6-2017

Flexible wildcard searchable encryption system

Yang YANG

Ximeng LIU
Singapore Management University, xmliu@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Jian WENG

DOI: https://doi.org/10.1109/TSC.2017.2714669

Published in IEEE Transactions on Services Computing, June 2017, Advance online
https://doi.org/10.1109/TSC.2017.2714669

Flexible Wildcard Searchable Encryption System

Yang Yang, *Member, IEEE,* Ximeng Liu, *Member, IEEE,* Robert H. Deng, *Fellow, IEEE*, Jian Weng

Abstract—Searchable encryption is an important technique for public cloud storage service to provide user data confidentiality protection and at the same time allow users performing keyword search over their encrypted data. Previous schemes only deal with exact or fuzzy keyword search to correct some spelling errors. In this paper, we propose a new wildcard searchable encryption system to support wildcard keyword queries which has several highly desirable features. First, our system allows multiple keywords search in which any queried keyword may contain zero, one or two wildcards, and a wildcard may appear in any position of a keyword and represent any number of symbols. Second, it supports simultaneous search on multiple data owner's data using only one trapdoor. Third, it provides flexible user authorization and revocation to effectively manage search and decryption privileges. Fourth, it is constructed based on homomorphic encryption rather than Bloom filter and hence completely eliminates the false probability caused by Bloom filter. Finally, it achieves a high level of privacy protection since matching results are unknown to the cloud server in the test phase. The proposed system is thoroughly analyzed and is proved secure. Extensive experimental results indicate that our system is efficient compared with other existing wildcard searchable encryption schemes in the public key setting.

Index Terms—searchable encryption, wildcard search, user revocation, multiple users, top-k.

1 Introduction

Cloud storage [1] provides subscribers ubiquitous, dynamic, scalable and on-demand storage service. While cloud storage brings more convenience and benefit than ever, it also introduces significant security and privacy threats to customers' data [2]. To ensure privacy of the outsourced data in the untrusted public cloud, data encryption is an effective way to prevent inside/outside adversaries from accessing the sensitive information. Meanwhile, it is necessary to support data retrieval function (without decryption) on encrypted data to facilitate the data usage. Such mechanism is referred to as searchable encryption (SE).

Consider an electronic health record (EHR) storage system as an application example of SE. Suppose data user Alice wants to store her sensitive EHR in the public health cloud. She firstly extracts a set of keywords describing the EHR and encrypts these keywords using SE encryption algorithm to build an index. Then, the keyword encryption index and the encrypted EHR are uploaded to the cloud. In the data retrieval phase, a data user Bob who is Alice's doctor or relative generates a trapdoor to make a keyword query. The cloud uses the trapdoor to search on the encrypted EHRs and returns the matching files to Bob. In the search process, no plaintext information about the keyword and EHR should be leaked to the cloud.

In 2004, Boneh et al. [3] put forth the concept of public key encryption with keyword search (PEKS) to enable keyword query over encrypted data. In order to share the query authority among multiple users, different access control methods were adopted to searchable encryption, such as proxy re-encryption [4] and attribute based encryption (ABE) [5], [6]. A few fuzzy keyword search schemes [11]-[15] were proposed to tolerate certain spelling errors. However, these schemes can only make exact keyword search or edit distance based similarity search.

To achieve flexible search functions, the concept of wildcard searchable encryption was proposed [16] to enable wildcard keyword search. In the query phase, a data user queries a keyword containing a wildcard, which may represent one or more symbols. For example, Alice's doctor Bob may use the keyword "05/**/2016" to search for all Alice's EHRs created during the month of May of 2016 and use the keyword "*ache" to search for Alice's EHRs containing "headache", "stomachache" or "heartache". However, most of the existing wildcard SE solutions in the literature are constructed based on Bloom filter [14]. A notable drawback of Bloom filter is that false-positive probability is inevitable. These Bloom filter based wildcard searchable encryption schemes [17], [18], [19] return false results to users with a non-negligible probability. Moreover, these schemes are built in symmetric key setting; hence, a data owner has to reveal her private key to authorize the search privilege to other users and the authorization is non-revokable.

1.1 Related Work

The problem of secure search on encrypted database was studied by Jarecki et al. [7], which supports arbitrary boolean queries. Later, Sepehri et al. proposed a new privacy-preserving query processing method on partitioned database based on the multi-party compu-

Y. Yang and X. Liu are with College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China, 350116; and School of Information Systems, Singapore Management University, Singapore 188065 (email: yang.yang.research@gmail.com, snbnix@gmail.com.)

R. H. Deng is with School of Information Systems, Singapore Management University, Singapore 188065 (email: robertdeng@smu.edu.sg).

Y. Yang is also with Fujian Provincial Key Laboratory of Information Processing and Intelligent Control (Minjiang University), Fuzhou, China, 350121.

X. Liu is the corresponding author.

tation methodology [8], and a high scalable proxy re-encryption scheme with secure equality queries [9]. Sun et al. [10] put forth a multi-client searchable encryption scheme over database and support boolean queries. Except for these searchable encryption schemes on database, the secure keyword query over non-structural encrypted data is also investigated.

Li et al. [11] proposed a fuzzy keyword search scheme over encrypted data for cloud computing. They exploited edit distance to measure keywords similarity and design two methods to construct fuzzy keyword sets. A symbol-based tree was used to accelerate the search algorithm. In 2013, Li et al. [12] extended their scheme [11] to the multiple user scenario utilizing ABE encapsulation. In 2014, Wang et al. [13] proposed a multi-keyword fuzzy searchable encryption scheme utilizing Bloom filter [14] and locality sensitive hash function. The scheme tolerates small edit distance errors and support multi-keyword search. Fu et al. [15] improved the accuracy so that more spelling mistakes can be tolerated.

The limitation of fuzzy searchable encryption scheme is that only small edit distance errors, such as spelling errors, can be corrected. It is almost useless if the query keyword has a large edit distance from the exact keyword. In 2010, Sedghi et al. [16] constructed a searchable encryption scheme with wildcards in public key setting and based on bilinear pairing. Hidden vector encryption (HVE) is the core component of Sedghi's scheme, which derives from identity based encryption. In their construction, the position of the wildcards need to be specified and each wildcard represents only one character. It requires a large amount of modulo exponentiation operations in its encryption, trapdoor generation and test algorithms. The test algorithm also needs several time consuming bilinear pairing operations.

In 2011, Bosch et al. [17] introduced a conjunctive wildcard searchable encryption scheme in symmetric key setting. Pseudo random function and Bloom filter [14] were utilized to construct the scheme. It has improved efficiency than Sedghi's scheme. However, it merely enumerates all the commonly used keywords that the wildcard keyword can represent from the lexicon. Then, these expanded keywords are all inserted into a Bloom filter. The method has limited applicability since not all the keywords can be extracted from the lexicon, such as chemical formulas, biological product and abbreviation expressions.

In 2012, Suga et al. [18] proposed a wildcard searchable encryption scheme based on Bloom filter, in which each keyword has its own Bloom filter. The storage overhead grows with the number of extracted keywords from the document. The disadvantage of the scheme is that one wildcard can only represent one character. For instance, if a user desires to search all keywords that begin with "acid", he has to submit the trapdoors for wildcard keywords "acid$\star\star$", "acid$\star\star\star$" and "acid$\star\star\star\star\star\star\star\star$" respectively so that the keywords "acidic", "acidity" and "acidification" can be matched. To overcome this problem, Hu et al. [19] introduced an improved scheme such that one wildcard can represent any number of characters. Hu's scheme is constructed based on Suga's scheme [18] but utilizes a different method to insert a keyword into Bloom filter.

A serious drawback of Bloom filter based searchable encryption schemes [17], [18], [19] is the inevitability of false probability. Bloom filter [14] is a data structure to efficiently determine the membership of an element, which is represented by an array of m bits and initially set to 0. It needs r independent hash functions ($h_t : \{0,1\}^* \rightarrow [1,m],\ 1 \leq t \leq r$), each of which maps an element into one of the m positions. Each element in set $S = \{s_1, \cdots, s_n\}$ is mapped to the Bloom filter BF. To determine whether an element a belongs to S, one should check whether all the bits at positions $h_t(a),\ 1 \leq t \leq r$ are set to 1 in BF. If not, a is not a member of S. Otherwise, a is possible to be a member of S. Bloom filter has false-positive probability due to the reason that each position in BF may be set to 1 by one or more other elements. The false-positive probability is $f_p = [1-(1-1/m)^{rn}]^r \approx e^{-m/n} ln(p) ln(1-p)$, where $p = (1-1/m)^{rn} \approx e^{-rn/m}$. The false-positive probability grows with n/m.

1.2 Contribution

In this paper, to address the above problems, we propose a flexible wildcard searchable encryption scheme supporting multiple users. It is constructed in public key setting without relying on Bloom filter, is efficient, and achieves high security level. The main contributions of this paper are listed below.

- ***Flexible wildcard representation***. In some existing wildcard searchable encryption schemes [16], [18], the wildcard only represents a single symbol. In our system, a wildcard can represent any number of symbols. A data user can use "acid$\star$" to search all the derivative words of "acid". Moreover, up to two wildcards are supported in our system and the wildcards may appear in any positions of the keyword. We note that keywords with more than two wildcards are seldomly used in practice.
- ***Flexible search function***. Our system is the first wildcard SE which allows a data user to use one trapdoor to simultaneously search on multiple data owner's files. For example, a medical doctor can issue one wildcard keyword query to simultaneously search over multiple patients' encrypted EHRs. Moreover, in the search algorithm, the user can use multiple keyword to generate one trapdoor. These query keywords may contain zero, one or two wildcards. The user can issue "AND" or "OR" queries on these keywords and the top-k documents that have the highest relevance scores is returned to the user.
- ***Flexible user authorization and revocation***. Our system enables a data owner to authorize her research

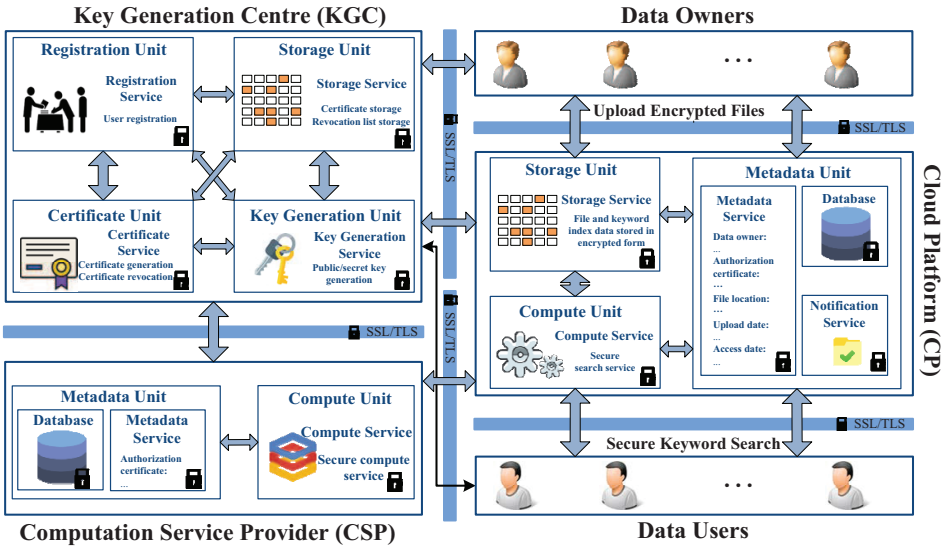


Fig. 1: Cloud based System Architecture

privilege to other users within a pre-defined period of time and automatically revoke these users when the authorization period expires.

- *No false probability*. Most of the existing fuzzy or wildcard searchable encryption schemes are constructed based on Bloom filter [17], [18], [19], [13], [15] and hence suffer from false-positive probability. Our system guarantees that all returned files are exact match files.

We conduct extensive experiments on the proposed system to evaluate its computation and communication performances. The simulation results affirm that this system is efficient and practical compared with existing schemes in the public key setting.

2 System Architecture and Security Model

2.1 Cloud Based System Architecture

Fig. 1 shows the cloud based system architecture. The entities and operations are introduced below.

- **Key generation center (KGC)** is fully trusted and tasked to manage and distribute public/secret keys in the system, which includes user registration, key generation, certificate and storage units. The registration unit provides registration service for the system users; the key generation unit generates public/secret keys for the system and the users; the certificate unit provides certificate generation and revocation services; and the storage units stores the certificates and revocation list.
- **Cloud platform (CP)** stores users' encrypted files and is responsible to execute data retrieval operations, which includes the storage, process and metadata units. The storage unit provides storage service, which stores the encrypted files and the secure keyword indexes. The process unit provides process service, which operates secure search. The metadata unit provides metadata service, and the metadata includes the information of the data owner, authorization certificates, file location, file upload and access date. The metadata server also provides notification services to the system users, which is a separate service dedicated to monitor if changes have been made to data owners' accounts. These information are stored in the database and mainly used to provide better service to the customers. These three types of units interact with each other to provide services.
- **Computing service provider (CSP)** has powerful computation resources, which includes the compute and metadata units. The compute unit of CSP interacts with that in CP to perform secure calculations; and the metadata unit stores the authorization certificates for the CSP to operate authorized computations. It is assumed that CP and CSP do not collude with each other.
- ***Data owner*** encrypts keywords and files, and sends them to CP for storage.
- ***Data user*** creates keyword trapdoors which are used by the CP to search on encrypted data.
- The secure socket layer (SSL) or transport layer security (TLS) protocols are made use to secure all communications between CP and CSP, data owner and CP, data user and CP, and the KGC with other parties. The SSL/TLS protocols aim primarily to provide the privacy and data integrity between two communicating entities.

Cloud computing provides various kinds of services to the customers (shown in Fig. 2), such as infrastructure-as-a-service (IaaS), platform-as-a service (PaaS) and software-as-a-service (SaaS). The information, process and storage services belong to PaaS. To provide security protection to the cloud, security services arouses wide

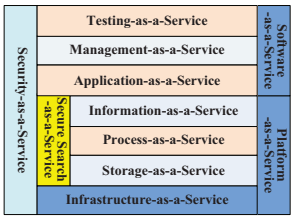


Fig. 2: Cloud Service Architecture

attention and becomes a hot research topic [20]. Secure search, the problem addressed in this paper, is one of the security services in the cloud and regarded as secure search-as-a-service [21].

2.2 Attack Model

We follow the attack model in [22], [23]. We assume that KGC is a fully trusted entity, CP and CSP are "honest-but-curious" who are honest to execute the protocols but curious with the plaintext of user data. An adversary $\mathcal{A}^*$ is defined in this attack model whose purpose is to recover the plaintext of data owner's privacy-preserving documents and data user's retrieved results. $\mathcal{A}^*$ has the following capabilities.

(1) $\mathcal{A}^*$ could *eavesdrop* all communications.
(2) $\mathcal{A}^*$ could *compromise* CP and try to get the plaintext from the encrypted files sent by the data owner and CSP.
(3) $\mathcal{A}^*$ could *compromise* CSP and try to obtain the plaintext from the ciphertext sent by CP in the interactive protocol.
(4) $\mathcal{A}^*$ could *compromise* data owner or data user (except the challenge user) and get their privileges with the aim of getting the challenge user's plaintext information.

However, the attacker $\mathcal{A}^*$ is not allowed to compromise: (1) CP and CSP simultaneously, and (2) the challenge user. These are typical restrictions in cryptographic protocols [24].

2.3 Security Model

The security model in [25], [26] is adopted in this work. Consider three parties: system user (a.k.a "D_1"), CP (a.k.a "S_1") and CSP (a.k.a "S_2"), where system user includes data owner and user. We construct three simulators $(Sim_{D_1}, Sim_{S_1}, Sim_{S_2})$ to against three types of attackers $(\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ that corrupt D_1, S_1 and S_2, respectively. These attackers are deemed as non-colluding and semi-honest. Due to the page limitation, please refer to [25], [26], [27] for the detailed definition of the security model.

3 Crypto Primitives and Protocols

3.1 Paillier Cryptosystem with Threshold Decryption

The Paillier cryptosystem [28] with threshold decryption (PCTD) in [27], [29] is utilized for data encryption and for preventing the secret key leakage risk in this paper. It is a non-deterministic encryption system and based on the decisional composite residuosity assumption. The Paillier system could provide the confidentiality of the outsourced data in the cloud platform, and also realizes the homomorphic properties. In that way, various calculations can be directly operated on the ciphertext without decryption to realize secure outsourced calculation. Moreover, it has lower computation overhead than fully homomorphic encryption systems. Let $\mathcal{L}(X)$ denote the bit length of X.

KeyGen: Let κ be the security parameter and p, q be two large prime numbers such that $\mathcal{L}(p) = \mathcal{L}(q) = \kappa$. Let $N = pq$ and $\lambda = lcm(p-1, q-1)/2$ [1]. Define a function $L(x) = \frac{x-1}{N}$ and select a generator g of order $ord(g) = (p-1)(q-1)/2$. The system public parameter is $PP = (g, N)$. The master secret key of the system is $SK = \lambda$. A user i in the system is assigned a secret key $sk_i \in Z_N$ and a public key $pk_i = g^{sk_i} \mod N^2$.

Encryption: On input a plaintext $m \in Z_N$, a user randomly selects $r \in [1, N/4]$ and uses his public key pk_i to encrypt m to ciphertext $[m]_{pk_i} = (C_1, C_2)$, in which $C_1 = pk_i^r(1+mN) \mod N^2$ and $C_2 = g^r \mod N^2$.

Decryption with sk_i: On input ciphertext $[m]_{pk_i}$ and secret key sk_i, the message can be recovered by computing $m = L(C_1/C_2^{sk_i} \mod N^2)$.

Decryption with master secret key: Using master secret key $SK = \lambda$ of the system, any ciphertext $[m]_{pk_i}$ encrypted by any public key can be decrypted by computing $C_1^\lambda = (pk_i^r)^\lambda(1+mN\lambda) = (1+mN\lambda) \mod N^2$. Since $gcd(\lambda, N) = 1$ holds [2], we have $m = L(C_1^\lambda \mod N^2)\lambda^{-1} \mod N$.

Master secret key splitting: The master secret key $SK = \lambda$ can be randomly split into two parts $SK_1 = \lambda_1$ and $SK_2 = \lambda_2$ such that $\lambda_1 + \lambda_2 \equiv 0 \mod \lambda$ and $\lambda_1 + \lambda_2 \equiv 1 \mod N^2$.

Partial Decryption with SK_1 (PD1): On input the ciphertext $[m]_{pk_i} = (C_1, C_2)$, we can use $SK_1 = \lambda_1$ to compute $C_1^{(1)} = (C_1)^{\lambda_1} = (pk_i^r)^{\lambda_1}(1+mN\lambda_1) \mod N^2$.

Partial Decryption with SK_2 (PD2): On input $[m]_{pk_i}$ and $C_1^{(1)}$, we can use $SK_2 = \lambda_2$ to compute $C_1^{(2)} = (C_1)^{\lambda_2} = (pk_i^r)^{\lambda_2}(1+mN\lambda_2) \mod N^2$. The message can be recovered by computing $m = L(C_1^{(1)} \cdot C_1^{(2)})$.

Ciphertext Refresh (CR): CR algorithm is utilized to refresh a ciphertext $[m]_{pk_i} = (C_1, C_2)$ to a new ciphertext $[m']_{pk_i} = (C_1', C_2')$ such that $m = m'$. It selects a random $r' \in Z_N$ and calculates $C_1' = C_1 \cdot h_i^{r'} \mod N^2, C_2' = C_2 \cdot g^{r'} \mod N^2$.

It is easy to verify that PCTD is **additive homomorphic**: $[m_1]_{pk_i} \cdot [m_2]_{pk_i} = [m_1 + m_2]_{pk_i}$ and $([m]_{pk_i})^r = [r \cdot m]_{pk_i}$ for random $r \in Z_N$.

The following protocols in [27] are utilized in the proposed system. Let pk_A and pk_B be the public keys of users A and B. pk_Σ is an public key that will be defined later.

1. lcm : lowest common multiple.
2. gcd: greatest common divider.

Secure Addition Protocol across Domains (**SAD**): Given $[X]_{pk_A}$ and $[Y]_{pk_B}$, **SAD** protocol securely calculates $[X+Y]_{pk_\Sigma}$.

Secure Multiplication Protocol across Domains (**SMD**): Given $[X]_{pk_A}$ and $[Y]_{pk_B}$, **SMD** protocol securely calculates $[X \cdot Y]_{pk_\Sigma}$.

Secure Less Than Protocol across Domains (**SLT**): Given $[X]_{pk_A}$ and $[Y]_{pk_B}$, **SLT** protocol securely calculates $[u]_{pk_\Sigma} = \mathbf{SLT}([X]_{pk_A}, [Y]_{pk_B})$, where $u = 1$ if $X < Y$ and $u = 0$ if $X \geq Y$.

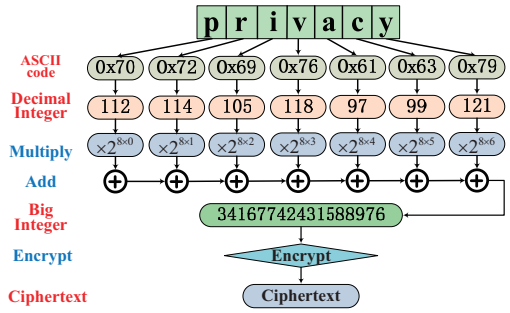


Fig. 3: Example of **K2C** operation

3.2 Keyword Representation and Encryption

In order to encode a keyword to an element in Z_N, we firstly transform each character in the keyword to its ASCII code. Then, the hexadecimal ASCII code is converted to a decimal. According to the position of the character in the keyword, each decimal is multiplied with a weight. Then, these weighted decimal numbers are added together and encrypted using PCTD algorithm. This algorithm is named as keyword to ciphertext algorithm (**K2C**) and illustrated in Figure 3.

3.3 Secure Greater or Equal Protocol (SGE)

Given two encrypted numbers $[X]_{pk_A}$ and $[Y]_{pk_B}$ with $X, Y \geq 0$, the goal of **s**ecure **g**reater or **e**qual protocol (**SGE**) is to obtain an encrypted data $[u^*]_{pk_\Sigma}$ to show the relationship between X and Y (i.e. $X \geq Y$ or $X < Y$). We require $\mathcal{L}(X), \mathcal{L}(Y) < \mathcal{L}(N)/4$ to make the protocol properly work. The description of the **SGE** protocol is as follows.

Step 1: CP calculates $[X_1]_{pk_A} = ([X]_{pk_A})^2 \cdot [1]_{pk_A} = [2X+1]_{pk_A}$, $[Y_1]_{pk_B} = ([Y]_{pk_B})^2 = [2Y]_{pk_B}$.

CP chooses random number r_1, r_2, s.t. $\mathcal{L}(r_1) < \mathcal{L}(N)/4 - 1$, $\mathcal{L}(r_2) < \mathcal{L}(N)/8$. Then, CP flips a coin $s \in \{0,1\}$ randomly. CP and CSP jointly execute the following operations.

If $s = 1$, $[\gamma]_{pk_\Sigma} \leftarrow \mathbf{SAD}(([X_1]_{pk_A})^{r_1}, ([Y_1]_{pk_B})^{N-r_1})$.

If $s = 0$, $[\gamma]_{pk_\Sigma} \leftarrow \mathbf{SAD}(([Y_1]_{pk_B})^{r_1}, ([X_1]_{pk_A})^{N-r_1})$.

Then, CP computes $l = [\gamma]_{pk_\Sigma} \cdot [r_2]_{pk_\Sigma}$ and $l' = \mathbf{PD1}_{SK_1}(l)$ and sends (l, l') to CSP.

Step 2: CSP decrypts $l'' = \mathbf{PD2}_{SK_2}(l, l')$. If $\mathcal{L}(l'') > \mathcal{L}(N)/2$, CSP denotes $u' = 0$ and $u' = 1$ otherwise. Then, CSP uses pk_Σ to encrypt u', and sends $[u']_{pk_\Sigma}$ to CP.

Step 3: Once $[u']_{pk_\Sigma}$ is received, CP computes as follows: if $s = 1$, CP denotes $[u^*]_{pk_\Sigma} = \mathbf{CR}([u']_{pk_\Sigma})$; otherwise, CP computes $[u^*]_{pk_\Sigma} = [1]_{pk_\Sigma} \cdot ([u']_{pk_\Sigma})^{N-1} = [1-u']_{pk_\Sigma}$. If $u^* = 1$, it indicates $X \geq Y$; $u^* = 0$ otherwise.

3.4 Encrypted Keyword Equivalence Testing Protocol

Given two encrypted keywords $[X]_{pk_A}$ and $[Y]_{pk_B}$ $(X, Y \geq 0)$, a secure keyword equivalent test protocol across domains (**KET**) outputs an encrypted result $[u^*]_{pk_\Sigma}$ to indicate whether two keywords are the same. We restrict $\mathcal{L}(X), \mathcal{L}(Y) < \mathcal{L}(N)/4$ to make the protocol properly work. CP and CSP jointly computes $[u_1]_{pk_\Sigma} = \mathbf{SGE}([X]_{pk_A}, [Y]_{pk_B})$, $[u_2]_{pk_\Sigma} = \mathbf{SGE}([Y]_{pk_B}, [X]_{pk_A})$, $[u]_{pk_\Sigma} = \mathbf{SMD}([u_1]_{pk_\Sigma}, [u_2]_{pk_\Sigma})$. If $u^* = 1$, it indicates that the two keywords are the same. Otherwise, $u^* = 0$.

4 Wildcard Search

4.1 Important Tools

In this subsection, we introduce the important tools used to realize wildcard searchable encryption. These tools play the role of "magic scissor" to securely partition ciphertext into several encrypted parts according to our requirements.

A.1. Secure Multi-Bit Extraction Protocol (**MBE**)

Given a ciphertext $[X]$ and a positive integer ϖ ($X \in Z_N$, $\varpi < \mathcal{L}(N)$), the goal of secure **m**ulti-**b**it **e**xtraction protocol (**MBE**) is to calculate $[x]$ such that x is the least significant ϖ-bit of the X's bit representation. The **MBE** protocol is shown as below.

(1) CP selects $r \in_R Z_N$, calculates $Y = [X] \cdot [r]$, $Y' = \mathbf{PD1}_{SK_1}(Y)$ and sends (Y, Y') to CSP.
(2) CSP calculates $y = \mathbf{PD2}_{SK_2}(Y, Y')$, $y_1 = (y \mod 2^\varpi) + 2^\varpi$ and sends $[y_1]$ to CP.
(3) CP calculates $r_1 = (r \mod 2^\varpi)$, $[x'] = [y_1] \cdot [r_1]^{N-1}$.
(4) CP and CSP jointly computes $[u] = \mathbf{SGE}([x'], [2^\varpi])$.
(5) Then, CP computes $[x] = [x'] \cdot [u]^{N-2^\varpi}$.

Next, we illustrate the **MBE** protocol.

(1) To protect the privacy of X, CP firstly randomizes X with a random number $r \in Z_N$ by calculating $Y = [X] \cdot [r] = [X + r]$.
(2) After decryption, CSP gets $y = X + r \mod N$. Since $X, r \in Z_N$ and r is randomly selected, CSP can not deduce any information of X from y. Then, CSP computes $(y \mod 2^\varpi)$ to get the least ϖ bits of y. The purpose of adding 2^ϖ is to deal with the carry bit of $(X \mod 2^\varpi) + (r \mod 2^\varpi)$, which will be explained in detail later.
(3) r_1 is the least ϖ bits of r. CP calculates $[x'] = [y_1] \cdot [r_1]^{N-1} = [(y \mod 2^\varpi) - (r \mod 2^\varpi) + 2^\varpi]$.
(4) If $(y \mod 2^\varpi) \geq (r \mod 2^\varpi)$, we have $x' \geq 2^\varpi$ and $[u] = \mathbf{SGE}([x'], [2^\varpi]) = [1]$.
If $(y \mod 2^\varpi) < (r \mod 2^\varpi)$, we have $x' < 2^\varpi$ and $[u] = \mathbf{SGE}([x'], [2^\varpi]) = [0]$.

(5) If $(y \mod 2^{\varpi}) \geq (r \mod 2^{\varpi})$, it indicates that $(X \mod 2^{\varpi}) + (r \mod 2^{\varpi}) = (y \mod 2^{\varpi})$. Then, $[x] = [(y \mod 2^{\varpi}) - (r \mod 2^{\varpi})] = [X \mod 2^{\varpi}]$.
If $(y \mod 2^{\varpi}) < (r \mod 2^{\varpi})$, it indicates that $(X \mod 2^{\varpi}) + (r \mod 2^{\varpi}) = (y \mod 2^{\varpi}) + 2^{\varpi}$. Then, $[x] = [(y \mod 2^{\varpi}) - (r \mod 2^{\varpi}) + 2^{\varpi}] = [X \mod 2^{\varpi}]$.

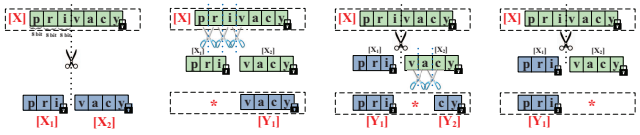
*A.2. Secure Ciphertext Partition Protocol (**SCP**)*

In the wildcard searchable encryption construction, we need to partition a ciphertext $[X]$ into two encrypted parts $[X_1], [X_2]$ without revealing the plaintext.

Given $[X]$ and a positive integer ϖ ($X \in Z_N$, $\varpi < \mathcal{L}(N)$), the goal of **s**ecure **c**iphertext **p**artition protocol (**SCP**) is to calculate $[X_1], [X_2]$ such that X_1 is the least significant ϖ-bit of the X's bit representation and $X = X_1 + X_2 \times 2^{\varpi}$. The **SCP** protocol is shown as below.

(1) CP and CSP jointly calculate $[X_1] = \mathbf{MBE}([X], \varpi)$;
(2) CP computes $Z = [X] \cdot [X_1]^{N-1}$ and $[X_2] = Z^a \mod N$, where $a = (2^{\varpi})^{-1}\ (mod\ N)$.

An example of **SCP** protocol is shown in Figure 4(a). If $[X]$ is a ciphertext of keyword "privacy" and $\varpi = 8 \times 3 = 24$, the $\mathbf{SCP}([X], \varpi)$ will output $[X_1], [X_2]$ such that $[X_1]$ is the encryption of "pri" and $[X_2]$ is the encryption of "vacy". The **SCP** protocol acts as a "magic scissor" to securely cut the encrypted keyword to two encrypted strings.



(a) **SCP** (b) **FW** (c) **MW** (d) **BW**

Fig. 4: Examples of **SCP**, **FW**, **MW**, and **BW** protocols

4.2 Single Wildcard Search

Here, we begin to deal with the keyword search with single wildcard over encrypted data, which can present any number of symbols. Since the single wildcard could appear in the front, in the middle or in the back of the string, we design three protocols to handle these situations. These protocols are executed by CP and CSP. Both of them do not know the plaintext of the result.

Suppose a document of data owner A contains a keyword X and encrypted to $[X]_{pk_A}$. The data user B generates a keyword with wildcard to issue a query. The following protocols will test whether the submitted encrypted query matches $[X]_{pk_A}$. Let $\star$ represent the wildcard. When the wildcard appears in the front or in the middle of the keyword, data user B specifies a positive integer ν_1 to be the maximum character number that the wildcard can be substituted. When the wildcard appears in the back of the keyword, it represents any number of characters.

*B.1. Secure Front Wildcard Match Protocol (**FW**)*

A data user B generates a wildcard keyword in the form of "$\star + Y_1$", where Y_1 is a string. Encrypt Y_1 to $[Y_1]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}$ and ν_1 as input, the **FW** protocol, shown as Algorithm 1, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$\star + Y_1$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 1: SECURE FRONT WILDCARD MATCH PROTOCOL (**FW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, ν_1.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 **for** $i = 0$ *to* ν_1 **do**
3 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \mathbf{SCP}([X]_{pk_A}, 8 \times i)$;
4 $[u_i]_{pk_\Sigma} = \mathbf{KET}([X_2]_{pk_A}, [Y_1]_{pk_B})$;
5 CP calculates $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [u_i]_{pk_\Sigma}$;
6 CP and CSP jointly calculate $[u]_{pk_\Sigma} = \mathbf{SLT}([0]_{pk_A}, [u]_{pk_\Sigma})$;
7 **Return** $[u]_{pk_\Sigma}$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). The protocol includes $\nu_1 + 1$ rounds (line 2). In round i, $[X]_{pk_A}$ is partitioned into two parts $[X_1]_{pk_A}, [X_2]_{pk_A}$ and the plaintext of $[X_1]_{pk_A}$ contains i symbols (line 3). X_1 is the string that the wildcard represents. In line 4, if string $X_2 = Y_1$, we have $u_i = 1$; otherwise, $u_i = 0$. Then, add u_i to u by computing $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [u_i]_{pk_\Sigma} = [u + u_i]_{pk_\Sigma}$. After the $\nu_1 + 1$ rounds, if $u > 0$, the protocol outputs $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$. An example of **FW** is shown in Figure 4(b).

*B.2. Secure Middle Wildcard Match Protocol (**MW**)*

A data user B generates a wildcard keyword in the form of "$Y_1 + \star + Y_2$", where Y_1, Y_2 are strings and Y_1 contains η_1 symbols, and then encrypts Y_1, Y_2 to $[Y_1]_{pk_B}, [Y_2]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}, [Y_2]_{pk_B}, \nu_1$ and η_1 as input, the **MW** protocol, shown as Algorithm 2, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$Y_1 + \star + Y_2$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 2: SECURE MIDDLE WILDCARD MATCH PROTOCOL (**MW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, $[Y_2]_{pk_B}$, ν_1, η_1.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \mathbf{SCP}([X]_{pk_A}, 8 \times \eta_1)$;
3 $[u_1]_{pk_\Sigma} = \mathbf{KET}([X_1]_{pk_A}, [Y_1]_{pk_\Sigma})$;
4 $[u_2]_{pk_\Sigma} = \mathbf{FW}([X_2]_{pk_A}, [Y_2]_{pk_\Sigma}, \nu_1)$;
5 $[u]_{pk_\Sigma} = \mathbf{SMD}([u_1]_{pk_\Sigma}, [u_2]_{pk_\Sigma})$;
6 **Return** $[u]_{pk_\Sigma}$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). In line 2, $[X]_{pk_A}$ is partitioned into two parts $[X_1]_{pk_A}, [X_2]_{pk_A}$ and the plaintext of $[X_1]_{pk_A}$ contains η_1 symbols. In line 3, if string $X_1 = Y_1$, we have $u_1 = 1$; otherwise, $u_1 = 0$. In line 4, utilizing **FW** protocol, it tests whether "$\star + Y_2$"

matches X_2. If they match, we have $u_2 = 1$; otherwise, $u_2 = 0$. Then, multiple u_1, u_2 by computing $[u]_{pk_\Sigma} = \texttt{SMD}([u_1]_{pk_\Sigma}, [u_2]_{pk_\Sigma}) = [u_1 \cdot u_2]_{pk_\Sigma}$. If $u_1 = 1, u_2 = 1$, we have $u = 1$; otherwise, $u = 0$. An example of **MW** is shown in Figure 4(c).

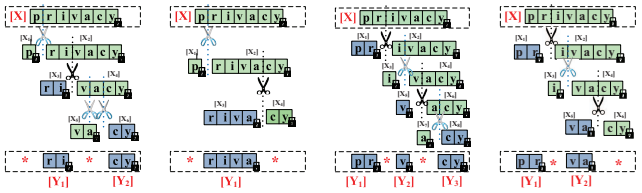
B.3. Secure Back Wildcard Match Protocol (**BW**)

A data user B generates a wildcard keyword in the form of "$Y_1 + \star$", where Y_1 is a string and contains η_1 symbols, and then encrypts Y_1 to $[Y_1]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}$ and η_1 as input, the **BW** protocol, shown as Algorithm 3, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$Y_1 + \star$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 3: SECURE BACK WILDCARD MATCH PROTOCOL (**BW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, η_1.
Output: $[u]_{pk_\Sigma}$.
1 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \texttt{SCP}([X]_{pk_A}, 8 \times \eta_1)$;
2 $[u]_{pk_\Sigma} = \texttt{KET}([X_1]_{pk_A}, [Y_1]_{pk_B})$;
3 **Return** $[u]_{pk_\Sigma}$.

In line 1, $[X]_{pk_A}$ is partitioned into two parts $[X_1]_{pk_A}, [X_2]_{pk_A}$ and the plaintext of $[X_1]_{pk_A}$ contains η_1 symbols. X_2 is the string that the wildcard represents. In line 2, if string $X_1 = Y_1$, we have $u = 1$; otherwise, $u = 0$. In the protocol, "privacy" and "privacy$\star$" are deemed as match keywords. An example of **BW** is shown in Figure 4(d).



(a) **FMW** (b) **FBW** (c) **MMW** (d) **MBW**

Fig. 5: Examples of **FMW**, **FBW**, **MMW**, **MBW** protocols

4.3 Two Wildcards Search

We now proceed to consider keyword search with two wildcards over encrypted data. Since a wildcard could appear in the front, the middle or the back of a string, we design four protocols to handle different situations. These protocols are executed by CP and CSP but without learning the plaintext of the match result.

When the wildcard appears in the front or in the middle of the keyword, data user B specifies a positive integer to denote the maximum character number that the wildcard can be substituted. Denote the positive integers as ν_1, ν_2 for the two wildcards. When the wildcard appears in the back of the keyword, it represents any number of characters.

C.1. Secure Front & Middle Wildcards Match Protocol (**FMW**)

A data user B generates a wildcard keyword in the form of "$\star + Y_1 + \star + Y_2$", where Y_1, Y_2 are strings and Y_1 contains η_1 symbols, and then encrypts Y_1, Y_2 to $[Y_1]_{pk_B}, [Y_2]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}, [Y_2]_{pk_B}, \nu_1, \nu_2$ and η_1 as input, the **FMW** protocol, shown as Algorithm 4, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$\star + Y_1 + \star + Y_2$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 4: SECURE FRONT & MIDDLE WILDCARDS MATCH PROTOCOL (**FMW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, $[Y_2]_{pk_B}$, ν_1, ν_2, η_1.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 **for** $i = 0$ *to* ν_1 **do**
3 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \texttt{SCP}([X]_{pk_A}, 8 \times i)$;
4 $([X_3]_{pk_A}, [X_4]_{pk_A}) = \texttt{SCP}([X_2]_{pk_A}, 8 \times \eta_1)$;
5 $[s_i]_{pk_\Sigma} = \texttt{KET}([X_3]_{pk_A}, [Y_1]_{pk_B})$;
6 **for** $j = 0$ *to* ν_2 **do**
7 CP and CSP jointly calculate $([X_5]_{pk_A}, [X_6]_{pk_A}) = \texttt{SCP}([X_4]_{pk_A}, 8 \times j)$;
8 $[t_j]_{pk_\Sigma} = \texttt{KET}([X_6]_{pk_A}, [Y_2]_{pk_B})$;
9 $[u_{i,j}]_{pk_\Sigma} = \texttt{SMD}([s_i]_{pk_A}, [t_j]_{pk_B})$;
10 $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [u_{i,j}]_{pk_\Sigma}$;
11 CP and CSP jointly calculate $[u]_{pk_\Sigma} = \texttt{SLT}([0]_{pk_A}, [u]_{pk_\Sigma})$;
12 **Return** $[u]_{pk_\Sigma}$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). **FMW** protocol utilizes two-level iterative computations to test the keywords. The outer loop traverses different symbol numbers that the first wildcard can be substituted; the inner loop traverses different symbol numbers for the second wildcard. In line 3, $[X]_{pk_A}$ is partitioned into $[X_1]_{pk_A}, [X_2]_{pk_A}$ and X_1 contains i symbols, for $0 \le i \le \nu_1$. X_1 is the string that the first wildcard represents. In line 4, $[X_2]_{pk_A}$ is partitioned into $[X_3]_{pk_A}, [X_4]_{pk_A}$ and X_3 contains η_1 symbols. In line 5, if $X_3 = Y_1$, we have $s_i = 1$; otherwise, $s_i = 0$. In line 7, $[X_4]_{pk_A}$ is partitioned into $[X_5]_{pk_A}, [X_6]_{pk_A}$ and X_5 contains j symbols, for $0 \le j \le \nu_2$. X_5 is the string that the second wildcard represents. In line 8, if $X_6 = Y_2$, we have $t_j = 1$; otherwise, $t_j = 0$. In line 9, s_i multiplies t_j by computing $[u_{i,j}]_{pk_\Sigma} = \texttt{SMD}([s_i]_{pk_A}, [t_j]_{pk_B}) = [s_i \cdot t_j]_{pk_\Sigma}$. If $s_i = 1, t_j = 1$, we have $u_{i,j} = 1$; otherwise, $u_{i,j} = 0$. In line 10, $u_{i,j}$ is added to u. In line 11, if $u > 0$, the protocol outputs $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$. An example of **FMW** is shown in Figure 5(a).

C.2. Secure Front & Back Wildcards Match Protocol (**FBW**)

A data user B generates a wildcard keyword in the form of "$\star + Y_1 + \star$", where Y_1 is a string and contains η_1 symbols, and then encrypts Y_1 to $[Y_1]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}, \nu_1$ and η_1 as input, the **FBW** protocol, shown as Algorithm 5, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$\star + Y_1 + \star$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 5: SECURE **F**RONT & **B**ACK **W**ILDCARDS MATCH PROTOCOL (**FBW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, ν_1, η_1.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 **for** $i = 0$ *to* ν_1 **do**
3 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \mathbf{SCP}([X]_{pk_A}, 8 \times i)$;
4 $([X_3]_{pk_A}, [X_4]_{pk_A}) = \mathbf{SCP}([X_2]_{pk_A}, 8 \times \eta_1)$;
5 $[s_i]_{pk_\Sigma} = \mathbf{KET}([X_3]_{pk_A}, [Y_1]_{pk_B})$;
6 $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [s_i]_{pk_\Sigma}$;
7 CP and CSP jointly calculate $[u]_{pk_\Sigma} = \mathbf{SLT}([0]_{pk_A}, [u]_{pk_\Sigma})$;
8 **Return** $[u]_{pk_\Sigma}$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). **FBW** protocol utilizes an iterative computation to test the keywords, which traverses different symbol numbers that the first wildcard can be substituted. In line 3, $[X]_{pk_A}$ is partitioned into $[X_1]_{pk_A}, [X_2]_{pk_A}$ and X_1 contains i symbols, for $0 \leq i \leq \nu_1$. X_1 is the string that the first wildcard represents. In line 4, $[X_2]_{pk_A}$ is partitioned into $[X_3]_{pk_A}, [X_4]_{pk_A}$ and X_3 contains η_1 symbols. X_4 is the string that the second wildcard represents. In line 5, if $X_3 = Y_1$, we have $s_i = 1$; otherwise, $s_i = 0$. In line 6, s_i is added to u. In line 7, if $u > 0$, the protocol outputs $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$. An example of **FBW** is shown in Figure 5(b).

C.3. Secure Middle & Middle Wildcards Match Protocol (**MMW**)

A data user B generates a wildcard keyword in the form of "$Y_1 + \star + Y_2 + \star + Y_3$", where Y_1, Y_2, Y_3 are strings and Y_1, Y_2 contains η_1, η_2 symbols, respectively, and then encrypts Y_1, Y_2 to $[Y_1]_{pk_B}, [Y_2]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}, [Y_2]_{pk_B}, [Y_3]_{pk_B}, \nu_1, \nu_2, \eta_1$ and η_2 as input, the **MMW** protocol, shown as Algorithm 6, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$Y_1 + \star + Y_2 + \star + Y_3$" matches X, we have $u = 1$; otherwise, $u = 0$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). **MMW** protocol utilizes two-level iterative computations to test the keywords. The outer loop traverses different symbol numbers that the first wildcard can be substituted; the inner loop traverses different symbol numbers for the second wildcard. In line 2, $[X]_{pk_A}$ is partitioned into $[X_1]_{pk_A}, [X_2]_{pk_A}$ and X_1 contains η_1 symbols. In line 3, if $X_1 = Y_1$, we have $u_1 = 1$; otherwise, $u_1 = 0$. In line 5, $[X_2]_{pk_A}$ is partitioned into $[X_3]_{pk_A}, [X_4]_{pk_A}$ and X_3 contains i symbols, for $0 \leq i \leq \nu_1$. X_3 is the string that the first wildcard represents. In line 6, $[X_4]_{pk_A}$ is partitioned into $[X_5]_{pk_A}, [X_6]_{pk_A}$ and X_5 contains η_2 symbols. In line 7, if $X_5 = Y_2$, we have $s_i = 1$; otherwise, $s_i = 0$. In line 9, $[X_6]_{pk_A}$ is partitioned into $[X_7]_{pk_A}, [X_8]_{pk_A}$ and X_7 contains j symbols, for $0 \leq j \leq \nu_2$. X_7 is the string that the second wildcard represents. In line 10, if $X_8 = Y_3$, we have $t_j = 1$; otherwise, $t_j = 0$. In line 11, s_i multiplies t_j by computing $[u_{i,j}]_{pk_\Sigma} = \mathbf{SMD}([s_i]_{pk_A}, [t_j]_{pk_B}) = [s_i \cdot t_j]_{pk_\Sigma}$. If $s_i = 1, t_j = 1$, we have $u_{i,j} = 1$; otherwise, $u_{i,j} = 0$. In line 12, $u_{i,j}$ multiplies u_1 by computing $\mathbf{SMD}([u_{i,j}]_{pk_\Sigma}, [u_1]_{pk_\Sigma}) = [u_{i,j} \cdot u_1]_{pk_\Sigma}$. If $u_{i,j} = 1, t_j = 1$, it outputs $[u_{i,j}]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u_{i,j}]_{pk_\Sigma} = [0]_{pk_\Sigma}$. In line 13, $u_{i,j}$ is added to u. In line 14, if $u > 0$, the protocol outputs $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$. An example of **MMW** is shown in Figure 5(c).

Algorithm 6: SECURE **M**IDDLE & **M**IDDLE **W**ILDCARDS MATCH PROTOCOL (**MMW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, $[Y_2]_{pk_B}$, $[Y_3]_{pk_B}$, $\nu_1, \nu_2, \eta_1, \eta_2$.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \mathbf{SCP}([X]_{pk_A}, 8 \times \eta_1)$;
3 $[u_1]_{pk_\Sigma} = \mathbf{KET}([X_1]_{pk_A}, [Y_1]_{pk_B})$;
4 **for** $i = 0$ *to* ν_1 **do**
5 CP and CSP jointly calculate $([X_3]_{pk_A}, [X_4]_{pk_A}) = \mathbf{SCP}([X_2]_{pk_A}, 8 \times i)$;
6 $([X_5]_{pk_A}, [X_6]_{pk_A}) = \mathbf{SCP}([X_4]_{pk_A}, 8 \times \eta_2)$;
7 $[s_i]_{pk_\Sigma} = \mathbf{KET}([X_5]_{pk_A}, [Y_2]_{pk_B})$;
8 **for** $j = 0$ *to* ν_2 **do**
9 CP and CSP jointly calculate $([X_7]_{pk_A}, [X_8]_{pk_A}) = \mathbf{SCP}([X_6]_{pk_A}, 8 \times j)$;
10 $[t_j]_{pk_\Sigma} = \mathbf{KET}([X_8]_{pk_A}, [Y_3]_{pk_B})$;
11 $[u_{i,j}]_{pk_\Sigma} = \mathbf{SMD}([s_i]_{pk_A}, [t_j]_{pk_B})$;
12 $[u_{i,j}]_{pk_\Sigma} = \mathbf{SMD}([u_{i,j}]_{pk_\Sigma}, [u_1]_{pk_\Sigma})$;
13 $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [u_{i,j}]_{pk_\Sigma}$;
14 CP and CSP jointly calculate $[u]_{pk_\Sigma} = \mathbf{SLT}([0]_{pk_A}, [u]_{pk_\Sigma})$;
15 **Return** $[u]_{pk_\Sigma}$.

C.4. Secure Middle & Back Wildcards Match Protocol (**MBW**)

A data user B generates a wildcard keyword in the form of "$Y_1 + \star + Y_2 + \star$", where Y_1, Y_2 are strings and Y_1, Y_2 contains η_1, η_2 symbols, respectively, and then encrypts Y_1, Y_2 to $[Y_1]_{pk_B}, [Y_2]_{pk_B}$ using **K2C** algorithm. Taken $[X]_{pk_A}, [Y_1]_{pk_B}, [Y_2]_{pk_B}, \nu_1, \eta_1$ and η_2 as input, the **MBW** protocol, shown as Algorithm 7, outputs a ciphertext $[u]_{pk_\Sigma}$. If the wildcard keyword "$Y_1 + \star + Y_2 + \star$" matches X, we have $u = 1$; otherwise, $u = 0$.

Algorithm 7: SECURE MIDDLE & BACK WILDCARDS MATCH PROTOCOL (**MBW**)

Input: $[X]_{pk_A}$, $[Y_1]_{pk_B}$, $[Y_2]_{pk_B}$, ν_1, η_1, η_2.
Output: $[u]_{pk_\Sigma}$.
1 Initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$;
2 CP and CSP jointly calculate $([X_1]_{pk_A}, [X_2]_{pk_A}) = \mathbf{SCP}([X]_{pk_A}, 8 \times \eta_1)$;
3 $[u_1]_{pk_\Sigma} = \mathbf{KET}([X_1]_{pk_A}, [Y_1]_{pk_B})$;
4 **for** $i = 0$ *to* ν_1 **do**
5 CP and CSP jointly calculate $([X_3]_{pk_A}, [X_4]_{pk_A}) = \mathbf{SCP}([X_2]_{pk_A}, 8 \times i)$;
6 $([X_5]_{pk_A}, [X_6]_{pk_A}) = \mathbf{SCP}([X_4]_{pk_A}, 8 \times \eta_2)$;
7 $[s_i]_{pk_\Sigma} = \mathbf{KET}([X_5]_{pk_A}, [Y_2]_{pk_B})$;
8 $[s_i]_{pk_\Sigma} = \mathbf{SMD}([s_i]_{pk_\Sigma}, [u_1]_{pk_\Sigma})$;
9 $[u]_{pk_\Sigma} = [u]_{pk_\Sigma} \cdot [s_i]_{pk_\Sigma}$;
10 CP and CSP jointly calculate $[u]_{pk_\Sigma} = \mathbf{SLT}([0]_{pk_A}, [u]_{pk_\Sigma})$;
11 **Return** $[u]_{pk_\Sigma}$.

Firstly, initialize $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$ (line 1). **MBW** protocol utilizes a iterative computations to test the keywords,

which traverses different symbol numbers that the first wildcard can be substituted. In line 2, $[X]_{pk_A}$ is partitioned into $[X_1]_{pk_A}, [X_2]_{pk_A}$ and X_1 contains η_1 symbols. In line 3, if $X_1 = Y_1$, we have $u_1 = 1$; otherwise, $u_1 = 0$. In line 5, $[X_2]_{pk_A}$ is partitioned into $[X_3]_{pk_A}, [X_4]_{pk_A}$ and X_3 contains i symbols, for $0 \leq i \leq \nu_1$. X_3 is the string that the first wildcard represents. In line 6, $[X_4]_{pk_A}$ is partitioned into $[X_5]_{pk_A}, [X_6]_{pk_A}$ and X_5 contains η_2 symbols. In line 7, if $X_5 = Y_2$, we have $s_i = 1$; otherwise, $s_i = 0$. In line 8, s_i multiplies u_1 by computing $\texttt{SMD}([s_i]_{pk_\Sigma}, [u_1]_{pk_\Sigma}) = [s_i \cdot u_1]_{pk_\Sigma}$. If $s_i = 1, u_1 = 1$, it outputs $[s_i]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[s_i]_{pk_\Sigma} = [0]_{pk_\Sigma}$. In line 9, s_i is added to u. In line 10, if $u > 0$, the protocol outputs $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$; otherwise, $[u]_{pk_\Sigma} = [0]_{pk_\Sigma}$. An example of **MBW** is shown in Figure 5(d).

5 Proposed System

5.1 Key Generation

Let $SEnc/SDec$ be a symmetric encryption/decryption algorithm pair (with key space $\mathcal{K}$) and $Sig/Verify$ be a signature/verification algorithm pair that are cryptographically secure.[3] Define hash functions $H_1 : \{0,1\}^* \rightarrow Z_N$ and $H_2 : Z_N \rightarrow \mathcal{K}$.

Running $KeyGen$ algorithm of PCTD, KGC generates the system public parameter $PP = (g, N)$, master secret key $MSK = \lambda$ and user A_i's public/secret key pair $pk_{A_i} = g^{\theta_i}, sk_{A_i} = \theta_i$. KGC calculates the master public key $MPK = g^{\lambda}$. MSK will be kept secret by KGC and MPK is public. Then, KGC executes master secret key splitting algorithm of PCTD to generate partial strong secret keys $SK_1 = \lambda_1$ and $SK_2 = \lambda_2$, which are secretly sent to CP and CSP, respectively. sk_{A_i} is confidentially sent to user A_i and pk_{A_i} is public. In order to protect the privacy of user A_i's identity, KGC generates a unique anonymous identity $AID_{A_i} \in \{0,1\}^{\vartheta}$, where ϑ is a positive integer and 2^{ϑ} is larger than the total number of users in the system.

5.2 User Authorization and Revocation

If data owner A authorizes the search and decryption right to data user B, A will set the valid period VP to indicate the start and end time (such as VP = "20170101-20180101"). A generates an authorization certificate for B with certificate number CN. To guarantee the uniqueness of CN, it should begin with AID_A to indicate that CN is generated by A. The authorization certificate $CER_{A,B}$ is $\langle cer = (CN, AID_B, VP, pk_\Sigma), Sig(cer, sk_A)\rangle$, where $pk_\Sigma = g^{sk_\Sigma}, sk_\Sigma = H_1(CN, sk_A)$.[4] The secret key sk_Σ is confidentially sent to B. $CER_{A,B}$ is transmitted to KGC, CP, CSP and user B. The authorization will be automatically invalid when VP expires.

If A plans to revoke B's privilege within VP, he generates a revocation certificate $RVK_{A,B}$ as $\langle rvk = (revoke, CN), Sig(rvk, sk_A)\rangle$. Then, $RVK_{A,B}$ is sent to KGC, CP, CSP and user B.

If B wants to simultaneous issue query on data owners $(A_1, \cdots, A_m)$'s files, he should apply for the certificates $CER_{A_i,B}$ $(1 \leq i \leq m)$ from each data owner. Then, he applies for certificate from KGC. After verifying the certificates, KGC computes the valid period $VP_\Sigma = VP_1 \bigcap \cdots \bigcap VP_m$ and generates the certificate $CER_{\Sigma,B}$ as $\langle cer = (CN, AID_B, VP_\Sigma, pk_\Sigma), Sig(cer, MSK)\rangle$, where $pk_\Sigma = g^{sk_\Sigma}, sk_\Sigma = H_1(CN, MSK)$, CN begins with KGC's identity $ID_{KGC} \in \{0,1\}^{\vartheta}$ to indicate that CN is generated by KGC[5]. sk_Σ is confidentially delivered to user B and pk_Σ is public to CP, CSP and user B. To revoke $CER_{A_i,B}$ within VP_Σ, KGC generates a revocation certificate $RVK_{\Sigma,B}$ as $\langle rvk = (revoke, CN), Sig(rvk, MSK)\rangle$. Then, $RVK_{\Sigma,B}$ is sent to CP, CSP and user B.

5.3 Encryption

A data owner A outsources a file M (with file identity $ID \in Z_N$) to the cloud. He extracts a set of keywords $(kw_1, \cdots, kw_{n_1})$ to describe the file and encrypts them to $\mathbb{W} = ([kw_1]_{pk_A}, \cdots, [kw_{n_1}]_{pk_A})$ using **K2C** algorithm. He randomly selects $K \in Z_N$ as the file encryption key and encrypts it to $[K]_{pk_A}$. The file M is encrypted to $C = SEnc(M, K')$, where $K' = H_2(K) \in \mathcal{K}$. Then, the encrypted index $(\mathbb{W}, [ID]_{pk_A}, [K]_{pk_A})$ and file ciphertext C are sent to CP.

5.4 Query Generation

Data user B figures out a set of query keywords $\{qw_1, \cdots, qw_{n_2}\}$. The queried keyword may contain zero, one or two wildcards. If the wildcard is in the front or middle of the keyword, B specifies a positive number to indicate the maximum characters that the wildcard can be substituted. B encrypts the query keyword qw_i to Q_i according to the keyword type, $1 \leq i \leq n_2$. Table 1 depicts eight types of keywords and their parameter settings. The parameters in blue represent that they are not actually the input of the corresponding protocol.

Let $\mathbb{Q} = (Q_1, \cdots, Q_{n_2})$. The data user B designates a relationship to the query (i.e. AND or OR) to make conjunctive or disjunctive query. Then, data user sends anonymous identity AID_B, the encryption $\mathbb{Q}$ of $\{qw_1, \cdots, qw_{n_2}\}$, a signature $Sig(\mathbb{Q}, sk_B)$ and the relationship (AND or OR) to CP as query trapdoor.

Since there are eight types of keywords, we present a concrete selection method (Supplemental Material A.1) to encrypt the keyword, which is based on Table 1 and the protocols in Section 4.

5.5 Search

Receiving the query trapdoor $\langle AID_B, \mathbb{Q}, Sig(\mathbb{Q}, sk_B)\rangle$ and query relationship (AND or OR), CP firstly verifies

3. The concrete algorithms will not be specified in this paper.

4. To simplify the presentation, we utilize the element in Z_N to be the secret key of Sig algorithm. In practical usage, the signature key can be easily derived from the element in Z_N using the hash function.

5. ID_{KGC} may be set to ϑ zeros.

TABLE 1: Parameter Comparison of the Eight Protocols

$f = \lvert \star \rvert$	String Type	Protocol	Parameters							
			$[X]$	$[Y_1]$	$[Y_2]$	$[Y_3]$	ν_1	ν_2	η_1	η_2
0	Y	**KET**	$[X]_{pk_A}$	$[Y]_{pk_B}$	–	–	-1	-1	-1	-1
1	$\star + Y_1$	**FW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	–	–	> 0	-1	-1	-1
	$Y_1 + \star$	**BW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	–	–	-1	-1	> 0	-1
	$Y_1 + \star + Y_2$	**MW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	$[Y_2]_{pk_B}$	–	> 0	-1	> 0	-1
2	$\star + Y_1 + \star + Y_2$	**FMW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	$[Y_2]_{pk_B}$	–	> 0	> 0	> 0	-1
	$* + Y_1 + \star$	**FBW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	–	–	> 0	-1	> 0	-1
	$Y_1 + \star + Y_2 + \star + Y_3$	**MMW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	$[Y_2]_{pk_B}$	$[Y_3]_{pk_B}$	> 0	> 0	> 0	> 0
	$Y_1 + \star + Y_2 + \star$	**MBW**	$[X]_{pk_A}$	$[Y_1]_{pk_B}$	$[Y_2]_{pk_B}$	–	> 0	-1	> 0	> 0

TABLE 2: Performance of **K2C**, **MBE**, **SCP**, **KET**, **BW** Protocols

	Computation Overhead (s)							Communication Overhead (KB)						
N	**512**	**768**	**1024**	**1280**	**1536**	**1792**	**2048**	**512**	**768**	**1024**	**1280**	**1536**	**1792**	**2048**
K2C	0.002	0.009	0.018	0.034	0.058	0.084	0.121	–						
MBE	0.045	0.151	0.307	0.559	1.007	1.438	2.322	1.27	1.91	2.55	3.19	3.83	4.47	5.11
SCP	0.042	0.146	0.310	0.569	0.993	1.841	2.346	1.27	1.91	2.55	3.19	3.83	4.47	5.11
KET	0.127	0.380	0.828	1.687	2.809	3.933	5.879	4.34	6.51	8.68	10.86	13.04	15.22	17.39
BW	0.114	0.397	0.969	1.884	3.476	4.407	6.878	5.61	8.43	11.24	14.06	16.88	19.69	22.51

TABLE 3: Performance of **FW**, **MW**, **FBW**, **MBW** Protocols

	Computation Overhead (s)						Communication Overhead (KB)					
ν_1	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
FW	1.537	1.569	1.743	2.224	2.253	2.555	25.82	37.08	48.33	59.56	70.84	82.05
MW	2.206	2.234	2.386	2.810	2.836	3.106	41.68	52.91	64.20	75.40	86.64	97.95
FBW	1.905	1.913	2.114	2.483	2.674	3.270	30.926	44.739	58.565	72.374	86.078	100.026
MBW	2.474	2.452	2.669	3.065	3.241	3.827	32.074	45.867	59.713	73.515	87.211	101.163

TABLE 4: Performance of **FMW**, **MMW** Protocols

	(ν_1, ν_2)	**(1,1)**	**(1,2)**	**(1,3)**	**(2,1)**	**(2,2)**	**(2,3)**	**(3,1)**	**(3,2)**	**(3,3)**
Computation Overhead (s)	**FMW**	1.867	1.891	2.024	1.923	1.938	2.112	2.068	2.136	2.674
	MMW	2.439	2.459	2.589	2.493	2.517	2.686	2.627	2.715	3.232
Communication Overhead (KB)	**FMW**	90.953	122.685	154.333	136.497	184.077	231.657	181.889	245.351	308.779
	MMW	92.096	123.833	155.479	137.625	185.218	232.795	183.027	246.489	309.917

TABLE 5: Execution Time

	Encryption Time (s)										Query Generation Time (s)				
$\lvert KW \rvert$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**
Time	0.019	0.020	0.023	0.033	0.038	0.039	0.046	0.056	0.057	0.060	0.043	0.047	0.052	0.056	0.061

TABLE 6: Search Performance

	$(\lvert \mathbb{W} \rvert, \lvert \mathbb{Q} \rvert)$	**(3,1)**	**(3,2)**	**(3,3)**	**(4,1)**	**(4,2)**	**(4,3)**	**(6,1)**	**(6,2)**	**(6,3)**
Computation Overhead (s)	AND	2.274	2.61	2.938	2.303	2.665	2.955	2.618	2.950	3.159
	OR	1.957	2.029	2.169	1.979	2.068	2.168	2.034	2.164	2.294
Communication Overhead (MB)	AND	0.339	0.677	1.016	0.452	0.903	1.355	0.677	1.355	2.032
	OR	0.333	0.665	0.998	0.443	0.887	1.33	0.665	1.33	1.995

whether B is authorized to access to the data. If B has the privilege, CS verifies the signature of $\mathbb{Q}$ using user B's public key pk_B. If it is not valid, the query will be rejected. Otherwise, CP responds to the search query. According to the parameters in Q_j, CP inputs $[kw_i]_{pk_A}$ in $\mathbb{W}$ and Q_j in $\mathbb{Q}$ to corresponding keyword match protocols (**KET**, **FW**, **MW**, **BW**, **FMW**, **FBW**, **MMW** or **MBW**). Then, CP obtains the encrypted match result $[u_{i,j}]_{pk_\Sigma}$. If kw_i matches qw_j, we have $u_{i,j} = 1$; otherwise, $u_{i,j} = 0$. To facilitate the protocol selection, we present a concrete selection method (Supplemental Material A.2) to choose the keyword match protocol, which is based on Table 1 and the protocols in Section 4.

If the data user makes an OR query, CP initializes $[u^*]_{pk_\Sigma} = [0]_{pk_\Sigma}$ and calculates $[u^*]_{pk_\Sigma} = \prod_{i\in[n_1],j\in[n_2]}[u_{i,j}]_{pk_\Sigma}$. If $u^* > 0$, it indicates the trapdoor matches the keywords; otherwise, $u^* = 0$.

If the data user makes an AND query, CP initializes $[u]_{pk_\Sigma} = [1]_{pk_\Sigma}$. CP and CSP jointly calculates

$$[u]_{pk_\Sigma} = \textbf{SMD}([u]_{pk_\Sigma}, [u_{i,j}]_{pk_\Sigma}),$$

for $1 \le i \le n_1, 1 \le j \le n_2$. If $u^* = 1$, it indicates the trapdoor matches the keywords; otherwise, $u^* = 0$.

Then, CP sends $([u^*]_{pk_\Sigma}, [ID]_{pk_A})$ to data user for the searched documents.

5.6 Decryption

Receiving the returned results, the user B decrypts $[u^*]_{pk_\Sigma}$ to get the search result u^*. If B makes an OR query, B will rank u^* and ask CP to return the top-k documents that has the highest u^* values, which is the relevance score. If B makes an AND query, B will ask CP to return all or part of the documents that have $u^* = 1$. B will send $[ID]_{pk_A}$ to CP for the document query. Receiving the returned files, the user B decrypts the file encryption key K, which is hashed to symmetric key $K' = H_2(K_i) \in \mathcal{K}$. Then, the file M is recovered using K'.

Remark: In this system, the CP sends the test result of all searched files to user for decryption. If the returned data is large, however, the transmission overhead between CP and user will be large. An effective way to solve this problem is letting the CP directly put the test results into the secure top-k rank algorithm in [30], which will output the top-k documents that have the highest relevance scores. Then, only the top-k files are returned to data user. This optimization method increases the computation overhead on CP and CSP, but reduces computation overhead of data user and the communication overhead between CP and data user. Moreover, due to the characteristic of secure top-k algorithm in [30], CP can not distinguish the returned results from the original ciphertext, which protects user's privacy to the maximum extent.

6 Performance Analysis

We implement this system and the protocols using JAVA on a Windows 10 64-bit personal computer with Intel(R) Core(TM) i5-6600T CPU @2.70GHz, 8GB RAM. Multi-thread programming technology is utilized to program the protocols and system. In general, the execution time of the system grows with the number of searched keywords, wildcards, users and influenced by the parameter $\mathcal{L}(N)$. These influential factors are analyzed below.

6.1 Performance of Crypto Primitives and Protocols

The computation and communication costs of the protocols are evaluated and the results are shown in Tables 2-4. The overhead of these protocols grows with the parameter $\mathcal{L}(N)$ since the modular addition, multiplication and exponentiation calculations increase with $\mathcal{L}(N)$. In Table 2, we set $\mathcal{L}(N)$ to be 512, 768, 1024, 1280, 1536, 1792 and 2048 bits to test the **MBE**, **SCP**, **KET**, **BW** protocols and **K2C** algorithm.

Since the performance of the **FW**, **MW**, **FMW**, **FBW**, **MMW**, **MBW** protocols depends on the values of ν_1, ν_2, we fix $\mathcal{L}(N) = 1024$ to evaluate these protocols in Tables 3-4. The performance of these protocols grows with the maximum character numbers ν_1, ν_2 that the wildcards can be substituted.

When $\mathcal{L}(N) = 1024$, the system achieves 80-bit security level [31]. In order to trade off the security and performance, it is recommended to set $\mathcal{L}(N) = 1024$ in practical environment.

- The execution time of **K2C** algorithm is shown in Table 2. It costs only 0.018 s to encrypt a keyword when $\mathcal{L}(N) = 1024$. This algorithm does not incur any communication cost.
- Table 2 shows the performance of **MBE**, **SCP**, **KET** and **BW** protocols. When $\mathcal{L}(N) = 1024$, **MBE** protocol incurs computation cost 0.307 s and communication cost 2.55 KB; **SCP** protocol incurs computation cost 0.310 s and communication cost 2.55 KB; **KET** protocol incurs computation cost 0.828 s and communication cost 8.68 KB; **BW** protocol incurs computation cost 0.969 s and communication cost 11.24 KB.
- Table 3 shows the performances of **FW**, **MW**, **FBW** and **MBW** protocols increase with ν_1, which is the maximum number of symbols that the wildcard can be substituted. In these experiments, we set $\mathcal{L}(N) = 1024$. When $\nu_1 = 6$, **FW** protocol incurs computation cost 2.555 s and communication overhead 82.05 KB; **MW** protocol incurs computation cost 3.106 s and communication overhead 97.95 KB; **FBW** protocol incurs computation cost 3.270 s and communication overhead 100.026 KB; **MBW** protocol incurs computation cost 3.827 s and communication overhead 101.163 KB.
- Table 4 shows the performances of **FMW** and **MMW** protocols increase with ν_1 and ν_2, which are the maximum numbers of symbols that the wildcards can be substituted. When $\nu_1 = 3, \nu_2 = 3$, **FMW** protocol incurs computation cost 2.674 s and communication overhead 308.779 KB; **MMW** protocol in-

curs computation cost 3.232 s and communication overhead 309.917 KB.

6.2 System Performance

The system performance is evaluated in this subsection. We set $\mathcal{L}(N) = 1024$ to achieve 80-bit security level.

- The key generation algorithm in this system needs 0.09 ms to generate public parameter PP, master public/secret key pair MPK/MSK and partial strong secret keys SK_1/SK_2. For each user, KGC spends 0.04 ms to create its public/private key pair.
- In encryption phase, the data owner encrypts keywords using **K2C** algorithm. The performance grows with the number of keywords that are extracted from the file. When the keywords number varies from 1 to 10, Table 5 shows that it spends less than 0.060 s to build the encrypted index $(\mathbb{W}, [ID]_{pk_A}, [K]_{pk_A})$.
- In query phase, the data user specifies a set of query keywords, which may include wildcard. The performance of the query algorithm increases with the number of searched items. When the keywords number varies from 1 to 5, Table 5 shows that it spends less than 0.061 s to calculate $\mathbb{Q}$, which is the encryption of the query keywords.
- In search phase, CS tests whether the trapdoor and encrypted index match. The performance grows with the number of keywords in trapdoor and that in encrypted index. The number and position of the wildcard also affect the performance. In the experiment, we randomly select the queried keyword, which may include zero, one or two wildcards. The maximum symbol number ν that the wildcard can be substituted is also randomly chosen. For keyword that contains only one wildcard, we select a random $\nu \in \{1, \cdots, 6\}$. For keyword that contains two wildcards, we select random $\nu_1, \nu_2 \in \{1, \cdots, 3\}$. Table 6 shows the average cost of search algorithm when it is executed for 1000 times. When $(|\mathbb{W}|, |\mathbb{Q}|) = (6, 3)$ for an AND query, it requires 3.159 s computation cost and 2.032 MB communication cost. When $(|\mathbb{W}|, |\mathbb{Q}|) = (6, 3)$ for an OR query, it requires 2.294 s computation cost and 1.995 MB communication cost.

TABLE 7: Computation Overhead Comparison (s)

	Symmetric Key			Asymmetric Key	
Algorithms	**[19]**	**[18]**	**[17]**	**[16]**	**Ours**
Setup	0.001	0.001	0.001	0.145	0.009
Enc	0.323	0.320	0.001	2.145	0.060
Query	0.001	0.001	0.001	2.014	0.061
Search	0.001	0.001	0.001	3.236	2.294/3.159

The efficiency of this system is compared with [16], [17], [18], [19] in Table 7. In the experiment, data owner extracts six keywords from the document and data user chooses three keywords to issue a query. The query keywords are randomly selected. Each algorithm is executed 1000 times to get the average time.

It is obvious that the wildcard searchable encryption schemes in symmetric key setting [17], [18], [19] have better efficiency. However, these systems only support single user scenario. If a data owner authorizes his search privilege to other user, he has to reveal his private key to the authorized user. It is not practical in network application. Moreover, these symmetric schemes are all constructed based on Bloom filter and suffer from false-positive probability. The scheme [16] is a public key setting, which has larger computation overhead compared with ours. Moreover, it does not support user revocation nor simultaneously searching on multiple data owner's encrypted data using only one trapdoor.

7 Security Analysis

In this section, we will firstly prove the security of the protocols under the security model defined in Section 2.3. Then, we analyze the security of the proposed wildcard searchable encryption system.

7.1 Protocols Security Proof

Theorem 1. *The **SGE** protocol is secure against the semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ without random oracles.*

Refer Supplemental Material B for the security proof.

Theorem 2. *The **KET** protocol is secure against the semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ without random oracles.*

Proof. **KET** protocol just calls **SGE** and **SMD** as subprotocols and all data are encrypted using PCTD encryption. Since **SGE** and **SMD** protocols are proved secure in Theorem 1 and [27], the **KET** protocol is also secure in the presence of attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. □

Theorem 3. *The **MBE** protocol is secure against the semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ without random oracles.*

Refer Supplemental Material B for the security proof.

Theorem 4. *The **SCP** protocol is secure against the semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ without random oracles.*

Proof. **SCP** protocol calls **MBE** as subprotocol and all data are encrypted using PCTD encryption. Since **MBE** protocol is proved secure in Theorem 3, the **SCP** protocol is secure in the presence of attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. □

Theorem 5. *The **FW** protocol is secure against the semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ without random oracles.*

Proof. **FW** protocol calls **KET**, **SCP** and **SLT** as subprotocols and all data are encrypted using PCTD encryption.

Since these protocols are proved secure in Theorem 2, 4 and [27], the **FW** protocol is secure in the presence of attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. □

The security proofs of **MW**, **BW**, **FMW**, **FBW**, **MMW** and **MBW** are similar to that of **FW** in the presence of semi-honest (non-colluding) attackers $\mathcal{A} = (\mathcal{A}_{D_1}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.

7.2 System Security Proof

The security of this system is analyzed as follows.

- Key Generation: The privacy of master secret key and users' private keys is based on the hardness of discrete logarithm problem. Adversary can not deduce MSK and sk_{A_i} from MPK and pk_{A_i}, respectively. Due to the security of PCTD [27], the secret keys of CP and CSP are secure.
- User Authorization and Revocation: Assume that the signature scheme Sig is strong unforgeable. It ensures that the authorization/revocation certificate can not be forged when user and KGC's secret keys are confidentially kept.
- Encryption: The semantic security of PCTD [27] guarantees the security of encrypted index. Assume the symmetric encryption algorithm $SEnc$ is cryptographically secure. It ensures that the encrypted file is secure.
- Query: The security of trapdoor is guaranteed by the semantic security of PCTD. Unauthorized user can not issue a valid query due to the strong unforgeable of Sig algorithm.
- Search: In the search phase, the encrypted index and query trapdoor are put into different protocols (**KET**, **FW**, **MW**, **BW**, **FMW**, **FBW**, **MMW**, **MBW**) according the queried keyword type. Since these protocols are proved secure above and **SMD** protocol is proved secure in [27], the search algorithm is secure.

Next, we following the attack model in Section 2.2 to prove this system is secure against adversary $\mathcal{A}^*$.

(1) Suppose $\mathcal{A}^*$ eavesdrop all the interactive information between the system user and CP, and all the data transmitted between CP and CSP. $\mathcal{A}^*$ still can not get any plaintext information since all these data are protected by PCTD and symmetric encryption algorithm $SEnc$.

(2-3) Suppose $\mathcal{A}^*$ compromise CP or CSP and obtain λ_1 or λ_2. However, $\mathcal{A}^*$ could not simultaneously compromise CP and CSP to get the master secret key λ. If $\mathcal{A}^*$ compromises CSP and get the intermediate computation results in the protocols, $\mathcal{A}^*$ can not deduce any valuable information because all original data are blinded by a random number using the "blinding technology" [32].

(4) Suppose $\mathcal{A}^*$ compromise data owners or data users (except the challenge user) and get their secret keys. He is not able to get the challenge user's plaintext information since the secret keys belong to different users are independent.

Resist to Collusion Attack: We show that our system is resistant to collusion attack from authorized users. Suppose data users $(B_1, \cdots, B_n)$ collude to seek for the search and decryption privilege of the challenge data user B^* and $B^* \notin (B_1, \cdots, B_n)$. First, they are not able to submit a valid search query for B^*, because they can not forge a valid query trapdoor signature $Sig(\mathbb{Q}, sk_{B^*})$ without the secret key sk_{B^*}. They can not derive sk_{B^*} from their own secret keys $(sk_{B_1}, \cdots, sk_{B_n})$, since these secret keys are independently generated. Second, even though they can eavesdrop the communication channel to get the search result of B^*, they can not recover the plaintext since the decryption key sk_Σ is unknown.

8 Conclusion

In this paper, we proposed a new and flexible wildcard searchable encryption system for secure cloud storage service, which supports flexible wildcard representation, flexible search function and flexible user authorization/revocation. The system does not suffer from false probability in returning search results to users and no privacy information is leaked to cloud service including the relevance score of the match files. The system also supports AND and OR relationship based multiple keywords search and top-k search. We formally proved security of the underlying protocols and the system. We implemented the system on a PC and the test results showed that the proposed system achieves high efficiency compared with existing schemes in the public key setting.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant No. 61402112, 61472307, 61472309, 61502086; the Singapore National Research Foundation under the NCR Award Number NRF2014NCR-NCR001-012; AXA Research Fund; Fujian Provincial Key Laboratory of Information Processing and Intelligent Control (Minjiang University) MJUKF201734; Fujian Major Project of Regional Industry 2014H4015; and Major Science and Technology Project of Fujian Province under Grant No. 2015H6013.

References

[1] Kamara S, Lauter K. Cryptographic cloud storage[C]//International Conference on Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2010: 136-149.

[2] Singh A, Chatterjee K. Cloud security issues and challenges: A survey[J]. Journal of Network and Computer Applications, 2017, 79: 88-115.

[3] Boneh D, Di Crescenzo G, Ostrovsky R, et al. Public key encryption with keyword search[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2004: 506-522.

[4] Yang Y, Ma M. Conjunctive Keyword Search With Designated Tester and Timing Enabled Proxy Re-Encryption Function for E-Health Clouds[J]. IEEE Transactions on Information Forensics and Security, 2016, 11(4): 746-759.

[5] Yang Y. Attribute-based data retrieval with semantic keyword search for e-health cloud[J]. Journal of Cloud Computing, 2015, 4(1): 1.
[6] Qiu S, Liu J, Shi Y, et al. Hidden policy ciphertext-policy attribute-based encryption with keyword search against keyword guessing attack[J]. Science China Information Sciences, 2017, 60(5): 052105.
[7] Jarecki S, Jutla C, Krawczyk H, Rosu M, Steiner M. Outsourced symmetric private information retrieval. In Proc. ACM CCS 2013, pp. 875-888.
[8] Sepehri M, Cimato S, Damiani E. Privacy-preserving query processing by multi-party computation. The Computer Journal, vol. 58, no. 10, 2015, pp. 2195-2212.
[9] Sepehri M, Cimato S, Damiani E, Yeun CY. Data sharing on the cloud: A scalable proxy-based protocol for privacy-preserving queries. In Proceedings of TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1, pp. 1357-1362.
[10] Sun SF, Liu JK, Sakzad A, Steinfeld R, Yuen TH. An Efficient Non-Interactive Multi-client Searchable Encryption with Support for Boolean Queries. In Proc. ESORICS 2016, pp. 154-172.
[11] Li J, Wang Q, Wang C, et al. Fuzzy keyword search over encrypted data in cloud computing[C]//INFOCOM, 2010 Proceedings IEEE. IEEE, 2010: 1-5.
[12] Li J, Chen X. Efficient multi-user keyword search over encrypted data in cloud computing[J]. Computing and Informatics, 2013, 32(4): 723-738.
[13] Wang B, Yu S, Lou W, et al. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud[C]//IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014: 2112-2120.
[14] Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.
[15] Fu Z, Wu X, Guan C, et al. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement [J]. IEEE Transactions on Information Forensics and Security, 2016, 11(12): 2706-2716.
[16] Sedghi S, Van Liesdonk P, Nikova S, et al. Searching keywords with wildcards on encrypted data[C]//International Conference on Security and Cryptography for Networks. Springer Berlin Heidelberg, 2010: 138-153.
[17] Bosch C, Brinkman R, Hartel P, et al. Conjunctive wildcard search over encrypted data[C]//Workshop on Secure Data Management. Springer Berlin Heidelberg, 2011: 114-127.
[18] Suga T, Nishide T, Sakurai K. Secure keyword search using Bloom filter with specified character positions[C]//International Conference on Provable Security. Springer Berlin Heidelberg, 2012: 235-252.
[19] Hu C, Han L. Efficient wildcard search over encrypted data[J]. International Journal of Information Security, 2015: 1-9.
[20] Su S, Teng Y, Cheng X, Xiao K, Li G, Chen J. Privacy-Preserving Top-k Spatial Keyword Queries in Untrusted Cloud Environments. IEEE Transactions on Services Computing. 2015 Sep 24.
[21] Singh A, Srivatsa M, Liu L. Search-as-a-service: Outsourced search over outsourced storage. ACM Transactions on the Web (TWEB). 2009 Sep 1, 3(4):13.
[22] Liu X, Deng R H, Ding W, et al. Privacy-preserving outsourced calculation on floating point numbers[J]. IEEE Transactions on Information Forensics and Security, 2016, 11(11): 2513-2527.
[23] Liu X, Choo R, Deng R, et al. Efficient and Privacy-Preserving Outsourced Calculation of Rational Numbers[J]. IEEE Transactions on Dependable and Secure Computing, publish online, DOI.10.1109/TDSC.2016.2536601.
[24] Do Q, Martini B, Choo K K R. A forensically sound adversary model for mobile devices[J]. PloS one, 2015, 10(9): e0138449.
[25] Kamara S, Mohassel P, Raykova M. Outsourcing Multi-Party Computation[J]. IACR Cryptology ePrint Archive, 2011, 2011: 272.
[26] Liu X, Qin B, Deng R, et al. An Efficient Privacy-Preserving Outsourced Computation over Public Data[J]. IEEE Transactions on service computing, publish online, DOI: 10.1109/TSC.2015.2511008.
[27] Liu X, Deng R, Choo K K R, et al. An Efficient Privacy-Preserving Outsourced Calculation Toolkits with Multiple Keys[J]. IEEE Transactions on Information Forensics and Security, publish online.
[28] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 1999: 223-238.
[29] Bresson E, Catalano D, Pointcheval D. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications[C]//International Conference on the Theory and Application of Cryptology and Information Security. Springer Berlin Heidelberg, 2003: 37-54.
[30] Liu X, Lu R, Ma J, et al. Privacy-preserving patient-centric clinical decision support system on naive Bayesian classification[J]. IEEE journal of biomedical and health informatics, 2016, 20(2): 655-668.
[31] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST special publication 800-57, NIST Special Publication, vol. 800, no. 57, pp. 1-142.
[32] Peter A, Tews E, Katzenbeisser S. Efficiently outsourcing multiparty computation under multiple keys[J]. IEEE transactions on information forensics and security, 2013, 8(12): 2046-2058.

Yang Yang received the B.Sc. degree from Xidian University, Xi'an, China, in 2006 and Ph.D. degrees from Xidian University, China, in 2012. She is an associate professor in the college of mathematics and computer science, Fuzhou University. She is also a research fellow (postdoctor) under supervisor Robert H. Deng in School of Information System, Singapore Management University. Her research interests are in the area of information security and privacy protection.

Ximeng Liu received the B.Sc. degree from Xidian University, Xi'an, China, in 2010 and Ph.D. degrees from Xidian University, China, in 2015. He was the research assistant at School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore from 2013 to 2014. Now, he is a research fellow at School of Information System, Singapore Management University, Singapore. His research interests include cloud security and big data security.

Robert H. Deng is an AXA Chair Professor of Cybersecurity in School of Information Systems, Singapore Management University. His research interests include data security and privacy, network and system security. He has served/is serving on the editorial boards of many international journals in security, such as IEEE Transactions on Information Forensics and Security and IEEE Transactions on Dependable and Secure Computing. He is Fellow of IEEE.

Jian Weng received the Ph.D. degree from Shanghai Jiao Tong University, in 2008. From April 2008 to March 2010, he was a post-doctor in the School of Information Systems, Singapore Management University. Currently, he is a professor and executive dean with the School of Information Technology, Jinan University. He has published more than 60 papers in cryptography conferences and journals such as Eurocryp, Asiacrypt, PKC, and IEEE TIFS.