Dynamic and Efficient Private Keyword Search over Inverted Index-Based Encrypted Data

RUI ZHANG, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences
RUI XUE, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences
TING YU, Qatar Computing Research Institute, Hamad Bin Khalifa University
LING LIU, College of Computing, Georgia Institute of Technology

Querying over encrypted data is gaining increasing popularity in cloud based data hosting services. Security and efficiency are recognized as two important and yet conflicting requirements for querying over encrypted data. In this paper we propose efficient private keyword search scheme (EPKS for short) that support binary search and extend it to dynamic settings (called DEPKS) for inverted index-based encrypted data. First, we describe our approaches of constructing a searchable symmetric encryption scheme that supports binary search. Second, we present a novel framework for EPKS, and provide its formal security definitions in terms of *plaintext privacy* and *predicate privacy* by modifying Shen et al.'s security notions [Shen et al. 2009]. Third, built on the proposed framework, we design an EPKS scheme whose complexity is logarithmic in the number of keywords. The scheme is based on the groups of prime order and enjoys strong notions of security, namely *statistical plaintext privacy* and *statistical predicate privacy*. Fourth, we extend EPKS scheme to support dynamic keyword and document updates. The extended scheme not only maintains the properties of logarithmic-time search efficiency and plaintext privacy and predicate privacy, but also has fewer rounds of communications for updates, compared with existing dynamic search encryption schemes. We experimentally evaluate the proposed EPKS and DEPKS scheme and show that they are significantly more efficient in terms of both keyword search complexity and communication complexity than existing randomized searchable symmetric encryption schemes.

CCS Concepts: •**Security and privacy** → **Management and querying of encrypted data;**

Additional Key Words and Phrases: Searchable symmetric encryption, binary search, plaintext privacy, predicate privacy, dynamic updates

1. INTRODUCTION

The proliferation of a new breed of cloud applications that store and process data at remote service providers has led to the emergence of search over encrypted data as an important research problem. In a typical setting of the problem, a query generated at the client side is transformed into a representation such that it can be evaluated directly on encrypted data at the remote service provider. The returned results might be processed by the client after decryption to determine the final answers.

Informally, a practical encryption scheme used above should satisfy the following properties: search time is logarithmic (or sublinear) in the number of ciphertexts, and the ciphertexts together with search tokens reveal no information about the under-

Rui Zhang and Rui Xue are supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grants No. XDA06010701, and National Natural Science Foundation of China (No.61402471, 61472414). Ling Liu is partially supported by the National Science Foundation under Grants IIS-0905493, CNS-1115375, NSF 1547102, SaTC 1564097, and Intel ISTC on Cloud Computing.
Author's addresses: R. Zhang and R. Xue, State Key Laboratory Of Information Security (SKLOIS), Institute of Information Engineering, Chinese Academy of Sciences; T. Yu, Qatar Computing Research Institute and Hamad Bin Khalifa University; L. Liu, College of Computing, Georgia Institute of Technology.

DOI: http://dx.doi.org/10.1145/0000000.0000000

lying plaintext. Unfortunately, to the best of our knowledge, all existing encryption schemes supporting keyword search can barely achieve the above two properties simultaneously. Some of them [Kamara et al. 2012; Kamara and Papamanthou 2013; Bellare et al. 2007] achieve sublinear-time search complexity but their search tokens leak information of the query. Some others [Shen et al. 2009; Yoshino et al. 2012] achieve strong notion of security but with low efficiency. Recently, Lu proposed logarithmic-time search schemes [Lu 2012] for range predicate queries that achieve both plaintext privacy and predicate privacy. However, his schemes only support queries over numerical values and are based on the groups of composite order, which results in inefficiency of pairing computation. Plaintext privacy here refers to the risk that if a third party may learn the main content of a document by inferring any association between frequent search keywords and encrypted dataset from the search index. Predicate privacy here refers to the risk that a third party may learn any information beyond the outcome of search by inference on the query keyword corresponding to a given search token.

Besides the above properties, many cloud applications also require dynamic updates of searchable symmetric encryption (SSE), namely after encrypted index for specific collections of documents have been setup, dynamic additions or deletions of keywords and documents can be performed without re-building the index. Many recent works [Kamara et al. 2012; Kamara and Papamanthou 2013; Cash et al. 2014] offer solutions to dynamic searchable symmetric encryption (DSSE). Unfortunately, no known DSSE schemes achieve both the above security and efficiency requirements. Most existing DSSE constructions [Kamara et al. 2012; Kamara and Papamanthou 2013; Cash et al. 2014] aim at practical efficiency with different trade-offs between security, efficiency, and the ability to supporting dynamic updates.

In this paper, we first propose an Efficient Private Keyword Search (EPKS for short) scheme for inverted index-based encrypted data, which satisfy both the above mentioned requirements, that is, (1) they search index in time logarithmic in the number of keywords, and (2) the ciphertexts together with search tokens reveal essentially no information about the underlying keywords. Then, we extend EPKS scheme to provide the capability of dynamic keywords and documents updates with low communication and computation cost. Our contribution can be summarized as follows:

— First, we describe our approaches of constructing EPKS schemes for inverted index-based encrypted data. Intuitively, we want to keep the randomness of index and token, but make the scheme support binary search. Technically, we transform the unordered plaintexts of index into a "special ordered array" and use vectors to express them. This enables our schemes to perform binary search by computing the inner-product of the index vector and a search token vector using Billnear Map (see Section 4.2. Based on the re-ordered keyword structures, we present a modified framework and its security definitions of EPKS in terms of *plaintext privacy* and *predicate privacy* following Shen et al.'s security notions [Shen et al. 2009].
— Then, we propose a construction for EPKS based on the groups of prime order, which can be implemented efficiently. It not only supports binary search and as a consequence the time complexity of search is reduced from $O(n^2)$ to $O(\log n)$, where n is the number of keywrods, but also offers stronger notions of security: *statistical plaintext privacy* and *statistical predicate privacy*, which means the scheme is secure even for any *computationally-unbounded* adversary. To the best of our knowledge, this could be the first SSE scheme to achieve information-theoretical security.
— Third, we extend EPKS scheme to DEPKS scheme for supporting dynamic keyword and document updates. DEPKS has fewer rounds of communication for updates,

specifically, one round of interaction for each update compared to one and half round of interaction in existing DSSE schemes.
— Finally, we analytically and experimentally show that EPKS and DEPKS are significantly more efficient for querying over encrypted data in the cloud environment, compared with existing randomized SSE schemes.

2. PREVIOUS WORK

2.1. Searchable Symmetric Encryption

Searchable symmetric encryption (SSE) was first formally defined explicitly by Song et al. in [Song et al. 2000], where they gave a non-interactive solution with search complexity linear to the size of the encrypted data.

Formal security notions for searchable symmetric encryption have evolved over time. The first security notion known as *semantic security against chosen keyword attack* (CKA1) was formulated by Goh [Goh 2003]. Several works on SSE schemes use CKA1 as a security definition [Goh 2003; Chang and Mitzenmacher 2005; Curtmola et al. 2006]. Later, Curtmola et al. [Curtmola et al. 2006] proposed a stronger security notion: *adaptive security against chosen-keyword attack* (CKA2). While several CKA2-secure SSE schemes are proposed in the literature [Curtmola et al. 2006; Kurosawa and Ohtaki 2012; Liesdonk et al. 2010; Kamara et al. 2012; Kamara and Papamanthou 2013; Cash et al. 2013a; Jarecki et al. 2013], none of them are explicitly *probabilistic*; that is, the search tokens they generate with pseudorandom functions (PRFs) or pseudorandom permutations (PRPs) are deterministic, which we call these SSE schemes *non-randomized SSE* schemes. It means that the same token will always be generated for the same keyword, and it will lead to the leakage of statistical information about a user's search pattern [Islam et al. 2012].

To deal with this problem, Shen et al. [Shen et al. 2009] designed a symmetric-key predicate-only encryption (SKPOE) scheme with probabilistic token based on predicate encryption [Katz et al. 2008; Shi and Waters 2008] (We call those SSE schemes, in which the token generation algorithms are probabilistic, *randomized SSE* schemes). They considered a new security notion called *predicate privacy*. The property of predicate privacy is that tokens reveal no information about the encoded query predicate. However, due to the use of complex probabilistic encryption algorithm to generate tokens, the construction of [Shen et al. 2009] requires at least linear time complexity in the number of keywords to complete the search. Based on their work, Lu [Lu 2012] proposed privacy-preserving logarithmic-time search schemes for range queries of numerical values. Unfortunately, the constructions of his schemes are also based on the groups of composite order, which need very large parameter size.[1] Moreover, the pairing computation is much slower over a composite-order than a prime-order elliptic curve.[2] [Guillevic 2013]

In the cloud computing setting, a lot of works have been done to support efficient and more expressive search, such as multi-keyword ranked search [Sun et al. 2013; Cao et al. 2011a; Wang et al. 2012], Similarity Search [Wang et al. 2012], query over encrypted graph-structured data [Cao et al. 2011b] and biometric identification [Wang et al. 2015], but at the expense of weaker security guarantees.

[1] For a supersingular elliptic curve of composite order with $N = q_1q_2q_3q_4$ used in SKPOE [Shen et al. 2009] and RPE [Lu 2012], the size of $\mathbb{G}=\mathbb{G}_1\times\mathbb{G}_2\times\mathbb{G}_3\times\mathbb{G}_4$ is 2776-3260 bits and the size of element in $\mathbb{G}_T$ should be $\geq 5556- \geq 6524$ bits. While, for a supersingular elliptic curve of prime order, the size of $\mathbb{G}$ is only 256 bits and the size of element in $\mathbb{G}_T$ should be 2644-3224 bits.

[2] For 128-bit security level, a pairing on an elliptic curve of composite order with two primes is about 254 times slower than over a prime-order elliptic curve.

2.2. Dynamic Searchable Symmetric Encryption

The constructions by Song et al. [Song et al. 2000], Goh [Goh 2003] and Chang [Chang and Mitzenmacher 2005] all support insertions and deletions of documents but require linear search time. The recently introduced dynamic scheme by Kamara et al. [Kamara et al. 2012] was the first one with sublinear search time, but it leaks hashes of the unique keywords contained in the updated document. The scheme of Kamara and Papamanthou [Kamara and Papamanthou 2013] overcomes the above limitation by increasing the space of the user's data structure. Very recently, Cash et al. [Cash et al. 2014] first implemented dynamic symmetric searchable encryption schemes in very-large databases. But like the above schemes, their schemes are based on the CKA2 security model and cannot achieve predicate privacy: the search tokens as well as the update tokens are deterministic and result in leakage of a user's search pattern.

2.3. Oblivious RAM

Oblivious random-access machine (ORAM) [Goldreich and Ostrovsky 1996] can be used to hide every memory access during searches and updates in SSE and DSSE [Damgård et al. 2011; Goldreich 1987; Kushilevitz et al. 2012; Pinkas and Reinman 2010; Shi et al. 2011; Stefanov and Shi 2013; Cash et al. 2013b; Stefanov et al. 2013; Stefanov et al. 2014]. ORAM provides the strongest levels of security (not only search pattern privacy but also access pattern privacy), namely the server only learns the size of the document collection. However, ORAM schemes are very inefficient in practice to handle large amount of queries and data due to poly-logarithmic overhead on all parameters.

3. PROBLEM STATEMENT

3.1. The System Model

Consider a cloud data storage service, where a data owner has a set of documents D to be outsourced to the cloud server in an encrypted form. To enable efficient query over encrypted documents, we consider the inverted index-based data structure for storing the outsourced files. Specifically, the data owner builds an encrypted searchable inverted index set C with keywords $w_1, w_2, \cdots, w_n$ from D, and then both the encrypted index set C and the encrypted document set $\mathtt{E}(D)$ are outsourced to the cloud server. For every query of a keyword w_i, a data user computes a search token TK and sends it to the cloud server. Upon receiving TK from the data user, the cloud server queries over the encrypted index set C and returns the candidate encrypted documents. Finally, the data user decrypts the candidate documents and verifies each document by checking the existence of the keyword.

3.2. Privacy Requirements

Intuitively, a searchable encryption scheme is secure if the server learns nothing about the query as well as the documents except the encrypted query results [Song et al. 2000; Boneh et al. 2004; Goh 2003; Shen et al. 2009]. In this section, we discuss in detail the specific privacy requirements for index-based data storage structures, where the cloud server searches over a set of searchable index instead of searching on encrypted data directly.

Data privacy is a basic requirement which requires the data (or documents) to be outsourced should not be revealed to any unauthorized parties including cloud service providers. Typically, it can be guaranteed by symmetric encryption algorithms. The user who has the secret key can effectively decrypt the encoded data after retrieving them from the cloud server.

With respect to *plaintext privacy*, if the cloud server deduces any association between frequent keywords and encrypted dataset from the index, it may learn the main content of a document. Therefore, searchable index should be constructed in such a way that prevents the cloud server from performing such kind of association attacks. In the non-randomized SSE schemes [Song et al. 2000; Goh 2003; Chang and Mitzenmacher 2005; Curtmola et al. 2006; Liesdonk et al. 2010; Kamara et al. 2012; Kamara and Papamanthou 2013], this kind of security is also called *security against chosen keyword attack (CKA)*. In our schemes, the randomness of encrypted index guarantees to prevent such attacks from the cloud server.

Data users usually prefer to keep their query from being exposed to others, i.e., the keyword indicated by the corresponding token. In the literature [Curtmola et al. 2006; Stefanov et al. 2014; Kamara and Papamanthou 2013; Kamara et al. 2012], this kind of leakage is called *search pattern*. Namely search patterns reveal whether the same search was performed in the past or not. Curtmola et al. [Curtmola et al. 2006] claimed that with the exception of oblivious RAMs (ORAMs), all non-randomized SSE constructions leak the user's search pattern since their search tokens are deterministic. Like existing randomized SSE schemes, our schemes use randomly generated tokens to guarantee the privacy of a user's search pattern. This security notion of randomized SSE schemes is called *predicate privacy* [Shen et al. 2009]. Note that randomizing token generation algorithm only contributes to defend outside adversaries of the cloud server but not inner adversaries (e.g., cloud administrators), because the entry of index touched in each search process discloses the search pattern as well. A possible approach to reduce this leakage is to re-order the keywords and re-generate the index periodically, say semimonthly or monthly.

Besides the above privacy requirements, we note that the sequence of search outcomes of all SSE constructions will likely reveal the information of the keywords since the cloud server will always return the same document set for the same queried keyword. This kind of leakage is referred to as *access pattern* [Curtmola et al. 2006]. Only ORAM can hide access patterns. But ORAM is computationally intensive and do not scale well for real world datasets. In practice, one may reduce (but not eliminate) the leakage of access patterns. For example, one could randomly insert fake documents in the bitmaps [Islam et al. 2012].

In this paper, we focus on ensuring *plaintext privacy against any adversary* and *predicate privacy against outside adversaries of the cloud server*. The above mentioned techniques can be combined with our schemes to reduce search and access pattern leakage as well.

4. PRELIMINARIES

4.1. Notations

Throughout the paper, we use the following notation.

For an integer $n \in \mathbb{N}$, let $[n]$ be the set $\{1, \cdots, n\}$, and let U_n be the uniform distribution over the set $\{0,1\}^n$. For a finite set S, let $x \leftarrow S$ be the process of sampling a value x according to the distribution over S, and let $x \xleftarrow{R} S$ the random choose process of a value x from the uniform distribution over S. Let $\vec{x}$ be a vector $(x_1, \cdots, x_{|\vec{x}|})$, where $|\vec{x}|$ and $x_i (1 \leq i \leq |\vec{x}|)$ respectively denote the number of elements and the i-th element of vector $\vec{x}$. Let $\vec{x}^{\mathsf{T}}$ be the transposition of vector $\vec{x}$.

We overload the notation g^M to matrices: let $g^M \in \mathbb{G}^{n \times n}$ be the matrix defined as $(g^M)_{i,j} = g^{M_{i,j}}$, where $1 \leq i, j \leq n$. Let M^{-1} be the inverse matrix of M. Let $g^{\vec{x} \cdot M}$ be the product defined as $(g^{x_1 M_{1,1} + \cdots + x_n M_{n,1}}, \cdots, g^{x_1 M_{1,n} + \cdots + x_n M_{n,n}})$ and let $g^{M \cdot \vec{x}^{\mathsf{T}}}$ be the product defined as $(g^{M_{1,1} x_1 + \cdots + M_{1,n} x_n}, \cdots, g^{M_{n,1} x_1 + \cdots + M_{n,n} x_n})$.

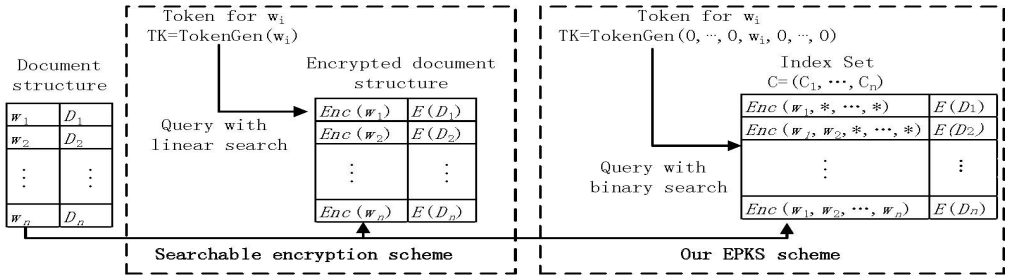


Fig. 1. The modified keyword structure

The scheme is parameterized by the security parameter λ. A function ϵ is *negligible* if for every polynomial $p(\cdot)$ there exist an N such that for all integers $n > N$ it holds that $\epsilon(n) < \frac{1}{p(n)}$.

4.2. Bilinear Map

Let `GroupGen` be a probabilistic polynomial-time algorithm that takes as input a security parameter 1^λ, and outputs $(\mathbb{G}, \mathbb{G}_T, q, g, e)$, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order q, $\mathbb{G}$ is generated by $g \in \mathbb{G}$, and q is a λ-bit prime number. Let $\mathbb{G}_T$ be a (different) group of order q. A bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ has the following properties.

(1) *Bilinearity:* for all $g_1, g_2 \in \mathbb{G}$, $a, b \in \mathbb{Z}$ it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
(2) *Non-degeneracy:* $e(g, g) \neq 1$;
(3) It follows that $g_T \stackrel{\text{def}}{=} e(g, g)$ generates $\mathbb{G}_T$.

Let $\vec{x} = \{x_1, \cdots, x_n\}$ and $\vec{y} = \{y_1, \cdots, y_n\}$ be two n-dimensional vectors. The bilinear map of $e(g_1^{\vec{x}}, g_2^{\vec{y}})$ is computed as follows:

$$e(g_1^{\vec{x}}, g_2^{\vec{y}}) \;=\; e(g_1^{x_1}, g_2^{y_1}) \cdot e(g_1^{x_2}, g_2^{y_2}) \cdots e(g_1^{x_n}, g_2^{y_n}) = e(g_1, g_2)^{x_1 y_1 + \cdots + x_n y_n}$$

5. SEARCHABLE SYMMETRIC ENCRYPTION WITH BINARY SEARCH

For a searchable encryption scheme, the most difficult part is how to lower the search time complexity while keeping the algorithm randomized. One intuitive idea is to enable the scheme to support binary search. Before introducing our approaches, we first review the inverted index-based data structure in details.

5.1. Inverted Index-Based Data Structure

An inverted index is an index data structure storing a mapping from content, such as keywords to a set of documents. The purpose of an inverted index is to allow fast full text searches. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

The inverted index-based data structure is shown on the left of Fig. 1. Let $D = \{d_1, d_2, \cdots, d_m\}$ be the set of documents to be stored in an untrusted cloud server, where $|D| = m$ is the total number of documents. Each document d_i contains a set of keywords. Let $W = \{w_1, w_2, \cdots, w_n\}$ be the set of keywords in D, where $|W| = n$ is the total number of keywords. $D_{i \in [n]} \subseteq D$ denotes the set of documents containing keyword w_i. The keywords and the sets of documents are respectively encrypted by encryption algorithms `Enc` and `E`, and stored in the cloud as shown in Fig. 1. Note that `Enc` and `E` are different encryption schemes: `Enc` is a searchable encryption scheme supporting binary search proposed in this paper and `E` can be any secure symmetric encryption scheme, such as Advanced Encryption Standard (AES). The n encrypted

ALGORITHM 1: Query with Binary Search (C, TK, pp)

Input: C, TK, pp
Output: result
$\texttt{round} = 0$; $\texttt{head} = 1$; $\texttt{end} = n$; $\texttt{mid} = 0$; $\texttt{result} = \perp$;
repeat
  $\texttt{mid} = \lfloor(\texttt{head} + \texttt{end})/2\rfloor$;
  if $\texttt{Test}(TK, C_{\texttt{mid}}, pp) = 1$ **then**
    $\texttt{end} = \texttt{mid} - 1$;
  else
    $\texttt{head} = \texttt{mid} + 1$;
  end
  $\texttt{round} + +$;
until $\texttt{head} > \texttt{end}$;
if $C_{\texttt{head}}.value \neq \phi$ **then**
  $\texttt{result} = C_{\texttt{head}}.value$;
end

keywords can be regarded as index of the encrypted documents and expressed as $C = \{C_1, \cdots, C_n\} = \{\texttt{Enc}(w_1), \texttt{Enc}(w_2), \cdots, \texttt{Enc}(w_n)\}$. A data user can issue a query with a search token $TK = \texttt{TokenGen}(w_i)$ for the documents that containing keyword w_i, where TokenGen denotes the corresponding token generation algorithm. To avoid leakage of the queried keyword from encrypted index and search token, the encryption and token generation algorithms should be randomized as existing randomized searchable encryption schemes [Shen et al. 2009; Yoshino et al. 2012]. However, with the randomness of encrypted index and tokens, the cloud sever has to successively compare the search token TK with each $C_{i\in[n]}$ in the search process as shown on the left dashed box in Fig. 1. Obviously, the time complexity of such search method is linear in the number of keywords.

5.2. Improvement of Keyword Structures

For efficient search, our first technique is to transform the unordered encrypted index to an ordered structure, which can support efficient search such as binary search in logarithmic-time complexity. We modify the keyword structure for satisfying the ordered requirement of binary search. As shown on the right dashed box in the Fig. 1, we modify each index item to $C_i = \texttt{Enc}(w_1, \cdots, w_i, *, \cdots, *)$, where $*$ can be any value in the domain of keywords. Correspondingly, the search token generation is changed into $TK = \texttt{TokenGen}(0, \cdots, 0, w_i, 0, \cdots, 0)$. Then, we use vectors to express them, that is, $< w_1, \cdots, w_i, *, \cdots, * >$ and $< 0, \cdots, 0, w_i, 0, \cdots, 0 >$. So that the cloud server can perform binary search by computing the inner product of the search token vector and the index vector and locate the matched index item. Computing inner-product of these two vectors is actually a process of checking whether the queried keyword satisfies the predicate of each index item using Bilinear Map (recall Section 4.2). If it is, then the server continues to check the search token with the former half of index; otherwise, the server continues to check the search token with the latter half of index.

The pseudo-code for querying with binary search is shown in Algorithm 1. Given a search token TK for keyword w_i and the encrypted index C, the algorithm exposes the corresponding index item C_i and returns $\texttt{E}(D_i)$ or $\perp$ for "not found". The $\texttt{Test}(TK, C_i, pp)$ algorithm outputs a bit to indicate whether the index item C_i matches with the queried keyword corresponding to the given search token TK. The symbol pp in Algorithm 1 indicates the public parameters of our schemes (Readers can refer to the Section 6.1 and 7).

The limitation of this method is that the order of the keywords needs to be known by the data user (token generator) which is not required in other SSE schemes. However, we think this is acceptable in the private key setting where the data user usually is the data owner or the data owner should send the secret key as well as the order of keywords to the data user through a secure channel. For the privacy of underlying keywords, we require that the order of keywords should be arbitrary and decided by the data owner. The reason we do not use the lexical order of the keywords is that it will reveal partial information about the keywords, e.g., the adversary will successfully guess the first keyword since it probably begins with the letter A.

6. FRAMEWORK AND DEFINITIONS OF EPKS

6.1. The Framework of EPKS

Let $\Pi = (\texttt{Setup}, \texttt{Enc}, \texttt{TokenGen}, \texttt{Test}, \texttt{Query})$ be an EPKS scheme over the set of keywords $\mathcal{W}_\lambda$ consists of the following probabilistic polynomial time (PPT) algorithms as follows:

— $\texttt{Setup}(1^\lambda) \rightarrow (pp, sk)$: On input the security parameter 1^λ, output public parameters pp and a secret key sk.
— $\texttt{Enc}(W, sk, pp) \rightarrow C$: On input the keywords set $W = \{w_1, \cdots, w_n\} \subseteq \mathcal{W}_\lambda$, the symmetric key sk and public parameters pp, output searchable encrypted index $C = (C_1, \cdots, C_n)$.
— $\texttt{TokenGen}(w_i, i, sk, pp) \rightarrow TK$: On input the keyword $w_i \in W$ and its sequence number i, the secret key sk and public parameters pp, output a search token TK.
— $\texttt{Test}(TK, C_i, pp) \rightarrow \{0, 1\}$: On input a search token TK, each encrypted index item C_i and public parameters pp, output a bit indicating whether the item matches with the queried keyword corresponding to the search token.
— $\texttt{Query}(TK, C, pp) \rightarrow \texttt{E}(D_i)$ or $\perp$: On input a token TK, the searchable encrypted index $C = \{C_1, \cdots, C_n\}$ and public parameters pp, perform binary search with running $\texttt{Test}$ algorithm, output the candidate set of encrypted documents $\texttt{E}(D_i)$ or $\perp$.

Remark. We restrict ourselves to symmetric key cryptography conditioned on the event in which the data user has knowledge of the keywords collection when he generate the token for keyword w_i. To correctly create the search token, keywords w_i and its sequence number i are both needed in the process of token generation.

Correctness. The query correctness of an EPKS scheme can be defined as follows:

Definition 6.1 (*Correctness*). For all λ, all $W \subseteq \mathcal{W}_\lambda$, letting $(pp, sk) \leftarrow \texttt{Setup}(1^\lambda)$, $C \leftarrow \texttt{Enc}(W, sk, pp)$, $TK \leftarrow \texttt{TokenGen}(w_i, i, sk, pp)$, and the Algorithm 1 is performed correctly,

— If $w_i \in W$ and i is the sequence number of w_i in set W, then $\texttt{Query}(TK, C, pp) = \texttt{E}(D_i)$.
— Otherwise, $\Pr[\texttt{Query}(TK, C, pp) = \perp] > 1 - \epsilon(\lambda)$, where $\epsilon(\lambda)$ is a negligible function.

6.2. Modeling Plaintext Privacy and Predicate Privacy for Efficient Search

In this section we introduce the notions of plaintext privacy and predicate privacy for our EPKS scheme. Recall Shen et al.'s security notions of plaintext privacy and predicate privacy [Shen et al. 2009], ask that both ciphertexts and tokens reveal no information about the encoded predicate, but the definitions do not completely apply to the framework of EPKS. The definition of plaintext privacy in [Shen et al. 2009] only consider the security of single ciphertext, that is, the challenge is one ciphertext corresponding to challenge keyword $w_{b \in \{0,1\}}$. If directly applying this definition in EPKS framework, the adversary will easily distinguish two ciphertexts with dif-

$\mathsf{Expt}^{(b)}_{sPP1,\Pi,\mathcal{A}}$	$\mathsf{Expt}^{(b)}_{sPP2,\Pi,\mathcal{A}}$
1: $(pp, sk) \leftarrow \mathtt{Setup}(1^\lambda)$.	1: $(pp, sk) \leftarrow \mathtt{Setup}(1^\lambda)$.
2: $(W_0^*, W_1^*, state) \leftarrow \mathcal{A}(1^\lambda)$, where $W_0^*, W_1^* \subseteq \mathcal{W}_\lambda$ and $\lvert W_0^*\rvert = \lvert W_1^*\rvert = n$.	2: $(w_0^*, w_1^*, j^*, state) \leftarrow \mathcal{A}(1^\lambda)$, where $w_0^*, w_1^* \in \mathcal{W}$ and $j^* \in [n]$.
3: $C^* \leftarrow \mathtt{Enc}(W_b^*, sk, pp)$.	3: $TK^* \leftarrow \mathtt{TokenGen}(w_b^*, j^*, sk, pp)$, where $j^* \in [n]$.
4: $b' \leftarrow \mathcal{A}^{\mathtt{Enc}(\cdot,sk,pp),\mathtt{TokenGen}(\cdot,\cdot,sk,pp)}(C^*, state)$, where $b' \in \{0,1\}$.	4: $b' \leftarrow \mathcal{A}^{\mathtt{Enc}(\cdot,sk,pp),\mathtt{TokenGen}(\cdot,\cdot,sk,pp)}(TK^*, state)$, where $b' \in \{0,1\}$.
5: For all token queries $(w_{i,j}, j)$, where $j \in [n]$, $\mathtt{Query}(TK_{w_{i,j}}, C_{W_0^*}, pp) = \mathtt{Query}(TK_{w_{i,j}}, C_{W_1^*}, pp)$, then output b', otherwise output $\perp$.	5: For all ciphertext queries W_i, $\mathtt{Query}\left(TK_{w_0^*}, C_i, pp\right) = \mathtt{Query}\left(TK_{w_1^*}, C_i, pp\right)$. then output b', otherwise output $\perp$.

Fig. 2. The Experiments of $\mathsf{Expt}^{(b)}_{sPP1,\Pi,\mathcal{A}}$ and $\mathsf{Expt}^{(b)}_{sPP2,\Pi,\mathcal{A}}$

ferent order in the keyword set. Considering plaintext privacy in binary search, the challenge should be a set of ciphertexts $C^* = \{C_1^*, \cdots, C_n^*\}$ corresponding to challenge keyword set $W^*_{b\in\{0,1\}}$. Therefore we need to re-define the models of plaintext privacy and predicate privacy under the EPKS framework.

Typically, our notions consider adversaries that are given the public parameters of the scheme, and can interact with the encryption oracle $\mathtt{Enc}$ and the token generation oracle $\mathtt{TokenGen}$. The encryption oracle $\mathtt{Enc}$ takes as input any adversarially-chosen vectors of keywords. For i-th encryption query $W_i = (w_{i,1}, \cdots, w_{i,n})$, the encryption oracle $\mathtt{Enc}$ responds with $C_i \leftarrow \mathtt{Enc}(W_i, sk, pp)$. The token generation oracle $\mathtt{TokenGen}$ shares a state with the encryption oracle $\mathtt{Enc}$, takes as inputs queries of the form $(w_{i,j}, j)$, where $w_{i,j}$ is a keyword in W_i and $j \in [n]$ denotes the sequence number of $w_{i,j}$, and responds with $TK_{w_{i,j}} \leftarrow \mathtt{TokenGen}(w_{i,j}, j, sk, pp)$.

We also consider the selective variants of plaintext privacy and predicate privacy that ask adversaries to announce ahead of time the challenge keywords.

6.2.1. Plaintext Privacy. The basic notion of plaintext privacy asks that it should not be possible to learn any information about keywords from the ciphertexts beyond the absolute minimum necessary. For example, in the context of cloud services, plaintext privacy refers to the risk that an unauthorized cloud server may learn the main content of a document by inference on any association between frequent keywords and encrypted dataset from the index. Therefore, the design of our searchable index should be constructed in such a way that can prevent the third party cloud server from performing such kind of association inference attacks.

Definition 6.2 (*plaintext privacy*). An EPKS scheme $\Pi = (\mathtt{Setup}, \mathtt{Enc}, \mathtt{TokenGen}, \mathtt{Test}, \mathtt{Query})$ is *plaintext privacy* if for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that

$$\mathsf{Adv}^{sPP1}_{\Pi,\mathcal{A}}(\lambda) \stackrel{\mathsf{def}}{=} \left|\Pr\left[\mathsf{Expt}^{(0)}_{sPP1,\Pi,\mathcal{A}} = 1\right] - \Pr\left[\mathsf{Expt}^{(1)}_{sPP1,\Pi,\mathcal{A}} = 1\right]\right| \leq \epsilon(\lambda),$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$ the experiment $\mathsf{Expt}^{(b)}_{sPP1,\Pi,\mathcal{A}}$ is defined as shown in Fig. 2.Public parameters and a secret key is generated by running $\mathtt{Setup}(1^\lambda)$. Adversary $\mathcal{A}$ is given input 1^λ. It outputs a pair of keywords sets W_0^*, W_1^* of the same length. A uniform bit $b \in \{0,1\}$ is chosen, and then a challenge ciphertext $C^* \leftarrow \mathtt{Enc}(W_b^*, sk, pp)$ is computed and given to $\mathcal{A}$. The adversary $\mathcal{A}$ is given oracle access to $\mathtt{Enc}(\cdot, sk, pp)$ and $\mathtt{TokenGen}(\cdot, \cdot, sk, pp)$, but is not allowed to query the latter on which $\mathtt{Query}(TK_{w_{i,j}}, C_{W_0^*}, pp) \neq \mathtt{Query}(TK_{w_{i,j}}, C_{W_1^*}, pp)$. Eventually, $\mathcal{A}$ outputs a bit b', which is also the output of the experiment. If $b' = b$, we say that $\mathcal{A}$ succeeds.

Remark. In addition, such a scheme is *statistically* plaintext private if the above holds for any *computationally-unbounded* adversary.

6.2.2. Predicate Privacy. The basic notion of predicate privacy asks that it should not be possible to learn any information, beyond the outcome of search, on the queried keyword $w_{i,j}$ corresponding to a given token $TK_{w_{i,j}}$.

Definition 6.3 (*predicate privacy*). An EPKS scheme $\Pi = (\mathtt{Setup}, \mathtt{Enc}, \mathtt{TokenGen}, \mathtt{Test}, \mathtt{Query})$ is *predicate privacy* if for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\lambda)$ such that

$$\mathsf{Adv}^{sPP2}_{\Pi,\mathcal{A}}(\lambda) \stackrel{\mathsf{def}}{=} \left|\Pr\left[\mathsf{Expt}^{(0)}_{sPP2,\Pi,\mathcal{A}}=1\right] - \Pr\left[\mathsf{Expt}^{(1)}_{sPP2,\Pi,\mathcal{A}}=1\right]\right| \leq \epsilon(\lambda),$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$ the experiment $\mathsf{Expt}^{(b)}_{sPP2,\Pi,\mathcal{A}}$ is defined as shown in Fig. 2. Public parameters and a secret key is generated by running $\mathtt{Setup}(1^\lambda)$. Adversary $\mathcal{A}$ is given input 1^λ. It outputs a pair of keywords w_0^*, w_1^* and a sequence number $j*$. A uniform bit $b \in \{0,1\}$ is chosen, and then a challenge search token $TK^* \leftarrow \mathtt{TokenGen}(w_b^*, j^*, sk, pp)$ is computed and given to $\mathcal{A}$. The adversary $\mathcal{A}$ is given oracle access to $\mathtt{Enc}(\cdot, sk, pp)$ and $\mathtt{TokenGen}(\cdot,\cdot, sk, pp)$, but is not allowed to query the former on which $\mathtt{Query}\left(TK_{w_0^*}, C_i, pp\right) \neq \mathtt{Query}\left(TK_{w_1^*}, C_i, pp\right)$. Eventually, $\mathcal{A}$ outputs a bit b', which is also the output of the experiment. If $b' = b$, we say that $\mathcal{A}$ succeeds.

Remark. In addition, such a scheme is *statistically* predicate private if the above holds for any *computationally-unbounded* adversary.

7. CONSTRUCTION OF EPKS

Inspired by [Cao et al. 2011b], we present a concrete EPKS scheme on the cyclic groups of prime order, which can be efficiently implemented. Compared with existing randomized SSE schemes, this scheme offers stronger security, i.e., statistical plaintext privacy and statistical predicate privacy.

Let $\Pi = (\mathtt{Setup}, \mathtt{Enc}, \mathtt{TokenGen}, \mathtt{Test}, \mathtt{Query})$ be an EPKS scheme. The detail construction of each algorithm is as follows.

— $\mathtt{Setup}(1^\lambda) \to (sk, pp)$. The algorithm samples $(\mathbb{G}, \mathbb{G}_T, q, g, e) \leftarrow \mathtt{GroupGen}(1^\lambda)$, where $\mathbb{G}$ is a cyclic group of prime order q, an $(n+1)$-bit vector $\vec{I} \stackrel{R}{\leftarrow} \{0,1\}^{n+1}$ subjects to $\vec{I} \neq \{\vec{0}\}$, two full rank matrices $M', M'' \stackrel{R}{\leftarrow} \mathbb{Z}_q^{(n+1)\times(n+1)}$, and a collision-resistant hash function $H: \{0,1\}^* \to \mathbb{Z}_q$. It then sets $pp = (\mathbb{G}, \mathbb{G}_T, q, g, e, H)$ as public parameters and $sk = (\vec{I}, M', M'')$ as the secret key. Note that M', M'' are invertible with all but a negligible probability.

— $\mathtt{Enc}(W, sk, pp) \to C$. With a collection of keywords $W = \{w_1, \cdots, w_n\}$, the algorithm encrypts it with secret key sk and outputs the encrypted index $C = \{C_1, C_2, \cdots, C_n\}$, where each element C_i is generated as follows.

(1) Generate an $(n+1)$-dimensional vector $\vec{p_i} = (p_{i,1}, \cdots, p_{i,n+1})$ as follows. For $j = 1$ to $n+1$, if $j \leq i$, set $p_{i,j} = H(w_j)$; if $i < j \leq n$, set $p_{i,j} = x_{i,j} \stackrel{R}{\leftarrow} \mathbb{Z}_q$; if $j = n+1$, set $p_{i,j} = 1$.

(2) Split $\vec{p_i}$ into two vectors $\vec{p_i'}$ and $\vec{p_i''}$ with the splitting indicator $\vec{I}$ as follows. For $1 \leq j \leq n+1$, if $I_j = 0$, set $p'_{i,j} = p''_{i,j} = p_{i,j} \pmod q$; if $I_j = 1$, set $p'_{i,j} + p''_{i,j} = p_{i,j} \pmod q$, where $p'_{i,j} \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and $p''_{i,j} = p_{i,j} - p'_{i,j} \pmod q$.

(3) Encrypt these two vectors as $C_i = (g^{\vec{p}_i M'}, g^{\vec{p}_i'' M''}) = (C_{i,1}, C_{i,2})$ for keyword w_i.

— $\texttt{TokenGen}(w_i, i, sk, pp) \to TK$. With the queried keyword w_i and its sequence number i, secret key sk and public parameters pp, the algorithm creates search token TK for keyword w_i as follows.

(1) Generate an $(n+1)$-dimensional vector $\vec{t} = \{t_1, \cdots, t_{n+1}\}$ as follows. For $j = 1$ to $n+1$, if $j = i$, set $t_j = r \xleftarrow{R} \mathbb{Z}_q$; if $j \neq i$ and $j \leq n$, set $t_j = 0$; if $j = n+1$, set $t_j = a \xleftarrow{R} \mathbb{Z}_q$.

(2) Split $\vec{t}$ into two vectors $\vec{t}'$ and $\vec{t}''$ with the splitting indicator $\vec{I}$ as follows. For $j = 1$ to $n+1$, if $I_j = 1$, set $t_j' = t_j'' = t_j \pmod q$; if $I_j = 0$, set $t_j' + t_j'' = t_j \pmod q$, where $t_j' \xleftarrow{R} \mathbb{Z}_q$ and $t_j'' = t_j - t_j' \pmod q$.

(3) Compute $\beta = H(w_i)$ and output $TK = (g^{M'^{-1}\vec{t}'^{\mathsf{T}}}, g^{M''^{-1}\vec{t}''^{\mathsf{T}}}, g^{r\beta+a}) = (TK_1, TK_2, TK_3)$ for w_i.

— $\texttt{Test}(TK, C_i, pp) \to \{0, 1\}$. With the search token TK for keyword w_i and ciphertext C_i, the algorithm tests whether the following equation holds. If Equation (1) holds it returns 1; otherwise it returns 0.

$$e(C_{i,1}, TK_1) \cdot e(C_{i,2}, TK_2) \stackrel{?}{=} e(g, TK_3) \tag{1}$$

— $\texttt{Query}(TK, C, pp) \to \texttt{E}(D_i)$ or $\perp$. With the search token TK for keyword w_i and encrypted index C, the server performs the binary search according to Algorithm 1. In each round of search, it runs $\texttt{Test}$ algorithm. Once the execution of binary search is completed, it outputs the corresponding encrypted documents set $\texttt{E}(D_i)$ or a symbol $\perp$ for "not found".

Correctness. We state the following theorem about the correctness of EPKS scheme.

THEOREM 7.1. *If each algorithm is performed correctly, EPKS scheme* Π *satisfies the correctness as defined in Definition 6.1.*

PROOF. With a search token TK for keyword $w_j \in W$ and an encrypted index item of keyword w_i where $j \leq i$, the left side of Equation (1) can be computed as follows.

$$\begin{aligned}
& e(C_{i,1}, TK_1) \cdot e(C_{i,2}, TK_2) \\
= \; & e(g^{\vec{p}_i M'}, g^{M'^{-1}\vec{t}'^{\mathsf{T}}}) \cdot e(g^{\vec{p}_i'' M''}, g^{M''^{-1}\vec{t}''^{\mathsf{T}}}) \\
= \; & e(g, g)^{\vec{p}_i \vec{t}'^{\mathsf{T}} + \vec{p}_i'' \vec{t}''^{\mathsf{T}}} \\
= \; & e(g, g)^{\vec{p}_i \vec{t}^{\mathsf{T}}} \\
= \; & e(g, g)^{r\beta + a} \\
= \; & e(g, TK_3)
\end{aligned}$$

According to the feature of binary search in $\texttt{Query}$ algorithm, we observe that if the sequence number of queried keyword is less than or equal to the sequence number of index, that is, $j \leq i$, then Equation (1) holds; otherwise Equation (1) does not hold with probability $1 - 1/q$ for a large prime q.

Therefore, for a search token TK formed by a keyword $w \notin W$, the probability that $\texttt{Query}(TK, C, pp)$ correctly outputs $\perp$, once it has completed binary search, is greater than or equal to $1 - 1/q^{\lceil \log n \rceil}$. □

Security. We state the following theorem about the security of EPKS construction.

THEOREM 7.2. *EPKS scheme* Π *is statistically plaintext private and statistically predicate private when* $n \geq 2$.

7.0.3. Proof of Plaintext Privacy

LEMMA 7.3. *The EPKS scheme* Π *is statistically plaintext private when* $n \geq 2$.

PROOF. Let $\mathcal{A}$ be a computationally unbounded adversary that makes a polynomial number of queries to the encryption oracle `Enc` and the token generation oracle `TokenGen`. We prove that the distribution of $\mathcal{A}$'s view in the experiment $\mathsf{Expt}^{(0)}_{sPP1,\Pi,\mathcal{A}}$ is statistically close to the distribution of $\mathcal{A}$'s view in the experiment $\mathsf{Expt}^{(1)}_{sPP1,\Pi,\mathcal{A}}$ (we refer the reader to Definition 6.2 for the descriptions of these experiments). We denote these two distributions by $\mathsf{View}^{(0)}_{PP1}$ and $\mathsf{View}^{(1)}_{PP1}$, respectively.

Denote by $W_0^* = (w_{0,1}^*, \cdots, w_{0,n}^*)$ and $W_1^* = (w_{1,1}^*, \cdots, w_{1,n}^*)$ the two challenge keywords collections with which $\mathcal{A}$ sends to the encryption oracle `Enc`. Having already fixed hash function H and $sk = (\vec{I}, M', M'')$, we can assume that

$$\mathsf{View}^{(b)}_{PP1} = ((C_{1,1}^*, C_{1,2}^*), \cdots, (C_{n,1}^*, C_{n,2}^*)) = ((g^{\vec{p}_1' M'}, g^{\vec{p}_1' M''}), \cdots, (g^{\vec{p}_n' M'}, g^{\vec{p}_n' M''}))$$

for $b \in \{0,1\}$, where $(w_1, \cdots, w_n) = (w_{0,1}^*, \cdots, w_{0,n}^*)$ for $b = 0$, $(w_1, \cdots, w_n) = (w_{1,1}^*, \cdots, w_{1,n}^*)$ for $b = 1$.

Observe that M' and M'' are uniformly chosen from $\mathbb{Z}_q^{(n+1)\times(n+1)}$, thus for every $i \in [n]$ the distributions of $\vec{p}_i' M'$ and $\vec{p}_i'' M''$ are uniform as long as $\vec{p}_i', \vec{p}_i'' \neq \vec{0}$. Then we further prove the joint distributions of $(\vec{p}_1' M', \cdots, \vec{p}_n' M')$ and $(\vec{p}_1'' M'', \cdots, \vec{p}_n'' M'')$ are also uniform.

The above two distributions can be denoted by $P'M'$ and $P''M''$, where P' and P'' are two matrices respectively consist of $\vec{p}_1', \cdots, \vec{p}_n'$ and $\vec{p}_1'', \cdots, \vec{p}_n''$. We can infer that the two distributions of $P'M'$ and $P''M''$ are uniform as long as $\vec{p}_1', \cdots, \vec{p}_n'$ and $\vec{p}_1'', \cdots, \vec{p}_n''$ are linearly independent. We first consider the linear dependence of vectors $\vec{p}_1, \cdots, \vec{p}_n$. Note that H is a collision-resistant hash function, thus the probability of any two $H(w_i)$ and $H(w_j)$ are identical is negligible. If any two vectors of $\vec{p}_1, \cdots, \vec{p}_n$ are identical, then vectors $\vec{p}_1, \cdots, \vec{p}_n$ are linearly dependent. Since each $x_{i,j}$ in $\vec{p}_i$ is uniformly chosen from $\mathbb{Z}_q$, the probability that vector $\vec{p}_{n-1} = (H(w_1), \cdots, H(w_{n-1}), x_{n-1,n}, 1)$ and $\vec{p}_n = (H(w_1), \cdots, H(w_{n-1}), H(w_n), 1)$ are identical is $1/q$. Therefore, the probability that vectors $\vec{p}_1, \cdots, \vec{p}_n$ are linearly dependent is at most $1/q$. This implies the probability that vectors $\vec{p}_1', \cdots, \vec{p}_n'$ are linearly dependent is at most $1/q$, which is negligible, after randomly split by splitting indicator vector $\vec{I} \neq \vec{0}$. The same clearly holds also for vectors $\vec{p}_1'', \cdots, \vec{p}_n''$. Therefore, the statistical distance between $\mathsf{View}^{(0)}_{PP1}$ and $\mathsf{View}^{(1)}_{PP1}$ is negligible in λ. □

7.0.4. Proof of Predicate Privacy

LEMMA 7.4. *The EPKS scheme* Π *is statistically predicate private when* $n \geq 2$.

PROOF. Let $\mathcal{A}$ be a computationally unbounded adversary that makes a polynomial number of queries to the encryption oracle `Enc` and the token generation oracle `TokenGen`. We prove that the distribution of $\mathcal{A}$'s view in the experiment $\mathsf{Expt}^{(0)}_{sPP2,\Pi,\mathcal{A}}$ is statistically close to the distribution of $\mathcal{A}$'s view in the experiment $\mathsf{Expt}^{(1)}_{sPP2,\Pi,\mathcal{A}}$ (we refer the reader to Definition 6.3 for the descriptions of these experiments). We denote these two distributions by $\mathsf{View}^{(0)}_{PP2}$ and $\mathsf{View}^{(1)}_{PP2}$, respectively.

Denote by (w_0^*, j^*) and (w_1^*, j^*) the two challenge keywords with which $\mathcal{A}$ sends to the token generation oracle `TokenGen`. Having already fixed hash function H and $sk = (\vec{I}, M', M'')$, we can assume that

$$\mathsf{View}^{(b)}_{PP2} = (TK_1^*, TK_2^*, TK_3^*) = (g^{M'^{-1}\vec{t}'^{\intercal}}, g^{M''^{-1}\vec{t}'^{\intercal}}, g^{r\beta^* + a})$$

for $b \in \{0,1\}$, where $w_b^* = w_0^*$ for $b=0$, $w_b^* = w_1^*$ for $b=1$, and $r, a \xleftarrow{R} \mathbb{Z}_q$.

Observe that only TK_3^* is related to the underlying keyword w_b^*. The first two parts TK_1^* and TK_2^* are only related to the sequence number j^*, which is identical in terms of choosing $b \in \{0,1\}$. Thus for $b \in \{0,1\}$ we can directly infer that the distribution of TK_3^* is statistically-close to uniform, as $a \xleftarrow{R} \mathbb{Z}_q$. This implies the statistical distance between $\mathsf{View}_{PP2}^{(0)}$ and $\mathsf{View}_{PP2}^{(1)}$ is negligible in λ. □

8. FROM EPKS TO DEPKS

Due to the requirement of addition, deletion and update the keywords and documents after the index have been built, a practical SSE scheme should support dynamic updates with low communication and computation cost. Fortunately, EPKS scheme can be easily extended to dynamic settings.

In order to add new keywords into the index after they have been generated, we let ℓ be a predefined public parameter, which is the maximum dimension of vectors used in the algorithms. That is, as long as the current number of keywords is less than ℓ, a new keyword can be easily added into the index without re-setup the whole system.

Let $\Pi^D = (\mathtt{Setup}, \mathtt{Enc}, \mathtt{TokenGen}, \mathtt{UpdateToken}, \mathtt{Update}, \mathtt{Test}, \mathtt{Query})$ be a DEPKS scheme over the set of keywords $\mathcal{W}_\lambda$ consists of the following probabilistic polynomial time (PPT) algorithms as follows:

— $\mathtt{Setup}(1^\lambda, \ell) \to (pp, sk)$: On input the security parameter 1^λ and an upper bound ℓ, output public parameters pp and a secret key sk.
— $\mathtt{Enc}(W, sk, pp) \to C$: On input the keywords set $W = \{w_1, \cdots, w_n\} \subseteq \mathcal{W}_\lambda$, where $n \le \ell$, the symmetric key sk and public parameters pp, output searchable encrypted index $C = (C_1, \cdots, C_n)$.
— $\mathtt{TokenGen}(w_i, i, sk, pp) \to TK$: On input the keyword $w_i \in W$ and its sequence number i, the secret key sk and public parameters pp, output a search token TK.
— $\mathtt{UpdateToken}(w_i', i, sk, pp) \to C_i'$: On input the updated keyword w_i' and its sequence number $i \in [n+1]$, the symmetric key sk, and public parameters pp, output an updated ciphertext C_i'.
— $\mathtt{Update}(C_i', E(D_i'), i, \tau_u) \to (C', n')$: On input the updated ciphertext C_i', the corresponding encrypted document set $E(D_i')$, the sequence number $i \in [n+1]$ and the update type $\tau_u \in \{update, add, delete\}$, output a new encrypted index C' and a new counter n', which indicates the number of keywords contained in the underlying plaintext of current ciphertexts.
— $\mathtt{Test}(TK, C_i, pp) \to \{0,1\}$: On input a search token TK, each encrypted index item C_i and public parameters pp, output a bit indicating whether the encrypted index item satisfies the queried keyword corresponding to the search token.
— $\mathtt{Query}(TK, C, pp) \to \mathtt{E}(D_i)$ or $\perp$: On input a search token TK, the set of searchable encrypted index $C = \{C_1, \cdots, C_n\}$ and public parameters pp, perform binary search with running $\mathtt{Test}$ algorithm, output the candidate set of encrypted documents $\mathtt{E}(D_i)$ or $\perp$.

Correctness. The query correctness of a DEPKS scheme can be defined as follows:

Definition 8.1 (*Correctness*). For all λ, all $W \subseteq \mathcal{W}_\lambda$, letting $(pp, sk) \leftarrow \mathtt{Setup}(1^\lambda)$, $C \leftarrow \mathtt{Enc}(W, sk, pp)$, $TK \leftarrow \mathtt{TokenGen}(w_i, i, sk, pp)$, $C_i' \leftarrow \mathtt{UpdateToken}(w_i', i, sk, pp)$, $(C', n') \leftarrow \mathtt{Update}(C_i', E(D_i'), i, \tau_u)$, and the Algorithm 1 is performed correctly,

— If $w_i \in W$, and i is the sequence number of w_i in set W, then $\mathtt{Query}(TK, C, pp) = \mathtt{E}(D_i)$.
— Otherwise, $\Pr[\mathtt{Query}(TK, C, pp) = \perp] > 1 - \epsilon(\lambda)$, where $\epsilon(\lambda)$ is a negligible function.

Security. The security definitions of DEPKS are similar to the security definitions of EPKS except that the adversary is allowed to query update token generation oracle `UpdateToken` as well as encryption oracle `Enc` and search token generation oracle `TokenGen`. Note that, for all update token generation queries $(w'_{i,j}, j)$, $\texttt{Query}\left(TK_{w_0^*}, C'_i, pp\right) = \texttt{Query}\left(TK_{w_1^*}, C'_i, pp\right)$.

9. CONSTRUCTION OF DEPKS

Let $\Pi^D = (\texttt{Setup}, \texttt{Enc}, \texttt{TokenGen}, \texttt{UpdateToken}, \texttt{Update}, \texttt{Test}, \texttt{Query})$ be an DEPKS scheme. The detail construction of each algorithm is as follows.

— $\texttt{Setup}(1^\lambda, \ell) \to (sk, pp)$. This algorithm performs the same as the `Setup` algorithm of EPKS except that it generates an $(\ell+1)$-bit vector $\vec{I} \xleftarrow{R} \{0,1\}^{\ell+1}$ and two full rank matrices $M', M'' \xleftarrow{R} \mathbb{Z}_q^{(\ell+1)\times(\ell+1)}$.

— $\texttt{Enc}(W, sk, pp) \to C$. With a collection of keywords $W = \{w_1, \cdots, w_n\}$, where $n \le \ell$, the algorithm encrypts it with secret key sk and outputs the encrypted index $C = \{C_1, C_2, \cdots, C_n\}$, where each element C_i is generated as follows.

(1) Generate an $(\ell+1)$-dimensional vector $\vec{p}_i = (p_{i,1}, \cdots, p_{i,\ell+1})$ as follows. For $j = 1$ to $\ell+1$, if $j \le i$, set $p_{i,j} = H(w_j)$; if $i < j \le \ell$, set $p_{i,j} = x_{i,j} \xleftarrow{R} \mathbb{Z}_q$; if $j = \ell+1$, set $p_{i,j} = 1$.

(2) Split $\vec{p}_i$ into two vectors $\vec{p}'_i$ and $\vec{p}''_i$ with the splitting indicator $\vec{I}$ as follows. For $1 \le j \le \ell+1$, if $I_j = 0$, set $p'_{i,j} = p''_{i,j} = p_{i,j} \pmod q$; if $I_j = 1$, set $p'_{i,j} + p''_{i,j} = p_{i,j} \pmod q$, where $p'_{i,j} \xleftarrow{R} \mathbb{Z}_q$ and $p''_{i,j} = p_{i,j} - p'_{i,j} \pmod q$.

(3) Encrypt these two vectors as $C_i = (g^{\vec{p}_i M'}, g^{\vec{p}''_i M''}) = (C_{i,1}, C_{i,2})$ for keyword w_i.

— $\texttt{TokenGen}(w_i, i, sk, pp) \to TK$. With the queried keyword w_i and its sequence number i, secret key sk and public parameters pp, the algorithm creates a search token TK for keyword w_i as follows.

(1) Generate an $(\ell+1)$-dimensional vector $\vec{t} = \{t_1, \cdots, t_{\ell+1}\}$ as follows. For $j = 1$ to $\ell+1$, if $j = i$, set $t_j = r \xleftarrow{R} \mathbb{Z}_q$; if $j \ne i$ and $j \le \ell$, set $t_j = 0$; if $j = \ell+1$, set $t_j = a \xleftarrow{R} \mathbb{Z}_q$.

(2) Split $\vec{t}$ to two vectors $\vec{t}'$ and $\vec{t}''$ with the splitting indicator $\vec{I}$ as follows. For $j = 1$ to $\ell+1$, if $I_j = 1$, set $t'_j = t''_j = t_j \pmod q$; if $I_j = 0$, set $t'_j + t''_j = t_j \pmod q$, where $t'_j \xleftarrow{R} \mathbb{Z}_q$ and $t''_j = t_j - t'_j \pmod q$.

(3) Compute $\beta = H(w_i)$ and output $TK = (g^{M'^{-1}\vec{t}'^{\mathsf{T}}}, g^{M''^{-1}\vec{t}''^{\mathsf{T}}}, g^{r\beta+a}) = (TK_1, TK_2, TK_3)$ for w_i.

— $\texttt{UpdateToken}(w'_i, i, sk, pp) \to C'_i$. With the updated keyword w'_i and its sequence number i, where $i \le \ell$ and $i \in [n+1]$ and n is the current number of ciphertexts, secret key sk and public parameters pp, the algorithm generates the updated ciphertext C'_i as the same as the steps of `Enc` algorithm.

— $\texttt{Update}(C'_i, E(D'_i), i, \tau_u) \to (C', n')$: With the updated ciphertext C'_i, the corresponding encoded documents set $E(D'_i)$, the sequence number $i \in [n+1]$ of the ciphertext to be updated, where n is the current number of ciphertexts, and an update type $\tau_u \in \{update, add, delete\}$, the algorithm operates as following steps:

(1) if $\tau_u = update$ and $i \in [n]$, then replace the i-th ciphertext C_i with C'_i and the corresponding encoded documents set $E(D_i)$ with $E(D'_i)$, and set $n' = n$;

(2) if $\tau_u = add$, $i = n+1$ and $n < \ell$, then add the ciphertext C'_i at the end of the ciphertexts, i.e., add it to the $(n+1)$-th position, and let it point to the address of storing $E(D'_i)$, and set $n' = n+1$;

(3) if $\tau_u = delete$ and $i \in [n]$, then delete the corresponding encoded documents set and set $C_i = C'_i$, $E(D_i) = \phi$ and $n' = n$.

Finally, the algorithm outputs the updated ciphertexts set C' and n'.

— $\texttt{Test}(TK, C_i, pp) \rightarrow \{0, 1\}$. The algorithm performs the same as the Test algorithm of EPKS.
— $\texttt{Query}(TK, C, pp) \rightarrow \texttt{E}(D_i)$ or $\perp$. The algorithm performs the same as the Query algorithm of EPKS.

Correctness. We state the following theorem about the correctness of DEPKS scheme.

THEOREM 9.1. *If each algorithm is performed correctly, DEPKS scheme Π^D satisfies the correctness as defined in Definition 8.1.*

The proof of this theorem is similar to the proof of Theorem 7.1 except the dimension of vectors are extended to ℓ, thus we omit it here.

Security. We state the following theorem about the security of DEPKS scheme.

THEOREM 9.2. *DEPKS scheme Π^D when $\ell \geq n \geq 2$ is statistically plaintext private and statistically predicate private.*

The proofs are similar to the proofs of Lemma 7.3 and Lemma 7.4 except the dimension of vectors are extended to ℓ. The security notions still can be guaranteed by the randomness of ciphertexts, search tokens and update tokens.

9.1. The Support of Dynamic Updates

In a dynamic searchable encryption scheme, data adding, deletion and update operations should be supported without needing to either re-index the entire data collection or make use of generic and expensive dynamization techniques. The existing SSE schemes that support data dynamics were proposed in [Liesdonk et al. 2010; Kamara et al. 2012; Kamara and Papamanthou 2013; Cash et al. 2014; Stefanov et al. 2014]. The constructions in [Kamara et al. 2012; Kamara and Papamanthou 2013] support dynamically update and delete documents but require re-encrypting each node of the tree-based multi-map data structure when updating and deleting keywords. Moreover, for updates and deletions, they need 1.5 rounds of interaction (i.e., three messages between client and server). That is, the client generates an update token with the help of the server. Given such token, the server can update the index. Such interactive token generation may affect the update efficiency. As for the privacy, as far as we know, existing dynamic searchable encryption schemes [Liesdonk et al. 2010; Kamara et al. 2012; Kamara and Papamanthou 2013; Cash et al. 2014; Stefanov et al. 2014] are rely on the security notion of CKA2, which does not consider the information leakage from the deterministic update token as well as the search token.

Our DEPKS construction can address the above problems and easily fulfill more efficient updating, adding and deletion both for keywords and documents. The client generates update token without the help with the server and the update operation is completed in 1 round of interaction. Besides, in our DEPKS scheme, the update token generation algorithm is randomized as well as search token generation algorithm and encryption algorithm. Therefore, our scheme can achieve *predicate privacy*, in which given the index, search tokens and update tokens, the adversary can not obtain any information about the queried keyword.

9.1.1. Keywords Dynamic Updates. To update a keyword w_i, the client (data owner) takes as input the keyword w_i' to be updated and the location i of the updated keyword, the secret key sk and public parameters pp and runs UpdateToken to generate the updated ciphertext C_i'. Once the cloud server receives C_i', the encoded documents set $E(D_i')$, the sequence number i and an update type τ_u from the client, it performances as the three cases of algorithm Update to output the new index and the number of keywords n' after updating.

Table I. Comparison of SSE Schemes

	Scheme	Security	Index size	Search time	Dynamics
Non-randomized	SWP00 [Song et al. 2000]	CPA	N/A	$O(\|\|D\|\|)$	Yes
	Z-IDX [Goh 2003]	CKA1	$O(m)$	$O(m)$	Yes
	CM05 [Chang and Mitzenmacher 2005]	CKA1	$O(m \cdot n)$	$O(m)$	Yes
	SSE-1 [Curtmola et al. 2006]	CKA1	$O(\sum_{i=1}^{n} \|D_i\| + n)$	$O(\|D_i\|)$	No
	SSE-2 [Curtmola et al. 2006]	CKA2	$O(m \cdot n)$	$O(\|D_i\|)$	No
	KO12 [Kurosawa and Ohtaki 2012]	UC	$O(m \cdot n)$	$O(n)$	No
	KPR12 [Kamara et al. 2012]	CKA2	$O(\sum_{i=1}^{n} \|D_i\| + n)$	$O(\|D_i\|)$	Yes
	OXT [Cash et al. 2013a]	CKA2	$O(\sum_{i=1}^{n} \|D_i\| + n)$	$O(\|D_i\|)$	No
Randomized	SKPOE [Shen et al. 2009]	2PP	$O(n^2)$	$O(n^2)$	No
	SK-PE [Yoshino et al. 2012]	2PP	$O(n^2)$	$O(n^2)$	No
	EPKS	statistical 2PP	$O(n^2)$	$O(n \log n)$	No
	DEPKS	statistical 2PP	$O(n^2)$	$O(n \log n)$	Yes

Note: m is the total number of documents. n is the total number of keywords in documents. D is the ensemble of documents. $\|D\|$ is the bit length of the set of documents. $|D_i|$ is the number of documents that contain the keyword w_i. UC means universal composability. We write 2PP for both plaintext privacy and predicate privacy.

Note that the vectors in `Enc` and `TokenGen` are always ℓ-dimensional, we require that the client can append a keyword and its corresponding documents set at the end of the index as long as current number of ciphertexts is less than the preset upper bound ℓ. There are two advantages of preset maximum dimensions of vectors. The first is that it can dynamically add and delete keywords without re-building the whole index. The second is that it can keep the search correctness as long as the client generate search token and the server performs correctly after adding and deleting keywords.

9.1.2. Documents Dynamic Updates. Basically, adding and deletion a document can be done by updating the corresponding keywords. Specifically, when the client wants to add or delete a document, it respectively adds or deletes the document in the documents set corresponding to each keyword, runs `UpdateToken` algorithm to generate the new index item and sends the updated index item, the re-encrypted documents sets, sequence numbers and corresponding update types to the server. The server updates the index item and encoded documents sets in turn according to update types.

10. IMPLEMENTATION AND EVALUATION

Most existing randomized SSE schemes that achieve plaintext privacy and predicate privacy, such as SKPOE [Shen et al. 2009], SK-PE1 [Yoshino et al. 2012] and RPE [Lu 2012], are constructed based on the groups of composite order that require very large parameter sizes according to National Institute of Standards and Technology (NIST) recommendations. Moreover, the pairing computation is much slower over a composite-order than an elliptic curve [Guillevic 2013]. Fortunately, Freeman [Freeman 2010] and Lewko [Lewko 2012] provided a generic conversion from the composite-order to the prime-order setting. Based on this, Yoshino et al. gave a more efficient prime-order group instantiation, called SK-PE2 of SK-PE1 [Yoshino et al. 2012]. Therefore, our experiments mainly focus on the implementation of EPKS and DEPKS and the comparisons between EPKS and SK-PE2.

10.1. Complexity Analysis

Table I summarizes the four important characters: security, storage space complexity, search time complexity and dynamics of our schemes and other existing SSE schemes.

Note that the last four SSE schemes (including our schemes) are randomized, that is, both encryption and token generation are probabilistic, which offer stronger security i.e., plaintext privacy and predicate privacy. Therefore, we mainly compare our schemes with existing randomized SSE schemes, i.e., SKPOE [Shen et al. 2009] and SK-PE [Yoshino et al. 2012] in this paper.

As shown in Table I, compared with existing randomized SSE schemes, our schemes significantly reduces search time complexity. In addition, EPKS and DEPKS achieve information-theoretical security, i.e., statistical plaintext privacy and statistical predicate privacy.

10.2. Experimental Evaluation

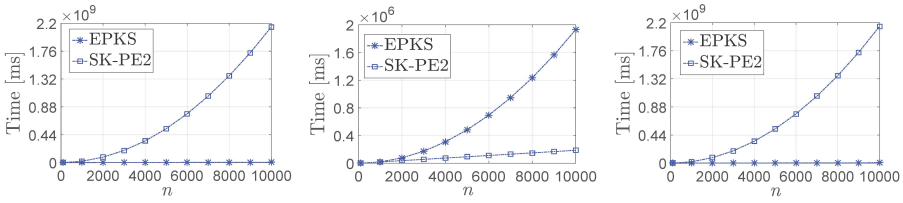
For our experiments, we build datasets indexed by different number of keywords (i.e., $n = 100, 1000, 2000, \cdots, 10000$). We encrypted the datasets with AES and encrypted the indexes with SK-PE2, EPKS and DEPKS respectively, and the encrypted data and indexes were stored on our machine. We then executed random queries over these encrypted data.

We implemented our constructions in JAVA with Java Pairing Based Cryptography library (JPBC) [JPB]. Our experiments were run on Intel(R) Core(TM) i7-3520M CPU at 2.90GHz processor and 3537MB memory size. In our implementation, the bilinear map is instantiated as Type A pairing (base field size is 512-bit), which offers a level of security equivalent to 1024-bit DLOG [JPB].

10.2.1. EPKS Implementation. Fig. 3 shows the comparisons of computation, storage and communication overhead between EPKS and SK-PE2 [Yoshino et al. 2012]. For the time cost of encrypting each index item, EPKS is less than SK-PE2 when n is small (say $n < 480$). With the increasing of n, the time cost of encryption of EPKS is grow faster than SK-PE2. Similarly, for the time cost of token generation, EPKS is less than SK-PE2 when n is small (say $n < 970$). But when n is large, EPKS is slower than SK-PE2 for generating a search token. However, index encryption and search token generation can be done offline. From the observation of Fig. 3(c), EPKS is much more efficient than SK-PE2 for query. For example, EPKS takes about 44 minutes to complete searching the whole index with 5000 keywords, in contrast to more than 6 days in SK-PE2. Moreover, with the increasing of n, this advantage becomes more significant. Fig. 3(d) and Fig. 3(e) respectively show the storage and communication overhead comparisons between EPKS and SK-PE2 [Yoshino et al. 2012]. Note that here the storage overhead and communication overhead are respectively measured by the size of index and the size of search token. The size of encrypted documents is not considered since it is decided by the size of original documents and the symmetric encryption scheme, which beyond the scope of this paper. The experimental results of Fig. 3(d) conform to the theoretical results in Table I, which shows that EPKS have the same index size with SK-PE2 [Yoshino et al. 2012]. As shown in Fig. 3(e), for the bandwidth consumption of a single query, the cost of EPKS is slightly more than SK-PE2.

10.2.2. DEPKS Implementation. To support dynamically keyword and document updates, we extend the vectors from n-dimension to ℓ-dimension in the construction of DEPKS. Therefore, the time complexity of each algorithm in DEPKS is related to the preset parameter ℓ. Here, we set $\ell = 10000$ for the following experiments. With the fixed vector dimension $\ell = 10000$, the time cost for encrypting each index item is about 64 minutes and the time cost for generating a search token is about 32 minutes.

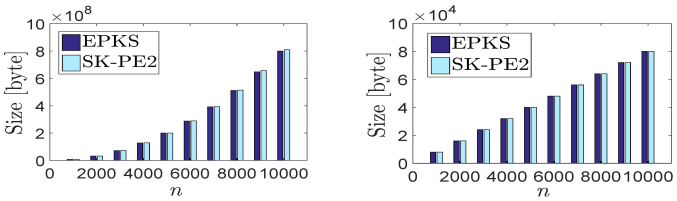
Fig. 4 shows the performance of DEPKS respectively in terms of computation, storage and communication overhead. As demonstrated in the Fig. 4(a), with the preset value $\ell = 10000$, the time cost for query increases logarithmically with the number



(a) The Computation Overhead of Encrypting One Index Item

(b) The Computation Overhead of Token Generation

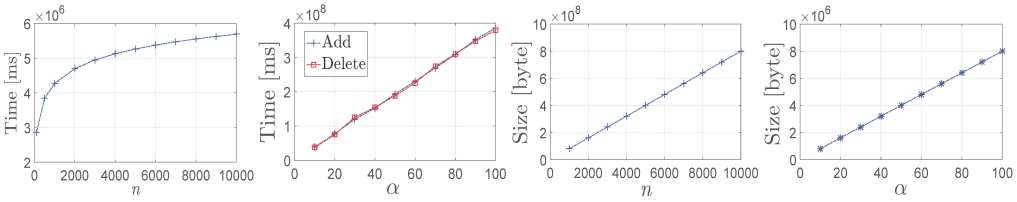
(c) The Computation Overhead of Query with One Keyword



(d) The Storage Overhead of Index

(e) The Communication Overhead of Query with One Keyword

Fig. 3. The Computation, Storage and Communication Overhead Comparisons between EPKS and SK-PE2



(a) The Computation Overhead of Query

(b) The Computation Overhead of Adding and Deleting a Document with α Keywords

(c) The Storage Overhead of Index

(d) The Communication Overhead of Adding or Deleting a Document with α Keywords

Fig. 4. The Computation, Storage and Communication Overhead of DEPKS

of keywords. For a single query, the cloud only needs to take about 95 minutes to complete search the whole index with 10000 keywords. Fig. 4(b) shows the computation overhead of adding and deleting a document with α keywords. The time cost for adding or deleting a document increases linearly with the keywords number of the document. In addition, the time costs for adding and deleting a document with same keywords number are almost the same. As shown in Fig. 4(c), the index size of DEPKS increases linearly with the number of keywords. For example, the cloud server needs about 382M to store an encrypted index with 5000 keywords. While, when the keywords number is increased to 10000, it needs about 763M to store the index. As

demonstrated in Fig. 4(d), the bandwidth consumption for adding or deleting a single document increases linearly with the number of keywords of the document.

As can be seen, although the computation costs on client side in our schemes are more than existing scheme [Yoshino et al. 2012], when the number of keywords is large (e.g., $n = 10000$), our EPKS and DEPKS schemes achieve high efficiency of query over encrypted data. The larger the index size (i.e., the number of keywords) is, the higher query efficiency gain can be achieved.

11. CONCLUSIONS

In this paper we have proposed EPKS and DEPKS scheme for inverted index-based encrypted data. First, we have described our approaches of constructing a searchable symmetric encryption scheme that supports binary search. Then, we have presented frameworks and formal security definitions both for EPKS and DEPKS. Built on the proposed frameworks, we have respectively designed scheme for EPKS and DEPKS, which are based on the groups of prime order. Both proposed schemes have high efficiency and enjoy strong notions of security, namely *statistical plaintext privacy* and *statistical predicate privacy*. Moreover, DEPKS not only maintains the properties of logarithmic-time search efficiency and high privacy, but also has fewer rounds of communication for updates, specifically, one round of interaction for each update compared to one and half round of interaction in existing DSSE schemes.

REFERENCES

The java pairing based cryptography library. http://gas.dia.unisa.it/projects/jpbc

Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. 2007. Deterministic and Efficiently Searchable Encryption. In *CRYPTO 2007*. LNCS, Vol. 4622. Springer Berlin Heidelberg, 535–552.

Dan Boneh, Giovanni Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public Key Encryption with Keyword Search. In *EUROCRYPT 2004*. LNCS, Vol. 3027. Springer Berlin Heidelberg, 506–522.

Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. 2011a. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM 2011*. 829–837.

Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. 2011b. Privacy-Preserving Query over Encrypted Graph-Structured Data in Cloud Computing. In *ICDCS'11*. 393–402.

David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *NDSS'14*.

David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013a. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO 2013*. LNCS, Vol. 8042. Springer Berlin Heidelberg, 353–373.

David Cash, Alptekin Küpçü, and Daniel Wichs. 2013b. Dynamic Proofs of Retrievability via Oblivious RAM. In *EUROCRYPT 2013*. LNCS, Vol. 7881. Springer Berlin Heidelberg, 279–295.

Yan-Cheng Chang and Michael Mitzenmacher. 2005. Privacy preserving keyword searches on remote encrypted data. In *ACNS'05*. Springer-Verlag, Berlin, Heidelberg, 442–455.

Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06*. ACM, New York, NY, USA, 79–88.

Ivan Damgård, Sigurd Meldgaard, and JesperBuus Nielsen. 2011. Perfectly Secure Oblivious RAM without Random Oracles. In *Theory of Cryptography*, Yuval Ishai (Ed.). LNCS, Vol. 6597. Springer Berlin Heidelberg, 144–163.

David Mandell Freeman. 2010. Converting Pairing-based Cryptosystems from Composite-order Groups to Prime-order Groups. In *EUROCRYPT'10*. Springer-Verlag, Berlin, Heidelberg, 44–61.

Eu-Jin Goh. 2003. Secure Indexes. Cryptology ePrint Archive, Report 2003/216. (2003). http://eprint.iacr.org/2003/216/

O. Goldreich. 1987. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *STOC '87*. ACM, New York, NY, USA, 182–194.

Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *J. ACM* 43, 3 (May 1996), 431–473.

Aurore Guillevic. 2013. Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves. In *ACNS 2013*. LNCS, Vol. 7954. Springer Berlin Heidelberg, 357–372.

Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS 2012*.

Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. 2013. Outsourced symmetric private information retrieval. In *CCS '13*. ACM, New York, NY, USA, 875–888.

Seny Kamara and Charalampos Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *FC 2013*. LNCS, Vol. 7859. Springer Berlin Heidelberg, 258–274.

Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *CCS'12*. ACM, New York, NY, USA, 965–976.

Jonathan Katz, Amit Sahai, and Brent Waters. 2008. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In *EUROCRYPT 2008*, Nigel Smart (Ed.). LNCS, Vol. 4965. Springer Berlin Heidelberg, 146–162.

Kaoru Kurosawa and Yasuhiro Ohtaki. 2012. UC-Secure Searchable Symmetric Encryption. In *FC'12*. LNCS, Vol. 7397. Springer Berlin Heidelberg, 285–298.

Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2012. On the (in)Security of Hash-based Oblivious RAM and a New Balancing Scheme. In *SODA '12*. SIAM, 143–156.

Allison Lewko. 2012. Tools for Simulating Features of Composite Order Bilinear Groups in the Prime Order Setting. In *EUROCRYPT'12*. Springer-Verlag, Berlin, Heidelberg, 318–335.

Peter Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. 2010. Computationally Efficient Searchable Symmetric Encryption. In *Secure Data Management*. LNCS, Vol. 6358. Springer Berlin Heidelberg, 87–100.

Yanbin Lu. 2012. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *NDSS 2012*.

Benny Pinkas and Tzachy Reinman. 2010. Oblivious RAM Revisited. In *CRYPTO 2010*. LNCS, Vol. 6223. Springer Berlin Heidelberg, 502–519.

Emily Shen, Elaine Shi, and Brent Waters. 2009. Predicate Privacy in Encryption Systems. In *TCC '09*. Springer-Verlag, Berlin, Heidelberg, 457–473.

Elaine Shi, T.-H.Hubert Chan, Emil Stefanov, and Mingfei Li. 2011. Oblivious RAM with O((logN)3) Worst-Case Cost. In *ASIACRYPT 2011*. LNCS, Vol. 7073. Springer Berlin Heidelberg, 197–214.

Elaine Shi and Brent Waters. 2008. Delegating Capabilities in Predicate Encryption Systems. In *ICALP '08*. Springer-Verlag, Berlin, Heidelberg, 560–578.

Dawn Xiaodong Song, D. Wagner, and A. Perrig. 2000. Practical techniques for searches on encrypted data. In *SP 2000.* 44 –55.

Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage. In *NDSS'14*.

E. Stefanov and E. Shi. 2013. ObliviStore: High Performance Oblivious Cloud Storage. In *SP 2013*. 253–267.

Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *CCS '13*. ACM, New York, NY, USA, 299–310.

Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. 2013. Privacy-preserving Multi-keyword Text Search in the Cloud Supporting Similarity-based Ranking. In *ASIA CCS '13*. ACM, New York, NY, USA, 71–82.

Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. 2012. Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data. *IEEE Transactions on Parallel and Distributed Systems* 23, 8 (2012), 1467–1479.

C. Wang, K. Ren, S. Yu, and K. Urs. 2012. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *INFOCOM 2012*. 451–459.

Qian Wang, Shengshan Hu, Kui Ren, Meiqi He, Minxin Du, and Zhibo Wang. 2015. *ESORICS 2015*. Springer International Publishing, Cham, Chapter CloudBI: Practical Privacy-Preserving Outsourcing of Biometric Identification in the Cloud, 186–205.

Masayuki Yoshino, Noboru Kunihiro, Ken Naganuma, and Hisayoshi Sato. 2012. Symmetric Inner-Product Predicate Encryption Based on Three Groups. In *Provable Security*. LNCS, Vol. 7496. Springer Berlin Heidelberg, 215–234.

Online Appendix to: Dynamic and Efficient Private Keyword Search over Inverted Index-Based Encrypted Data

RUI ZHANG, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences
RUI XUE, SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences
TING YU, Qatar Computing Research Institute, Hamad Bin Khalifa University
LING LIU, College of Computing, Georgia Institute of Technology

 1533-5399/2016/-ART0000 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000