UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

# Comparision queries over encrypted data using multilinear maps

propusă de

Ioniţă Alexandru

**Coordonator ştiinţific:**
Prof. dr. Ferucio Laurenţiu Ţiplea

**Sesiunea:** iulie 2018

# Comparision queries over encrypted data using multilinear maps

Alexandru Ioniţă

June 28, 2018

Avizat,
Îndrumător Lucrare de Licenţă,
Prof. dr. Ferucio Laurenţiu Ţiplea
Data 28.06.2018 Semnătura _____

**DECLARAŢIA privind originalitatea conţinutului lucrării de licenţă**

Subsemnatul IONIŢĂ ALEXANDRU, cu domiciliu în or. IAŞI, ROMÂNIA, născut la data de 11.06.1996, identificat prin CNP 1960611226767, absolvent a Universităţii ,,Alexandru Ioan Cuza" din Iaşi, Facultatea de Informatică, specializarea Informatică, promoţia 2015-2018, declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 şi 5 referitoare la plagiat, că lucrarea de licenţă cu titlul:
***Comparision queries over encrypted data using multilinear maps***, elaborată sub îndrumarea dl. Prof. dr. Ferucio Laurenţiu Ţiplea, pe care urmează să o susţin în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime. De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop. Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată, ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi,
28.06.2018

Ioniţă Alexandru

_____

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul "Comparision queries over encrypted data using multilinear maps", codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru Ioan Cuza" Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi,
28.06.2018

Ioniţă Alexandru

# Contents

# 1 Introduction

## 1.1 Problem description

In our days, people value more and more their internet privacy. We will give a few examples, to illustrate the need of Searchable Encryption schemes.
Let's take an e-mail server. That the database of such e-mail server should be encrypted in order to respect it's users privacy, such that no one except the user will be able to read information from there. However, data leakage may happen when the user searches for e-mails containing certain phrases. That is why we notice the need of a model that prevents such leakage. An ideal scheme will encrypt the queries as well, and answer them without decrypting them.

Another example could be a database which is distributed in a cloud server. Users may store sensitive data in the cloud, and may not trust the cloud server. Which is why a encrypted database will protect the user's privacy.

In computer science, a searchable encryption model refers to a system that can perform encrypted queries over encrypted data. The main goal is not to leak any information about the queries or the data, while performing queries. Thus the answer of a query is a collection of cyphertexts. This operation is often realised returning cyphertexts which match a certain token, generated for that specific query.

The most trivial query is finding elements in the cryptotext equal to some value. The query must be hidden (encrypted) such that the value that need to be found remains unkown for public users.
More complex queries are comparison queries (find $x$, such that $x \leqslant e$), or subset queries (given a set $S$, find all $x$ such that $x \in S$). These queries can be further complicated by adding conjunctions or disjunctions to them (e.g given $e$, $S$, find $x$ such that $x \in S$ and $x \geqslant e$).

## 1.2 Related Work

We will present here some *Searchable encryption* systems, which drew our attention during our research.

One of the first *SE* schemes was proposed by Dawn Xiaodong Song, David Wagner and Adrian Perrig in *Practical Techniques for Searches on Encrypted Data* [1]. It is a proven secure cryptographic scheme for searching over encrypted data without leaking information about the queries. Their approach has a form of probabilistic search: If searching for a word $W$, the scheme will return all positions where $W$ occurs, as well as some possible wrong matches. The user will be able to eliminate the false results when decrypting the results he recieved.
Their scheme uses a stream cipher to encrypt the plaintext. Then, the queries are encrypted in a similar manner, and for searching some word in the cyphertext, the algorithm checks if the xor value between the a cyphertext word and the encrypted query matches a certain form. In the end, it returns a set of positions (and/or cyphertexts) where the match was found.

For a given plain text of size $n$ heir system works in $O(n)$ time for both encryption and query, and almost no additional space is used for ecryption. However, in their approach every word has a fixed size length, which require a padding for supporting variable length words.
Their scheme does not provide support for other types of queries, like subset or comparison. However, it does support boolean operation on queries (e.g. search for $W$ and $W'$, $W$ or $W'$), proximity queries ($W$ near $W'$) and phrase search ($W$ immediately preceeds $W'$).

Another very interesting system is presented by Boneh, Salai and Waters in *Conjunctive, Subset, and Range Queries on Encrypted Data* [2]. They describe a complex SE public-key system which supports all the queries described in the title of the work.
Their system, called *Hidden Vector Encryption* (or, shortly, *HVE*) is based on bilinear maps of composite order. Initially, they describe the model for euality queris, and then they extend it to comparison and subset queries.
The overall complexity of their scheme is described in th table below:

| Operation | Cyphertext Size | Token Size |
|---|---|---|
| Equality $(x = a)$ | $O(N)$ | $O(N)$ |
| Comparison $(x > a)$ | $O(N)$ | $O(N)$ |
| Subset $(x \in A)$ | $O(N)$ | $O(N)$ |

For equality and comparison queries, there are better results, but the big improvement of HVE is that it can be extended to support more complex queries, based on conjunctions. Their final result on such queries are described in the table below:

| Operation | Cyphertext Size | Token Size |
|---|---|---|
| Equality conjunction $((x_1 = a_1) \wedge ... \wedge (x_w = a_w))$ | $O(w)$ | $O(w)$ |
| Comparison conjunction $((x_1 \geqslant a_1) \wedge ... \wedge (x_w \geqslant a_w))$ | $O(wN)$ | $O(w)$ |
| Subset conjunction $((x_1 \in A_1) \wedge ... \wedge (x_w \in A_w))$ | $O(wN)$ | $O(wN)$ |

In [2], Bonet et al reduced the searchable encryption comparison queries to public key broadcast linear encryption, and using the result from their previous work in [3] they created an $O(\sqrt{N})$ size cyphertext scheme for comparison queries over encrypted data.

## 1.3 Our Contribution

This paper focuses on comparison queries over encrypted data. We have focused our research mostly on two systems: the *Hidden Vector Encryption*, presented by Boneh and Waters in [2] and *Public Key Broadcast Linear Encryption*, presented by the same authors in [3]. The transition from *Searchable Encryption* to *Broadcast Encryption* is made in [2].

This paper tries to improve the current space complexity of the comparison queries in current searchable encryption schemes. ($O(\sqrt{N})$ achived by Boneh and Waters in [2] and [3]). Based on the *Public Key Broadcast Linear Encryption* we have developed a new system that approaches the same problem. Our system comes with a big improvement in the complexity (achieving $O(log(N))$ space and $O(log^2(N))$ time complexity), but it has also a drawback: It is using multilinear maps, for which there isn't yet known a secure implementation model.

## 1.4 Paper Structure

This paper is structured as follows:

In *Section 1*, we familiarize the reader with the context of our work. We establish the problems that we attempt to solve and their origin from real-life problems.

In *Section 2*, we will introduce a few theoretical notions required to understand our model: cyclic groups (2.1), bilinear and multilinear maps (2.3), Difiie-Hellman Key-Sharing Protocol(2.4), as well as general primitives in *Searchable Encryption*(2.4) and *Public Key Broadcast Encryption Schemes*(2.5).

In *Section 3* we describe our contribution: The scheme and how it developed until its final form, correctness and security proofs. In *Section 3.8* we describe some improvements and additions that can be made to our model in further works.

In *Section 4* we describe the conclusions we made after our research.

# 2 Notations and Definitions

In this section we will introduce a series of notations and primitieves which will ease the understanding of the schemes presented in the following sections.
Our construction relies heavily on multilinear maps. Because of this, we need to present basic theory related to pairing based cryptography (bilinear and multilinear maps).
Also, we used a similar method to the Diffie-Hellman Key sharing protocol, to share secret keys in our construction, that's why we choosed to present this protocol in this section.

## 2.1 Notations

We will refer to a *Public Key Broadcast Encryption System* as PKBLE System, and a *Searchable Encryption* system as a SE system.

In this paper we will use $N$ for denoting the number of users in a *PKBLE* system, or for the index space size in a *Searchable encryption* system. $n$ will represent the number of bits of $N$ ($n = log(N)$).

## 2.2 Cyclic Groups

A tuple $(G, \cdot)$ formed by a nonempty set $G$, and an operation $\cdot$ is a group iff it respects the following properties:

- **Closure** $\forall x, y \in G,\ x \cdot y \in G$.
- **Associativity** $\forall x, y, z \in G,\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$.
- **Identity Element** $\exists e$ such that $e \cdot x = x \cdot e, \forall x \in G$.
- **Inverse Element** $\forall x \in G, \exists y = x^{-1} \in G$ such that $x \cdot y = y \cdot x = e$.

If, in addition the following property holds:

- **Comutativity** $\forall x, y \in G,\ x \cdot y = y \cdot x$ holds.

then the group is called *abelian* or *comutative*.

We will further use exponential notation for repeated multiplications: $x \cdot x = x^2$, $x \cdot x \cdot x = x^3$, $x \cdot x \cdots x = x^n$.

A group is called cyclic, iff $\exists g \in G$, such that the repeated exponentiation of $g$ generates the whole group (we will name it *generator* for the group G).

The order of an element $a \in G$ (noted as $ord_G(a)$), represents the smallest positive number such that $a^{ord_G(a)} = a$. The order of the group (noted as $ord(G)$) is the size of it (the number of distinct elements). Thus, if for some element $a$, $ord_G(a) = ord(G)$, then $a$ is a generator for $G$.

An important property of the cyclic groups is that each element $a \in G$ can be expressed as $a = g^i$, where $g$ is a generator for $G$.

## 2.3 Pairing-based Cryptography

This subsection describes the basic elements used in pairing-based Cryptography, which are bilinear and multilinear maps.

### 2.3.1 Bilinear maps

Let $G_1$, $G_2$ be two multiplicative cyclic groups of order $p$, and $G_T$ another multiplicative cyclic group of order $p$.

A pairing is a function $e : G_1 \times G_2 \rightarrow G_T$, which satisfies the properties:

- **Bilinearity**: $\forall a, b \in \mathbb{Z}, \forall g_1 \in G_1, \forall g_2 \in G_2 : e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$
- **Non-Degenerate**: $e \neq 1$
- **Computability**: $e$ can be computed efficient.

From the bilinearity property, we can deduce the following equalities which can be usefull:

1. $e(g_1, g_2) = g_T$, where $g_T$ is generator in $G_T$.
2. $e(g_1^a, g_2) = e(g_1, g_2^a) = e(g_1, g_2)^a$
3. $e(g_1^q, g_2) = e(g_1, g_2^q) = e(g_1, g_2)^q = e(g_1, g_2)^0 = 1$.
4. $e(g_1^a g_1^b, g_2) = e(g_1^a, g_2) \cdot e(g_1^b, g_2) = e(g_1, g_2)^a \cdot e(g_1, g_2)^b$

The bilinear maps described above has a prime order $p$. There also exist bilinear maps with composite order. That is, order of $G_T$ is a composite number n.

### 2.3.2 Multilinear maps

One extension of the bilinear maps are $n - multilinear$ maps, which can be defined as follows:
Let $G_1, G_2, ..., G_n$ be $n$ multiplicative cyclic groups of order $p_1, p_2, ...p_n$, and $G_T$ another multiplicative cyclic group of order $T$. Then we can say that $e : G_1 \times G_2 \times ... \times G_n \rightarrow G_T$ is a *n-multilinear map* if:

- **Multilinearity**: $\forall a_1, a_2...a_n \in \mathbb{Z}, \forall g_1 \in G_1, \forall g_2 \in G_2 ... \forall g_n \in G_n$ we have that $e(g_1^{a_1}, g_2^{a_2}, ..., g_n^{a_n}) = e(g_1, g_2, ..g_n)^{a_1 a_2 ... a_n}$
- **Non-Degenerate**: $e \neq 1$
- **Computability**: $e$ can be computed efficient.

As with bilinear maps, rom the bilinearity property, we can deduce the following equalities which can be usefull:

1. $e(g_1, g_2, .., g_n) = g_T$, where $g_T$ is generator in $G_T$.
2. $e(g_1^a, g_2, ..., g_n) = e(g_1, g_2^a, ..., g_n) = e(g_1, g_2, ..., g_n)^a$
3. $e(g_1^p, g_2, ..., g_n) = e(g_1, g_2, .., g_n)^p = e(g_1, g_2)^0 = 1$. (iff $ord(G_T) = p$)
4. $e(g_1^a g_1^b, g_2, ...., g_n) = e(g_1^a, g_2, ...g_n) \cdot e(g_1^b, g_2, ..., g_n) = e(g_1, g_2, ..., g_n)^a \cdot e(g_1, g_2, ..., g_n)^b$

## 2.4 Diffie-Hellman key sharing

The Diffie Hellman Key sharing protocol is a secure way for two users to exchange keys over a public (unsecure) channel. It works in the following way:

Suppose that Alice and Bob want to share with each other a secret key, but they don't have a secure channel to communicate. They must not, under this condition, send the key over the channel. The Diffie-Hellman Key sharing protocol suggests the following solution:

- They initially agree to use a modulus $p$, and a base $g$.
- Alice thinks at a random integer $a$, and Bob at a random integer $b$.
- Alice makes public $g^a$, and Bob $g^b$.
- Now they both can compute $g^{ab} = (g^a)^b = (g^b)^a$, which will be their shared key.

There is a simple extension of this protocol, using bilinear maps, which extends the number of participants to 3:
We have 3 users - A, B and C, which want to share a key through an unsecure channel:

- They initially agree on $G, G_1, G_2$, three multiplicative cyclic groups and a bilinear map $e : G_1 \times G_2 \to G$. Also, they set 3 generators of this groups: $g \in G, g_1 \in G_1, g_2 \in G_2$.
- They pick then 3 (secret) random integers - $a, b, c$.
- A makes public $g_1^a$, $g_2^a$, B $g_1^b$, $g_2^b$ and C $g_1^c$, $g_2^c$.
- Now they both can compute $e(g_1, g_2)^{abc}$, which will be their shared key. (A computes $e(g_1^b, g_2^c)^a = e(g_1, g_2)^{bca}$, B computes $e(g_1^c, g_2^a)^b = e(g_1, g_2)^{cab}$, and C computes $e(g_1^a, g_2^b)^c = e(g_1, g_2)^{abc}$)

We can observe that, if we use a multilinear map instead of a bilinear maps, we can easaly extend the protocol for more users: A $n$-multilinear map will be an efficient way for sharing a key between $n + 1$ users.

## 2.5 Searchable encryption scheme structure

We will begin by describing some general notations we will be using further:
As described in [bowa], a general Seachable encryption scheme will consist of the following:

- $\mathbf{Setup_{SE}}()$ A probabilistic algorithm that outputs a public key $PK$ and a secret key $SK$.
- $\mathbf{Encrypt_{SE}}(PK, I, M)$ An algorithm that encripts the pair (I, M) using the public key $PK$. Here $I$ represents the searchable field, called **index**, and M is the message. It returns a cyphertext **C**.
- $\mathbf{genToken_{SE}}(SK, P)$ Takes as input a secret key SK and a description of a predicate P. It outputs a token $TK_P$, which will be used in decryption.

- **Query$_{\mathbf{SE}}$**$(TK_P)$ The decryption algorithm recieves a token $TK_P$ (for some predicate P) and returns a set of cyphertexts that match the token.

## 2.6 Broadcasting Encryption

Briefly, broadcasting encryption represents an ecnrypting scheme for a number of users such that the ecnrypted text can be decrypted only by a certain subset of the users. In this work, is of interest only a scheme that ranks the users, allows the cyphertext to be decrypted by higher ranked users than the one that encrypted the plaintext. In this way, we will use the broadcast ecryption in our scheme for comparision queries, using the reduction to a SE-system with comparison queries, as shown in *Section 3.2.*

In [2], the PKBLE system is presented as follows:

**Setup$_{\mathbf{PKBLE}}$**$(\mathbf{N}, \lambda)$ A probabilistic algorithm that takes as input N , the number of users in the system, and a security parameter $\lambda$. The algorithm runs in polynomial time in $\lambda$ and outputs a public key PK and private keys $SK_1, ..., SK_N$ , where $SK_u$ is given to user $u$.

**Encrypt$_{\mathbf{PKBLE}}$**$(\mathbf{PK}, \mathbf{i}, \mathbf{M})$ Takes as input a public key $PK$, an integer $i$ satisfying $1 \leqslant i \leqslant N + 1$, and a message $M$. It outputs a ciphertext $C$. This ciphertext is intended for users $\{i, i + 1, ..., N\}$.

**Decrypt$_{\mathbf{PKBLE}}$**$(\mathbf{j}, \mathbf{SK_j}, \mathbf{C})$ Takes as input the private key $SK_j$ for user $j$ and a ciphertext $C$. The algorithm outputs a message $M$ or $\perp$.

**Correctness**
The system must satisfy the following *correctness* property: $\forall i, j \in \{1, ..., N + 1\}$(where $j \leqslant N$ ), and all messages $M$ :

$$Let \begin{cases} (PK, (SK1, ..., SKN)) \leftarrow Setup_{PKLBE}(N, \lambda) \\ C \leftarrow Encrypt_{PKLBE}(PK, i, M) \end{cases}$$

If $j \geqslant i$ then $Decrypt_{PKLBE}(j, SK_j, C) = M$.

**Security**.
We define security of an PKLBE system using two games. The first game is a message hiding game and says that a ciphertext created using index $i = N + 1$ is unreadable by anyone.
The second game is an index hiding game and captures the intuition that a broadcast ciphertext created using index i reveals no non-trivial information about i. We will consider all these games for a fixed number of users, $N$.

**Game 1 - Message Hiding**.
The first game, called the *Message Hiding Game* says that an adversary cannot break semantic security when encrypting using index $i = N + 1$. The game proceeds as follows:

- **Setup** The challenger runs the Setup PKLBE algorithm and gives the adversary PK and all secret keys $\{SK_1, ..., SK_N\}$.

- **Challenge** The adversary outputs two equal length messages $M_0$ , $M_1$ . The challenger flips a R coin $\beta \in \{0,1\}$ and sets $C \leftarrow Encrypt_{PKLBE}(PK, N+1, M_\beta)$. The challenger gives C to the adversary.

- **Guess** The adversary returns a guess $\beta' \in \{0,1\}$ of $\beta$.

  We define the advantage of adversary A in winning the game as:

$$MHAdv_A = |Pr[\beta' = \beta] - 1/2|$$

**Game 2 - Index Hiding**.
The second game, called the Index Hiding Game says that an adversary cannot distinguish between an encryption to index i and one to index i + 1 without the key SK i . The game takes as input a parameter $i \in \{1, ..., N\}$ which is given to both the challenger and the adversary.
The game proceeds as follows:

- **Setup** The challenger runs the $Setup_{PKLBE}$ algorithm and gives the adversary PK and the set of private keys $\{SK_j$ with $j \neq i\}$.

- **Challenge** The adversary outputs a message M. The challenger flips a coin $\beta \in \{0,1\}$ and R computes $C \leftarrow Encrypt_{PKLBE}(PK, i+\beta, M)$. The challenger returns C to the adversary.

- **Guess** The adversary returns a guess $\beta' \in \{0,1\}$ of $\beta$.

  We define the advantage of adversary A as the quantity:

$$IHAdv_A[i] = |Pr[\beta' \neq \beta] - 1/2|$$

  In words, the game captures the fact that even if all users other than $i$ collude they cannot distinguish whether $i$ or $i+1$ was used to create a ciphertext C.

With this games we define a secure PKLBE as follows:
(Definition D.1 in [2])
We say that an $N$-user public-PLBE system is secure if for all polynomial time adversaries A we have that MH $Adv_A$ and IH $Adv_A[i]$ for $i = \{1, ..., N\}$ , are negligible functions of $\lambda$.

# 3 Scheme

Our system is based on the *PKBLE* system described by Boneh and Waters in /citetraitor.

## 3.1 Reducind PKBLE to SE

To ease the understanding of our system, we will present actually present in this work a $PKBLE$-system. Using this model, we can create a $SE$-system which supports comparison queris, with the results from Bonet and Waters in [2] (The result can be found in *Appendix C* of the work): They have showed that any secure $PKBLE$-sustem gives a secure $SE$-system as follows:

- **Setup**($\lambda$) Run $Setup_{PKLBE}(n, \lambda)$ to obtain a public key $PK$ and $n$ secret keys $(SK_1, ..., SK_n)$. Output $PK$ and $SK = (SK_1, ..., SK_n)$.
- **Encrypt**$(PK, s, M)$ where $s \in \{1, ..., n\}$. Output $C \leftarrow Encrypt_{PKLBE}(PK, s, M)$.
- **GenToken**$(SK, \langle P \rangle)$ A predicate P is a number $i \in \{1, ..., n\}$. Output $TK \leftarrow (i, SK_i)$.
- **Query**$(TK, C)$ Let $TK = (i, SK_i)$. Run $Decrypt_{PKLBE}(i, SK_i, C)$.

Using a public-PLBE we thus obtain a $SE$-system. Security follows easily from the properties of public-PLBE.
*Theorem C.1* in [2] states that:
The $/phi(n, 1)$ -searchable encryption system is secure assuming the underlying public-PLBE is secure.

## 3.2 Our Construction

Our construction will try to make a public broadcast ecryption scheme using $O(log(N))$ cyphertext size. We will try to extend the scheme presented in [3], replacing the bilinear map with a multilinear map with $log_2(N)$ dimensions.

We will have then the cyclic groups $G_1$ of order $p_1$, $G_2$ of order $p_2$, ..., $G_n$ of order $p_n$, and group $G_T$ of order $p_1p_2...p_n$, and the multilinear map $e : G_1 \times G_2 \times ... \times G_n \rightarrow G_T$.

First, we want that the cyphertext generated by user $b = (b_1b_2...b_n)$ to be decryptable by users $b'$, where $b' \geqslant b$. Thus, we want to create a vector $k$ (depending on $b$ and $b'$ )such that $e(g^{k_1}, g^{k_2}...g^{k_n}) = 1$ iff $b' \geqslant b$, otherwise $e(g^{k_1}, g^{k_2}...g^{k_n})$ has a random value. This will be the first step in our broadcast encryption scheme, because if we multiply $e(g^{k_1}, g^{k_2}...g^{k_n})$ by $M$ (the message we want to encrypt), we observe that $e(g^{k_1}, g^{k_2}...g^{k_n}) * M = M$ iff $b' \geqslant b$ (ie the user $b'$ must be able to decrypt the message $M$, encrypted by the user $b$).
We will construct $k_i$ as bescribed in the table below:

| | $b_i = 1$ | $b_i = 0$ |
|---|---|---|
| $b'_i = 1$ | $r_ip_i$ | $r_ip_ip_{i+1}...p_n$ |
| $b'_i = 0$ | $r_i$ | $r_ip_i$ |

It can be easily seen that it respects the above constraints.

Now we need to mask the plaintext. We have done this using an approach similar to the Diffie-Hellman key sharing technique. We will multiply $M$ by some $e(g, g, g, ..., g)^q K$, where $K$ is a secret key, generated in the setup function, and $q$ is a random number generated at each encryption. We then generate a vector of pairs:

$$b_i = (b_{i,0}, b_{i,1})$$

For user $U = \overline{u_1 u_2 ... u_n}$, where $u_i \in \{0, 1\}$ ($\overline{u_1 u_2 ... u_n}$ is the $base - 2$ representation of $U$), we will compute $B_U = b_{1,u_1} + b_{2,u_2} + ... + b_{n,u_n}$. And then we generate a secret key for user $U$ as $sk_U = K \cdot B_U^{-1}$.

In our final system, we combine the $b$ and $k$ vectors in a multilinear map, such that we obtain our desired result: The cryptotext resulted from user $u$ can be decrypted by user $u' \Leftrightarrow u' \geqslant u$.

## 3.3 Scheme description

The full description of the scheme is the following:

**setup**$_{PKBLE}(N)$*:*
Let N denote the number of users. (When reducing to searchable encryption we suppose that we want to index out databasse by some integer number in range $[1..N]$). We will use $n = log(N)$.
The setup algorithm first generates $P = p_0 p_1 p_2 ... p_n$, where $p_0$, $p_1$ ... $p_n$ are $n + 1$ random primes. It creates then an n-multilinear map $e$ over a group $G$ of composite order $P$, $e : G^n \to G$. Let $g$ be a random generator of $G$.

Then, we generate some random exponents $r_1$, $r_2$, ... $r_n \in \mathbb{Z}_P$ and $r'_1$, $r'_2$, ... $r'_n \in \mathbb{Z}_P$.

And finally, we generate the secret keys for out $N = 2^n$ users. We generate the following array of pairs of exponents: $(b_{1,0}, b_{1,1}), (b_{2,0}, b_{2,1}), ...(b_{n,0}, b_{n,1})$, and a general secret key $K$, which must be hidden from the users.
For user $U = \overline{u_1 u_2 ... u_n}$, where $u_i \in \{0, 1\}$ ($\overline{u_1 u_2 ... u_n}$ is the $base - 2$ representation of $U$), we will compute $B_U = b_{1,u_1} + b_{2,u_2} + ... + b_{n,u_n}$.
The secret key $sk_U$ for user $U$ will then be $sk_U = K \cdot B_U^{-1}$.

**encrypt**$_{PKBLE}(M, U)$: User $U = \overline{u_1 u_2 ... u_n}$(where $u_i \in \{0, 1\}$) wants to encrypt element $M$.

First, we will compute the exponents $k$: $k_1$, $k_2$, .. $k_n \in \mathbb{Z}_P \times \mathbb{Z}_P$ as follows:

$$k_i = (k_{i,0},\ k_{i,1}) = \begin{cases} (r_i p_i,\ r_i p_i p_{i+1} ... p_n) & u_i = 1 \\ (r'_i,\ r_i p_i) & u_i = 0 \end{cases}$$

Then, we a secret random array $a$: $a_1$, $a_2$ ... $a_n$, and a secret random security parameter $q$, which is invertible in $\mathbb{Z}_P$.

We will compute

$$C = M \cdot e(g, g..., g)^{a_1+a_2+...+a_n} \cdot e(g, g..., g)^{qK}$$

and the following vectors of pairs:

$$e_{i,u_i} = g^{k_{i,u_i}}$$
$$d_{i,u_i} = g^{k_{i,u_i}} g^{a_i}$$
$$f_{i,u_i} = g^{k_{i,u_i}} g^{qb_{i,u_i}}$$

Return as output $(k, C, f, d)$.

**decrypt**$_{PKBLE}(U, (k, C, f, d))$: User $U$ wants to decrypt the message $(k, C, f, d)$. For each bit $u_i$ of $U$, we will choose the appropriate value from $k$, $f$ and $d$ (as shown above). For simplicity, we will refer to it simply as $k_i = k_{i,u_i}...$

And after that, two additional vector, D and F, are constructed as follows:

$$D_i = e(e_1, e_2, ..., e_{i-1}, d_i, e_{i+1}, ..., e_n)$$
$$F_i = e(e_1, e_2, ..., e_{i-1}, f_i, e_{i+1}, ..., e_n)$$

Then, compute

$$B_i = e(e_1, e_2, ..e_{i-1}, g, e_{i+1}, ..., e_n)$$

To decrypt, we need to compute the value

$$C * \frac{B_1 B_2 ... B_n}{D_1 D_2 ... D_n} \left( \frac{B_1 B_2 ... B_n}{F_1 F_2 ... F_n} \right)^{sk_U}$$

If the user is allowed to decrypt, he will obtain the message initially encrypted, $M$. Otherwise, he will recieve random bits of information.

## 3.4 Correctness and Security

To prove the security of our scheme, we will use the gams presented in *Section 2.6*.

### 3.4.1 Correctness

We will say that our PKBLE system is correct iff it satisfied the correctness property enunciated in *Section 2.6*.

User $U$ has encrypted a message $M$. We will prove that only a user $V \geqslant U$ can decrypt the message.

The decryption algorithm will compute

$$C * \frac{B_1 B_2 ... B_n}{D_1 D_2 ... D_n} \left( \frac{B_1 B_2 ... B_n}{F_1 F_2 ... F_n} \right)^{sk_U}$$

First, let $K_p = \prod k_i$. We will then analyze the value $\frac{B_i}{D_i}$:

$$\frac{B_i}{D_i} = \frac{e(g,g,...,g)^{K_p/k_i}}{e(g,g...,g)^{K_p} e(g,g,...,g)^{K_p a_i/k_i}} = \frac{1}{e(g,g...,g)^{K_p} e(g,g,...,g)^{a_i}}$$

Similary, we get that:

$$\frac{B_i}{F_i} = \frac{e(g,g,...,g)^{K_p/k_i}}{e(g,g...,g)^{K_p} e(g,g,...,g)^{K_p b_{i,u_i}/k_i}} = \frac{1}{e(g,g...,g)^{K_p} e(g,g,...,g)^{q b_{i,v_i}}}$$

We will now have now 2 cases:

1. If $V \geqslant U$ then $K_p$ is a multiple of $p_1 p_2 ... p_n$, thus $K_p \equiv 0 \ mod \ P \Rightarrow$ $e(g,g,...,g)^{K_p} = 1$ First we will compute the following components used in decryption:

$$B_i = e(e_1, e_2, ..., e_{i-1}, g, e_{i+1}, ..., e_n)$$
$$D_i = e(e_1, e_2, ..., e_{i-1}, d_i, e_{i+1}, ..., e_n)$$
$$F_i = e(e_1, e_2, ..., e_{i-1}, f_i, e_{i+1}, ..., e_n)$$

   Then, we have that

$$\frac{B_i}{D_i} = \frac{1}{e(g,g,...,g)^{a_i}} \tag{1}$$
$$\frac{B_i}{F_i} = \frac{1}{e(g,g,...,g)^{b_i}} \tag{2}$$
$$(1)(2) \Rightarrow \begin{cases} \prod \frac{B_i}{D_i} = (\prod e(g,g,...,g)^{a_i})^{-1} = (e(g,g,...,g)^{a_1+a_2+...+a_n})^{-1} \\ \prod \frac{B_i}{F_i} = (\prod e(g,g,...,g)^{b_{i,v_i}})^{-1} = (e(g,g,...,g)^{b_{1,v_1}+...+b_{n,v_n}})^{-1} \end{cases} \tag{3}$$
$$\Rightarrow C * \frac{B_1 B_2 ... B_n}{D_1 D_2 ... D_n} \left( \frac{B_1 B_2 ... B_n}{F_1 F_2 ... F_n} \right)^{sk_U} = \tag{4}$$
$$= M \cdot \frac{e(g,..,g)^K \cdot e(g,g...,g)^{a_1+a_2+...+a_n}}{e(g,g,...,g)^{a_1+a_2+...+a_n} \cdot \left(e(g,...,g)^{b_{1,u_1}+...+b_{n,u_n}}\right)^{sk_U}} = \tag{5}$$
$$= M \cdot \frac{e(g,..,g)^K}{\left(e(g,...,g)^{b_{1,v_1}+...+b_{n,v_n}}\right)^{sk_V}} = \tag{6}$$
$$= M \cdot \frac{e(g,..,g)^K}{(e(g,...,g)^{B_V})^{sk_V}} = M \cdot \frac{e(g,..,g)^K}{e(g,...,g)^{B_v \cdot sk_V}} = M \cdot \frac{e(g,..,g)^K}{e(g,...,g)^K} = M \tag{7}$$

   - if $b_i < x_i$ then $K_p$ is a random number $\Rightarrow$ $e(g,g,...,g)^{K_p}$ is also random, which means we cannot decrypt correctly in this case.

2. If $V < U$ then $K_p$ has a random value, and we cannot decrypt the message.

### 3.4.2 Security

We will prove the security of our system, using the two games presentes in *Section 2.6*: *Message Hiding* and *Index Hiding*.

**Game 1 - Message Hiding**

We have our messages $M_0$ and $M_1$, both encrypted by user $u$. This is how the cyphertexts associated with them look (note that we need only the $C$ component of the cyphertext, as the others do not depend on the message to be encrypted):

$$C_0 = M_0 \cdot e(g, g..., g)^{a_1+a_2+...+a_n} \cdot e(g, g..., g)^K$$
$$C_1 = M_1 \cdot e(g, g..., g)^{a_1+a_2+...+a_n} \cdot e(g, g..., g)^K$$

Let $T = e(g, g..., g)^{a_1+a_2+...+a_n} \cdot e(g, g..., g)^K$. Our ecuations become:

$$C_0 = M_0 \cdot T, \quad C_1 = M_1 \cdot T$$

This is trivial?

**Game 2 - Index Hiding**

In this game, the adversary must distinguish between index $i$ ans $i+1$, knowing all $sk_u$, $u \neq i$.

The challenger encrypts the message $M$ with one of the secret keys of users $i$ or $i+1$. The adversary must guess which secret key was used to encrypt. The two cyphertexts vould look like this:

$$sk_i: \quad C_0 = M \cdot e(g, g..., g)^{a_1+a_2+...+a_n} \cdot e(g, g..., g)^q K$$
$$e_{i,u_i} = g^{k_{i,u_i}}$$
$$d_{i,u_i} = g^{k_{i,u_i}} g^{a_i}$$
$$f_{i,u_i} = g^{k_{i,u_i}} g^{qb_{i,u_i}}$$

$$sk_{i+1}: \quad C_1 = M \cdot e(g, g..., g)^{a'_1+a'_2+...+a'_n} \cdot e(g, g..., g)^K$$
$$e'_{i,u_i} = g^{k'_{i,u_i}}$$
$$d'_{i,u_i} = g^{k'_{i,u_i}} g^{a'_i}$$
$$f'_{i,u_i} = g^{k'_{i,u_i}} g^{q'b_{i,u_i}}$$

Note that for the second encryption (with user's $i+1$ secret key) we use different vectors $a$, $k$ (denoted here $a'$ and $k'$). This implies also changes in $e$, $d$, and $f$. Also, in the second case it is used a different parameter $q$ (denoted $q'$).

## 3.5 Complexity Analysis

We will analyze each algorithm regarding time and space complexity.
First, the *setup* algorithm provides uses $O(n) = O(log(N))$ space and time. It generates the vectors $r, r', b$, each of them containing $O(n)$ elements.

The *encryption* algorithm generates the vectors $D$ and $F$. Each of them has $n$ elements, and for each element we have $O(n)$ time complexity for computing it. It results a total space complexity of $O(n) = O(log(N))$ and $O(n^2) = O(log^2(N))$ time complexity.

The *decryption* algorithm needs to compute the $B$ vector, which results in $O(n)$ space, $O(n^2)$ time. Also, computing $C * \frac{B_1B_2...B_n}{D_1D_2...D_n}(\frac{B_1B_2...B_n}{F_1F_2...F_n})^{sk_U}$ needs $O(n^2)$ time. This results in a overall complexity for decryption of $O(n)$ (equivalent to $O(log(N))$) space, $O(n^2)$(equivalent to $O(log^2(N))$) time

Overall, we can observe that the cyphertext size is $O(log(N))$ complexity, and the overall time complexity for decryption and encryption is $O(log^2(N))$ which is an improvement to the present schemes.

## 3.6 Extensions

There are a few possible extensions or modifications which could add benefits or more security to the scheme. We will briefly describe a few of them below.

One of the possible extensions of this model could be generating a new security vector for each user, or for each encryption. (Replacing the current vector $a$ which is generated once in the setup and then used for all users).

Another interesting modification will be replacing the multilinear map with a bilinear map. The purpose of this change is that bilinear maps are more secure than multilinear maps. A starting point could be replacing equations like:

$$e(g^{a_1}, g^{a_2}, .., g^{a_n}) = e(g, g..., g)^{a_1a_2...a_n}$$

with

$$e(g^{a_1}, e(g^{a_2}, .., e(g^{a_{n-1}}g^{a_n}))) = e(g, g)^{a_1a_2...a_n}$$

Another possible extension could be modifying the current scheme to support subset queries.

## 3.7 Implementation

In this section we will describe implementation details of the scheme.

*$setup_{PKBLE}(N)$:*
This is the most difficult part of our scheme implementation. We face two important problems here:

- **generating the group $G$ and the n-multilinear map** $e$: This is difficult because most of the proposed methos for generating multilinear maps are cracked, or not secure enough.
- **generating vector** $b$: If we look close into the scheme, we observe that for any user $U$, the folowing sum $B_U = b_{1,u_1} + b_{2,u_2} + ... + b_{n,u_n}$ must be invertible in the group $G$. (as $sk_U = K \cdot B_U^{-1}$)

For generating our multilinear map, we propose the model presented in [**?** , ??] (detalii despre diverse scheme de generare a aplicatiilor multilineare?)

For the second tricky part of our implemntation, generating the vector $b$, we propose two methods. For both of them we suppose that we work in our system with $N \approx 10^9$ users, which means $n = log(10^9) \approx 30$. Note that this is an extreme case, as for such numbers, it will require great computing power, as the setup function requires $O(N)$ time, because it needs to generate every key for it's users.

**Method 1**
One approach will be to randomly generate $b$. We will show below that the probability that generating the numbers in such manner has a high probability of succes. The algorithm can be rerun until a valid vector is found. We will show that the expected number of runs is small.

To find out the probability of generating a sum which is not coprime with $P$ is similar to the probability of choosing $N$ numbers such that the sum is coprime with $P$.

This probability for one random number to be coprime with $P$ is:

$$\frac{\phi(P)}{P} = \frac{\prod_{i=0}^{n}(p_i - 1)}{\prod_{i=0}^{n} p_i}$$

**Method 2**

For the first $n - 1$ users, generate complete random values for the vector $b$. We will generate then tow values for the last element of the vector $b$ (We remind you that vector $b$ is a vector of pairs)
We then generate all $w = 2^{n-1}$ possible combinations that can be obtained from the vector $b$: $c_1, c_2, ...c_w$. Now, for each $p_i$ in $p_1, p_2, ..., p_n$ we create a set of remainders, $Sp_i$, which initially is empty. Then, for each $c_i$ and $p_j$ we add to $Sp_j$ the value $c_i \mod p_j$

We them choose randomly $2n$ numbers: $r_{10}, r_{11}, ...r_{n0}, r_{n1}$ such that $r_{ij} \notin Sp_i$. Then we need to generate $b_{n,0}$ and $b_{n,1}$ such that the following constrains are met:

$$\begin{aligned} b_{1,w} &\equiv r_{1,w} \mod p_1 \\ b_{2,w} &\equiv r_{2,w} \mod p_2 \\ &... \\ b_{n,w} &\equiv r_{n,w} \mod p_1 \end{aligned}$$

for $w \in \{0, 1\}$.

Finally, using CRT we can compute our 2 ramining values. This approach assures an answer on the first run of the algorithm.

*encryption*$(M, u)$: For the *encryption* method, we only need to generate the array $a$ (using a secure random generator) and compute the components of the cyphertext: $C$, $k$, $d$, $e$, $f$.

*decryption*$(u, (k, C, e, d, f))$: For the *decryption* method, the implementation is trivial. It require just the computation of

$$C \cdot \frac{B_1 B_2 ... B_n}{D_1 D_2 ... D_n} \left( \frac{B_1 B_2 ... B_n}{F_1 F_2 ... F_n} \right)^{sk_U}$$

# 4 Conclusions

Our system modelates a PKBLE system with $O(log(N))$ cyphertext size, with the help of which we can achieve a SE-system with comparison queries with the same cyphertext size complexity.
We can see that our system improves the previous results regarding comparison queries in SE schemes. The previous systems used $O(\sqrt{N})$ size cyphertexts. Also, it remains open to a few extensions:

- create a new system, based on the one presented in this paper but which is using bilinear maps instead of multilinear maps
- extend the system to support more complex queries (subset, conjunctions, etc.)

The system's drawback is that it is based on multilinear maps, which are cryptographic primitives for which there isn't yet a secure model for generating them. However, multilinear maps are a subject of great interest and many researchers are working on finding a secure implementation for them.

# 5 Bibliography

# References


[1] Dawn Xiaodong Song, David Wagner, Adrian Perrig. *Practical Techniques for Searches on Encrypted Data*. In *IEEE Symp. on Security and Privacy, 2000*, 2000.

[2] Dan Boneh, Brent Waters. *Conjunctive, Subset and Range Queries on Encrypted data*. Vadhan S.P. (eds) Theory of Cryptography. TCC 2007. Lecture Notes in Computer Science, vol 4392, Springer, Berlin, Heidelberg, 2007.

[3] Dan Boneh, Amy Salai, Brent Waters. *Fully collusion resistant traitor tracing with short cyphertexts and private keys*. In *Eurocrypt '06*, 2006.

[4] Dan Boneh, Brent Waters. *A Fully Collusion Resistant Broadcast, Trace and Revoke System*. In *ACM Conference on Computer and Communication Security (CCS)*, 2006.

# drafts

Mostly useless stuff, but I don't want to delete it yet...
$genToken_{AI}(P)$*:* We consider a predicate $P$ given as input ($P$ can be a index, meaning that we must decrypr everything greater or equal to that index.). We will further refer to this index as

$encryption(M, u)$:
For the *encryption* method, we need to generate each time a set of random exponents $1_a, ..., a_n$, with the condition that $gcd(a_1 + ... + a_n, p_0 p_1 ... p_n) = 1$. This can be done by random generating the first $n - 1$ exponents, and for the last one, we take care that the required condition is satisfied.

## Applications

Using this approach, we can make comparision queries over encrypted data.

search a mail in an ecnrypted e-mail database over the date (where date is stored as a number).

Searching by sorted numeric fields (dates, age, ).

prefix matching - we can search for words that have some prefix (larger than $[prefix]000000$ and smaller than $[prefix]zzzzzzz$, where 0 is the smallest character, number available and $z$ the greatest).