Fuzzy Keyword Search over Encrypted Data in the Public Key Setting

Qiuxiang Dong[1,2,3], Zhi Guan[1,2,3,⋆], Liang Wu[4], and Zhong Chen[1,2,3]

[1] Institute of Software, School of EECS, Peking University, China
[2] MoE Key Lab of High Confidence Software Technologies (PKU)
[3] MoE Key Lab of Network and Software Security Assurance (PKU)
[4] Computer Network Information Center, Chinese Academy of Sciences
{dongqx,guanzhi,chen}@infosec.pku.edu.cn, wuliang@cnic.cn

Abstract. Searchable encryption is used to support searches over encrypted data stored on cloud servers. Traditional searchable encryption only supports exact keyword search instead of more flexible fuzzy keyword search. To solve this problem, a recent emerging paradigm, named fuzzy keyword searchable encryption, has been proposed. There have been some proposals designed for fuzzy keyword search in the symmetric key setting, but none efficient schemes in the public key setting. In this paper, we propose a new primitive of interactive public key encryption with fuzzy keyword search (IPEFKS), which supports efficient fuzzy keyword search over encrypted data in the public key setting. We construct and implement a homomorphic encryption based IPEFKS scheme. To compare this scheme with the existing ones, we implement LWW-FKS, which, to the best of our knowledge, is the most efficient among the existing schemes. The experimental results show that IPEFKS is much more efficient than LWW-FKS.

Keywords: Fuzzy Keyword Search, Public key Encryption with Keywords Search, Cloud Computing.

1 Introduction

1.1 Background

In recent years, due to the appealing features of cloud computing, more and more data have been centralized into cloud. To protect clients' privacy, data with sensitive information are usually stored in the encrypted form. However, encrypted storage makes it hard to perform searches over the data. To cope with this problem, various techniques for searching over encrypted data, namely searchable encryption, have been proposed in the literature [1–4].

Searchable encryption schemes can be divided into two categories [14], the symmetric key searchable encryption (SSE) [3, 4] and the public key searchable encryption (PSE) [1, 2]. In SSE, the data sender and data receiver are the same

⋆ Corresponding author.

entity or different entities sharing the same secret key. While in PSE, they are different entities and may not share a secret key. By comparison, SSE is more efficient than PSE, while PSE can be deployed in scenarios where it is infeasible to share a common secret key between different entities.

One of the drawbacks of traditional searchable encryption schemes, both SSE and PSE, is that they only support exact keyword search. This will affect system usability, since typos and format inconsistencies often occur when users input keywords. To enhance system usability, searchable encryption schemes with fuzzy keyword search capability [9] are necessary.

In the symmetric key setting, some efficient proposals have been designed for fuzzy keyword search over encrypted data [12,13,15,16]. However, in the public key setting, to the best of our knowledge, there have been only two researches on achieving this functionality [5,9]. In addition, we find that both schemes are not efficient for real applications. In this work, we construct and implement a homomorphic encryption based IPEFKS scheme, which achieves much higher efficiency compared with these two schemes.

1.2 Contributions

Our contributions in this paper are threefold.

1. We propose the primitive of IPEFKS that supports fuzzy keyword search over encrypted data and define its security requirement. We give a construction based on homomorphic encryption and prove that it is secure under the assumption that there exists an IND-CPA secure homomorphic encryption scheme.
2. We leverage a nice property of bilinear pairings in such a way that the cloud server can build an inverted index over encrypted data without knowing the encrypted keywords. Since an inverted index supports much more efficient search than a forward index when the number of documents is large, we may find uses for this approach in other applications.
3. We evaluate the efficiency of the homomorphic encryption based IPEFKS scheme by implementing the FV.SH homomorphic encryption scheme [8]. To show the efficiency advantage of IPEFKS over the other existing schemes, we implement the scheme proposed by Li *et al.* [9], which, to the best of our knowledge is the most efficient scheme among the existing ones. We name this scheme LWW-FKS for simplicity. The experimental results demonstrate that IPEFKS is much more efficient than it.

1.3 Organization

The rest of this paper is organized as follows. In section 2, we present the related work on fuzzy keyword searchable encryption. In section 3, we give the definition of IPEFKS and a concrete construction of it. In section 4, we show how to allow the cloud server to build an inverted index over encrypted data in a secure way. In section 5, we compare the efficiency of IPEFKS with LWW-FKS.

In section 6 we show security advantage of IPEFKS over LWW-FKS. Section 7 concludes this paper.

2 Related Work

Our work is built on fuzzy keyword searchable encryption. The most important related researches in this field are discussed below.

Fuzzy keyword search (FKS) over encrypted data in the public key setting is firstly proposed by Li *et al.* in [9]. Given a keyword q, FKS intends to find the encrypted documents containing keywords that are within a certain edit distance from q. The edit distance between two words w_1 and w_2 is the number of operations, including substitution, deletion and insertion, required to transform one of them into the other. Given a keyword, the basic idea of [9] is to enumerate all wildcard-based keywords that are within a predefined edit distance to it. Therefore, this scheme transforms a single fuzzy keyword search operation into several exact keyword search operations. The succeeding work done by Wang *et al.* [15] focuses on efficiency. They present a symmetric fuzzy keyword searchable encryption scheme with trie-traverse searching index in order to achieve high efficiency.

As discussed in [13], both schemes cited above can not use any other wildcard patterns than the ones prepared by the sender because the wildcard is realized by exact keyword search. The authors of [13] split a keyword into several characters and can achieve more splendid keyword matching patterns, e.g., wildcard search, partial matching. However, as for FKS, they give the same wildcard-based approach as [9]. Another recent work on symmetric FKS has been presented in [12]. The authors construct a symmetric similarity keyword search scheme using a fuzzy extractor and a protocol for computing Hamming distance. Actually, all above mentioned schemes do not support fuzzy keyword search in the real sense. The key issue to design FKS schemes is computing the edit distance between two encrypted keywords. To solve this problem, the authors of [16] and [5] share the common idea of translating metric space. Concretely, they translate edit distance into a new metric space with locality sensitive function family.

Except for LWW-FKS, the scheme proposed in [5] is the other FKS scheme in the public key setting among all others mentioned above. It is based on the embedding algorithm in [10] and the similarity search scheme in [6]. It uses PIS (Private Information Storage) [11] and PIR (Private Information Retrieval) [7], both of which are interactive cryptographic protocol with low efficiency, especially when used to store and retrieve documents consisting of a large number of bits. This makes this scheme inefficient for encrypted document retrieval.

Note that another research [18] sharing a similar name with ours solves the problem of keyword guessing attack but not the problem of fuzzy keyword search, which is the focus of this paper. In conclusion, the only somewhat practical FKS scheme in the public key setting is LWW-FKS. In this paper, we present a much more efficient FKS scheme than it.

3 IPEFKS

3.1 Preliminary

We denote vectors by lower-case bold italic letters, say $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, etc. We assume all keywords are strings of the same length, if this is not the case, we append some wildcard symbols (e.g., $*$) to the shorter ones.

The authors of [10] show that there exist $0 < \alpha < \beta < c_2$ and an embedding Ψ from $\{0,1\}^N$ with edit distance ed to $\{0,1\}^{c_2(log_2(1/\delta))}$ with Hamming distance $\mathcal{HD}$ such that:

- If $ed(\mathbf{x}, \mathbf{y}) \leq t$, then $\mathcal{HD}(\Psi(\mathbf{x}), \Psi(\mathbf{y})) \leq \alpha log_2(1/\delta)$.
- If $ed(\mathbf{x}, \mathbf{y}) \geq 2^{c_1(\sqrt{log_2 N log_2 log_2 N})} t$, then $\mathcal{HD}(\Psi(\mathbf{x}), \Psi(\mathbf{y})) \geq \beta log_2(1/\delta)$.

The above fact indicates that we can use Hamming distance to represent edit distance. Therefore, we focus on searchable encryption with Hamming distance as the distance measurement.

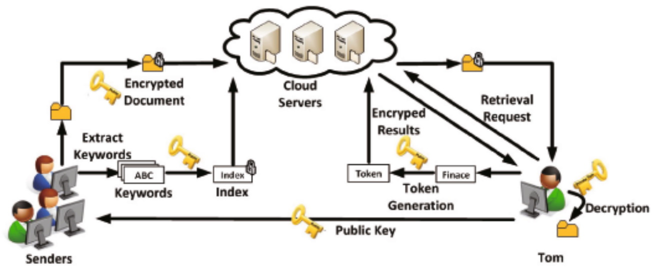

Fig. 1. Work flow of the IPEFKS scheme in the public key setting

3.2 Definition of IPEFKS

Definition 1. The IPEFKS scheme consists of the following five polynomial time algorithms, i.e., **KeyGen**, **IPEFKS**, **TokenGen**, **Search** and **Retrieve**, defined as follows:

- **KeyGen**(1^λ): Takes a security parameter λ as input, and generates a public/private key pair pk, sk.
- **IPEFKS**(pk, $\mathbf{w}$): Takes a public key pk and a keyword $\mathbf{w}$ as inputs, and produces a searchable encryption of $\mathbf{w}$.
- **TokenGen**(pk, $\mathbf{v}$): Given a public key pk and a query keyword $\mathbf{v}$, this algorithm produces a *token* $T_{\mathbf{v}}$.
- **Search**($T_{\mathbf{v}}$, $S_{\mathbf{w}}$): With a token $T_{\mathbf{v}} = \textbf{TokenGen}(pk, \mathbf{v})$ and a keyword ciphertext $S_{\mathbf{w}} = \textbf{IPEFKS}(pk, \mathbf{w})$, outputs an encrypted result $ER_{\mathbf{w}}$.
- **Retrieve**(sk, $ER_{\mathbf{w}}$, th): Given a private key sk, the encrypted result $ER_{\mathbf{w}}$ of the keyword $\mathbf{w}$, and a Hamming distance threshold th, outputs **YES** if $\mathcal{HD}(\mathbf{v}, \mathbf{w}) < th$, otherwise, outputs **NO**.

The work flow of the IPEFKS scheme is illustrated in **Figure.1**. The receiver Tom runs the **KeyGen** algorithm to generate his public/private key pair *pk*, *sk* and announces *pk* publicly. When a sender wants to send a document to Tom. She/he extracts the keywords of the document. Then the sender runs the **IPEFKS** algorithm to encrypt every keyword under Tom's public key, and these encrypted keywords form the encrypted index of the document. The index and the encrypted documents are then sent to the cloud server. When Tom wants to retrieve documents containing keywords within a certain Hamming distance threshold compared with his interested keyword "Finace" (i.e., Tom omits an 'n' of the keyword "Finance"), he runs the **TokenGen** algorithm to generate a *token* and sends it to the cloud server. Upon receiving this query request, the cloud server runs the **Search** algorithm and sends the encrypted results back to Tom, who then runs the **Retrieve** algorithm and returns the retrieval request back to the server. Finally, the encrypted documents are returned back to Tom. Tom uses his private key to decrypt the encrypted documents. Note that we do not specify how the documents are encrypted because it is unrelated with the search functionality.

We treat the cloud server as an honest-but-curious attacker in the security model and give a rigorous security definition as follows:

Definition 2. An IPEFKS scheme is semantically secure against an adaptive chosen keyword attack if every PPT (Probabilistic Polynomial Time) attacker has a negligible advantage in the following attack game.

1. **Setup**. The challenger runs the **KeyGen** algorithm and obtains a key pair (*pk*, *sk*) and sends *pk* to the attacker.
2. **Phase 1**. The attacker can ask for arbitrarily many keyword ciphertexts and query tokens from the challenger(the attacker does not know the associated keywords). In addition, the attacker can choose a query token and ask the challenger for the search results of the chosen query.
3. **Challenge**. At some point, the attacker sends the challenger two equal-length keywords $\mathbf{w}_0$, $\mathbf{w}_1$, on which it wants to be challenged. The challenger picks a random bit $b \in \{0, 1\}$ and gives the attacker the keyword ciphertext $C = \mathbf{IPEFKS}(pk, \mathbf{w}_b)$ and the token $T = \mathbf{TokenGen}(pk, \mathbf{w}_b)$.
4. **Phase 2**. The attacker can continue to ask for more keyword ciphertexts, query tokens and search results as in **Phase 1**.
5. **Guess**. The attacker outputs a guess bit b' for b. The advantage of an adversary $\mathcal{A}$ is defined to be $\mathrm{Adv}_{\mathcal{A}} = |\Pr[b = b'] - 1/2|$.

3.3 Construction

In this section, we give a construction of the IPEFKS scheme based on a homomorphic encryption scheme, named FV.SH encryption scheme [8]. Before presenting the concrete construction, we give some intuitions. The Hamming distance of two binary strings $\mathbf{x} = \mathbf{x}_1 \cdots \mathbf{x}_m$ and $\mathbf{y} = \mathbf{y}_1 \cdots \mathbf{y}_m$ is $\sum_{i=1}^{m}(\mathbf{x}_i \oplus \mathbf{y}_i)$.

Our construction leverages a nice property of the FV.SH encryption scheme, that is when the plaintexts are polynomials with coefficients in Z_2, the addition operation is equal to the XOR operation.

The FV.SH encryption scheme used in the following construction consists of four polynomial time algorithms, the key generation algorithm **KeyGen**, the encryption algorithm **Enc**, the decryption algorithm **Dec** and the ciphertext addition algorithm **Add**. Detailed description of these algorithms are given in section 5.1. We don't care about the ciphertext Multiplication algorithm, since it is unrelated with our work. The detailed construction is shown below.

- **KeyGen**(1^λ): Given a security parameter λ, generates a public/private key pair, pk and sk, of the FV.SH encryption scheme.
- **IPEFKS**(pk, $\mathbf{w}$): Given an m-bit keyword $\mathbf{w} = w_1 w_2 \cdots w_m$, encodes the keyword $\mathbf{w}$ as a polynomial $\mathbf{w}$ (we reuse the notation for simplicity) in $Z_2[x]/x^d + 1$, where d is a parameter in the FV.SH encryption scheme, and computes $S_\mathbf{w} = \mathbf{Enc}_{pk}(\mathbf{w})$.
- **TokenGen**(pk, $\mathbf{v}$): Given an m-bit keyword $\mathbf{v} = v_1 v_2 \cdots v_m$, encodes the keyword $\mathbf{v}$ as a polynomial $\mathbf{v}$ in $Z_2[x]/x^d + 1$ and computes $T_\mathbf{v} = \mathbf{Enc}_{pk}(\mathbf{v})$.
- **Search**($T_\mathbf{v}$, $S_\mathbf{w}$): Computes the encrypted result $ER_\mathbf{w}$ by adding $T_\mathbf{v}$ and $S_\mathbf{w}$ by running the **Add** algorithm of the FV.SH encryption scheme.
- **Retrieve**(sk, $ER_\mathbf{w}$, th): Decrypts $ER_\mathbf{w}$ by running $\mathbf{Dec}_{sk}(ER_\mathbf{w})$, adds the coefficients of $\mathbf{Dec}_{sk}(ER_\mathbf{w})$, and sets the result to $\mathcal{HD}(\mathbf{w}, \mathbf{v})$. If $\mathcal{HD}(\mathbf{w}, \mathbf{v}) < th$, outputs **YES**, otherwise outputs **NO**.

In the **IPEFKS** and **TokenGen** algorithm, we encode the keyword by taking each bit of the keyword as the coefficient of an $(m-1)$-degree polynomial.

3.4 Correctness and Security

Lemma 1. Given two binary vectors $\mathbf{w} = w_1 \cdots w_m$ and $\mathbf{v} = v_1 \cdots v_m$, the Hamming distance of $\mathbf{w}$ and $\mathbf{v}$ equals $\sum_{i=1}^{m} w_i \oplus v_i$, where the notation $\oplus$ denotes the XOR operation.

Correctness is guaranteed by the property of the FV.SH encryption scheme, i.e., when the keywords are encoded as polynomials in $Z_2[x]/x^d + 1$, the addition operation is equal to the XOR operation.

Theorem 1. The IPEFKS scheme is semantically secure against a chosen keyword attack (CKA) if the FV.SH homomorphic encryption scheme is IND-CPA secure.

Proof: Suppose there exists an adversary $\mathcal{A}$ that has a non-negligible advantage ϵ in breaking the IPEFKS. We show that there is an adversary $\mathcal{B}$ that can achieve the same advantage in winning a chosen plaintext attack (CPA) game for attacking the IND-CPA secure FV.SH encryption scheme. Therefore, we have a contradiction. We can conclude that $\mathcal{A}$ cannot exist and thus that IPEFKS is semantically secure against a chosen keyword attack (CKA).

In the chosen plaintext attack, where $\mathcal{C}$ is the challenger, the adversary $\mathcal{B}$ is supposed to give two messages m_0, m_1 to $\mathcal{C}$. It then receives an encryption $C_{IND-CPA} = \mathbf{Enc}_{pk}(m_b)$ from $\mathcal{C}$, where b is a random bit chosen by $\mathcal{C}$. $\mathcal{B}$ outputs a guess bit b' and wins if $b' = b$.

In the chosen keyword attack, $\mathcal{B}$ works as a simulator who acts as the challenger. The adversary $\mathcal{A}$ outputs two messages m_0 and m_1 ($m_0 \neq m_1$). $\mathcal{B}$ sends m_0 and m_1 to $\mathcal{C}$, who then returns a challenge $C_{IND-CPA} = \mathbf{Enc}_{pk}(m_b)$; $\mathcal{B}$ passes this challenge to $\mathcal{A}$. Finally $\mathcal{A}$ gives a guess b' for b, $\mathcal{B}$ outputs b' as its guess bit for the CPA game described above.

As for queries from $\mathcal{A}$ in Phase 1 and Phase 2, $\mathcal{B}$ maintains two lists, L_1, L_2, which are initially empty. The two lists consist of tuples $< \mathbf{w}_i, \mathbf{Enc}_{pk}(\mathbf{w}_i) >$. When $\mathcal{A}$ asks for a keyword ciphertext, $\mathcal{B}$ responds by sending $\mathbf{Enc}_{pk}(\mathbf{w})$ for a randomly chosen keyword $\mathbf{w}$ and appends the tuple $< \mathbf{w}, \mathbf{Enc}_{pk}(\mathbf{w}) >$ to the list L_1. When $\mathcal{A}$ asks for a token, $\mathcal{B}$ responds by sending $\mathbf{Enc}_{pk}(\mathbf{w})$ for a randomly chosen keyword $\mathbf{w}$ and appends the tuple $< \mathbf{w}, \mathbf{Enc}_{pk}(\mathbf{w}) >$ to the list L_2. When $\mathcal{A}$ sends back a token and asks for the search results, $\mathcal{B}$ searches for the token $\mathbf{Enc}_{pk}(\mathbf{w})$ and the corresponding keyword $\mathbf{w}$ in the list L_2 and then computes the Hamming distance between $\mathbf{w}$ and the keywords in the list L_1 and returns back $\mathbf{IPEFKS}(pk, \mathbf{w}_i)$ satisfying $\mathcal{HD}(\mathbf{w}, \mathbf{w}_i) \leq th$, where th is chosen by $\mathcal{B}$. If the token sent by $\mathcal{A}$ is not in the list L_2, then $\mathcal{B}$ returns $\perp$, and the CKA game exits.

The attacker $\mathcal{A}$ cannot distinguish whether it is interacting with a real searcher or the simulator $\mathcal{B}$ since the message distribution are the same. The advantage $\mathcal{B}$ gains in the CPA game is the same as that of $\mathcal{A}$ in the CKA game. Since FV.SH is IND-CPA secure, which indicates that the advantage gained by $\mathcal{B}$ must be negligible, we get a secure IPEFKS scheme.

4 Building an Inverted Index over Encrypted Data

In the construction described in section 3.3, we do not specify the index structure of the encrypted documents. When there are a large number of documents, inverted index structure is preferable to forward index structure [19]. In this section, we show how to enable the cloud server to build an inverted index over encrypted data in a secure way.

4.1 Preliminary

Bilinear pairings: Let $\mathbf{G}_1$ and $\mathbf{G}_2$ be two cyclic multiplicative group of prime order p, g be a generator of the group $\mathbf{G}_1$ and $e : \mathbf{G}_1 \times \mathbf{G}_1 \longrightarrow \mathbf{G}_2$ be a bilinear map between them. The map satisfies the following properties:

1.Computable: given $u, v \in \mathbf{G}_1$ there is a polynomial time algorithm to compute $e(u, v) \in \mathbf{G}_2$.

2.Bilinear: for any integers $x, y \in [1, p]$ we have $e(g^x, g^y) = e(g, g)^{xy}$.

3.Non-degenerate: $e(g, g)$ is a generator of $\mathbf{G}_2$.

4.2 Construction

The cloud server publishes the bilinear pairing parameters $(p, \mathbf{G}_1, \mathbf{G}_2, e, g)$. The cloud server and senders process as follows:
Sender: To send a document with identifier ID and keywords set $u(ID) = \{w_1, w_2, \cdots, w_n\}$, the sender processes as follows:

1. Gets the public parameters, including the public key pk of the receiver, the bilinear pairing parameters $(p, \mathbf{G}_1, \mathbf{G}_2, e, g)$ of the cloud server;
2. Generates a random number $r \in Z_p^*$ and evaluates $(CA_i, CB_i) = (g^r, w_i^r)$, where CA_i and CB_i is the corresponding term for the i^{th} keyword in the ciphertext, which will be used by the cloud server to build an inverted index without disclosing the keywords privacy.
3. Runs the keyword encryption algorithm **IPEFKS** to encrypt every keyword in the set $u(ID)$;
4. Sends $CA_i = g^r$, $CB_i = w_i^r$, ID, and the searchable encrypted keywords **IPEFKS**(pk, w_i) (for $i = 1, 2, \cdots, n$) together with the encrypted document to the cloud server.

Cloud Server: Assume that there exist m terms in the inverted index. The j^{th} term is $\big(SA_j = g^{r_j},\ SB_j = w_j^{r_j},\ \mathbf{IPEFKS}(pk, w_j),\ D_j\big)$, where D_j is the set of document identifiers such that every $d \in D_j$ contains the keyword w_j. Upon receiving a sending request from the sender, the cloud server processes as follows:
1. Decides whether w_i is equal to one of the m keywords by checking whether the equation $e(CA_i, SB_j) = e(SA_j, CB_i)$ is right for a certain j, where $j \in \{1, \cdots, m\}$;
2. Adds the document identifier ID to the set D_j if the equation $e(CA_i, SB_j) = e(SA_j, CB_i)$ holds for a certain j, otherwise establishes a new quadruple, i.e, $(SA_{m+1} = CA_i,\ SB_{m+1} = CB_i,\ \mathbf{IPEFKS}(pk, w_i),\ D_{m+1})$ for the new keyword w_i, where $D_{m+1} = \{ID\}$.

Correctness of this construction is based on the bilinear property of bilinear pairings and security relies on the fact that Diffie-Hellman problem on the $\mathbf{G}_2$ group is hard. Because of the constrained space, we do not give a rigorous security proof here.

5 Performance Analysis

In this section, we evaluate the efficiency of the IPEFKS scheme by implementing the recently proposed FV.SH encryption scheme and comparing with LWW-FKS. We use the notations illustrated in Table 1 in the remaining part of this paper.

Table 1. Notation Table

Notation	Description	Notation	Description
d	$d = 2^k$ for some k, and is the largest bit-length of the embedded keywords output by the embedding algorithm Ψ	q	the fixed modulo
$\chi = D_{\mathbb{Z}^d,\sigma}$	denotes the discrete Gaussian distribution with standard deviation σ over $\mathbb{R}$	t	the fixed modulo
R	$R = \mathbb{Z}/f(x)$, $\mathbb{Z}$ is the set of integers and $f(x) = x^d + 1$, R is the set of polynomials whose degree is not larger than d	$\lfloor x \rfloor$	rounding down
R_q	the set of polynomials in R with coefficients in $(-q/2, q/2]$	$\lfloor x \rceil$	rounding to the nearest integer
R_t	the set of polynomials in R with coefficients in $(-t/2, t/2]$	Δ	$\Delta = \lfloor q/t \rfloor$

5.1 Implementation of the FV.SH Encryption Scheme

In the FV.SH encryption scheme, ciphertexts consist of polynomials in the ring R_q. Plaintexts are polynomials in the ring R_t. d, q, t and σ are system parameters chosen in such a way that correctness and security are guaranteed. The FV.SH encryption scheme consists of the following algorithms:

- **FV.SH.KeyGen**(1^λ): Samples secret key $sk \xleftarrow{R} R_2$ uniformly. Samples $p_1 \xleftarrow{R} R_q$ uniformly and an error $e \xleftarrow{R} \chi$. Computes the public key $pk = (p_0 = [-(p_1 s + e)]_q, p_1)$
- **FV.SH.Enc**(pk, m): Samples $u \xleftarrow{R} R_2$, $e_1, e_2 \xleftarrow{R} \chi$ uniformly and returns $ct = (ct[0] = [p_0 u + e_1 + \Delta \cdot m]_q, ct[1] = [p_1 u + e_2]_q)$
- **FV.SH.Add**(ct_1, ct_2): Returns $([ct_1[0] + ct_2[0]]_q, [ct_1[1] + ct_2[1]]_q)$
- **FV.SH.Dec**(ct): Computes $\left[\left\lfloor \frac{[ct[0]+sk \cdot ct[1]]_q}{\Delta} \right\rceil\right]_t$

We set two tuples of parameters for the FV.SH encryption scheme. Both tuples provide 128 bits of security with distinguishing advantage 2^{-64}. In addition, some optimizations (e.g., using the bounded discrete Gaussian distribution to replace the real discrete Gaussian distribution) are taken. For interested readers, please refer to [8] for details. We implement the scheme in C using FLINT, namely Fast Library for Number Theory [21]. We test the code on an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz running Linux 3.2.0-34-generic x86_64. The efficiency of the FV.SH encryption scheme is shown in Table 2.

5.2 Efficiency Comparison

As we have shown in section 2, the proposal shown in [5] is not practical for encrypted document retrieval. As a result, in this section, we only need to compare IPEFKS with LWW-FKS. To cope with the scenario where there are a large number of documents, we use an inverted index here. The process of building an

Table 2. Timing in *ms* for FV.SH operations key generation, encryption, decryption and homomorphic addition in C. Parameters (P_1) and (P_2) have 128 bits of security with distinguishing advantage 2^{-64}.

Parameters	FV.SH.KeyGen	FV.SH.Enc	FV.SH.Dec	FV.SH.Add
(P_1) $d = 4096$ $\sigma = 16, q = 2^{128}$	80	33	7	3
(P_2) $d = 8192$ $\sigma = 8, q = 2^{128}$	153	60	15	6

inverted index over encrypted data is shown in section 4. Without loss of generality, we assume the average keyword length to be 10 and the number of typos in the searcher's inputs is smaller than 4, which we think is reasonable when the keyword length is 10. The exact keyword searchable encryption scheme used in LWW-FKS is PEKS [1], which takes the type A parameter in the PBC library [20]. In LWW-FKS, the computation cost of the search algorithm, keyword encryption algorithm and token/trapdoor generation algorithm is proportional to $O(10^t kn)$, while the computation cost in IPEFKS is only linear in kn, where kn denotes the keyword number and t denotes the edit distance threshold.

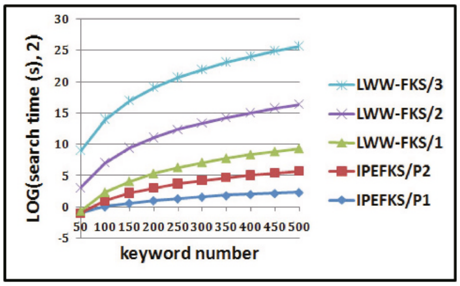

Fig. 2. Search time comparison between IPEFKS and LWW-FKS

Figure.2 shows the experimental results of the search time spent by the cloud server in processing a keyword search request. It can be seen that when the edit distance gets larger, the search time of LWW-FKS increases greatly, while the search time of IPEFKS is only related with the keyword number but not the edit distance.

Figure.3 and Figure.4 present the keyword encryption time and token(for IPEFKS)/trapdoor(for LWW-FKS) generation time respectively. When the edit distance gets larger, in LWW-FKS, more keyword encryptions are performed and more trapdoors are generated, thus leading to increasing time consumption. While, in IPEFKS, the edit distance does not affect these two processes, therefore only constant time is needed. The above experimental results demonstrate that IPEFKS is much more efficient than LWW-FKS.

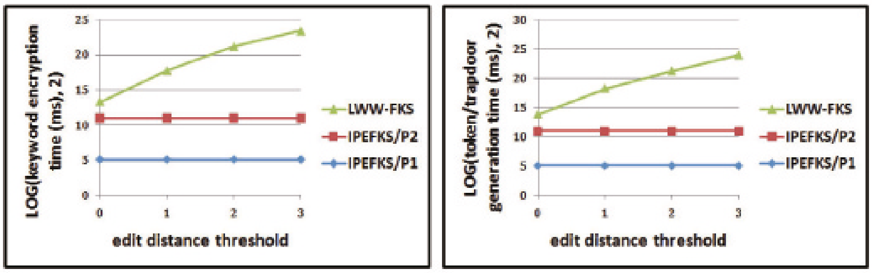

Fig. 3. Keyword encryption time and token/trapdoor generation time comparison

6 Security Analysis

IPEFKS is resistant against keyword guessing attack [17], which poses a great threat to clients' data privacy. While LWW-FKS suffers from this attack, because in LWW-FKS, cloud server has the capability of deciding whether a keyword is associated with the trapdoor received from the searcher independently. Specifically, given a token associated with a keyword W, the cloud server can encrypt a guessed keyword q and then runs the test algorithm of LWW-FKS to check whether $W = q$. When the cardinality of the keyword set is only polynomial in the security parameter, which is indeed the case in real applications, the cloud server can implement this attack successfully. In IPEFKS, the cloud server cannot test whether a keyword is associated with the trapdoor received from a searcher all by itself, because the **Retrieval** algorithm takes the secret key as an input, which is only known by the clients themselves.

7 Conclusion

In this paper, we present a new primitive, named IPEFKS, to solve the problem of fuzzy keyword search over encrypted data. By comparison, IPEFKS is not only more efficient but also more secure than all existing schemes. Moreover, to enable the cloud server to build an inverted index over encrypted data, we propose an approach, which leverages a nice property of bilinear pairings. To the best of our knowledge, we are the first to implement the FV.SH encryption scheme in C, which we think may be of independent interest for other works. Future work includes enhancing the efficiency of the FV.SH encryption scheme by using SIMD operations and other optimizations and giving experimental results on the real datasets by implementing the prototype of the system.

Acknowledgment. This work is partially supported by the HGJ National Significant Science and Technology Projects under Grant No. 2012ZX01039-004-009, Key Lab of Information Network Security, Ministry of Public Security under Grant No.C11606.

References

1. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
2. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and Efficiently Searchable Encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
3. Goh, E.J.: Secure indexes. IACR Cryptology ePrint Archive 2003, 216 (2003)
4. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searcheson encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
5. Bringer, J., Chabanne, H.: Embedding edit distance to enable private keyword search. Human-centric Computing and Information Science 2(2) (2012)
6. Bringer, J., Chabanne, H., Kindarji, B.: Error-tolerant searchable encryption. In: Proceedings of the 2009 IEEE International Conference on Communications (2009)
7. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
8. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive 2012, 144 informal publication (2012)
9. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: INFOCOM 2010, pp. 441–445 (2010)
10. Ostrovsky, R., Rabani, Y.: Low distortion embeddings for edit distance. J. ACM 54(5) (October 2007)
11. Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC 1997, pp. 294–303. ACM, New York (1997)
12. Pang, X., Yang, B., Huang, Q.: Privacy-preserving noisy keyword search in cloud computing. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 154–166. Springer, Heidelberg (2012)
13. Suga, T., Nishide, T., Sakurai, K.: Secure keyword search using bloom filter with specified character positions. In: Takagi, T., Wang, G., Qin, Z., Jiang, S., Yu, Y. (eds.) ProvSec 2012. LNCS, vol. 7496, pp. 235–252. Springer, Heidelberg (2012)
14. Tang, Q.: Search in encrypted data: Theoretical models and practical applications. IACR Cryptology ePrint Archive 2012, 648 (2012)
15. Wang, C., Ren, K., Yu, S., Urs, K.M.R.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: INFOCOM, pp. 451–459 (2012)
16. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, ICDE 2012. IEEE Computer Society, Washington, DC (2012)
17. Yau, W.-C., Heng, S.-H., Goi, B.-M.: Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In: Rong, C., Jaatun, M.G., Sandnes, F.E., Yang, L.T., Ma, J. (eds.) ATC 2008. LNCS, vol. 5060, pp. 100–105. Springer, Heidelberg (2008)
18. Xu, P., Jin, H., Wu, Q., Wang, W.: Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. IEEE Transactions on Computers 99(PrePrints), 1 (2012)
19. Information Retrieval: Implementing and Evaluating Search Engines. MIT Press, Cambridge (2010) ISBN 978-0-262-02651-2
20. Lynn, B.: Pairing-Based Cryptography Library, `http://crypto.stanford.edu/pbc/`
21. Hart, W.: FLINT: Fast Library for Number Theory, `http://www.flintlib.org`