國立交通大學

資訊科學與工程研究所

碩 士 論 文

用於共享式訊息之公開金鑰加密系統

A Public-Key Encryption Scheme for a Shared Message

研 究 生：簡韶瑩

指導教授：曾文貴　教授

中 華 民 國 九 十 八 年 六 月

用於共享式訊息之公開金鑰加密系統

學生：簡韶瑩　　　　　　　　　　　　　　指導教授：曾文貴 博士

國立交通大學資訊工程學系

摘要

在本論文裡，我們提出一個適用於共享式訊息之公開金鑰加密系統。在訊息被多個儲存伺服器共享的情況下，我們可以利用這個公開金鑰加密系統將訊息加密成密文，並且保證即使一定數量的儲存伺服器合作，仍無法得知關於共享式訊息的任何資訊。這樣的系統特別可以運用在保護分散式儲存系統內的使用者隱私權。我們提出一個非互動式的公開金鑰加密系統用於共享式訊息的加密，並且證明在 Decisional Diffie-Hellman assumption 底下能夠抵擋惡意的攻擊者。本論文是第一篇適用於分享式訊息的公開金鑰加密系統，並且是有效率的，能夠運用至大規模的系統。

關鍵字： 分散式儲存系統、隱私權保護、秘密分享

A Public-Key Encryption Scheme for a Shared Message

Student: Shao-Yin Chien Advisor: Dr. Wen-Guey Tzeng

Department of Computer Science

National Chiao Tung University

Abstract

This paper proposed a public key encryption scheme for a shared message. The message is shared among storage servers such that the message can be encrypted into a ciphertext without leaking any information even if a given threshold of storage servers cooperate. This is especially suitable for the applications, for example, distributed storage systems. We present a non-interactive public key encryption scheme on a shared message and prove that it is semantically secure against malicious adversaries, under the Decisional Diffie-Hellman assumption. This is the first public key encryption performed on a shared plaintext. Our scheme is efficient and practical for large scale systems.

Keywords: distributed storage system, privacy preserving, secret sharing

誌　　　　　謝

首先感謝我的指導老師曾文貴教授，在我碩士班兩年間的學習過程中，帶領我深入密碼學的領域。老師認真積極的教學態度，使我受益良多。另外，我要感謝口試委員，中研院呂及人教授、交大謝續平教授與交大蔡錫鈞教授，在論文上給我許多建議與指導，讓我的論文更加完善。除此之外，我也要感謝實驗室學姊林孝盈、學長朱成康，以及實驗室學弟和其他的碩士班同學的幫忙。

最後，我要感謝我的家人，不論在精神或物質上都給我極大的支持，讓我在無後顧之憂的情況下可以順利完成學業。在此，謹以此文獻給所有我想要感謝的人。要謝的人太多了，只好謝天了。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Network infrastructures are well-established in recent years. People can easily access the Internet from anywhere at anytime. More and more network services provide easy ways to access information on the Internet. For example, a user can upload a personal schedule or a document to the service provider, and retrieve it whenever he wants. Most of these network services offer a storage space for users so that they don't have to retain the data by their own. Similarly, the trend of data out-sourcing has accelerated dramatically. Many organizations and companies out-source their data to reliable service providers, to reduce routine data management work, such as backup. Saving data on network storage is convenient, but privacy issues come up with it. When datum is transfered over a public channel, it needs to be assured that the data can only be known to the owner or someone authorized. In this paper, we investigate this issue of protecting the privacy of data in distributed storage systems over public networks.

We consider the following scenario. An *owner* holds a message and outsources it to a distributed storage system. To protect the privacy, the owner uses a secret sharing scheme [25] to share the data into *storage servers*. Thus, any collusion of less than t storage servers gets no information about the message for a threshold t. When the owner wants to send the message to a remote receiver, he may apply one of the following methods:

1. The owner retrieves all message shares and combines them into the original message. Then the owner encrypts the message by the receiver's public key and sends the ciphertext to the receiver.

2. The owner sends a command to the distributed storage system. Then each storage server encrypts his message share into a partial ciphertext and sends his partial ciphertext to the receiver. The receiver decrypts all partial ciphertexts and combine them into the original message.

3. Similar to method 2, but there is a *combiner* in charge of combining partial ciphertexts into a complete ciphertext. The combiner just sends the complete ciphertext to the receiver. The receiver decrypts the ciphertext and gets the original message.

The method 1 is inconvenient to the owner. The method 2 needs quite a lot of communication between the distributed storage system and the receiver. We prefer the method 3, which is described in Figure 1.1. This scenario is suitable for the receiver who has limited communication bandwidth with the Internet. Also, the receiver uses less computational time. In

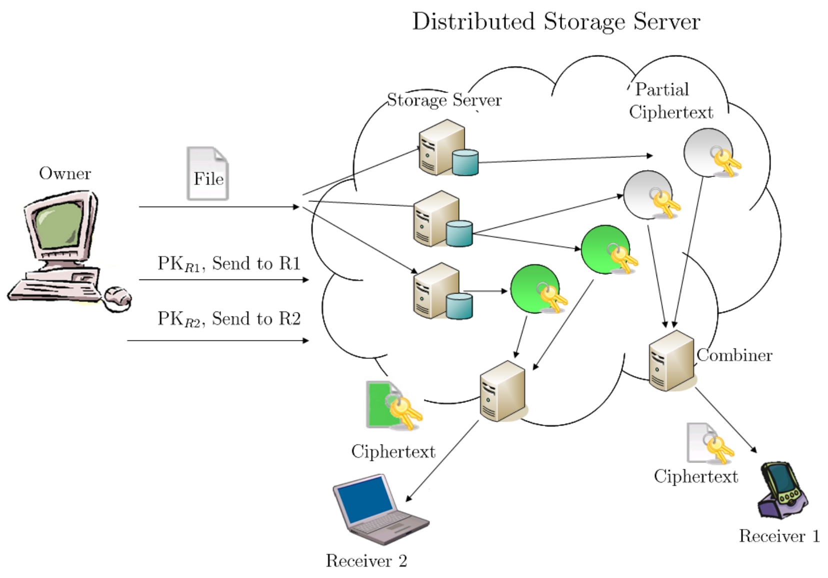

Figure 1.1: A message is shared on the storage servers, and it can be dynamically forwarded to different receivers.

this paper, we propose a public key encryption scheme to encrypt a message which is shared among many storage servers.

The rest of paper is organized as follows: In Chapter 2, 3, we review some definitions and preliminaries. In Chapter 4, we provide the main construction of the scheme against semi-honest adversary. In Chapter 5, we discuss the construction against malicious adversary. In Chapter 6, we discuss the implementation of our scheme. In Chapter 7, we state conclusions.

Related works. Many distributed storage systems have been developed in the past years. Earlier schemes [10, 18] do not consider privacy protection for user's data. Recent schemes [19, 24, 1, 15, 17] employ encryption to protect privacy. In such systems, proxy re-encryption schemes [20, 5, 2] are solutions for message forwarding, as they translate the ciphertext from the owner to the receiver. Some distributed storage systems use secret sharing scheme to protect the encryption keys [27, 6], and some others apply secret sharing directly on the data [8, 21, 26].

A threshold public key encryption system [4, 9] is a public key system with the private key shared among n decryption servers so that at least t servers are needed to decrypt a ciphertext. On the other hand, a collusion of less than t servers cannot decrypt the ciphertext. This setting is different from ours, in which the message is shared.

Our contribution. Contrast to previous threshold public key encryption systems in which the secret key is shared, a message is shared into storage servers in our model. When the message owner wants to send the message to a remote receiver, he just sends a command to the storage servers. Each storage server produces a partial ciphertext to an entity, called the combiner. If the combiner obtains partial ciphertexts from at least t storage servers, he then combines those partial ciphertexts as a complete ciphertext and sends the complete ciphertext to the receiver. Thus, the shared message can be encrypted into different ciphertexts for different receivers. This helps the owner to forward the message to different receivers dynamically, without sharing keys between the owner and the receiver.

Sharing a message into many storage servers also provides reliable access to the data. Since t out of n shares are sufficient to construct the original message, the owner can retrieve the message back even some storage servers fails. Similarly, this scheme can produce a correct ciphertext with only t correct partial ciphertexts even if some storage servers are malicious.

For security of our scheme, we consider both semi-honest and malicious adversaries. We show that our basic scheme is semantically secure against a semi-honest adversary. The message is kept secure between storage servers since the message is shared between them. Any collusion of less than t storage servers learns no information about the message. By adding non-interactive zero-knowledge proof tags to the exchanged messages, we show that the scheme is secure against malicious adversaries.

This scheme is efficient and scalable, since it can be executed in the non-interactive setting. This scheme saves the bandwidth between the storage system and the receiver, since only one ciphertext is transfered to the receiver.

Chapter 2

System Model

Let $\mathbb{G}_q$ denote an order-q multiplicative group and g be a generator of $\mathbb{G}_q$. In this paper, all group computations are over $\mathbb{G}_q$ and the corresponding exponent arithmetic is done in $\mathbb{Z}_q$.

2.1 Public Key Encryption for a Shared Message

A Public Key Encryption scheme for a Shared Message (PKESM) scheme achieves the following properties. First, it can be applied when a message is shared among a set of storage servers. Second, the shared message can be dynamically forwarded to different receivers. Third, the message is kept private during and after the forwarding process. A PKESM scheme is composed of the following parts:

$\mathsf{Setup}(\ell, t)$. Given security parameters (ℓ, t), this algorithm outputs public encryption key pairs (y, x) for receiver.

Deal(M). This algorithm takes a message M and outputs message shares ($m_1, \ldots, m_n$) for n storage servers $P_1 \ldots P_n$.

PartialEnc(y, m_i). With a public key y of the receiver and a message share m_i, this algorithm encrypts the message share by y and outputs a partial ciphertext σ_i.

Combine(S). With a set S of at least t partial ciphertexts, this algorithm combines the partial ciphertexts into a valid ciphertext C, which can only be correctly decrypted by the designated receiver.

2.2 Security model

A secure PKESM scheme must satisfy the following requirements:

Correctness. For any message M, given t correct partial ciphertexts, the function Combine(S) really outputs a valid ciphertext C. We say that C is valid if it can be decrypted to M by the receiver only.

Privacy. Given at most $t - 1$ message shares and all partial ciphertexts, no probabilistic polynomial-time adversary can learn any information about the message M.

Under the semantic-security notation [16], we formally define the privacy property in the following IND-CPA game between the adversary $\mathcal{A}$ and the challenger.

Init: Both the adversary and the challenger are given security parameters (ℓ, t) as input. The adversary outputs a corruption set CS of storage servers that he wants to corrupt, where $|\mathrm{CS}| < t$.

Setup: The challenger runs $\mathsf{Setup}(\ell, t)$ to construct the key pair (y, x) for the receiver. The public key y is given to the adversary.

Challenge: The adversary chooses two different messages M_0, M_1, and sends them to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$ and runs $\mathsf{Deal}(M_b)$. Then the challenger gives the message shares of all corrupted storage server to the adversary. Next, the challenger runs $\mathsf{PartialEnc}(y, m_i)$ for $1 \leq i \leq n$ to produce all partial ciphertexts. The challenger sends all partial ciphertexts to the adversary. Then challenger runs $\mathsf{Combine}(S)$ for a set S of t partial ciphertexts, and sends the combined ciphertext to the adversary.

Guess: The adversary $\mathcal{A}$ outputs a guess b' and wins the game if $b' = b$.

We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_{\ell,\mathsf{t}}^{\mathsf{PKESM}}(\mathcal{A}) = |\Pr[b' = b] - \frac{1}{2}|$.

Definition 1. *We say that a public key encryption scheme for a shared message is secure if for any probabilistic polynomial time Turning machine $\mathcal{A}$, sufficiently large ℓ, and (t, n) with $0 < t \leq n$, $\mathsf{Adv}_{\ell,\mathsf{t}}^{\mathsf{PKESM}}(\mathcal{A})$ is negligible.*

2.3 Security Assumption

The security of our scheme is based on the Decisional Diffie-Hellman (DDH) assumption [11]. Given a multiplicative group $\mathbb{G}_q$ of prime order q, $|q| = \ell$, and a generator $g \in \mathbb{G}_q$, the DDH assumption holds if the two distributions $D_0 = (g^\alpha, g^\beta, g^{\alpha\beta})$, $D_1 = (g^\alpha, g^\beta, g^\gamma)$ are computationally indistinguishable for random α, β, γ in $\mathbb{Z}_q$. Given a probabilistic polynomial-time adversary $\mathcal{A}$, we define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_\ell^{\mathsf{DDH}}(\mathcal{A}) = |\Pr[\mathcal{A}(D_b) = b] - \frac{1}{2}|$

Chapter 3

Preliminaries

ElGamal Encryption System. ElGamal encryption is a probabilistic public key cryptosystem [13]. The private key x is randomly chosen from $\mathbb{Z}_q$, and the public key is $y = g^x$. A message $M \in \mathbb{G}_q$ is encrypted as

$$(C_1, C_2) = (g^r, M(y)^r),$$

where r is a random element in Z_q. The decryption works as following:

$$\frac{C_2}{(C_1)^x} = \frac{M(y^r}{(g^r)^x} = M$$

Secret Sharing Scheme. A secret sharing scheme is used to share a secret value into shares such that one can only recover the secret value with t or more shares, where t is a threshold value. We use this technology to achieve the data privacy in the distributed storage system. If the message owner wants to deal a message M into a set of storage servers $P_1, \ldots, P_n$, he runs the following algorithm:

Secret_Sharing(M, t, n)

1. The owner chooses t random values $s_1, \ldots, s_t$ in $\mathbb{Z}_q$.

2. The owner constructs a polynomial $f(x) = s_t x^t + \cdots + s_1 x + 1 \pmod{q}$.

3. The owner computes message shares $m_i = M^{f(i)}$ for $1 \leq i \leq n$.

Let the output of $\mathsf{Secret\_Sharing}(M, t, n)$ be $(m_1, \ldots, m_n)$. The owner sends the message share m_i to P_i over a secure channel for $1 \leq i \leq n$. Note that less than t shares give no information about M in this scheme. On reconstruction, given t shares $m_{i_1}, \ldots, m_{i_t}$, the combiner computes Lagrange coefficients

$$\lambda_j = \prod_{1 \leq k \leq t, k \neq j} \frac{-i_k}{i_j - i_k} \bmod q$$

And, the message M is

$$\begin{aligned} \prod_{j=1}^{t} m_{i_j}^{\lambda_j} &= M^{\sum_{j=1}^{t} \lambda_j f(i_j)} \\ &= M \end{aligned}$$

Chapter 4

Public Key Encryption Scheme for a Shared Message

In this chapter, we present our public key encryption scheme for a shared message. Basically, the owner performs Deal to share the message into a group of storage servers in a distributed storage system. If the owner wants to send the message to a receiver, he just sends a command to the distributed storage system to perform the PKESM scheme. Then each storage server performs ParitialEnc and outputs a partial ciphertext independently. t or more partial ciphertexts can be combined into a valid ciphertext by Combine. The message is encrypted by the receiver's public key, and no information is leaked. We provide a detailed description about this protocol as follows:.

Setup(ℓ, t). Given security parameters (ℓ, t), set a multiplicative group $\mathbb{G}_q$ with $|q| \geq \ell$ and a generator $g \in \mathbb{G}_q$. Generate a secret key $x \in \mathbb{Z}_q$ for each receiver and set the receiver's public key as $y = g^x$.

Deal(M). Given a message $M \in \mathbb{G}_q$, the owner deals M for n storage servers $P_1, \ldots, P_n$. First, the owner runs $\mathsf{Secret\_Sharing}(M, t, n)$ and gets the output $(m_1, \ldots, m_n)$. Then the owner sends m_i to P_i via a secure channel.

$\mathsf{PartialEnc}(y, m_i)$. Given a receiver's public key y and a message share m_i, the storage server P_i randomly chooses a number $r_i \in \mathbb{Z}_q$ and outputs a partial ciphertext

$$\sigma_i = (g^{r_i}, m_i y^{r_i}).$$

$\mathsf{Combine}(S = \{\sigma_{i_1}, \ldots, \sigma_{i_t}\})$. Given a set S of t partial ciphertexts for a receiver, with $\sigma_{i_j} = (C_{i_j,1}, C_{i_j,2})$, the combiner outputs the ciphertext

$$C = (\prod_{j=1}^{t}(C_{i_j,1})^{\lambda_j}, \prod_{j=1}^{t}(C_{i_j,2})^{\lambda_j}), \text{ where } \lambda_j = \prod_{1 \le k \le t, k \ne j} \frac{-i_k}{i_j - i_k} \bmod q$$

To see this really produces a valid ciphertext, we have

$$\begin{aligned} \prod_{j=1}^{t}(C_{i_j,1})^{\lambda_j} &= \prod_{j=1}^{t}(g^{r_{i_j}})^{\lambda_j} \\ &= g^{\hat{r}}, \text{ for } \hat{r} = \sum_{j=1}^{t} r_{i_j}\lambda_j \end{aligned}$$

$$\begin{aligned} \prod_{j=1}^{t}(C_{i_j,2})^{\lambda_j} &= \prod_{j=1}^{t}(m_{i_j} y^{r_{i_j}})^{\lambda_j} \\ &= M^{\sum_{j=1}^{t} f(i_j)\lambda_j}(y)^{\hat{r}} \\ &= M^{f(0)}(y)^{\hat{r}} = M \cdot y^{\hat{r}} \end{aligned}$$

This algorithm can be executed without any secret information. Thus, every one can play the role of the combiner.

4.1 Security Analysis

In this chapter we prove the semantic security of our protocol. We prove the IND-CPA security against a static, semi-honest adversary, under the DDH assumption.

Theorem 1. *Suppose that the DDH assumption holds for $\mathbb{G}_q$. Then the public key encryption scheme for a shared message is semantically secure against a static and semi-honest adversary.*

Proof. We assume that a probabilistic polynomial time adversary $\mathcal{A}$ breaks our scheme with a non-negligible advantage. We build a poly-time algorithm $\mathcal{R}$ to break the DDH assumption. $\mathcal{R}$ acts as a challenger to $\mathcal{A}$ in the PKESM game. The following is the detailed description.

Init: Both the adversary $\mathcal{A}$ and $\mathcal{R}$ are given security parameters (ℓ, t) as input. Algorithm $\mathcal{R}$ is given $(\mathbb{G}_q, g)$ and a DDH challenge $(g^\alpha, g^\beta, g^\gamma)$ from the DDH challenger, where $|q| = \ell$. The adversary $\mathcal{A}$ outputs a corruption set $\text{CS} = \{P_{i_1}, P_{i_2}, \ldots, P_{i_{t-1}}\}$ of $t-1$ storage servers.

Setup: $\mathcal{R}$ sets $y = g^\alpha$ as the receiver's public key and sends it to $\mathcal{A}$.

Challenge: The adversary $\mathcal{A}$ chooses two different messages $M_0, M_1 \in \mathbb{G}_q$ and sends them to $\mathcal{R}$. $\mathcal{R}$ chooses a random bit $b \in_R \{0, 1\}$ and generates the message shares $(m_{b,1}, m_{b,2}, \ldots, m_{b,n})$ by running $\mathsf{Deal}(M_b)$. $\mathcal{R}$ sends the message $m_{b,i}$ to $\mathcal{A}$ for $P_i \in \text{CS}$.

Then, $\mathcal{R}$ chooses n random numbers $a_i \in \mathbb{Z}_q$ for $1 \leq i \leq n$, and generates the partial ciphertext of P_i, $P_i \notin$ CS

$$\sigma_{b,i} = ((g^{\beta})^{a_i}, m_{i,b}(g^{\gamma})^{a_i})$$

For corrupted storage server $P_i \in$ CS, $\mathcal{R}$ generate the partial ciphertexts by PartialEnc and sends the random numbers used in the PartialEnc to the adversary $\mathcal{A}$. $\mathcal{R}$ sends all partial ciphertexts to the adversary.

Guess: The adversary $\mathcal{A}$ outputs a guess b'. Then $\mathcal{R}$ replies the DDH challenger with $\gamma = \alpha\beta$ if $b' = b$. Otherwise, $\mathcal{R}$ replies $\gamma =$`random`.

We show that $\mathcal{A}$ cannot guess b with a non-negligible advantage from the partial ciphertexts or the combined ciphertext. If $\gamma = \alpha\beta$, the view of $\mathcal{A}$ is identical to the real execution. If γ is a random value, the view of $\mathcal{A}$ in the case of $\mathcal{R}$ dealing M_0 is identical to the case of $\mathcal{R}$ dealing M_1, since for given $C_{i,2} = (m_{i,b})g^{\gamma a_i}$, there exists another γ' such that $(m_{i,1-b})g^{\gamma' a_i} = C_{i,2}$. Thus, we have

$$
\begin{aligned}
& \mathsf{Adv}_{\ell}^{\mathsf{DDH}}(\mathcal{R}) \\
= \; & |\Pr[\gamma = \alpha\beta]\Pr[b' = b|\gamma = \alpha\beta] + \Pr[\gamma \text{ is } \texttt{random}]\Pr[b' \neq b|\gamma \text{ is } \texttt{random}] - \frac{1}{2}| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] + \Pr[b' \neq b|\gamma \text{ is } \texttt{random}] - 1| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] - \Pr[b' = b|\gamma \text{ is } \texttt{random}]| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] - \frac{1}{2}| \\
\geq \; & \frac{1}{2}(\mathsf{Adv}_{\ell,\mathsf{t}}^{\mathsf{PKESM}}(\mathcal{A}))
\end{aligned}
$$

$\square$

Chapter 5

Against Malicious Adversaries

We have proved that the basic scheme is secure against the semi-honest adversary. Nevertheless, a malicious adversary may send "wrong" partial ciphertexts from corrupted storage servers. The receiver will get the wrong message. Therefore, the combiner has to verify each partial ciphertext. We say a PKESM scheme is secure in the malicious adversary model, if this scheme satisfies the previous *correctness* and *privacy* properties, and additionally the following *robustness* property.

> *Robustness.* For any message M, if at least t partial ciphertexts pass the verification performed by the combiner, the scheme outputs a valid ciphertext of M.

The combiner is allowed to verify the partial ciphertexts, but does not know the message shares. Otherwise, the combiner will get the message M if he collects more than t message shares. In this chapter, we propose a scheme of verifying the partial ciphertext without knowing the message share. To achieve this, we need a public key pair (pk_i, sk_i) for each storage server P_i, and a zero-knowledge proof for discrete logarithm.

5.1 Zero-Knowledge Proof for Discrete Logarithm

We present a proof of knowledge of discrete logarithm to show that $z = \log_g X = \log_h Y$ [7].

Let $X = g^z$ and $Y = h^z$ for two generators $g, h \in \mathbb{G}_q$. Alice executes the following algorithm to convince Bob that $\log_g X = \log_h Y$ without leaking z to Bob.

`ProofDLEq`(g, h, X, Y)

1. Alice chooses a random number $d \in \mathbb{Z}_q$ and sends $a_1 = g^d$, $a_2 = h^d$ to Bob.

2. Bob sends a challenge $c \in \mathbb{Z}_q$ to Alice.

3. Alice responds with $R = d - cz \bmod q$.

4. Bob checks that $a_1 = g^R X^c$ and $a_2 = h^R Y^c$.

This protocol is sound under the Discrete Logarithm assumption. Note that if Alice knows the challenge c in advance, she can cheat Bob with a random R, and outputs $a_1 = g^R X^c$, $a_2 = h^R Y^c$.

This proof needs three rounds interactions. We can apply the Fiat-Shamir heuristic to transform this proof into a one round, non-interactive protocol `NIProofDLEq` [12]. In the non-interactive setting, Alice replaces a random challenge with $c = H(g||h||X||Y||a_1||a_2)$, where $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a collision-resistant hash function. Bob verifies that $c = H(g||h||X||Y||g^R X^c||h^R Y^c)$. If yes, Bob believes that $\log_g X = \log_h Y$.

Let the output of $\texttt{NIProofDLEq}(g, h, X, Y)$ be (c, R). We use this zero-knowledge proof to verify the partial ciphertext without knowing the message share.

5.2 Main Construction

Intuitively, when a storage server P_i receives a message share m_i, he commits m_i by using his secret key sk_i. The owner then verifies whether the commitment is valid. When storage server outputs a partial ciphertext, he has to attach a proof tag. The combiner uses the commitment and the proof tag to verify the partial ciphertext.

The following algorithms are used to verify whether the partial ciphertexts are valid.

$\mathsf{Setup}(\ell, t)$. Given security parameters (ℓ, t), set a multiplicative group $\mathbb{G}_q$ with $|q| \geq \ell$ and a generator $g \in \mathbb{G}_q$. Generate a secret key $x \in \mathbb{Z}_q$ for each receiver and set the receiver's public key as $y = g^x$. Generate a key pair (pk_i, sk_i) for each storage server P_i.

$\mathsf{Deal}(M)$. Given a message $M \in \mathbb{G}_q$, the owner deals M for n storage servers $P_1, \ldots, P_n$ by $\mathsf{Secret\_Sharing}(M, t, n)$.

Commit(sk_i, m_i). Given a message share m_i, storage server P_i commits the message share m_i as $\theta_i = m_i^{sk_i}$ with his secret key sk_i. In addition, P_i sends $V_i =$`NIProofDLEq`(g, m_i, pk_i, θ_i) to the owner.

ComVerify$(pk_i, m_i, \theta_i, V_i)$. Given a message share m_i, a commitment θ_i and a proof tag $V_i =$ `NIProofDLE1` (g, m_i, pk_i, θ_i), the owner verifies the proof tag. If the commitment is correct, the owner signs on the commitment θ_i.

PartialEnc(y, m_i). Given a receiver's public key y and a message share m_i, the storage server P_i randomly chooses a number $r_i \in \mathbb{Z}_q$ and outputs a partial ciphertext

$$\sigma_i = (g^{r_i}, m_i y^{r_i}).$$

ProofPC$(y, pk_i, sk_i, r_i, \sigma_i = (C_{i,1}, C_{i,2}), \theta_i)$. Given the receiver's public key y, a key pair (pk_i, sk_i) for P_i, a random number r_i used in PritialEnc(y, m_i), a partial ciphertext σ_i, and a commitment θ_i, the storage server P_i generates a proof for the partial ciphertext σ_i. P_i computes two values $(y_1 . y_2) = (g^{sk_i r_i}, y^{sk_i r_i})$ and outputs the following proofs:

$$(y_1, y_2)$$

$$\texttt{NIProofDLEq}(g, C_{i,1}, pk_i, y_1)$$

$$\texttt{NIProofDLEq}(g, y, y_1, y_2)$$

$$\texttt{NIProofDLEq}(g, C_{i,2}, pk_i,\ \theta_i \cdot y_2)$$

$$\begin{tikzcd}
 & C_{i,1} = g^{r_i} \arrow[d, leftrightarrow] & & \\
pk_i = g^{sk_i} \arrow[r, leftrightarrow] & y_1 = g^{sk_i r_i} \arrow[r, leftrightarrow] & y_2 = y^{sk_i r_i} \arrow[dr, leftrightarrow] & \\
 & & & C_{i,2} = m_i \cdot y^{r_i} \\
 & & \theta_i = m_i^{sk_i} \arrow[ur, leftrightarrow] &
\end{tikzcd}$$

Figure 5.1: The relationship between partial ciphertext σ_i, the public key pk_i of storage server P_i, and the commitment θ_i.

Since the commitment θ_i has been verified by the owner, P_i just has to prove the validity of $y_2 = y^{sk_i r_i}$. This can be achieved by the public key $pk_i = g^{sk_i}$ of P_i and $C_{i,1} = g^{r_i}$. First, P_i shows that $g^{sk_i r_i}$ and pk_i has the same exponent sk_i by the first proof tag. Then P_i shows that $g^{sk_i r_i}$ and $y^{sk_i r_i}$ has the same exponent by the second proof tag. Finally, P_i proves the correctness of $C_{i,2}$ by the commitment θ_i, y_2 and the last proof tag. The relationship between these proofs are shown in the figure 5.1.

$\mathsf{PCVerify}(\sigma_i = (C_{i,1}, C_{i,2}), \theta_i, ZK_i)$. Given a partial ciphertext σ_i, a commitment θ_i signed by the owner and correctness proof ZK_i as

$$(y_1, y_2) = (g^{sk_i r_i}, y^{sk_i r_i})$$

$$(c_1, R_1) = \texttt{NIProofDLEq}(g, C_{i,1}, pk_i, y_1)$$

$$(c_2, R_2) = \texttt{NIProofDLEq}(g, y, y_1, y_2)$$

$$(c_3, R_3) = \texttt{NIProofDLEq}(g, C_{i,2}, pk_i,\ \theta_i \cdot y_2),$$

the combiner verifies that

$$c_1 = H(g||C_{i,1}||pk_i||y_1||g^{R_1} pk_i^{c_1}||(C_{i,1})^{R_1} y_1^{c_1})$$

$$c_2 = H(g||y||y_1||y_2||g^{R_2}y_1^{c_2}||y^{R_2}y_2^{c_2})$$

$$c_3 = H(g||C_{i,2}||pk_i||\theta_i \cdot y_2||g^{R_3}pk_i^{c_3}||(C_{i,2})^{R_3}(\theta_i \cdot y_2)^{c_3})$$

$\mathsf{Combine}(S = \{\sigma_{i_1}, \ldots, \sigma_{i_t}\})$. Given a set S of t partial ciphertexts passed the $\mathsf{PCVerify}$, with $\sigma_{i_j} = (C_{i_j,1}, C_{i_j,2})$, the combiner outputs the ciphertext

$$C = (\prod_{j=1}^{t}(C_{i_j,1})^{\lambda_j}, \prod_{j=1}^{t}(C_{i_j,2})^{\lambda_j}), \text{ where } \lambda_j = \prod_{1\le k\le t, k\ne j}\frac{-i_k}{i_j - i_k} \bmod q$$

5.3 Security Analysis

We give the proof of our scheme against malicious adversaries in the random oracle model in this chapter. The proof is similar to the proof in the basic scheme. We assume that a probabilistic polynomial time adversary $\mathcal{A}$ breaks our scheme with a non-negligible advantage. We build an algorithm $\mathcal{R}$ to beak the DDH assumption. $\mathcal{R}$ simulates the algorithms in the basic scheme, and also simulates the three algorithms Commit, $\mathsf{ComVerify}$ and $\mathsf{ProofPC}$. $\mathcal{R}$ has to convince the adversary $\mathcal{A}$ in $\mathsf{PCVerify}$ even $\mathcal{R}$ does not know the discrete logarithm (α, β, γ) of DDH challenge $(g^\alpha, g^\beta, g^\gamma)$.

In the random oracle model [3], the hash function H is modeled as a random oracle. The adversary $\mathcal{A}$ makes hash queries of H at any time during his attack. The simulator $\mathcal{R}$ maintains a list H-List to record the result of all hash queries. $\mathcal{R}$ handles the hash queries as follows:

On receiving a query w to H:

1. If (w, e) exists in H-List for some e, $\mathcal{R}$ outputs e as $H(w)$.

2. Otherwise, $\mathcal{R}$ randomly chooses $e \in \mathbb{Z}_q$, adds (w, e) into H-List, and outputs e as $H(w)$.

The proof is described below.

Init: Both of $\mathcal{A}$ and $\mathcal{R}$ are given security parameters (ℓ, t) as input. Algorithm $\mathcal{R}$ is given $(\mathbb{G}_q, g)$ and a DDH challenge $(g^\alpha, g^\beta, g^\gamma)$ from the DDH challenger, where $|q| = \ell$. The adversary $\mathcal{A}$ outputs a corruption set CS $= \{P_{i_1}, P_{i_2}, \ldots, P_{i_{t-1}}\}$ of $t-1$ storage servers.

Setup: $\mathcal{R}$ runs $\mathsf{Setup}(\ell, t)$ to construct the key pairs (pk_i, sk_i) for storage servers P_i, $1 \leq i \leq n$. $\mathcal{R}$ sends pk_i to $\mathcal{A}$, $1 \leq i \leq n$, and sends sk_i to $\mathcal{A}$ for $P_i \in$ CS. $\mathcal{R}$ sets $y = g^\alpha$ as the receiver's public key and sends y to $\mathcal{A}$. $\mathcal{R}$ gives the control permission of corrupted storage servers to $\mathcal{A}$.

Challenge: The adversary $\mathcal{A}$ chooses two different messages $M_0, M_1 \in \mathbb{G}_q$ and sends them to $\mathcal{R}$. $\mathcal{R}$ chooses a random bit $b \in \{0, 1\}$ and generates the message shares $(m_{b,1}, m_{b,2}, \ldots, m_{b,n})$ by running $\mathsf{Deal}(M_b)$. $\mathcal{R}$ sends the message $m_{b,i}$ to $\mathcal{A}$ for $P_i \in$ CS.

Then $\mathcal{R}$ runs the algorithm $\mathsf{Commit}(sk_i, m_{b,i})$ normally for $P_i \notin$ CS, and output `valid` for this commitments in $\mathsf{ComVeify}$. For the commitments coming from corrupted storage servers, $\mathcal{R}$ runs $\mathsf{ComVeify}$ normally and outputs `valid` or `invalid`.

Next, for $P_i \notin$ CS, $\mathcal{R}$ chooses a random number $a_i \in \mathbb{Z}_q$. $\mathcal{R}$ generates the partial ciphertext of P_i as

$$\sigma_{b,i} = ((g^{\beta})^{a_i}, m_{i,b}(g^{\gamma})^{a_i})$$

Furthermore, $\mathcal{R}$ produces the proof tags for $\sigma_{b,i}$. Since $\mathcal{R}$ does not know the random value βa_i, $\mathcal{R}$ cannot construct the proof normally. Instead, $\mathcal{R}$ inserts an entry into the H-list to decide the challenge in advance. Then $\mathcal{R}$ can produce correct proof tags for $\sigma_{b,i} = (C_{b,i,1}, C_{b,i,2})$. The following is the detailed description.

1. $\mathcal{R}$ sets $y_1 = (C_{b,i,1})^{sk_i}$, $y_2 = (g^{\gamma a_i})^{sk_i}$.

2. $\mathcal{R}$ randomly chooses $c_1, R_1 \in \mathbb{Z}_q$.

3. Let $w_1 = (g||C_{b,i,1}||pk_i||y_1||g^{R_1}pk_i^{c_1}||(C_{b,i,1})^{R_1}(y_1)^{c_1})$. If there exists an entry (w, e) in H-List with that $w = w_1$, go back to step 2. Otherwise, $\mathcal{R}$ adds (w_1, c_1) to H-List.

4. $\mathcal{R}$ generates $(c_2, R_2), (c_3, R_3)$ similar to step 2, 3 with

$$w_2 = (g||y||y_1||y_2||g^{R_2}y_1^{c_2}||y^{R_2}y_2^{c_2})$$

$$w_3 = (g||C_{b,i,2}||pk_i||\theta_i \cdot y_2||g^{R_3}pk_i^{c_3}||(C_{b,i,2})^{R_3}(\theta_i \cdot y_2)^{c_3})$$

5. $\mathcal{R}$ outputs $(y_1, y_2, c_1, R_1, c_2, R_2, c_3, R_3)$ as the proof .

Guess: The adversary $\mathcal{A}$ outputs a guess b'. Then $\mathcal{R}$ replies the DDH challenger with $\gamma = \alpha\beta$ if $b' = b$. Otherwise, $\mathcal{R}$ replies $\gamma =$`random`.

When $\mathcal{A}$ queries the hash oracle in the verification, he will get the value pre-inserted by $\mathcal{R}$. Without the hash oracle, $\mathcal{A}$ cannot generate the hash value and he cannot execute $\mathsf{PCVerify}$. Thus, $\mathcal{R}$ passes the verification.

We analyze the security of privacy in the following.

Theorem 2. *(Hiding of commitment). Given two message shares m_0, m_1, a public key $pk = g^{sk}$ and a commitment $\theta = m_b^{sk}$ for a random bit b, it's hard to distinguish b for any probabilistic polynomial-time Turing machine adversaries.*

Proof. If there is an adversary $\mathcal{A}'$ can decide $b = 0$ or $b = 1$ with non-negligible advantage, we define the advantage as $\mathsf{Adv}_{\ell}^{\mathsf{commit}}(\mathcal{A}')$. We construct a algorithm $\mathcal{R}'$ to break the DDH assumption with non-advantage with $\mathcal{A}'$.

Given DDH challenge $(g^{\alpha}, g^{\beta}, g^{\gamma})$, $\mathcal{R}'$ randomly chooses $b \in \{0, 1\}$. $\mathcal{R}'$ sets $m_b = g^{\alpha}$ and randomly chooses $m_{1-b} \in \mathbb{G}_q$. $\mathcal{R}'$ sets $pk = g^{\beta}$ and $\theta = g^{\gamma}$. $\mathcal{R}'$ sends (m_0, m_1, pk, θ) to $\mathcal{A}'$. If $\mathcal{A}'$ returns b' with $b' = b$, $\mathcal{R}$ returns the DDH challenger with $\gamma = \alpha\beta$. Otherwise, $\mathcal{R}'$ returns $\gamma =$`random`.

If γ is random, the distribution of (m_0, m_1, pk, θ) for $b = 0$ is equal to the distribution for $b = 1$. Thus we have

$$\begin{aligned}
& \mathsf{Adv}_{\ell}^{\mathsf{DDH}}(\mathcal{R}') \\
= \; & |\Pr[\gamma = \alpha\beta]\Pr[b' = b|\gamma = \alpha\beta] + \Pr[\gamma \text{ is } \texttt{random}]\Pr[b' \neq b|\gamma \text{ is } \texttt{random}] - \frac{1}{2}| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] + \Pr[b' \neq b|\gamma \text{ is } \texttt{random}] - 1| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] - \Pr[b' = b|\gamma \text{ is } \texttt{random}]| \\
= \; & \frac{1}{2}|\Pr[b' = b|\gamma = \alpha\beta] - \frac{1}{2}| \\
\geq \; & \frac{1}{2}(\mathsf{Adv}_{\ell}^{\mathsf{commit}}(\mathcal{A}'))
\end{aligned}$$

Therefore, we can break the DDH assumption if an adversary can decide $b = 0$ or $b = 1$ with non-negligible advantage.

$\square$

Secondly, we prove that it's hard to distinguish two messages M_0 and M_1 from the partial ciphertexts $\sigma_{b,i}$, $1 \leq i \leq n$. This is similar to the proof in the basic scheme. If $\gamma = \alpha\beta$, the view of $\mathcal{A}$ is identical to the real execution. If γ is a random value, $\mathcal{A}$ cannot distinguish that $\mathcal{R}$ was dealing M_0 or M_1 since the two views are identical. Thus, we have

$$\mathsf{Adv}_{\ell}^{\mathsf{DDH}}(\mathcal{R}) \geq \frac{1}{2}(\mathsf{Adv}_{\ell,\mathsf{t}}^{\mathsf{PKESM}}(\mathcal{A}))$$

By Theorems 1 and 2, we have that the advantage of $\mathcal{R}$ breaking the DDH assumption is bounded by:

$$2 \cdot \mathsf{Adv}_{\ell}^{\mathsf{DDH}}(\mathcal{R}) \geq \frac{1}{2}\mathsf{Adv}_{\ell}^{\mathsf{commit}}(\mathcal{A}) + \frac{1}{2}(\mathsf{Adv}_{\ell,\mathsf{t}}^{\mathsf{PKESM}}(\mathcal{A}))$$

Algorithm	# of modular exponentiation	Executed by
Deal	n	Owner
Commit	3	Storage server
ComVerify	$4n$	Owner
PartialEnc	2	Storage server
ProofPC	8	Storage server
PCVerify	12n	Combiner
Combine	2t	Combiner

Table 5.1: Number of modular exponentiation performed in each algorithm.

For robustness, the security is based on the soundness of the zero-knowledge proof. It has been shown that the proof of discrete logarithm is sound [7]. Thus, it is hard for an adversary to cheat the owner with a wrong commitment with non-negligible probability, or to cheat cheating the combiner with a wrong partial ciphertext.

5.4 Performance Analysis

We measure the performance in the number of modular exponentiations. Generating a proof of discrete logarithm needs 2 modular exponentiations, and verifying the proof needs 4 modular exponentiations. For n storage servers, the following table lists the number of modular exponentiations executed in each algorithm.

Chapter 6

Discussion

We construct the PKESM scheme by a secret sharing scheme and the ElGamel encryption scheme. It is possible to construct a PKESM scheme by other homomorphic public key encryption schemes, such as RSA[23] or Paillier[22]. In the RSA encryption, the order of the group is not public. Thus, it is hard to compute the Lagrange coefficients λ_j in Combine since the combiner cannot compute the inverse in a RSA group. To avoid this, we can multiply λ_j with $\Delta = n!$. The value $\Delta\lambda_j$ is an integer and no inverse computation is required. It can be further referenced to [14]. But there are some other issues. In the malicious adversary mode, a zero-knowledge proof scheme should be designed for verifying an RSA partial ciphertext.

Chapter 7

Conclusions

We presented a non-interactive public key encryption scheme which is performed on a shared plaintext. This scheme can be used to forward a shared message to different receivers without a pre-installed key between the owner and the receiver. We prove its IND-CPA security under the DDH assumption. This paper is the first one that performs encryption in such model.

Bibliography

[1] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI* (2002).

[2] Ateniese, G., Fu, K., Green, M., and Hohenberger, S. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS* (2005), The Internet Society.

[3] Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security* (1993), pp. 62–73.

[4] Boneh, D., Boyen, X., and Halevi, S. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA* (2006), D. Pointcheval, Ed., vol. 3860 of *Lecture Notes in Computer Science*, Springer, pp. 226–243.

[5] Canetti, R., and Hohenberger, S. Chosen-ciphertext secure proxy re-encryption. In *ACM Conference on Computer and Communications*

Security (2007), P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds., ACM, pp. 185–194.

[6] Cattaneo, G., Catuogno, L., Sorbo, A. D., and Persiano, P. The design and implementation of a transparent cryptographic file system for unix. In *USENIX Annual Technical Conference, FREENIX Track* (2001), C. Cole, Ed., USENIX, pp. 199–212.

[7] Chaum, D., and Pedersen, T. P. Wallet databases with observers. In *CRYPTO* (1992), E. F. Brickell, Ed., vol. 740 of *Lecture Notes in Computer Science*, Springer, pp. 89–105.

[8] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability* (2000), H. Federrath, Ed., vol. 2009 of *Lecture Notes in Computer Science*, Springer, pp. 46–66.

[9] Delerablée, C., and Pointcheval, D. Dynamic threshold public-key encryption. In *CRYPTO* (2008), D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, pp. 317–334.

[10] Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and Welch, B. The bayou architecture: Support for data sharing among mobile users. *Mobile Computing Systems and Applications, IEEE Workshop on 0* (1994), 2–7.

[11] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (1976), 644–654.

[12] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO* (1986), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer, pp. 186–194.

[13] Gamal, T. E. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31*, 4 (1985), 469–472.

[14] Gennaro, R., Halevi, S., Krawczyk, H., and Rabin, T. Threshold rsa for dynamic and ad-hoc groups. In *EUROCRYPT* (2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, pp. 88–107.

[15] Goh, E.-J., Shacham, H., Modadugu, N., and Boneh, D. Sirius: Securing remote untrusted storage. In *NDSS* (2003), The Internet Society.

[16] Goldwasser, S., and Micali, S. Probabilistic encryption. *J. Comput. Syst. Sci. 28*, 2 (1984), 270–299.

[17] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. Plutus: Scalable secure file sharing on untrusted storage. In *FAST* (2003), USENIX.

[18] Kistler, J. J., and Satyanarayanan, M. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst. 10*, 1 (1992), 3–25.

[19] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S. E., Eaton, P. R., Geels, D., Gummadi, R., Rhea, S. C., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. Y. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS* (2000), pp. 190–201.

[20] Libert, B., and Vergnaud, D. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography* (2008), R. Cramer, Ed., vol. 4939 of *Lecture Notes in Computer Science*, Springer, pp. 360–379.

[21] Miyamoto, T., and Kumagai, S. An optimal share transfer problem on secret sharing storage systems. In *ISPA* (2007), I. Stojmenovic, R. K. Thulasiram, L. T. Yang, W. Jia, M. Guo, and R. F. de Mello, Eds., vol. 4742 of *Lecture Notes in Computer Science*, Springer, pp. 371–382.

[22] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT* (1999), pp. 223–238.

[23] Rivest, R. L., Shamir, A., and Adleman, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (1978), 120–126.

[24] Rowstron, A. I. T., and Druschel, P. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP* (2001), pp. 188–201.

[25] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.

[26] Subbiah, A., and Blough, D. M. An approach for fault tolerant and secure data storage in collaborative work environments. In *StorageSS* (2005), V. Atluri, P. Samarati, W. Yurcik, L. Brumbaugh, and Y. Zhou, Eds., ACM, pp. 84–93.

[27] Waldman, M., Rubin, A. D., and Cranor, L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *In Proc. 9th USENIX Security Symposium* (2000), pp. 59–72.