FACULTY OF COMPUTER SCIENCE

ALEXANDRU IOAN CUZA UNIVERSITY OF IASI

Graduation Thesis

Implementing Searchable Encryption schemes over Multilinear Maps

written by

Talif Victor-Mihai

July, 2018

Thesis Coordinator:

Prof. Dr. Ferucio Laurențiu Țiplea

Faculty of Computer Science

Alexandru Ioan Cuza University of Iasi

Implementing Searchable Encryption schemes over Multilinear Maps

Talif Victor-Mihai

July, 2018

Coordinator:
Prof. Dr. Ferucio Laurențiu Țiplea

Contents

1 Introduction

1.1 Problem Description

The Searchable Encryption technique allows for the retrieval of desired encrypted content without requiring for it to be decrypted prior to the query.
In our methods, we use predicate encryption schemes. For these, each ciphertext **Ct** is accompanied by an attribute vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ and a number of keys $\mathbf{K}$ associated with predicates. Said keys will decrypt Ct only if the predicate is satisfied by the attribute vector $\mathbf{x}$. These methods enable finely-grained access control to the encrypted data, as well as making the encrypted data queriable. [9]

1.2 Related Work

The main method category we are taking into analysis includes the Hidden Vector Encryption(HVE) schemes, proposed by Dan Boneh and Brent Waters. These are predicate encryption schemes in which the attribute vector x has binary entries and each key K is associated with a predicate $\mathbf{y} = (y_1, y_2, ..., y_n)$, which takes as entries either binary values or the wildcard symbol ($\star$), symbolizing a neutral entry ("doesn't matter"). By introducing said symbol, these schemes provide comparison ($x \geq a$) and subset or range ($x \in S$) query functionalities. [5]

We analyze the original implementation of HVE, as well as one designed by Vincenzo Iovino and Giuseppe Persiano over groups of prime order. [9]

These schemes use bilinear maps as safe key-exchange tools and more.

1.3 Contribution

In this paper, we look to expand these schemes over multilinear maps, constructions which enable extended secret sharing and key exchange functionalities, in order to adapt them for future advanced use. Thus, we give a new dimension to the searchable encryption problem and its known solution, HVE, while also generalizing the existing schemes and giving them a greater use width.

For this, we greatly expand their domain of definition, starting with bilinear groups of prime order and expanding not only to multilinear maps, but generalizing for any order by constructing over composite values as well.

By doing so, we open the problem to cases of any cryptographically-efficient order and of any linearity.

1.4 Paper structure

The paper begins by introducing preliminary notions in the first chapter and formulating the searchable encryption problem in the following.

We then present the best known solution for said problem, Hidden Vector Encryption, and formulate two well known constructions.

Our contribution comes in the following chapter, where we look to expand the presented constructions and formulate our own, generalizing and adapting them for multilinear maps.

We then provide additional information for the open problem of feasible multilinear maps by presenting one recent well-known implementation of them: multilinear maps with indistinguishability obfuscation.

We then provide our conclusions in regards to our constructions and their usage.

2 Preliminaries

2.1 Cyclic groups

A group G is defined as a pair $(M, \cdot)$, where M is a set and $\cdot$ is binary operation defined over the set such that $\cdot : M \times M \to M$. For ease of notation, M will now be synonymous to G.

A group has 4 main properties :

1. Closure : $\forall a, b \in G,\ a \cdot b \in G$
2. Associativity : $\forall a, b, c \in G :\ a \cdot (b \cdot c) = (a \cdot b) \cdot c$
3. Identity element : $\exists e \in G : \forall a \in G,\ e \cdot a = a \cdot e = a$
4. Inverse element : $\forall a \in G, \exists a^{-1} \in G : a \cdot a^{-1} = a^{-1} \cdot a = e$

Furthermore, there is another property for abelian groups :

5. Commutativity : $\forall a, b \in G : a \cdot b = b \cdot a$

For ease of notation, we will establish an exponential notation for repeated applications of the operation on the same element :

$$\forall a \in G :\ a \cdot a \overset{\text{def}}{=} a^2 \text{ (and its associative generalization)}$$

A group is called cyclic if there is an element $a \in G$ such that by repeated exponentiation using the operation, it generates the entire set G. We call a a generator of G.

The order of a group is the size of its associated set. We denore this as $ord(G) = |G|$. The order of a group element $a \in G$ is the minimum value of $k \in \mathbb{N}^\star$ such that $a^k = e$. We denote this as $ord(a) = k$.

We notice 2 properties that easily result from this :

1. $ord(e) = 1$
2. $ord(g) = ord(G)$, where g is a generator for G (if it were any less, there would be ungenerated elements, hence the contradiction; any more and it would generate extra values)

Furthermore, each element a of G can be expressed using a generator g as $a = g^i$, where $1 \leq i \leq ord(G)$. Denoting $ord(a) = k$, it follows that $g^{ik} = e$, therefore ik is a multiple of $ord(G)$. From this, k divides $ord(G)$.

Proof. It is a consequence of Lagrange's Theorem. The element generates a subgroup H of G. Lagrange's Theorem proves that the size of H divides the size of G and, implicitly, the size of H is k, the element's order.

We denote the *discrete logarithm* of b in relation to a in the group G the integer x of minimal value such that $a^x = b$ in the group G.

2.2 Bilinear Maps

A bilinear map $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear application that is also non-degenerate. Bilinear maps have the following properties :

1. $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, where g_1 generator for G_1, g_2 generator for G_2, $a, b \in \mathbb{Z}$
2. $e(g_1, g_2) = g_T$, where g_T generator for G_T

Bilinear maps are the prime method of secret sharing and key exchange used in cryptographic protocols related to searchable encryption, including HVE. They are the main tool used in both encrypting and decrypting the message and are perfectly adapted to our predicate based decryption.

2.3 Multilinear Maps

A multilinear map $e : G_1 \times G_2 \times ... \times G_n \rightarrow G_T$ is a multilinear application that is also non-degenerate. Multilinear maps have the following properties :

1. $e(g_1^{a_1}, g_2^{a_2}, ..., g_n^{a_n}) = e(g_1, g_2, ..., g_n)^{\prod_{i=1}^{n} a_i}$
2. $e(g_1, g_2, ..., g_n) = g_T$, where $g_1, g_2, ..., g_n and g_T$ the respective generators

In the following subsections, we will introduce the concept of Decisional Diffie-Hellman assumption(DDH) and its use of bilinear and multilinear maps.

2.4 Diffie-Hellman assumptions

There is no generally efficient method of computing discrete logarithms in cyclic groups. This provides a great functionality for public-key cryptosystems, as well as the primary way of proving security.

2.4.1 Computational Diffie-Hellman assumption (CDH)

We consider the Diffie-Hellman key-exchange : Alice and Bob decide to operate using a finite cyclic group G and one of its generators, g. Alice and Bob respectively pick two random exponents $a, b \in \overline{1,\ ord(G)}$ and output g^a,g^b to one another. Thus, they generate a shared secret key g^{ab}.

1. A chooses exponent $a \in \overline{1,\ ord(G)}$, B chooses exponent $b \in \overline{1,\ ord(G)}$
2. A sends g^a to B, B sends g^b to A
3. A computes the key $(g^b)^a$, B computes the key $(g^a)^b$
4. A and B now share the secret key K=g^{ab}

Any passive eavesdropper must have a hard time computing the secret key if they happen to obtain g^a, g^b. This allows us to define the Computational Diffie-Hellman assumption as follows : given the function $DH_G(g^a, g^b) = g^{ab}$, the group G satisfies the CDH assumption if there is no efficient algorithm to compute $DH_G(x, y)$, $\forall x, y \in G$.

However, this only works for a 2-party agreement. In order to efficiently share a key in a 3-party environment, the use of bilinear maps was introduced. We will now define the 3-way Diffie-Hellman key-exchange, using the finite cyclic group G, its generator g and the bilinear map $e : G \times G \to G_T$:

1. A chooses exponent $a \in \overline{1,\ ord(G)}$, B chooses exponent $b \in \overline{1,\ ord(G)}$, C chooses exponent $c \in \overline{1,\ ord(G)}$
2. A sends g^a to B and C, B sends g^b to A and C, C sends g^c to B and C
3. A computes $e(g^b, g^c) = e(g, g)^{bc}$, then raises it to its previously generated exponent, $(e(g, g)^{bc})^a = e(g, g)^{abc}$; B and C similarly generated $e(g, g)^{abc}$
4. A, B and C now share the secret key K=$e(g, g)^{abc}$

The CDH assumption can now be rephrased for 3-party contexts. This is called the Computational Bilinear Diffie-Hellman assumption (CBDH) and states as follows

: given the function $BDH_G(g^a, g^b, g^c) = e(g,g)^{abc}$, the group G satisfies the CBDH assumption if there is no efficient algorithm to compute $BDH_G(x,y,z)$, $\forall x,y,z \in G$.

It only follows to extend the concept to a greater number of parties. We do this with the aid of multilinear maps. More specifically, in a $N+1$-party context $(A_1, A_2, ..., A_{N+1})$, with $N \in \mathbb{N}$, $N \geq 2$, we will make use of a N-linear map e for the Diffie-Hellman key-exchange:

1. $\forall i \in \overline{1,\ N+1}$, A_i chooses exponent $a_i \in \overline{1,\ ord(G)}$
2. $\forall i \in \overline{1,\ N+1}$, A_i sends a_i to A_j, $\forall j \in \overline{1,\ N+1}$, $j \neq i$
3. $\forall i \in \overline{1,\ N+1}$, A_i computes $\underline{(e(g^{a_1}, g^{a_2}, ..., g^{a_{N+1}}))^{a_i}} = e(g,g,...,g)^{\Pi_{i=1}^{N+1} a_i}$ (the map exponents are a_j, $\forall j \in \overline{1,\ N+1}$, $j \neq i$)
4. $\forall i \in \overline{1,\ N+1}$, A_i now share the secret key $e(g, g..., g)^{\Pi_{i=1}^{N+1} a_i}$

We now rephrase CDH for multiparty contexts. This is called the Computational Multilinear Diffie-Hellman assumption (CMDH) and states as follows: given the function $MDH_G(g^{a_1}, g^{a_2}, ..., g^{a_N}) = e(g,g,...,g)^{\Pi_{i=1}^{N} a_i}$, the group G satisfies the CMDH assumption if there is no efficient algorithm to compute $MDH_G(x_1, x_2, ..., x_N)$, $\forall x_1, x_2, ..., x_N \in G$.

2.4.2 Decisional Diffie-Hellman assumption (DDH)

CDH is a good assumption for security purposes, but insufficient for practical cryptographic use. While an eavesdropper cannot compute the shared secret key g^{ab}, it can often happen that partial secrets and valuable information can be uncovered. As stated in [3], an eavesdropper may still be able to predict a concerning percentage of the shared key. Thus, a stronger security assumption is required. It needs to analyze the probability of the eavesdropper predicting a correct result for the shared key.

We now define the Decisional Diffie-Hellman assumption (DDH). As we did for CDH, we take into account multiple context, namely the 2-party, the 3-party and the multiparty ones.

For the 2-party context, the DDH is as follows : given the same function $DH_G(g^a, g^b) = g^{ab}$, the group G satisfies the DDH assumption if given a, b, c randomly chosen in $\overline{1,\ ord(G)}$, there is no efficient algorithm that distinguishes between the triplets $< g^a, g^b, g^{ab} >$ and $< g^a, g^b, g^c >$. To further explicate the concept, it's equivalent and easier to understand to state that there is no efficient algorithm that probabilistically determines if, given the triplet, $ab = c$.

Given G, its generator g and denoting $n = ord(G)$, we define the following distribution $P(\lambda)$: [5]

$$\begin{gathered} a, b \overset{R}{\leftarrow} \overline{1,\ ord(G)} \\ Z \leftarrow ((n, G, e), g, g^a, g^b) \\ T \leftarrow g^{ab} \\ Output : (Z, T) \end{gathered}$$

Therefore, we obtain $(Z, T) \leftarrow P(\lambda)$. We define the advantage of algorithm $\mathcal{A}$ in solving the Decisional Diffie-Hellman problem :

$$DH_{Adv_{\mathcal{A}}}(\lambda) := \left| Pr[\mathcal{A}(Z, T) = 1] - Pr[\mathcal{A}(Z, R) = 1] \right|$$

where $R \overset{R}{\leftarrow} G$, which we can denote as $R = g^c$, $c \overset{R}{\leftarrow} \overline{1,\ ord(G)}$.

For the 3-party context, given the use of bilinear maps, the Decisional Bilinear Diffie-Hellman assumption (BDH) states the following : the group G satisfies BDH assumption if given a, b, c, d randomly chosen in $\overline{1,\ ord(G)}$, there is no efficient probabilistic algorithm that determines if $abc = d$. More specifically, by generating a random element in G_T, denoted $R = e(g, g)^d$, there is a negligible probability that $R = e(g, g)^{abc}$.

Given G, its generator g and the bilinear map $e : G \times G \to G_T$ and also denoting $n = ord(G)$, we define the following distribution $P(\lambda)$:

$$\begin{gathered} a, b, c \overset{R}{\leftarrow} \overline{1,\ ord(G)} \\ Z \leftarrow ((n, G, e), g, g^a, g^b, g^c) \\ T \leftarrow e(g, g)^{abc} \\ Output : (Z, T) \end{gathered}$$

Therefore, we obtain $(Z, T) \leftarrow P(\lambda)$. We define the advantage of algorithm $\mathcal{A}$ in solving the Decisional Bilinear Diffie-Hellman problem :

$$BDH_{Adv_{\mathcal{A}}}(\lambda) := \left| Pr[\mathcal{A}(Z, T) = 1] - Pr[\mathcal{A}(Z, R) = 1] \right|$$

where $R \overset{R}{\leftarrow} G_T$, which we can denote as $R = e(g, g)^d$, $d \overset{R}{\leftarrow} \overline{1,\ ord(G)}$.

For the multiparty context, given the use of multilinear maps, the Decisional Multilinear Diffie-Hellman assumption (MDH) states the following : the group G satisfies MDH assumption if given $a_1, a_2, ..., a_{N+1}, a_R$ randomly chosen in $\overline{1,\ ord(G)}$, there is no efficient probabilistic algorithm that determines if $\prod_{i=1}^{N+1} a_i = a_R$. Namely, a randomly generated element of G_T, $R = e(g, g, ..., g)^{a_R}$ has a negligible probability of satisfying $R = e(g, g, ..., g)^{\Pi_{i=1}^{N+1} a_i}$.

Given G, its generator g and the multilinear map $e : G^N \to G_T$ and also denoting $n = ord(G)$, we define the following distribution $P(\lambda)$:

$$\begin{gathered}
a_1, a_2, ..., a_{N+1} \overset{\text{R}}{\leftarrow} \overline{1,\ ord(G)} \\
Z \leftarrow ((n, G, e), g, g^{a_1}, g^{a_2}, ..., g^{a_{N+1}}) \\
T \leftarrow e(g, g, ..., g)^{\Pi_{i=1}^{N+1} a_i} \\
Output : (Z, T)
\end{gathered}$$

Therefore, we obtain $(Z, T) \leftarrow P(\lambda)$. We define the advantage of algorithm $\mathcal{A}$ in solving the Decisional Multilinear Diffie-Hellman problem :

$$MDH_{Adv_{\mathcal{A}}}(\lambda) := \Bigg| Pr[\mathcal{A}(Z, T) = 1] - Pr[\mathcal{A}(Z, R) = 1] \Bigg|$$

where $R \overset{\text{R}}{\leftarrow} G_T$, denoted as $R = e(g, g, ..., g)^{a_R}$, $a_R \overset{\text{R}}{\leftarrow} \overline{1,\ ord(G)}$.

For all the mentioned cases, we say that the assumption in cause is satisfied if the advantage function has a negligible function for any algorithms $\mathcal{A}$.

It is worth noting that we formulated these constructions symmetrically (the groups that define the maps are identical, G). The assymetrical constructions follow the same pattern, using the appropriatedly generated elements from groups $G_1, G_2, ..., G_N$, according to the way we defined e:

$$e : G_1 \times G_2 \times ... \times G_N \to G_T.$$

Functionality. One of the main uses of DDH and the security it grants is for establishing a secure Non-Interactive Key Exchange (NIKE[6]). This looks to grant systems the possibility to easily create a shared secret key between any number of users which can act as a known key for their group, while any attempts at eavesdropping in order to obtain said key to leak no useful information and keep any attack-intended computation hard.

3 Searchable Encryption

3.1 Reasoning

Safe data storage is a key piece of life in this day and age. Whether we are talking about personal use, for storing virtual data of great relevance to the user or data desired to remain private, 'under lock and key', or if we reference large-scale business use, where company secrets must without a shadow of doubt not fall in foreign hands, data storage facilities are required to provide near impossibility of information leaks.

Strong encryption does not stand without shortcomings though. And a valid one and the one in question regarding our approached subject is accessibility and retrievability : How can we easily query heavily encrypted data without providing any unnecessary information to any eavesdropping party?

This is what Searchable Encryption provides. The encrypted data can be easily queried and handled without any need for decryption. This guarantees that no information will be leaked on the database level.

Among the techniques used for this, we choose to talk about Predicate Encryption schemes.

3.2 Predicate Encryption Schemes

Let's consider the following example. Among a plethora of encrypted data files lie the nuclear codes of your respective country (if denuclearized, consider a thing of similar importance and impact). You desire to retrieve them without leaking anything. Therefore, without decryption.

If you had the bright idea of using Predicate Encryption schemes, the codes are hiding in plain sight. Or more so, in plain text. The codes will have your selection of suggestive and specific characteristics associated with them. To name a few examples: *isImportant*, *isTheBomb*, *isRadiating* etc.

Given these, the retrieval of said encrypted data is really simple. True enough, the eavesdroppers can also query just as easily, but they have no information on how to decrypt it. Thus, it is useless for them.

This is the way Predicate Encryption schemes operate. In technical terms, any stored cyphertext $\mathbf{Ct}$ has an attribute vector $\mathbf{x}$ associated with it. The keys used in decrypting it, K, are dependant on query predicates and will succesfully decrypt $\mathbf{Ct}$ if and only if the predicates in question are satisfied by the attribute vector $\mathbf{x}$. This offers finely-grained access control to the date, as well as fulfilling the required query function.

Efficient Predicate Encryption schemes for searchable encryption must focus on four query-related aspects:

1. **Complexity**. Firstly, any query should be computationally simplified, taking into consideration both the time and the space complexity of the system behind it.

2. **Query complexity**. We must also take into account that our system is prefered to support various type of queries. If we have an attribute that takes numerical value, for example *cost*, one might desire to query based on a minimal or maximal value for it. Hence, we'd like to support comparison queries for our system. other examples of a query type that might be desired are range queries, where the value must be bounded between 2 values, as well as subset queries, where the value must be an element of a certain subset of values.

3. **Conjunctive queries**. Adding to the previous idea referring to query complexity, we'd like to chain multiple predicates to form a more complex query. For example, in an encrypted data set of pictures, we'd like to retrieve the ones who satisfy the following : $color = red$, $size = small$, $hasFlowers = true$. Given these, we'd like to easily perform the query without substantially increasing its complexity.

4. **Predicate-based decription**. The query predicate aids the decryption algorithm in revealing the plain message hidden in the cyphertext Ct.

Based on these, we can now formulate a construction for a searchable encryption system.

3.3 Basic Structure

We start by defining important notions and notations for our construction.

Let Σ be the alphabet over which our predicates are defined. Namely, for our analysis, let it be the set of binary strings of length lower than a fixed value λ. Let a predicate P over Σ be defined as a function $P : \Sigma \to \{0, 1\}$, where for a binary string $I \in \Sigma$, we say that I satisfies P if and only if $P(I) = 1$.

Let Φ be a subset over Σ. We call a **Φ-searchable** public key system a construction comprising of the following 4 algorithms:

1. ***Setup***(λ). The probabilistic algorithm that takes a security parameter as input and generates the cryptosystem, outputting the public key **PK** and masterkey **MsK**.

2. ***Encrypt*(PK, *(I,M)*)**. The algorithm takes as input a pair (I, M), where M is the message to be encrypted and I is the attribute vector ("*a searchable field*"[5]), called the **index**. It then encrypts the message using the public key PK, generating the cyphertext **Ct**.

3. ***GenToken*(MsK, <*P*>)**. Given the predicate $P \in \Sigma$ and its description, the algorithm generates a decryption token **TK** using the masterkey MsK.

4. ***Query*(TK, Ct)**. The query algorithm. Given the cyphertext Ct and the token TK, if the predicate $P \in \Sigma$ used to generate TK is satisfied by the index I, the algorithm should successfully decrypt plaintext M. Otherwise, it should return $\perp$ (Invalid).

Correctness. After generating a public-key system (PK, MsK), we generate a cyphertext $C \leftarrow \textit{Encrypt}(PK, (I, M))$. Given a predicate $P \in \Sigma$, we seek to find out if I satisfies P. We can do this by generating the token $TK \leftarrow \textit{GenToken}(MsK, <P>)$ and the verifying if $M \leftarrow \textit{Decrypt}(TK, C)$, as this will only happen if $P(I) = 1$.

We call this property **query correctness**, as we basically query using predicate P in our cryptosystem. Extending this to the general case, we obtain the following :

$$\begin{gathered}\forall (I, M) \in \Sigma \times \mathcal{M},\ \forall P \in \Phi :\\ (PK, MsK) \overset{\text{R}}{\leftarrow} \textit{Setup}(\lambda)\\ C \overset{\text{R}}{\leftarrow} \textit{Encrypt}(PK, (I, M))\\ TK \overset{\text{R}}{\leftarrow} \textit{GenToken}(MsK, <P>)\\ \text{If } P(I) = 1 \text{ then } \textit{Query}(TK, C) = M.\\ \text{If } P(I) = 0 \text{ then } \textit{Query}(TK, C) = \perp.\end{gathered}$$

Security game. Given a Φ-searchable system $\mathcal{E}$, we propose a query security game in order to verify that any number of tokens TK do no reveal unintended information about the plaintext :

Setup. The challenger runs $\textit{Setup}(\lambda)$ and gives the adversary the public key PK.

Query phase 1. The adversary adaptively selects predicates $P_1, P_2, ..., P_{q_1} \in \Phi$ and outputs their descriptions to the challenger. It in turn replies with the tokens $TK_j \leftarrow \textit{GenToken}(MsK, <P_j>)$. These will be refered to as **predicate queries**. [5]

Challenge. The adversary issues two pairs (I_0, M_0) and (I_1, M_1) restricted to satisfying the following statements:

1. $\forall j \in \overline{1,\ q_1},\ P_j(I_0) = P_j(I_1)$

2. If $M_0 \neq M_1$, $\forall j \in \overline{1,\ q_1}$, $P_j(I_0) = P_j(I_1) = 0$

The challenger randomly selects $x \in \{0, 1\}$ and generates:

$$C_* \leftarrow Encrypt(PK, (I_x, M_x))$$

and gives it to the adversary. The two previously stated restrictions assure that the tokens will not lead to a trivial break of challenge. The first one ensures that the tokens given to the adversary will not aid in him in easily distinguishing I_0 from I_1, while the second one ensures the same for distinguishing M_0 from M_1.

Query phase 2. The adversary continues to request tokens based on his adaptive choice of predicates $P_{q_1+1}, P_{q_1+2}, .., P_q$ subjected to the two restrictions above.

Guess. The adversary guesses a value $x^{'} \in \{0, 1\}$ for x.

We define the advantage of adversary $\mathcal{A}$ in attacking the system $\mathcal{E}$ as his probability of making a correct guess as opposed to him making a random guess:

$$QuAdv_{\mathcal{A}} = \left|Pr[x^{'} = x] - \tfrac{1}{2}\right|$$

Selective Security. We also propose a slightly weaker security game, where the adversary commits to the two binary strings I_0, I_1 prior to the setup:

Setup. The adversary selects the two binary strings I_0, I_1. The challenger runs $Setup(\lambda)$ and gives the adversary the public key PK.

Query phase 1. The adversary adaptively selects predicates $P_1, P_2, ..., P_{q_1} \in \Phi$ and outputs their descriptions to the challenger. They are subjected to the following restriction :

$$\forall j \in \overline{1,\ q_1},\ P_j(I_0) = P_j(I_1)$$

The challenger replies with the tokens $TK_j \leftarrow GenToken(MsK, <P_j>)$.

Challenge. The adversary outputs two messages M_0, M_1 having the following restriction :

$$\text{If } M_0 \neq M_1,\ \forall j \in \overline{1,\ q_1},\ P_j(I_0) = P_j(I_1) = 0$$

The challenger randomly selects $x \in \{0, 1\}$ and generates $C_* \leftarrow Encrypt(PK, (I_x, M_x))$ and gives it to the adversary.

Query phase 2. The adversary continues to request tokens based on his adaptive choice of predicates $P_{q_1+1}, P_{q_1+2}, .., P_q$ subjected to the two restrictions above.

Guess. The adversary guesses a value $x^{'} \in \{0, 1\}$ for x.

We similarly define the advantage of adversary $\mathcal{A}$ in attacking the system $\mathcal{E}$ as his probability of making a correct guess as opposed to him making a random guess:

$$sQuAdv_{\mathcal{A}} = \left|Pr[x^{'} = x] - \tfrac{1}{2}\right|$$

Comparative consequences. As a follow-up to the previous security games, we present an example of comparative queries and how tokens generated from comparative predicates should not reveal any unwanted information about the plaintext.

Let $P_\sigma \in \Phi$ be the predicate that evaluates if the value is greater than or equal to $\sigma \in \Sigma$, where Σ has a partial order relation defined over it :

$$P_\sigma(x) = \begin{cases} 1, & \text{if } x \geq \sigma \\ 0, & \text{otherwise} \end{cases}$$

Given multiple tokens of predicates $P_{\sigma_1}, P_{\sigma_2}, ..., P_{\sigma_n}$, subjected to $\sigma_1 \leq \sigma_2 \leq ... \leq \sigma_n$, an adversary can use the tokens to distinguish between two cyphertexts associated with indexes with values situated in different ranges of type $[\sigma_i, \sigma_{i+1}]$, but has no additional information when trying to distinguish between cyphertexts associated with indexes in the same range.

Identity-Based Encryption (IBE). A coveted goal for past cryptographers, the identity-based encryption scheme came as a following of Shamir's proposition of creating a system in which any string can be used as a public key : name, e-mail etc. After multiple unsatisfactory construction, Dan Boneh and Matt Franklin's 2001 construction[4] was the first widely adopted solution for the problem.

An IBE scheme $\mathcal{E}$ is composed of 4 algorithms : *Setup*, *Extract*, *Encrypt* and *Decrypt*.

1. **Setup(λ)**. Takes as input a security parameter and publicly outputs system parameters **par** (message space $\mathcal{M}$, cyphertext space $\mathcal{C}$) and privately outputs the masterkey MsK to the private key generator (PKG).

2. **Extract(ID, MsK)**. Taking as input a random binary string **ID**, the PKG uses it and the masterkey MsK to generate a private key k_{ID} associated with it.

3. **Encrypt(par, ID, M)**. Generates a ciphertext Ct by encrypting message **M** using **ID** and system parameter **par**.

4. **Decrypt(k_{ID}, Ct)**. Decrypts cypertext using the associated private key k_{ID}.

Article [5] also presents an anonymous IBE construction resulted from the SE structure. An anonymous IBE has the added property of the cyphertext not revealing anything about the identity of the recipient. Let $P_\sigma \in \Phi$ be the predicate that evaluates if the equal to $\sigma \in \Sigma$:

$$P_\sigma(x) = \begin{cases} 1, & \text{if } x = \sigma \\ 0, & \text{otherwise} \end{cases}$$

We denote Φ_{eq}=P_σ for all values of $\sigma \in \Sigma$. Our construction has the following structure:

1. **Setup(λ)**. The exact algorithm used in the searchable encryption construction. Outputs PK as the IBE parameters and MsK as the masterkey.
2. **Encrypt(PK,*(I,M)*)**. The exact algorithm used in the searchale encryption construction. Outputs cyphertext Ct.
3. **Extract(MsK, *I*)**. Given that the index $I \in \Sigma$, this algorithm outputs $TK_I \leftarrow GenToken(MsK, <P_I>)$.
4. **Decrypt(TK$_I$, Ct)**. Outputs $Query(TK_I, Ct)$.

If the searchable encryption construction is corect, given the IBE construction we previously presented, it easily follows that the newly-formed IBE construction will be valid.

3.4 Trivial Construction

As presented in [5], we describe the trivial construction of a searchable encryption system:

Let Φ={P_1, P_2, ..., P_γ}.

1. **Setup'(λ)**. The algorithm runs $Setup(\lambda)$ γ times. This outputs $PK' \leftarrow (PK_1, PK_2, ..., PK_n)$ and $MsK' \leftarrow (MsK_1, MsK_2, ..., MsK_n)$.
2. **Encrypt'(PK', *(I,M)*)**. For $1 \leq j \leq \gamma$:
$$C_j \leftarrow \begin{cases} \text{Encrypt}(PK_j, M), & \text{if } P_j(I) = 1 \\ \text{Encrypt}(PK_j, \perp), & \text{otherwise} \end{cases}$$

   We output $C \leftarrow (C_1, C_2, ..., C_\gamma)$.

3. **GenToken(MsK', <P$_j$>)**. The description $<P_j>$ is actually the index j the predicate has in Φ. Therefore, we output $TK \leftarrow (j, MsK_j)$.

4. **Decrypt'(TK,C)**. Given the aforementioned structure, we output $Query(MsK_j, C)$.

Correctness. We use the same notation as in the construction. Let (I, M) be a random index-message pair.

For each C_i, the message M will be encrypted if the predicate P_i is satisfied; otherwise, $\perp$ is encrypted.

A query with token TK associated to predicate P_j will always successfully decrypt C_j; if I satisfies P_j, the output will be M. If not, the output will be $\perp$.

Security proof. We construct a security game similar to the previous one. Let $\mathcal{A}$ be the adversary in this game.

The game will have $n+1$ iterations. We analyze the generalized iteration i.

1. The challenger runs $Setup(\lambda)$ and obtains $PK' \leftarrow (PK_1, PK_2, ..., PK_n)$ and $MsK' \leftarrow (MsK_1, MsK_2, ..., MsK_n)$. It gives PK to $\mathcal{A}$. It then responds to every predicate description the adversary presents to him, sending him the respective tokens.

2. The adversary presents 2 pairs (I_0, M_0) and (I_1, M_1) subjected to the previously mentioned query security game restrictions. For each value of $j \in \overline{1,n}$, we compute:

$$C_j \leftarrow \begin{cases} Encrypt'(PK_j, M_0), & \text{if } P_j(I_0) = 1 \text{ and } j \geq i \\ Encrypt'(PK_j, M_1), & \text{if } P_j(I_1) = 1 \text{ and } j < i \\ Encrypt'(PK_j, \perp), & \text{otherwise} \end{cases}$$

   The challenger then gives the adversary $C \leftarrow (C_1, C_2, ..., C_n)$.

3. Adversary $\mathcal{A}$ continues to request query tokens (the associated predicates are subjected to the query security game restrictions). In the end, $\mathcal{A}$ guesses $\beta \in \{0, 1\}$. We denote $\mathrm{EXP}_{QU}^{(i)}[\mathcal{A}]$ the probability of β being 1.

We notice that, knowing the previous security game and the absolute value sum inequality :

$$2 \cdot QuAdv_{\mathcal{A}} = \left|\mathrm{EXP}_{QU}^{(1)}[\mathcal{A}]\text{-}\mathrm{EXP}_{QU}^{(n+1)}[\mathcal{A}]\right| \leq \sum_{i=1}^{n+1} \left|\mathrm{EXP}_{QU}^{(i)}[\mathcal{A}]\text{-}\mathrm{EXP}_{QU}^{(i+1)}[\mathcal{A}]\right|$$

Considering the structure of the criptotexts in consecutive iterations, we notice that it would have minute differences and knowing that our structure does not have any chosen plaintext attacks, the value should be negligible. Therefore, $QuAdv_{\mathcal{A}}$ is of negligible value. Thus, security is proven.

An obvious shortcoming of this "brute-force" approach is the fact that it has to generate a structure linearly proportionate to the size of the predicate set. Both spacially, as it generates a large number of keys, and timewise, as the message is encrypted multiple times. It comes naturally that we'd prefer a more universal approach; given a large predicate set, it would facilitate our construction greatly if we'd have a constant scaling with it. We'd like the number of possible predicates to not affect our complexity and rather only have predicate length influence it.

4 Hidden Vector Encryption

4.1 Definition

In article [5], Dan Boneh and Brent Waters introduce a powerful Searchable Encryption construction, able to not only scale well with a large set of possible predicates, but also provide the desired functionalities of comparative, range and subset queries.

This is done by adapting the nature of the predicates. Let Σ, as previously mentioned, be a finite set of values over which we define our indexes. For the current case, we consider $\Sigma = \{0, 1\}$, the binary alphabet over which we formulate our indexes I. Let $\Sigma_\star = \Sigma \bigcup \{\star\}$ be the alphabet over which we define our predicates. the **wildcard** symbol ($\star$) acts as a neutral symbol, an entry that marks our intention not to care for its associated index in the predicate. In our evaluation, a comparison to this symbol will always return *true*.

A predicate P is defined as $P = (\sigma_1, \sigma_2, ..., \sigma_l) \in \Sigma_\star^l$ and, given as input an index $x = (x_1, x_2, ..., x_l) \in \Sigma^l$, will evaluate it as such:

$$P(x) = \begin{cases} 1, & \text{if } \forall 1 \leq i \leq l : (\sigma_i = x_i) \vee (\sigma_i = \star) \\ 0, & \text{otherwise} \end{cases}$$

In simpler terms, the predicate must match the input on each binary entry. Therefore, a predicate checks if its requirements, expressed as binary entries, are matched in its offered input, while the wildcard entries are irrelevant for its evaluation, as the predicate does not care for it.

4.2 Bilinear Maps over groups of composite order

The construction that follows makes use of bilinear maps over groups of composite order $n = pq$, where p, q are large primes.

In a group G of composite order $n = pq$, two generating subgroups can be formed, G_p of order p and G_q of order q, such that each element can be expressed uniquely as a product $(g_p^{i_1} \cdot g_q^{i_2})$, where g_p generator of G_p and g_q generator of G_q and $1 \leq i_1 \leq p$, $1 \leq i_2 \leq q$. Furthermore, if the group itself has a generator g, we can take $g_p = g^q$ and $g_q = g^p$.

This leads to special properties if used as a basis for a bilinear map. Let $e : G \times G \to G_T$ be a bilinear map. From the previously mentioned properties of bilinear maps, it follows that:

$$\forall x \in G_p, \forall y \in G_q:$$
$$e(x, y) = e(g_p^{i_1}, g_q^{i_2}) = e(g_p, g_q)^{i_1 i_2} = (e(g, g)^{pq})^{i_1 i_2} = (e(g, g)^{ord(G)})^{i_1 i_2} = 1^{i_1 i_2} = 1$$

This grants us a new way of obtaining identity from our bilinear map, by creating so called blinding factors from the each subgroup that end up annulling themselves via computation.

4.3 Construction

4.3.1 Boneh-Waters construction

As presented in [5], the construction will generalize $\Sigma = \mathbb{Z}_m$. For the rather practical case of binary indexes, it follows that $m = 2$. The message space $\mathcal{M}$ is set to be a small subset of G_T, namely one with size $|\mathcal{M}| < |G_T|^{1/4}$. This aids us in identifying a valid plain message M$\in$ $\mathcal{M}$ and differentiating it from an incorrectly decrypted message, which signals that the query is to return $\perp$.

Much like the previously mentioned searchable encryption system constructions, the HVE system will also be composed of 4 algorithms : *Setup*, *Encrypt*, *GenToken* and *Query*.

1. **Setup(λ)**. The algorithm generates two prime numbers p, $q > m$. It then generates group G of order $n = pq$. It then randomly generates the following elements:

$$(u_1, h_1, w_1), (u_2, h_2, w_2), ..., (u_l, h_l, w_l) \in G_p^3,$$
$$g, v \in G_p,$$
$$g_q \in G_q,$$
$$\alpha \in \mathbb{Z}_p$$

These all compose the masterkey **MsK**.
It then generates a number of blinding factors:

$$(R_{u,1}, R_{h,1}, R_{w,1}), (R_{u,2}, R_{h,2}, R_{w,2}), ..., (R_{u,l}, R_{h,l}, R_{w,l}) \in G_q,$$
$$R_v \in G_q$$

It then publishes the public key **PK** using the description of group G and:

$$\begin{gathered} g_q, \\ V = vR_v, \\ A = e(g,v)^{\alpha}, \\ \forall i,\ 1 \leq i \leq l : \begin{cases} U_i = u_i R_{u,i} \\ H_i = h_i R_{h,i} \\ W_i = w_i R_{w,i} \end{cases} \end{gathered}$$

The message space $\mathcal{M}$ is also set at this point, as to satisfy:

$$|\mathcal{M}| < |G_T|^{1/4}$$

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in \mathcal{M}$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_n$ and random $Z, (Z_{1,1}, Z_{1,2}), (Z_{2,1}, Z_{2,2}), ..., (Z_{l,1}, Z_{l,2}) \in G_q$

   (b) Output the following cypertext:
   $$C = \left(C' = MA^s;\ \ C_0 = V^s Z;\ \ \forall i, 1 \leq i \leq l : \begin{cases} C_{i,1} = (U_i^{I_i} H_i)^s Z_{i,1} \\ C_{i,2} = W_i^s Z_{i,2} \end{cases} \right)$$

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm generates the set S of indexes $1 \leq i \leq l$ such that $I_i \neq \star$. It then generates the random values $(r_{i,1}, r_{i,2}) \in \mathbb{Z}_p^2$, for each $i \in S$. The token is generated thus:

$$TK{=}\big(I_\star; K_0 = g^{\alpha} \prod_{i \in S} (u_i^{I_i} h_i)^{r_{i,1}} w_i^{r_{i,2}}; \forall i \in S : K_{i,1} = v^{r_{i,1}}, K_{i,2} = v^{r_{i,2}}\big)$$

4. **Query(TK,C)**. Keeping the preceding notations, we compute:

$$M \leftarrow C' \;/\; \left(e(C_0, K_0) \;/\; \prod_{i \in S} e(C_{i,1}, K_{i,1})\, e(C_{i,2}, K_{i,2}) \right)$$

If M$\in \mathcal{M}$, then output M. Else, output $\perp$.

Proof. For our proof of query correctness, we keep the same notation used in the aforementioned construction, right down to the random index-message pair (I, M).

We analyze the the result of the query computation. Firstly, $C' = M \cdot A^s = M \cdot e(g, v)^{\alpha s}$.

For the following computations, we recall an important property of bilinear maps over groups of composite order $n = pq$, namely that given $a_p \in G_p$ and $a_q \in G_q$, $e(a_p, a_q) = 1$.

Furthermore we make use of another important property of bilinear maps. Given their bilinearity, it easily follows that:

$$\forall a, b, c, d \in G\text{: } e(ab, cd) = e(a, c)\, e(a, d)\, e(b, c)\, e(b, d).$$

The property can be generalized for any number of factors on either of the entries as follows:

$$e(\prod_{i=1}^{m} a_i, \prod_{j=1}^{n} b_j) = \prod_{i=1}^{m} \prod_{j=1}^{n} e(a_i, b_j).$$

A final property to take into account is the fact that our bilinear map is defined as a symmetric bilinear map : $e : G \times G \to G_T$. Therefore, $\forall a, b \in G\text{: } e(a, b) = e(b, a)$.

We have $C_0 = V^s Z = v^s (R_v^s Z)$ and $K_0 = g^\alpha \prod_{i \in S} (u_i^{I_i} h_i)^{r_{i,1}} w_i^{r_{i,2}} \overset{\text{not.}}{=} g^\alpha \pi \in G_p$.

Therefore:

$$e(C_0, K_0) = e(v^s (R^s Z), g^\alpha \pi) = e(v^s, g^\alpha)\, e(v^s, \pi)\, e(R^s Z, g^\alpha \pi) = e(v, g)^{\alpha s}\, e(v^s, \pi)$$

We also have $C_{i,1} = (U_i^{I_i} H_i)^s Z_{i,1} = (u_i^{I_i} h_i)^s ((R_{u,i}^{I_i} R_{h,i})^s Z_{i,1})$, where the first factor is in G_p and the rest in G_q and $K_{i,1} = v^{r_{i,1}} \in G_p$. Thus:

$$e(C_{i,1}, K_{i,1}) = e((u_i^{I_i} h_i)^s, v^{r_{i,1}}) = e((u_i^{I_i} h_i)^{r_{i,1}}, v^s) = e(v^s, (u_i^{I_i} h_i)^{r_{i,1}})$$

Lastly, $C_{i,2} = W_i^s Z_{i,2} = w_i^s (R_{w,i}^s Z_{i,2})$ and $K_{i,2} = v^{r_{i,2}}$. Therefore:

$$e(C_{i,2}, K_{i,2}) = e(w_i^s, v^{r_{i,2}}) = e(w_i^{r_{i,2}}, v^s) = e(v^s, w_i^{r_{i,2}})$$

Given the product in the *Query* computation, we are now able to analyze query correctness. A correct query is obtained when the attribute vector satisfies the query predicate. In our case, for each entry that is not the wildcard symbol, the predicate and the attributes must match. Given that the computation only uses indexes $i \in S$, a mismatch would only happen on those indexes if the attribute vector does not satisfy the predicate. Such a mismatch leads to erronous computation and decrypting, which, given our choice regarding the message space $\mathcal{M}$, grants an almost certain result $M' \notin \mathcal{M}$, thus leading to the output $\perp$.

We now analyze a correct query.

$$\prod_{i \in S} e(C_{i,1}, K_{i,1})\, e(C_{i,2}, K_{i,2}) = \prod_{i \in S} e(v^s, (u_i^{I_i} h_i)^{r_{i,1}})\, e(v^s, w_i^{r_{i,2}}) =$$
$$\prod_{i \in S} e(v^s, (u_i^{I_i} h_i)^{r_{i,1}} w_i^{r_{i,2}}) = e(v^s, \prod_{i \in S} (u_i^{I_i} h_i)^{r_{i,1}} w_i^{r_{i,2}}) = e(v^s, \pi)$$

The final computation is :

$$C' \Big/ \left(e(C_0, K_0) / e(v^s, \pi) \right) = M\, e(v,g)^{\alpha s} \Big/ \left(e(v,g)^{\alpha s}\, e(v^s, \pi) / e(v^s, \pi) \right) =$$
$$M\, e(v,g)^{\alpha s} / e(v,g)^{\alpha s} = M$$

As previously stated, a mismatch would lead to the first product not being equal to $e(v^s, \pi)$, thus leading to an incorrect result.

Comparative queries. We now present the HVE implementation of the advanced query types, starting with comparative queries.

Given the comparative statement $x \geq a$ and knowing a predetermined value range for x, $[0, l]$, we describe the structure of the attribute subvector and predicate generated.

The attribute subvector $s = (s_1, s_2, ..., s_l)$ associated with the statement will be :

$$s_i = \begin{cases} 1, & \text{if } i \geq a, \\ 0, & \text{otherwise} \end{cases}$$

The associated predicate description $p = (p_1, p_2, ..., p_l)$ will have the following form:

$$p_i = \begin{cases} 1, & \text{if } i = a, \\ \star, & \text{otherwise} \end{cases}$$

Thus, the predicate will be satisfied by attributes with entry 1 on the specified index a, those being the attributes that, given their previously presented structure, will satisfy the statement $x \geq a$.

Similarly, we can compose query elements for the $x \leq a$ statements.

The attribute subvector $s = (s_1, s_2, ..., s_l)$ associated with the statement will be :

$$s_i = \begin{cases} 1, & \text{if } i > a, \\ 0, & \text{otherwise} \end{cases}$$

The associated predicate description $p = (p_1, p_2, ..., p_l)$ will have the following form:

$$p_i = \begin{cases} 0, & \text{if } i = a, \\ \star, & \text{otherwise} \end{cases}$$

The predicate will be satisfied by elements with entry 0 on the specified position, hence our attributes respecting the statement $x \leq a$.

Range queries. Range queries can be viewed as composite queries, or in clearer terms, the merging of two queries for statements $x \geq a_{lower}$ and $x \leq a_{upper}$ respectively.

Conjunctive queries. By picturing the attribute vector as a whole, specifically as a conjunction of queries, we can easily structure the attribute vector as the merging of subvectors in a previously determined order.

Similarly, the query predicate can be structured as the merging of subpredicates, having the same order mentioned earlier.

Security proof. We formulate a security game for this construction. Let $\mathcal{A}$ be the adversary.

For this security game, the adversary starts by committing to two index vectors $L_0, L_1 \in \Sigma^l$. We name X the set of indexes i such that $L_{0,i} = L_{1,i}$ and $\bar{X}$ the set of indexes i such that $L_{0,i} \neq L_{1,i}$.

The adversary can also issue descriptions for predicate $L \in \Sigma_\star^l$. We name S the set of indexes i where $L_i \neq \star$.

We first distinguish between multiple 3 query types:

Type 1. If $S \cap \bar{X} = \emptyset$. The predicate therefore will not check any of the indexes where the vectors differ. Since the predicate cannot differentiate between them, any difference in the associated messages will give their associated vector away. This means that for this type of query, $M_0 = M_1$, where the messages in questions are given in the challenge phase.

Type 2. When the query is not type 1 and there is an index $i \in S \cap \bar{X}$ such that $L_{0,i} \neq L_i \neq L_{1,i}$. That is, there is an index such that neither of the two entries of the vectors match the entry of the predicate.

Type 3. When the query is neither type 1, nor type 2 and there is an index $i \in S \cap \bar{X}$ such that $L_i \neq L_{1,i} = L_{0,i}$. That is, there is an index such that the two vectors share a common value for the entry, yet the predicate entry does not match said value.

The three types are mutually exclusive and form a complete partition of all possible cases.

The security game is placed in the context of bilinear DDH. Given 3 random exponents $a, b, c \in \mathbb{Z}_n$, we analyze whether the adversary can distinguish $e(g_p, g_p)^{abc}$ from a random element of G_T.

We now present the security game:

1. **Init**. As previously mentioned, adversary $\mathcal{A}$ commits to vectors L_0, L_1 and presents them to the challenger. The challenger secretly picks one, $\beta \in \{0, 1\}$.

2. **Setup**. The challenger first generates the following random elements:

$$(R_{u,1}, R_{h,1}, R_{w,1}), (R_{u,2}, R_{h,2}, R_{w,2}), ..., (R_{u,l}, R_{h,l}, R_{w,l}) \in G_q,$$
$$R_v \in G_q,$$
$$(t_i, x_i, y_i)_{i \in \overline{1,l}} \in \mathbb{Z}_n^3$$

   Then the challenger publishes the group description of G and g_q, $V - g_p R_v$ and $A = e(g_p^a, g_p^b)$. It then creates for all $i \in \overline{1, l}$:

$$U_i = (g_p^b)^{t_i} R_{u,i}, \quad H_i = (g_p^b)^{-t_i L_{\beta,i}} g_p^{y_i} R_{h,i}, \quad W_i = g_p^{x_i} R_{w,i}$$

3. **Query Phase 1**. The adversary makes predicate queries to the challenger. They are handled in the manner appropriate to their type.

   Type 1. A type 1 query implies that $M_0 = M_1$. Therefore, when receiving a type 1 query, the challenger aborts it and takes a random guess.

   Type 2 and 3. For these types, the query can be used to tell the ciphertexts apart. We use the notation previously presented for the predicate entry L_j that has a different value from vector entry $L_{\beta,j}$.
   The challenger chooses random elements $(r_{i,1}, r_{i,2}) \in \mathbb{Z}_n$, for all $i \in S$. Next it computes:

$$K_0 = \prod_{i \in S} ((g_p^b)^{r_{i,1}(L_i - L_{\beta,i})} g_p^{r_{i,1} y_i} g_p^{r_{i,2} x_i})$$

   Additionally, for all $i \in S$, $i \neq j$:

$$K_{i,1} = (g_p^a)^{r_{i,1}}, \, K_{i,2} = (g_p^a)^{r_{i,2}}$$

   It then creates:

$$K_{j,1} = g_p^{r_{j,1}} (g_p^a)^{\frac{-1}{L_j - L_{\beta,j}}}, \, K_{j,2} = g_p^{r_{j,2}}$$

4. **Challenge**. The adversary gives the challenger the messages M_0, M_1. If they are identical, the challenger takes a random guess.

   The challenger generates the following random elements:

$$Z \in G_q, \quad (Z_{i,1}, Z_{i,2})_{i \in \overline{1,l}} \in G_q^2$$

The challenge is outputted as follows:

$$C' = M_\beta T' C_0 = g^c Z,$$
$$\forall i \in \overline{1,l}\colon\ C_{i,1} = (g^c)^{y_i} Z_{i,1},\ C_{i,2} = (g^c)^{x_i} Z_{i,2}$$

5. **Query Phase 2**. Just like the query security game, Query Phase 2 is exactly the same as Query Phase 1.
6. **Guess**. The adversary outputs a guess $\beta' \in \{0,1\}$. The challenger outputs 1 if $\beta' = \beta$ and 0 otherwise.

The advantage computations are the same as the ones for the Bilinear Diffie-Hellman game.

4.3.2 Iovino-Persiano construction

We now turn to an alternate implementation of Hidden Vector Encryption. In 2008, Vincenzo Iovino and Giuseppe Persiano adapted the previous construction and utilized groups of prime order for their construction.

Bilinear maps over groups of prime order no longer have the properties presented in the previous section, as no subgroups exist. However, the remarkable property they have is that each element in the group is a generator; that is, the order of each element has the same value as the order of the group. From this, we can also easily deduce that, given any two elements a, b of the group, we will be able to compute the discrete logarithm, both for a in regards to base b and for b in regards to base a.

The construction closely follows the original. The query predicate generates tokens which, if satisfied by the attribute vector, generate a structure which annuls the encryption.

The system will be composed of 4 algorithms : *Setup*, *Encrypt*, *GenToken* and *Query*.

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates a large prime number p. It then generates group G of order p. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$.

   Afterwards, it randomly generates the following elements:

   $$y \in \mathbb{Z}_p$$
   $$\forall\, 1 \le i \le n\ : t_i, v_i, r_i, m_i \in \mathbb{Z}_p$$

   These all compose the masterkey **MsK**:

   $$MsK = \big(y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}}\big)$$

   It then generates the following values:

$$Y = e(g,g)^y$$
$$\forall\, 1 \le i \le n \;:$$
$$T_i = g^{t_i}$$
$$V_i = g^{v_i}$$
$$R_i = g^{r_i}$$
$$M_i = g^{m_i}$$

It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = \big(\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}\big))$$

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_p$ and random $s_1, s_2, ..., s_n \in \mathbb{Z}_p$.

   (b) Compute the following cypertext elements:

$$\Omega = M \cdot Y^{-s}$$
$$C_0 = g^s$$
$$\forall\, 1 \le i \le n \;:$$
$$X_i = \begin{cases} T_i^{s-s_i}, & \text{if } I_i = 1 \\ R_i^{s-s_i}, & \text{if } I_i = 0 \end{cases}$$
$$W_i = \begin{cases} V_i^{s_i}, & \text{if } I_i = 1 \\ M_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \le i \le l$ such that $I_i \ne \star$.

   If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$K_0 = g^y$$

   Otherwise, for each element i of S, we will randomly generate value a_i, such that in the end, we have:

$$\sum_{i \in S} a_i = y$$

   For each of these generated elements, we compute:

$$Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases},$$

$$L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases}$$

We output TK as either K_0 or $(Y_i, L_i)_{i\in\overline{1,n}}$, depending on which of the aforementioned cases is at hand.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

   (a) $M \leftarrow \Omega \cdot e(C_0, K_0)$

   (b) $M \leftarrow \Omega \cdot \prod_{i\in S} e(X_i, Y_i)e(W_i, L_i)$

   M will be correct if the attribute vector satisfies the predicate in question.

Proof. For our proof of query correctness, we keep the same notation used in the aforementioned construction, right down to the random index-message pair (I, M).

For case (a), we have :

$$\Omega \cdot e(C_0, K_0) = M \cdot e(g, g)^{-ys} \cdot e(g, g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector, we have :

$$\Omega \cdot \prod_{i\in S} e(X_i, Y_i)e(W_i, L_i) = M \cdot e(g, g)^{-ys} \cdot \prod_{i\in S} e(g, g)^{(s-s_i)a_i} \cdot e(g, g)^{s_i a_i} =$$

$$M \cdot e(g, g)^{-ys} \cdot \prod_{i\in S} e(g, g)^{sa_i} = M \cdot e(g, g)^{-ys} \cdot e(g, g)^{s\cdot \sum_{i\in S} a_i} = M \cdot e(g, g)^{-ys} \cdot e(g, g)^{ys} = M$$

In our computation, regardless of whether $I_i = 1$ or $I_i = 0$, we reached:

$$(s - s_i) \cdot t_i \cdot \tfrac{a_i}{t_i} = (s - s_i) \cdot r_i \cdot \tfrac{a_i}{r_i} = (s - s_i) \cdot a_i$$
$$s_i \cdot v_i \cdot \tfrac{a_i}{v_i} = s_i \cdot m_i \cdot \tfrac{a_i}{m_i} = s_i \cdot a_i$$

Therefore, given our previous assumption of a satisfied predicate, we chose to generalize the equation given the identical partial results that were to be reached. Taking all these into account, we prove query correctness.

If the predicate is not satisfied, then there must be at least one index entry i where computation lead awry. In other terms, one of t_i, r_i and one of v_i, m_i is not properly annuled by its inverse. Thus, it leads to incorrect computation and decryption and message M is not correctly deciphered. As a result, this enforces query correctness, as incorrect queries will not reveal the message.

Security Proof. We formulate a security game. Assume BDH holds. Assume $\mathcal{A}$ is an adversary with a non-negligible advantage (larger than 1/2) for the semantic security algorithm. We prove that this contravenes BDH. Let $A = g^a$, $B = g^b$ and $C = g^c$ random elements of G.

The challenger, for whom BDH holds, receives as input the instance parameters $\mathcal{I}$, as well as $[A, B, C, Z]$, where Z is either $e(g, g)^{abc}$ or a random element of G_T.

The security game is structured as follows:

1. **Init**. The challenger runs $\mathcal{A}$ and receives attribute string x for the challenge.

2. **Setup**. The challenger computes $Y = e(A, B)$ and then generates the random values $t_i, v_i, r_i, m_i \in \mathbb{Z}_p$ and then computes:

$$T_i = \begin{cases} g^{t_i}, & \text{if } x_i = 1 \\ B^{t_i}, & \text{if } x_i = 0 \end{cases} \qquad V_i = \begin{cases} g^{v_i}, & \text{if } x_i = 1 \\ B^{v_i}, & \text{if } x_i = 0 \end{cases}$$
$$R_i = \begin{cases} B^{r_i}, & \text{if } x_i = 1 \\ g^{r_i}, & \text{if } x_i = 0 \end{cases} \qquad M_i = \begin{cases} B^{m_i}, & \text{if } x_i = 1 \\ g^{m_i}, & \text{if } x_i = 0 \end{cases}$$

   Given that $[\mathcal{I}, Y, (T_i, R_i, V_i, M_i)_{1 \leq i \leq n}]$ acts as the public key of the system, the challenger runs $\mathcal{A}$ on it.

3. **Query Phase 1**. The adversary formulates queries for predicates with description y such that x does not satisfy any.

   For one of these predicates (we take its generalized name, y), we select an entry index j such that $x_j \neq y_j \neq \star$. Given our previous assumption of nonsatisfiability, it naturally follows that there will always be such an entry.

   For every non-$\star$ entry index i in y (looking back at the construction, their set is set S), if $i \neq j$, we generate a random value $a'_i \in \mathbb{Z}_p$. We denote $a' = \sum a'_i$.

   Let $Y_j = \begin{cases} A^{\frac{1}{t_j}} g^{\frac{-a'}{t_j}}, & \text{if } y_j = 1 \\ A^{\frac{1}{r_j}} g^{\frac{-a'}{r_j}}, & \text{if } y_j = 0 \end{cases}$ and $L_j = \begin{cases} A^{\frac{1}{v_j}} g^{\frac{-a'}{v_j}}, & \text{if } y_j = 1 \\ A^{\frac{1}{m_j}} g^{\frac{-a'}{m_j}}, & \text{if } y_j = 0 \end{cases}$.

For all other elements of S (elements $i \neq j$), we construct the following:

$$Y_i = \begin{cases} B^{\frac{a'_i}{t_i}}, & \text{if } x_i = y_i = 1 \\ B^{\frac{a'_i}{r_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{t_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{r_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ \emptyset, & \text{otherwise} \end{cases} \quad \text{and } L_i = \begin{cases} B^{\frac{a'_i}{v_i}}, & \text{if } x_i = y_i = 1 \\ B^{\frac{a'_i}{m_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{v_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{m_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ \emptyset, & \text{otherwise} \end{cases}$$

On further computation, we notice that, following the structure of the general construction, $a_i = b \cdot a'_i$ (for $i \in S$ and $i \neq j$) and $a_j = b \cdot (a - a')$. Summing them up, we have:

$$\sum_{i \in S} a_i = a \cdot b = y$$

4. **Challenge**. The adversary presents the challenger with two messages $M_0, M_1 \in G_T$. The challenger randomly selects one, M_β, $\beta \in \{0,1\}$. It then generates random values $s_i \in \mathbb{Z}_p$ and computes the following:

$$\Omega = M_\beta Z^{-1}, \; C_0 = C,$$

$$X_i = \begin{cases} C^{t_i} g^{-t_i s_i}, & \text{if } x_i = 1 \\ C^{r_i} g^{-r_i s_i}, & \text{if } x_i = 0 \end{cases}, \; W_i = \begin{cases} g^{-v_i s_i}, & \text{if } x_i = 1 \\ g^{-m_i s_i}, & \text{if } x_i = 0 \end{cases}$$

We observe that for $Z = e(g,g)^{abc}$, Ω coincides with the construction for $s = c$. If Z is a random element from G_T, Ω is independent from β (due to the randomness).

5. **Query Phase 2**. Query Phase 2 has the exact same stucture as Query Phase 1.

6. **Guess**. The adversary outputs guess $\beta' \in \{0,1\}$. The challenger outputs 1 if $\beta = \beta'$.

   If $Z = e(g,g)^{abc}$, given that $\mathcal{A}$ has a non-negligible advantage for the semantic security algorithm, the probability for the challenger to output 1 is non-negligibly larger than 1/2. For the random Z case, the very same probability is less than 1/2. This contradicts BDH and therefore proves security.

5 Our constructions

In this section, we will expand the previously presented construction in order to enable their functionality over multilinear maps. For starters, the second construction will be expanded for groups of prime order.

5.1 Multilinear Iovino-Persiano HVE over groups of prime order

The construction we propose closely resembles the Iovino-Persiano structure that inspired it, being an adaptation to the multilinear context maintaining the same algorithmic complexity.

The system will be composed of 4 algorithms : *Setup*, *Encrypt*, *GenToken* and *Query*.

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates a large prime number p. It then generates group G of order p. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$, where $e : G^{2n} \to G_T$ is our multilinear map function.

   Afterwards, it randomly generates the following elements:

   $$y \in \mathbb{Z}_p$$
   $$\forall\, 1 \le i \le n \ : t_i, v_i, r_i, m_i \in \mathbb{Z}_p$$

   These all compose the masterkey **MsK**:

   $$MsK = \big(y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}}\big)$$

   It then generates the following values:

   $$\begin{gathered} Y = e(g, g, ..., g)^y \\ \forall\, 1 \le i \le n \ : \\ T_i = g^{t_i} \\ V_i = g^{v_i} \\ R_i = g^{r_i} \\ M_i = g^{m_i} \end{gathered}$$

   It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

   $$PK = \big(\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}\big))$$

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

(a) Select random $s \in \mathbb{Z}_p$ and random $s' \in \mathbb{Z}_p$.

(b) Compute the rows s_i and s'_i for $i \in \overline{1,n}$ such that :

$$\prod_{i \in \overline{1,n}} s_i = s - s'$$
$$\prod_{i \in \overline{1,n}} s'_i = s'$$

The fact that this computation is possible will be addressed later, for the row generated for the $GenToken$ function.

(c) Compute the following cypertext elements:

$$\Omega = M \cdot Y^{-s}$$
$$C_0 = g^s$$
$$\forall\, 1 \leq i \leq n \ :$$
$$X_i = \begin{cases} T_i^{s_i}, & \text{if } I_i = 1 \\ R_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$
$$W_i = \begin{cases} V_i^{s'_i}, & \text{if } I_i = 1 \\ M_i^{s'_i}, & \text{if } I_i = 0 \end{cases}$$

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \leq i \leq l$ such that $I_i \neq \star$.

If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$K_0 = g^y$$

Otherwise, for each element i of S, we will randomly generate value a_i, such that in the end, we have:

$$\prod_{i \in S} a_i = y$$

Given the prime order of group G, it follows that having computed the product of all but the last element, we will be able to invert said product and multiply it by y, thus obtaining what is needed of us. Computationally, this is of similar complexity to the original implementation: instead of randomly generating $|S|-1$ values in G and determining the last one by deducing the current value from y, we randomly generate $|S|-1$ values in G and given our group's properties, we invert the result and multiply it by y to obtain our result. Best implementations of modular inversing achieve it in logarithmic time.

For each of these generated elements, we compute:

$$Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases},$$

$$L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases}$$

We output TK as either K_0 or $(Y_i, L_i)_{i \in \overline{1,n}}$, depending on which of the aforementioned cases is at hand.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

   (a) $M \leftarrow \Omega \cdot e(C_0, K_0, g, g, ..., g)$

   (b) $M \leftarrow \Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n)$

   M will be correct if the attribute vector satisfies the predicate in question.

Proof. For case (a), we have :

$$\Omega \cdot e(C_0, K_0, g, g, ..., g) = M \cdot e(g, g, ..., g)^{-ys} \cdot e(g^s, g^y, ..., g) = \\ M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector and denoting the rows:

$$\alpha_i = \begin{cases} t_i, & \text{if } I_i = 1 \\ r_i, & \text{if } I_i = 0 \end{cases}$$

$$\beta_i = \begin{cases} v_i, & \text{if } \mathcal{I}_i = 1 \\ m_i, & \text{if } \mathcal{I}_i = 0 \end{cases}$$

We use this notation for ease of writing, as if the predicate is satisfied, the values, regardless of the attribute vector entry, will be reduced. Therefore:

$$\Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n) = \\ = \\ \Omega \cdot e(g^{\alpha_1 s_1}, g^{\alpha_2 s_2}, ..., g^{\alpha_n s_n}, g^{\frac{a_1}{\alpha_1}}, g^{\frac{a_2}{\alpha_2}}, ..., g^{\frac{a_n}{\alpha_n}}) \cdot e(g^{\beta_1 s'_1}, g^{\beta_2 s'_2}, ..., g^{\beta_n s'_n}, g^{\frac{a_1}{\beta_1}}, g^{\frac{a_2}{\beta_2}}, ..., g^{\frac{a_n}{\beta_n}}) = \\ = \Omega \cdot e(g, g, ..., g)^{\prod_{i \in \overline{1,n}} s_i a_i} \cdot e(g, g, ..., g)^{\prod_{i \in \overline{1,n}} s'_i a_i} = \\ = \Omega \cdot e(g, g, ..., g)^{(s-s')y} \cdot e(g, g, ..., g)^{s'y} = \Omega \cdot e(g, g, ..., g)^{sy} = \\ = M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

This proves query correctness.

If the predicate was not satisfied, much like in the original implementation, we'd have at least one erronous inverse, thus not leading to a correct decryption.

Security Proof. We formulate a security game. Assume MDH holds. Assume $\mathcal{A}$ is an adversary with a non-negligible advantage (larger than 1/2) for the semantic security algorithm. We prove that this contravenes MDH. Let $A_1 = g^{e_1}$, $A_2 = g^{e_2}$,..., $A_{n+1} = g^{e_{n+1}}$ random elements of G.

The challenger, for whom MDH holds, receives as input the instance parameters $\mathcal{I}$, as well as $[(A_i)_{i\in\overline{1,n}}, Z]$, where Z is either $e(g, g, ..., g)^{\prod_{i\in\overline{1,n+1}} e_i}$ or a random element of G_T.

The security game is structured as follows:

1. **Init**. The challenger runs $\mathcal{A}$ and receives attribute string x for the challenge.

2. **Setup**. The challenger computes $Y = e(A_1, A_2, ..., A_n)$ and then generates the random values $t_i, v_i, r_i, m_i \in \mathbb{Z}_p$ and then computes:

$$T_i = \begin{cases} g^{t_i}, & \text{if } x_i = 1 \\ A_i^{t_i}, & \text{if } x_i = 0 \end{cases} \qquad V_i = \begin{cases} g^{v_i}, & \text{if } x_i = 1 \\ A_i^{v_i}, & \text{if } x_i = 0 \end{cases}$$
$$R_i = \begin{cases} A_i^{r_i}, & \text{if } x_i = 1 \\ g^{r_i}, & \text{if } x_i = 0 \end{cases} \qquad M_i = \begin{cases} A_i^{m_i}, & \text{if } x_i = 1 \\ g^{m_i}, & \text{if } x_i = 0 \end{cases}$$

   Given that $[\mathcal{I}, Y, (T_i, R_i, V_i, M_i)_{1\le i\le n}]$ acts as the public key of the system, the challenger runs $\mathcal{A}$ on it.

3. **Query Phase 1**. The adversary formulates queries for predicates with description y such that x does not satisfy any.

   For one of these predicates (we take its generalized name, y), we select an entry index j such that $x_j \ne y_j \ne \star$. Given our previous assumption of nonsatisfiability, it naturally follows that there will always be such an entry.

   For every non-$\star$ entry index i in y (looking back at the construction, their set is set S), if $i \ne j$, we generate a random value a'_i. We denote $a' = \prod a'_i$.

   Let $Y_j = \begin{cases} (\prod_{i\notin S} A_i^{\frac{1}{t_j}}) A_j^{\frac{1}{t_j}} g^{\frac{(a')^{-1}}{t_j}}, & \text{if } y_j = 1 \\ (\prod_{i\notin S} A_i^{\frac{1}{r_j}}) A_j^{\frac{1}{r_j}} g^{\frac{(a')^{-1}}{r_j}}, & \text{if } y_j = 0 \end{cases}$ and

$$L_j = \begin{cases} (\prod_{i\notin S} A_i^{\frac{1}{v_j}}) A_j^{\frac{1}{v_j}} g^{\frac{(a')^{-1}}{v_j}}, & \text{if } y_j = 1 \\ (\prod_{i\notin S} A_i^{\frac{1}{m_j}}) A_j^{\frac{1}{m_j}} g^{\frac{(a')^{-1}}{m_j}}, & \text{if } y_j = 0 \end{cases}.$$

   For all other elements of S (elements $i \ne j$), we construct the following:

$$Y_i = \begin{cases} A_i^{\frac{a'_i}{t_i}}, & \text{if } x_i = y_i = 1 \\ A_i^{\frac{a'_i}{r_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{t_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{r_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ g, & \text{otherwise} \end{cases} \quad \text{and } L_i = \begin{cases} A_i^{\frac{a'_i}{v_i}}, & \text{if } x_i = y_i = 1 \\ A_i^{\frac{a'_i}{m_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{v_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{m_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ g, & \text{otherwise} \end{cases}$$

On further computation, we notice that, following the structure of the general construction, $a_i = e_i \cdot a'_i$ (for $i \in S$ and $i \neq j$) and $a_j = \prod_{i \notin S} e_i \cdot e_j \cdot (a')^{-1}$. Multiplying them, we have:

$$\prod_{i \in S} a_i = \prod_{i \in \overline{1,n}} e_i = y$$

4. **Challenge**. The adversary present the challenger with two messages $M_0, M_1 \in G_T$. The challenger randomly selects one, M_β, $\beta \in \{0,1\}$. It then generates random value $s' \in \mathbb{Z}_p$ and based on it, generates for each $i \in \overline{1,n}$ the values $s_i, s'_i \in \mathbb{Z}_p$ such that:

$$\prod_{i \in \overline{1,n}} s_i = e_{n+1} - s'$$
$$\prod_{i \in \overline{1,n}} s'_i = s'$$

and computes the following:

$$\Omega = M_\beta Z^{-1},\ C_0 = A_{n+1},$$

$$X_i = \begin{cases} g^{t_i s_i}, & \text{if } x_i = 1 \\ g^{r_i s_i}, & \text{if } x_i = 0 \end{cases},\ W_i = \begin{cases} g^{v_i s'_i}, & \text{if } x_i = 1 \\ g^{m_i s'_i}, & \text{if } x_i = 0 \end{cases}$$

We observe that for $Z = e(g, g, ..., g)^{\prod_{i \in \overline{1,n+1}} e_i}$, Ω coincides with the construction for $s = e_{n+1}$. If Z is a random element from G_T, Ω is independent from β (due to the randomness).

5. **Query Phase 2**. Query Phase 2 has the exact same stucture as Query Phase 1.

6. **Guess**. The adversary outputs guess $\beta' \in \{0,1\}$. The challenger outputs 1 if $\beta = \beta'$.

If $Z = e(g, g, ..., g)^{\prod_{i \in \overline{1,n+1}} e_i}$, given that $\mathcal{A}$ has a non-negligible advantage for the semantic security algorithm, the probability for the challenger to output 1 is non-negligibly larger than 1/2. For the random Z case, the very same probability is less than 1/2. This contradicts MDH and therefore proves security.

5.2 Iovino-Persiano HVE over groups of composite order

In this subsection we make minute changes to the aforementioned construction to grant it support for groups of composite order.

In these groups, some elements may not have a modular inverse. This is indeed the case for elements i in $\mathbb{Z}_N$, where n is composite, if $(i, N) \neq 1$. A simple fix for this is to consider only the elements of $\mathbb{Z}_N^\star$, the exact subset that excludes the troublesome elements we mentioned in the previous sentence. Hence, the construction is as follows.

The system will be composed of 4 algorithms : *Setup*, *Encrypt*, *GenToken* and *Query*.

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates $N = pq$, where p and q are large primes. It then generates group G of order N. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$.

   Afterwards, it randomly generates the following elements:

$$y \in \mathbb{Z}_N$$
$$\forall\, 1 \leq i \leq n\ : t_i, v_i, r_i, m_i \in \mathbb{Z}_N^\star$$

   These all compose the masterkey **MsK**:

$$MsK = \big(y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}}\big)$$

   It then generates the following values:

$$\begin{gathered} Y = e(g,g)^y \\ \forall\, 1 \leq i \leq n\ : \\ T_i = g^{t_i} \\ V_i = g^{v_i} \\ R_i = g^{r_i} \\ M_i = g^{m_i} \end{gathered}$$

   It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = \big(\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}\big))$$

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_N$ and random $s_1, s_2, ..., s_n \in \mathbb{Z}_N$.

   (b) Compute the following cypertext elements:

$$
\begin{gathered}
\Omega = M \cdot Y^{-s} \\
C_0 = g^s \\
\forall\, 1 \le i \le n \;: \\
X_i = \begin{cases} T_i^{s-s_i}, & \text{if } I_i = 1 \\ R_i^{s-s_i}, & \text{if } I_i = 0 \end{cases} \\
W_i = \begin{cases} V_i^{s_i}, & \text{if } I_i = 1 \\ M_i^{s_i}, & \text{if } I_i = 0 \end{cases}
\end{gathered}
$$

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_n) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \le i \le n$ such that $I_i \ne \star$.

   If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

   $$K_0 = g^y$$

   Otherwise, for each element i of S, we will randomly generate value $a_i \in \mathbb{Z}_N$, such that in the end, we have:

   $$\sum_{i \in S} a_i = y$$

   For each of these generated elements, we compute:

   $$Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases} ,$$

   $$L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases}$$

   We output TK as either K_0 or $(Y_i, L_i)_{i \in \overline{1,n}}$, depending on which of the aforementioned cases is at hand.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

   (a) $M \leftarrow \Omega \cdot e(C_0, K_0)$

   (b) $M \leftarrow \Omega \cdot \prod_{i \in S} e(X_i, Y_i) e(W_i, L_i)$

M will be correct if the attribute vector satisfies the predicate in question.

The construction is computationally correct, as all inverse elements in $\mathbb{Z}_N^\star$ are now computable.

Proof. The proof follows the exact same line of reason the one in subsection 3.3.2 does.

For case (a), we have :

$$\Omega \cdot e(C_0, K_0) = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector, we have :

$$\Omega \cdot \prod_{i \in S} e(X_i, Y_i) e(W_i, L_i) = M \cdot e(g,g)^{-ys} \cdot \prod_{i \in S} e(g,g)^{(s-s_i)a_i} \cdot e(g,g)^{s_i a_i} =$$
$$M \cdot e(g,g)^{-ys} \cdot \prod_{i \in S} e(g,g)^{s a_i} = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{s \cdot \sum_{i \in S} a_i} = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{ys} = M$$

Security Proof. We formulate a security game. Assume BDH holds. Assume $\mathcal{A}$ is an adversary with a non-negligible advantage (larger than 1/2) for the semantic security algorithm. We prove that this contravenes BDH. Let $A = g^a$, $B = g^b$ and $C = g^c$ random elements of G.

The challenger, for whom BDH holds, receives as input the instance parameters $\mathcal{I}$, as well as $[A, B, C, Z]$, where Z is either $e(g,g)^{abc}$ or a random element of G_T.

The security game is structured as follows:

1. **Init**. The challenger runs $\mathcal{A}$ and receives attribute string x for the challenge.

2. **Setup**. The challenger computes $Y = e(A, B)$ and then generates the random values $t_i, v_i, r_i, m_i \in \mathbb{Z}_N^\star$ and then computes:

$$T_i = \begin{cases} g^{t_i}, & \text{if } x_i = 1 \\ B^{t_i}, & \text{if } x_i = 0 \end{cases} \quad V_i = \begin{cases} g^{v_i}, & \text{if } x_i = 1 \\ B^{v_i}, & \text{if } x_i = 0 \end{cases}$$
$$R_i = \begin{cases} B^{r_i}, & \text{if } x_i = 1 \\ g^{r_i}, & \text{if } x_i = 0 \end{cases} \quad M_i = \begin{cases} B^{m_i}, & \text{if } x_i = 1 \\ g^{m_i}, & \text{if } x_i = 0 \end{cases}$$

   Given that $[\mathcal{I}, Y, (T_i, R_i, V_i, M_i)_{1 \le i \le n}]$ acts as the public key of the system, the challenger runs $\mathcal{A}$ on it.

3. **Query Phase 1**. The adversary formulates queries for predicates with description y such that x does not satisfy any.

For one of these predicates (we take its generalized name, y), we select an entry index j such that $x_j \neq y_j \neq \star$. Given our previous assumption of nonsatisfiability, it naturally follows that there will always be such an entry.

For every non-$\star$ entry index i in y (looking back at the construction, their set is set S), if $i \neq j$, we generate a random value $a'_i \in \mathbb{Z}_N$. We denote $a' = \sum a'_i$.

Let $Y_j = \begin{cases} A^{\frac{1}{t_j}} g^{\frac{-a'}{t_j}}, & \text{if } y_j = 1 \\ A^{\frac{1}{r_j}} g^{\frac{-a'}{r_j}}, & \text{if } y_j = 0 \end{cases}$ and $L_j = \begin{cases} A^{\frac{1}{v_j}} g^{\frac{-a'}{v_j}}, & \text{if } y_j = 1 \\ A^{\frac{1}{m_j}} g^{\frac{-a'}{m_j}}, & \text{if } y_j = 0 \end{cases}$.

For all other elements of S (elements $i \neq j$), we construct the following:

$$Y_i = \begin{cases} B^{\frac{a'_i}{t_i}}, & \text{if } x_i = y_i = 1 \\ B^{\frac{a'_i}{r_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{t_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{r_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ \emptyset, & \text{otherwise} \end{cases} \quad \text{and } L_i = \begin{cases} B^{\frac{a'_i}{v_i}}, & \text{if } x_i = y_i = 1 \\ B^{\frac{a'_i}{m_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{v_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{m_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ \emptyset, & \text{otherwise} \end{cases}$$

On further computation, we notice that, following the structure of the general construction, $a_i = a \cdot a'_i$ (for $i \in S$ and $i \neq j$) and $a_j = a \cdot (b - a')$. Summing them up, we have:

$$\sum_{i \in S} a_i = a \cdot b = y$$

4. **Challenge**. The adversary presents the challenger with two messages $M_0, M_1 \in G_T$. The challenger randomly selects one, M_β, $\beta \in \{0,1\}$. It then generates random values $s_i \in \mathbb{Z}_N$ and computes the following:

$$\Omega = M_\beta Z^{-1},\ C_0 = C,$$

$$X_i = \begin{cases} C^{t_i} g^{-t_i s_i}, & \text{if } x_i = 1 \\ C^{r_i} g^{-r_i s_i}, & \text{if } x_i = 0 \end{cases},\ W_i = \begin{cases} g^{-v_i s_i}, & \text{if } x_i = 1 \\ g^{-m_i s_i}, & \text{if } x_i = 0 \end{cases}$$

We observe that for $Z = e(g,g)^{abc}$, Ω coincides with the construction for $s = c$. If Z is a random element from G_T, Ω is independent from β (due to the randomness).

5. **Query Phase 2**. Query Phase 2 has the exact same stucture as Query Phase 1.

6. **Guess**. The adversary outputs guess $\beta' \in \{0,1\}$. The challenger outputs 1 if $\beta = \beta'$.

If $Z = e(g, g)^{abc}$, given that $\mathcal{A}$ has a non-negligible advantage for the semantic security algorithm, the probability for the challenger to output 1 is non-negligibly larger than 1/2. For the random Z case, the very same probability is less than 1/2. This contradicts BDH and therefore proves security.

5.3 Multilinear Iovino-Persiano HVE over groups of composite order

We now expand the construction from the previous subsection in order to enable it to support multilinear maps. A notable problem arises.

Just like in the previous construction, if we were to follow our multilinear construction over groups of prime order, we'd encounter a few more modular inverses that the bilinear case did not have. These too are fixed by restricting the exponent set to $Z_N^\star$.

The computations in cause are the row generations of s_i, s_i' and a_i. Each of these require a computable inverse for the final step. Therefore, if we restrict their entries to elements of $Z_N^\star$, correctness is guaranteed. As an added consequence to this, the final results, namely s, s' and y respectively, will also be elements of $Z_N^\star$, as they are the results of multiplications of elements of said group.

Therefore, we have the following construction.

The system will be composed of 4 algorithms : *Setup*, *Encrypt*, *GenToken* and *Query*.

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates $N = \prod_{i \in \overline{1,n}} p_i$, where all p_i are large prime factors. It then generates group G of order N. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$, where $e : G^{2n} \to G_T$ is our multilinear map function.

   Afterwards, it randomly generates the following elements:

$$y \in \mathbb{Z}_N^\star$$
$$\forall\, 1 \leq i \leq n \ : t_i, v_i, r_i, m_i \in \mathbb{Z}_N^\star$$

   These all compose the masterkey **MsK**:

$$MsK = \big(y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}}\big)$$

   It then generates the following values:

$$Y = e(g, g, ..., g)^y$$
$$\forall\, 1 \leq i \leq n \ :$$
$$T_i = g^{t_i}$$
$$V_i = g^{v_i}$$
$$R_i = g^{r_i}$$
$$M_i = g^{m_i}$$

It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = \big(\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}})\big)$$

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_N^\star$ and random $s' \in \mathbb{Z}_N^\star$.

   (b) Compute the rows s_i and s'_i in $\mathbb{Z}_N^\star$ for $i \in \overline{1,n}$ such that :

$$\prod_{i \in \overline{1,n}} s_i = s - s'$$
$$\prod_{i \in \overline{1,n}} s'_i = s'$$

   (c) Compute the following cypertext elements:

$$\Omega = M \cdot Y^{-s}$$
$$C_0 = g^s$$
$$\forall\, 1 \le i \le n\ :$$
$$X_i = \begin{cases} T_i^{s_i}, & \text{if } I_i = 1 \\ R_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$
$$W_i = \begin{cases} V_i^{s'_i}, & \text{if } I_i = 1 \\ M_i^{s'_i}, & \text{if } I_i = 0 \end{cases}$$

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \le i \le l$ such that $I_i \ne \star$.

   If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$K_0 = g^y$$

   Otherwise, for each element i of S, we will randomly generate value a_i in $\mathbb{Z}_N^\star$, such that in the end, we have:

$$\prod_{i \in S} a_i = y$$

   For each of these generated elements, we compute:

$$Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases},$$

$$L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases}$$

We output TK as either K_0 or $(Y_i, L_i)_{i\in\overline{1,n}}$, depending on which of the aforementioned cases is at hand.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

   (a) $M \leftarrow \Omega \cdot e(C_0, K_0, g, g, ..., g)$

   (b) $M \leftarrow \Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n)$

Proof. The proof follows the exact structure as the one in subsection 4.1.
For case (a), we have :

$$\Omega \cdot e(C_0, K_0, g, g, ..., g) = M \cdot e(g, g, ..., g)^{-ys} \cdot e(g^s, g^y, ..., g) = \\ M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector and denoting the rows:

$$\alpha_i = \begin{cases} t_i, & \text{if } I_i = 1 \\ r_i, & \text{if } I_i = 0 \end{cases}$$

$$\beta_i = \begin{cases} v_i, & \text{if } \mathcal{I}_i = 1 \\ m_i, & \text{if } \mathcal{I}_i = 0 \end{cases}$$

We use this notation for ease of writing, as if the predicate is satisfied, the values, regardless of the attribute vector entry, will be reduced. Therefore:

$$\Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n) = \\ = \\ \Omega \cdot e(g^{\alpha_1 s_1}, g^{\alpha_2 s_2}, ..., g^{\alpha_n s_n}, g^{\frac{a_1}{\alpha_1}}, g^{\frac{a_2}{\alpha_2}}, ..., g^{\frac{a_n}{\alpha_n}}) \cdot e(g^{\beta_1 s'_1}, g^{\beta_2 s'_2}, ..., g^{\beta_n s'_n}, g^{\frac{a_1}{\beta_1}}, g^{\frac{a_2}{\beta_2}}, ..., g^{\frac{a_n}{\beta_n}}) = \\ = \Omega \cdot e(g, g, ..., g)^{\prod_{i\in\overline{1,n}} s_i a_i} \cdot e(g, g, ..., g)^{\prod_{i\in\overline{1,n}} s'_i a_i} = \\ = \Omega \cdot e(g, g, ..., g)^{(s-s')y} \cdot e(g, g, ..., g)^{s'y} = \Omega \cdot e(g, g, ..., g)^{sy} = \\ = M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

Security Proof. We formulate a security game. Assume MDH holds. Assume $\mathcal{A}$ is an adversary with a non-negligible advantage (larger than 1/2) for the semantic security algorithm. We prove that this contravenes MDH. Let $A_1 = g^{e_1}$, $A_2 = g^{e_2}$,..., $A_{n+1} = g^{e_{n+1}}$ random elements of G. It is also to be noted that all $e_i \in \mathbb{Z}_N^\star$.

The challenger, for whom MDH holds, receives as input the instance parameters $\mathcal{I}$, as well as $[(A_i)_{i\in\overline{1,n}}, Z]$, where Z is either $e(g, g, ..., g)^{\prod_{i\in\overline{1,n+1}} e_i}$ or a random element of G_T.

The security game is structured as follows:

1. **Init**. The challenger runs $\mathcal{A}$ and receives attribute string x for the challenge.

2. **Setup**. The challenger computes $Y = e(A_1, A_2, ..., A_n)$ and then generates the random values $t_i, v_i, r_i, m_i \in \mathbb{Z}_N^\star$ and then computes:

$$T_i = \begin{cases} g^{t_i}, & \text{if } x_i = 1 \\ A_i^{t_i}, & \text{if } x_i = 0 \end{cases} \qquad V_i = \begin{cases} g^{v_i}, & \text{if } x_i = 1 \\ A_i^{v_i}, & \text{if } x_i = 0 \end{cases}$$
$$R_i = \begin{cases} A_i^{r_i}, & \text{if } x_i = 1 \\ g^{r_i}, & \text{if } x_i = 0 \end{cases} \qquad M_i = \begin{cases} A_i^{m_i}, & \text{if } x_i = 1 \\ g^{m_i}, & \text{if } x_i = 0 \end{cases}$$

   Given that $[\mathcal{I}, Y, (T_i, R_i, V_i, M_i)_{1\le i\le n}]$ acts as the public key of the system, the challenger runs $\mathcal{A}$ on it.

3. **Query Phase 1**. The adversary formulates queries for predicates with description y such that x does not satisfy any.

   For one of these predicates (we take its generalized name, y), we select an entry index j such that $x_j \neq y_j \neq \star$. Given our previous assumption of nonsatisfiability, it naturally follows that there will always be such an entry.

   For every non-$\star$ entry index i in y (looking back at the construction, their set is set S), if $i \neq j$, we generate a random value $a'_i \in \mathbb{Z}_N^\star$. We denote $a' = \prod a'_i$.

   Let $Y_j = \begin{cases} (\prod_{i\notin S} A_i^{\frac{1}{t_j}}) A_j^{\frac{1}{t_j}} g^{\frac{(a')^{-1}}{t_j}}, & \text{if } y_j = 1 \\ (\prod_{i\notin S} A_i^{\frac{1}{r_j}}) A_j^{\frac{1}{r_j}} g^{\frac{(a')^{-1}}{r_j}}, & \text{if } y_j = 0 \end{cases}$ and

   $L_j = \begin{cases} (\prod_{i\notin S} A_i^{\frac{1}{v_j}}) A_j^{\frac{1}{v_j}} g^{\frac{(a')^{-1}}{v_j}}, & \text{if } y_j = 1 \\ (\prod_{i\notin S} A_i^{\frac{1}{m_j}}) A_j^{\frac{1}{m_j}} g^{\frac{(a')^{-1}}{m_j}}, & \text{if } y_j = 0 \end{cases}$.

   For all other elements of S (elements $i \neq j$), we construct the following:

$$Y_i = \begin{cases} A_i^{\frac{a'_i}{t_i}}, & \text{if } x_i = y_i = 1 \\ A_i^{\frac{a'_i}{r_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{t_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{r_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ g, & \text{otherwise} \end{cases} \quad \text{and } L_i = \begin{cases} A_i^{\frac{a'_i}{v_i}}, & \text{if } x_i = y_i = 1 \\ A_i^{\frac{a'_i}{m_i}}, & \text{if } x_i = y_i = 0 \\ g^{\frac{a'_i}{v_i}}, & \text{if } x_i = 0 \text{ and } y_i = 1 \\ g^{\frac{a'_i}{m_i}}, & \text{if } x_i = 1 \text{ and } y_i = 0 \\ g, & \text{otherwise} \end{cases}$$

On further computation, we notice that, following the structure of the general construction, $a_i = e_i \cdot a'_i$ (for $i \in S$ and $i \neq j$) and $a_j = \prod_{i \notin S} e_i \cdot e_j \cdot (a')^{-1}$. Multiplying them, we have:

$$\prod_{i \in S} a_i = \prod_{i \in \overline{1,n}} e_i = y$$

4. **Challenge**. The adversary present the challenger with two messages $M_0, M_1 \in G_T$. The challenger randomly selects one, M_β, $\beta \in \{0,1\}$. It then generates random value $s' \in \mathbb{Z}_N^\star$ and based on it, generates for each $i \in \overline{1,n}$ the values $s_i, s'_i \in \mathbb{Z}_N^\star$ such that:

$$\prod_{i \in \overline{1,n}} s_i = e_{n+1} - s'$$
$$\prod_{i \in \overline{1,n}} s'_i = s'$$

and computes the following:

$$\Omega = M_\beta Z^{-1},\ C_0 = A_{n+1},$$

$$X_i = \begin{cases} g^{t_i s_i}, & \text{if } x_i = 1 \\ g^{r_i s_i}, & \text{if } x_i = 0 \end{cases},\ W_i = \begin{cases} g^{v_i s'_i}, & \text{if } x_i = 1 \\ g^{m_i s'_i}, & \text{if } x_i = 0 \end{cases}$$

We observe that for $Z = e(g,g,...,g)^{\prod_{i \in \overline{1,n+1}} e_i}$, Ω coincides with the construction for $s = e_{n+1}$. If Z is a random element from G_T, Ω is independent from β (due to the randomness).

5. **Query Phase 2**. Query Phase 2 has the exact same stucture as Query Phase 1.

6. **Guess**. The adversary outputs guess $\beta' \in \{0,1\}$. The challenger outputs 1 if $\beta = \beta'$.

If $Z = e(g,g,...,g)^{\prod_{i \in \overline{1,n+1}} e_i}$, given that $\mathcal{A}$ has a non-negligible advantage for the semantic security algorithm, the probability for the challenger to output 1 is non-negligibly larger than 1/2. For the random Z case, the very same probability is less than 1/2. This contradicts MDH and therefore proves security.

6 Feasible Multilinear Maps

Since the emergence of the open problem, many proposed multilinear maps were proven erronous or attackable.

We turn to presenting a proposed map that still benefits from not having any counterproof.

This category of multilinear maps are the multilinear maps with obfuscation.

6.1 Multilinear maps with obfuscation

In recent papers[1, 7, 8], the concepts of semantically secure multilinear maps and indistinguishability obfuscation (iO) have been shown to be equivalent. Therefore, if one could be obtained, so can the other.

The idea of iO was proposed in [2]. The idea was to have the program or algorithm "scramble" itself, while still maintaining the same functionality and efficiency. As such, given an **obfuscator** $\mathcal{O}$ and a program P, $\mathcal{O}(P)$ will work exactly as P, but will in some sense be "unintelligible" for the observer, acting as a "virtual black box". Therefore, an obfuscator can provide strong software security to the program.

Article [8] proposes a construction known as **Multilinear Jigsaw Puzzles**.

We follow with the definitions for indistinguishability obsfuscators and the construction in question.

Indistinguishability obfuscators. A uniform PPT machine iO is called an indistinguishability obfuscator for a circuit class $\mathcal{C}_\lambda$ (a set of programs) if the following conditions are satisfied:

1. For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$Pr[C'(x) = C(x) : iO(\lambda, C) \to C'] = 1$$

2. For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in C_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then

$$\big| Pr[D(iO(\lambda, C_0)) = 1] - Pr[D(iO(\lambda, C_1)) = 1] \big| \leq \alpha(\lambda)$$

Jigsaw Specifier. In the construction in question, the plaintext space is generated by the "Setup" part of the algorithm. Therefore, it is necessary to have an element to choose which elements of said space are to be chosen for encoding. This element is called a Jigsaw specifier. It takes as input the number of encoding levels k (which is equal to the multilinear size), the number of elements to be generated l and a probabilistic circuit A that, receiving the prime p, outputs p and the pairs (S_i, a_i), with $i \in \overline{1, l}$, $S_i \in \overline{1, k}$ the encoding level and $a_i \in \mathbb{Z}_p$:

$$(k, l, A)(p) \to (p, (S_i, a_i)_{i \in \overline{1,l}})$$

Multilinear Form. A multilinear form $\mathcal{F}$ is a tuple consisting of positive integer parameters k and l, a circuit π with l input wires and binary addition ($\oplus$), binary multiplication ($\otimes$), unary negation ($\ominus$) and unary "ignore" ($\square$) gates and F is an assignment of an index set $I \subseteq \overline{1,k}$ to each input wire of π satisfying the following properties:

1. For any $\oplus$-gate and $\ominus$-gate, the input and output wires of said gate are assigned the same set I.
2. For any $\otimes$-gate, if the two inputs are assigned sets I_1, I_2 and these sets are disjoint, the output wire of the gate is assigned the set $I_1 \cup I_2$.
3. For any $\square$-gate, the output wire is assigned the empty set ($\emptyset$).
4. The (final) output wire is assigned the set $\overline{1,k}$.

The multilinear property follows from the fact that for any multiplicative gate, the input sets must be disjoint. From this, it's easy to see that for any path from input to the output wire, each index in $\overline{1,k}$ can only be added once, as multiplication gates have disjoint inputs and all other gates do not add elements to the output set. Therefore, the output is viewed as multilinear in the inputs.

The multilinear form is used in this construction as an evaluation tool for the output of the jigsaw specifier. Given (k, l, A) a jigsaw specifier, $X = (p, (S_i, a_i)_{i \in \overline{1,l}})$ its output and $\mathcal{F} = (k', l', \pi, F)$ a multilinear form, if $k = k'$ and $l = l'$ and if π is assigned $S_1, S_2, ..., S_l$ to its input wires by F, we say that $\mathcal{F}$ is compatible with X and can evaluate it.

The **multilinear evaluation** is the output of circuit π and satisfies the following statements:

1. For every $\ominus$-gate with (S, a) as input, the output is $(S, -a)$. Here, $a, (-a) \in \mathbb{Z}_p$.
2. For every $\oplus$-gate with (S, a_1) and (S, a_2) as inputs, the output is $(S, a_1 + a_2)$. Here, $a_1, a_2, a_1 + a_2 \in \mathbb{Z}_p$.
3. For every $\otimes$-gate with (S_1, a_1) and (S_2, a_2) as inputs, with $S_1 \bigcap S_2 = \emptyset$, the output is $(S_1 \cup S_2, a_1 \cdot a_2)$. Here, $a_1, a_2, a_1 \cdot a_2 \in \mathbb{Z}_p$.

Multilinear Jigsaw Puzzle. Multilinear jigsaw puzzles are a simplified extension for the one of the more popular multilinear map constructions, GGH'13[7]. These structures provide considerably less public functionality, as the encodings are only done by the trusted who generated our instance, as well as not providing any public nontrivial encodings for 1 and 0. Whilst this makes the system narrower, it also increases security, as by its lesser public information, it avoids a known attack on the original construction, the "weak discrete logarithm" attack.

A multilinear jigsaw puzzle MJP is composed of 2 algorithms, jigsaw generator $JGen$ and jigsaw verifier $JVer$:

1. **JGen**. The algorithm takes a description of the plaintext elements and outputs their encodings, the so-called puzzle pieces. This is also composed of 2 subalgorithms, $JGen = (InstGen, Encode)$:

   (a) **InstGen**. The instance generator. Taking as input a security parameter λ and the multilinear size n, it generates a large prime $p > 2^{\lambda}$, public system parameters $prms$ and a secret state s :

   $$InstGen(\lambda, n) \rightarrow (p, prms, s)$$

   (b) **Encode**. The encoding algorithm. Takes as input the output of $InstGen$ and a pair (S, a) from the output of the associated jigsaw generator and outputs an encoding of a which is relative to the associated encoding level S:

   $$\forall i,\ 1 \leq i \leq l : Encode((p, prms, s), (S_i, a_i)) \rightarrow (S_i, u_i)$$

   The total output $(S_i, a_i)_{i \in \overline{1,l}}$, together with the system parameters $prms$ is considered our "puzzle":

   $$puzzle = (prms, (S_i, a_i)_{i \in \overline{1,l}})$$

2. **JVer**. The verification algorithm. It takes as input the $puzzle$ generated by $JGen$ and a multilinear form $\mathcal{F}$ and it uses the latter to verify the first. It outputs a boolean value, signifying whether the multilinear form accepts or rejects $puzzle$.

   Correctness. Using the notation $\mathcal{F}(X)$ for the output of the circuit π, the verification is considered correct if one of the following scenarios is reached:

   (a) $\mathcal{F}(X) = (\{\overline{1,k}\}, 0)$ and $JVer(puzzle, \mathcal{F}) = 1$

   (b) $\mathcal{F}(X) \neq (\{\overline{1,k}\}, 0)$ and $JVer(puzzle, \mathcal{F}) = 0$

   If none of the scenarios is reached, the verification is deemed incorrect.

The security of this construction is based on the indistinguishability of two different puzzles by any single multilinear form.

7 Conclusions

The search for a secure and efficient multilinear map is still a rather open problem. Yet through the new constructions presented in this paper, we intend to offer a functional adapted solution to the searchable encryption problem for both the current and future multilinear map constructions.

Given that HVE is the best known construction for searchable encryption, we believe that the new constructions, being HVE at heart, are likely to be some of the better solutions for multilinear searchable encryption problem.

One of the main functionality these expands, aside from just a reconstruction of the searchable encryption problem, is NIKE[6]. Given the theoretical proof that NIKE is solved by multilinear maps, all that is needed to validate the new constructions and put them to good use is an efficiently computable and secure multilinear map satisfying the DDH assumption.

References

[1] Martin R Albrecht, Pooya Farshim, Dennis Hofheinz, Enrique Larraia, and Kenneth G Paterson. Multilinear maps from obfuscation. In *Theory of Cryptography Conference*, pages 446–473. Springer, 2016.

[2] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001.

[3] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[4] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[5] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference*, pages 535–554. Springer, 2007.

[6] Eduarda SV Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G Paterson. Non-interactive key exchange. In *Public-Key Cryptography–PKC 2013*, pages 254–271. Springer, 2013.

[7] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–17. Springer, 2013.

[8] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[9] Vincenzo Iovino and Giuseppe Persiano. Hidden-vector encryption with groups of prime order. In *International Conference on Pairing-Based Cryptography*, pages 75–88. Springer, 2008.