2016

Edit distance based encryption and its application

Phuong Viet Xuan Tran
University of Wollongong, tvxp750@uowmail.edu.au

Guomin Yang
University of Wollongong, gyang@uow.edu.au

Willy Susilo
University of Wollongong, wsusilo@uow.edu.au

Kaitai Liang
Aalto University, kaitai.liang@aalto.fi

Publication Details

Phuong, T. Viet Xuan., Yang, G., Susilo, W. & Liang, K. (2016). Edit distance based encryption and its application. Lecture Notes in Computer Science, 9723 103-119. Melbourne, Australia Proceedings of The 21st Australasian Conference on Information Security and Privacy (ACISP)

Edit distance based encryption and its application

Abstract
Edit distance, also known as Levenshtein distance, is a very useful tool to measure the similarity between two strings. It has been widely used in many applications such as natural language processing and bioinformatics. In this paper, we introduce a new type of fuzzy public key encryption called Edit Distance-based Encryption (EDE). In EDE, the encryptor can specify an alphabet string and a threshold when encrypting a message, and a decryptor can obtain a decryption key generated from another alphabet string, and the decryption will be successful if and only if the edit distance between the two strings is within the pre-defined threshold. We provide a formal definition and security model for EDE, and propose an EDE scheme that can securely evaluate the edit distance between two strings embedded in the ciphertext and the secret key. We also show an interesting application of our EDE scheme named Fuzzy Broadcast Encryption which is very useful in a broadcasting network.

Keywords
edit, application, distance, encryption, its

Disciplines
Engineering | Science and Technology Studies

Publication Details
Phuong, T. Viet Xuan., Yang, G., Susilo, W. & Liang, K. (2016). Edit distance based encryption and its application. Lecture Notes in Computer Science, 9723 103-119. Melbourne, Australia Proceedings of The 21st Australasian Conference on Information Security and Privacy (ACISP)

Edit Distance Based Encryption and Its Application

Tran Viet Xuan Phuong[1], Guomin Yang[1], Willy Susilo[1], and Kaitai Liang[2]

[1] Centre for Computer and Information Security Research
School of Computing and Information Technology
University of Wollongong, Australia.
tvxp750@uowmail.edu.au, {gyang, wsusilo}@uow.edu.au
[2] Department of Computer Science, Aalto University, Finland.
kaitai.liang@aalto.fi

Abstract. Edit distance, also known as Levenshtein distance, is a very useful tool to measure the similarity between two strings. It has been widely used in many applications such as natural language processing and bioinformatics. In this paper, we introduce a new type of fuzzy public key encryption called Edit Distance-based Encryption (EDE). In EDE, the encryptor can specify an alphabet string and a threshold when encrypting a message, and a decryptor can obtain a decryption key generated from another alphabet string, and the decryption will be successful if and only if the edit distance between the two strings is within the pre-defined threshold. We provide a formal definition and security model for EDE, and propose an EDE scheme that can securely evaluate the edit distance between two strings embedded in the ciphertext and the secret key. We also show an interesting application of our EDE scheme named Fuzzy Broadcast Encryption which is very useful in a broadcasting network.

Keywords: Edit Distance, Fuzzy Encryption, Dynamic Programming, Viète's Formulas

1 Introduction

Measuring the similarity between two strings is an important task in many applications such as natural language processing, bio-informatics, and data mining. One of the common similarity metrics that has been widely used in the above applications is the Edit Distance (a.k.a. Levenshtein distance), which counts the minimum number of operations (namely, insertion, deletion, and substitution) required to transform one string into the other. In this paper, we investigate a challenging problem of building fuzzy public key encryption schemes based on edit distance.

Our work is motivated by an open problem raised by Sahai and Waters in [21], where the notion of Fuzzy Identity-Based Encryption (IBE) was proposed. The Fuzzy IBE scheme introduced in [21] can be regarded as the first Attribute-Based Encryption (ABE) scheme with a threshold access policy. To be more precise, it allows to use a private key corresponding to an identity string I' to decrypt a ciphertext encrypted with another identity string I if and only if the "set overlap" between I and I' (i.e., $|I \cap I'|$) is larger than a pre-defined threshold. One of the open problems raised in [21] is to construct fuzzy encryption schemes based on other similarity metrics.

We should note that edit distance is very different from the "set overlap" distance used in Fuzzy IBE. For example, consider the biometric identity application of Fuzzy IBE described in [21], given two strings $I =$ "ATCG" and $I' =$ "GACT", we have $|I \cap I'| = 4$ (i.e., the distance is 0). However, the edit distance between I and I' is 3. It is easy to see that the order of the alphabets in those strings will affect the edit distance, but not the set overlap distance. This simple example shows that to a certain extent edit distance provides better accuracy than the set overlap distance in measuring the similarity of two strings. As another example, given an encryption string I = "admirer" and a threshold distance $d = 1$, for edit distance, we can allow a decryption key associated with I' = "admirers" to decrypt the message; while for set overlap distance, we can have some totally unrelated anagrams of I, such as I' = "married", whose corresponding secret key can also decrypt the message. Due to the difference between the two distances (or similarity metrics), we cannot easily extend the technique used in [21] to construct a fuzzy encryption scheme for edit distance. Also, in order to distinguish our fuzzy encryption scheme based on edit distance from the Fuzzy IBE proposed in [21], we name our new encryption scheme Edit Distance-based Encryption (or EDE, for short).

1.1 This Work

In this paper, we introduce the notion of Edit Distance-based Encryption (EDE), formalize its security, and propose a practical scheme in the standard model.

Edit distance can be measured in polynomial time using different techniques, such as dynamic programming or recursion. However, in an EDE scheme, the two strings I and I' are embedded in the ciphertext CT and the user secret key SK, respectively. Hence, the problem becomes how to measure the distance of I and I' using CT and SK. We observe that the most important operation in the edit distance algorithms is the equality test between two alphabets $I[x]$ and $I'[y]$. Based on this observation, our proposed EDE scheme uses bilinear map [6] to solve this issue. We illustrate our idea using the following example.

Suppose we have two strings I = "ATTGA" and I' = "AGTA". We first encode each alphabet as a group element. Then in the encryption process, we create a randomized vector $\boldsymbol{I} = (A^s, T^s, T^s, G^s, A^s)$ using the same random number s. Similarly, we create another randomized vector $\boldsymbol{I'} = (A^r, G^r, T^r, A^r)$ in the key generation process. Then we apply bilinear map to conduct equality test between I and I' using the two vectors $\boldsymbol{I}$ and $\boldsymbol{I'}$ which are included in the ciphertext and the secret key respectively. The crux of the idea is illustrated in Figure 1. In order to deal with the threshold problem, we apply the technique of Viète's formulas [22] to solve the problem. In the encryption process, we create a vector $\boldsymbol{d} = (1, 2, \ldots, d, 0, \ldots, 0)$ for the threshold distance d and embed the vector $\boldsymbol{d}$ in the ciphertext. Also, based on the edit distance d' between I and I', we create another vector $\boldsymbol{d'} = (1, 2, \ldots, d', *, \ldots, *)$ where $*$ denotes the wildcard (i.e., don't care) symbol. Then based on $\boldsymbol{d}$ and $\boldsymbol{d'}$, we ensure that the decryption can be successful if and only if $d' \leq d$. Also, we overcome the issue of malleability by using the composite order group in constructing the EDE scheme. We prove that our proposed scheme is selectively secure under the L-composite Decisional Diffie-Hellman (L-cDDH) assumption.

We also show an interesting application of our EDE scheme named Fuzzy Broadcast Encryption (FBE), which is very useful in broadcasting networks. An FBE scheme allows the encryptor (i.e., message sender) to specify a set of receiver identities during the encryption process, and a user can decrypt the message if and only if the *minimum* edit distance between his/her identity and all the identities chosen by the encryptor is below a threshold that is also specified by the encryptor during the encryption process.

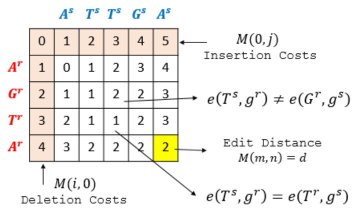
A^s T^s T^s G^s A^s
A^r G^r T^r A^r
$M(0,j)$ Insertion Costs
$e(T^s,g^r) \neq e(G^r,g^s)$
Edit Distance $M(m,n) = d$
$M(i,0)$ Deletion Costs
$e(T^s,g^r) = e(T^r,g^s)$

Fig. 1. Edit Distance Evaluation using Bilinear Map

1.2 Related Work

Since the seminal work of Sahai and Waters [21], many Attribute Based Encryption (ABE) schemes with the threshold access structure have been proposed (e.g., [9, 5, 11, 8]). In [9], Goyal et al. extended the work of Sahai and Waters to construct more expressive Key-Policy (KP) ABE where the access structure is defined via a tree of threshold gates. Bethencourt et al. [5] proposed the first Ciphertext-Policy (CP) ABE using the same access structure. Under the motivation of reducing the ciphertext size, which is linear in the size of the encryption attribute set in most of the existing ABE schemes, Herranz et al. [11] proposed a constant-size ABE scheme for the threshold access structure, which is essentially the same as the set overlap distance metric used in Fuzzy IBE [21]. In [8], Ge et al. proposed another constant-size ABE scheme with the same threshold access structure but under a relatively weaker assumption. As of independent interest, some interesting fuzzy encryption techniques have been proposed in the literature, such as [15, 17, 16].

Another type of fuzzy identity-based encryption is the Wildcarded IBE (or WIBE for short) proposed by Abdalla et al. [3, 1, 2]. A WIBE allows wildcard symbols to appear in an identity string used in the encryption process, and the wildcard positions will be ignored when measuring the equality of two identity strings. Another notion that is similar to WIBE is the Hidden Vector Encryption (HVE) [14, 12, 22, 18, 19], which also allows wildcards to appear in either the encryption string or the key generation string. However, both WIBE and HVE are based on the fuzzy equality test between two strings, which is different from the problem we aim to solve in this paper.

There are also a few works on the privacy-preserving edit distance evaluation between two strings [4, 13, 20, 7, 23]. These works mainly focused on finding the edit distance of two (perhaps encrypted) strings in a privacy-preserving manner, and hence is completely different from this work.

2 Preliminaries

2.1 Edit Distance

Consider a finite alphabet set $\mathcal{A}$ whose elements are used to construct strings. Let Z_I , Z_D and Z_S be finite sets of integers. Let the function $I : \mathcal{A} \to Z_I$ be the insertion cost function, i.e., $I(a)$ is the cost of inserting the element $a \in A$ into a given string. Similarly, define the deletion cost function as $D : \mathcal{A} \to Z_D$ so that $D(a)$ is the cost of deleting the element $a \in A$ from a given string. Finally, define the substitution cost function $S : \mathcal{A} \times \mathcal{A} \to Z_S$ so that for $a, b \in \mathcal{A}$, $S(a, b)$ is the cost of replacing the element a by the element b in a given string.

Given two strings of length m and n, denoted by $X \in \mathcal{A}^m$ and $Y \in \mathcal{A}^n$ respectively, consider the sequence of insertion, deletion and substitution operations needed to transform X into Y and the corresponding aggregate cost of the transformation.

Definition 1. *The edit distance between X and Y is defined as the minimum aggregate cost of transforming X into Y.*
The general definition of edit distance given above considers different weights for different operations. In this paper we will consider a simpler definition which is given below.
Definition 2. *For all $a, b \in \mathcal{A}$, let $I(a) = D(a) = 1$, $S(a, b) = 1$ when $a \neq b$, and $S(a, a) = 0$. Then, the edit distance is defined as the minimum number of insertion, deletion and substitution operations required to convert X into Y.*

Dynamic Programming for Edit Distance Let $X = X_1X_2...X_m \in \mathcal{A}^m$ and $Y = Y_1Y_2...Y_n \in \mathcal{A}^n$ be two strings. We use $M(i, j)$ to denote the edit distance between the two sub strings $X_1X_2...X_i$ and $Y_1Y_2...Y_j$. The problem of finding the edit distance between X and Y can be solved in $O(mn)$ time via dynamic programming [10], which will be used in our scheme.

Let $M(0, 0) = 0$. For $1 \leq i \leq m, 1 \leq j \leq n$, define $M(i, 0) = \sum_{k=1}^{i} I(x_k)$, and $M(0, j) = \sum_{k=1}^{j} D(y_k)$.

Then, the edit distance $M(m, n)$ is defined by the following recurrence relation for $1 \leq i \leq m, 1 \leq j \leq n$:

$$M(i,j) = \min \left\{ \begin{array}{l} M(i-1,j) + D(Y_j), \\ M(i,j-1) + I(X_i), \\ M(i-1,j-1) + S(X_i,Y_j), \end{array} \right\}.$$

2.2 The Viète's Formulas

Consider two vectors :$\overrightarrow{v} = (v_1, v_2, \ldots, v_L)$, $\overrightarrow{z} = (z_1, z_2, \ldots, z_L)$ where $\overrightarrow{v}$ contains both alphabets and wildcard symbols (*) and $\overrightarrow{z}$ only contains alphabets.
Let $J = \{j_1, \ldots, j_n\} \subset \{1, \ldots, L\}$ denote the wildcard positions in $\overrightarrow{v}$. Then according to [22], the statement ($v_i = z_i \vee v_i = *$ for $i = 1 \ldots L$) can be expressed as: $\sum_{i=1, i \notin J}^{L} v_i \prod_{j \in J} (i - j) = \sum_{i=1}^{L} z_i \prod_{j \in J} (i - j)(1)$.

Expand $\prod_{j \in J} (i - j) = \sum_{k=0}^{n} \lambda_k i^k$, where λ_k are the coefficients dependent on J, then (1) becomes: $\sum_{i=1, i \notin J}^{L} v_i \prod_{j \in J} (i - j) = \sum_{k=0}^{n} \lambda_k \sum_{i=1}^{L} z_i i^k (2)$.
To hide the computation, we choose random group element H_i and put v_i, z_i as the exponents of group elements: $H_i^{v_i}, H_i^{z_i}$. Then (2) becomes:

$$\prod_{i=1, i \notin J}^{L} H_i^{v_i \prod_{j \in J}(i-j)} = \prod_{k=0}^{n} (\prod_{i=1}^{L} H_i^{z_i i^k})^{\lambda_k}$$

Using the Viète's formulas we can construct the coefficient λ_k in (2) by:

$$\lambda_{n-k} = (-1)^k \sum_{1 \leq i_1 < i_2 < \ldots < i_k \leq n} j_{i_1} j_{i_2} \ldots j_{i_k}, \ 0 \leq k \leq n.$$

where $n = |J|$. For example, if we have $J = \{j_1, j_2, j_3\}$, the polynomial is $(x - j_1)(x - j_2)(x - j_3)$, then $\lambda_3 = 1, \lambda_2 = -(j_1 + j_2 + j_3), \lambda_1 = (j_1 j_2 + j_1 j_3 + j_2 j_3), \lambda_0 = -j_1 j_2 j_3$.

2.3 Bilinear Map on Composite Order Groups and Its Assumption

Let p, q be two large prime numbers and $n = pq$. Let $\mathbb{G}, \mathbb{G}_T$ be cyclic groups of order n. We say $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is bilinear map over composite order groups if e satisfies the following properties: (1) Bilinearity : $e(u^a, v^b) = e(u^b, v^a) = e(u, v)^{ab}$ for all u,$v \in \mathbb{G}$ and a,$b \in \mathbb{Z}_p$; (2)Non-degeneracy : $e(g, g) \neq 1$.

Let $\mathbb{G}_p$ and $\mathbb{G}_q$ be two subgroups of $\mathbb{G}$ of order p and q, respectively. Then $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q$, $\mathbb{G}_T = \mathbb{G}_{T,p} \times \mathbb{G}_{T,q}$. We use g_p and g_q to denote generators of $\mathbb{G}_p$ and $\mathbb{G}_q$, respectively. It is easy to see that $e(h_p, h_q) = 1$ for all elements $h_p \in \mathbb{G}_p$ and $h_q \in \mathbb{G}_q$ since $e(h_p, h_q) = e(g_p^a, g_q^b) = e(g^{qa}, g^{pb}) = e(g, g)^{pqab} = 1$ for a generator g of $\mathbb{G}$.

The Decisional $L-$cBDHE assumption:

$$\text{Let } g_p, h \xleftarrow{R} \mathbb{G}_p, g_q \xleftarrow{R} \mathbb{G}_q, \alpha \xleftarrow{R} \mathbb{Z}_n, Z = (g_p, g_q, h, g_p^{\alpha}, \ldots, g_p^{\alpha^L}, g_p^{\alpha^{L+2}}, \ldots, g_p^{\alpha^{2L}}),$$
$$T = e(g_p, h)^{\alpha^{L+1}}, \text{ and } R \leftarrow \mathbb{G}_{T,p}$$

We say that the decisional $L-$cBDHE assumption holds if for any probabilistic polynomial-time algorithm A:$|\Pr[A(Z, T) = 1] - \Pr[A(Z, R) = 1]| \leq \epsilon(k)$, where $\epsilon(k)$ denotes an negligible function of k.

3 Edit Distance Based Encryption

An Edit Distance Based Encryption (EDE) scheme consists of the following four probabilistic polynomial-time algorithms:

- **Setup**$(1^n, \Sigma)$: on input a security parameter 1^n, an alphabet Σ, the algorithm outputs a public key PK and a master secret key MSK.
- **Encrypt**$(PK, \overrightarrow{v}, M, d)$: on input a public key PK, a message M, a vector $\overrightarrow{v} \in \Sigma^n$ and a distance d, the algorithm outputs a ciphertext CT.
- **KeyGen**$(MSK, \overrightarrow{x})$: on input a master secret key MSK, a vector $\overrightarrow{x} \in \Sigma^m$, the algorithm outputs a decryption key SK.
- **Decrypt**(CT, SK): on input a ciphertext CT and a secret key SK, the algorithm outputs either a message M if $EditDistance(\overrightarrow{v}, \overrightarrow{x}) \leq d$, or a special symbol $\perp$.

Security Model. The security model for an EDE scheme is defined via the following game between an adversary A and a challenger B.

- **Setup**: The challenger B run **Setup**$(1^n, \Sigma)$ to generate the PK and MSK. PK is then passed to A.
- **Query Phase 1**: The challenger answers all private key queries for a vector $\overrightarrow{\sigma}$ by returning : $sk_\sigma \leftarrow KeyGen(MSK, \overrightarrow{\sigma})$.
- **Challenge**: A submits two equal-length messages M_0 and M_1, a target vector $\overrightarrow{v}^* \in \Sigma^n$ and threshold τ such that $EditDistance(\overrightarrow{v}^*, \overrightarrow{\sigma}) > \tau$ for any vector $\overrightarrow{\sigma}$ that has been queried in Phase 1. The challenger then flips a coin $\beta \leftarrow \{0, 1\}$ and computes the challenge ciphertext $C^* \leftarrow Encrypt(PK, \overrightarrow{v}^*, M_\beta, \tau)$, which is given to A.
- **Query Phase 2**: same as Query Phase 1 except that $EditDistance(\overrightarrow{v}^*, \overrightarrow{\sigma}) > \tau$ for any vector σ queried in this phase.
- **Output**: A outputs a bit β' as her guess for β.

Define the advantage of A as$\mathbf{Adv}_A^{\mathsf{EDE}}(k) = |\Pr[\beta' = \beta] - 1/2|$.

Selective Security. In the selective security model, the adversary A is required to submit the target vector $\overrightarrow{v}^* \in \Sigma^n$ and threshold τ before the game setup, and A is only allowed to make private key queries for any vector $\overrightarrow{\sigma}$ that satisfies $EditDistance(\overrightarrow{v}^*, \overrightarrow{\sigma}) > \tau$ throughout the game.

4 Edit Distance Based Encryption Scheme

In this section, we introduce our EDE scheme, which is based on the the Dynamic Programming [10] algorithm for calculating edit distance.

– **Setup**$(1^n, \Sigma)$: The setup algorithm first chooses $L = poly(n)$ as the maximum number of length of a word that would appear in the encryption and key generation. It then picks large primes p, q, generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of composite order $n = pq$, and selects generators $g_p \in \mathbb{G}_p, g_q \in \mathbb{G}_q$. After that, generate

$$\begin{aligned}
&v_0, v'_0, b_0, g, f, \omega, h_1, \ldots, h_L, u_1, \ldots, u_L \in_R \mathbb{G}_p, x_1, \ldots, x_L, x'_1, \ldots, x'_L \in_R \mathbb{Z}_n,\\
&v_1 = v_0^{x_1}, \ldots, v_L = v_0^{x_L}, v'_1 = (v'_0)^{x_1}, \ldots, v'_L = (v'_0)^{x_L}, b_1 = b_0^{x'_1}, \ldots, b_L = b_0^{x'_L},\\
&R_g, R_f, R_{v_0}, \ldots, R_{v_L}, R_{v'_0}, \ldots, R_{v'_L}, R_{b_0}, \ldots, R_{b_L}, R_{h_1}, \ldots, R_{h_L}, R_{u_1}, \ldots, R_{u_L} \in \mathbb{G}_q,\\
&G = gR_g, F = fR_f, Y = e(g, \omega),\\
&V_0 = v_0R_{v_0}, \ldots, V_L = v_LR_{v_L}, V'_0 = v'_0R_{v'_0}, \ldots, V'_L = v'_LR_{v'_L}, B_0 = b_0R_{b_0}, \ldots, B_L = b_LR_{b_L},\\
&H_1 = h_1R_{h_1}, \ldots, H_L = h_LR_{h_L}, U_1 = u_1R_{u_1}, \ldots, U_L = u_LR_{u_L},
\end{aligned}$$

and set the public key and secret key as:

$$\begin{aligned}
PK &= \{Y, G, F, (V_0, \ldots, V_L), (V'_0, \ldots, V'_L), (B_0, \ldots, B_L), (H_1, \ldots, H_L), (U_1, \ldots, U_L)\},\\
MSK &= \{g, f, \omega, (v_0, \ldots, v_L), (v'_0, \ldots, v'_L), (b_0, \ldots, b_L), (h_1, \ldots, h_L), (u_1, \ldots, u_L)\}.
\end{aligned}$$

– **Encrypt**$(PK, \overrightarrow{v} = (v_1, \ldots, v_{n_1}) \in \Sigma^{n_1}, M, d)$: On input the public key PK, a vector $\overrightarrow{v} = (v_1, \ldots, v_{n_1})$ with $n_1 \le L$, it first generates for each alphabet v_i a vector $\boldsymbol{x_i} = (v_i, 1, \ldots, 1_L)$, and expands $\overrightarrow{v}$ to $\overrightarrow{v} = (v_1, v_2, \ldots, v_{n_1}, \ldots, 1_L)$ and sets $\overrightarrow{d} = (1, \ldots, d, 0_{d+1}, \ldots, 0_L)$. Then choose $s \in_R \mathbb{Z}_n$, and $Z_1, Z_2, Z_3, Z_4, Z_5 \in_R \mathbb{G}_q$, and compute

$$\begin{aligned}
&C_0 = MY^s, C_1 = G^sZ_1, C_2 = F^sZ_2, C_{3,i} = (V_i \prod_{j=1}^{L} H_i^{x_{ij}})^s Z_3,\\
&C_4 = (V'_0 \prod_{i=1}^{L} H_i^{v_i})^s \cdot Z_4, C_{5,k,t} = (V'_t(B_k \prod_{i=1}^{L} (U_i)^{d_i i^k})(\prod_{j=1}^{L} (H_j)^{v_j j^t}))^s \cdot Z_5.
\end{aligned}$$

Set the ciphertext as: $CT = (n_1, C_0, C_1, C_2, \{C_{3,i}\}_{i=1}^{n_1}, C_4, \{\{C_{5,k,t}\}_{k=0}^{L}\}_{t=0}^{L})$.

– **KeyGen**$(MSK, \overrightarrow{z} = (z_1, \ldots, z_m) \in \Sigma^m)$: Given a key vector $\overrightarrow{z} = (z_1, \ldots, z_m)$, it generates $\boldsymbol{y_i} = (z_i, 1, \ldots, 1_L)$ for each alphabet z_i, and creates $\overrightarrow{\sigma} = (1, 2, ..., L)$ and expands $\overrightarrow{z}$ to $\overrightarrow{z} = (z_1, z_2, \ldots, z_m, \ldots, 1_L)$. Then choose $r_1, r_2 \in_R Z_n$, and compute

$$K_1 = g^{r_1}, K_2 = g^{r_2}, K_{3,i} = (v_i \prod_{j=1}^{L} h_i^{y_{ij}})^{r_2},$$

$$\begin{pmatrix} K_{4,0,0} = \omega(v'_0 \prod_{i=1}^{L} h_i^{z_i})^{r_2}((b_0 \prod_{i=1}^{L}(u_i^{\sigma_i}))(v'_0 \prod_{j=1}^{L} h_j^{z_j}))^{r_1} f^{r_1} \\ K_{4,1,0} = (b_1 \prod_{i=1}^{L}(u_i^{\sigma_i})^i (v'_0 \prod_{j=1}^{L} h_j^{z_j}))^{r_1} \\ \ldots, \\ K_{4,L,0} = (b_L \prod_{i=1}^{L}(u_i^{\sigma_i})^{i^L} (v'_0 \prod_{j=1}^{L} h_j^{z_j}))^{r_1} \end{pmatrix}, \begin{pmatrix} K_{4,0,t} = (b_0 \prod_{i=1}^{L}(u_i^{\sigma_i})(v'_t \prod_{j=1}^{L} h_j^{z_j})^{j^t})^{r_1} \\ K_{4,1,t} = (b_1 \prod_{i=1}^{L}(u_i^{\sigma_i})^i (v'_t \prod_{j=1}^{L} h_j^{z_j})^{j^t})^{r_1} \\ \ldots, \\ K_{4,L,t} = (b_L \prod_{i=1}^{L}(u_i^{\sigma_i})^{i^L} (v'_t \prod_{j=1}^{L} h_j^{z_j})^{j^t})^{r_1} \end{pmatrix}$$

, $(t = 1, \ldots, L)$. Then set the user secret key as $SK = (m, K_1, K_2, \{K_{3,i}\}_{i=1}^{m}, \{\{K_{4,k,t}\}_{k=0}^{L}\}_{t=0}^{L})$.

– **Decrypt**(CT, SK): The decryption algorithm first executes the dynamic programming algorithm for edit distance by following **Algorithm** 1 which returns a distance $d' = cost[len_v - 1]$, the matching indices array $pos[0][]$ for $\overrightarrow{v}$ and $pos[1][]$ for $\overrightarrow{z}$. It sets $\tau = L - d'$, and applies the Viète's formulas to compute

Algorithm 1: Edit distance evaluation via dynamic programming

```
input  : $CT, SK$
output: $d', pos$
$len_v = n + 1$;
$len_z = m + 1$;
Creat $cost[len_v]$;
Creat $newcost[len_v]$;
Creat $pos[2][]$  //setup two arrays to store the position matching pos[0][] for vector v, pos[1][] for vector z;
for $i \leftarrow 0$ to $len_v$ do
    $cost[i] = i$;
end
$k = 0$;
for $j \leftarrow 1$ to $len_z$ do
    $newcost[0] = j$;
    for $i \leftarrow 1$ to $len_v$ do
        // matching current letters in both strings
        $match = (e(K_2, C_{3,i-1}) == e(C_1, K_{3,j-1}))?0 : 1$ (1);
        // store the i match in array pos[0], j match in array pos[1]
        if $i \notin pos[0], j \notin pos[1]$ then
            $pos[0][k++] = i, pos[1][k++] = j$, ;
        end
        // computing cost for each transformation
        $replace = cost[i-1] + match$;
        $insert = cost[i] + 1$;
        $delete = newcost[i-1] + 1$;
        // keep minimum cost
        $newcost[i] = Math.min(Math.min(cost - insert, cost - delete), cost - replace)$;
    end
    // swap cost-newcost arrays
    $swap[] = cost$;
    $cost = newcost$;
    $newcost = swap$;
end
// return the cost for transforming all letters in both strings and array list pos including pos[0], pos[1]
return $cost[len_v - 1], pos$;
```

- for the index set $\Omega_v = \{L \backslash \{pos[0][0], \ldots, pos[0][d'-1]\}\} = \{\omega_1, \ldots, \omega_{L-d'}\}$

$$a_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} \omega_{i_1}\omega_{i_2}\ldots\omega_{i_k} \quad (0 \le k \le \tau),$$

- for the index set $\Omega_z = \{L \backslash \{pos[1][0], \ldots, pos[1][d'-1]\}\} = \{\bar{\omega}_1, \ldots, \bar{\omega}_{L-d'}\}$

$$\bar{a}_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} \bar{\omega}_{i_1} \bar{\omega}_{i_2} \ldots \bar{\omega}_{i_k} \quad (0 \le k \le \tau),$$

- for the threshold index set $J = \{j_1, \ldots, j_\tau\}$ with $j_1 = d'+1, \ldots, j_\tau = L$

$$\hat{a}_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} j_{i_1} j_{i_2} \ldots j_{i_k} \quad (0 \le k \le \tau).$$

Then recover M as:

$$M = \frac{e(K_2, C_4) \cdot e(K_1, C_2) \prod_{k=0}^{\tau} e(K_1^{\frac{1}{\bar{a}_0 \hat{a}_0}}, \prod_{t=0}^{\tau} C_{5,k,t}^{a_t})^{\hat{a}_k}}{\prod_{k=0}^{\tau} e(\prod_{t=0}^{\tau} K_{4,k,t}^{\bar{a}_t}, C_1^{\frac{1}{\bar{a}_0 \hat{a}_0}})^{\hat{a}_k}} \cdot C_0.$$

Correctness.

In Algorithms 1:

$$\begin{aligned} e(g^{r_2}, (V_i \prod_{j=1}^{L} H_i^{x_{ij}})^s Z_3) &\overset{?}{=} e(G^s Z_1, (v_i \prod_{j=1}^{L} h_i^{y_{ij}})^{r_2}) \\ e(g, v_i)^{sr_2} e(g, \prod_{j=1}^{L} h_i^{x_{ij}})^{sr_2} &\overset{?}{=} e(g, v_i)^{sr_2} e(g, \prod_{j=1}^{L} h_i^{y_{ij}})^{sr_2} \end{aligned}$$

We then illustrate an example:

Input: "AAGTA", "AAAGG"
Output:
- $d' = 2$
- $pos =< pos[0][], pos[1][] >$, with $pos[0][] = \{1, 2\}, pos[1][] = \{1, 2\}$

In message recovery

$$\begin{aligned} C_0 &= M \cdot e(g, \omega)^s \\ e(K_2, C_4) &= e(g^{r_2}, (V_0' \prod_{i=1}^{L} H_i^{v_i})^s \cdot Z_4) \\ &= e(g, v_0')^{sr_2} e(g, \prod_{i=1}^{L} (h_i^{v_i})^{sr_2}) \\ e(K_1, C_2) &= e(g^{r_1}, F^s Z_2) = e(g, f)^{sr_1} \\ \prod_{k=0}^{\tau} e(K_1^{\frac{1}{\bar{a}_0 \hat{a}_0}}, \prod_{t=0}^{\tau} C_{5,k,t}^{a_t})^{\hat{a}_k} &= \prod_{k=0}^{\tau} e(g^{\frac{r_1}{\bar{a}_0}}, \prod_{t=0}^{\tau} (V_t'(B_k \prod_{i=1}^{L} (U_i)^{d_i i^k})(\prod_{j=1}^{L} (H_j)^{v_j j^t}))^{sa_t})^{\hat{a}_k} \\ &= e(g, v_0')^{\frac{sr_1 \sum_{t=0}^{\tau} x_t a_t \sum_{k=0}^{\tau} \hat{a}_k}{\bar{a}_0 \hat{a}_0}} e(g, b_0)^{\frac{sr_1 \sum_{t=0}^{\tau} a_t \sum_{k=0}^{\tau} x_k' \hat{a}_k}{\bar{a}_0 \hat{a}_0}} \prod_{i=1}^{L} e(g, u_i)^{\frac{sr_1 \sum_{t=0}^{\tau} a_t \prod_{k=1}^{\tau} (i - d_k)}{\bar{a}_0 \hat{a}_0}} \end{aligned}$$

$$\prod_{j=1}^{L} e(g, h_j)^{\frac{v_j s r_1 \prod_{t=1}^{\tau}(i-\omega_t) \sum_{k=0}^{\tau} \hat{a}_k}{\bar{a}_0 \hat{a}_0}}.$$

$$\prod_{k=0}^{\tau} e(\prod_{t=0}^{\tau} K_{4,k,t}^{\bar{a}_t}, C_1^{\frac{1}{\bar{a}_0\hat{a}_0}})^{\hat{a}_k} = e(\omega^{\bar{a}_0\hat{a}_0}(v_0' \prod_{i=1}^{L}(h_i^{z_i}))^{r_2\bar{a}_0\hat{a}_0} \prod_{k=1}^{\tau}((\prod_{t=1}^{\tau}(v_t'(b_k \prod_{i=1}^{L}(u_i)^{\sigma_i i^k})(\prod_{j=1}^{L}(h_j)^{z_j j^t}))^{r_1\bar{a}_t})^{\hat{a}_k} f^{r_1\bar{a}_0\hat{a}_0}, G^{\frac{s}{\bar{a}_0\hat{a}_0}} Z_1)$$

$$= e(g, \omega)^s e(g, v_0')^{sr_2} e(g, \prod_{i=1}^{L}(h_i^{z_i})^{sr_2})$$

$$e(g, v_0')^{\frac{r_1 s \sum_{t=0}^{\tau} x_t \bar{a}_t \sum_{k=1}^{\tau} \hat{a}_k}{\bar{a}_0}} e(g, b_0)^{\frac{s r_1 \sum_{t=0}^{\tau} \bar{a}_t \sum_{k=0}^{\tau} x_k' \hat{a}_k}{\bar{a}_0 \hat{a}_0}} \prod_{i=1}^{L} e(g, u_i)^{\frac{s r_1 \sum_{t=0}^{\tau} \bar{a}_t \prod_{k=1}^{\tau}(i-\sigma_k)}{\bar{a}_0\hat{a}_0}})$$

$$\prod_{j=1}^{L} e(g, h_j)^{\frac{s r_1 z_j \prod_{t=1}^{\tau}(i-\bar{\omega}_t) \sum_{k=0}^{\tau} \hat{a}_k}{\bar{a}_0 \hat{a}_0}} e(g, f)^{sr_1}.$$

5 Security Analysis for The Proposed EDE Scheme

Theorem 1. *Assume that the Decisional $L-$cBDHE assumption holds, then for any PPT adversary, our EDE scheme is selectively secure.*

Let B denote the algorithm to solve the Decisional $L-$cBDHE problem. B is given a challenge instance Z, T' of the problem, where $Z = (g_p, g_q, h, g_p^{\alpha}, \ldots, g_p^{\alpha^L}, g_p^{\alpha^{L+2}}, \ldots, g_p^{\alpha^{2L}})$ and T' is either $T = e(g_p, h)^{\alpha^{L+1}}$ or $R \in_R \mathbb{G}_{T,p}$.

B simulates the game for A as follows:

- **Init**: A submits a target vector $\overrightarrow{v}^* \in \Sigma^n$, and target threshold τ. Let $\overrightarrow{d} = (1, \ldots, \tau, \tau+1, \ldots, L)$ denote a vector of length L. We denote $ind(\overrightarrow{d}) = \{1 \le i \le L | d_i = 0\}$ and $\overline{ind}(\overrightarrow{d}) = \{1 \le i \le L | d_i \ne 0\}$, and $ind(\overrightarrow{d})|_j^{\phi}$ as $\{i \in ind(\overrightarrow{d}) | j \le i \le \phi\}$.
- **Setup**: In this phase, $\mathcal{B}$ generates:

$$\gamma, \psi, v_0, v_0', b_0, g, f, h_1', \ldots, h_L', u_1', \ldots, u_L' \in_R \mathbb{G}_p, x_1, \ldots, x_L, x_1', \ldots, x_L' \in_R \mathbb{Z}_n,$$
$$v_1 = v_0^{x_1}, \ldots, v_L = v_0^{x_L}, v_1' = (v_0')^{x_1}, \ldots, v_L' = (v_0')^{x_L}, b_1 = b_0^{x_1'}, \ldots, b_L = b_0^{x_L'},$$
$$R_y, R_g, R_f, R_{v_0}, \ldots, R_{v_L}, R_{v_0'}, \ldots, R_{v_L'}, R_{b_0}, \ldots, R_{b_L}, R_{h_1'}, \ldots, R_{h_L'}, R_{u_1'}, \ldots, R_{u_L'} \in_R \mathbb{G}_q,$$

$$G = g_p R_g, F = g_p^{\psi} R_f, Y = e(g_p^{\alpha}, g_p^{\alpha^L} g_p^{\gamma}),$$
$$V_t = g_p^{v_0 x_t} R_{v_t}, V_t' = g_p^{v_0' x_t} R_{v_t'}, \text{ with } t = 1, \ldots, L,$$
$$B_k = g_p^{b_0 x_k'} \prod_{k \in ind(\overrightarrow{d})} g_p^{\alpha^{L+1-i} d_i i^k} R_{b_k}, \text{ with } k = 1, \ldots, L,$$
$$H_i = g_p^{h_i'} R_{h_i}, \{U_i = g_p^{u_i' - \alpha^{L+1-i}} R_{u_i'}\}_{i \in ind(\overrightarrow{d})}, \{U_i = g_p^{u_i'} R_{u_i'}\}_{i \in \overline{ind}(\overrightarrow{d})}$$

The corresponding master secret key components are: $g = g_p, f = g_p^{\psi}, h_i = g_p^{h_i'}, \{u_i = g_p^{u_i' - \alpha^{L+1-i}}\}_{i \in \overline{ind}(\overrightarrow{d})}$, $\{u_i = g_p^{u_i'}\}_{i \in ind(\overrightarrow{d})}, v_t = v_0^{x_t}, v_t' = {v_0'}^{x_t}$, with $t = 1, \ldots, L, b_k = b_0^{x_t'} \prod_{i \in ind(\overrightarrow{d})} g_p^{\alpha^{L+1-i} d_i}$, with $k =$

$1, \ldots, L$. Notice that the master key component ω is $g_p^{\alpha^{L+1}+\alpha\gamma}$. Since B does not have $g_p^{\alpha^{L+1}}$, B cannot compute ω directly.

- **Query Phase 1**: A queries the user secret key for a string $\overrightarrow{z} = (z_1, z_2, \ldots, z_m)$ under the constraint that $EditDistance(\overrightarrow{v}^*, \overrightarrow{z}) > \tau$. Assume $EditDistance(\overrightarrow{v}^*, \overrightarrow{z}) = \sigma$ and denote $\overrightarrow{\sigma} = (1, 2, \ldots, \sigma, 0, \ldots, 0)$ and $\overrightarrow{d} = (1, 2, \ldots, \tau, 0, \ldots, 0)$. Note that since $\sigma > \tau$, there exists at least one position i such that $d_i = 0$ and $\sigma_i \neq 0$. Let $\phi \in ind(\overrightarrow{d})$ be the smallest integer such that $\sigma_\phi \neq d_\phi$.
  B simulates the user key generation process as follows:

$$\begin{aligned} K_{4,0,0} &= \omega(v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i})^{r_2}(b_0 \prod_{i=1}^{L}(u_i^{\sigma_i}))^{r_1}(v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i}))^{r_1} f^{r_1} \\ &= g_p^{\alpha^{L+1}+\alpha\gamma}(v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i})^{r_2}(g_p^{b_0} \prod_{i\in ind(\overrightarrow{d})} g_p^{\alpha^{L+1-i}d_i} \prod_{\overline{ind(\overrightarrow{\sigma})}} (g^{u_i'-\alpha^{L+1-i}})^{\sigma_i} \\ &\cdot \prod_{ind(\overrightarrow{\sigma})} (g^{u_i'})^{\sigma_i})^{r_1}(v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j})^{r_1} f^{r_1} \\ &\overset{def}{=} g_p^{\alpha^{L+1}+\alpha\gamma}(v_0' \prod_{i=1}^{L} u_i^{\bar{w}_i})^{r_2}(g_p^{X})^{r_1}(v_0' \prod_{i=j}^{L} h_j^{\bar{w}_j})^{r_1} f^{r_1} \end{aligned}$$

where $X = \sum_{\overline{ind(\overrightarrow{d})}} \alpha^{n+1-i}d_i + b_0 + \sum_{\overline{ind(\overrightarrow{\sigma})}}(u_i' - \alpha^{L+1-i})\sigma_i + \sum_{ind(\overrightarrow{\sigma})} u_i'\sigma_i$. Since $\sum_{\overline{ind(\overrightarrow{\sigma})}}(u_i' - \alpha^{L+1-i})\sigma_i + \sum_{ind(\overrightarrow{\sigma})} u_i'\sigma_i = -\sum_{\overline{ind(\overrightarrow{\sigma})}} \alpha^{L+1-i}\sigma_i + \sum_{i=1}^{L} u_i'\sigma_i$, and recall $\sigma_i = d_i$ for $i \in \overline{ind(\overrightarrow{d})}|_1^{\phi-1}$ and $\sigma_\phi \neq d_\phi$. Hence, we have:

$$X = \sum_{\overline{ind(\overrightarrow{d})}|_1^{\phi}} \alpha^{L+1-i}(d_i - \sigma_i) + \sum_{i=1}^{L} u_i'\sigma_i + b_0 = \alpha^{L+1-\phi}\Delta_\phi + \sum_{i=1}^{L} u_i'\sigma_i + y$$

where $\Delta_\phi = (d_\phi - \sigma_\phi)$. Then we choose $\hat{r}, r_2'$ randomly in $\mathbb{Z}_n$, and set $r_1 = \frac{-\alpha^\phi}{\Delta_\phi} + r_1', r_2 = r_2'$. Then $K_{4,0,0}$ can be represented as:

$$\begin{aligned} K_{4,0,0} &= g_p^{\alpha^{L+1}+\alpha\gamma}(v_0' \prod_{i=1}^{L}(h_i)^{\bar{w}_i})^{r_2'} \cdot (g_p^{\alpha^{L+1-\phi}\Delta_\phi + \sum_{i=1}^{L} u_i'\sigma_i})^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'}(v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} f^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \\ &= g_p^{\alpha^{L+1}+\alpha\gamma}(v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i})^{r_2'} \cdot g_p^{-\alpha^{L+1}}(g_p^{\alpha^{L+1-\phi}\Delta_\phi})^{r_1'}(g_p^{\sum_{i=1}^{L} u_i'\sigma_i})^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \\ &\quad (v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} f^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \\ &= (v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i})^{r_2'} \cdot (g_p^{\alpha^{L+1-\phi}\Delta_\phi})^{r_1'}(g_p^{\sum_{i=1}^{L} u_i'\sigma_i})^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'}(v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} f^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \end{aligned}$$

Then we simulate $T_{4,k,t}$ with $k, t \neq 0$ as:

$$\begin{aligned} T_{4,k,t} &= ((b_k \prod_{i=1}^{L}(u_i^{i^L\sigma_i})(v_t' \prod_{j=1}^{L} h_i^{\bar{w}_i i^t}))^{r_1} \\ &= (g_p^{b_k} \prod_{\phi\in ind(\overrightarrow{d})|_{\phi+1}^{L}} g_p^{\alpha^{L+1-\phi}d_\phi} \prod_{\phi\in\overline{ind(\overrightarrow{\sigma})}|_{\phi+1}^{L}} (g^{u_\phi'-\alpha^{L+1-\phi}})^{\sigma_\phi} \cdot \prod_{\phi\in ind(\overrightarrow{\sigma})} (g^{u_\phi'})^{\sigma_\phi})^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \\ &\quad \cdot(v_t' \prod_{j=1}^{L} h_j^{\bar{w}_j i^t}))^{\frac{-\alpha^\phi}{\Delta_\phi}+r_1'} \end{aligned}$$

Next, it generates for each alphabet in $\overrightarrow{z}$: $\begin{cases} \boldsymbol{y_1} & = (z_1, 1, \ldots, 1_L) \\ \ldots, & \\ \boldsymbol{y_m} & = (z_m, 1, \ldots, 1_L) \end{cases}$, then computes $K_{3,i} = (v_i \prod_{j=1}^{L} h_i^{y_{ij}})^{r'_2}$. Other elements in the key can also be simulated: $K_1 = g^{r_1} = g^{\frac{-\alpha^\phi}{\Delta_\phi} + r'_1}, K_2 = g^{r'_2}$.

- **Challenge**: A sends two message M_0, M_1 to B. The challenger then flips a coin $\beta \leftarrow \{0,1\}$. First, $\mathcal{B}$ generates for each alphabet in $\overrightarrow{v}^*$: $\begin{cases} \boldsymbol{x_1^*} & = (v_1, 1, \ldots, 1_L) \\ \ldots, & \\ \boldsymbol{x_m^*} & = (v_m, 1, \ldots, 1_L) \end{cases}$, then generates $Z_1, Z_2, Z_3, Z_4, Z_5 \xleftarrow{R} \mathbb{G}_q$ and sets:

$$C_0 = M_b \cdot T' \cdot e(g_p^\alpha, h)^\gamma, C_1 = hZ_1, C_2 = h^\psi Z_2, C_{3,i} = h^{v_0 x_i + \sum_{j=1}^{L} h'_i x^*_{i,j}} Z_3,$$
$$C_4 = h^{v'_0 + \sum_{i=1}^{L} h'_i v^*_i} Z_4, C_{5,k,t} = h^{b_k x'_k \sum_{i=1}^{L} u'_i d_i i^k + v'_t \sum_{j=1}^{L} v^*_j h'_j j^t} Z_5$$

where $h = g_p^c$ for some unknown $c \in \mathbb{Z}_p$. B returns the challenge ciphertext

$$CT^* = (n_1, C_1, C_2, \{C_{3,i}\}_{i=1}^{n_1}, C_4, \{\{C_{5,k,t}\}_{k=0}^{L}\}_{t=0}^{L})$$

to A. If $T' = T = e(g_p, h)^{\alpha^L + 1}$, then:

$$C_0 = M_b \cdot e(g_p, g_p^c)^{\alpha^{L+1}} \cdot e(g_p^\alpha, g_p^c)^\gamma = M_b \cdot e(g_p, g_p^{\alpha^{L+1}})^c \cdot e(g_p^\alpha, g_p^\gamma)^c = M_b \cdot Y^c$$
$$C_1 = (g_p^c) \cdot Z_1 = G^c \cdot Z'_1, C_2 = (g_p^c)^\psi \cdot Z_2 = F^c \cdot Z'_2,$$
$$C_{3,i} = (g_p^c)^{v_0 x_i + \sum_{j=1}^{L} h'_i x^*_{i,j}} Z_3 = (g_p^{v_0 x_i + \sum_{j=1}^{L} h'_i x^*_{i,j}})^c Z_3 = (V_i \prod_{j=1}^{L} H_i^{x^*_{ij}})^c \cdot Z'_3,$$
$$C_4 = (g_p^c)^{v'_0 + \sum_{i=1}^{L} h'_i v^*_i} \cdot Z_4 = ((g_p)^{v'_0 + \sum_{i=1}^{L} h'_i v^*_i})^c \cdot Z_4 = (V'_0 \prod_{i=1}^{L} H_i^{v^*_i})^c \cdot Z'_4.$$
$$C_{5,k,t} = (g_p^c)^{b_k x'_k \sum_{i=1}^{L} u'_i d_i i^k + v'_t \sum_{j=1}^{L} v^*_j h'_j j^t} \cdot Z_5 = ((g_p)^{b_k x'_k \sum_{i=1}^{L} u'_i d_i i^k + v'_t \sum_{j=1}^{L} v^*_j h'_j j^t})^c \cdot Z_5.$$
$$= (V'_t (B_k \prod_{i=1}^{L} (U_i)^{d_i i^k})(\prod_{j=1}^{L} (H_j)^{v^*_j j^t}))^c \cdot Z'_5.$$

the challenge ciphertext is a valid encryption of M_b. On the other hand, when T' is uniformly distributed in $\mathbb{G}_{T,p}$, the challenge ciphertext is independent of b.

- **Query Phase 2**: Same as Phase 1.
- **Guess**: A output $b' \in \{0,1\}$. If $b' = b$ then B outputs 1; otherwise outputs 0.

If $b' = 0$, then the simulation is the same as in the real game. Hence, A will have the probability $\frac{1}{2} + \epsilon$ to guess b correctly. If $b' = 1$, then T' is random in $\mathbb{G}_{T,p}$, then A will have probability $\frac{1}{2}$ to guess b correctly.
Therefore, B can solve the Decisional $L-$cBDHE assumption also with advantage ϵ. □

6 Extension - Fuzzy Broadcast Encryption (FBE)

We demonstrate an extension of the proposed EDE scheme to achieve Fuzzy Broadcast Encryption. To illustrate how the scheme works, let's consider the following example. Suppose we encrypt a message under a keyword vector $W = \{\mathsf{Labour\ Party}, \mathsf{Defence\ Unit}\}$ and a threshold distance $d = 2$. Subsequently, people who have the attributes related to the keyword $w = \mathsf{Labor\ Party}$ or $w' = \mathsf{Defense\ Unit}$ can decrypt the message since the minimum edit distance between w (w', respectively) and all the keywords in W is 1, which is less than the threshold $d = 2$.

6.1 Definition

A Fuzzy Broadcast Encryption (FBE) scheme consists of the following four probabilistic polynomial-time algorithms:

- **Setup**$(1^n, \Sigma)$: on input a security parameter 1^n, an alphabet Σ, the algorithm outputs a public key PK and the corresponding master secret key MSK.
- **Encrypt**$(PK, M, W = (w_{1,l_1}, w_{2,l_2}, \ldots, w_{k,l_k}) \in \Sigma^{n_1}, d)$: on input a public key PK, a list of k keywords $W = (w_{1,l_1}, w_{2,l_2}, \ldots, w_{k,l_k})$ in which each keyword w_{i,l_i} has l_i characters, and a threshold distance d, the algorithm outputs a ciphertext CT.
- **Key Gen**$(MSK, w \in \Sigma^m)$: on input the master secret key MSK and a keyword w of length m, the algorithm outputs a secret key SK_w.
- **Decrypt**(CT, SK_w): on input a ciphertext CT with keywords $W = (w_{1,l_1}, w_{2,l_2}, \ldots, w_{k,l_k})$ and a secret key SK_w with keyword w, the algorithm outputs M if $Min\{EditDistance(w_{i,l_i}, w)\}_{i=1}^{k} \leq d$, or $\perp$ otherwise.

6.2 Security Model

The security model for a FBE scheme is defined via the following game between an adversary A and a challenger B.

- **Setup**: The challenger B runs **Setup**$(1^n, \Sigma)$ to generate PK and MSK. PK is then passed to A.
- **Query Phase 1**: The challenger answers private key queries for any keyword w by returning $sk_w \leftarrow KeyGen(MSK, w)$.
- **Challenge**: A submits two equal-length messages M_0 and M_1, a target set of keywords $W^* = \{W_{i,l_i}\}_{i=1}^{k}$, and a threshold τ such that $Min\{EditDistance(W_{i,l_i},\ w)\}_{i=1}^{k} > \tau$ for all keywords w that have been queried in Phase 1. The challenger then flips a coin $\beta \leftarrow \{0, 1\}$, and computes the challenge ciphertext $C^* \leftarrow Encrypt(PK, M_\beta, \{W_{i,l_i}\}_{i=1}^{k}, \tau)$ which is given to A.
- **Query Phase 2**: same as Query Phase 1 under the restriction that $Min\{EditDistance(W_{i,l_i}, w)\}_{i=1}^{k} > \tau$ for any keyword w queried.
- **Output**: A outputs a bit β' as her guess for β.

Define the advantage of A as: $\mathbf{Adv}_A^{\mathsf{FBE}}(k) = |\Pr[\beta' = \beta] - 1/2|$.

Selective Security. In the selective model, the adversary A is required to submit the target vector W^* and threshold τ before the game setup.

6.3 FBE Scheme

Below we present a FBE scheme based on our EDE scheme.

- **Setup**$(1^n, \Sigma)$: The setup algorithm first chooses $L = poly(n)$ as the maximum number of length of a word that would appear in the encryption and key generation. It then picks large primes p, q, generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of composite order $n = pq$, and selects generators $g_p \in \mathbb{G}_p, g_q \in \mathbb{G}_q$. After that, generate

$$\begin{aligned}
&v_0, v'_0, b_0, g, f, \omega, h_1, \ldots, h_L, u_1, \ldots, u_L \in_R \mathbb{G}_p, x_1, \ldots, x_L, x'_1, \ldots, x'_L \in_R \mathbb{Z}_n,\\
&v_1 = v_0^{x_1}, \ldots, v_L = v_0^{x_L}, v'_1 = (v'_0)^{x_1}, \ldots, v'_L = (v'_0)^{x_L}, b_1 = b_0^{x'_1}, \ldots, b_L = b_0^{x'_L},\\
&R_g, R_f, R_{v_0}, \ldots, R_{v_L}, R_{v'_0}, \ldots, R_{v'_L}, R_{b_0}, \ldots, R_{b_L}, R_{h_1}, \ldots, R_{h_L}, R_{u_1}, \ldots, R_{u_L} \in \mathbb{G}_q,\\
&G = gR_g, F = fR_f, Y = e(g, \omega),\\
&V_0 = v_0R_{v_0}, \ldots, V_L = v_LR_{v_L}, V'_0 = v'_0R_{v'_0}, \ldots, V'_L = v'_LR_{v'_L}, B_0 = b_0R_{b_0}, \ldots, B_L = b_LR_{b_L},\\
&H_1 = h_1R_{h_1}, \ldots, H_L = h_LR_{h_L}, U_1 = u_1R_{u_1}, \ldots, U_L = u_LR_{u_L},
\end{aligned}$$

  and set the public key and secret key as:

$$\begin{aligned}
&PK = \{Y, G, F, (V_0, \ldots, V_L), (V'_0, \ldots, V'_L), (B_0, \ldots, B_L), (H_1, \ldots, H_L), (U_1, \ldots, U_L)\},\\
&MSK = \{g, f, \omega, (v_0, \ldots, v_L), (v'_0, \ldots, v'_L), (b_0, \ldots, b_L), (h_1, \ldots, h_L), (u_1, \ldots, u_L)\}.
\end{aligned}$$

- **Encrypt**$(PK, W = (w_{1,l_1}, w_{2,l_2}, \ldots, w_{k',l_{k'}}), M, d)$: On input the public key PK, a list of k keywords $W = (w_{1,l_1}, w_{2,l_2}, \ldots, w_{k',l_{k'}})$ in which each keyword w_{i,l_i} has l_i alphabets, it first generates for each alphabet $w_{i,j}$ in keyword w_{i,l_i} a vector

$$\begin{cases}
\boldsymbol{x_{11}} = (w_{11}, 1, \ldots, 1_L), \ldots, \boldsymbol{x_{1l_1}} & = (w_{1l_1}, 1, \ldots, 1_L),\\
\ldots & \\
\boldsymbol{x_{k'1}} = (w_{k'1}, 1, \ldots, 1_L), \ldots, \boldsymbol{x_{k'l'_k}} & = (w_{k'l_{k'}}, 1, \ldots, 1_L).
\end{cases}$$

  Define

$$\begin{cases}
\boldsymbol{w_1} & = (w_{11}, w_{1,2}, \ldots, w_{1,l_1}, \ldots, 1_L),\\
\ldots, & \\
\boldsymbol{w_{k'}} & = (w_{k'1}, w_{k'2}, \ldots, w_{k',l_{k'}}, \ldots, 1_L),
\end{cases}$$

  and $\overrightarrow{d} = (1, \ldots, d, 0_{d+1}, \ldots, 0_L)$. Then choose $s \in_R \mathbb{Z}_n$, and $Z_1, Z_2, Z_3, Z_4, Z_5 \in_R \mathbb{G}_q$, and compute

$$\begin{aligned}
&C_0 = MY^s, C_1 = G^sZ_1, C_2 = F^sZ_2, C_{3,\delta,i} = (V_i \prod_{j=1}^{L} H_i^{x_{\delta ij}})^s Z_3,\\
&C_{4,\delta} = (V'_0 \prod_{i=1}^{L} H_i^{w_{\delta i}})^s \cdot Z_4, C_{5,k,\delta,t} = (V'_t(B_k \prod_{i=1}^{L} (U_i)^{d_i i^k})(\prod_{j=1}^{L} (H_j)^{v_{\delta j} j^t}))^s \cdot Z_5.
\end{aligned}$$

  Set the ciphertext as: $CT = (\{l_\delta\}_{\delta=1}^{k'}, C_0, C_1, C_2, \{\{C_{3,\delta,i}\}_{\delta=1}^{k'}\}_{i=1}^{l_\delta}, \{C_{4,\delta}\}_{\delta=1}^{k'}, \{\{\{C_{5,k,\delta,t}\}_{k=0}^{L}\}_{\delta=1}^{k'}\}_{t=0}^{L})$.

- **KeyGen**$(MSK, \bar{w} = (\bar{w}_1, \ldots, \bar{w}_m) \in \Sigma^m)$: given a keyword $\bar{w}$ of length m, it generates $\boldsymbol{y_i} = (\bar{w}_i, 1, \ldots, 1_L)$ for each alphabet $\bar{w}_i$, and creates $\overrightarrow{\sigma} = (1, 2, ..., L)$ and expands $\bar{w}$ to $\bar{w} = (z_1, z_2, \ldots, z_m, \ldots, 1_L)$. Then choose $r_1, r_2 \in_R Z_n$, and compute

$$K_1 = g^{r_1}, K_2 = g^{r_2}, K_{3,i} = (v_i \prod_{j=1}^{L} h_i^{y_{ij}})^{r_2},$$

$$\begin{pmatrix} K_{4,0,0} = \omega(v_0' \prod_{i=1}^{L} h_i^{\bar{w}_i})^{r_2}((b_0 \prod_{i=1}^{L} (u_i^{\sigma_i}))(v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{r_1} f^{r_1} \\ K_{4,1,0} = (b_1 \prod_{i=1}^{L} (u_i^{\sigma_i})^i (v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{r_1} \\ \ldots, \\ K_{4,L,0} = (b_L \prod_{i=1}^{L} (u_i^{\sigma_i})^{i^L} (v_0' \prod_{j=1}^{L} h_j^{\bar{w}_j}))^{r_1} \end{pmatrix}, \begin{pmatrix} K_{4,0,t} = (b_0 \prod_{i=1}^{L} (u_i^{\sigma_i})(v_t' \prod_{j=1}^{L} h_j^{\bar{w}_j})^{j^t})^{r_1} \\ K_{4,1,t} = (b_1 \prod_{i=1}^{L} (u_i^{\sigma_i})^i (v_t' \prod_{j=1}^{L} h_j^{\bar{w}_j})^{j^t})^{r_1} \\ \ldots, \\ K_{4,L,t} = (b_L \prod_{i=1}^{L} (u_i^{\sigma_i})^{i^L} (v_t' \prod_{j=1}^{L} h_j^{\bar{w}_j})^{j^t})^{r_1} \end{pmatrix}$$

, for $t = 1, \ldots, L$. Then set the secret key as $SK = (m, K_1, K_2, \{K_{3,i}\}_{i=1}^{m}, \{\{K_{4,k,t}\}_{k=0}^{L}\}_{t=0}^{L})$.
- **Decrypt**(CT, SK): The decryption algorithm first executes the dynamic programming algorithm for edit distance by following **Algorithm** 2 which returns a minimum distance d', the index pos_w of the corresponding keyword w in W, the matching indices array $pos[0][]$ for w and $pos[1][]$ for $\bar{w}$. It sets $\tau = L - d'$, and applies the Viète's formulas to compute
  - for the index set $\Omega_v = \{L \backslash \{pos[0][0], \ldots, pos[0][d'-1]\}\} = \{\omega_1, \ldots, \omega_{L-d'}\}$

$$a_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} \omega_{i_1} \omega_{i_2} \ldots \omega_{i_k} \quad (0 \le k \le \tau)$$

  - for the index set $\Omega_z = \{L \backslash \{pos[1][0], \ldots, pos[1][d'-1]\}\} = \{\bar{\omega}_1, \ldots, \bar{\omega}_{L-d'}\}$

$$\bar{a}_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} \bar{\omega}_{i_1} \bar{\omega}_{i_2} \ldots \bar{\omega}_{i_k} \quad (0 \le k \le \tau)$$

  - for the index set $J = \{j_1, \ldots, j_\tau\}$ with $j_1 = d'+1, \ldots, j_\tau = L$

$$\hat{a}_{\tau-k} = (-1)^k \sum_{1 \le i_1 < i_2 < \ldots < i_k \le \tau} j_{i_1} j_{i_2} \ldots j_{i_k} \quad (0 \le k \le \tau).$$

  Then recover M as

$$M = \frac{e(K_2, C_{4,pos_w}) \cdot e(K_1, C_2) e(K_1^{\frac{1}{a_0}}, \prod_{t=0}^{\tau} C_{5,k,pos_w,t}^{a_t})}{e(\prod_{t=0}^{\tau} K_{4,k,t}^{\bar{a}_t}, C_1^{\frac{1}{\bar{a}_0}})} \cdot C_0.$$

Theorem 2. *Assume that the Decisional $L-$cBDHE assumption holds, then for any PPT adversary, our FBE scheme is selectively secure.*

The security proof follows that of **Theorem 1** and is omitted here.

Algorithm 2: Multi-keyword Edit Distance Evaluation via Dynamic Programming

```
input  : $CT, SK$
output: distance $d'$, index $pos_w$, array $pos[2][]$
Create $Array[len(W)]$;
Create $pos[2][]$;
Create $Array < pos > aPos$;
for $\theta \leftarrow 1$ to $len(W)$ do
    $len_v = n_\theta + 1$;
    $len_z = m + 1$;
    Creat $cost[len_v]$;
    Creat $newcost[len_v]$;
    for $i \leftarrow 0$ to $len_v$ do
        $cost[i] = i$;
    end
    $k = 0$;
    for $j \leftarrow 1$ to $len_z$ do
        $newcost[0] = j$;
        for $i \leftarrow 1$ to $len_v$ do
            $match = (e(K_1, C_{3,\theta,i-1}) == e(C_1, K_{3,j-1}))?0 : 1$;
            if $i \notin pos[0], j \notin pos[1]$ then
                $pos[0][k++] = i, pos[1][k++] = j$, ;
            end
            $aPos.add(pos)$;
            $cost - replace = cost[i-1] + match$;
            $cost - insert = cost[i] + 1$;
            $cost - delete = newcost[i-1] + 1$;
            $newcost[i] = Math.min(Math.min(cost - insert, cost - delete), cost - replace)$;
        end
        $swap[] = cost$;
        $cost = newcost$;
        $newcost = swap$;
    end
    $Array[t++] = cost[len_v - 1]$;
    $Refresh\ pos$;
end
return $Min(Array[]), pos_w = index[Array[i] == Min(Array[])], pos = aPos[pos_w]$;
```

7 Conclusions and Future Work

We introduced a new type of fuzzy public key encryption in this paper. Our new encryption scheme, called Edit Distance-based Encryption (EDE), allows a user associated with an identity or attribute string to decrypt a ciphertext encrypted under another string if and only if the edit distance between the two strings are within a threshold specified by the encrypter. We provide the formal definition, security model, and a concrete EDE scheme in the standard model. We also showed an extension of our EDE scheme for fuzzy broadcast encryption.

Our EDE scheme cannot preserve the privacy of the keyword used for encryption due to the way used by the dynamic programming algorithm to check the matching positions of two keywords. We leave the construction of an anonymous EDE scheme, which implies a Fuzzy Public-key Encryption with Keyword Search scheme, as our future work.

Acknowledgement

Kaitai Liang is supported by privacy-aware retrieval and modelling of genomic data (PRIGENDA, No. 13283250), the Academy of Finland.

References

1. M. Abdalla, J. Birkett, D. Catalano, A. W. Dent, J. Malone-Lee, G. Neven, J. C. N. Schuldt, and N. P. Smart. Wildcarded identity-based encryption. *J. Cryptology*, 24(1):42–82, 2011.
2. M. Abdalla, A. D. Caro, and D. H. Phan. Generalized key delegation for wildcarded identity-based and inner-product encryption. *IEEE Trans. Inf. Forensic Secur.*, 7(6):1695–1706, 2012.
3. M. Abdalla, D. Catalano, A. W. Dent, J. Malone-Lee, G. Neven, and N. P. Smart. Identity-based encryption gone wild. In *ICALP '06*, pp. 300–311, 2006.
4. M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES '03*, pp. 39–44.
5. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P '07*, pp. 321–334, 2007.
6. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology – CRYPTO*, pp. 213–229, 2001.
7. J. H. Cheon, M. Kim, and K. Lauter. Homomorphic computation of edit distance. Cryptology ePrint Archive, Report 2015/132, 2015.
8. A. Ge, R. Zhang, C. Chen, C. Ma, and Z. Zhang. Threshold ciphertext policy attribute-based encryption with constant size ciphertexts. In *ACISP '12*, pp. 336–349, 2012.
9. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS '06*, pp. 89–98, 2006.
10. D. Gusfield. *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, 1997.
11. J. Herranz, F. Laguillaumie, and C. Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In *PKC '10*, pp. 19–34, 2010.
12. V. Iovino and G. Persiano. Hidden-vector encryption with groups of prime order. In *Pairing*, pp. 75–88, 2008.
13. S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE S&P '08*, pp. 216–230, 2008.
14. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT '08*, pp. 146–162, 2008.
15. K. Liang, J. K. Liu, D. S. Wong, and W. Susilo. An efficient cloud-based revocable identity-based proxy re-encryption scheme for public clouds data sharing. In *ESORICS '14*, pp. 257–272. Springer, 2014.
16. K. Liang and W. Susilo. Searchable attribute-based mechanism with efficient data sharing for secure cloud storage. *IEEE Trans. Inf. Forensic Secur.*, 10(9):1981–1992, 2015.
17. K. Liang, W. Susilo, and J. K. Liu. Privacy-preserving ciphertext multi-sharing control for big data storage. *IEEE Trans. Inf. Forensic Secur.*, 10(8):1578–1589, 2015.
18. J. H. Park. Efficient hidden vector encryption for conjunctive queries on encrypted data. *IEEE Trans. on Knowl. and Data Eng.*, 23(10):1483–1497, 2011.
19. T. V. X. Phuong, G. Yang, and W. Susilo. Efficient hidden vector encryption with constant-size ciphertext. In *ESORICS '14*, pp. 472–487, 2014.
20. S. Rane and W. Sun. Privacy preserving string comparisons based on levenshtein distance. In *IEEE WIFS '10*, pp. 1–6, 2010.
21. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT '05*, pp. 457–473, 2005.
22. S. Sedghi, P. van Liesdonk, S. Nikova, P. H. Hartel, and W. Jonker. Searching keywords with wildcards on encrypted data. In *SCN '10*, pp. 138–153, 2010.
23. X. S. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *ACM CCS '15*, 2015.