# 國立交通大學

## 資訊科學與工程研究所

## 碩士論文

# 公開金鑰可搜尋多關鍵字之加密系統

# Public Key Searchable Encryption with Conjunctive Queries

研究生：謝嘉雯

指導教授：陳榮傑　教授

中華民國 101 年 七 月

# 公開金鑰可搜尋多關鍵字之加密系統

# Public Key Searchable Encryption with Conjunctive Queries

研究生：謝嘉雯　　Student：Chai-Wen Hsieh

指導教授：陳榮傑　　Advisor：Rong-Jaye Chen

國立交通大學

資訊科學與工程研究所

碩士論文

A Thesis
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

July 2012

Hsinchu, Taiwan, Republic of China

中華民國 101 年七月

Public Key Searchable Encryption with Conjunctive Queries

Student：Chai-Wen Hsieh Advisors：Dr. Rong-Jaye Chen

Institute of Computer Science and Engineering
College of Computer Science,
National Chiao Tung University

# ABSTRACT

Currently, there has been a trend for users to store their encrypted private data over the Internet on a data server. Most applications rely on the data server with powerful computing power to perform searching on those encrypted data under the circumstances that server cannot access the plaintext of the data. The searchable encryption therefore becomes a crucial technique that supports searching functionality over encrypted data. Searchable encryption provides storage and computing efficiency for searching on certain keywords without requiring the decryption key. Researchers have been focused on the public key searchable encryption since it supports for multi-user settings and is considered more suitable for actual use than symmetric key searchable encryption. On the other hand, improving the searching functionality such as supporting conjunctive queries and other types of queries has been deeply studied. In this thesis, we survey the prominent public key searchable encryption schemes. Then we propose our design of public key searchable encryption with conjunctive queries that allows the users sharing the encrypted data among multiple users without sharing the decryption keys, as well as sending arbitrary conjunctive queries $(KW_1 \wedge \cdots \wedge KW_w)$ to the server without leaking information of any individual conjuncts. Our design of searchable encryption is based on bilinear pairing based cryptography, which implies it requires shorter key size than the traditional RSA public-key encryptions and implies improvement of efficiency.

公開金鑰可搜尋多關鍵字之加密系統

學生：謝嘉雯　　　　　　　　　　指導教授：陳榮傑 教授

國立交通大學資訊科學與工程研究所碩士班

# 摘要

使用者將加密後的私密資料儲存至位於網際網路上的資料儲存伺服器已成為趨勢。有很多應用依賴具有強大運算能力的伺服器根據關鍵字搜尋加密文件，可搜尋之加密系統因此成為重要的技術。對於加密文件中搜尋關鍵字，此系統提供有效率的儲存與計算方式，並且無需解密金鑰。在可搜尋之加密系統的研究之中，比起對稱金鑰可搜尋之加密系統，公開金鑰可搜尋之加密系統較為被重視，因其支持多使用者的環境設定。另一方面，搜尋功能如多關鍵字之查詢的改進也已被深入探討。在這篇論文中，我們概括論述了重要的公開金鑰可搜尋之加密系統。接著我們提出一個公開金鑰可搜尋多關鍵字之加密系統，提供讓使用者在不透漏解密金鑰的情況下分享加密文件之功能，以及要求伺服器在加密之文件中搜尋多個關鍵字$(KW_1 \wedge \cdots \wedge KW_w)$，並且不洩漏任何單一關鍵字的資訊。我們的搜尋之加密系統基於雙線性配對之密碼系統，比起傳統 RSA 公開金鑰加密系統具有更短的密鑰長度，提高計算上的效能。

# 誌謝

這篇碩士論文的完成，首先要感謝我的指導老師陳榮傑教授，不僅在專業知識部分給予我方向帶領我，也在待人處事方面以身作則，為學生樹立良好典範，親切地對待學生有如對待子女一般，謝謝老師。謝謝范俊逸教授、張仁俊教授、李金鳳教授擔任我的口試委員，給予我許多論文上寶貴的指正與意見，我也從口試時的討論中獲得了一些靈感，由衷感激。

感謝密碼理論實驗室的志賢學長、順隆學長、鈞祥學長、用翔學長、輔國學長、佩娟學姊，在我的研究所生涯中，你們分享的經驗讓我受用無窮，謝謝你們。感謝同學禮鼎，不管在研究上或生活上都給了我很大的幫助，謝謝你。

此外，感謝系女籃的學妹們，你們用心地舉辦各種活動，讓我的研究生活充滿活力和歡笑，讓我真正感受到我們是永遠無法拆散的 team，謝謝妳們。

最後要感謝我的家人栽培我，無條件地支持我所有的決定，盡可能為我提供最大的幫助，也謝謝哥哥和姊姊的關心，你們的包容是我一生中最大的福氣。

謹以此論文獻給我摯愛的家人以及所有關心我的師長和朋友們，謝謝你們。

# Contents

# List of Tables

# List of Figures

# 1 Introduction

In this era of information, several issues have been given utmost attention: how data are stored, security of the data and information retrieval process; that is, how to retrieve data of use after they have been stored to the storage while remaining its' secrecy. Many services have been carried out to meet these needs. Recently, the most popular information technology vision, cloud computing, have risen with its numerous benefits: unlimited computing resources on demands, the ease for cloud users to build up a datacenter both publicly and privately, a hedge against data lost, and so on. Cloud computing services such as Amazon Elastic Compute Cloud can attract customers from enterprises to individual users who wish to save the effort of deploying much hardware which requires the capital outlays and human resources to maintain it.

Data Security has emerged from the issues which cloud storage services are facing. Cloud users should be assured that their data are secure against curious or malicious eavesdroppers who are not authorized by eligible users. Generally, cloud user stores plaintext on cloud storage server. Cloud storage server is therefore capable of searching on full text in any fashion. The cloud user then requests the server to return the data of interest. However, in some cases, even the data server is restricted from accessing full plaintext when the cloud users wish to store confidential information on the cloud storage. A trivial solution to data security is to have cloud users store encrypted data on the server. Upon receiving request for certain data, the server responded with the entire ciphertext back to the cloud user. Cloud users with correct decryption key are authorized users and have access to the plaintext. Table 1.1 compares the advantages and disadvantages between different data types stored on cloud storage server.

| Data type | Advantages | Disadvantages |
|---|---|---|
| Plaintext Message | Server can easily perform operations like sorting, searching, compressing to optimize computing performance. | Breach of security at the server side. The confidential content of cloud user could be leaked out by cloud server. |
| Encrypted Ciphertext | Users are assured the security of private data at server side. Only user-authorized party can decrypt the stored ciphertext. | Unable to access the ciphertext for server causes extra cost of storage and performance. Requires users to preprocess plaintext beforehand. |

Table 1.1 Comparison between Data Types on Cloud Storage

For users who require absolute data privacy, or say, private database, adopting encrypted ciphertext as their cloud storage data type is inevitable. Nevertheless, searching ability is magnified as the amount of data grows. In this case, server should be able to search on encrypted data and return exactly the data cipher which user is searching for. Traditional public-key encryption scheme is incapable of providing this functionality. Thus we need additional encryption scheme to accomplish this goal.

The term “searchable encryption” has been applied to represent encryption algorithms that provide searching functionality over encrypted data without possession of decryption key. Various forms of searchable encryption have been widely discussed in the past few years [27].

The paper is structured as follows: Section 2 introduces the mathematical background. Section 3 reviews the previous prominent public key searchable encryptions. Section 4 covers our design of public key searchable encryption with conjunctive queries. The paper concludes and proposes some future work ideas in Section 5.

# 2 Mathematical Background

In this chapter, we review elliptic curves and bilinear pairings.

## 2.1 Elliptic Curves

Suppose that $E: y^2 = x^3 + ax + b$ is an elliptic curve defined over a finite field $\mathrm{F_q}$ and $q$ is power of a prime $p > 3$. E has $q + 1 - t$ points in $\mathrm{F_q}$ and $-2\sqrt{q} \le t \le 2\sqrt{q}$. These points plus $\mathcal{O}$, an imaginary identity point at infinity, become a group with addition structure. The group is denoted as $E(F_q)$. That is,

$$E(F_q) = \{(x, y) \cup \mathcal{O} \mid y^2 = x^3 + ax + b,\ a, b \in F_q, x, y \in F_q\}.$$

The group addition operation is defined as follows. Given points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on the curve, we first draw the line through P and Q. The line intersects the curve in $S' = (x_3, y_3{}')$. We then reflect S' over the x-asis to obtain $S = P + Q = (x_3, -y_3') = (x_3, y_3)$. Suppose that λ is the slope of the line through P and Q, then the coordinates of $P + Q = (x_3, y_3)$ are $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$, where

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } \mathrm{P} \neq \mathrm{Q} \\ (3{x_1}^2 + a)/2y_1 & \text{if } \mathrm{P} = \mathrm{Q} \end{cases}$$

We also define $E(F_{q^d}) = \{(x,y) \cup \mathcal{O} \mid y^2 = x^3 + ax + b,\ a,b \in F_q, x,y \in F_{q^d}\}$. Suppose that $\overline{\mathrm{F_q}}$ is the algebraic closure of $F_q$, then $E(F_q) \subseteq E(F_{q^d}) \subseteq E(\overline{F_q})$.

Suppose that $r \mid \#E(F_q) = q + 1 - t$ , then we define $E[r] = \{P \in E(\overline{F_q}) | rP = \mathcal{O}\}$. $E[r]$ are called the r-torsion points. The r-torsion point plays an important role in pairing's definitions.

We can also find a smallest positive integer $k$ such that $r \mid q^k - 1$. $k$ is called the embedding degree. There are two important facts about the embedding degree. One is that $E[r] \subseteq E(F_{q^k})$ and then we can compute the r-torsion points in $E(F_{q^k})$ rather than in $E(\overline{F_q})$. The other fact is that $\mu_r \subseteq F_{q^k}$ where $\mu_r = \{x \in \overline{F_q} \mid x^r = 1\}$.

## 2.2 Rational functions and Divisors

$F_{q^k}[X,Y]$ represents the ring of polynomials in two variables $X,\ Y$ with coefficients in $F_{q^k}$. A rational function $h = f/g$ where $f,g \in F_{q^k}[X,Y]$ and $f$ is coprime to $g$.

Given an elliptic curve $E$ and a rational function $h = f/g$, we consider the points that $f(x,y) = 0$ and $(x,y) \in E(F_{q^k})$. We call those points zeroes of h. We also consider the points that $g(x,y) = 0$ and $(x,y) \in E(F_{q^k})$. Those points are called poles of $h$. In addition, the zeroes are the points where $E$ and $h$ intersect.

The divisor is a useful tool for keeping track of the zeros and poles [28]. We use divisors to indicate which points are zeros or poles and their orders for a rational function over an elliptic curve. A divisor $D$ on and elliptic curve $E$ is the finite linear combination of the formal symbols with integer coefficients:

$$D = \sum_{P \in E} n_P [P].$$

If $n_P > 0$, it indicates that $P$ is a zero, and if $n_P < 0$, it indicates that $P$ is a pole. We define $Div(E)$ as the group of divisors. For a divisor $D = \sum n_P[P]$, we define $supp(D) = \{P \in E \mid n_P \neq 0\}$ as the support of $D$, $deg(D) = \sum n_P$ as the degree of D, and $sum(D) = \sum n_P P$.

Now we consider only the set of divisors of degree zero. The set forms a subgroup $Div^0(E) \subset Div(E)$. Let f be a rational function. The evaluation of a rational function f on a divisor $D = \sum n_P[P]$ is defined by

$$f(D) = \prod_{P \epsilon supp(D)} f(P)^{n_p}.$$

The divisor of a rational function f is defined as $div(f) = \sum n_{P,f}(P)$ where $n_{P,f}$ is the zero or pole order of point $P$ on $f$. The degree of $div(f)$ must be zero [3]. A divisor $D \in Div^0(E)$ is principal if it is the divisor of a function. The following is an important fact.

**<u>Theorem 2.1 [28]</u>**

Let $E$ be an elliptic curve and $D$ be a divisor on $E$ with $deg(D) = 0$. Then there is a function $f$ on $E$ with $div(f) = D$ if and only if

$$sum(D) = \mathcal{O}.$$

## 2.3 The Tate Pairing

The Tate pairing and the Weil pairing [28] are two well-studied pairings. Under the same security level, The Tate pairing is generally considered more efficient than the Weil pairing.

Let $E$ be an elliptic curve defined over a finite field $F_q$ and q is power of a prime $p > 3$. Let $G$ be a cyclic subgroup of $E(F_q)$ of order $r$ which is coprime to $q$. The embedding degree is $k$ such that $r \mid q^k - 1$. The Tate pairing is a map

$$< P, Q >_r\!:\ E(F_{q^k})[r] \times E(F_{q^k})/rE(F_{q^k}) \rightarrow F_{q^k}^*/(F_{q^k}^*)^r.$$

$E(F_{q^k})[r]$ is defined as $E[r] \cap E(F_{q^k})$ and $rE(F_{q^k})$ is $\{\, rS \mid S \in E(F_{q^k}) \,\}$ and $(F_{q^k}^*)^r$ is $\{\, \alpha^r \mid \alpha \in F_{q^k} \,\}$. The groups $\mu_r$ and $F_{q^k}^*/(F_{q^k}^*)^r$ are isomorphic.

Let $P \in E(F_{q^k})[r]$ and let $Q \in E(F_{q^k})$. $Q$ represents a coset in $E(F_{q^k})/rE(F_{q^k})$. Let $f$ be a rational function with divisors $(f) = r[P] - r[\mathcal{O}]$. Choose a $S \in E(F_{q^k})$ such that $S \neq P, (P - Q), -Q\ or\ \mathcal{O}$. Let $\mathrm{D}$ be a divisor and $\mathrm{D = [Q + S] - [S]}$. The Tate pairing is defined to be

$$< \mathrm{P, Q} >_{\mathrm{r}} = \mathrm{f(D)}.$$

$\mathrm{f(D)} \in \mathrm{F}_{\mathrm{q}^{\mathrm{k}}}^{*}$ represents a coset in $\mathrm{F}_{\mathrm{q}^{\mathrm{k}}}^{*} / (\mathrm{F}_{\mathrm{q}^{\mathrm{k}}}^{*})^{\mathrm{r}}$. In fact, we often want to standardize the coset representative. Therefore the reduced Tate pairing is defined to be

$$\mathrm{t_r} = \ < \mathrm{P}, \mathrm{Q} >_{\mathrm{r}}{}^{\mathrm{q^k - 1/r}}.$$

The Tate pairing has bilinearity property and other important properties. See Theorem 2.2 [3].

**Theorem 2.2**

Let E be an elliptic curve defined over a finite field $F_q$ and $q$ is power of a prime $p > 3$. Let $G$ be a cyclic subgroup of $E(F_q)$ of order $r$ which is coprime to $q$. The embedding degree is $k$. The Tate pairing satisfies:

1. Bilinearity: For all $P,\ P_1,\ P_2 \in E(F_{q^k})[r]$ and $Q,\ Q_1,\ Q_2 \in E(F_{q^k})$,

   $$t_r(P_1 + P_2, Q) = t_r(P_1, Q)t_r(P_2, Q) \text{ and}$$
   $$\mathrm{t_r(P, Q_1 + Q_2) = t_r(P, Q_1)t_r(P, Q_2)}.$$

2. Non-degeneracy:

   $$\forall P, t_r(P, Q) = 1 \text{ if and only if } Q = \mathcal{O} \text{ and}$$
   $$\forall Q, t_r(P, Q) = 1 \text{ if and only if } P = \mathcal{O}.$$

To compute the Tate pairing, we need to evaluate a rational function $f$ that $(f) = r[P] - r[\mathcal{O}]$. The Miller's algorithm [24] can help us find the function and compute the result of the Tate pairing.

## 2.4 Supersingular curves and Distortion Maps

Suppose that $E: y^2 = x^3 + ax + b$ is an elliptic curve defined over a finite field $F_q$ and $q$ is power of a prime $p > 3$. E has $q + 1 - t$ points in $F_q$ and $-2\sqrt{q} \leq t \leq 2\sqrt{q}$. If p | t, then $E$ is said to be supersingular. Otherwise, $E$ is said to be ordinary. An important property of supersingular curves is that their embedding degrees are low. Their embedding degrees are from 1 to 6. Low embedding degree is crucial for the efficiency of computing a pairing. Another important property of supersingular curves is the existence of distortion maps.

A distortion map $\phi$ maps a point $P \in E(F_q)$ to a point $\phi(P) \in E(F_{q^k})$ such that $P$ and $\phi(P)$ are linearly independent. If $E$ is supersingular and $k > 1$, the distortion map exists. If $E$ is ordinary and $k > 1$, then no distortion map exists for curve E [21]. By using the distortion map, we can define the modified Tate pairing.

Let $E$ be a supersingular curve defined over a finite field $F_q$ and $q$ is power of a prime $p > 3$. Let $G$ be a cyclic subgroup of $E(F_q)$ of order $r$ which is coprime to $q$. The embedding degree is $k$ such that $r \mid q^k - 1$. A distortion map $\phi$ exists. The modified Tate pairing is a map $\hat{t_r}: G \times G \longrightarrow \mu_r$ and defined to be

$$\hat{t_r}(P, Q) = t_r\big(P, \phi\,(Q)\big).$$

We note that the first input and the second input of the modified Tate pairing are from the same group. Therefore we say the modified Tate pairing is symmetric.

Table 2.1 [3] contains some popular supersingular curves.

| $k$ | Elliptic curve data |
|---|---|
| 2 | $E: y^2 = x^3 + a$ over $F_p$, where $p$ is a prime and $p \equiv 2\ (mod\ 3)$<br>$E$ has $p+1$ points<br>Distortion map $(x, y) \longmapsto (\zeta_3 x, y)$, where ${\zeta_3}^3 = 1$. |
| 2 | $E: y^2 = x^3 + x$ over $F_p$, where $p$ is a prime and $p \equiv 3\ (mod\ 4)$<br>$E$ has $p+1$ points<br>Distortion map $(x, y) \longmapsto (-x, iy)$, where $i^2 = -1$. |

Table 2.1: Supersingular curves

In the rest of this thesis, we usually treat pairings as “black boxes.” It can help us focus on the design of the encryption scheme. Therefore, we now give an abstract definition of the pairing.

**Definition 2.3 (Bilinear Pairing)**

Let $G_1$ and $G_2$ be two additive cyclic elliptic-curve groups and $G_{\mathrm{T}}$ be a multiplicative cyclic group. $G_1$, $G_2$, and $G_{\mathrm{T}}$ are all of prime order $p$. Let $P$ be a generator of $G_1$ and $Q$ be a generator of $G_2$. A bilinear pairing is a map: $e: G_1 \times G_2 \rightarrow G_{\mathrm{T}}$ that satisfies the following properties:

1. Bilinearity,
2. Non-degeneracy,
3. Computability.

These properties are further discussed as follows:

**Bilinearity**

$$\forall\ P\ \in G_1, Q\ \in G_2\ \ and\ \ \forall a, b \in Z_p^*$$

$$e(aP, bQ) = e(P, Q)^{ab}$$

**Non-degeneracy**

$$\forall\ P \in G_1, P \neq 0$$

$$e(P, P) = g\ (gernerator\ of\ G_{\mathrm{T}})$$

That is,

$$P \neq 0 \Rightarrow e(P, Q) \neq 1$$

**Computability**

$\exists\ A$ , a polynomial-time algorithm

$\forall\ P\ \in G_1, Q\ \in G_2$

$A$ computes $e(P, Q)$ efficiently.

If $G_1 = G_2$, we have the non-degenerate symmetric bilinear pairing $e: G \times G\ \rightarrow\ G_{\mathrm{T}}$. Otherwise the pairing is called asymmetric. Tate pairing are generally considered more efficient than Weil pairing[3][28]. We note that symmetric pairing is usually realized as the modified Tate pairing and the asymmetric pairing is usually realized as the reduced Tate pairing.

# 3 Review of Searchable Encryptions

## 3.1 Searchable Encryption

Searchable encryption is a cryptosystem that enables the users to search over ciphertext without requiring the decryption key. Let the data be the documents which user wants to encrypt with; or in practice, the data is the symmetric encryption key that encrypt the documents. Searchable encryption transforms the data and a set of related keywords into the ciphertext. A trapdoor associated with a keyword is generated by the user to search over the ciphertext for the keyword. After some computation, if the user has legitimate decryption key, the trapdoor will match with the ciphertext which contains the keyword. The idea of searchable encryption is illustrated in Figure 3.1.

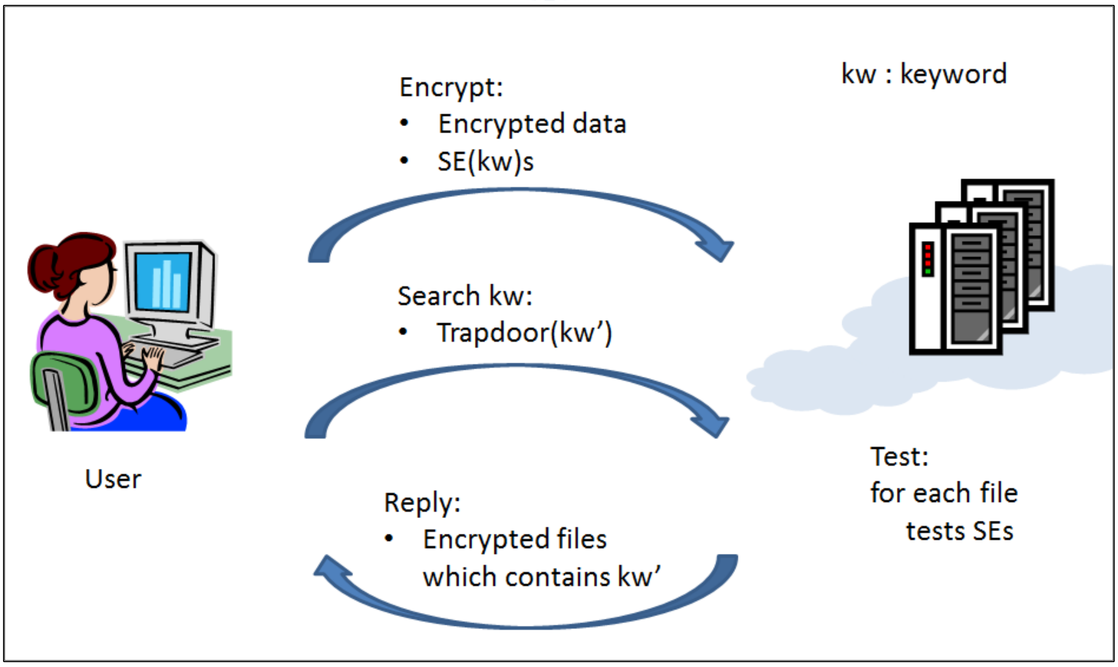

Figure 3.1 Searchable Encryption

Searchable encryption schemes can be categorized into symmetric key or public-key searchable encryption. The searchable encryption in the symmetric key setting allows only the owner of the secret key to create searchable ciphertext, while anyone can create searchable ciphertext using the public parameters in the public key setting. However, the symmetric key setting is generally faster than the public key setting.

The security of a searchable encryption can be shown by proving that a probabilistic polynomial-time algorithm $A$ differentiates the encrypted message and keywords from random data with negligible probability. The security model shows how much computing power the adversary $A$ can have. Various security models offer a trade-off between efficiency and security level. For symmetric key setting, a scheme must prove that searchable ciphertext and trapdoor do not reveal any information to adversary $A$. For public key setting, the searchable ciphertext and the trapdoor that does not match must be proved to reveal nothing to the adversary $A$. Two most-used models in the public key setting are the random oracle model and the standard model. The random oracle model is used when it comes to avoiding complications, while the standard model is stronger but more costly.

The efficiency of a searchable encryption scheme can be evaluated in the following aspects:

**Computational complexity**

The complexity needed to create searchable ciphertext, to generate trapdoor, and to search.

**Communication complexity**

The complexity needed for searchable ciphertext be send/returned between the user and the server.

**Storage complexity**

The complexity needed to store public/private parameters, searchable ciphertext and trapdoor, as well as the storage needed by the server while performing search.

According to the key setting and the security models, Figure 3-1 depicts the searchable encryption category along with the prominent schemes in this category. In this paper, we focus on public key setting searchable encryption schemes.

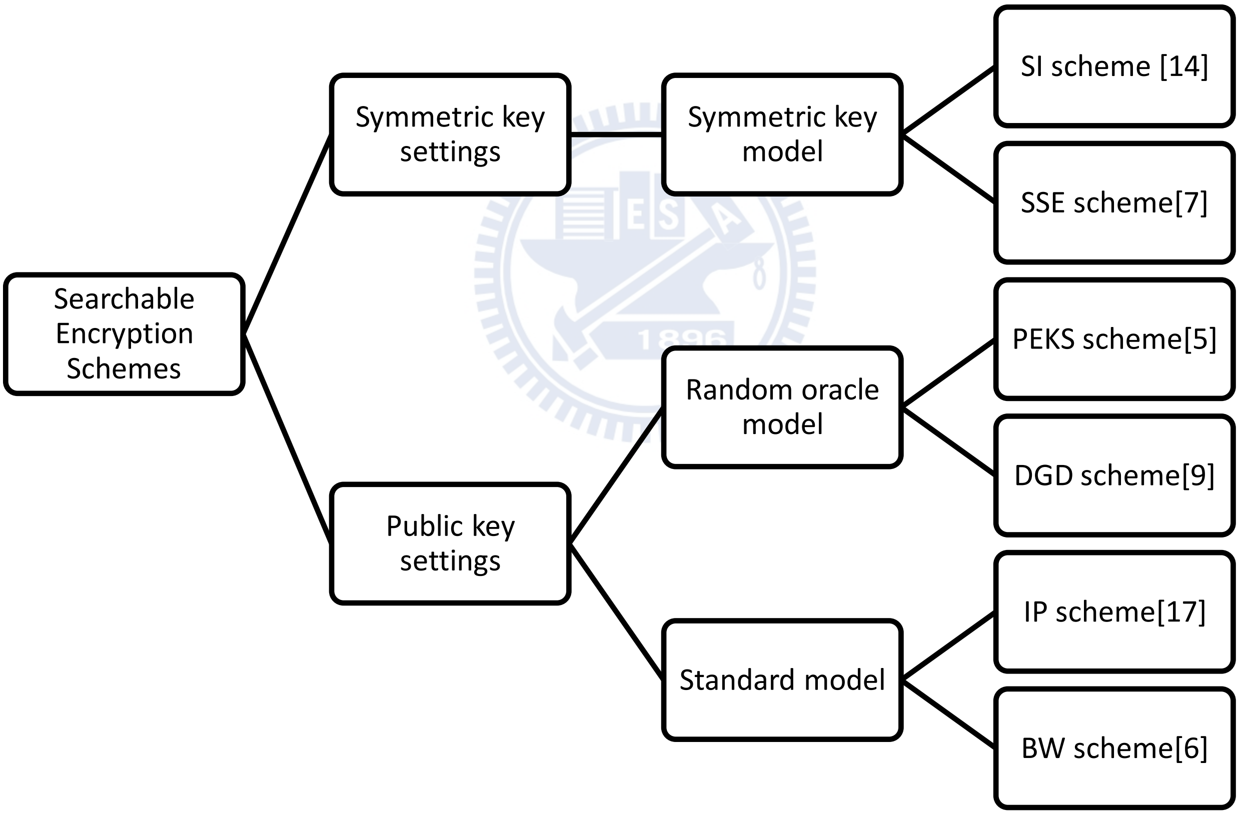

Figure 3.2 Prominent Schemes of Searchable Encryption

## 3.2 Public Key Encryption with Keyword Search

Public key encryption with Keyword Search (PEKS) is introduced by Boneh et al. [5] It is the first asymmetric searchable encryption scheme that can be applied to email gateway routing. The word “public-key” points out that anyone can encrypt a message with its keywords using receiver’s public key. Suppose Bob wants to send Alice an email with keywords $W_1, \dots, W_2$ using Alice’s public key $A_{pub}$. Bob sends ciphertext looked like this:

$$\left[E_{A_{pub}}[email], PEKS\left(A_{pub}, W_1\right), \dots, PEKS\left(A_{pub}, W_k\right)\right]$$

where $k$ is a relatively small number. Then Alice can send a trapdoor $T_W$ to the email gateway server to search all the ciphertext containing keyword $W$ using her private key. The server gains no knowledge about the encrypted emails except which ciphertext contains keyword $W$. The server then sends back the set of ciphertext that contains keyword $W$ to Alice.

**<u>Definition 3.1 (Public-key Encryption with Keyword Search)</u>**

A public-key searchable encryption scheme that consists of the following polynomial time randomized algorithms:

1. KeyGen$(1^k)$: takes a security parameter k and outputs public/private keys $A_{pub}, A_{priv}$.
2. PEKS$(A_{pub}, W)$: takes a public key $A_{pub}$ and a word $W$, outputs a searchable encryption of $W$.
3. Trapdoor$(A_{priv}, W)$: takes user’s private key $A_{priv}$ and a keyword $W$, produces a trapdoor $T_W$.

4. $\text{Test}(A_{pub}, S, T_{W'})$: takes a public key $A_{pub}$, a searchable encryption $S = \text{PEKS}(A_{pub}, W)$, and a trapdoor $T_{W'}$, outputs the test result: if $W = W'$, return ‘yes’; else return ‘no’.

The concrete construction of PEKS based on Decision Diffie-Hellman assumption is as follows:

### $\mathbf{KeyGen}(\mathbf{1}^{\boldsymbol{k}})$

The input security parameter determines the size $p$ of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, from the symmetric bilinear pairing $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_\mathrm{T}$. Two hash functions $H_1: \{0,1\}^* \rightarrow \mathbb{G}$ and $H_2: \mathbb{G}_T \rightarrow \{0,1\}^{\log p}$ are defined. Then randomly choose $\alpha \in \mathbb{Z}_p^*$ and a generator $g \in \mathbb{G}$. Then, output public/private key pair $A_{pub} = [g, h = g^{\alpha}]$ and $A_{priv} = \alpha$.

### $\mathbf{PEKS}(\boldsymbol{A_{pub}}, \boldsymbol{W})$

Randomly choose $\mathrm{r} \in \mathbb{Z}_p^*$, and then compute $\mathrm{t} = e(H_1(W), h^{\mathrm{r}}) \in \mathbb{G}_T$.

Output $\text{PEKS}\big(A_{pub}, W\big) = [g^r, H_2(t)]$.

### $\mathbf{Trapdoor}(\boldsymbol{A_{priv}}, \boldsymbol{W})$

Compute the trapdoor for keyword $W$ as $T_W = H_1(W)^{\alpha} \in \mathbb{G}$.

### $\mathbf{Test}(\boldsymbol{A_{pub}}, \boldsymbol{S}, \boldsymbol{T_W})$

Here $S = \text{PEKS}\big(A_{pub}, W\big)$ . Let $S = [A, B]$ and test if

$H_2(e(T_W, A)) = B$. If the '=' holds, return 'yes'; else return 'no'.

Due to the constraints of its design, PEKS scheme is applicable to search on only small number of keywords instead of the entire file.

## 3.3 Multi-user Searchable Data Encyrption

A Multi-user Searchable Data Encryption scheme (DGD) proposed by Dong et al.[9] is a cryptosystem that offers functionalities of sharing encrypted data on a untrusted server among a group of authorized user, performing keyword search on encrypted data without decryption key, and adding/revoking users without restarting the service. Users rely on the data storage server to honestly perform searching calculation for them but do not trust the server with data content – the server is considered to be "honest but curious." Three parties are involved in DGD system:

1. Users: The authorized users are able to read/write/search over encrypted data on untrusted server. The authorized users are fully trusted. After revocation, the revoked user will no longer be able to access the data.
2. Server: The server is responsible for processing the received encrypted data, storing the encrypted data, searching on receiving user's query and return the encrypted data that contains the query keyword.
3. Key management server (KMS): The fully trusted KMS is responsible for generating/revoking user keys. Compare to untrusted data server, securing the KMS requires less effort. Also, the KMS can be kept offline most of the time.

Before introducing the multi-user searchable data encryption, we first introduce two definitions: negligible function and pseudorandom function.

**Definition 3.2 (Negligible Function)**

A function $\mathrm{negl}(x)$ is negligible if for every positive polynomial $f(\cdot)$ there exists an integer $N$ such that for all $> N$, $\mathrm{negl}(x) < \frac{1}{f(x)}$ .

**Definition 3.3 (Pseudorandom Function)**

A function $f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a pseudorandom function if for all probabilistic polynomial time algorithm $A$, there exists a negligible function $\mathrm{negl}$ such that

$$\left|\Pr\left[A^{f_k(\cdot)} = 1\right] - \Pr\left[A^{F(\cdot)} = 1\right]\right| < \mathrm{negl}(n)$$

where random key $k \overset{R}{\leftarrow} \{0,1\}^n$ and function $F: \{0,1\}^* \overset{R}{\rightarrow} \{0,1\}^*$.

Now let’s see the definition of the DGD scheme.

**Definition 3.4 (Multi-user Searchable Data Encryption)**

A searchable encryption scheme that consists of the following probabilistic polynomial time randomized algorithms:

1. $\mathrm{Init}(1^k)$: The KMS takes the security parameter $k$ and outputs public key $Params$ and a master key set $MSK$.
2. $\mathrm{Keygen}(MSK, i)$: The KMS takes the master key set $MSK$ and a user’s identity $i$, generates the secret key set $\mathcal{K}_{u_i}, \mathcal{K}_{s_i}$. User side key $\mathcal{K}_{u_i}$ is then securely sent to the user $i$, and server side key $\mathcal{K}_{s_i}$ is sent to the server.
3. $\mathrm{Enc}\left(\mathcal{K}_{u_i}, D, kw(D)\right)$: The user $i$ uses his user side key $\mathcal{K}_{u_i}$ to en-

crypt a document $D$ with a set of associated keywords $kw(D)$. The output is user-side ciphertext $c_i^*\big(D, kw(D)\big)$.

4. $\mathrm{Re-enc}\left(i, \mathcal{K}_{s_i}, c_i^*\big(D, kw(D)\big)\right)$ : On receiving the ciphertext $c_i^*\big(D, kw(D)\big)$ from user $i$, the server fetches the server side key $\mathcal{K}_{s_i}$, and outputs re-encrypted ciphertext $c\big(D, kw(D)\big)$.
5. $\mathrm{Trapdoor}\big(\mathcal{K}_{u_i}, w\big)$: The user $i$ uses his user side key $\mathcal{K}_{u_i}$ to generate a trapdoor $T_i(w)$ related to a keyword $w$
6. $\mathrm{Search}\big(i, T_i(w), E(D), \mathcal{K}_{s_i}\big)$: The server takes as input the trapdoor $T_i(w)$ and user's identity $i$, then test for each $c\big(D, kw(D)\big) \in E(D)$ if keyword $w \in kw(D)$. If 'yes', the server invokes pre-decrypt algorithm to obtain $c_i'(D)$ and send $c_i'(D)$ to the user $i$.
7. $\mathrm{Dec}\left(\mathcal{K}_{u_i}, c_i'(D)\right)$: The user takes his user key $\mathcal{K}_{u_i}$ , and decrypts $c_i'(D)$ to obtain data $D$.
8. $\mathrm{Revoke}(i)$: Given $i$, the data server updates the user-key mapping set $\mathcal{K}s = \mathcal{K}s \setminus (i, \mathcal{K}_{s_i})$.

The DGD scheme is based on proxy cryptography. In the following sections, we will first review ElGamal encryption scheme $\mathcal{E}$, then describe the proxy encryption scheme using the algorithm in ElGamal encryption scheme $\mathcal{PE}$. Next, the keyword encryption scheme $\mathcal{KE}$ is defined. Finally, with $\mathcal{PE}$ and $\mathcal{KE}$ schemes, the Multi-user Searchable Data Encryption are presented.

### 3.3.1 ElGamal Proxy Encryption

Before defining ElGamal proxy encryption scheme, the ElGamal encryption scheme $\mathcal{E}$ is defined as follows:

$\boldsymbol{\mathcal{E}} - \mathbf{Init}(\mathbf{1}^{\boldsymbol{k}})$

Choose prime numbers $p, q$ such that $q \mid p - 1$, a cyclic group $\mathbb{G}$ with generator $g$ such that $\mathbb{G}$ is the unique order $q$ subgroup of $\mathbb{Z}_p^*$. Choose $x \overset{R}{\leftarrow} \mathbb{Z}_q$ and compute $h = g^x$. Outputs the public key $pk = (\mathbb{G}, g, h, q)$ and private key $sk = x$.

$\boldsymbol{\mathcal{E}} - \mathbf{Enc}(\mathbf{pk}, \mathbf{m})$

Choose $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and output ciphertext $c(m) = (g^r, h^r m)$.

$\boldsymbol{\mathcal{E}} - \mathbf{Dec}(\boldsymbol{pk}, \boldsymbol{m})$

Decrypt ciphertext as $h^r m \cdot (g^r)^{-x} = g^{rx - rx} m = m$.

The proxy encryption scheme $\mathcal{PE}$ consists of 6 algorithms:

$\boldsymbol{\mathcal{PE}} - \mathbf{Init}(\mathbf{1}^{\boldsymbol{k}})$

KMS runs $\boldsymbol{\mathcal{E}} - \mathbf{Init}(\mathbf{1}^{\boldsymbol{k}})$ to obtain $(\mathbb{G}, g, q, x)$, then it outputs public parameters $(\mathbb{G}, g, q)$, and master key $\mathrm{MSK} = x$.

$\boldsymbol{\mathcal{PE}} - \mathbf{Keygen}(\boldsymbol{MSK}, \boldsymbol{i})$

For each user $i$, KMS chooses $x_{i1} \overset{R}{\leftarrow} \mathbb{Z}_q$ and computes $x_{i2} = x - x_{i1}$. Then the KMS securely transmits $x_{i1}$ to the user $i$ and $(i, x_{i2})$

to the proxy server.

$\boldsymbol{\mathcal{PE}} - \mathbf{U} - \mathbf{Enc}(\boldsymbol{x_{i1}}, \boldsymbol{m})$

The user chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and outputs ciphertext $c^*(m) = (g^r, g^{rx_{i1}}m)$. Then the user sends the ciphertext to the proxy server.

$\boldsymbol{\mathcal{PE}} - \mathbf{P} - \mathbf{Enc}\left(\boldsymbol{i}, \boldsymbol{x_{i2}}, \boldsymbol{c_i^*}(\boldsymbol{m})\right)$

In this proxy re-encryption algorithm, the proxy server finds $(i, x_{i2})$ where $x_{i2}$ is user‘s server side key, and computes $(g^r)^{x_{i2}} \cdot g^{rx_{i1}}m = g^{rx}m$. The stored ciphertext becomes $\mathrm{c}(m) = (g^r, g^{rx}m)$.

$\boldsymbol{\mathcal{PE}} - \mathbf{P} - \mathbf{Dec}\left(\boldsymbol{j}, \boldsymbol{x_{j2}}, \mathbf{c}(\boldsymbol{m})\right)$

In this proxy side decryption algorithm, the proxy server finds $j$’s server side key $x_{j2}$ and computes $g^{rx}m \cdot (g^r)^{-x_{j2}} = g^{rx_{j1}}m$. The ciphertext is partially decrypted as $c'(m) = (g^r, g^{rx_{j1}}m)$ and is sent to user $j$.

$\boldsymbol{\mathcal{PE}} - \mathbf{U} - \mathbf{Dec}\left(\boldsymbol{x_{j1}}, \boldsymbol{c_j'}(\boldsymbol{m})\right)$

User fully decrypts the ciphertext as $g^{rx_{j1}}m \cdot (g^r)^{-x_{j1}} = m$.

## 3.3.2 Keyword Encryption

Derived from the proxy encryption scheme, the keyword encryption scheme is capable of securely encrypting keywords, allowing user to search over the encrypted data by generating trapdoors. The keyword encryption scheme $\mathcal{KE}$ is defined as follows:

**$\mathcal{KE} - \mathbf{Init}(\mathbf{1}^{k})$**

The KMS runs $\mathcal{PE} - \mathbf{Init}(\mathbf{1}^{k})$ to obtain $(\mathbb{G}, g, q, x)$. Compute $h = g^{x}$ and choose hash function $H$, a pseudorandom function $f$ and a random key s for $f$. Then the KMS outputs public parameters $(\mathbb{G}, g, q, h, H, f)$, and master key $\mathrm{MSK} = (x, s)$.

**$\mathcal{KE} - \mathbf{Keygen}(\boldsymbol{MSK}, \boldsymbol{i})$**

For each user $i$, the KMS runs $\mathcal{PE} - \mathbf{Keygen}(\boldsymbol{MSK}, \boldsymbol{i})$ to obtain $x_{i1}, x_{i2}$. Then the KMS securely transmits $(x_{i1}, s)$ to the user $i$ and $(i, x_{i2})$ to the proxy server.

**$\mathcal{KE} - \mathbf{U} - \mathbf{Enc}(\boldsymbol{x_{i1}}, \boldsymbol{kw})$**

The user chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$. The user side trapdoor for keyword $kw$ is encrypted as $c^*(kw) = (\widehat{c_1}, \widehat{c_2}, \widehat{c_3}) = \left(g^{r+\sigma}, (\widehat{c_1})^{x_{i1}}, H(h^{r})\right)$ where $\sigma = f_s(kw)$. Then the user sends the ciphertext $c^*(kw)$ to the proxy server.

**$\mathcal{KE} - \mathbf{P} - \mathbf{Enc}\left(\boldsymbol{i}, \boldsymbol{x_{i2}}, \boldsymbol{c_i^*}(\boldsymbol{kw})\right)$**

The proxy server computes trapdoor $c(kw) = (c_1, c_2)$ such that $c_1 = (\widehat{c_1})^{x_{i2}} \cdot \widehat{c_2} = \widehat{c_1}^{\,x} = (g^{r+\sigma})^{x} = h^{r+\sigma}$ and $c_2 = H(h^{r})$.

Because the keyword encryption scheme is used to generate searchable encryption which does not need to be decrypted, hence there is no decrypting algorithm.

### 3.3.3 Multi-user Searchable Data Encryption

Combining the previous $\mathcal{PE}$ and $\mathcal{KE}$ algorithms, the Multi-user Searchable Data Encryption $\mathcal{SE}$ is described as the following 8 algorithms.

$\mathbf{Init}(\mathbf{1}^{\boldsymbol{k}})$

The KMS runs $\boldsymbol{\mathcal{KE}} - \mathbf{Init}(\mathbf{1}^{\boldsymbol{k}})$ to obtain public parameters $(\mathbb{G}, g, q, h, H, f)$, and master key $\text{MSK} = (x, s)$.

$\mathbf{Keygen}(\boldsymbol{MSK}, \boldsymbol{i})$

For each user $i$, the KMS runs $\boldsymbol{\mathcal{KE}} - \mathbf{Keygen}(\boldsymbol{MSK}, \boldsymbol{i})$ to obtain $\mathcal{K}_{u_i}, \mathcal{K}_{s_i}$. Then the KMS securely transmits $\mathcal{K}_{u_i}$ to the user $i$ and $(i, \mathcal{K}_{s_i})$ to the proxy server. The server side user-key mapping set is updated as $\mathcal{K}s = \mathcal{K}s \cup (i, \mathcal{K}_{s_i})$.

$\mathbf{Enc}\left(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_i}}, \boldsymbol{D}, \boldsymbol{kw}(\boldsymbol{D})\right)$

The user calls $c_i^*(D) = \boldsymbol{\mathcal{PE}} - \mathbf{U} - \mathbf{Enc}\left(\boldsymbol{i}, \boldsymbol{\mathcal{K}}_{u_i}, \boldsymbol{D}\right)$ to encrypt data $D$, and compute $c_i^*(kw) = \boldsymbol{\mathcal{KE}} - \mathbf{U} - \mathbf{Enc}\left(\boldsymbol{x_{i1}}, \boldsymbol{kw}(\boldsymbol{D})\right)$ for each for keyword $kw \in kw(D)$. The user side ciphertext is

$$c_i^*\left(D, kw(D)\right) = \left(c_i^*(D), c_i^*(kw_1), \dots, c_i^*(kw_k)\right)$$

where $k = |kw(D)|$.

$\mathbf{Re-enc}\left(\boldsymbol{i}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_i}}, \boldsymbol{c_i^*}\left(\boldsymbol{D}, \boldsymbol{kw}(\boldsymbol{D})\right)\right)$

The proxy server finds $\mathcal{K}_{s_i} = (i, x_{i2})$, the server side key of user $i$. Then the server invokes $c(D) = \boldsymbol{\mathcal{PE}} - \mathbf{P} - \mathbf{Enc}\left(\boldsymbol{i}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_i}}, \boldsymbol{c_i^*}(\boldsymbol{m})\right)$, and the server calls $c(kw_k) = \boldsymbol{\mathcal{KE}} - \mathbf{P} - \mathbf{Enc}\left(\boldsymbol{i}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_i}}, \boldsymbol{c_i^*}(\boldsymbol{kw_k})\right)$ for each $c_i^*(kw_k)$. The re-encrypted data $c\big(D, kw(D)\big) = \big(c(D), c(kw_1), \dots, c(kw_k)\big)$ is then inserted into the data storage $E(D) = E(D) \cup c\big(D, kw(D)\big)$.

### Trapdoor $\left(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_j}}, \boldsymbol{w}\right)$

The user $j$ chooses random number $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and uses his user side key $\mathcal{K}_{u_j} = \left(x_{j1}, s\right)$ to compute a trapdoor $T_j(w) = (t_1, t_2)$ for a keyword $w$, where $t_1 = g^{-r} g^{\sigma_w}$, $t_2 = h^r g^{-x_{j1} r} g^{x_{j1}\sigma_w} = g^{x_{j2} r} g^{x_{j1}\sigma_w}$, and $\sigma_w = f_s(w)$.

### Search $\left(\boldsymbol{j}, \boldsymbol{T_j}(\boldsymbol{w}), \boldsymbol{E}(\boldsymbol{D}), \boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}}\right)$

The server perform search on receiving trapdoor $T_j(w) = (t_1, t_2)$ from the user $j$ with $\mathcal{K}_{s_j} = x_{j2}$. The server first compute $T = t_1^{x_{j2}} \cdot t_2 = g^{x\sigma_w}$. Then for each keyword cipher $c(kw) = (c_1, c_2) = (h^{r+\sigma}, H(h^r))$ in every ciphertext $c\big(D, kw(D)\big) \in E(D)$, test if $c_2 = H(c_1 \cdot T^{-1})$; ‘true’ implies $w = kw$ great probability, or say, a match is found. The server then partially decrypt all matched encrypted data $c(D)$ by invoking $c_j'(D) = \boldsymbol{\mathcal{PE}} - \mathbf{P} - \mathbf{Dec}\left(\boldsymbol{j}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}}, \mathbf{c}(\boldsymbol{D})\right)$. Note that $c(kw_k)$ does not need to be decrypted.

$\mathbf{Pre-dec}\left(\boldsymbol{j}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}}, \mathbf{c}(\boldsymbol{D})\right)$

The server runs $c_j'(D) = \boldsymbol{\mathcal{PE}} - \mathbf{P} - \mathbf{Dec}\left(\boldsymbol{j}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}}, \mathbf{c}(\boldsymbol{D})\right)$ to partially decrypt the encrypted ciphertext and sends $c_j'(D)$ to user $j$.

$\mathbf{Dec}\left(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_j}}, \boldsymbol{c_j'}(\boldsymbol{D})\right)$

User $j$ fully decrypts the pre-decrypted ciphertext $c_j'(D)$ by calling $D = \boldsymbol{\mathcal{PE}} - \mathbf{U} - \mathbf{Dec}\left(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_j}}, \boldsymbol{c_j'}(\boldsymbol{D})\right)$.

$\mathbf{Revoke}(\boldsymbol{i})$

To revoke user $i$, the data server simply updates the user-key mapping set $\mathcal{K}s = \mathcal{K}s \setminus (i, \mathcal{K}_{s_i})$.

The correctness of the searching algorithm depends on the collision resistance of hash function $H$. Hence, there exists a negligible function such that

$$\Pr[Search\ algorithm\ returns\ 'true' with\ w \neq kw] < negl(k)$$

## 3.4 Hidden-Vector Encryption

Boneh and Waters[6] proposed a public-key encryption system that utilized Hidden Vector Encryption (HVE) such that conjunctive equality, comparison, range, and subset queries are allowed. We call it ”HVE” scheme. In HVE scheme, the ciphertext $C$ is related to a vector $\boldsymbol{x} \in \{0,1\}^n$, and the key is related to a vector $\boldsymbol{y} \in \{0,1,*\}^n$ where the notation “ * ” represents “don’t care”. Both $\boldsymbol{x}$ and $\boldsymbol{y}$ are “hidden vector” that contain keywords implicitly. A ciphertext can be decipher once all entries of $\boldsymbol{y}$ except * (don’t care) on a private key match the corresponding entries of the vector $\boldsymbol{x}$ on the ciphertext. Symmetric pairing setting with composite group order is used to construct HVE. Here we introduce the latter scheme, Hidden-Vector encryption with groups of prime order (IP scheme), introduced by Iovino and Persiano[17]. The IP scheme apply the reductions of the original HVE to its construction to obtain a more efficient scheme supporting conjunctions of equality queries, range queries and subset queries.

**Definition 3.2 (Hidden Vector Encryption Scheme)**

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ are strings of length $n$ where $\boldsymbol{x} \in \{0,1\}^n$ and $\boldsymbol{y} \in \{0,1,*\}^n$. Define a predicate $P_x(\boldsymbol{y}) = 1$ if and only if $\boldsymbol{x}_i = \boldsymbol{y}_i$ or $\boldsymbol{y}_i = *$ , for $i = 1, \cdots, n;$ $P_x(\boldsymbol{y}) = 0$ otherwise. An HVE is a set of probabilistic polynomial-time algorithms (Setup,Enc,KeyGeneration,Dec) :

1. $\text{Setup}(1^k, n)$: Take the i security parameter $k$ and the attribute length $n = \text{poly}(k)$ and output the public key set $Pk$ and a master key set $MSK$.
2. $\text{Enc}(Pk, M, \boldsymbol{x})$: Take as input the public key set $Pk$, the plaintext $M \in \mathbb{G}_T$, and the attribute vector $\boldsymbol{x} \in \{0,1\}^n$. Output the ciphertext $C_{t_x}$.
3. $\text{KeyGeneration}(Msk, \boldsymbol{y})$: Take as input the master key set $MSK$ and string $\boldsymbol{y} \in \{0,1,*\}^n$. Output the decryption key $K_{\boldsymbol{y}}$.
4. $\text{Dec}(Pk, K_y, C_{t_x})$: Take as input the public key set $Pk$, the ciphertext $C_{t_x}$, and the secret key $K_{\boldsymbol{y}}$. Output the message $M$.

The concrete construction of the IP scheme is stated as follows:

**Setup$(\mathbf{1}^{\boldsymbol{k}}, \boldsymbol{n})$**

Take the input security parameter $k$ and the attribute length $n = \mathrm{poly}(k)$. Choose an instance $I = \{q, \mathbb{G}, \mathbb{G}_T, g, e\}$ and $\mathrm{y} \overset{R}{\leftarrow} \mathbb{Z}_q$, where $q$ is the group order of $\mathbb{G}$ and $\mathbb{G}_T$, $e$ is a symmetric bilinear pairing $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$ and $g$ is a generator of $\mathbb{G}$. Set $Y = e(g,g)^{\mathrm{y}}$. Choose random numbers $t_i, v_i, r_i, m_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and set $\mathrm{T_i = g^{t_i}}$, $\mathrm{V_i = g^{v_i}}$, $\mathrm{R_i = g^{r_i}}$, $\mathrm{M_i = g^{m_i}}$ for $i = 1, \cdots, n$. Then output the public key set

$$Pk = [\, I, Y, (T_i, V_i, R_i, M_i)_{i=1}^{n} \,]$$

and the master key set

$$MSK = [\, \mathrm{y},\ (t_i, v_i, r_i, m_i)_{i=1}^{n} \,].$$

**Enc$(\boldsymbol{Pk}, \boldsymbol{M}, \boldsymbol{x})$**

Take as input the public key set $Pk$, the plaintext $M \in \mathbb{G}_T$, and the attribute vector $\boldsymbol{x} \in \{0,1\}^n$. Choose random $s \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $s_i \overset{R}{\leftarrow} \mathbb{Z}_q^*$ for $i = 1, \cdots, n$ and compute the ciphertext $C_{t_x} = [\, \Omega,\ C_0, (X_i, W_i)_{i=1}^{n} \,]$, where $\Omega = M \cdot Y^{(-s)}$, $C_0 = g^s$ and

$$X_i = \begin{cases} {T_i}^{s-s_i}\ , & if\ \ \boldsymbol{x}_i = 1; \\ {R_i}^{s-s_i}\ , & if\ \ \boldsymbol{x}_i = 0; \end{cases} \qquad W_i = \begin{cases} {V_i}^{s}\ , & if\ \ \boldsymbol{x}_i = 1; \\ {M_i}^{s}\ , & if\ \ \boldsymbol{x}_i = 0; \end{cases}$$

Then, return the ciphertext $C_{t_x}$.

**KeyGeneration$(\boldsymbol{Msk}, \boldsymbol{y})$**

Take as input the master key set $MSK$ and string $\boldsymbol{y} \in \{0,1,*\}^n$. Denote $S_{\boldsymbol{y}}^1$ and $S_{\boldsymbol{y}}^0$ to be the set of indices $i$ such that $\boldsymbol{y_i} = 1$ and

$\boldsymbol{y_i} = 0$. Let $S_{\boldsymbol{y}} = S_{\boldsymbol{y}}^1 \cup S_{\boldsymbol{y}}^0$ be the set of indices $i$ for which $\boldsymbol{y_i} \neq *$. If $S_{\boldsymbol{y}} = \emptyset$, that is, $\boldsymbol{y} = (*, \dots, *)$, let $K_{\boldsymbol{y}} = g^{\mathrm{y}}$. Else, for each $i \in S_{\boldsymbol{y}}$, choose $\alpha_i \overset{R}{\leftarrow} \mathbb{Z}_q^*$ at random such that $\sum_{i \in S_y} \alpha_i = \mathrm{y}$, where $\mathrm{y}$ is from the $MSK$. Compute $K_{\boldsymbol{y}} = (Y_i, L_i)_{i=1}^n$ where

$$Y_i = \begin{cases} g^{\frac{\alpha_i}{t_i}} & \text{, if } \boldsymbol{y}_i = 1; \\ g^{\frac{\alpha_i}{r_i}} & \text{, if } \boldsymbol{y}_i = 0; \\ \emptyset & \text{, if } \boldsymbol{y}_i = *; \end{cases} \qquad L_i = \begin{cases} g^{\frac{\alpha_i}{v_i}} & \text{, if } \boldsymbol{y}_i = 1; \\ g^{\frac{\alpha_i}{m_i}} & \text{, if } \boldsymbol{y}_i = 0; \\ \emptyset & \text{, if } \boldsymbol{y}_i = *; \end{cases}$$

Then, output the decryption key $K_{\boldsymbol{y}}$ relative to attribute vector $\boldsymbol{y}$.

**Dec($\boldsymbol{Pk}, \boldsymbol{K_y}, \boldsymbol{C_{t_x}}$)**

Take as input the public key set $Pk$, the ciphertext $C_{t_x}$, and the secret key $K_y$. If $S_{\boldsymbol{y}} = \emptyset$, then $K_y = g^{\mathrm{y}}$, decrypt the ciphertext $C_{t_x}$ as

$$\mathrm{Dec}(Pk, K_y, C_{t_x}) = \Omega \cdot e(C_0, K_y)$$

Else, decrypt the ciphertext $C_{t_x}$ as

$$\mathrm{Dec}(Pk, K_y, C_{t_x}) = \Omega \cdot \prod_{i \in S_y} e(X_i, Y_i)\, e(W_i, L_i)$$

If predicate $P_x(\boldsymbol{y}) = 1$, then the decryption result is the plaintext $M$.

# 4 Our Construction

In Section 4.1 we described our construction in detail. We then introduce query applications of our design in Section 4.2. We give an simplified example with smaller numbers to our construction in Section 4.3. In section 4.4 we further discuss some related issues. And finally in Section 4.5, we give out our experiment results.

## 4.1 Public Key Searchable Encryption with Conjunctive Queries

We construct a searchable encryption scheme on elliptic curve groups, based on El Gamal Proxy Re-encryption and Hidden Vector Encryption. Users can share encrypted data among all authorized users while users are able to perform conjunctive keyword search. In our construction, authorized user share encrypted data over the data server that supports the following operations:

**Get** – The user requests the shared data with its id.
**Search** – The user asks the data server to perform conjunctive keyword search by sending a query trapdoor associated with the keywords.
**Insert** – The user inserts new data into the data server by running the data encryption algorithm to encrypt the data and the keywords.
**Remove** – The user requests the data server to remove encrypted data of certain id and its related keyword encryptions.

Since the data server – or called the proxy server since it stand as a proxy between users - is considered to be "honest and curious" which points out that the server will perform the search operation honestly but is curious about the data content. While performing the search operation for users, it is important that the data server gains no other information except:

1. which user sent the query, and
2. the set of encrypted documents which contain the queried keywords

That is, the data server will learn nothing about the data content, keywords to be queried and other information.

In our design, the authorized users are able to:

**Encrypt** – Users encrypt data with the associating keywords and pass it to the data server.

**Query** – Users query for keywords conjunctively over the encrypted data on the data server by producing a trapdoor related to the keywords.

**Decrypt** – Users decrypt the encrypted data that is returned from the data server.

Note that only authorized users in possession of a secret key can do the above operations. The user's secret key is called user side key, which is generated and distributed securely to the users by a Key Management Server (KMS), while the corresponding server side key is securely transmits to the data sever by the KMS. Two keys – the user side key and the server side key – are related with a master key that is held secretly by the KMS. Hence, the KMS should keep the master key secure in order to keep the entire system free from attack.

We assume no authorized user reveals his user side key to the data server; otherwise the data server can reconstruct the master key by multiplying the user side key with the server side key related to it. We also assume there is an impartial KMS which keeps master key secret and reveals nothing but the public parameters. Under these assumptions, we build up our construction for authorized users to store and share data on untrusted server without revealing the data content to the data server, while conjunctive queries over the encrypted data is supported by the data server.

Each algorithm in our searchable encryption scheme consists of two parts: an elliptic curve proxy encryption part to encrypt the symmetric session key that encrypts the data, and a hidden-vector encryption part to generate the conjunctive query searchable encryptions related the keywords of data. We give the definition of our construction as follows:

**Definition 4.1**

**(Public Key Searchable Encryption with Conjunctive Queries)**

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be strings of length $n$ where $\boldsymbol{x} \in \{0,1\}^n$ and $\boldsymbol{y} \in \{0,1,*\}^n$. Let $\boldsymbol{x}(D)$ be the attribute vector related to data $D \in G_T$, and $E(D)$ be encrypted data on data server. Define a predicate $P_x(\boldsymbol{y}) = 1$ if and only if $\boldsymbol{x}_i = \boldsymbol{y}_i$ or $\boldsymbol{y}_i = *$ , for $i = 1, \cdots, n;$ $P_x(\boldsymbol{y}) = 0$ otherwise. We construct a searchable encryption scheme consisting of the following nine algorithms:

1. Init$(1^k, n)$: The KMS takes the security parameter $k$ and attribute length $n = \text{poly}(k)$, then outputs public key $Params$ and a master key set $MSK$.
2. Keygen$(MSK, i)$: The KMS takes the master key set $MSK$ and a user's identity $i$, generates the secret key set $\mathcal{K}_{u_i}, \mathcal{K}_{s_i}$. User side key $\mathcal{K}_{u_i}$ is then securely sent to the user $i$, and server side key $\mathcal{K}_{s_i}$ is sent to the server.
3. User Encrypt$\left(\mathcal{K}_{u_i}, D, \boldsymbol{x}_D\right)$: The user $i$ uses his user side key $\mathcal{K}_{u_i}$ to encrypt a document $D$ with a set of associated attribute vector $\boldsymbol{x}_D$. The output is user-side ciphertext $c_i^*(D, \boldsymbol{x}_D)$.
4. Server Re − encrypt $\left(i, \mathcal{K}_{s_i}, c_i^*(D, \boldsymbol{x}_D)\right)$: On receiving the ciphertext

$c_i^*(D, \boldsymbol{x}_D)$ from user $i$, the server fetches the server side key $\mathcal{K}_{s_i}$, and outputs re-encrypted ciphertext $c(D, \boldsymbol{x}_D)$.

5. $\mathrm{Trapdoor}\left(\mathcal{K}_{u_i}, \boldsymbol{y}\right)$: On input the attribute y, the user $i$ uses his user side key $\mathcal{K}_{u_i}$ to generate a trapdoor $T_i(\boldsymbol{y})$.
6. $\mathrm{Search}\left(T_i(\boldsymbol{y}), E(D), \mathcal{K}_{s_i}\right)$: The server takes as input the trapdoor $T_i(\boldsymbol{y})$ and user's server side key $\mathcal{K}_{s_i}$, then test for each $c(D, \boldsymbol{x}_D) \in E(D)$ if predicate $P_{\boldsymbol{x}_D}(\boldsymbol{y}) = 1$. If 'yes', the server invokes pre-decrypt algorithm to obtain $c_i'(D)$ and send $c_i'(D)$ to the user $i$.
7. $\mathrm{Server\ Pre - decrypt}\left(i, \mathcal{K}_{s_i}, \mathrm{c}(D)\right)$: The server takes the encrypted data that contains queried keyword from the trapdoor and user's identity $i$ as input, pre-decrypt the encrypted data with its server side key $\mathcal{K}_{s_i}$ as $c_i'(D)$. Send $c_i'(D)$ to user $i$.
8. $\mathrm{Dec}\left(\mathcal{K}_{u_i}, c_i'(D)\right)$: The user takes his user key $\mathcal{K}_{u_i}$, and decrypts $c_i'(D)$ to obtain data $D$.
9. $\mathrm{Revoke}(i)$: Given $i$, the data server updates the user-key mapping set $\mathcal{K}s = \mathcal{K}s \setminus (i, \mathcal{K}_{s_i})$.

The following is the concrete construction of our searchable encryption. Note that both the data encryption and attribute vector (keyword related) encryption are based on pairing-based cryptography.

$\mathbf{Init}(\mathbf{1}^{\boldsymbol{k}}, \boldsymbol{n})$

The KMS first takes the input security parameter $k$ and the attribute length $n = \mathrm{poly}(k)$. The KMS chooses an instance

$I = \{q, \mathbb{G}, \mathbb{G}_T, g, e\}$ and $\mathcal{K} \overset{R}{\leftarrow} \mathbb{Z}_q$, where $q$ is the group order of $\mathbb{G}$ and $\mathbb{G}_T$, $e$ is a symmetric bilinear pairing $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_\mathrm{T}$ and $g$ is a generator of $\mathbb{G}$. Set $Y = e(g,g)^{\mathcal{K}}$. Then the KMS chooses random numbers $t_i, v_i, r_i, m_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and computes

$$T_i = g^{t_i},\ V_i = g^{v_i},\ R_i = g^{r_i},\ M_i = g^{m_i}$$

and

$$T_i' = g^{-t_i},\ V_i' = g^{-v_i},\ R_i' = g^{-r_i},\ M_i' = g^{-m_i}$$

for $i = 1, \cdots, n$. The public parameters is published by the KMS as

$$Params = [I, Y, (T_i, V_i, R_i, M_i)_{i=1}^n, (T_i', V_i', R_i', M_i')_{i=1}^n],$$

and the the master key is kept secret as

$$MSK = \left[\mathcal{K},\ (t_i, v_i, r_i, m_i)_{i=1}^n\right].$$

### Keygen$(\boldsymbol{MSK}, \boldsymbol{j})$

On input the $MSK$, for each user $i$, the KMS randomly chooses $\mathcal{K}_{u_j} \overset{R}{\leftarrow} \mathbb{Z}_q$, and compute $\mathcal{K}_{s_j} = \mathcal{K}/\mathcal{K}_{u_j}$. Then the KMS securely transmits $\mathcal{K}_{u_j}$ to the user $j$ and $(j, \mathcal{K}_{s_j})$ to the data server. The server side key mapping set $\mathcal{K}s$ is updated as $\mathcal{K}s = \mathcal{K}s \cup (j, \mathcal{K}_{s_j})$.

### User Encrypt$(\boldsymbol{\mathcal{K}_{u_i}}, \boldsymbol{D}, \boldsymbol{x_D})$

The user takes as input the data $D \in \mathbb{Z}_p$ where $\mathbb{Z}_p$ is the base filed of $\mathbb{G}$, the user side key $\mathcal{K}_{u_j}$ and attribute vector $\boldsymbol{x}_D \in \{0,1\}^n$. The user chooses random number $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and computes $S = g^r$. Let $D = d_1 \parallel d_2$ and $S = (x_s, y_s)$. Then he computes $C = [c_1, c_2]$ where $c_1 = x_s \cdot d_1$, $c_2 = y_s \cdot d_2$ and $Q_u = S^{\mathcal{K}_{u_j}}$. Next, the user chooses $s \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $s_i \overset{R}{\leftarrow} \mathbb{Z}_q^*$ for $i = 1, \cdots, n$, and computes $\Omega_u = Y^{k_{uj}(-s)}$, $C_0 = g^s$, and

$$X_i = \begin{cases} {T_i}^{s-s_i}, & if \ \boldsymbol{x}_{Di} = 1; \\ {R_i}^{s-s_i}, & if \ \boldsymbol{x}_{Di} = 0; \end{cases} \qquad W_i = \begin{cases} {V_i}^{s}, & if \ \boldsymbol{x}_{Di} = 1; \\ {M_i}^{s}, & if \ \boldsymbol{x}_{Di} = 0; \end{cases}$$

for $i = 1, \cdots, n$. Finally, $c_j^*(D, x_D) = \left[c_j^*(D), c_j^*(x_D)\right]$ is sent to the data server where $c_j^*(D) = [C, Q_u]$ as ciphertext and $c_j^*(x_D) = \left[\Omega_u,\ C_0,\ (X_i, W_i)_{i=1}^n\right]$ as searchable encryption.

### $\textbf{Server Re} - \textbf{encrypt}\left(\boldsymbol{j}, \boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}}, \boldsymbol{c_j^*}(\boldsymbol{D}, \boldsymbol{x_D})\right)$

The proxy server finds the server side key of user $j$, $\mathcal{K}_{s_j} = (j, x_{j2})$. It then re-encrypts the ciphertext $c_i^*(D, x_D) = \left[c_j^*(D), c_j^*(x_D)\right]$ by computing $Q = {Q_u}^{\mathcal{K}_{s_j}} = S^{\mathcal{K}}$ and $\Omega = {\Omega_u}^{\mathcal{K}_{s_j}} = Y^{-\mathcal{K}S}$. Finally, $(D, x_D) = [c(D), c(x_D)]$, where $c(D) = [C, Q]$ and $c(x_D) = \left[\Omega,\ C_0,\ (X_i, W_i)_{i=1}^n\right]$ is inserted into the data storage $E(D) = E(D) \cup c(D, x_D)$.

### $\textbf{Trapdoor}(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_h}}, \boldsymbol{y})$

The user $h$ takes as input his user side key $\mathcal{K}_{u_h}$ and string $\boldsymbol{y} \in \{0,1,*\}^n$. Denote $S_{\boldsymbol{y}}^1$ and $S_{\boldsymbol{y}}^0$ to be the set of indices $i$ such that $S_{\boldsymbol{y}}^1 = \{i \mid \boldsymbol{y_i} = 1\}$ and $S_{\boldsymbol{y}}^0 = \{i \mid \boldsymbol{y_i} = 0\}$. Let $S_{\boldsymbol{y}} = S_{\boldsymbol{y}}^1 \cup S_{\boldsymbol{y}}^0$ be the set of indices $i$ for which $\boldsymbol{y_i} \neq *$. If $S_{\boldsymbol{y}} = \emptyset$, that is, $\boldsymbol{y} = (*, \ldots, *)$, let $T_u = g^{\mathcal{K}_{u_h}}$. Else, for each $i \in S_{\boldsymbol{y}}$, choose a number $\alpha_i \overset{R}{\leftarrow} \mathbb{Z}_q^*$ such that $\sum_{i \in S_y} \alpha_i = \mathcal{K}_{u_h}$. Compute $T_u = (Y_i, L_i)_{i=1}^{n}$ where

$$Y_i = \begin{cases} {T'_i}^{\alpha_i}\ , & \text{if } \boldsymbol{y}_i = 1; \\ {R'_i}^{\alpha_i}\ , & \text{if } \boldsymbol{y}_i = 0; \\ \emptyset\,, & \text{if } \boldsymbol{y}_i = \ * \ ; \end{cases} \qquad L_i = \begin{cases} {V'_i}^{\alpha_i}, & \text{if } \boldsymbol{y}_i = 1; \\ {M'_i}^{\alpha_i}, & \text{if } \boldsymbol{y}_i = 0; \\ \emptyset\ \ , & \text{if } \boldsymbol{y}_i = \ * \ ; \end{cases}$$

Then, the user sends the trapdoor $T_u$ relative to attribute vector $\boldsymbol{y}$ to the data server.

### Search$(\boldsymbol{T_u}, \boldsymbol{\mathcal{K}_{s_h}})$

Take as input $\mathcal{K}_{s_h}$ the server side key of user $h$, and the trapdoor $T_u$, the data server perform search by calculating whether $Test = 1$ for each $c(D, x_D) \in E(D)$. If $S_{\boldsymbol{y}} = \emptyset$, then $T_u = g^{\mathcal{K}_{u_h}}$, the data server calculates $Test$ as

$$\begin{aligned} Test &= \Omega \cdot e\left(C_0, \mathcal{K}_{s_h} T_u\right) \\ &= 1 \end{aligned}$$

Else, the data server calculates $Test$ as

$$Test \ = \Omega \cdot \left[ \prod_{i \in S_y} e(X_i, Y_i)\, e(W_i, L_i) \right]^{x_{sh}}$$

If predicate $P_{\boldsymbol{x}}(\boldsymbol{y}) = 1$, then $Test \ = 1$ since

$$\begin{aligned}
Test\ &= \Omega \cdot \left[\prod_{i\in S_y} e(X_i,Y_i)\, e(W_i,L_i)\right]^{x_{sh}} \\
&= \Omega \cdot \left[e(g,g)^{\sum_{i\in S_y}\alpha_i(s-s_i)+\sum_{i\in S_y}\alpha_i s_i}\right]^{x_{sh}} \\
&= \Omega \cdot \left[e(g,g)^{\sum_{i\in S_y}\alpha_i s}\ \right]^{x_{sh}} \\
&= \Omega \cdot [e(g,g)^{x_{uh}s}\ ]^{x_{sh}} \\
&= e(g,g)^{-xs} \cdot e(g,g)^{xs} \\
&= 1
\end{aligned}$$

**Server Pre − decrypt** $\left(\boldsymbol{j},\boldsymbol{\mathcal{K}}_{\boldsymbol{s_j}},\mathbf{c}(\boldsymbol{D})\right)$

On inputs user id $j$ and encrypted data $\mathrm{c}(D) = [C,Q]$, the data server pre-decrypt $\mathrm{c}(D)$ to $c_j'(D)$ in order for user $j$ to decrypt the encrypted data. The data server computes $Q_u = Q^{-\mathcal{K}_{s_j}}$. $c_j'(D) = [C,Q_u]$ is then sent to the user $j$.

**Dec** $\left(\boldsymbol{\mathcal{K}}_{\boldsymbol{u_j}},\boldsymbol{c_j'}(\boldsymbol{D})\right)$

User $j$ fully decrypts the pre-decrypted ciphertext $c_j'(D) = [C,Q_u]$ where $C = [c_1,c_2]$. He computes $S = {Q_u}^{\mathcal{K}_{u_j}}$ and $D = d_1 \parallel d_2$ where $d_1 = c_1 \cdot {x_s}^{-1}$, $d_2 = c_2 \cdot {y_s}^{-1}$ to obtain the plaintext data $D$.

**Revoke**$(\boldsymbol{i})$

To revoke user $i$, the data server simply updates the user-key mapping set $\mathcal{K}s = \mathcal{K}s \setminus (i,\mathcal{K}_{s_i})$.

Thus we complete the construction of our public key searchable encryption with conjunctive queries. We will further discuss the experimental performance of each function in section 4.3.

## 4.2 Conjunctive Queries

In this section we show how conjunctive queries can be applied on our scheme. Let $I = (m_1, \cdots, m_w)$ be a keyword set to be encrypted for future search. Let $x \in \{0,1\}^n$ and $y \in \{0,1,*\}^n$ be attribute vectors that are related to the data and the trapdoor respectively. Let $\boldsymbol{X} = (x_1, \cdots, x_w)$ and $\boldsymbol{Y} = (y_1, \cdots, y_w)$ be a vector of consecutive attribute vector $x$ or $y$, and $n$ be the length of attribute vector. Define a predicate $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if $x_{i,j} = y_{i,j}$ or $y_{i,j} = *$ , for $i = 1, \cdots, w$ and $j = 1, \cdots, n$; $P_{x_i}(y_i) = 0$ otherwise. Note that in the hidden vector encryptions we described in Chapters 2 and 3, for simplicity we take only one attribute vector $x \in \{0,1\}^n$ or $y \in \{0,1,*\}^n$ as an input. In fact, the actual input is the hidden vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ consisting of $w$ attribute vectors. In the following we will describe the design the attribute vectors in order to perform conjunctive comparison queries, conjunctive range queries, and conjunctive subset queries.

**Conjunctive Comparison Queries**

Suppose there are $w$ conjunctive queries, then the width of the hidden vector encryption is $\ell = nw$. Let $I = (m_1, \cdots, m_w) \in (1, \cdots, n)^w$, that is, $m_i$ is a number ranging from 1 to n. Build an attribute vector $\boldsymbol{X}$ as:

$$x_{i,j} = \begin{cases} 1, & \text{if } j \geq m_i, \\ 0, & \text{otherwise.} \end{cases}$$

For example, let $w = 2$, then $\boldsymbol{X} \in \{0,1\}^{2n}$ such that

| | 1 | | | $m_1$ | | | n | 1 | | | $m_2$ | | | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{X}$ | 0 | … | 0 | 1 | 1 | … | 1 | 0 | … | 0 | 1 | 1 | … | 1 |

To test whether if $a_i \geq m_i$ for any query keyword $a_i$ in $A = (a_1, \cdots, a_w) \in (1, \cdots, n)^w$, we build an attribute vector $\boldsymbol{Y}$ as:

$$y_{i,j} = \begin{cases} 1, & \text{if } j = a_i, \\ *, & \mathit{other}\text{wise.} \end{cases}$$

For example, assign $w = 2$, then $\boldsymbol{Y} \in \{0,1,*\}^{2n}$ looks like

| | 1 | | | $a_1$ | | | n | 1 | | | $a_2$ | | | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{Y}$ | * | … | * | 1 | * | … | * | * | … | * | 1 | * | … | * |

Attribute vector $\boldsymbol{X}$ is then hidden in the searchable encryption that is generated in User Encrypt step, and attribute vectors $\boldsymbol{Y}$ is then hidden in the trapdoor generated by user in Trapdoor step. In Search step, the predicate $P_{\boldsymbol{X}}(\boldsymbol{Y})$ is tested to see if a ciphertext contains keywords that match/satisfy the trapdoor. The predicate $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if $a_i \geq m_i$ for $i = 1, \cdots, w$. If $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$, then the data with keyword set $I = (m_1, \cdots, m_w)$ is considered to be containing keywords such that $m_1 \leq a_1 \land \cdots \land m_w \leq a_w$.

**Conjunctive Range Queries**

A system that supports conjunctive comparison queries also supports conjunctive range queries. Let $I$ be a set of $w$ keywords $I =$

$(m_1, \cdots, m_w) \in (1, \cdots, n)^w$. A range query searches for plaintexts where keyword $m \in [a, b]$. For example, let $w = 1$. To do conjuctive range queries, build the attribute vector $\boldsymbol{X} \in \{0,1\}^n$ as:

| | 1 | | | $m$ | | | n |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{X}$ | 0 | … | 0 | 1 | 1 | … | 1 |

Let the attribute vector $\boldsymbol{Y} \in \{0,1,*\}^n$ be:

| | 1 | | | $a$ | | | | $b$ | | | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{Y}$ | 0 | … | 0 | 0 | $*$ | … | $*$ | 1 | 1 | … | 1 |

The predicate $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if $m \geq a$ and $m \leq b$. To do conjunctive range queries, attach more attribute vectors to $\boldsymbol{X}$ and $\boldsymbol{Y}$ for different $m_i$'s.

**Conjunctive Subset Queries**

Here we show how to design attribute vectors so the subset queries is searchable. Let $= (m_1, \cdots, m_w) \in T$ , where $T$ is a size-n set of all possible $m$. Let an attribute vector $\boldsymbol{X}$ be:

$$x_{i,j} = \begin{cases} 1, & \text{if } j = m_i, \\ 0, & \text{otherwise.} \end{cases}$$

To test whether if $m_i \in A_i$ for any query set $A_i \in T$ in $A = (A_1, \cdots, A_w)$ for $i = 1, \cdots, w$, build an attribute vector $\boldsymbol{Y}$ as:

$$y_{i,j} = \begin{cases} 0, & \text{if } j \notin A_i, \\ * , & other\text{wise.} \end{cases}$$

The predicate $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if $m_i \in A_i$ for all $i = 1, \cdots, w$. That is, $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$, $m_i$ in $I = (m_1, \cdots, m_w)$ satisfies that $m_1 \in A_1 \wedge \cdots \wedge m_w \in A_w$. For example, let $w = 1$, build the attribute vector $\boldsymbol{X} \in \{0,1\}^n$ as:

|  | 1 |  |  | $m$ |  |  | n |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{X}$ | 0 | … | 0 | 1 | 0 | … | 0 |

Build the attribute vector $\boldsymbol{Y} \in \{0,*\}^n$ according to set $A = \{2,3,n\}$ as:

|  | 1 | 2 | 3 | 4 | 5 |  |  | n |
|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{Y}$ | 0 | $*$ | $*$ | 0 | 0 | … | 0 | $*$ |

Note that $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if $m \in A$. Arbitrary number of conjunctive subset queries are also allowed by setting larger $w$.

**Subset queries using Bloom filters**

We notice that in the subset queries, the space needed increases significantly as $n$, the size of $T$ of all possible keywords, increases. The hidden attribute vector $\boldsymbol{X}$ is of size $nw$**,** with the same size for $\boldsymbol{Y}$**.** We give a design using the Bloom filters to reduce the space requirement as the size of $T$ is large.

Bloom filters[4] utilizes multiple functions $H_1, \cdots H_d: \{0,1\}^* \to T$. A bloom filter $\boldsymbol{B}$ is a vector of size $n$, such that $\boldsymbol{B} \in \{0,1\}^n$. For a keyword $m$ of arbitrary length, the bloom filter of this word is $\boldsymbol{B} \in \{0,1\}^n$ that contains '1' at positions $H_1(m), \cdots, H_d(m)$. With $I = (m_1, \cdots, m_w)$, we have bloom filter $\boldsymbol{B} \in \{0,1\}^n$ that contains '1' at positions $H_k(m_i)$, for $k = 1, \cdots, d$, $i = 1, \cdots, w$ We design the attribute vectors $\boldsymbol{X}$ as:

$$x_{i,j} = \begin{cases} 1, & \text{if } j = H_k(m_i), \\ 0, & \text{otherwise.} \end{cases}$$

In another word, the attribute vector $\boldsymbol{X}$ is set to be the bloom filter $\boldsymbol{B}$ of keyword set $I = (m_1, \cdots, m_w)$**.** Then for a set $A = (m'_1, \cdots, m'_s)$, build an attribute vector $\boldsymbol{Y}$ as:

$$y_{i,j} = \begin{cases} 1, & \text{if } j = H_k(m'_i), \\ * & , \quad other\text{wise.} \end{cases}$$

That is, the attribute vector $\boldsymbol{Y}$ is set to be the bloom filter $\boldsymbol{B'}$ of keyword set $A$. The predicate $P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1$ if and only if set $A \in I$. The predicate $P_{\boldsymbol{X}}(\boldsymbol{Y})$ indicates whether all words $m'$ in set $A$ are contained in set . If yes, then the bloom filter $\boldsymbol{B}$ is marked '1' at the corresponding position, so does the bloom filter $\boldsymbol{B'}$**.** If no, then $\boldsymbol{B'}$ "could" contains '1's not in $\boldsymbol{B}$ with very high probability (small collision probability). By choosing $d$, number of functions $H_1, \cdots H_d$, and $n$, the size of a bloom filter $\boldsymbol{B}$, the false positive probability can be very small. Say,

$$\Pr[P_{\boldsymbol{X}}(\boldsymbol{Y}) = 1 \, with \; A \notin \; I] < negl(k)$$

## 4.3 Experiments

In this section we describe the implementation of our public key searchable encryption. First we describe the pairing library used in our program in section 4.3.1. Then we have performance evaluation in section 4.3.2.

### 4.3.1 The Pairing-based Cryptography Library

The pairing-based cryptography (PBC) library [23] is an open source library that is released under the GNU Lesser General Public License. The PBC library is written in C and provides routines such as elliptic curve generation, elliptic curve arithmetic and pairing computation.

We have tested the speed of the PBC library. We performed our experiments on a 2.4 GHz Intel Xeon E5620 processor running Ubuntu 11.10. The security level we choose is 128-bit. Table 4.1 is the key size comparison under different security levels [29].

| Date | Minimun of Strength | Symmetric Key | RSA and DH | Elliptic Curve |
|---|---|---|---|---|
| 2010 | 80 | 80 | 1024 | 160 |
| 2011-2030 | 112 | 112 | 2048 | 224 |
| > 2030 | 128 | 128 | 3072 | 256 |

Table 4.1: NIST Recommended Key Sizes(bits)

There are seven types of pairings defined in the PBC library. The seven types are type A, type B, type C, type D, type E, type F and type G. Type A, type B and Type C are supersingular curves. Type D, type E, type F and type G are based on complex multiplication (CM) method[28].

However, type B and type C are not implemented yet.

The CM equation is

$$\mathrm{DV}^2 = 4\mathrm{q} - \mathrm{t}^2,$$

where the discriminant D is positive. We omit the details of the CM method here.

Type A pairings are constructed on the curve $\mathrm{E}: \mathrm{y}^2 = \mathrm{x}^3 + \mathrm{x}$ over $\mathrm{F}_\mathrm{q}$, where $\mathrm{q}$ is a prime and $\mathrm{q} \equiv 3 \pmod 4$. E is a supersingular curve, so this pairing is a symmetric pairing $\mathrm{e}: \mathrm{G}_1 \times \mathrm{G}_1 \rightarrow \mathrm{G}_\mathrm{T}$. $\mathrm{G}_\mathrm{T}$ is a subgroup of $\mathrm{F}_{\mathrm{q}^2}$ because the embedding degree is 2. Therefore we choose the group order $\mathrm{r}$ to be 256-bit long and $\mathrm{q}$ to be 1536-bit long, because $q^2$ must be 3072-bit long to achieve the same security level as 256-bit long in elliptic curve.

Type D pairings are constructed on the MNT curves of embedding degree 6 [25]. This pairing is an asymmetric pairing $\mathrm{e}: \mathrm{G}_1 \times \mathrm{G}_2 \rightarrow \mathrm{G}_\mathrm{T}$. $\mathrm{G}_\mathrm{T}$ is a subgroup of $\mathrm{F}_{\mathrm{q}^6}$ because the embedding degree is 6. Given different discriminant in the CM equation, the bits in $\mathrm{q}$ and the bits in $\mathrm{r}$ are determined. Therefore we choose two suitable type D pairings. One is that the discriminant is 31387, $\mathrm{q}$ is 522-bit long and $\mathrm{r}$ is 514-bit long. The other is that discriminant is 873867, $\mathrm{q}$ is 486-bit long and $\mathrm{r}$ is 442-bit long

Type E pairings are constructed on the curves of embedding 1 [21]. The pairing is a symmetric pairing $\mathrm{e}: \mathrm{G}_1 \times \mathrm{G}_1 \rightarrow \mathrm{G}_\mathrm{T}$. $\mathrm{G}_\mathrm{T}$ is a subgroup of $\mathrm{F}_\mathrm{q}$ because the embedding degree is 1. Therefore we choose the group order $\mathrm{r}$ to be 256-bit long and $\mathrm{q}$ to be 3072-bit long, because $q$ must be 3072-bit long to achieve the same security level as 256-bit long in elliptic curve.

Type F pairings are constructed on the curves of embedding 12. This pairing is an asymmetric pairing $\mathrm{e}: \mathrm{G}_1 \times \mathrm{G}_2 \rightarrow \mathrm{G}_\mathrm{T}$. $\mathrm{G}_\mathrm{T}$ is a subgroup of

$F_{q^{12}}$ because the embedding degree is 12. Therefore we choose the group order $r$ to be 256-bit long and $q$ to be 256-bit long.

Type G pairings are constructed on the curves of embedding 10 which Freeman suggests [11]. Given different discriminant in the CM equation, the bits in $q$ and the bits in $r$ are determined. Therefore we choose one suitable type G pairings. The curve is that the discriminant is 35707, $q$ is 301-bit long and $r$ is 279-bit long. Table 4.2 is a comparison of the pairings in the PBC library.

| | Embedding Degree | Symmetric Pairing | Supersingular |
|---|---|---|---|
| Type A | 2 | yes | yes |
| Type D | 6 | no | no |
| Type E | 1 | yes | no |
| Type F | 12 | no | no |
| Type G | 10 | no | no |

Table 4.2: Pairings in the PBC library

| | Pairing Time (ms) | Multiplication Time in $G_T$ (ms) | Addition Time in $G_1$ (ms) | Addition Time in $G_2$ (ms) |
|---|---|---|---|---|
| Type A | 38 | 0.009 | 0.042 | 0.042 |
| Type D-311387 | 48 | 0.023 | 0.011 | 0.078 |
| Type D-873867 | 35 | 0.020 | 0.010 | 0.068 |
| Type E | 87 | 0.009 | 0.108 | 0.108 |
| Type F | 49 | 0.037 | 0.006 | 0.009 |
| Type G-35707 | 45 | 0.036 | 0.006 | 0.090 |

Table 4.3: Comparison of Speed of Different Pairings

For each type, we choose 10 random inputs to the pairing function and compute the average time. We also choose 100 random elements for $G_1$, $G_2$ and $G_3$ for each type and compute the average time of an addition or an multiplication. The result of our test is shown in Table 4.3. We note that in our encryption scheme, we need a symmetric pairing. And the Type E pairing is the slowest pairing. Therefore, in our implementation, we choose the Type A pairing.

### 4.3.2 Experimental Result

We implemented our algorithms on a 2.4 GHz Intel Xeon E5620 processor running Ubuntu 11.10. We used 1536-bit prime $q$ for pairing. In the first experiment, we measured the execution time of each of the following operations:

1. Initialization – KMS outputs public key and a master key set.
2. Key Generation – KMS generates user side key and server side key.
3. User Encryption – the user side proxy and searchable encryption.
4. Server Re-encryption – the server side proxy re-encryption
5. Trapdoor – the user side trapdoor generation.
6. Search – the trapdoor/searchable encryption matching test.
7. Server Pre-decryption – the server side proxy decryption.
8. User Decryption – the user side proxy decryption.
9. Revocation – the server side revocation of the user.

Figure 4.1 shows the results. Our test data are 2011 eprint pdf files. Note that the pdf data were encrypted with symmetric key encryption AES128. Then we took the session key of AES128 encryption as our plaintext. The user who successfully decrypts the ciphertext will retrieve the session key of the encrypted pdf file. We did not calculate the AES128 encrypting time, so the size of the pdf files was irrelevant. The time was measured in milliseconds, and it is the average of 10000 executions. We set the size hidden vectors $N$=10. We can now see that the Initialization took up most of the time. The main cause is that it needs $4N$ times pairing element powers. So are the user encryption and search algorithms, which both need $2N$ element powers. The Search and Trapdoor algorithm are significantly influenced by the number of $*$ (don’t care) appears in the attribute vector $\boldsymbol{Y}$. The more $*$ , the less computation is needed, which happens in most of the application where don’t care term is much more than ‘0’s and ‘1’s.

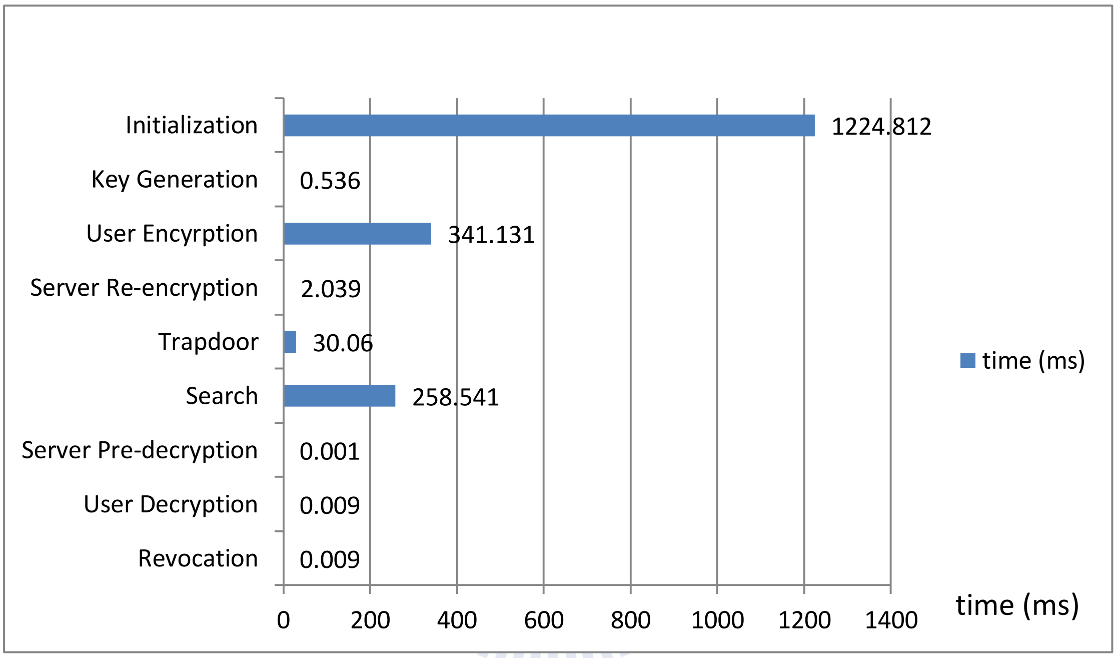

Figure 4.1 Performance of Individual Operations

It is obvious that all algorithms that spend longer time are searchable encryption related. So the second and third experiments came as follows: we measured the algorithms by two parts: the proxy encryption part and hidden vector encryption part.

Figure 4.2 showed the result of 10000 executions of proxy encryption part algorithms. We showed that under $N$=10, $N$=100, $N$=1000, we had similar execution time for the proxy encryption part. Hence, the number of keywords does not affect the encryption of the data but only affect the searchable encryption in our algorithm. As to the result of 10000 executions of HVE encryption part, under $N$=10, $N$=100, $N$=1000, we had the execution time of Initialization, User Encryption, Trapdoor and Search algorithms in direct proportional with the size of hidden attribute vector $N$. Hence, it is crucial to optimize the size of hidden vector since it causes significant increases in computing time.

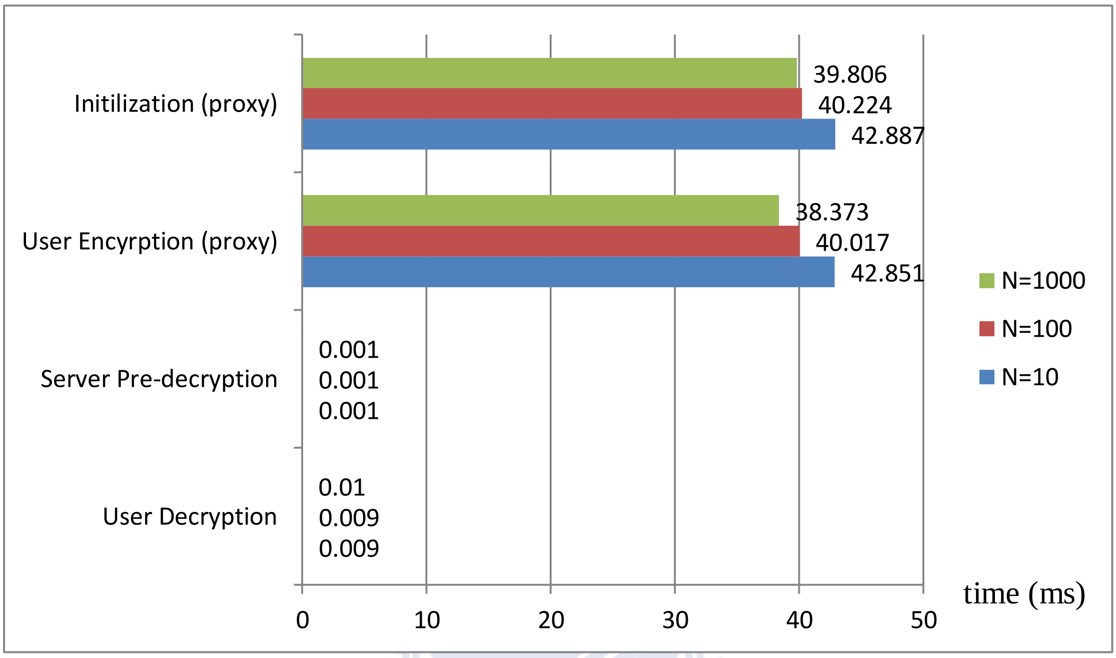

Figure 1.2 Performance of Proxy Part

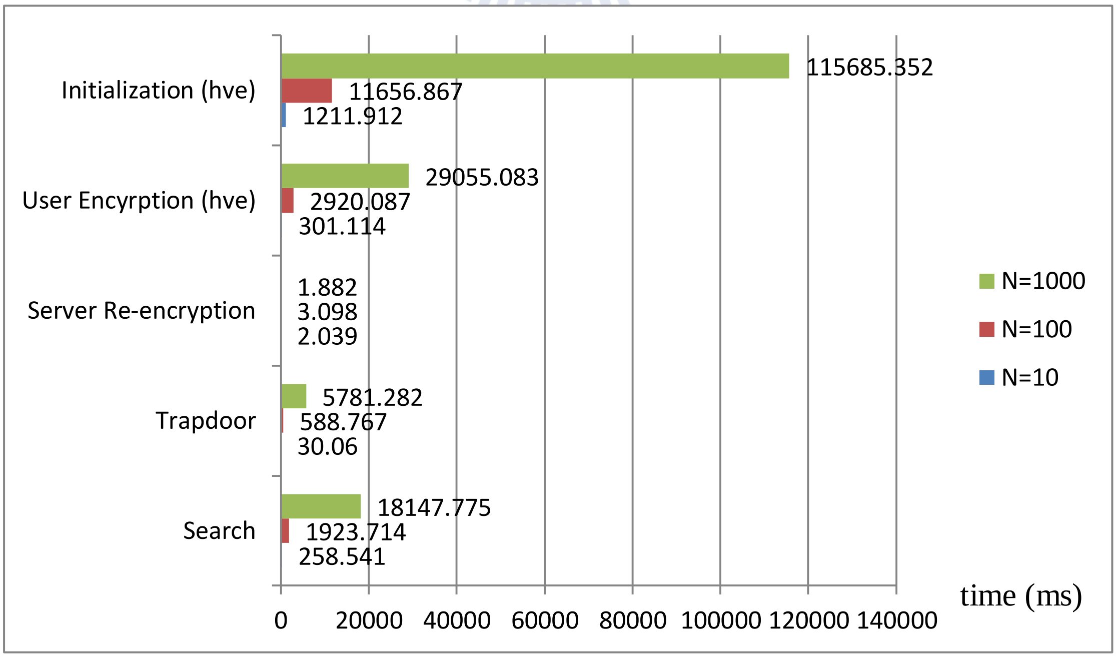

Figure 4.3 Performance of HVE Part

# 5 Conclusion

## 5.1 Summary

We introduced the idea of searchable encryption that is used to solve the problem of how to efficiently search on encrypted data. In Chapter 2, we introduced the mathematical background including elliptic curve and bilinear pairings. In Chapter 3, we reviewed three prominent public key searchable encryptions: public key encryption with keyword search, multi-user searchable data encryption, and hidden-vector encryption. We described the scheme, and then gave out its definition as well as its concrete construction. In Chapter 4, we described our design of searchable encryption, providing a solution to sharing data on untrusted server with conjunctive keyword search. After describing our construction in detail, we introduced several applications of our scheme, including conjunctive comparison queries, range queries, and subset queries. We mentioned an interesting application that can reduce the space needed by conjunctive subset queries by apply Bloom filters on the hidden vectors. Then we described our implementation and evaluated the performance of our algorithms.

## 5.2 Future Work

For further research, we recommend for the following topics:

1. **Multi-user searchable data encryption without key management center:** In our design and DGD scheme, we need a key management

center to hold the master key. Generating user side and server side keys of all users with a single master key implies the risk of collusion attack. Also, renewing master keys requires the user to encrypt his previous encrypted data and searchable encryption again. We expect there is a multi-user searchable encryption scheme that runs without key management center.

2. **Improve the performance of HVE encryption.** As we can see in the performance evaluation in Chapter 4.3, most computation are cost by pairing computation. By redesigning the algorithms, we expect the precompile pairing comes in handy while, if possible, consecutive pairing computes with the same first argument. Precompile pairing improves performance significantly on a type A pairing.
3. **Applications of HVE.** By designing the hidden vector ***X*** and ***Y*** properly, the hidden vector encryption provides can do many operations while the vectors are hidden. We look for more applications of HVE.

# Bibliography

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, “Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions,” *Journal of Cryptology*, vol. 21, no. 3, pp. 350–391, 2008.

[2] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage,” in *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, 2005, pp. 29–44.

[3] I. F. Blake, G. Seroussi, and N. P. Smart, *Advances in elliptic curve cryptography*. Cambridge Univ Pr, 2005.

[4] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in *Advances in Cryptology-Eurocrypt 2004*, 2004, pp. 506–522.

[6] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” *Theory of Cryptography*, pp. 535–554, 2007.

[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 79–88.

[8] A. De Caro, V. Iovino, and G. Persiano, “Fully secure anonymous hibe and secret-key anonymous ibe with short ciphertexts,” *Pairing-Based Cryptography-Pairing 2010*, pp. 347–366, 2010.

[9] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, no. 3, pp. 367–397, 2011.

[10] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, 1985, pp. 10–18.

[11] D. Freeman, "Constructing pairing-friendly elliptic curves with embedding degree 10," *Algorithmic Number Theory*, pp. 452–465, 2006.

[12] D. Freeman, "Converting pairing-based cryptosystems from composite-order groups to prime-order groups," *Advances in Cryptology–EUROCRYPT 2010*, pp. 44–61, 2010.

[13] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in Applied Cryptography and Network Security, 2007, pp. 288–306.

[14] E. J. Goh, "Secure indexes," *Technical Report 2003/216, IACR ePrint Cryptography Archive* (2003), http://eprint.iacr.org/2003/216

[15] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security*, 2004, pp. 31–45.

[16] J. Hoffstein, J. C. Pipher, and J. H. Silverman, *An introduction to mathematical cryptography*. Springer Verlag, 2008.

[17] V. Iovino and G. Persiano, "Hidden-vector encryption with groups of prime order," *Pairing-Based Cryptography–Pairing 2008*, pp. 75–88, 2008.

[18] A. Ivan and Y. Dodis, "Proxy cryptography revisited," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2003.

[19] M. Jakobsson, "On quorum controlled asymmetric proxy re-encryption," in *Public Key Cryptography*, 1999, pp. 632–632.

[20] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proceedings of the Theory and*

*Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, 2008, pp. 146–162.

[21]N. Koblitz and A. Menezes, “Pairing-based cryptography at high security levels,” *Cryptography and Coding*, pp. 13–36, 2005.

[22] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, “Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption,” *Advances in Cryptology–EUROCRYPT 2010*, pp. 62–91, 2010.

[23] B. Lynn, “PBC library–the pairing-based cryptography library,” http://crypto.stanford.edu/pbc/

[24] V. Miller, “Short programs for functions on curves,” *Unpublished manuscript*, vol. 97, pp. 101–102, 1986.

[25] A. Miyaji, M. Nakabayashi, and S. Takano, “New explicit conditions of elliptic curve traces for FR-reduction,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2001.

[26] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” *Advances in Cryptology–EUROCRYPT 2005*, pp. 557–557, 2005.

[27] S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, “Towards an information theoretic analysis of searchable encryption,” *Information and Communications Security*, pp. 345–360, 2008.

[28] L. C. Washington, *Elliptic curves: number theory and cryptography*, vol. 50. Chapman & Hall, 2008.

[29] “NIST Recommended Key Sizes.” http://www.nsa.gov/business/programs/elliptic_curve.shtml.

# Appendix : Source Code

We call our construction as EPSE, where E stands for Elliptic curve cryptography, P for Proxy Encryption, SE stands for Searchable Encryption. The following is our C code for our construction: A.1 gives our header file, A.2 gives an example test file of using our construction header file, and A.3 is our EPSE.c code.

## A.1 EPSE.h

```
#ifndef SELIB_EPSE_H_
#define SELIB_EPSE_H_
#include <pbc/pbc.h>
#include <openssl/sha.h>
#include "./pairingio.h"
#include <libgen.h>
#include <omp.h>
#include <dirent.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#define EPSEPARAM "EPSE.pairing"
#define N 10
#define NUM_OF_USER 100
typedef struct {
   element_t g;
   element_t Y;
   pairing_t pairing;
   element_t T[N], V[N], R[N], M[N];
   element_t Ti[N], Vi[N], Ri[N], Mi[N];
}PUB_PARAM;
typedef struct {
   element_t k;
   element_t t[N], v[N], r[N], m[N];
```

```
}PRIV_PARAM;
typedef struct {
   int id;
   element_t ku;
}USER;
typedef struct {
   element_t ks[NUM_OF_USER];
}SERVER;
typedef struct {
   char attry[N];
   element_t tr_of_allstar;
   element_t Y[N], L[N];
}TRAPDOOR;
typedef struct {
   char path[80];
   char attrx[N];
}PTEXT;
typedef struct {
   char path[80];
   element_t R;
   mpz_t c1, c2;
   element_t Sigma;
   element_t C0;
   element_t X[N], W[N];
}HVE;
/* The following are the 8 EPSE functions
   return 0 if succeed, -1 if failed*/
int epse_init(PUB_PARAM pub, PRIV_PARAM prv);
int epse_keygen(USER user, SERVER server, PUB_PARAM pub, PRIV_PARAM prv);
int epse_u_enc(USER user, PTEXT *pt, HVE *hve, SERVER server, PUB_PARAM pub);
int epse_u_dec(USER user, PTEXT *decipher, HVE *hve, SERVER server, PUB_PARAM pu
int epse_s_enc(int userid, SERVER server, HVE *hve, PUB_PARAM pub);
int epse_s_dec(int userid, SERVER server, HVE *hve, PUB_PARAM pub);
int epse_u_trapdoor(USER user, TRAPDOOR *tr, PUB_PARAM pub);
int epse_s_search(int userid, TRAPDOOR *tr, SERVER server, PUB_PARAM pub);
```

```
void hve_to_file(HVE *hve, char* path);
int comparebyte(char* c1, char* c2, int len);
// following two functions transcode/detranscode a char string message to a big number of type mpz_t
void transcode(unsigned char* message,mpz_t crypt);
void detranscode(mpz_t crypt,unsigned char * message);
#endif
```

## A.2 EPSEtest.c

```
#include "../selib/EPSE.h"
#include <pbc/pbc.h>
int i;
int main(){

// Initialize public parameters
    PUB_PARAM pub;
    PRIV_PARAM prv;
    // Load in a.param
    pbc_param_t pairing_param;
    //pairing_string_from_file(pairing,PAIRING_PARAM);
    pbc_param_init_a_gen(pairing_param, 256, 1536);
    pairing_init_pbc_param(pub.pairing,pairing_param);
    pairing_param_to_file(pairing_param, EPSEPARAM);

    element_init_G1(pub.g,pub.pairing);;
    element_init_GT(pub.Y,pub.pairing);;

    for(i=0;i<N;i++){
        element_init_G1(pub.T[i],pub.pairing);
        element_init_G1(pub.V[i],pub.pairing);
        element_init_G1(pub.R[i],pub.pairing);
        element_init_G1(pub.M[i],pub.pairing);
        element_init_G1(pub.Ti[i],pub.pairing);
        element_init_G1(pub.Vi[i],pub.pairing);
        element_init_G1(pub.Ri[i],pub.pairing);
        element_init_G1(pub.Mi[i],pub.pairing);
    }

// Initialize private parameters
    element_init_Zr(prv.k,pub.pairing);
    for(i=0;i<N;i++){
        element_init_Zr(prv.t[i],pub.pairing);
        element_init_Zr(prv.v[i],pub.pairing);
        element_init_Zr(prv.r[i],pub.pairing);
        element_init_Zr(prv.m[i],pub.pairing);
    }
// Create server and user1 , user2
    SERVER server;
    USER user01, user02;
    user01.id = 1;
    user02.id = 2;
    element_init_Zr(user01.ku,pub.pairing);
    element_init_Zr(user02.ku,pub.pairing);
    element_init_Zr(server.ks[user01.id],pub.pairing);
    element_init_Zr(server.ks[user02.id],pub.pairing);
    epse_keygen(user01,server,pub,prv);
    epse_keygen(user02,server,pub,prv);
// Create a plaintext
    PTEXT pt;
    strcpy(pt.attrx,"000111111");
// Create a hve
    HVE hve;
    strcpy(hve.path, "test");
    element_init_G1(hve.R,pub.pairing);
    mpz_inits(hve.c1,hve.c2,NULL);
    element_init_GT(hve.Sigma,pub.pairing);
    element_init_G1(hve.C0,pub.pairing);
    for(i=0; i<N; i++){
        element_init_G1(hve.X[i],pub.pairing);
        element_init_G1(hve.W[i],pub.pairing);
    }
    element_t test;
    element_init_GT(test,pub.pairing);
    element_t s, exp;
    element_init_Zr(s,pub.pairing);
    element_init_Zr(exp,pub.pairing);
    char buf[1024];
    DIR *dir;
    struct dirent *ent;
    dir = opendir ("epse2011/");
    if (dir != NULL) {
        /* print all the files and directories within directory */
        while ((ent = readdir (dir)) != NULL) {
            if(ent->d_name[0] != '.'){
                //printf ("%s\n", ent->d_name);
                sprintf(buf,"epse2011/%s",ent->d_name);
                strcpy(pt.path, buf);
                sprintf(buf,"epsetest/%s.se",ent->d_name);
                hve_to_file(&hve,buf);
                epse_s_enc(user01.id,server,&hve,pub);
            }
        }
        closedir (dir);
    } else {
        /* could not open directory */
        perror ("");
        return;
    }
// Create a deciphertext
    PTEXT decipher;
    strcpy(decipher.path,"plaintext.decipher");
// Create a trapdoor
    TRAPDOOR tr;
    strcpy(tr.attry,"***0***1**");
    element_init_G1(tr.tr_of_allstar,pub.pairing);
    for(i=0;i<N;i++){
        element_init_G2(tr.Y[i],pub.pairing);
        element_init_G2(tr.L[i],pub.pairing);
    }
    epse_u_trapdoor(user02,&tr,pub);
    epse_s_search(user02.id,&tr,server,pub);
    epse_u_dec(user02,&decipher,&hve,server,pub);
```

```
// user encrypt
    element_t R,RS;
    mpz_t c1, c2;
    mpz_inits(c1,c2,NULL);
    element_init_G1(R,pairing);
    element_init_G1(RS,pairing);
    ece_u_enc(g,kuj,R,c1,c2,"plaintext.txt","ece",pairing);
    ece_s_enc(RS,R,ksj,pairing);
    ece_s_enc(RS,R,ksh,pairing);
// user decrypt
    ece_u_dec(g,kuh,R,c1,c2,"ece.aes128","plain.decipher",pairing);
    element_clear(g);
    element_clear(k);
    return 1;
}
```

## A.3 EPSE.c

Here we provide only the essential functions of EPSE.c: keygen, user encrypt, server re-encrypt, trapdoor, search, server pre-decryt, and user decrypt.

```
int epse_keygen(USER user, SERVER server, PUB_PARAM pub, PRIV_PARAM
prv){
    element_random(user.ku);
    element_div(server.ks[user.id],prv.k,user.ku);
    return 0;
}
int epse_u_enc(USER user, PTEXT *pt, HVE *hve, SERVER server, PUB_PARAM
pub){
// Proxy Encryption Part
    element_t r,S;
    element_init_Zr(r,pub.pairing);
    element_init_G1(S,pub.pairing);
    element_random(r);
    element_mul(S,r,pub.g);
    element_mul(hve->R,user.ku,S);
    char command[1024] = {0};
    // generate session key
    sprintf(command, "openssl rand 16 > session_key");
    system(command);
    // encrypt file using AES-cbc under session key
    sprintf(command, "openssl aes-128-cbc -e -salt -in %s -out %s.aes128 -pass pas
    system(command);

    FILE *sessionkey = fopen("session_key","r");
    unsigned char aeskey[16], key1[9], key2[9];
    fread(aeskey,1,16,sessionkey);
    fclose(sessionkey);
    for(i=0;i<8;i++) key1[i]=aeskey[i];
    key1[8]='\0';
    for(i=0;i<8;i++) key2[i]=aeskey[i+8];
    key2[8]='\0';

    mpz_t m1, m2;
    mpz_inits(m1,m2,NULL);
    transcodage(key1,m1);
    transcodage(key2,m2);

    mpz_t xs,ys;
    mpz_inits(xs,ys,NULL);
    element_to_mpz(xs,element_x(S));
    element_to_mpz(ys,element_y(S));
    mpz_add(hve->c1,xs,m1);
    mpz_add(hve->c2,ys,m2);
    // encrypt session key under RSA and then remove the session key
    sprintf(command, "rm -f session_key");
    system(command);

// HVE Part
    element_t s,si[N];
    element_init_Zr(s,pub.pairing);
    element_t exp, negs, s_si;
    element_init_Zr(exp,pub.pairing);
    element_init_Zr(negs,pub.pairing);
    element_init_Zr(s_si,pub.pairing);
    // Choose s, si at random
    element_random(s);
    for (i=0; i<N; i++){
        element_init_Zr(si[i],pub.pairing);
        element_random(si[i]);
    }
    // Compute Sigma
    element_neg(negs,s);
    element_mul(exp,user.ku,negs);
    element_pow_zn(hve->Sigma,pub.Y,exp);
    // Compute C0
    element_mul(hve->C0,pub.g,s);
    // Compute Xi, Wi
    for (i=0; i<N; i++){
        element_init_G1(hve->X[i],pub.pairing);
        element_init_G1(hve->W[i],pub.pairing);
        element_sub(s_si,s,si[i]);
        if (pt->attrx[i] == '0'){
            element_pow_zn(hve->X[i],pub.R[i],s_si);
            element_pow_zn(hve->W[i],pub.M[i],si[i]);
        }else if (pt->attrx[i] == '1'){
            element_pow_zn(hve->X[i],pub.T[i],s_si);
            element_pow_zn(hve->W[i],pub.V[i],si[i]);
        }
    }
    return 0;
}
int epse_s_enc(int userid, SERVER server, HVE *hve, PUB_PARAM pub){
    element_mul(hve->R,hve->R,server.ks[userid]);
    element_pow_zn(hve->Sigma,hve->Sigma,server.ks[userid]);
    return 0;
}
int epse_u_trapdoor(USER user, TRAPDOOR *tr, PUB_PARAM pub){
    // Check if all ystr are * (don't care), if yes, write 1 and Ky = g^y to outfi
    int allstar = 1;
    for (i=0; i<N; i++){
```

```
    if(tr->attry[i] != '*'){
        allstar = 0;
        break;
    }
}
// Initialize and assign a according to ystr
element_t a[N], sum, diff;
element_init_Zr(sum,pub.pairing);
element_init_Zr(diff,pub.pairing);
for (i=0; i<N; i++){
    element_init_Zr(a[i],pub.pairing);
    if ( tr->attry[i] != '*' ){
        element_random(a[i]);
        element_add(sum,sum,a[i]);
    }
    else{
        element_set0(a[i]);
    }
}
// Modified the first randomed a[i] s.t. sum(ai)=y
for (i=0; i<N; i++){
    if ( tr->attry[i] != '*' ){
        element_sub(diff,sum,user.ku);
        element_sub(a[i],a[i],diff);
        break;
    }
}
// Compute Key Ky = [Y, L]
for (i=0; i<N; i++){
    if (tr->attry[i] == '*'){
        element_set0(tr->Y[i]);
        element_set0(tr->L[i]);
    }else if (tr->attry[i] == '1'){
        element_pow_zn(tr->Y[i],pub.Ti[i],a[i]);
        element_pow_zn(tr->L[i],pub.Vi[i],a[i]);
    }else if (tr->attry[i] == '0'){
        element_pow_zn(tr->Y[i],pub.Ri[i],a[i]);
        element_pow_zn(tr->L[i],pub.Mi[i],a[i]);
        }
    }
    if (allstar == 1){
        element_pow_zn(tr->tr_of_allstar,pub.g,user.ku);
    }else{
        element_set0(tr->tr_of_allstar);
    }
    return 0;
}
int epse_s_search(int userid, TRAPDOOR *tr, SERVER server, PUB_PARAM pub){
    if(!element_is0(tr->tr_of_allstar)){
        printf(" Searching allstar\n");
    }else{
        element_t result, tmp;
        element_init_GT(result,pub.pairing);
        element_init_GT(tmp,pub.pairing);
        // Createa tmp hve
        HVE hve;
        strcpy(hve.path, "test");
        element_init_G1(hve.R,pub.pairing);
        mpz_inits(hve.c1,hve.c2,NULL);
        element_init_GT(hve.Sigma,pub.pairing);
        element_init_G1(hve.C0,pub.pairing);
        for(i=0; i<N; i++){
                        element_init_G1(hve.X[i],pub.pairing);
                        element_init_G1(hve.W[i],pub.pairing);
        }
        FILE *ftr;
        char buf[1024];
        DIR *dir;
        struct dirent *ent;
        dir = opendir ("epsetest/");
        if (dir != NULL) {
            while ((ent = readdir (dir)) != NULL) {
                if(ent->d_name[0] != '.'){
                    sprintf(buf,"epsetest/%s",ent->d_name);
                    ftr = fopen(buf,"r");
                    elements_string_from_file(ftr,hve.Sigma);
                    elements_string_from_file(ftr,hve.C0);
                    for(i=0;i<N;i++){
                        elements_string_from_file(ftr,hve.X[i]);
                        elements_string_from_file(ftr,hve.W[i]);
                    }
                    fclose(ftr);
                    element_set1(result);
                    for(i=0;i<N;i++){
                        if( tr->attry[i] != '*' ){
                            pairing_apply(tmp,hve.X[i],tr->Y[i],pub.pairing);
                            element_mul(result,result,tmp);
                            pairing_apply(tmp,hve.W[i],tr->L[i],pub.pairing);
                            element_mul(result,result,tmp);
                        }
                    }
                    element_pow_zn(result,result,server.ks[userid]);
                    element_mul(result,result,hve.Sigma);
                }
            }
            closedir (dir);
        } else {
            /* could not open directory */
            perror ("Error, no such directory\n");
            return -1;
    }
    return 0;
}
int epse_s_dec(int userid, SERVER server, HVE *hve, PUB_PARAM pub){
    element_div(hve->R,hve->R,server.ks[userid]);
    return 0;
}
int epse_u_dec(USER user, PTEXT *decipher, HVE *hve, SERVER server,
PUB_PARAM pu
    element_t S;
    element_init_G1(S,pub.pairing);
    element_div(S,hve->R,user.ku);
    mpz_t m1, m2, xs, ys;
    mpz_inits(m1,m2,xs,ys,NULL);
    element_to_mpz(xs,element_x(S));
    element_to_mpz(ys,element_y(S));
    mpz_sub(m1,hve->c1,xs);
    mpz_sub(m2,hve->c2,ys);
    unsigned char buf[80];
    unsigned aeskey[16];
    detranscodage(m1,buf);
    for(i=0;i<8;i++) aeskey[i]=buf[i];
    detranscodage(m2,buf);
    for(i=0;i<8;i++) aeskey[i+8]=buf[i];
// put aeskey to session_key
    FILE *sessionkey = fopen("session_key","w");
    for(i=0;i<16;i++){
        fputc(aeskey[i],sessionkey);
    }
    fclose(sessionkey);
    char command[1024] = {0};
    sprintf(command, "od -x session_key");
    system(command);
    sprintf(command, "openssl aes-128-cbc -d -in %s.aes128 -out %s -pass pass:\"`c
    printf("system: %s\n",command);
    system(command);
    return 0;
}
```