

國立交通大學

資訊科學與工程研究所

碩士論文

具有固定大小搜尋符號之隱私的字串搜尋

Privacy Preserving String Matching
With a Constant Size Token

研究生：劉思維

指導教授：曾文貴 教授

中華民國 九十八年 六月

具有固定大小搜尋符號之隱私的字串搜尋
Privacy Preserving String Matching With a Constant Size Token

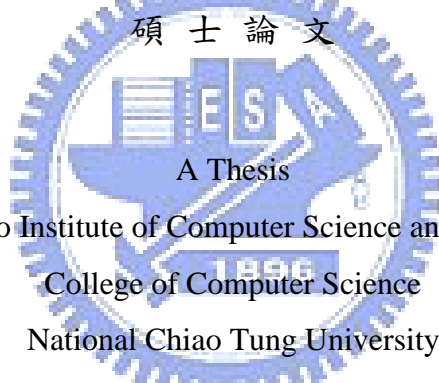
研究生：劉思維

Student：Szu-Wei Liu

指導教授：曾文貴

Advisor：Wen-Guey Tzeng

國立交通大學
資訊科學與工程研究所
碩士論文



Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

具有固定大小搜尋符號之隱私的字串搜尋

學生:劉思維

指導教授:曾文貴 博士

國立交通大學資訊科學與工程研究所

摘要

目前有許多密碼學上的方法能夠支援在加密資料上進行關鍵字的搜尋，但是較少相關的方法能夠支援更一般化的查詢方式，例如：字串搜尋。

在我們所提出的方法中，標籤代表加密過後的文件，搜尋符號則是代表用來搜尋文件的字串加密。根據我們所提出的方法，針對一份加密過後的文件當中，我們可以得知是否其包含一個特定的字串。並且在計算上難題的假設之下，我們可以證明我們的方法是安全的。

就我們所知，目前有一個相關的方法是針對在加密資料上進行字串搜尋，不過這個方法所需要的搜尋符號大小較大。在我們提出來的的方法中，我們可以將搜尋符號大小降低到固定大小，並且我們希望盡可能減少網路傳輸的通訊量以及減少使用者端的計算負擔。

關鍵字：具有隱私、字串搜尋、樣式比對、全文搜尋、關鍵字搜尋、同代像加密、隱私同態、RSA 假設

Privacy Preserving String Matching

With a Constant Size Token

Student: Szu-Wei Liu

Advisor: Dr. Wen-Guey Tzeng

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

Nowadays, many cryptographic schemes have been developed to achieve search on encrypted data, but most of them are keyword search. Few of schemes can support more general query like string matching on encrypted data.

In our protocol, the tag stands for the encrypted text, and the token stands for the encrypted pattern. We propose a scheme which can determine whether the encrypted text contains the dedicated pattern. Under the computational assumption, we can prove it secure.

As far as we know, there exists one protocol to our problem, but its token size is large. We can reduce the token size to the constant size, and we hope to reduce the communication size in the network and the computational time in the user side as far as possible.

Keywords: privacy preserving, string matching, pattern matching, full text search, keyword search, homomorphic encryption, privacy homomorphism, RSA assumption

誌 謝

本篇論文得以順利完成，都要感謝受到許多的幫助。

首先，我要感謝我的指導教授曾文貴教授，在這兩年求學過程中，老師給予我在研究上的指導與經驗上的分享，帶領我進入密碼學與資訊安全的領域，能夠有所收穫。老師嚴謹的態度，使我更加積極，不論是在課業或研究上，自己都認真付出，盡力作好。接著，我要感謝口試委員清大資工系孫宏民教授、交大資工系蔡錫鈞教授以及交大電機系黃育綸教授，在論文上給予我的寶貴建議，使得論文可以更加完善。

另外，我要感謝已經畢業的成康學長，在我就讀一年級的時候給予課業、研究與實驗室生活上的幫助，再加上最近透過電子郵件向他請教時，也提供論文方面許多寶貴經驗與協助。感謝實驗室的孝盈學姊，總是細心地協助解決研究上的疑惑與實驗室大小問題，還有煥宗與宣佐學長的幫忙。感謝同學韶瑩，在課業與研究上與他的討論過程都讓我倍感收穫，還有家宏、毅睿、證傑與政利，藉由同儕之間的切磋、鼓勵，大家一起進步。感謝學弟們與助理天心，大家一起聊天度過的歡樂時光，以及大大小小事情也都互相幫忙。感謝實驗室所有成員，大家兩年來的陪伴與幫忙。

最重要的，我要感謝我的家人，特別是爸爸與媽媽，一直以來給我的細心照料，無微不至，從小到大都是支持我的最大動力。還有我的女友佳玲，一路走來，幸好有妳時時鼓勵與幫忙，總是讓我能夠再次重拾信心與動力，真的很感謝妳的大力相助，希望妳也順利完成碩士學業。還有我的弟弟，偶爾與你之間天南地北的聊天對話，也是減壓的一大助力，希望你也順利完成碩士學業。

最後，我要感謝所有我曾經受到的幫助，以及交通大學與交大資工，提供優良的求學環境，讓我能夠盡情在其中完成學業，入寶山而滿載而歸。

謹以此論文獻給大家，一同分享喜悅。

Contents

Abstract in Chinese	i
Abstract	ii
Acknowledgement	iii
Contents	iv
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 Contribution	2
2 Preliminary	3
2.1 String Matching	3
2.1.1 The Naive String-Matching Algorithm	4
2.2 Privacy Homomorphism	5
2.2.1 RSA cryptosystem	6
2.2.2 ElGamal cryptosystem	7
3 Related Work	9



3.1	Framework	9
3.2	String Matching on Encrypted Data	12
3.3	Keyword Search on Encrypted Data	14
3.3.1	Secret-key Based Keyword Search	15
3.3.2	Public-key Based Keyword Search	19
3.3.3	String Matching with Keyword Search	21
3.4	Other Extensions	23
4	Proposed Protocol	24
4.1	Problem description	24
4.2	Heuristic Approaches	25
4.2.1	Scheme I: The Deterministic Scheme	26
4.2.2	Scheme II: The Randomized Scheme	26
4.2.3	Scheme III: The Slide Scheme	27
4.2.4	Scheme IV: The Final Scheme	28
4.3	Construction	29
4.3.1	Our Protocol	30
4.3.2	Correctness	33
4.3.3	Mask Exponent s	36
4.3.4	Mask Base Number r	36
5	Analysis	37
5.1	Security	37
5.1.1	RSA Assumption	37
5.1.2	Security Model	38
5.1.3	Security Reduction	38

5.2	Comparison	39
5.2.1	Time	39
5.2.2	Space	39
5.3	Avoidance	40
5.3.1	Small Token	40
5.3.2	Search Too Many Times	41
5.4	Discussion	41
5.4.1	False Positive	41
5.4.2	Exposure of Occurrence Position	41
5.4.3	Decryptable	42
6	Conclusion	43
	Bibliography	44



List of Tables

4.1	Character Space Encoding	30
4.2	Text Information	31
5.1	Protocol Space Comparison	39



List of Figures

2.1	Occurrences of Pattern	4
2.2	The Naive String-Matching Algorithm	5



Chapter 1

Introduction

Search on encrypted data has become popular in recent years. Because more and more application service providers have been increased, the users can process their data on the remote server for saving cost of storage and computation, just like “cloud computing”. One service provider supports storage, and the users can upload their data and download it in the future.

For the confidentiality, the data are encrypted before being uploaded to the server. But, if the user wants to search on encrypted file, the naive solution is to download the entire file, decrypt it, and then search on the local decrypted file. The naive solution requires much more cost of computation and communication, so the better solution is wondered.

Nowadays, there are many protocols proposed for this purpose. Usually, we have to construct the tag of each file and append the tag with the encrypted file to the remote server, and then tags will be searched in the future. Until now, most protocols concern on keyword search. The main limitation is that the user can only search the pre-defined “keywords” and can not search any pattern which occurs in the file for the general purpose. Sometimes the latter way is called “string matching”, “pattern

matching”, or “full text search”. It is natural, and we are interested to extend it to the encrypted data.

1.1 Contribution

Until now, there exists one protocol proposed by Oleshchuk[1]. It is designed directly for pattern matching on encrypted data, but the token size is large. Like Oleshchuk’s protocol, we use the same idea as the naive string-matching algorithm to develop a protocol which supports string matching on encrypted data, and the token size is constant. Furthermore, we can prove it secure under the computational assumption.



Chapter 2

Preliminary

In this chapter, we introduce the preliminaries which are used in our protocol. First, the problem of string matching is discussed, and then the naive string-matching algorithm is showed. Second, the idea of privacy homomorphism is discussed.

2.1 String Matching

The problem “String Matching” has been mostly studied in literature, and many programming languages or some utilities have it as the basic primitive function or the basic tool. Especially, the text-editing programs frequently use it. Here, most of definitions and examples follow [2].

Let Σ be the character space, e.g. $\{0, 1\}$ or $\{a, b, \dots, z\}$. The array of characters is called “string”. In other words, any string S can be expressed as an array $S[1 \dots n]$, and size of the array, n , is called length of the string S . A substring of the string S is expressed as $S[i \dots j]$ for all i, j such that $0 \leq i \leq j \leq n$.

The problem of string matching has two main roles like “text” T and “pattern” P . Both T and P are the string of finite length over the same character space. We

want to find all “occurrences” of the pattern P in the text T . Obviously, length of P must be smaller than or equal to length of T . The following figure shows that the pattern P occurrences at the position 2 of the text T .

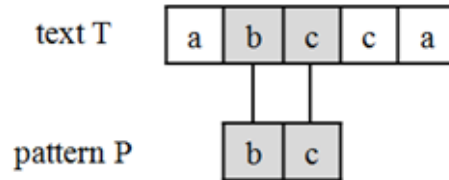


Figure 2.1: Occurrences of Pattern

2.1.1 The Naive String-Matching Algorithm

A natural approach to the string matching problem is to verify whether the pattern occurrences at any position of the text. Let the text be $T[1 \dots n]$ and the pattern be $P[1 \dots m]$ where $0 < m \leq n$. The naive string-matching algorithm is as follows.

- 1: **for** $i = 1$ to $n - m + 1$ **do**
- 2: **if** $T[i \dots i + m - 1] == P[1 \dots m]$ **then**
- 3: print i
- 4: **end if**
- 5: **end for**

There are two main operations like “comparing” and “sliding” in the naive string-matching algorithm. Comparing is happened on line 2 and requires to test whether the corresponding positions of the text and the pattern are same. After the test on line 2, sliding is happened as the text is shifted left one character and is compared with the same pattern repeatedly.

The following is an example.

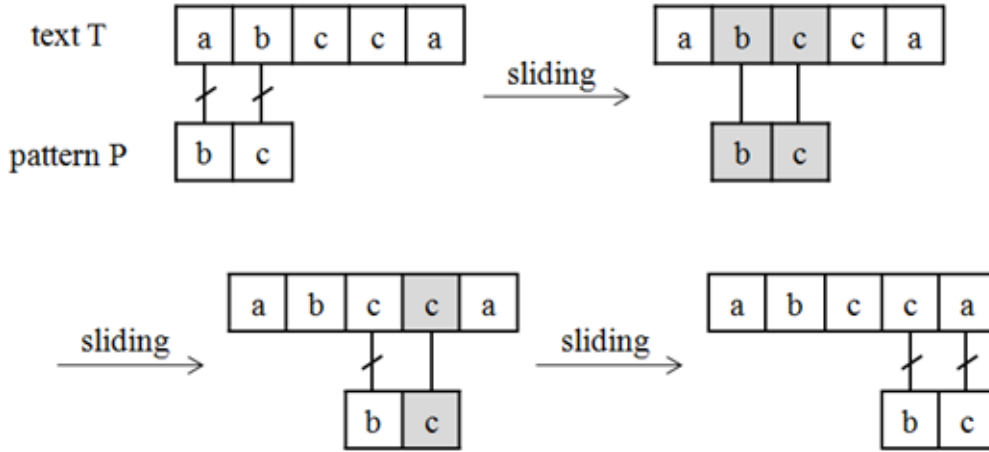


Figure 2.2: The Naive String-Matching Algorithm

Each test on line 2 is executed until some mismatch is found, so each comparing requires $O(m)$ times. The for-loop is executed $O(n - m + 1)$ times. Overall, the execution time of the naive string-matching algorithm is $O((n - m + 1)m)$.

There are more efficient ways to reduce the execution time with extra pre-processing. However, we focus on search on encrypted data like the naive way.

2.2 Privacy Homomorphism

“Privacy Homomorphism”(PH) is firstly proposed by Rivest et al.[3]. The encryption function called “privacy homomorphism” permits encrypted data to be operated without preliminary decryption of it.

For example, given two encryptions $E(m_1)$ and $E(m_2)$. There exist some operations like \otimes and \odot such that

$$E(m_1) \otimes E(m_2) = E(m_1 \odot m_2).$$

Sometimes an encryption scheme which has this property is also called a homomorphic encryption scheme.

The other similar idea is that given an encryption $E(m)$ and we can generate another different encryption $E(m')$ such that m and m' are meaningfully related if the encryption function is “malleable”[4].

For example, one-time pad is a perfect secure encryption scheme and malleable. Assume that there exists one ciphertext like $E_K(m) = K \oplus m = c$. We can get another ciphertext of plaintext $m \oplus r$ by $c \oplus r = (K \oplus m) \oplus r = K \oplus (m \oplus r) = E_K(m \oplus r)$.

The homomorphic or malleable encryption can support to search on encrypted data. On the other way, it may violate the integrity of data or authentication of signature. Here are some well-known encryption schemes which are privacy homomorphism.

2.2.1 RSA cryptosystem

RSA cryptosystem proposed by Rivest et al. is a public key cryptosystem and popular for the purposes of encryption and signature[5].

- Key Generation

Randomly choose two large and distinct primes p and q such that $|p| = |q|$.

Let $n = pq$. Choose random $e \in Z_{\phi(n)}^*$ and then compute $d = e^{-1} \bmod \phi(n)$.

Then the public key pk is (n, e) and the secret key sk is (n, d) .

- Encryption

Input a plaintext $m \in Z_n$ and output the ciphertext of m is

$$Enc_{pk}(m) = m^e \bmod n$$

- Decryption

Input a ciphertext $c \in Z_n$ and output the plaintext of c is

$$Dec_{sk}(c) = c^d \bmod n$$

Based on modular power, RSA is a homomorphic encryption. For example, assume that

$$Enc_{pk}(m_1) = m_1^e \bmod n$$

$$Enc_{pk}(m_2) = m_2^e \bmod n$$

Then, we can get a new ciphertext of $(m_1 \cdot m_2 \bmod n)$ as follows:

$$\begin{aligned} & Enc_{pk}(m_1) \cdot Enc_{pk}(m_2) \\ & \equiv (m_1^e \bmod n) \cdot (m_2^e \bmod n) \\ & \equiv (m_1 \cdot m_2 \bmod n)^e \\ & \equiv Enc_{pk}(m_1 \cdot m_2 \bmod n) \pmod{n} \end{aligned}$$

Our protocol is based on the idea of modular power. Under RSA assumption, we can prove it secure.

2.2.2 ElGamal cryptosystem

ElGamal cryptosystem proposed by ElGamal is a public key cryptosystem for the purposes of encryption and signature[6].

- Key Generation

Choose a large prime p and let g be a generator of Z_p^* . Then, choose a random x such that $0 \leq x \leq p - 2$ and compute $y = g^x \bmod p$. Finally, the public key pk is (p, g, y) and the secret key sk is (p, g, x) .

- Encryption

Input a plaintext $m \in Z_p$, and then choose a random r such that $1 \leq r \leq p-2$.

Output the ciphertext of m is

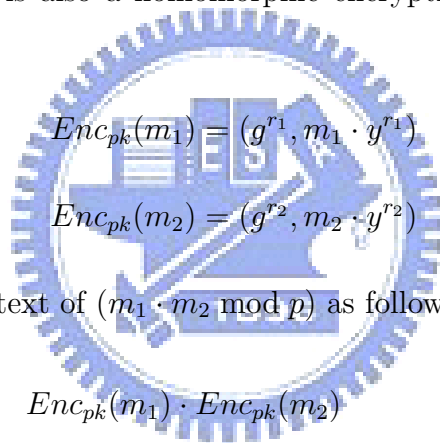
$$Enc_{pk}(m) = (g^r, m \cdot y^r)$$

- Decryption

Input a ciphertext $c = (A, B)$. Output the plaintext of c is

$$Dec_{sk}(c) = \frac{B}{A^x}$$

ElGamal encryption is also a homomorphic encryption. For example, assume that



we can get a new ciphertext of $(m_1 \cdot m_2 \bmod p)$ as follows:

$$\begin{aligned} & Enc_{pk}(m_1) \cdot Enc_{pk}(m_2) \\ & := (g^{r_1} \cdot g^{r_2}, m_1 \cdot y^{r_1} \cdot m_2 \cdot y^{r_2}) \\ & = (g^{(r_1+r_2)}, (m_1 m_2) \cdot y^{(r_1+r_2)}) \\ & = Enc_{pk}(m_1 m_2 \bmod p) \end{aligned}$$

Note: All above computations are finished in Z_p .

We can observe that the homomorphic encryption can aggregate many ciphertexts into one ciphertext. Our protocol uses this idea to reduce the token size and achieve the sliding operation on encrypted data.

Chapter 3

Related Work

In this chapter, we will introduce the related work about string matching and keyword search on encrypted data.

3.1 Framework

When the problem of string matching is extended to the encrypted data, there are some modifications for this purpose. Here, most of definitions and models follow [7].

There are three main parties in the problem like

- Data Owner

The party generates the original plaintext data, and then encrypts the data and uploads it to the server.

- Data Client

The party issues some instructions to the server for retrieving some encrypted data uploaded by the data owner.

- Server

The party serves for all the data owners and the data clients. For the data

owners, it supports storage space for storing the encrypted data uploaded from the data owner. For the data clients, it supports computational power for processing the encrypted data indicated by the data client.¹

We explain some definitions which occur in the following schemes and our protocol. There are two main roles in the problem of search on encrypted data like

- Tag

The tag stands for the encrypted data. Namely, the original plaintext data are encrypted into the tag. Therefore, searching on encrypted data is actually searching on the tag.

The process of generating tag must involve with the message like the text which will be searched in the future. In other words, it seems to map original plaintext data into the encrypted form. In our protocol for string matching on encrypted data, the tag is also called “encrypted text”.

There are some schemes which support both searching and decrypting on the tag. However, decryption of the tag is not necessary if we concentrate on search on encrypted data. One way can deal with it like using the symmetric encryption algorithm (e.g. AES, DES, etc.) on the plaintext data and appending it with the tag. Therefore, we prefer to use the word tag instead of ciphertext to represent the data which will be searched in the future.

- Token

It preserves some functionality like searching on the tag. More precisely, if any party wants to searching on the tag, the party requires to have the appropriate token which can be generated only by the data client’s secret key.

¹In our protocol, we assume that it is the semi-honest model which means the server must follow the protocol.

The process of generating token must involve with the message like the pattern which will search for the text. Generally speaking, the token is called “encrypted query”. In our protocol, the token is also called “encrypted pattern”.

Here, we give a general description about the protocol of search on encrypted data in the following sections. Most of schemes can be expressed as the following four algorithms.

- KeyGen

- Input: a security parameter which usually determines length of the key.
- Output: the keys.

Note: In the secret-key setting, GenTag and GenToken involve with the same secret key. In the public-key setting, GenTag and GenToken involve with the public key and the private key respectively.

- GenTag

- Input: the plaintext data and the key.
- Output: the tag which can be searched with the token.

- GenToken

- Input: the plaintext data and the key.
- Output: the token which can search on the tag.

- Search

- Input: the tag and the token.

- Output: True(Yes) means found, or False(No) otherwise.

Note: In some protocols, the search result is the occurrence positions instead of Yes or No like the string matching problem. It still satisfies the above definition with returning Yes if some occurrence is found or No if there is no any occurrence.

3.2 String Matching on Encrypted Data

Until now, there exists one protocol which was proposed by Oleshchuk and supports to search on encrypted data with pattern matching [1]. Oleshchuk proposed a solution of matching bitstrings in the bitstring encrypted by stream cipher using the naive string-matching algorithm. The main idea is that the encryption scheme is malleable.

As the string matching problem, there are two roles like text and pattern. The text T is encrypted by the stream cipher and uploaded to the remote server like

$$E_K(T) = T \oplus K$$

where \oplus is the operation of bitwise exclusive OR.

Define T_i^m denote a bit substring of the bitstring T that begins at the position i and has length m . Let length of T and K be n and the pattern P be a bitstring of length $m(m \leq n)$. The user wants to find all occurrences of P in the text T and does the following steps:

1. Randomly choose r as a bitstring of length m .
2. Compute the set $Q(r, K) = \{K_1^m \oplus r, K_2^m \oplus r, \dots, K_{n-m+1}^m \oplus r\}$.
3. Send $P \oplus r$ and $Q(r, k)$ to the remote server.²

²Such information uploaded to the server implies the length of P known to the server.

When the server receives the information from the user, it verifies if for all $i = 1, 2, \dots, n - m + 1$

$$(T_i^m \oplus K_i^m) \oplus (K_i^m \oplus r) = (P \oplus r)$$

For the confidentiality of the text T and the pattern P , T and P are encrypted before being sent to the server. Namely, the user can mask the pattern P by a random bitstring r . Since the stream cipher is malleable, the server can convert each partial ciphertext $(T_i^m \oplus K_i^m)$ with the mask $(K_i^m \oplus r)$ into another ciphertext $(T_i^m \oplus r) = (T_i^m \oplus K_i^m) \oplus (K_i^m \oplus r)$ which is compared with the masked pattern $(P \oplus r)$.

The size of $Q(r, K)$ is $O((n - m + 1)m)$ which almost increases linearly with respect to the size of T when $n \gg m$. It requires overwhelming communication size when the size of the pattern P is small especially. Also, this is an insecure solution because the keystream K can be restored from $Q(r, K)$ and then the text T does also since we can deduce K_1^{m+1} from $(K_1^m \oplus r) \oplus (K_2^m \oplus r)$ with high probability.

As the above construction, the author has claimed that it can be more secure if extending r into the set of r' s. Let J be the set of r' s like

$$J = \{r_1, r_2, \dots, r_k\} \subseteq \{0, 1\}^m$$

where each r_i is a random bitstring of length m . Also, the k , size of the set J , determines the strength of security and the resource of the computational time and the communication size in the protocol. Let ρ be a mapping defined like

$$\rho : \{1, 2, \dots, n - m + 1\} \rightarrow J$$

and then define a set

$$Q(\rho, K) = \{K_i^m \oplus \rho(i) | i = 1, 2, \dots, n - m + 1\}.$$

The intuition is that $Q(\rho, K)$ is used instead of $Q(r, K)$. Therefore, the attacker can not restore the keystream like previous scheme. Under such modifications, the user must send all bitstring $P \oplus r_i$ and the set $Q(\rho, K)$ to the server. Then, the search is as follows.

- For each $P \oplus r_{i_j}$ where $i_j = 1, 2, \dots, k$, the server verifies if for all i

$$(T_i^m \oplus K_i^m) \oplus (K_i^m \oplus \rho(i)) = (P \oplus r_{i_j}).$$

In the modified protocol, the real occurrence is happened only if $\rho(i) = r_{i_j}$. Since the information about the mapping ρ is not available to the server, it can not distinguish which occurrence is real or fake. Namely, the server may be confused with the false positives and it can not deduce the keystream K from the set $Q(\rho, K)$ again.

However, the token size is at least $O((n - m + 1)m)$ because of size of the set $Q(\rho, k)$. Such size is large overhead. We hope that it can be reduced as far as possible. On the other hand, Oleshchuk's protocol is efficient in computation because the operation of exclusive OR is simple and convenient. It is appropriate when the communication overhead is not important.

3.3 Keyword Search on Encrypted Data

Under the keyword search on encrypted data, the main role is the “keyword”. Adapting with the definitions of string matching, the keyword is also a string. Now, both the text and the pattern are keywords. It is wondered to determine if the pattern matches the text exactly.

Like one way of indexing for speeding up searching, a document is associated with some keywords. Then, the document is tested whether it is associated with the

dedicated keywords when the user searches on the documents.

Search on encrypted data based on keyword search uses the same idea. The document is associated with many keywords, or it is just composed of the keywords. Also, the keywords are encrypted.

Also, the tag stands for the encrypted keyword which will be searched in the future. On the other way, the token also stands for some encrypted keyword which will search on the tags.

With above ideas, keyword search on encrypted data is mainly classified into two categories as the secret-key and public-key based schemes as follows.

3.3.1 Secret-key Based Keyword Search

Under the secret-key setting, the data owner and the data client are usually the same party. For simplicity, we call, the data owner or data client, the user.

The scenario is that a user wants to upload data to the remote server and then retrieve the demand data in the future. For the confidentiality, the data are encrypted before being uploaded to the remote server. In such case, we want to let the server test whether the file is associated with some keywords, and should learn nothing beyond founded or not.

Song et al. proposed the first protocol which supports to search on encrypted data with keyword search[8]. In the Song et al.'s protocol, the document is composed of the keywords. For simplicity, assume that length of each keyword is same, or padding with 0's or 1's otherwise. Let W be the keyword space which contains all keywords and D be a document. As above, D is expressed as a sequence of keywords W_1, W_2, \dots, W_l where each $W_i \in W$.

There are some primitives used as follows:

- Let $G : K_G \rightarrow S$ be a pseudorandom generator, e.g. a stream cipher.
- Let $F : K_F \times \{0, 1\}^{n-m} \rightarrow \{0, 1\}^m$ where $m < n$ be a pseudorandom function.
- Let $E : K_E \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a pseudorandom permutation, e.g. a block cipher.
- Let $f : K_F \times \{0, 1\}^* \rightarrow K_F$ be a pseudorandom function.

The protocol is described as follows:

- KeyGen

- Input: a security parameter.
- Output: k_G, k_E and k_f which will input to G, E and f respectively and keep them secret.

- GenTag

- Input: a document $D = W_1 || W_2 || \dots || W_l$ and a pseudorandom bitstring $S = S_1 || S_2 || \dots || S_l$ which is generated by G with input k_G . For all i , $|W_i| = n$, $|S_i| = n - m$ and $n > m > 0$.
- Output: the *tag* of the document D .

- 1: **for** $i = 1$ to l **do**
- 2: $X_i = E_{k_E}(W_i)$
- 3: $k_i = f_{k_f}(X_i)$
- 4: $tag_i = X_i \oplus (S_i || F_{k_i}(S_i))$
- 5: **end for**
- 6: return $tag = tag_1 || tag_2 || \dots || tag_l$

- GenToken

- Input: a keyword W and the secret keys k_E and k_f .
- Output: $token = (X = E_{k_E}(W), k_X = f_{k_f}(X))$.

- Search

- Input: a $tag = tag_1 || tag_2 || \dots || tag_l$ and a $token = (X, k_X)$.
- Output: the occurrence positions of X .

```

1: for  $i = 1$  to  $l$  do
2:   if  $tag_i \oplus X$  is the form of  $S || F_{k_X}(S)$  then
3:     print  $i$ 
4:   end if
5: end for

```

Note: In fact, the above protocol supports decryption on the tag with some modification and it is still secure. For simplicity, we just explain the idea of the protocol.

As above construction, searching requires linear time with respect to number of the keywords of a document because of sequential scan. For speeding up, there is a tool as follows.



Bloom filter

The idea comes from [9]. The Bloom filter is composed of a m -bit string B and r independent hash functions h_1, h_2, \dots, h_r where $h_i : \{0, 1\}^* \rightarrow [1, m]$ for all i , and every bit of B is set to 0 initially.

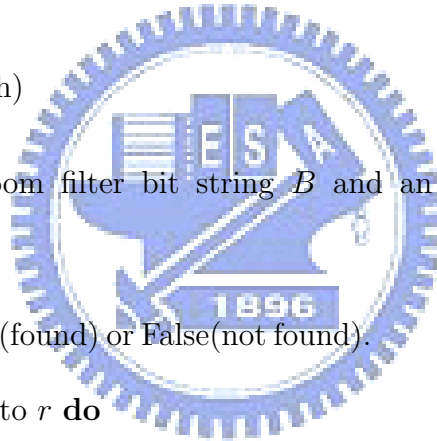
Adapting with the same scenario of above protocol, the following two algorithms are GenTag and Search respectively.

- BuildIndex(GenTag)

- Input: a encrypted document $D' = E_{k_E}(W_1) || E_{k_E}(W_2) || \dots || E_{k_E}(W_l) = X_1 || X_2 || \dots || X_l$.
- Output: m -bit string B .
 - 1: set every bit of B to 0
 - 2: **for** $i = 1$ to l **do**
 - 3: **for** $j = 1$ to r **do**
 - 4: $B[h_j(X_i)] = 1$
 - 5: **end for**
 - 6: **end for**
 - 7: return B

- SearchIndex(Search)

- Input: a Bloom filter bit string B and an encrypted keyword $X = E_{k_E}(W)$.
- Output: True(found) or False(not found).
 - 1: **for** $i = 1$ to r **do**
 - 2: **if** $B[h_i(X)] = 0$ **then**
 - 3: return False
 - 4: **end if**
 - 5: **end for**
 - 6: return True



The keyword must be encrypted before being inputted to BuildIndex and SearchIndex because we do not want to let the server search any keyword arbitrary. As above construction, there is false positive, but it can be controlled if r (the number of hash functions), m (the length of bit string B), and l (the number of keywords which is

inserted into the Bloom filter) are chosen appropriately[9].

The idea of the scheme is to build all indexes of required keywords, and then the searching time can be reduced to the constant time.

3.3.2 Public-key Based Keyword Search

Under the public-key setting, the data owner and the data client are usually the different parties.

The scenario comes from the email system. Assume that a sender Bob wants to send a mail to the receiver Alice. For privacy, the mail is encrypted with Alice's public key. Obviously, Alice is only party who can decrypt the encrypted mail. Sometimes Alice wants to let the mail server have some functionality to test her mails whether the mails contain some keywords and learn nothing beyond founded or not. For example, Alice lets the server have a token which can search the mails whether mail is associated with the keyword 'urgent'. If so, the server will directly forward it to her pager.

Boneh et al. proposed the first protocol which can search on encrypted data based on the public key system[10]. In Boneh et al.'s protocol, the mail is encrypted with the receiver's public key. Besides, each mail is associated with many keywords which will be searched with dedicated token in the future.

For example, Bob wants to send a mail to Alice, and the mail is encrypted by Alice's public key and it is associated with l tags as follows:

$$E_{pk_{Alice}}(mail)||tag_1||tag_2||\dots||tag_l$$

$E_{pk_{Alice}}(\cdot)$ means a public key encryption with Alice's public key and each tag_i is generated by GenTag on inputting some keyword and Alice's public key.

As the above example, Bob must previously choose the demand keywords associated with the encrypted mail. Then, the mail server can search on the tags with the tokens which can be generated only by Alice's private key.

The bilinear map is used in the protocol and described as follows: Let G_1 and G_2 be two groups of large prime order p . A bilinear map $e : G_1 \times G_1 \rightarrow G_2$ satisfies the following properties:

- Bilinear: given $g, h \in G_1$ and $a, b \in Z$, then $e(g^a, h^b) = e(g, h)^{ab}$.
- Non-degenerate: if g is a generator of G_1 , then $e(g, g)$ is a generator of G_2 .
- Computable: the map e can be computed in a polynomial time.

The protocol is described as follows:

- KeyGen

- Input: a security parameter to determine p , the size of the groups G_1 and G_2 . Choose a random $\alpha \in Z_p^*$ and a generator g of G_1 .
- Output: $pk = (g, h = g^\alpha)$ and $sk = \alpha$.

- GenTag

- Input: the public key pk and a keyword W . Compute $t = e(H_1(W), h^r) \in G_2$ for a random $r \in Z_p^*$ where $H_1 : \{0, 1\}^* \rightarrow G_1$.
- Output: $tag = (g^r, H_2(t))$ where $H_2 : G_2 \rightarrow \{0, 1\}^{\log p}$.

- GenToken

- Input: the secret key sk and a keyword W .
- Output: $tk = H_1(W)^\alpha \in G_1$.

- Search

- Input: the public key pk , a tag $tag = (A, B)$ and a token tk .
- Output: *yes* if $H_2(e(tk, A)) = B$; *no* otherwise.

The correctness comes from the bilinear map as follows.

$$\begin{aligned}
 & H_2(e(tk, A)) \\
 &= H_2(e(H_1(W)^\alpha, g^r)) \\
 &= H_2(e(H_1(W), g)^{\alpha r}) \\
 &= H_2(e(H_1(W), (g^\alpha)^r)) \\
 &= H_2(e(H_1(W), h^r)) \\
 &= B
 \end{aligned}$$

Keyword search is a way of indexing. When we want to search for some documents in the database, we usually issue a query with some keywords. These keywords are used to test whether the document is associated with them. Therefore, keyword is a good idea for indexing. Nevertheless, it is not natural when we want to find if the document contains some pattern. Therefore, string matching is suitable for this job.

3.3.3 String Matching with Keyword Search

As above section, the ideas of keyword search and the tool like Bloom filter can be extended for string matching with some modifications.

String Matching with Keyword Search

As above constructions of keyword search, there is a trivial way for string matching on encrypted data with keyword search. Based on the naive string-matching algo-

rithm, the two main roles are text and pattern. The text is encrypted as the *tag* like

$$text = c_1 || c_2 || \dots || c_l \xrightarrow{GenTag} tag = (tag_1, tag_2, \dots, tag_l)$$

where each c_i is a character and $l > 0$. Also, the pattern is encrypted as the *token* like

$$pattern = c'_1 || c'_2 || \dots || c'_m \xrightarrow{GenToken} token = (token_1, token_2, \dots, token_m)$$

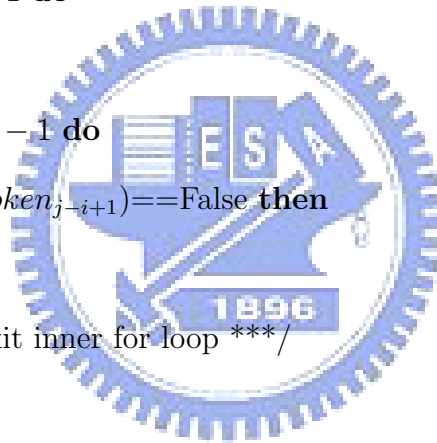
where each c'_i is a character and $0 < m \leq l$.

The string matching algorithm based on keyword search is like

```

1: for  $i = 1$  to  $l - m + 1$  do
2:    $flag = 1$ 
3:   for  $j = i$  to  $i + m - 1$  do
4:     if  $search(tag_j, token_{j-i+1}) == \text{False}$  then
5:        $flag = 0$ 
6:       break /*** exit inner for loop ***/
7:     end if
8:   end for
9:   if  $flag == 1$  then
10:     $print\ i$ 
11:   end if
12: end for

```



The above way seems to deal with the problem of string matching on encrypted data. However, the most important problem is that it is vulnerable of the statistical attack because the tokens are independent. In other words, the characters of the pattern are independent. Therefore, the server can test any occurrence of the

character belong to the pattern for the token. It causes the statistical analysis. The more details are discussed in the following section about heuristic approaches.

String Matching with Bloom Filter

As above, Bloom filter can speed up searching like the idea of indexing. Besides, it can also support string matching.

Let T be the text and find all substrings of T . For example, assume that the text T is “abc”. Then, all strings like “a”, “b”, “c” with length 1, “ab”, “bc” with length 2, and “abc” with length 3 are the substrings of T . Namely, if T is a string of length l , there are $l + (l - 1) + \dots + 1 = (l + 1)l/2 = O(l^2)$ substrings.

Let these substrings be the keywords and insert them to Bloom filter. Then, we can get a trivial construction of string matching by Bloom filter. For example, if we want to search some pattern, we can use SearchIndex with the input of encrypted pattern. In the scheme, the preprocessing time is $O(l^2)$ for inserting all substrings of the text but the searching time is still $O(1)$ which is constant.

3.4 Other Extensions

There are many issues about search on encrypted data. The issue about more flexible and efficiency for secret-key based keyword is discussed in [11]. The issue about decryption which means that the tag can be decrypted is discussed in [12, 13]. The issue about conjunction keyword search which supports to search many keywords in a query is discussed in [14, 15, 16, 17, 18, 19]. The issue about more general query like conjunction, subset, and range queries is discussed in [20].

Chapter 4

Proposed Protocol

In this chapter, we will show the heuristic approaches and the detailed construction of our protocol. In fact, the idea comes from the naive string-matching algorithm, so there are two primitive operations like “comparing” and “sliding”. In the following sections, we will focus on the achievement of the two primitive operations.

Loosely speaking, for comparing, the idea is that the tag is exactly matched by some information from the token. For sliding, the idea of privacy homomorphism is used. The token seems to be converted to another one, and it still expresses the token of the same pattern at the different position.

4.1 Problem description

Search on encrypted data by string matching instead of keyword search has the main advantage which is that the pattern does not be defined in advance, and any string of length not over than the text can be the pattern.

Background Sometimes the user wants to upload the files to the remote server. For the confidentiality, the files are encrypted before being uploaded. If the user

wants to search the files, the dedicated token which can be generated only by the file owner's secret key is required for this purpose. We hope to reduce the communication size of data (which are the information to finish the protocol, not the encrypted files itself) and the computational time of the user as far as possible.

Expression of data In our protocol, we use the idea of the naive string-matching algorithm. For practical purpose, the character space may be ASCII or some else finite character sets. The data are encrypted with respect to each character, so length of the tag is asymptotically same to length of the text.

4.2 Heuristic Approaches

In this section, we introduce our approaches incrementally. The first approach is a deterministic way to generate tag but is vulnerable of statistical attack.

In the second approach, we can get a randomized way to generate tag through some randomness involved. However, the token size may be too large to cause heavy communication cost.

Therefore, we use the multiplication homomorphism to get rid of such obstacle and get the third approach for saving the communication cost. Nevertheless, it is still vulnerable of statistical attack.

Finally, we explain the fourth approach which is the model of our protocol to get rid of the problems which occur in the previous approaches.

Here is an example which is used in the following approaches. We assume that a user wants to search the text $T = a||b||c||a||a$ with the pattern $P = b||c$, and both the text T and the pattern P are encrypted.

4.2.1 Scheme I: The Deterministic Scheme

In our protocol, the tag is generated by each character of the text. In other words, each character of the text seems to be encrypted into some ciphertext as the tag. For example,

$$T = a||b||c||a||a \xrightarrow{GenTag} Tag = (\#, \$, \%, \#, \#)$$

Obviously, we can see that the first, fourth and fifth tuples of the tag stand for the encryption of the same character. Namely, the tag just tells the ingredients of the text.

Therefore, the deterministic generation of the tag is vulnerable of statistical analysis since the encryption of a character is only a mapping to get the identifier of the character.

4.2.2 Scheme II: The Randomized Scheme

The idea of modification is that the generation of the tag must involve some extra information besides the original character. The information of the position is natural. If the GenTag is simply described as the function f on inputting a character and a position information, we can get the tag like

$$T = a||b||c||a||a \xrightarrow{GenTag} Tag = (f(a, 1), f(b, 2), f(c, 3), f(a, 4), f(a, 5))$$

Roughly speaking, the probability of $f(a, i) = f(a, j)$ where $i \neq j$ is negligible when f is appropriately chosen as a random function.

In fact, the deterministic idea is still used in our protocol. The tag is actually deterministic if the character c and the position i are fixed. Therefore, the user can generate the appropriate token of the pattern which can search the dedicated text

afterward. Therefore, we can get the token like

$$P = b||c \xrightarrow{GenToken}$$

$$Token = \{[f(b, 1), f(c, 2)], [f(b, 2), f(c, 3)], [f(b, 3), f(c, 4)], [f(b, 4), f(c, 5)]\}$$

For comparing, each pair of $[f(b, i), f(c, i+1)]$ verifies if the occurrence of the pattern takes place at the position i . For sliding, the next pair of $[f(b, i + 1), f(c, i + 2)]$ is used.

It asymptotically requires $O((|T| - |P| + 1) \cdot |P|)$ communication cost. If the length of the text is long, the communication cost is almost equal to the size of retrieving the entire encrypted text. We want to reduce the communication cost as far as possible.

4.2.3 Scheme III: The Slide Scheme

Scheme-II is similar to Oleshchuk et al.'s scheme[1]. In fact, the information of position is an abstract idea, and it can be any information kept only by the owner. Namely, the owner can generate any valid tag of a character on the dedicated position as the token. However, it can not reach our purpose since all tags of the pattern at any position are sent.

The privacy homomorphism is a good idea to solve this problem. The idea is to convert one token into another one of the same pattern at the different position as follows:

$$f(c, i) \xrightarrow{slide(c)} f(c, j) \text{ where } i \neq j$$

The token of the character c at the position i can be converted into the token of the same character c at the position j with the extra information like $slide(c)$. Therefore,

we can get the token like

$$P = b||c \xrightarrow{GenToken} Token = \{f(b, 1), f(c, 2) \text{ and } slide(b), slide(c)\}$$

For comparing, the initial information like $f(b, 1)$ and $f(c, 2)$ is taken first to compare with the tag. For sliding, the slide information like $slide(b)$ and $slide(c)$ can convert $f(b, 1)$ and $f(c, 2)$ into any $f(b, i)$ and $f(c, i+1)$ respectively and keep going comparing.

Under this construction, the communication cost can be reduced to $O(|P|)$.

4.2.4 Scheme IV: The Final Scheme

Although, the communication cost can be reduced to $O(|P|)$ in the Scheme-III. At least, it is same to the original problem of string matching with the naive string-matching algorithm. Nevertheless, it is still vulnerable of statistical analysis like the Scheme-II. There are two main reasons like

- The pattern is indirect revealed by the tokens of each character of the pattern. For example, if $Token = \{f(b, 1), f(c, 2) \text{ and } slide(b), slide(c)\}$, then we can get $f(b, 2)$ by

$$f(b, 1) \xrightarrow{slide(b)} f(b, 2)$$

Since the probability of $f(b, 2) \neq f(c, 2)$ is quite high, it tells that the first character and the second character of the pattern are different.

- The information beyond the occurrences of the pattern is revealed. For example, if $Token = \{f(b, 1), f(c, 2) \text{ and } slide(b), slide(c)\}$, then we can get any $f(b, i)$ for all i by

$$f(b, 1) \xrightarrow{slide(b)} f(b, 2), f(b, 3), \dots$$

Namely, the server can generate all tokens of each character of the pattern at any position. In other words, the server can search the text with any arrangement of the characters of the pattern.

Therefore, the idea of the final scheme comes from “encapsulation”. Originally, the token of a pattern is composed of the “initials” like $\{f(b, 1), f(c, 2)\}$ and the “slides” like $\{slide(b), slide(c)\}$. Now, we aggregate the initials and the slides to form the “aggregate initial” and the “aggregate slide” respectively. Roughly speaking, the individual information of each character of the pattern is not revealed. Our protocol is based on this idea, and the detailed construction is as the following section.

4.3 Construction

We first introduce the framework of our protocol. The protocol is composed of the following algorithms:

- $\text{KeyGen}(\lambda)$

Input a security parameter λ and output a secret key K . In the process, some tables are set up and kept secret.

- $\text{GenTag}(T, K)$

Input a text T and a secret key K and output the Tag of T . Usually, the Tag will be uploaded to the remote server.

- $\text{GenToken}(P, K)$

Input a pattern P and a secret key K and output the Token of P . Usually, the Token is generated only for the search.

- $\text{Search}(\text{Tag}, \text{Token})$



Input a *Tag* and a *Token* and output the occurrences. Usually, it is happened in the server side. The tag had been reserved in the server previously, and the user issues a query by the token uploaded to the server.

correctness In the search algorithm, the result of the execution is the occurrence of the patter P in the text T if the *Tag* is generated by the text T and the *Token* is generated by the pattern P .

4.3.1 Our Protocol

- KeyGen(λ)

Input a security parameter λ . Randomly choose two large and distinct primes p and q such that $|p| = |q| = \lambda$ and let $n = pq$. Then randomly choose $d \in Z_{\phi(n)}^*$ and $s \in Z_{\phi(n)}$. Finally, n is the public parameter and $K = (s, d)$ is the secret key.

After key generation, the user must determine a character space Σ and a 1-1 mapping $f : \Sigma \rightarrow \Sigma'$ as the following table:

Σ	c_1	c_2	\dots	c_k
Σ'	z_{i_1}	z_{i_2}	\dots	z_{i_k}

Table 4.1: Character Space Encoding

Especially, Σ' is a subset of Z_n^* . Then, all the following computations are done in Z_n^* . Finally, the user has to keep the mapping(table) secret.

- GenTag(T, K)

Input a text $T = c_1||c_2||\dots||c_l$ where each c_i is a character over the character

space Σ and a secret key $K = (s, d)$. Choose a random $r \in Z_n^*$ and output

$$\begin{aligned} Tag & \\ &= (tag_1, tag_2, \dots, tag_l) \\ &= ((z_1 r)^{sd} \bmod n, (z_2 r)^{(s+1)d} \bmod n, \dots, (z_l r)^{(s+l-1)d} \bmod n) \end{aligned}$$

where for all i , $z_i = f(c_i) \in Z_n^*$.

After generating the tag of a text, the user must maintain the following table.

TextID	$text_1$	$text_2$	\dots
r	r_1	r_2	\dots
length	l_1	l_2	\dots

Table 4.2: Text Information

The table shows the r_i and length l_i which are associated with the $text_i$. The user has to determine the dedicated r_i before generating the token which search on the tag of the $text_i$. Finally, the user has to keep the table secret.

- GenToken(P, K)

Input a $pattern = c'_1 || c'_2 || \dots || c'_m$ where each c'_i is a character over the character space Σ and a secret key $K = (s, d)$. The user has to determine r and length l of the target text in advance. If length of the $pattern$ is greater than l , then

terminate. Otherwise, output

$$\begin{aligned}
 &Token = (\\
 &\quad length = m, \\
 &\quad initial = (z'_1 r)^{sd} (z'_2 r)^{(s+1)d} \dots (z'_m r)^{(s+m-1)d} \bmod n, \\
 &\quad slide = (z'_1 z'_2 \dots z'_m \cdot r^m)^d \bmod n \\
 &)
 \end{aligned}$$

where for all i , $z'_i = f(c'_i) \in Z_n^*$.

- Search($Tag, Token$)

Input a $Tag = (tag_1, tag_2, \dots, tag_l)$ and a $Token = (length = m, initial, slide)$.

For efficiency, the extra variables like $temp1$ and $temp2$ can reduce the times of multiplication, and then does

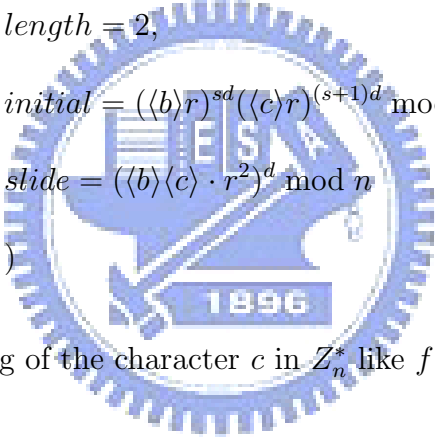
- 1: $temp1 = initial$
- 2: $temp2 = tag_1 \cdot tag_2 \cdot \dots \cdot tag_m \bmod n$
- 3: **for** $i = 1$ to $l - m + 1$ **do**
- 4: **if** $temp1 == temp2$ **then**
- 5: print i /* the occurrence position of $pattern$ */
- 6: **end if**
- 7: **if** $i < l - m + 1$ **then**
- 8: $temp1 = temp1 \cdot slide \bmod n$
- 9: $temp2 = tag_i^{-1} \cdot temp2 \cdot tag_{i+m} \bmod n$
- 10: **end if**
- 11: **end for**

For example, assume that there are a text $T = a||b||c||a||a$ and a pattern $P = b||c$.

With GenTag and GenToken, we can get

$$\begin{aligned}
Tag & \\
&= (tag_1, tag_2, \dots, tag_5) \\
&= ((\langle a \rangle r)^{sd} \bmod n, (\langle b \rangle r)^{(s+1)d} \bmod n, \\
&\quad (\langle c \rangle r)^{(s+2)d} \bmod n, (\langle a \rangle r)^{(s+3)d} \bmod n, (\langle a \rangle r)^{(s+4)d} \bmod n)
\end{aligned}$$

and

$$\begin{aligned}
Token = (& \\
&length = 2, \\
&initial = (\langle b \rangle r)^{sd} (\langle c \rangle r)^{(s+1)d} \bmod n, \\
&slide = (\langle b \rangle \langle c \rangle \cdot r^2)^d \bmod n \\
&) &
\end{aligned}$$


where $\langle c \rangle$ means encoding of the character c in Z_n^* like $f(c)$.

4.3.2 Correctness

In this section, the correctness of search in our protocol will be discussed.

Let the text $T = c_1||c_2||\dots||c_l$ and the pattern $P = c'_1||c'_2||\dots||c'_m$ where both c_i and c'_j are the character over the same character space for all i, j and $0 < m \leq l$.

Then, we can get the tag and the token like

$$\begin{aligned}
Tag & \\
&= (tag_1, tag_2, \dots, tag_l) \\
&= ((z_1 r)^{sd} \bmod n, (z_2 r)^{(s+1)d} \bmod n, \dots, (z_l r)^{(s+l-1)d} \bmod n)
\end{aligned}$$

and

$$\begin{aligned}
 &Token = (\\
 &\quad length = m, \\
 &\quad initial = (z'_1 r)^{sd} (z'_2 r)^{(s+1)d} \dots (z'_m r)^{(s+m-1)d} \bmod n, \\
 &\quad slide = (z'_1 z'_2 \dots z'_m \cdot r^m)^d \bmod n \\
 &)
 \end{aligned}$$

where $z_i = f(c_i)$, $z'_j = f(c'_j) \in Z_n^*$ for all i, j .

We differ the cases by length m of the pattern P .

- case 1: $m = 1$

The token tells $initial = (z'_1 r)^{sd} \bmod n$ and $slide = (z'_1 r)^d \bmod n$. We can multiply $initial$ by $slide$ any k time and get any $(z'_1 r)^{(s+k)d} \bmod n$ for $k \in Z^+$. If some z_i is equal to z'_1 , then we can compare the $tag_i = (z_i r)^{(s+i-1)d} \bmod n$ with $(z'_1 r)^{sd} \cdot ((z'_1 r)^d)^{i-1} \bmod n = (z'_1 r)^{sd} \cdot (z'_1 r)^{(i-1)d} \bmod n = (z'_1 r)^{(s+i-1)d} \bmod n$.

- case 2: $1 < m < l$

Assume that the pattern P is occurred at the position i of the text T where $1 \leq i \leq l-m+1$. Namely, $P = T[i \dots i+m-1]$ and $z'_1 = z_i, z'_2 = z_{i+1}, \dots, z'_m = z_{i+m-1}$. The search algorithm can print all i when the for-loop is executed on

the i -th times as follows:

$$\begin{aligned}
& temp1 \\
& \equiv initial \cdot slide^{(i-1)} \\
& \equiv ((z'_1 r)^{sd} (z'_2 r)^{(s+1)d} \dots (z'_m r)^{(s+m-1)d}) \cdot ((z'_1 z'_2 \dots z'_m \cdot r^m)^d)^{i-1} \\
& \equiv ((z'_1 r)^{sd} (z'_2 r)^{(s+1)d} \dots (z'_m r)^{(s+m-1)d}) \cdot ((z'_1 r)^{(i-1)d} (z'_2 r)^{(i-1)d} \dots (z'_m r)^{(i-1)d}) \\
& \equiv (z'_1 r)^{(s+i-1)d} (z'_2 r)^{(s+i)d} \dots (z'_m r)^{(s+m+i-2)d} \\
& \equiv (z_i r)^{(s+i-1)d} (z_{i+1} r)^{(s+i)d} \dots (z_{i+m-1} r)^{(s+m+i-2)d} \\
& \equiv tag_i \cdot tag_{i+1} \dots tag_{i+m-1} \\
& \equiv temp2 \pmod{n}
\end{aligned}$$

- case 3: $m = l$

In fact, the *slide* is useless in this case. Also, $z'_1 = z_1, z'_2 = z_2, \dots, z'_m = z_m = z_l$ if P exactly matches with T . Therefore, the search algorithm can print the occurrence position 1 and the for-loop is executed only once since *initial* is equal to $tag_1 \cdot tag_2 \dots tag_l$ as follows:

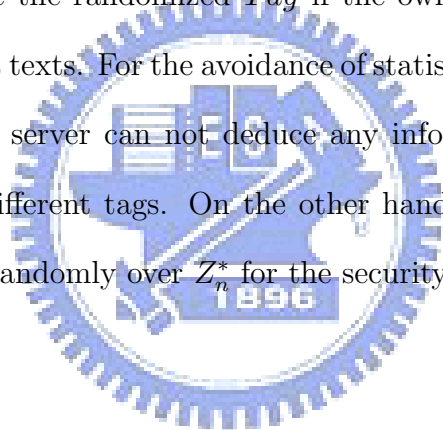
$$\begin{aligned}
& temp1 \\
& \equiv initial \\
& \equiv (z'_1 r)^{sd} (z'_2 r)^{(s+1)d} \dots (z'_m r)^{(s+m-1)d} \\
& \equiv (z_1 r)^{sd} (z_2 r)^{(s+1)d} \dots (z_m r)^{(s+m-1)d} \\
& \equiv (z_1 r)^{sd} (z_2 r)^{(s+1)d} \dots (z_l r)^{(s+l-1)d} \\
& \equiv tag_1 \cdot tag_2 \dots tag_l \\
& \equiv temp2 \pmod{n}
\end{aligned}$$

4.3.3 Mask Exponent s

In our protocol, the position information of a character c is used to be the exponent information. If the mask exponent s is omitted (or always set to 1), we can raise tag_1 to the i -th power and then compare with tag_i . Similarly, tag_2 can be compared with tag_{2i} for $i \in Z^+$ and so on. In other words, the ingredients of a text may be deduced through only encrypted text like tag. Avoiding such statistical analysis requires a random s .

4.3.4 Mask Base Number r

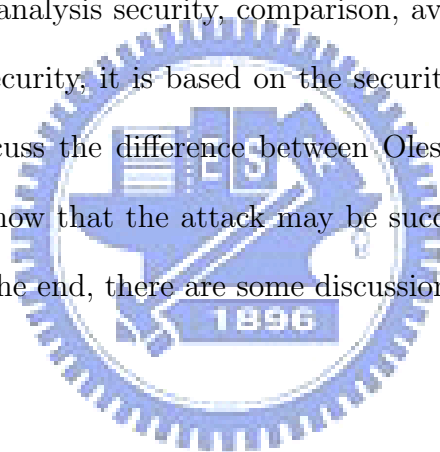
It is obvious to generate the randomized Tag if the owner wants to use the same secret key in the different texts. For the avoidance of statistical attack of the different texts, we hope that the server can not deduce any information on corresponding positions between the different tags. On the other hand, we hope that the entire base number like $z_i r$ is randomly over Z_n^* for the security analysis.



Chapter 5

Analysis

In this chapter, we will analysis security, comparison, avoidance and discussions of our protocol. For the security, it is based on the security proof by reduction. For the comparison, we discuss the difference between Oleshchuk's and our protocol. For the avoidance, we show that the attack may be successful if the inappropriate directions are used. In the end, there are some discussions.



5.1 Security

The RSA assumption is adopted in the following security reduction because our protocol is based on the similar operation like modular power.

5.1.1 RSA Assumption

Given (n, e, y) where $n = pq$, p and q are large primes, and e is coprime to $\phi(n)$, a random $y \in Z_n^*$. It is computationally infeasible to determine x such that $x^e \bmod n = y$.

5.1.2 Security Model

RSA assumption tells that RSA function is hard to invert. In our security model, we think that the adversary can not get the information like $z_i r$ from the i -th subtag $(z_i r)^{(s+i-1)d} \bmod n$ of the tag. Although, the information like z_i is enough to cause statistical analysis. In fact, the information like $z_i r$ is also enough, and we finish the security reduction in this scenario.

5.1.3 Security Reduction

Our protocol is based on RSA assumption. Assume that an adversary A can break our protocol that means to determine any $z_i r$ tuple of the Tag , and then we can use A to determine RSA problem as follows.

Let (n, e, y) be the RSA problem challenge. Randomly choose $s \in Z_{\phi(n)}$. Then, set r to x and d to e . Then, randomly choose $l \in Z^+$ and $z_i \in Z_n^*$ for all $i = 1, 2, \dots, l$.

Finally, n is the public parameter, and give A the tag like

$$\begin{aligned} Tag & \\ &= (tag_1, tag_2, \dots, tag_l) \\ &= ((z_1 x)^{se} \bmod n, (z_2 x)^{(s+1)e} \bmod n, \dots, (z_l x)^{(s+l-1)e} \bmod n) \\ &= (z_1^{se} y^s \bmod n, z_2^{(s+1)e} y^{(s+1)} \bmod n, \dots, z_l^{(s+l-1)e} y^{(s+l-1)} \bmod n) \end{aligned}$$

By contradiction, we assume that some adversary A can break our protocol. It means that A outputs m_i such that $m_i = z_i x$, and we can compute the $x = m_i z_i^{-1} \bmod n$ as the answer of the RSA problem challenge. It concludes the security reduction.

5.2 Comparison

The section will discuss the comparison of Oleshchuk and our protocol. Let n be the length of the text and m be the length of the pattern.

5.2.1 Time

Bitwise exclusive OR is the main operation in Oleshchuk's protocol, and modular multiplication is ours. Obviously, modular multiplication is much more overhead than bitwise exclusive OR, but the generation of the tag can be done off-line in the user side. Also, the search algorithm is executed in the server side. Actually, the operation of modular power or modular exponentiation can be speeded up through some way like square-and-multiply, etc.

However, both Oleshchuk and our protocol use the sequential scan based on the naive string-matching algorithm. It requires asymptotically $O((n - m + 1)m)$ time. There are many better ways to reduce the time like KMP[21] algorithm, etc. The protocol proposed by Oleshchuk supports KMP algorithm. We hope that our protocol can do in the future.

5.2.2 Space

Type	Key Size(bits)	Tag Size(bits)	Token Size(bits)
Oleshchuk(original)	n	n	$(n - m + 1)m + m$
Ours	$O(\lambda)$	$O(n\lambda)$	$\log m + O(\lambda)$

Table 5.1: Protocol Space Comparison

The table shows the demand space of Oleshchuk's protocol and ours. Especially, the size of the token in ours is $\log m + O(\lambda)$ seems not constant because of $\log m$.

However, the value of $\log m$ only tells the length of the pattern. We think that it is not major overhead.

In our protocol, the text is encrypted by each character, so the size of the tag is linear with respect to the size of the text. The owner keeps only the secret key (s, d) and two tables of the character space encoding and the text information. The sizes of the secret key and the table of the character encoding space are constant. The size of the table of the text information is linear with respect to the number of the texts.

5.3 Avoidance

Our protocol has been designed to avoid the statistical attacks as far as possible. However, the protocol reveals the length of the pattern from the token and the occurrence of the pattern. Through such information, the server can take the statistical attack successfully if the inappropriate directions are used.

5.3.1 Small Token

It means that the pattern contains fewer characters. In this case, the occurrences of the pattern may be happened frequently. For extreme example, the pattern contains only one character. There are many occurrences with high probability. Because of revealing length of the pattern by the token, it may be vulnerable of statistical analysis.

5.3.2 Search Too Many Times

If the user searches too many times on the same encrypted text like tag, revealing the occurrences of each pattern may cause the statistical attack with length of the pattern and a dictionary. For getting rid of this problem, the main idea is to reduce the search times or to re-encrypt the text after searching too many times.

Actually, re-encrypt means to ignore the original uploaded tag in the server, generate a new tag, and then upload it. Then, the following queries of string matching are taken place in the new tag in the future.

5.4 Discussion

In this section, we will discuss some issues of our protocol.

5.4.1 False Positive

In our protocol, the search algorithm may have false positive. However, we think that the probability is quite small under modulo n . If it is happened, the user can retrieve the file, decrypt it, and verify before use. Therefore, it requires post-filtering.

5.4.2 Exposure of Occurrence Position

Such situation also exists in Oleshchuk's protocol, and it may be vulnerable of statistical attack. Because of revealing the occurrence positions, small token or searching too many times may cause the statistical attack successfully. The result directly implies that the randomized generation of a token is meaningless because of exposure of occurrence positions. Namely, the adversary can do statistical analysis from the occurrence position through the search result and the length of the pattern

which is told from the token and then distinguish the tokens.

5.4.3 Decryptable

An encryption function usually has the corresponding inverse function as the decryption function. Here we just concern on search on encrypted data, and the original data can be encrypted by standard encryption algorithm like DES, AES, etc., and append it to the tags. Actually, the generation of tag uses the way which is similar to RSA function. Therefore, we hope that the tag can be decrypted in the future.



Chapter 6

Conclusion

We have proposed a protocol to deal with string matching on encrypted data with a constant size token. Nevertheless, it requires the sequential search. Is it possible to reduce the time with indexing or other ways? Also, in our protocol, the occurrence and length of the pattern are known to the server. It may cause the statistical attacks under special case of usage. Is it possible to avoid revealing such information to the server? We assume that the server is curious and follows the protocol. Is it possible to enhance our protocol to the malicious server who may not follow the protocol? Also, is it possible to extend the problem to the public-key setting? The most obvious application is the email system.

Bibliography

- [1] Vladimir Oleshchuk: Privacy Preserving Pattern Matching on Remote Encrypted Data. IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007:609-613
- [2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms, Second Edition The MIT Press and McGraw-Hill Book Company 2001
- [3] Ronald L. Rivest, Len Adleman, Michael L. Dertouzos: On data banks and privacy homomorphisms. Foundations of Secure Computation:169-179 New York: Academic Press 1978
- [4] Danny Dolev, Cynthia Dwork, Moni Naor: Non-Malleable Cryptography (Extended Abstract) STOC 1991:542-552
- [5] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM (CACM) 21(2):120-126 (1978)
- [6] Taher El Gamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. CRYPTO 1984:10-18

- [7] Hakan Hacigumus, Bijit Hore, Balakrishna R. Iyer, Sharad Mehrotra: Search on Encrypted Data. *Secure Data Management in Decentralized Systems* 2007:383-425
- [8] Dawn Xiaodong Song, David Wagner, Adrian Perrig: Practical Techniques for Searches on Encrypted Data. *IEEE Symposium on Security and Privacy* 2000:44-55
- [9] Eu-Jin Goh: Secure Indexes. *IACR Cryptology ePrint Archive*, Report 2003/216, 2003. <http://eprint.iacr.org/>
- [10] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano: Public Key Encryption with Keyword Search. *EUROCRYPT 2004*:506-522
- [11] Yan-Cheng Chang, Michael Mitzenmacher: Privacy Preserving Keyword Searches on Remote Encrypted Data. *ACNS 2005*:442-455
- [12] Thomas Fuhr, Pascal Paillier: Decryptable Searchable Encryption. *ProvSec* 2007:228-236
- [13] Dennis Hofheinz, Enav Weinreb: Searchable encryption with decryption in the standard model. *IACR Cryptology ePrint Archive*, Report 2008/423, 2008. <http://eprint.iacr.org/>
- [14] Philippe Golle, Jessica Staddon, Brent R. Waters: Secure Conjunctive Keyword Search over Encrypted Data. *ACNS 2004*:31-45
- [15] Dong Jin Park, Kihyun Kim, Pil Joong Lee: Public Key Encryption with Conjunctive Field Keyword Search. *WISA 2004*:73-86
- [16] Lucas Ballard, Seny Kamara, Fabian Monrose: Achieving Efficient Conjunctive Keyword Searches over Encrypted Data. *ICICS 2005*:414-426

- [17] Jin Wook Byun, Dong Hoon Lee, Jongin Lim: Efficient Conjunctive Keyword Search on Encrypted Data Storage System. EuroPKI 2006:184-196
- [18] Yong Ho Hwang, Pil Joong Lee: Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. Pairing 2007:2-22
- [19] Peishun Wang, Huaxiong Wang, Josef Pieprzyk: Keyword Field-Free Conjunctive Keyword Searches on Encrypted Data and Extension for Dynamic Groups. CANS 2008:178-195
- [20] Dan Boneh, Brent Waters: Conjunctive, Subset, and Range Queries on Encrypted Data. TCC 2007:535-554
- [21] Donald E. Knuth, James H. Morris Jr., Vaughan R. Pratt: Fast Pattern Matching in Strings. SIAM J. Comput. (SIAMCOMP) 6(2):323-350 (1977)

