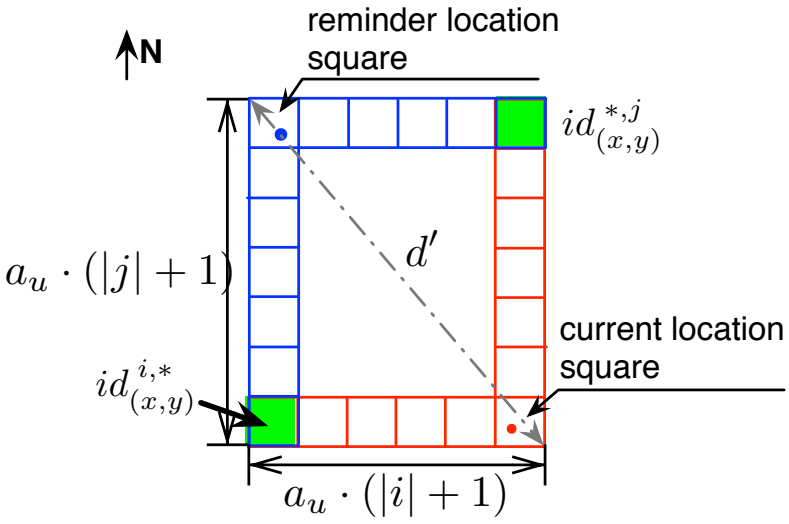


Figure 5.9: Distance Calculation Between Current Location Square and Reminder Location Square

$N_v = |j|+1$ in Figure 5.9. Hence, the distance inaccuracy ε is between 0 and $E(N_h,N_v) = a_u\sqrt{N_h^2+N_v^2} - a_u\sqrt{(N_h-2)^2+(N_v-2)^2}$. The function $E(N_h,N_v)$ is maximized when $N_h = N_v$ and the maximum value is $2\sqrt{2}a_u$. In other words, using our proposed area representation method, the distance inaccuracy range is $0 \leq \varepsilon \leq 2\sqrt{2}a_u$. As pointed out previously, a smaller a_u results in a smaller inaccuracy but larger uploading data and storage space, which is $O(4\lceil d_u/a_u \rceil + 1)$. A user can select an appropriate a_u meeting her accuracy requirement and the server may charge a user according to her storage usage.

5.6.4 Discussions

In order to reduce the inequality testing to equality testing, our system uses an approximate distance to estimate the distance between the reminder location and the user's current location. The difference between the real distance and the approximate distance is called *inaccuracy*, which is dependent on a_u. A smaller a_u gives smaller inaccuracy but incurs larger uploading data and occupies more space on the cloud server. We let the cloud sever

maintain a sorted index set for a user by the first trapdoor. This is to speed up the search process, in which the server can use binary search to find a given trapdoor. Therefore, the time complexity for the search performance is $O(N_c \log(N_c N_l))$, where N_c is the number of squares in a cross and N_l is the number of marked locations. $N_c N_l$ is the number of entries that are already in the user's index set and each search looks up N_c trapdoors in the set. The cloud server may use a balanced binary search tree or a B+ tree to maintain a user's index set. A straightforward improvement on the space and time complexity is to use Bloomier filter [34] to store the index set, which can reduce the search time to $O(N_c)$.

Our idea that represents an area by squares in the tessellation on the earth surface is inspired by the work of Narayanan *et al.* [80]. We did not construct our system based on the area representation approach proposed in [80], because there exists a distance gap between the neighbourhood circle and the non-neighbourhood circle in their approach. If the current location falls into the gap, the state of the current location is not determined, since it can be recognized as either nearby or not nearby depending on which part of the gap the current location is at.

We tessellate the surface of earth strip by strip. As discussed in [80], this type of area representations has a small error probability at the strip boundary, because the squares on either side of the boundary do not line up.

Users may not have the geolocation of the place when they want to add a reminder. A solution is to download the address-to-geolocation lookup table to the local storage before hand and transfer the address to the geolocation offline. An alternative approach is to lookup the geolocation on demand via the public map API, such as Google map API. However, an on-demand lookup request should contain some other nearby addresses to hide a user's real reminder location.

5.7 Security Analysis

In this section, we adapt our security definition proposed in [42] and prove the security of our protocol in the presence of a *non-adaptive* adversary, i.e., the cloud server in our system. Different from an earlier work done by Goh [54], the security definition in [42] also protects the query keywords from being inferred by an adversary, which is important in our work. A non-adaptive adversary is an adversary who is not allowed to form the location search request to the simulator in the proof, based on any previous index tuple of reminder location or trapdoor of the searching location. We note that this is acceptable in our application scenario since the location search request is only formed by users themselves.

The interaction between a user u and the cloud server s is determined by a collection of reminder messages and locations $\mathscr{M} = \{<m_i,(x_i,y_i)>\}_{i=1}^{n}$ and a sequence of locations, $\mathbf{L} = [(x'_1,y'_1),\cdots,(x'_q,y'_q)]$, that u wants to search for. An instantiation of such an interaction is called a q**-query history**, $\mathbf{H} = (\mathscr{M},\mathbf{L})$.

We use $\mathscr{F}_{(x,y)} = \{f(k_1,id)|id \in \mathscr{I}\mathscr{D}_{(x,y)}\}$ to denote the trapdoors of the valid squares with respect to (x,y). Given a q-query history $\mathbf{H}$, its **search pattern** is a symmetric matrix $\sigma(\mathbf{H})$ such that for $1 \leq i,j \leq q$, the element $\sigma_{ij} = \mathscr{F}_{(x'_i,y'_i)} \cap \mathscr{F}_{(x'_j,y'_j)}$ if $i \neq j$, and $\emptyset$ otherwise. The **access pattern** consists of a tuple $\tau(\mathbf{H}) = (\mathscr{M}(x'_1,y'_1),\cdots,\mathscr{M}(x'_q,y'_q))$, where $\mathscr{M}(x'_i,y'_i) \subseteq \{1,2\cdots,n\}$ denotes the index of the reminder messages (in set $\mathscr{M}$) whose reminder area contains (x'_i,y'_i), and a $n \times q$ matrix $\psi(\mathbf{H})$ such that for $1 \leq i \leq n$, $1 \leq j \leq q$, $\psi_{ij} = \mathscr{F}_{(x_i,y_i)} \cap \mathscr{F}_{(x_j,y_j)}$. The **marking pattern** is a symmetric matrix $\gamma(\mathbf{H})$ such that for $1 \leq i,j \leq n$, $\gamma_{ij} = \mathscr{F}_{(x_i,y_i)} \cap \mathscr{F}_{(x_j,y_j)}$ if $i \neq j$, and $\emptyset$ otherwise.

A **trace** of a history captures the information that we are willing to expose to the cloud server during a protocol execution. Given a history $\mathbf{H}$, the trace is a tuple $\omega(\mathbf{H}) = (|C_1|,\cdots,|C_n|, \sigma(\mathbf{H}),\tau(\mathbf{H}), \psi(\mathbf{H}),\gamma(\mathbf{H}))$, where $|C_i|$ is the size of the encryption of m_i. A trace also

contains some common information, such as the trapdoor set size $|\mathscr{T}|$ and key size $|k|$. Matrices $\sigma(\mathbf{H}), \psi(\mathbf{H})$, and $\gamma(\mathbf{H})$ leak the intersection pattern between any two locations to the cloud sever. We argue that this is acceptable in our scenario. Although the server may know the frequent marking or search trapdoors, the server cannot know the exact location in any way. In addition, the square sequence is randomly shuffled before submitted to the server. The server cannot infer any relative direction information with respect to any pair of locations.

A **view** captures what the cloud server can see in the system execution. In particular, given a history $\mathbf{H}$, the view $\pi(\mathbf{H})$ contains the index and the ciphertexts of the reminder messages, $\mathscr{C} = \{C_i\}_{i=1}^n$, the corresponding index set $\mathbb{T} = \{\mathscr{T}_i\}_{i=1}^n$ (see Figure 5.7), and the trapdoors of the searching locations $\mathbb{I} = [\mathscr{I}_1, \cdots, \mathscr{I}_q]$. $\pi(\mathbf{H})$ also contains the matched reminder message index and the index that the user requests to retrieve.

According to [42], the non-adaptive semantic security of a SSE system is defined as:

Definition 2. *A SSE system is secure in the sense of non-adaptive semantic security if there exists a polynomial-time simulator, being provided the trace of a given history, is able to simulate an adversary's view such that no probabilistic polynomial-time (PPT) distinguisher can distinguish the view in a real system execution from the simulated one with a non-negligible probability.*

In other words, the definition states that the cloud server could not extract any additional information (of a user) beyond the one we are willing to give it, i.e., the trace of a history in our case. The security of our system is presented in the following theorem. Our system is secure in the random oracle model because the pseudo random functions used in our system, i.e. f and g, are secure under DDH assumption in random oracle model [63]. The reason we use such pseudo random functions is due to its light-weight computational

overhead, which is significant to the mobile devices.

Theorem 1. *Our location based reminder system is non-adaptively semantic secure under the DDH assumption in the random oracle model.*

Proof: We focus our proof on location marking and location search protocol and omit the user enrollment protocol, because a user does not expose any private information in the enrollment protocol.

Throughout the proof, the adversary $\mathscr{A}$ is the cloud server in our model. To prove security, we construct a simulator $\mathscr{S}$ for any q-query $\mathbf{H}$ and any $q \in \mathbb{N}$ such that, given a trace $\omega(\mathbf{H})$, it can simulate a view $\pi^*(\mathbf{H})$ that is indistinguishable from the adversary's real view $\pi(\mathbf{H})$. By "indistinguishable", we mean that no PPT algorithm can distinguish the two distributions with a probability larger than $\varepsilon(\kappa)$, where $\varepsilon(\kappa)$ is a negligible function in κ. Here, κ is a system security parameter used in the construction of the pseudo random function and symmetric encryption system. For example, when κ is larger, we need to use a larger underlying group for pseudo random function and a longer key length for symmetric encryption system.

Case 1: If $q = 0$, $\mathscr{S}$ constructs $\pi^*(H) = \{\mathscr{C}^*, \mathbb{T}^*\}$ as follows. For $C_i^* \in \mathscr{C}^*$ $1 \leq i \leq n$, $C_i^* \leftarrow_R \{0,1\}^{|C_i|}$, where $\leftarrow_R$ denotes "randomly select from". For each $\mathscr{T}_i^* \in \mathbb{T}^*$, $1 \leq i \leq n$, $\mathscr{S}$ constructs $\mathscr{T}_i^* = \{h_j^*, e_j^*\}_{j=1}^{|\mathscr{T}|}$, where $e_j^* \leftarrow_R \{0,1\}^{|k|}$ and h_j^* is generated as follows. $\mathscr{S}$ computes $\bar{\mathscr{F}} = \bigcup_{1 \leq i,j \leq n} \gamma_{ij}$ and constructs a map Tab for every element $h \in \bar{\mathscr{F}}$ such that $\mathsf{Tab}[h] \leftarrow_R \{0,1\}^{\kappa}$. To generate the h^*'s in $\mathscr{T}_i^*$, $1 \leq i \leq n$, $\mathscr{S}$ checks $\bar{\mathscr{F}}_i = \bigcup_{1 \leq j \leq n} \gamma_{ij}$ and assigns $\mathsf{Tab}[h]$, $h \in \bar{\mathscr{F}}_i$, to the first $|\bar{\mathscr{F}}_i|$ entries of h^*'s. For each of the remaining h^*'s, $\mathscr{S}$ randomly picks an element from $\{0,1\}^{\kappa}$. Finally, $\mathscr{S}$ randomly shuffles $\mathscr{T}_i^*$.

Due to the semantic security of the selected symmetric encryption system, C_i^* and e_j^* are respectively indistinguishable from $C_i = Enc_k(m_i)$ and $Enc_{g(k_2,id_j)}(k_j)$ by a PPT distinguisher

. Due to the pseudo randomness of function f, h_j^* and $f(k_1, id_j)$ are indistinguishable.

Case 2: If $q \geq 1$, $\mathscr{S}$ computes a set union of all elements in three matrices, $\sigma(\mathbf{H}), \psi(\mathbf{H})$, and $\gamma(\mathbf{H})$. Let set $\bar{\mathscr{F}}$ denote the resulting union set. Similar to the construction in the previous case, $\mathscr{S}$ constructs a map Tab such that $\mathsf{Tab}[h] \leftarrow_R \{0,1\}^\kappa$, for all $h \in \bar{\mathscr{F}}$. The construction of $\mathscr{C}^*$ is the same as the one in case $q = 0$. For $\mathscr{T}_i^* \in \mathbb{T}^*$, $1 \leq i \leq n$, $\mathscr{T}_i^* = \{h_j^*, e_j^*\}_{j=1}^{|\mathscr{T}|}$, where $e_j^* \leftarrow_R \{0,1\}^{|k|}$ and h_j^* is generated as follows. $\mathscr{S}$ checks $\bar{\mathscr{F}}_i = (\bigcup_{1 \leq j \leq n} \gamma_{ij}) \cup (\bigcup_{1 \leq j \leq q} \psi_{ij})$ and assigns $\mathsf{Tab}[h]$, $h \in \bar{\mathscr{F}}_i$, to the first $|\bar{\mathscr{F}}_i|$ entries of h^* in $\mathscr{T}_i^*$. For each of the rest entries, $\mathscr{S}$ randomly picks an element from $\{0,1\}^\kappa$. $\mathscr{S}$ then randomly shuffles $\mathscr{T}_i^*$. To construct each $\mathscr{I}_i^*$ in $\mathbb{I}^*$, $1 \leq i \leq q$, $\mathscr{S}$ checks $\bar{\mathscr{F}}_i' = (\bigcup_{1 \leq j \leq q} \sigma_{ij}) \cup (\bigcup_{1 \leq j \leq n} \psi_{ji})$ and assigns $\mathsf{Tab}[h]$, $h \in \bar{\mathscr{F}}_i'$, to available entries in $\mathscr{I}_i$. The rest entries are random elements from $\{0,1\}^\kappa$. $\mathscr{S}$ then randomly shuffles $\mathscr{I}_i$. When $\tau(\mathbf{H})_i$ is not empty, $1 \leq i \leq q$, $\mathscr{S}$ constructs a request set $\mathscr{R}_i^*$ (see Figure 5.8) by using $\tau(\mathbf{H})_i$.

Due to the semantic security of the symmetric encryption system, C_i^* and e_j^* are indistinguishable from $C_i = Enc_k(m_i)$ and $Enc_{g(k_2, id_j)}(k_j)$, respectively. Due to the pseudo randomness of function f, $\mathbb{T}^*$ and $\mathbb{I}^*$ are indistinguishable from $\mathbb{T}$ and $\mathbb{I}$, respectively.

Simulator $\mathscr{S}$ uses the simulated information to interact with $\mathscr{A}$ as per the protocol specifications. The correctness of the system execution can be straightforwardly verified and omitted here. □

5.8 Bar based Approach

In this section, we first describe the bar based area representation approach. Next, we introduce to use bloom filter for private location matching and present the details of system constructions.

5.8.1 Bar Based Area Representation

Our idea is to use a horizontal bar to represent the reminder area, and use a vertical bar to represent the user's current location, as shown in Figure 5.10. The bar is divided into $2(\lceil d_u/a_u \rceil)+1$ squares in the tessellation and is centered at the square where the interested location (reminder location or current location) locates. Given a location point (x,y), we use $id^{i,*}_{(x,y)}$, $i = \pm 1, \pm 2, \cdots, \pm(\lceil d_u/a_u \rceil)$ to denote i-th square identifier to the east/west (negative numbers denote the west direction) of square $id_{(x,y)}$. and $id^{*,i}_{(x,y)}$ $i = \pm 1, \pm 2, \cdots, \pm(\lceil d_u/a_u \rceil)$ to denote the i-th square identifier to the north/south (negative numbers denote the south direction) of square $id_{(x,y)}$. All the squares in the cross, i.e., shaded squares in Figure 5.10, are called *valid* squares with respect to (x,y).

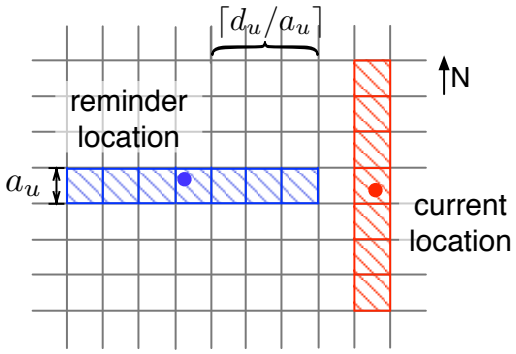


Figure 5.10: Using Bar to Represent An Area

Procedure 2: CreateHorizontalBar(x, y)

Data: location (x,y)
Result: $\mathscr{ID}_{(x,y)}$, the set of square identifiers on the west-east bar centered at (x,y)

1 $\mathscr{ID}_{(x,y)} \leftarrow \emptyset$;
2 $\mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID} \cup \{id_{(x,y)}\}$;
3 **for** $i \leftarrow 1$ *to* $\lceil d_u/a_u \rceil$ **do**
4 $\quad \mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID}_{(x,y)} \cup \{id^{i,*}_{(x,y)}, id^{-i,*}_{(x,y)}\}$;
5 **end**
6 **return** $\mathscr{ID}_{x,y}$

Procedure 3: CreateVerticalBar(x, y)

Data: location (x, y)
Result: $\mathscr{ID}_{(x,y)}$, the set of square identifiers on the north-south bar

$\mathscr{ID}_{(x,y)} \leftarrow \emptyset$;
$\mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID} \cup \{id_{(x,y)}\}$;
for $i \leftarrow 1$ *to* $\lceil d_u/a_u \rceil$ **do**
    $\mathscr{ID}_{(x,y)} \leftarrow \mathscr{ID}_{(x,y)} \cup \{id^{*,i}_{(x,y)}, id^{*,-i}_{(x,y)}\}$;
end
return $\mathscr{ID}_{x,y}$

We use two different procedures to create valid square identifier sets $\mathscr{ID}_{(x,y)}$ regarding the reminder location and the user's current location. Given a location point (x, y), a distance d_u, and square side length a_u, we use Procedure 2 if the point (x, y) is the reminder location point, otherwise we use Procedure 3.

Using valid square identifier sets, we can test whether the distance between the two points (x', y') and (x, y) is smaller than d_u or not. We calculate two valid square identifier sets $\mathscr{ID}_{(x',y')}$ and $\mathscr{ID}_{(x,y)}$ with respect to (x', y') and (x, y), respectively, and test whether they have intersections. If they do not have intersections, it means that the user's current location is not in the reminder area. If they have intersections, the user retrieves the reminder location point from the cloud server and calculates the distance between the reminder location and the current location. In Figure 5.11, we use the blue point and the red point to denote the reminder point (x, y) and the user's current location (x', y'), respectively. There are three cases in Figure 5.11. Figure 5.11a shows two bars do not have intersections. Figure 5.11b and Figure 5.11c show the two bars both has one intersection. However, the distance between the reminder location and the current location is greater than d_u in Figure 5.11b, and the distance between the reminder location and the current location is smaller than d_u in Figure 5.11c. Therefore, the user will additionally retrieve the reminder content from the cloud server if the case in Figure 5.11c happens.

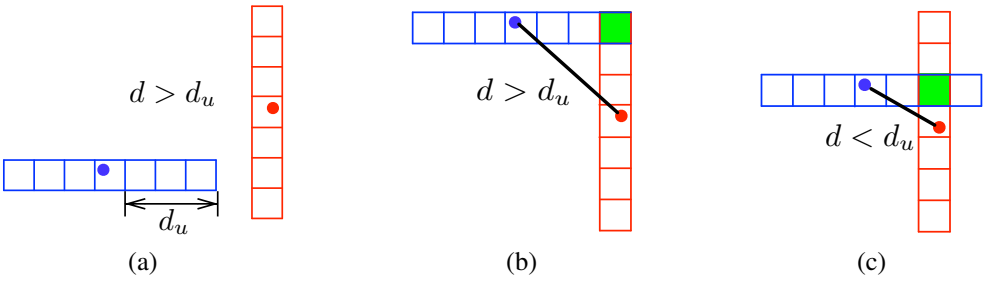


Figure 5.11: Three Cases of Cross Intersection

5.8.2 Bloom Filter For Private Location Search

Bloom filter [13] is a space-efficient randomized data structure for representing a set in order to support membership queries [22]. Suppose we want to use a m-bit bloom filter for a data set with N elements $\{d_i\}_{i=1}^{N}$. The initial bits of the bloom filter are all set to 0. We choose k hash functions $\{h_j(\cdot)\}_{j=1}^{k}$, with each output of the hash function in the range of $1-w$. For each element d_i, we compute the hash values $\{h_j(d_i)\}_{j=1}^{k}$, and set the $h_j(d_i)$-th component of the bloom filter to bit 1. To check if an element e is in the set $\{d_i\}_{i=1}^{N}$, we compute k hash numbers $\{h_j(e)\}_{j=1}^{k}$, and check if all k bits at index $\{h_j(e)\}_{j=1}^{k}$ is 1. If it is, e is in the data set with a large probability, otherwise, we can make sure that e is definitely not in the data set.

In our proposed idea, we use bloom filter for private search in the cloud server. Suppose the coordination of the reminder location is (x,y), for each $id \in \mathscr{ID}_{(x,y)}$, the user computes its hash values $\{h_j(id)\}_{j=1}^{k}$, and sends them to the server. The server sets the corresponding components to bit 1. After that, the user periodically uploads the hash values regarding her current location to the server, which searches in the bloom filter to find if there is any match, i.e. the intersection of the two bars representing the user's reminder location and current location, respectively.

5.8.3 Constructions

In this section, we give details of our system construction. We assume that the number of a user's active reminders is no more than N. For example, it is difficult for a person to handle 500 reminders at the same time. While using a dynamic bloom filter [59] is able to enhance the space efficiency and dynamically respond to the user's increasing demand, we would like to focus on comparing the usage of bloom filters and that of searchable encryptions for secure location based reminder system in this dissertation.

For a specific user u, given the reminder distance d_u and square length a_u, the number of square IDs in a bar is $2\lceil d_u/a_u \rceil + 1$. The capacity, in terms of reminders, of the bloom filter is the number of reminders it can hold while the false positive probability is less than a designated number. Since the more elements stored in the bloom filter, the greater probability a collision happens in the bloom filter, to reduce the error probability, we need to restrict the capacity of the bloom filter. Assume the capacity of the bloom filter is N. The bloom filter array size is M. Each component of the bloom filter is a tuple $< b, \mathscr{R} >$, where b is a bit denoting whether the component is set and $\mathscr{R}$ is a set of references associated with the component. Given a reference, the server can locate the ciphertext of a reminder message. The number of hash functions is t. To construct t hash functions, a user generates t hash keys $k_1, \cdots, k_t$, where each key is a random number. A hash function is constructed as: $h_i(x) = h(x||k_i) \mod M$, $i = 1, \cdots, t$.

The system mainly consists of three protocols: location marking protocol, location search protocol, and removing reminder protocol.

In the location marking protocol (Fig. 5.12), a user picks the reminder location and sets up a location based reminder, forms the identifier set $\mathscr{I}\mathscr{D}$ regarding the reminder location (x,y). For each $id \in \mathscr{I}\mathscr{D}$, the user computes k hash values $\{h_j(id)\}_{j=1}^{k}$. Since there are

Procedure 4: CreateIndexSets($\mathscr{I}\mathscr{D}$)

Data: a set of square identifiers $\mathscr{I}\mathscr{D}$

Result: a set of index sets $\mathscr{I}$, each element of $\mathscr{I}$ is a set of indices

$\mathscr{I} \leftarrow \emptyset$;
foreach $id \in \mathscr{I}\mathscr{D}$ **do**
    $\mathscr{I}_{id} = \{h_i(id)\}_{i=1}^{t}$; // $h_i(x) = h(x||k_i) \mod M$
    randomly permutes the elements of $\mathscr{I}_{id}$;
    $\mathscr{I} \leftarrow \mathscr{I} \cup \{\mathscr{I}_{id}\}$;
end
randomly permutes the elements of $\mathscr{I}$; // each element is a set
return $\mathscr{I}$

User u marks location (x,y) as a reminder location with a reminder message Msg. u also secretly maintains a vector of hash function secret keys $\{k_1, k_2, \cdots, k_t\}$. u chooses a symmetric encryption key k and keeps it secret.

1. u creates a horizontal bar set $\mathscr{I}\mathscr{D}_{(x,y)} \leftarrow$ CreateHorizontalBar(x,y). u generates a set of index sets $\mathscr{I} \leftarrow$ CreateIndexSets$\left(\mathscr{I}\mathscr{D}_{(x,y)}\right)$.
2. Using the symmetric encryption system, u encrypts the reminder location as $C_1 = Enc_k(x||y)$ and the reminder message as $C_2 = Enc_k(Msg)$. The user assembles them together to form a ciphertext tuple $C =< C_1, C_2 >$. User u sends $< C, \mathscr{I} >$ to the server.
3. Upon receiving $< C, \mathscr{I} >$ from user u, the server stores C and obtains a reference r_C to the stored location. For each index set $\mathscr{I}_i \in \mathscr{I}$ and each element $ind_j \in \mathscr{I}_i$, the server sets $B_u[ind_j].b \leftarrow 1$ and puts r_C into the associated reference set $B_u[ind_j].\mathscr{R} \leftarrow \{r_C\} \cup B_u[ind_j].\mathscr{R}$.

Figure 5.12: Location Marking Protocol

$2\lceil d_u/a_u \rceil + 1$ squares in a bar, the total number of computed hash values are $(2\lceil d_u/a_u \rceil + 1)k$. These hash values are then randomly shuffled by the user. The user uploads the hash values, and the encryption of the the reminder message and reminder location to the cloud server. The hash values are served as index of the bloom filter. The cloud server sets the corresponding bits at these indices to bits 1, and sets up a one to one mapping between these indices and the ciphertext of the reminder content and reminder location.

User u checks if a given location (x,y) is in any of her existing reminder area and fetches the reminder message.

1. User u constructs a vertical bar set $\mathscr{ID}_{(x,y)} \leftarrow \text{CreateVerticalBar}(x,y)$ and generates a set of index sets $\mathscr{I} \leftarrow \text{CreateIndexSets}\big(\mathscr{ID}_{(x,y)}\big)$. User u sends $\mathscr{I}$ to the server.
2. Upon receiving $\mathscr{I}$ from user u, the server performs as follows for each $\mathscr{I}_i \in \mathscr{I}$
   - Checks if $\bigwedge_{ind_j \in \mathscr{I}_i} B_u[ind_j].b = 1$.
   - If no such $\mathscr{I}_i$ exists, informs user u. If it is true for some $\mathscr{I}_i$, calculates set intersection $\mathscr{R}^* = \bigcap_{ind_j \in \mathscr{I}_i} B_u[ind_j].\mathscr{R}$, retrieves the first part of every referenced ciphtertext tuples $\mathscr{C}_1 = \{< C_1, r_C > | C =< C_1, * > \text{ and } r_C \in \mathscr{R}^*\}$, and sends $\mathscr{C}_1$ to u.
3. Upon receiving $\mathscr{C}_1$, user u decrypts each element to get a location coordinate tuple $\mathscr{L} = \{< (x',y'), r_C > | (x',y') = Dec_k(C_1) \text{and } < C_1, r_C > \in \mathscr{C}_1\}$. The decryption to a coordinate tuple is possible because each coordinate is of fixed bit size. User checks the distance between each pair of (x,y) and (x',y'), and obtains the valid reminder references $\mathscr{R}' = \{r_C | < (x',y'), r_C > \in \mathscr{L} \text{ and } (x-x')^2 + (y-y')^2 \leq d_u^2\}$. u sends $\mathscr{R}'$ to the server.
4. Upon receiving $\mathscr{R}'$, the server retrieves the ciphertext $\{C | r_C \in \mathscr{R}'\}$ and sends back to the user. The user uses her secret key k to recover the reminder message.

Figure 5.13: Location Search Protocol

In the location search protocol (see Fig. 5.13), the user periodically uploads the hash values regarding her current location to the cloud server. The cloud server checks the bloom filter to find a match. If any match exists, the cloud server sends the corresponding reminder location to the user. The user computes the distance between her current location and the reminder location. If the resulting distance is less than the reminder distance, the user retrieves the reminder content from the cloud server.

After the user successfully retrieves the reminder from the cloud server, the cloud server should be able to remove it from the storage, which is illustrated in the removing reminder protocol (see Fig. 5.14).

Given a reminder location (x, y) and reference r_C, remove the relevant reminder information stored on the server.

1. User u constructs a horizontal bar set $\mathscr{ID}_{(x,y)} \leftarrow \text{CreateHorizontalBar}(x, y)$, generates a set of index sets $\mathscr{I} \leftarrow \text{CreateIndexSets}\big(\mathscr{ID}_{(x,y)}\big)$, and sends $< \mathscr{I}, r_C >$ to the server.
2. Upon receiving $< \mathscr{I}, r_C >$ from the user, the server delete the ciphtertext tuple stored at r_C. For each $\mathscr{I}_i \in \mathscr{I}$ and each $ind_j \in \mathscr{I}_i$, the server removes r_C from $B[ind_j].\mathscr{R}$. If $B[ind].\mathscr{R}$ becomes empty, the server sets $B[ind].b \leftarrow 0$.

Figure 5.14: Removing Reminder Protocol

5.9 Analysis

The hashing key is randomly selected by a user and kept secret in the user's local storage. The hashing result will then be converted into the residue system modulo M. Because bit size of M is far smaller than the output size of hash function H, the server has no way to determine the real hashing result. The user randomly permutes each $\mathscr{I}_{id}$ such that linking the hashing output to the hashing key becomes more difficult for the server. Therefore, the server cannot recover the square identifier from the index set $\mathscr{I}_{id}$. The user also randomly permutes among index sets, i.e. $\mathscr{I}_{id}$'s in $\mathscr{I}$, so that the server cannot learn the relationship between any two squares.

All the message is encrypted by a symmetric encryption system. The server cannot learn the content because the encryption key k is kept secret by the user. The user should use CBC mode of symmetric key encryption such that replacing or modifying any block of the ciphertext will fail. Furthermore, the user can employ CCA2 secure symmetric key encryption system [*Chosen Ciphertext Secure (CCS): Stateful Symmetric Key CCA Encryption with Minimal Ciphertext Expansion*] to provide more protection to the reminder message.

Table 5.1: Overhead Comparison

	server storage	user indexing time	server matching time
Mercury	$M+(2A_u+1)nK\cdot l_r$ ‡	$O((2A_u+1)K\cdot \texttt{mul})$	$O(2A_u+1)$
[105]	$(4A_u+1)n\cdot(2l_g+l_e+l_r)$ *	$O((4A_u+1)l_g\cdot \texttt{mul})$	$O((4A_u+1)\log n)$

‡ l_r is the bit size of a reference.

* l_g and l_e are the ECC group size and the bit size of a key encryption, respectively.

Early matching problem: causing unnecessary decryption. all points in a $(2d+a)\times(2d+a)$ rectangle centered at the $id_{(x,y)}$ will trigger a matching and the following up decryption. However, the desired area is a plate around the reminder location. So the early matching area is $(2d_u+a_u)^2-\pi d_u^2$. The early matching probability is

$$1-\frac{\pi}{2+\frac{a_u}{d_u}}$$

Given a reminder distance d_u, the smaller a_u is the less probably the early matching happens. However, the smaller a_u will cause more computation during the location marking and search, as well as the server side storage.

Given the maximum reminder number N, the expected maximum of squares stored in the bloom filter is $N_s = N\cdot(2\lceil d_u/a_u\rceil+1)$

To achieve a false alarm probability P, the bloom filter size is given by $M=\frac{N_s\ln P}{(\ln 2)^2}$ and the needed hashing key number is $K=\frac{M}{N_s}\ln 2$. A typical usage is to set remind distance as 1, 000 meters and the square length 20 meters. If the maximum number of reminders is $N=500$, we will have 25000 elements to be stored in the bloom filter.

If we set $P=1\%$, we have $M=23,963<3KB$ and $K=7$. A typical configuration of the reminder system in [105] contains a 160-bit ECC group a 80-bit symmetric key, which gives $l_g=160$ and $l_e=80$. A typical server storage overhead is $(400+r_C)(4A_u+1)n$

The computation overhead on user side is to index a location, computing either pseudo-

random number as in [105] or hash function in Mercury. To index a square in [105], we need to do a hashing, a group exponentiation, and an symmetric encryption, while we just need to do k hashing operations in this dissertation. If we take hashing equivalent to a group multiplication `mul` in term of time overhead, we have the overall user indexing time in the following table. A fast group exponentiation takes $O(l_g\texttt{mul})$ operations. $A_u = \lceil d_u/a_u \rceil$

5.10 Simulation

We evaluate the performance of our system through simulations. The simulations were carried out on an Motorola Droid phone and a laptop, which simulates a cloud server. The smartphone runs an Android 2.2 OS and has a 550MHz ARM A8 processor, 256MB memory, and a 16GB SD card. The laptop has a 2.53GHz Intel Dual Core CPU and a 4GB memory. The cryptographic library used in our simulation is Crypto++5.6.1 [41] and we ported it to the Android platform. We used SHA-1 and SHA512 as the hash functions in our simulations and AES encryption in CFB mode as the semantic secure symmetric encryption in the system.

Our pseudo random function uses a hash-into-group hash function, which hashes an input into a group element and is a probabilistic algorithm. We have introduced its construction in Section 5.3.2. We first tested the success probability of the construction of the hash function, which took up to 10 attempts before it outputs *fail*. For test purpose, we hashed 1000 random numbers into four groups of different orders, 128, 160, 512, and 1024 bit groups. The first two group orders are popular selections for the elliptic curve groups (ECC) and the last two are for the quadratic residue groups over the Galois field mod a safe prime number, which is called DL groups in this section. We obtained 160-bit and 512-bit outputs using SHA-1 and SHA-512. In order to obtain a 128-bit output, we used SHA-1 and took the 128 least significant bits as the output. To obtain a 1024-bit output, the binary

string of the input number was divided into two halves, which are hashed into a 512-bit string, respectively. The two resulting hash values were then concatenated to form a 1024 bit output. The number of attempts needed to hash each number into the desired group was recorded. The hash is called *fail* if the attempt number is beyond 10. Figure 5.15a shows the result. We can see that, in our experiments, to hash a number into 128-bit and 1024-bit groups, we needed to try no more than 4 times. For 160-bit and 512-bit groups, we needed to re-hash a small portion of numbers for more than 5 times. We note that, in our experiments, all 1000 random numbers were successfully hashed into the groups in 10 attempts.

When a user marks a location, most of the time is spent on the calculation of the trapdoors of the cross squares, which is dependent on $\lceil d_u/a_u \rceil$. We recorded the time needed for a user to compute $<\mathscr{T}, C>$ for a location while the square side length a_u is fixed at 20 meters and the reminder distance is from 500 meters to 3500 meters, with a 200-meter increment. Our system works on an underlying cyclic group, on which the DDH problem is hard (see Section 5.3.1). There are two types of groups, a quadratic residue subgroup of a Galois field mode a safe prime number (DL group) and an elliptic group (ECC group). To compare their performances, we implemented our system over a 1024-bit DL group and a 160-bit ECC group, which has an equivalent security level according to NIST's guidance [82]. The recorded computation time needed for different reminder distances over the two groups are shown in Figure 5.15b. First, it can be seen from Figure 5.15b that the computation time increases linearly with the increase of the reminder distance. Second, it is shown that the implementation based on DL group costs much more time than the one based on ECC group. This is because the DL group size is much larger than the ECC group size and a group operation costs more time on a larger group.

Our system does not encrypt the reminder message using $g(k_2, id)$. We use a random key

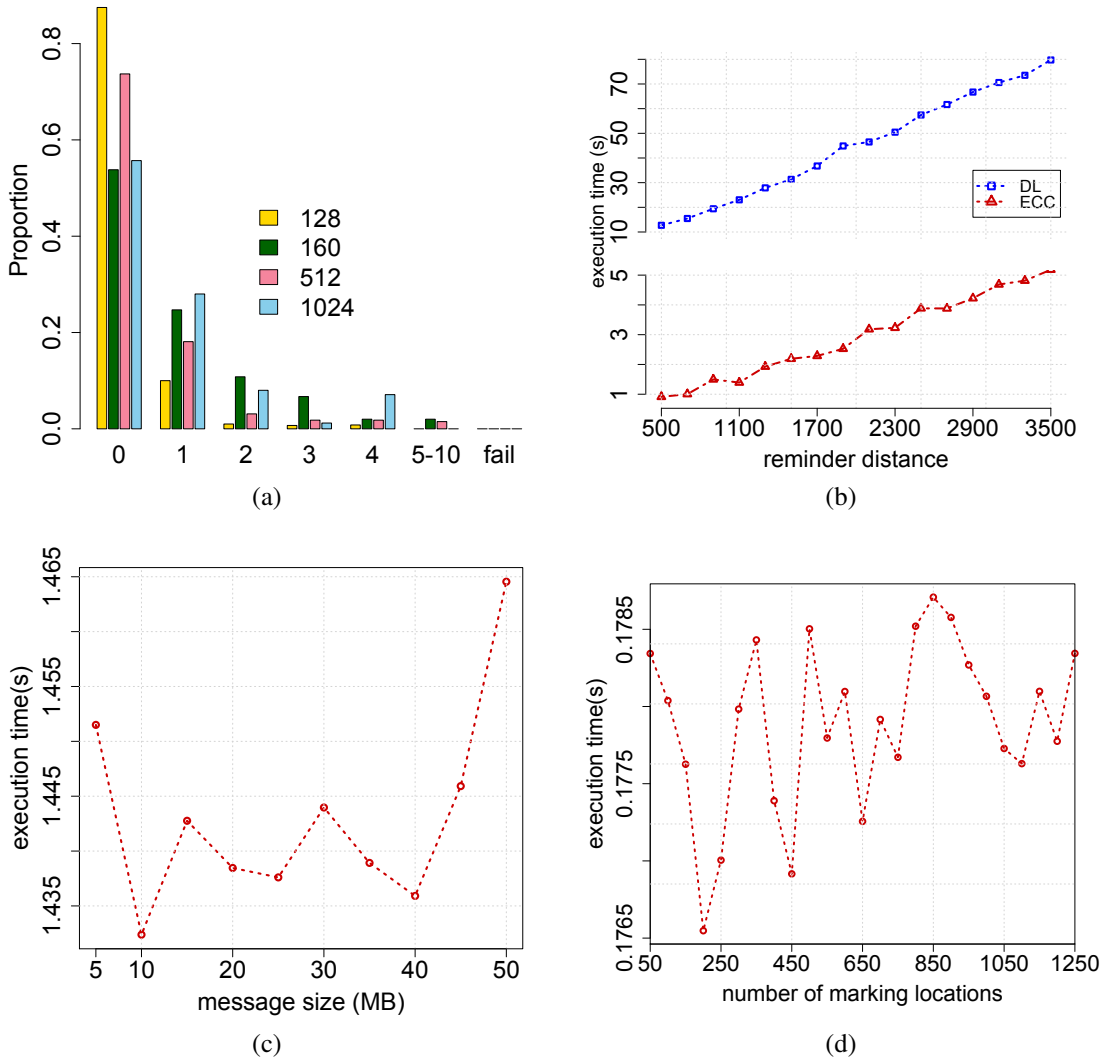


Figure 5.15: Simulation Result on the System Constructed Based on the Cross Based Area Representation Approach

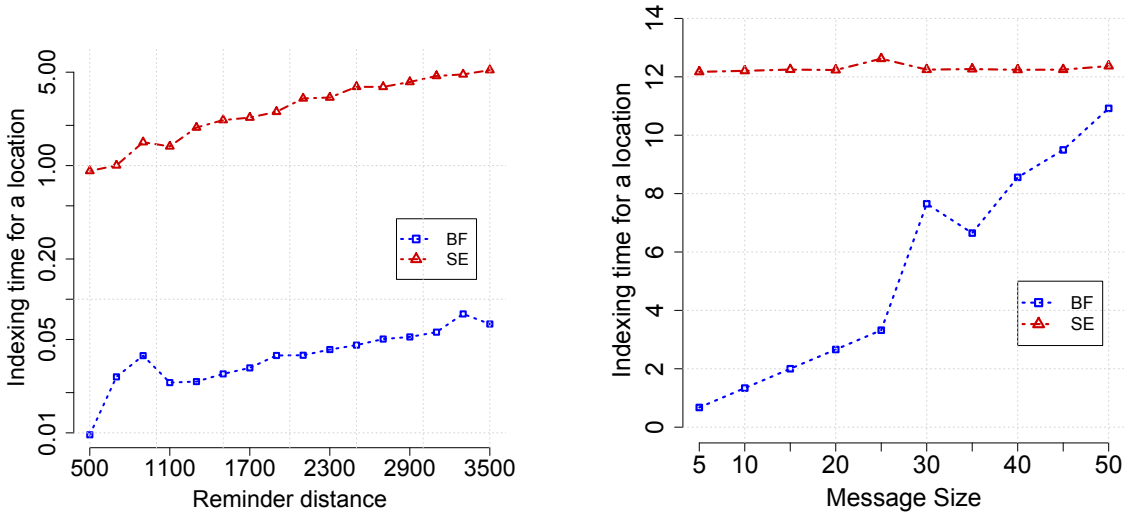
to encrypt the reminder message and use the second trapdoor to encrypt the random key. This saves time when the message size goes large. We tested the computation time of a user's marked location for different sizes of reminder messages. We fixed $a_u = 20$ meters and $d_u = 1000$ meters and increase the reminder message size from 5M bytes to 50M bytes with a 5M-byte increment. The result is shown in Figure 5.15c. As shown in Figure 5.15c, the execution time did not change much, from 1.435 seconds to 1.465 seconds, because the

extra time is just spent on the symmetric encryption, which is fast.

For the cloud server, our system lets the server keep a sorted set of (by trapdoor value) user index tuples to speed up the search for a trapdoor. We used a laptop to simulate the server and used a red-black tree [37] to maintain the sorted index set. We recorded the time for searching a random location when different number of locations have been marked. The number of marking locations is from 50 to 1250, with an increment of 50. We note that marking a location or searching a location incurs $4\lceil d_u/a_u \rceil + 1$ searches in the set. The result is shown in Figure 5.15d. It can be seen from the figure that with the increase of the number of marking locations, the search time increases but not linearly with the number, because the red-black tree's leaf nodes are not on the same level and the searching may end at some internal node. It is also shown in Figure 5.15d that the time variance gap is small, from 0.1765 second to 0.1787 second.

We compared the systems constructed based on the cross based area representation approach and the bar based area representation approach, respectively. In this section, the previous system is denoted by SE and the new system is denote by BF.

As a smart device application, we usually pay more attention to the time overhead of client side, because the smart device's computation capability is limited. The client side's computation time depends on the number of the squares to be indexed and the size of the reminder message to be encrypted. Fixing a square length as 20 meters and message size as 1024 bit, we collected the time that is used to mark a location, when the reminder distance increases from 500 meters to 3500 meters, with 200-meter increment. This reflects the system's time overhead on indexing a location. Then, we fix the reminder distance as 1000 meters, change message from 5M bytes to 50M bytes, and record the execution time on client side. The results are shown in Fig. 5.16a and 5.16b, respectively. We notice that It is obvious that the bloom filter based system, denoted by blue lines, is faster than the previous one on



(a) Indexing time (y-axis is log scaled)

(b) Marking with large message size

Figure 5.16: Comparison Between SE System and BF System

client side. It is because that the searchable encryption based system generates a pseudo random number in indexing a square, while the bloom filter system calculates K hashing values. Pseudo random number generation involves an exponentiation that is expensive. The recorded indexing time of the BF system is less than 0.07 seconds while that of the SE system is almost 5 seconds. When marking with large size message, the SE system usually costs 12 seconds while the time cost by BF system increase with the message size. When a message size is big, such as 50M bytes, the two execution time are close. Encrypting, processing and transmitting messages occupy most part of execution time.

Fixing the reminder distance, as 1000 meters, square length, 20 meters, and message size, 1024 bits, we recorded the time of searching a reminder when different numbers of reminders has been uploaded, from 50 to 1250. The time is the average value of 50 searches. The result is shown in Fig. 5.17. From the figure, we can see that the search time for SE system is between 0.1765 seconds and 0.1785 seconds, while BF system finishes one search in less than 0.0003 seconds. The difference is remarkable because SF system search among n reminders, which cost $O(\log n)$ time. On the same time, BF system searches one

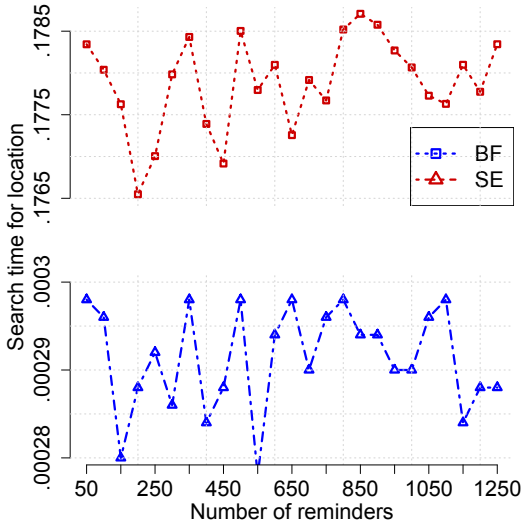


Figure 5.17: Comparison Between SE System and BF System: Searching Time

square in a short time, by only comparing K value and K is usually small, e.g. $K = 7$ in our experiment.

Using bloom filter to do the search is a probabilistic algorithm. In other words, it may return a wrong result. Bloom filter does not give us the false negative, i.e. claiming an existing element as non-existing one. It only has false positive error probability. As discussed previously, a desired error probability can be achieved by carefully choosing M, N, and K. In our simulation, we fixed the reminder distance as 1000 meters, square length as 20 meters, and message size as 1024 bits. We set the bloom filter's reminder capacity N as 500 and tried to see the bloom filter's performance when the number of inserted reminders increases. We search 1000 locations, whose geographic coordinates are randomly generated, in the bloom filter while different number of reminders have been inserted in the bloom filter. The number of inserted reminders is from 100 to 900, with 100 increment. The result is shown in the bar plot of Fig. 5.18. Each bar corresponds to a 1000-search experiment. The green part denotes the correct search — either found or not found.

The red part denotes the false alarm of bloom filter search — the bloom filter claims that

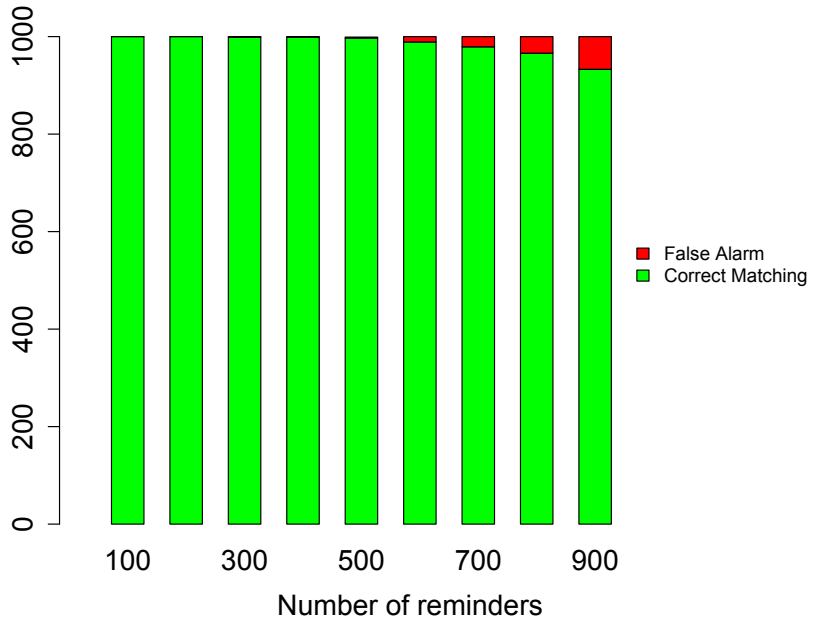


Figure 5.18: False Alarm of Bloom Filter

some square *id* on the location's vertical bar has been found because $BF[ind_j].b = 1$ for all $i \in \mathscr{I}_{id}$. $\mathscr{I}_{id}$ is the index set of *id*. We note that when we mark one reminder, KA_u of elements will be inserted into the bloom filters. As shown in the figure, the all searches are correct when the number of reminders are 100 and 200. When it increases to 300, 400, and 500, there is 1 false alarm while it still gives us 999 correct searches. When more reminders than the capacity are inserted, the more false alarms are observed. When there were 900 reminders inserted, the number of false alarm searches increase to 67 out of 1000 searches. However, we observed that the false alarm probability is 2.1% even when 700 reminders (20% more than the capacity) are inserted. We remark that even the false alarm of bloom filter searching happens, it may not result in a system wide false alarming. The system will also check if all associated reference sets share the common reference.

We discussed that a larger square length gives rise to the early matching problem, in which, a current location got matched with some reminder location but their actual distance is

larger than the reminder distance. The early matching problem causes unnecessary decryption of reminder locations and the following up calculations. We fixed a rectangle area of $2500 \times 2500\text{m}^2$, used the center point as the reminder location, and marked it in the server. The reminder distance is 1000. Then we changed the square length from 20 meters to 400 meters, with 20-meter increment. For each square length, we pick 2000 locations randomly from the area and do location search. If one search returns no reference, we call it a "Correct Non-Match"; We call one search "Early Match" if it returns some references but the current location is actually beyond the reminder distance. When a search really find a valid reminder location, we call it "Correct Match". The results are shown in Fig. 5.19. Each bin

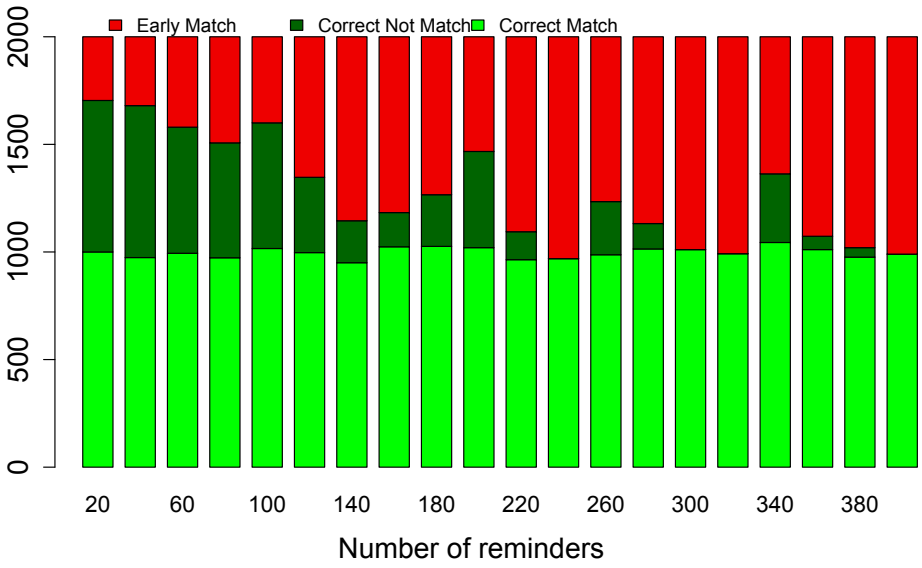


Figure 5.19: Early Matching

corresponds to a square length. The red bar denotes the early match. From the figure, we can see that the number of early matches grows up with the increase of the square length. This intend is same as what we analyzed previously. It is because a bigger square length results in a larger early match area while the reminder area stays same. The number correct match in all square lengths are approximately same, because the reminder area are same in all tests. So the rate of correct match is $1000^2\pi/(2500 \times 2500) \approx 0.5$. In other words,

there are approximately 1000 correct matches out of 2000 searches, which is also reflected by the green bars in Fig. 5.19.

5.11 Conclusions

In this dissertation, we have proposed a secure location based reminder system. This system enables the cloud server to securely search over the reminders stored on it. In addition, the user could receive the reminder message from the cloud server with whichever smart device she brings as long as she is in the reminder area. We have proposed novel area representation approaches, the cross based area representation approach and the improved bar based area representation approach, respectively, which are easy to construct and achieves lightweight storage overhead. We have theoretically proved the security of our system. The evaluation of our system has shown that it is efficient to implement our system.

CHAPTER 6

Conclusions and Future Work

In this section, we conclude our work and envision some futurn works to do along this research direction.

6.1 Conclusions

In this dissertation, we have studied four topics: identity privacy in OSNs, identity privacy in anonymous message submission, location privacy in LBSNs, and location privacy in location based reminders.

In the first topic, we have proposed a system to hide users' identity from untrusted server when she accesses her information on the server. We have designed a fine-grained access control mechanism to restrict unauthorized accesses to users' private information. In the second topic, we have proposed a shuffle protocol to hide the relationship between users' identities and their submitted messages. The protocol is constructed based on the secret sharing scheme and is more efficient compared to previous works. In the third topic, we have proposed a framework to preserve users' location privacy from the untrusted LBSN server. We have also taken the limited computation resources on the user side into consideration and have proposed an alternative approach by outsourcing computations to the server. In the last topic, we have proposed a system to hide users' location information from the untrusted cloud server in location based reminders. We have given a cross based approach and an improved bar based approach, respectively, to represent users' reminder area. The proposed approaches are efficient in terms of user's computation overhead and

saves a lot of search time on the server side.

6.2 Future Work

The mobile revolution happened in the past few years has posed much more privacy threats than ever. Smart devices keep people connecting to the digital world. In other words, people's privacy, including identity, location, habits, etc., has been exposed to attackers 24 hours 7 days. The change of the way how people use the Internet services has posed many new challenges on privacy issues. Therefore, I believe there are many works can be done along this research direction.

- **Identity privacy in unlocking via smart wearable devices.** Smart wearable devices, such as smart watches, have hit the market for years. Wearable devices are usually viewed as private properties and less likely shared with other people than phones. Many companies start to use smart wearable devices as an ID token and use them to unlock other devices, like laptops, cars or hotel doors. However, how to protect identity privacy in such scenario is still an open question. For example, people want to use their smart watch to unlock hotel doors while keeping the fact that they have been to the door secret from the server.

- **Location privacy in address search service.** The most often used location service is the conversion between a geographical address and its GPS point. Many location based services are built on top of this basic operation. Studying a privacy preserving and efficient way to carry out this conversion is of significant meaning to the mobile location privacy protection.

- **Identity/location privacy in Bluetooth low energy (BLE) beacon usage.** A typical BLE beacon scenario is that when a custom comes near a BLE beacon with her smart

phone, the beacon will push store advertisements to the user. The unwanted side of this case is the leaking of users' location privacy and identity privacy. The beacon or the server will identify the same user by the smart phone mac address and trace the user. Therefore, protecting user's identity and location information while still providing the same quality of service is interesting and worthy to study.

- **Utilities for privacy usage.** In this dissertation, I have explored many approaches to provide strong privacy protection for online interactions. However, these protections cannot redeem their real values unless the blocks of their adoption in practices are all removed. One of such blocks is utility function, which seems trivial to the academic research. For example, in the proposed privacy preserving online social network protocol (see Chapter 2), there should be a utility which lets user manage their visitor group in addition to the main protocol functionalities. The group management allows a user to see who can visit which data owned by the user, and to change or revoke the permission that has been granted to the data visitors. I believe more interesting problems will be disclosed when we dive into the study of the needed utility functions. In addition to the above mentioned utility, I have listed possible utilities needed for the other three proposed approaches.

  - anonymous messaging submission: it is useful to have a utility that lets user track whether his or her message has been read and allows the user to delete the message if it has not been read.
  - location based social networks: group management has been designed in the protocol but it is also great to have a utility that lets user manage their check-ins. In other words, users need to brow all his or her check in history and be capable to delete them.

- location based reminder: as many other alarm system, a reminder management utility is needed. The function includes add or delete a reminder, change reminding message, change reminding location, etc. While this sounds trivial, but how to achieve this goal efficient and securely may be an open question.

REFERENCES

[1] Endyanos imedia SLU. Photo reminder pro. https://itunes.apple.com/us/app/photo-reminder-pro/id421893170?mt=8, 2012.

[2] M. AdelsonVelskii, E. Landis, and J. P. R. S. W. D. C. *An algorithm for the organization of information*. Defense Technical Information Center, 1963.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, volume 29, pages 439–450. ACM, 2000.

[4] M. Albanese, A. De Benedictis, S. Jajodia, and P. Shakarian. A probabilistic framework for localization of attackers in manets. *Computer Security–ESORICS 2012*, pages 145–162, 2012.

[5] M. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. *Security and Cryptography for Networks*, pages 111–125, 2006.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: An online social network with user-defined privacy. *ACM SIGCOMM Computer Communication Review*, 39(4):135–146, 2009.

[7] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *WWW*, pages 237–246, 2008.

[8] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

[9] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *focs*, page 394. Published by the IEEE Computer Society, 1997.

[10] M. Bellare and P. Rogaway. Optimal asymmetric encryption-how to encrypt with rsa. In *Advances in Cryptology-Eurocrypt*, volume 94, pages 92–111, 1994.

[11] M. Bellare and P. Rogaway. Introduction to modern cryptography. *Lecture Notes*, 2001.

[12] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret (extended abstract). In *Advances in Cryptology?CRYPTO?86*, pages 251–260. Springer, 1987.

[13] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[14] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology–CRYPTO 2004*. Springer, 2004.

[15] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[16] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[17] D. Boneh, H. W. Montgomery, and A. Raghunathan. Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In *ACM CCS*, pages 131–140, 2010.

[18] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[19] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 97–106. IEEE, 2011.

[20] G. Brassard, C. Crépeau, and J. Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology?CRYPTO?86*, pages 234–238. Springer, 1987.

[21] J. Brickell and V. Shmatikov. Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD*, pages 76–85. ACM, 2006.

[22] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[23] J. Camenisch. *Group signature schemes and payment systems based on the discrete logarithm problem*. Citeseer, 1998.

[24] J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious transfer with access control. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 131–140. ACM, 2009.

[25] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. *Advances in Cryptology-EUROCRYPT 2009*, pages 425–442, 2009.

[26] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. *Advances in Cryptology-EUROCRYPT 2007*, pages 573–590, 2007.

[27] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *focs*, page 136. Published by the IEEE Computer Society, 2001.

[28] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM, 2011 Proceedings IEEE*, pages 829–837. IEEE, 2011.

[29] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[30] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[31] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[32] D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology?CRYPTO?92*, pages 89–105. Springer, 1993.

[33] S. Chawla. Lecture notes of randomize algorthms: Chernoff bounds. hhttp://www.cs.cmu.edu/afs/cs/academic/class/15859-f04/www/scribes/lec9.pdf, 2004.

[34] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 30–39. Society for Industrial and Applied Mathematics, 2004.

[35] Y.-R. Chen, J. D. Tygar, and W.-G. Tzeng. Secure group key management using uni-directional proxy re-encryption schemes. In *INFOCOM*, pages 1952–1960, 2011.

[36] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[37] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

[38] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *ACM CCS*, pages 340–350. ACM, 2010.

[39] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in verdict. In *USENIX Security*, 2013.

[40] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology?CRYPTO?94*, pages 174–187. Springer, 1994.

[41] Crypto++. `http://www.cryptopp.com/`, 2010.

[42] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, pages 79–88, 2006.

[43] I. Damgard, M. Geisler, and M. Kroigard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.

[44] G. Danezis and P. Mittal. Sybilinfer: Detecting sybil nodes using social networks. *NDSS.*, 2009.

[45] J. Domingo-Ferrer, A. Viejo, F. Sebé, and Ú. González-Nicolás. Privacy homomorphisms for social networks with private relationships. *Computer Networks*, 52(15):3007–3016, 2008.

[46] W. Du and M. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22. ACM, 2001.

[47] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[48] Facebook. http://www.facebook.com/press/info.php?statistics, 2010.

[49] A. Fiat and M. Naor. Broadcast Encryption, Advances in Cryptology-Crypto93. *Lecture Notes in Computer Science*, 773:480–491, 1994.

[50] foursquare. `http://blog.foursquare.com/2010/07/01/755614816/`, 2010.

[51] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[52] J. Freudiger, R. Shokri, and J.-P. Hubaux. Evaluating the privacy risk of location-based services. In *Financial Cryptography*, pages 31–46, 2011.

[53] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD Conference*, pages 121–132, 2008.

[54] E. Goh et al. Secure indexes. *An early version of this paper first appeared on the Cryptology ePrint Archive on October 7th*, 2003.

[55] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[56] O. Goldreich. Secure multi-party computation. *Working Draft*, 2000.

[57] P. Golle and A. Juels. Dining cryptographers revisited. In *Advances in Cryptology-Eurocrypt 2004*, pages 456–473. Springer, 2004.

[58] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*, 2003.

[59] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo. The dynamic bloom filters. *IEEE Trans. Knowl. Data Eng.*, 22(1):120–133, 2010.

[60] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *Proceedings of the VLDB Endowment*, 1(1):102–114, 2008.

[61] U. Hengartner. Hiding location information from location-based services. In *MDM*, pages 268–272, 2007.

[62] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, pages 418–435, 2010.

[63] S. Jarecki and X. Liu. Fast secure computation of set intersection. *Security and Cryptography for Networks*, pages 418–435, 2010.

[64] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving anonymity in location based services. 2006.

[65] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security*, pages 965–976, 2012.

[66] A. Kate. The pairing-based cryptography (pbc) library. http://crysp.uwaterloo.ca/software/PBCWrapper/.

[67] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD*, pages 239–257, 2007.

[68] A. Khoshgozaran, C. Shahabi, and H. Shirani-Mehr. Location privacy: going beyond k-anonymity, cloaking and anonymizers. *Knowl. Inf. Syst.*, 26(3):435–465, 2011.

[69] T. Kivinen and M. Kojo. More modular exponential (modp) diffie-hellman groups for internet key exchange (ike), 2003.

[70] A. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *IEEE Symposium on Security and Privacy, S&P (to appear, 2010)*.

[71] L. Li, X. Zhao, G. Xue, and G. Silva. Privacy preserving group ranking. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 214–223. IEEE, 2012.

[72] M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *Proc. of Infocom*, 2011.

[73] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology?CRYPTO 2000*, pages 36–54. Springer, 2000.

[74] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 229–240. ACM, 2012.

[75] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin. Attacks on webview in the android system. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 343–352. ACM, 2011.

[76] T. Luo, X. Jin, A. Ananthanarayanan, and W. Du. Touchjacking attacks on web in android, ios, and windows phone. In *Proceedings of the 5th International Symposium on Foundations & Practice of Security*, October 2012.

[77] B. Lynn. The pairing-based cryptography (pbc) library. `http://crypto.stanford.edu/pbc/`.

[78] Manas Gajare. Voice (audio) reminder. `https://play.google.com/store/apps/details?id=com.dexter.audio\_reminder\&hl=en`, 2012.

[79] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: Query processing for location services without compromising privacy. In *VLDB*, pages 763–774, 2006.

[80] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *Proc. of NDSS*, 2011.

[81] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. *Public Key Cryptography–PKC 2007*, pages 343–360, 2007.

[82] NIST, CSE. *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*, Mar. 2011. Available at http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf.

[83] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.

[84] M. Rabin. How to exchange secrets by oblivious transfer. Technical report, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981, 1981.

[85] C. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

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

[87] R. Shokri, G. Theodorakopoulos, J.-Y. L. Boudec, and J.-P. Hubaux. Quantifying location privacy. In *IEEE Symposium on Security and Privacy*, pages 247–262, 2011.

[88] E. Sirer, S. Goel, M. Robson, and D. Engin. Eluding carnivores: File sharing with strong anonymity. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, pages 19–es. ACM, 2004.

[89] K. Smith. http://www.businessinsider.com/10-icloud-tips-and-tricks-2012-8?op=1.

[90] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[91] A. Squicciarini, M. Shehab, and F. Paci. Collective privacy management in social networks. In *Proceedings of the 18th international conference on World wide web*, pages 521–530. ACM, 2009.

[92] D. Stinson. *Cryptography: theory and practice*. CRC press, 2006.

[93] J. Sun, X. Zhu, and Y. Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[94] P. Syverson, D. Goldschlag, and M. Reed. Onion routing for anonymous and private internet connections, 1999.

[95] B. Viswanath, A. Post, K. Gummadi, and A. Mislove. An analysis of social network-based sybil defenses. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pages 363–374. ACM, 2010.

[96] M. Waidner, B. Pfitzmann, et al. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. page 690. Springer-Verlag, 1989.

[97] C. Wang, K. Ren, S. Yu, and K. M. R. Urs. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *INFOCOM*, pages 451–459, 2012.

[98] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. *10th OSDI*, 2012.

[99] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *2010 IEEE Symposium on Security and Privacy*, pages 223–238. IEEE, 2010.

[100] W. Wong, D. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD*, pages 139–152. ACM, 2009.

[101] Z. Yang, S. Zhong, and R. Wright. Anonymity-preserving data collection. In *ACM SIGKDD*, pages 334–343. ACM, 2005.

[102] A. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167. IEEE, 1986.

[103] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE*, pages 366–375, 2008.

[104] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 3–17. IEEE, 2008.

[105] X. Zhao, L. Li, and G. Xue. Secure cloud-assisted location based reminder. In *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China - May 08 - 10, 2013*, pages 323–328, 2013.