IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 10, 2020, accepted May 31, 2020, date of publication June 9, 2020, date of current version June 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3001014

Practical Searchable Symmetric Encryption Supporting Conjunctive Queries Without Keyword Pair Result Pattern Leakage

CHANGSHE MA, YIPING GU, AND HONGFEI LI
School of Computer Science, South China Normal University, Guangzhou 510631, China
Corresponding author: Yiping Gu (yipinggu6@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672243.

ABSTRACT The research of searchable symmetric encryption (SSE) has been focused on the trade-off between security, performance, and functionality. Recently proposed Hidden Cross-Tags (HXT) protocol improves the Oblivious Cross-Tags (OXT) by avoiding the leakage of 'Keyword Pair Result Pattern' (KPRP), at the cost of increasing the storage size by two orders of magnitude. In this paper, we reconsider the principle of designing SSE protocols to prevent KPRP leakage. At first, we introduce a new primitive called subset membership check (SMC), where a set is encrypted such that its subset membership can be checked only through a protocol between Sender and Tester. The holistic security of SMC requires that nothing is revealed other than the membership of a subset after each execution of the protocol. We propose a hash-based SMC implementation with efficient computation, communication, and storage. Secondly, based on the hash-based SMC, we present a practical SSE protocol that supports conjunctive queries without KPRP leakage. Our SSE protocol, called 'Practical Hidden Cross-Tags' (PHXT), maintains the same storage size as OXT while preserving the same privacy and functionality as HXT. Compared with HXT, our PHXT reduces the storage size and the communication overhead by 91.29%, 64.29%, respectively, without degrading the computational efficiency.

INDEX TERMS Conjunctive query, encrypted bloom filter, holistic security, keyword pair result pattern, searchable symmetric encryption, subset membership check.

I. INTRODUCTION

Data outsourcing is a popular cloud service which reduces the burden of local resource consumption. In this setting, the data owner usually encrypts its data locally before outsourcing to prevent the server (with a corrupted insider) from accessing it. However, the data encrypted in the standard way no longer supports search functionality. To enable search functionality, a trivial solution is to send the encrypted database to the owner, which decrypts it and performs queries on the plaintext. This solution is out of acceptance due to the huge communication overheads and computational costs. Such a scenario has motivated a plenty of researches on searchable encryption schemes.

Searchable symmetric encryption (SSE) [1]–[6] provides a mechanism to search on encrypted data with acceptable performance and security level. The first practical SSE scheme was proposed by Song *et al.* [7], after which a lot of SSE schemes have been seen in the literature. Each of the existing SSE schemes has a trade-off between performance, security, and functionality. The performance consists of the storage size, the computational costs, and the communication overheads, which includes interaction rounds as well as the bandwidth between the client and the server. The security mainly focuses on reducing the leakage of queries, the less leakage, the better security level. The functionality mainly requires the SSE scheme to support query operations (such as conjunctive query, insert operation, and delete operation) as many as possible.

Especially, the performance of an SSE scheme is crucial for practical considerations. In [3], Cash *et al.* proposed the concept of 'Cross-Tags Set' (XSet) which is the basic building block for SSE schemes (such as the 'Oblivious Cross-Tags' (OXT) protocol [3] and the 'Hidden Cross-Tags' (HXT) protocol [8]) to support efficient conjunctive queries with worst-case sublinear complexity. Usually, XSet is represented by

The associate editor coordinating the review of this manuscript and approving it for publication was Amir Masoud Rahmani.

a bloom filter. The OXT protocol uses the standard bloom filter, which can be RAM-resident easily, while allowing the 'Keyword Pair Result Pattern' (KPRP) leakage exploited in recent attacks [9]–[11]. The HXT protocol prevents the KPRP leakage by encrypting the bloom filter of XSet bit by bit. Specifically, the HXT protocol uses a symmetric-key hidden vector encryption (HVE) scheme to encrypt each bit of its bloom filter to reduce the leakage from KPRP to the 'Whole Result Pattern' (WRP). Usually, HVE encrypts each bit to a ciphertext of 128 bits, which implies that the size of the encrypted bloom filter in HXT is larger than OXT by a factor about 128. As a result, the encrypted bloom filter will be too large to be RAM-resident. For example, the standard bloom filter size of wiki data [1] is about 3G bytes, then the encrypted bloom filter of HXT should be at least 384G bytes. For practicality, the smaller the size the bloom filter possesses, the better performance the SSE protocols attain. Motivated by these practical scenarios, we ask the following question.

Can we get the best of both worlds: prevent the KPRP leakage while having a small size bloom filter as in OXT?

We make progress on this question and give an affirmative answer by constructing a practical SSE scheme. Our construction has been focused on the conjunctive queries, like OXT and HXT, since such queries are the most common in many practical settings. In our construction, we assume that the server is honest but curious and there are only one reader and one writer.

Our Contributions: In this paper, we reconsider the principle of preventing KPRP leakage in SSE schemes with support for conjunctive queries. The contributions of this paper are twofold.

- Firstly, we introduce the concept of subset membership check (SMC) and its security model to test the membership of a subset holistically. SMC encrypts a set such that its subset membership can be checked only through a protocol between Sender and Tester. The security of SMC requires that nothing is revealed other than the membership of a subset after each execution of the protocol. We also propose a hash-based SMC (HSMC) implementation with efficient computation, communication, and storage. SMC enables a modular design of SSE schemes to prevent KPRP leakage.
- Secondly, based on HSMC, we propose an efficient SSE protocol, called 'Practical Hidden Cross-Tags' (PHXT), which combines the advantages of both OXT and HXT. Specifically, PHXT maintains the optimal storage size as OXT while improving its leakage from KPRP to WRP. Thanks to our hash-based SMC, the bloom filter of PHXT is encrypted without any ciphertext expansion and can be RAM-resident conveniently. Compared to OXT, our PHXT avoids the KPRP leakage while maintaining the same storage size. Compared to HXT, our PHXT reduces the storage size and communication overheads by 91.29% and 64.29%, respectively (see table 4), while keeping the same security level.

[1]enwiki-20190328-pages-articles22.xml

Organization: The rest of the paper is organized as follows. Section II surveys the related work. Section III provides the related notations and primitives, while other needed hardness assumptions and cryptographic primitives are given in Appendix VIII. In Section IV, we give the concept of SMC and an efficient hash-based implementation. We construct a practical SSE protocol in Section V, while its performance comparison and evaluations are given in Section VI and Section VII, respectively. In the last section, it is a conclusion.

II. RELATED WORK

Detailed reviews about SSE are referred to [12] and [13]. Here, we only review SSE schemes which support boolean queries. The early proposed SSE schemes [14]–[17] usually use a keyword field to support conjunctive queries and are provably secure against the indistinguishability under chosen keyword attacks (IND-CKA) in the random oracle (RO) model or standard model. The schemes of [14] and [15] suffer from linear communication cost which makes them be substantially impractical. The scheme of [16] improves [14], [15] with the constant cost of both communication and storage of each user at the cost of each encryption requiring one extra pairing operation per keyword per document. In 2007, Ryu and Takagi [17] proposed a conjunctive SSE construction that maintains the advantages of [16] while reducing the computational cost of each encryption without using any pairing operation per keyword per document. Subsequently, Wang *et al.* [18] constructed the first keyword field-free conjunctive SSE scheme with the size of each trapdoor for a document index linear in the number of keywords it contains. Furthermore, each query requires a bilinear map per keyword per document index.

All of the dedicated conjunctive SSE schemes mentioned above have linear search complexity. Indeed, there is a trivial sublinear conjunctive construction firstly proposed by Curtmola *et al.* [1] through reducing each conjunctive query to a series of single-keyword queries based on a masked index in inverted form. More specifically, the client allows the server to search for each individual keyword included in a conjunctive query. Then, the client can find the query result by intersection operations between the matching sets. However, this trivial solution not only reveals the set of documents matching each individual keyword, but also has the complexity proportional to the number of documents matching the most frequent keyword.

Recently, Cash *et al.* proposed the first non-trivial sublinear conjunctive scheme named OXT [3] with the complexity proportional to the number of documents matching the least frequent keyword. It guarantees an acceptable leakage and is provably IND-CKA secure under the decisional Diffie-Hellman (DDH) assumption, while supporting large-scale databases. OXT uses the XSet data structure, which is a collection of hash values derived from keyword-document identifier pairs, to represent relations between keyword and document. For the sake of efficiency, XSet is represented by a standard bloom filter, which is easy to be RAM-resident while allowing the KPRP leakage. Moreover, DH-type operations are imperative for OXT to guarantee the privacy, which constitutes the main computation cost of its search protocol. In 2018, Lai *et al.* proposed the HXT scheme [8], which improves OXT by replacing the KPRP leakage with the WRP leakage at the cost of enlarging the length of bloom filter by two orders of magnitudes. Specifically, HXT uses a symmetric hidden vector encryption (HVE) scheme to encrypt the bloom filter bit by bit, resulting in an encrypted bloom filter whose length is enlarged by a factor of at least 128. The advantage of OXT will be degraded since it is too expensive to implement a RAM-resident bloom filter with large size.

III. PRELIMINARIES

In this section, we give notations used throughout this paper and review the definition and security notions of the SSE scheme. We also describe the necessary building blocks such as T-Set and bloom filter for our constructions.

Notations: Let $C(\cdot)$ be a probabilistic polynomial time (PPT) algorithm, $c \leftarrow C(\cdot)$ represents that c is assigned the output of $C(\cdot)$. Let a, z be integers and $a \leq z$, $[a, z]$ represents the set $\{a, a+1, \cdots, z\}$, while $[z]$ represents $[1, z]$. Let S be a set or a sequence, $|S|$ represents the number of elements in S and $S^z = \{(s_1, s_2, \cdots, s_z)|s_i \in S \text{ for } i = 1, 2, \cdots, z\}$. $x \overset{\$}{\leftarrow} X$ denotes that x is sampled uniformly at random from the set X. $x \leftarrow \mathcal{D}$ denotes that x is sampled from the distribution $\mathcal{D}$. For a vector (or binary string) $\mathbf{v}$, $\mathbf{v}[i]$ denotes the i-th component (bit) of $\mathbf{v}$. For two binary strings a and b, $a||b$ denotes the concatenation of a and b. The function $neg(\lambda) : Z \mapsto (0, 1)$ is said to be negligible, if for every positive polynomial $p(\cdot)$ there exists an integer N_0 such that $neg(\lambda) < \frac{1}{p(\lambda)}$ for all $\lambda > N_0$. Let $\emptyset$ denotes the empty set or string.

A database denoted as $\mathsf{DB} = (id_i, W_i)_{i=1}^{d}$ represents that DB includes d documents, and the $i-th$ document is identified by the identifier id_i, which contains a set of keywords W_i. Let $W = \bigcup_{i=1}^{d} W_i$ and $m = |W|$, and $ID = \{id_i \mid 1 \leq i \leq d\}$. Also let $\mathsf{DB}[w]$ denote the set of identifiers including the keyword w. For a boolean query $\phi(\bar{w})$, $\mathsf{DB}[\phi(\bar{w})]$ represents the set of identifiers that satisfies the query $\phi(\bar{w})$.

A. SEARCHABLE SYMMETRIC ENCRYPTION: SYNTAX AND SECURITY

1) SYNTAX

In the single-writer single-reader setting, the data owner (called client) encrypts his database DB and sends the encrypted database EDB to the data server (called server), while maintaining the ability to search throughout the encrypted database with acceptable leakage. To enable search on the encrypted database, an SSE scheme usually generates some extra indexes over the plaintexts of documents before they are encrypted. Then, the ciphertexts of the documents and the indexes are delivered to the server, which stores them separately. Whereas the subsequent search process is mainly arranged on the indexes.

Definition 1: Let λ be the security parameter. An SSE scheme $\Pi = \{\mathsf{EDBSetup}, \mathsf{Search}\}$ consists of the algorithm $\mathsf{EDBSetup}$ and the protocol Search. The algorithm $\mathsf{EDBSetup}$ is run within the client. The protocol Search involves several rounds of interactions between the client and the server. They are detailed as follows.

- $(SK, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(1^\lambda, \mathsf{DB})$: An algorithm run in the client. Given a database DB, the client outputs a secret key set SK along with an encrypted database EDB, where SK is kept secretly in the client, while EDB is provided to the server.
- $\mathsf{DB}[\phi(\bar{w})] \leftarrow \mathsf{Search}(SK, \phi(\bar{w}), \mathsf{EDB})$: A protocol run between the client and the server interactively. The client's inputs are secret key set SK and a query $\phi(\bar{w})$, while the server's input is EDB. Finally, the client outputs the set of identifiers $\mathsf{DB}[\phi(\bar{w})]$ matching the query $\phi(\bar{w})$, while the server outputs nothing.

Let $\mathcal{A}$ be any PPT algorithm. The SSE scheme Π is said to be correct if the game $Cor_{\mathcal{A}}^{\Pi}(\lambda)$ defined as follows outputs 0 with all but negligible probability in λ. In $Cor_{\mathcal{A}}^{\Pi}(\lambda)$, the game runs $(SK, \mathsf{EDB} \leftarrow \mathsf{EDBSetup}(1^\lambda, \mathsf{DB})$, where DB is chosen by $\mathcal{A}$. It provides EDB to $\mathcal{A}$ who then adaptively chooses queries $\phi(\bar{w})$. For each query, the game runs Search protocol to get the answer S with inputs SK, $\phi(\bar{w})$, and EDB. The game outputs 1 if $S \neq \mathsf{DB}[\phi(\bar{w})]$ for some query $\phi(\bar{w})$, otherwise, the game outputs 0.

2) SECURITY

Let $\mathcal{L}$ be a leakage function that describes what an honest but curious server can learn from the encrypted database and queries during its interaction with an honest client. Informally, we say that an SSE scheme Π is secure if it guarantees that no information about queries and the database is revealed beyond the pre-described leakage function $\mathcal{L}$. If this holds for any non-adaptively adversary, then Π is said to be non-adaptive secure. On the other hand, if the adversary behaves adaptively, then Π is said to be adaptively secure. A formal security definition of SSE scheme was first proposed by Curtmola *et al.* [1] and later generalized by Chase *et al.* in structured encryption [19].

Definition 2: Let $\Pi = (\mathsf{EDBSetup}, \mathsf{Search})$ be an SSE scheme. For algorithms $\mathcal{A}$ and $\mathcal{S}$, we define experiments(algorithms) $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$ as follows:

- $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$: $\mathcal{A}(1^\lambda)$ chooses a database DB and gives it to the experiment which outputs $(SK, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(1^\lambda, \mathsf{DB})$. Given EDB, $\mathcal{A}$ adaptively chooses a polynomial number of queries. For each query q, the experiment runs the protocol $\mathsf{Search}(SK, q, \mathsf{EDB})$ and provides the transcript and the client's output to $\mathcal{A}$. At the end, the experiment outputs the bit returned by $\mathcal{A}$.
- $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$: $\mathcal{A}(1^\lambda)$ chooses a database DB, and the experiment runs $\mathsf{EDB} \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{DB}))$ and gives EDB to $\mathcal{A}$. Then $\mathcal{A}$ adaptively chooses a polynomial number

of queries. To respond, the experiment provides $\mathcal{A}$ with the output of $\mathcal{S}(\mathcal{L}(\mathsf{DB}, \mathbf{Q}))$, where $\mathbf{Q}$ is all of the queries issued by $\mathcal{A}$ so far. At the end, the experiment outputs the bit returned by $\mathcal{A}$.

We say that Π is $\mathcal{L}$-semantically-secure against adaptive attacks if for any PPT adversary $\mathcal{A}$ there exists a PPT algorithm $\mathcal{S}$ such that $|Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq neg(\lambda)$, where neg is a negligible function.

A *non-adaptive* version of this definition is a straightforward modification of the above game where the adversary provides all of the queries at the start of the game.

B. T-SET

A T-Set [3] is a special inverted-index data structure used for constructing efficient SSE schemes. The detailed descriptions of its notations, definitions, and security results are referred to [3]. Formally, a T-Set instantiation Σ = (Setup, GetTag, Retrieve) will consist of three algorithms. In this paper, we use the dictionary encryption, e.g., the Π_{bas} scheme of [20], to implement the T-Set, which is described as follows.

- $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.Setup}(\mathbf{T})$: It begins with selecting a secret key K_T and preparing a list L. For each $w \in W$, it computes $K_1 \parallel K_2 \leftarrow F(K_T, w)$, where F is a keyed PRF, then for each $c \in [1, |\mathbf{T}[w]|]$, it computes $\ell \leftarrow F(K_1, c)$ and $d \leftarrow \mathsf{Enc}(K_2, \mathbf{T}[w, c])$, and adds (ℓ, d) into a list L. Finally, the list L is translated into a dictionary TSet.
- $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, w)$: It generates $K_1 \parallel K_2 \leftarrow F(K_T, w)$, and returns $\mathsf{stag} \leftarrow K_1 \parallel K_2$ as the token corresponding to w.
- $\mathbf{T}_w \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{stag})$: It pares stag as $K_1 \parallel K_2$, prepares a list $\mathbf{T}_w$, and initializes $c = 1$ and $\ell = F(K_1, c)$. Then it does as follows: while $\mathsf{TSet}(\ell) \neq \perp$, it sets $\mathbf{T}_w[c] \leftarrow \mathsf{Dec}(K_2, \mathsf{TSet}(\ell))$, $c \leftarrow c + 1$, and $\ell \leftarrow F(K_1, c)$.

For an array $\mathbf{T}$ and a sequence of keywords $\mathbf{w}$, we define the leakage function $\mathcal{L}_T(\mathbf{T}, \mathbf{w})$ as the tuple (N, SP, RP), where N, SP, and RP are detailed as follows.

- N is the total number of keyword-identifier pairs, and equals $\Sigma_{w\in W}|\mathbf{T}[w]|$.
- SP is the search pattern of queries, revealing the equality pattern of queries. Initialize an empty list Q_{srch}, for the i-th query w, add (i, w) into Q_{srch}, and the search pattern for w is defined by $SP(w, Q_{srch}) = \{j \mid (j, w) \in Q_{srch}\}$, indicating the equality pattern of the queries Q.
- RP is the result pattern of queries. Formally, for the i-th query w, $RP[i] = \mathbf{T}[w]$.

Lemma 1 [20]: In the random oracle model, Π_{bas} is $\mathcal{L}_T$-adaptively-secure if F is a secure PRF and (Enc, Dec) is an IND-CPA secure symmetric encryption.

C. BLOOM FILTERS

A bloom filter is a probabilistic data structure used to represent a set of N elements. Its main functionality is to test the membership of an element. Let $S = \{s_1, s_2, \cdots, s_N\}$ be a set. The bloom filter first initializes a binary vector BF with length m, such that $\mathsf{BF}[i] = 0$ for all $i \in [m]$. Then, it chooses k independent hash functions $\{H_i\}_{1\leq i\leq k}$ each of which maps elements from S to $[m]$. To add the element s_j into the bloom filter, it sets $\mathsf{BF}[H_i(s_j)] = 1$ for all $1 \leq i \leq k$. To test the membership of an element s, it only needs to verify $\mathsf{BF}[H_i(s)] = 1$ for all $1 \leq i \leq k$. If it is the case, then return 1, 0 otherwise. When the result of a test is positive, there is a possibility (called false positive) P_e that the bloom filter has made a wrong answer. Actually, $P_e \leq (1 - e^{-k\cdot N/m})^k$ [21]. For a set with fixed number of elements, there is a trade-off between m, k and P_e. Given N, the smaller k is preferred when the corresponding false positive is accepted. The optimal choice of k is $k \approx \log_2(1/P_e)$, while $m \approx 1.44 \log_2 (1/P_e) \cdot N$ [21].

IV. SUBSET MEMBERSHIP CHECK

In this section, we introduce the concept of subset membership check and give an efficient hash-based implementation.

A. SYNTAX AND PRIVACY

Let Σ be a pre-defined set. Let $\Omega \subseteq \Sigma$ be the queried set. There are two players, the Sender and the Tester. Ω is encrypted as ESet, which is passed to the Sender. Subsequently, they want to check the membership of a subset $V = \{e_1, e_2, \cdots, e_n\}$, i.e. $V \subseteq \Omega$ or not. A trivial method is to authorize the Sender to check the membership of each element of V one by one. However, this allows the Sender to know which element is in Ω and which element is not. Indeed, we want to require that no information is leaked, except for the fact that $V \subseteq \Omega$ or not.

Definition 3: A subset membership check scheme SMC = {Setup, Test} consists of the algorithm Setup to generate an encrypted representation ESet and a key K_0, and the protocol Test between the Sender and the Tester to check the membership of a subset. The detailed description of SMC is as follows.

- $(\mathsf{ESet}, K_0) \leftarrow \mathsf{SMC.Setup}(1^\lambda, \Omega)$: takes as input a set Ω and the security parameter λ and outputs an encrypted representation ESet of Ω along with a secret key K_0, where ESet is provided to the Sender, K_0 is provided to the Tester.
- $b \leftarrow \mathsf{SMC.Test}(\mathsf{ESet}, V, K_0)$: takes as input the encrypted representation ESet, the subset V and the key K_0 and provides Sender with a bit b which indicates $V \subseteq \Omega$ or not. During the protocol Test, Sender generates the subset V in advance. After interactions with Tester, Sender obtains a bit b. If $b = 1$, then $V \subseteq \Omega$, $V \nsubseteq \Omega$ otherwise.

We say that an SMC scheme is computationally correct if for all security parameter λ, all sets Ω, and all subsets V, after sequentially running $\mathsf{SMC.Setup}(1^\lambda, \Omega)$ to get ESet and K_0 and $\mathsf{SMC.Test}(\mathsf{ESet}, V, K_0)$ to get b, $Pr[b = 1] \geq 1 - neg(\lambda)$ if $V \subseteq \Omega$, $Pr[b = 0] = 1 - neg(\lambda)$ otherwise.

Algorithm 1 HSMC

```
Setup
Input: $1^\lambda, \Omega$
Output: $K_0$, EBF
1: Select key $K_0$ for PRF $F_0$.
2: Select hash functions $\{H_j\}_{1\leq j\leq k}$ for BF, and hash function $H$ for holistic check.
3: Initialize BF $\leftarrow 0^m$.
4: for each $e \in \Omega$ do
5:     for $j = 1 : k$ do
6:         BF$[H_j(e)] \leftarrow 1$
7:     end for
8: end for
9: for $i = 1 : m$ do
10:     EBF$[i] \leftarrow$ BF$[i] \bigoplus F_0(K_0, i)$
11: end for
12: return EBF and $K_0$

token
Input: EBF and a subset $V = \{e_1, e_2, \cdots, e_n\}$
Output: Token tk
1: $d = \emptyset$
2: for $i = 1 : n$ do
3:     for $j = 1 : k$ do
4:         $d \leftarrow d||$ EBF$[H_j(e_i)]$
5:     end for
6: end for
7: tk $\leftarrow H(d)$
8: return tk

Check
Input: $K_0$, tk and a subset $V = \{e_1, e_2, \cdots, e_n\}$
Output: Result $b$
1: $d = \emptyset$
2: $b \leftarrow 0$
3: for $i = 1 : n$ do
4:     for $j = 1 : k$ do
5:         $d \leftarrow d||(1 \bigoplus F_0(K_0, H_j(e_i)))$
6:     end for
7: end for
8: if tk $= H(d)$ then
9:     $b \leftarrow 1$
10: end if
11: return $b$

Protocol: Test
Input: Sender's inputs are the subset $V = \{e_1, e_2, \cdots, e_n\}$ and EBF. Tester's input is $K_0$.
Output: Result $b$
1: Sender computes tk $\leftarrow$ token(EBF, $V$).
2: Sender sends $V$ and tk to Tester.
3: Tester computes $b \leftarrow$ Check($K_0$, tk, $V$).
4: Tester sends $b$ to Sender.
5: return $b$
```

Privacy: Informally, we say that a subset membership check scheme SMC is private if it guarantees that no information is leaked, except for the fact that $V \subseteq \Omega$ or not. Let $\mathcal{L}_{\mathsf{SMC}}$ be a leakage function that describes what an adversary can learn from the subset membership check scheme SMC in addition to the trivial leakage that $V \subseteq \Omega$ or not. A formal privacy definition is described below.

Definition 4: Let SMC $=$ (Setup, Test) be a subset membership check scheme. For PPT algorithms $\mathcal{A}$ and $\mathcal{S}$, we define experiments (algorithms) $\mathbf{Real}^{\mathsf{SMC}}_{\mathcal{A}}(\lambda)$ and $\mathbf{Ideal}^{\mathsf{SMC}}_{\mathcal{A},\mathcal{S}}(\lambda)$ as follows:

- $\mathbf{Real}^{\mathsf{SMC}}_{\mathcal{A}}(\lambda)$: $\mathcal{A}(1^\lambda)$ outputs a set Ω. The experiment computes (ESet, K_0) $\leftarrow$ SMC . Setup($1^\lambda, \Omega$) and gives ESet to $\mathcal{A}$. Then, $\mathcal{A}$ adaptively chooses a polynomial number of queries. For each query V, the experiment runs the protocol SMC . Test(ESet, V, K_0) and gives the transcript to $\mathcal{A}$. At the end, the experiment outputs the bit returned by $\mathcal{A}$.
- $\mathbf{Ideal}^{\mathsf{SMC}}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}(1^\lambda)$ outputs a set Ω. The experiment runs ESet $\leftarrow \mathcal{S}(\mathcal{L}_{\mathsf{SMC}}(\Omega))$ and gives ESet to $\mathcal{A}$. Then $\mathcal{A}$ adaptively chooses a polynomial number of queries. To respond, the experiment provides $\mathcal{A}$ with the output of $S(\mathcal{L}_{\mathsf{SMC}}(\Omega, \mathbf{V}))$, where $\mathbf{V}$ is all of the queries issued by $\mathcal{A}$ so far. At the end, the experiment outputs the bit returned by $\mathcal{A}$.

We say that SMC is $\mathcal{L}_{\mathsf{SMC}}$-semantically-secure against adaptive attacks if for any PPT adversary $\mathcal{A}$ there exists a PPT algorithm $\mathcal{S}$ such that $|Pr[\mathbf{Real}^{\mathsf{SMC}}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}^{\mathsf{SMC}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq neg(\lambda)$, where *neg* is a negligible function. We say that an SMC scheme has holistic security if the leakage function $\mathcal{L}_{\mathsf{SMC}}$ just reveals some trivial information about Ω or $\mathbf{V}$, from which $\mathcal{A}$ cannot derive any non-trivial result about subset membership.

The symmetric HVE scheme used in HXT [8] can be deemed as an SMC scheme with holistic security, as its leakage function only reveals the size of Ω and the wildcard pattern of each query.

B. HASHED-BASED SUBSET MEMBERSHIP CHECK

In this section, we represent the queried set Ω as a bloom filter BF and propose a hash-based subset membership check scheme (called HSMC) for Ω without enlarging the length of BF. A detailed description of HSMC can be found in Algorithm 1.

Let F_0 be a pseudorandom function with domain $\{0, 1\}$. Given the security parameter λ and the queried set Ω, the algorithm Setup outputs a secret key K_0 for F_0 and an encrypted bloom filter EBF for Ω. At first, the algorithm Setup creates the bloom filter of Ω in the standard way. Then, it encrypts BF bit by bit to get the encrypted bloom filter EBF.

More specifically, $\mathsf{EBF}[u] \leftarrow \mathsf{BF}[u] \bigoplus F_0(K_0, u)$ for all $u \in [m]$, where m is the length of BF. We note that each bit of BF is encrypted into one-bit ciphertext rather than a block of at least 128 bits as in HXT. Finally, K_0 is kept by Tester and EBF is stored by Sender.

To facilitate invocations, we abstract the main computation of the protocol Test in HSMC as two algorithms, token and Check. The algorithm token takes as input an encrypted bloom filter EBF and a subset V and outputs a token tk. The algorithm Check takes as input a token tk, a subset V and a key K_0 and outputs a bit b to indicate whether $V \subseteq \Omega$ or not. The protocol Test proceeds as follows. At first, Sender generates the token tk through the algorithm token. Then, Sender passes tk to Tester. After receiving tk, Tester gets the bit b through the algorithm Check. Finally, Tester passes the bit b to Sender. If $b = 1$, then $V \subseteq \Omega$, otherwise, $V \nsubseteq \Omega$. The detailed descriptions of the two algorithms token and Check are given as below.

Algorithm token*:* Given a subset $V = \{e_1, e_2, \cdots, e_n\}$ and EBF, the algorithm token first concatenates EBF's bits corresponding to each element of V. Concretely, let d be a nk-bit string whose $((i-1)k + j)$-th bit equals $\mathsf{EBF}[H_j(e_i)]$ for $1 \leq i \leq n$ and $1 \leq j \leq k$. Then, it generates a token tk through a hash function H taking as input d. Finally, it outputs the token tk.

Algorithm Check*:* Given the secret key K_0, a subset V, and the token tk corresponding to V, the algorithm Check outputs a boolean value b to judge that whether $V \subseteq \Omega$ or not. At first, it constitutes the string d by concatenating $1 \bigoplus F_0(K_0, H_j(e_i))$ $(1 \leq i \leq n$ and $1 \leq j \leq k)$ bit by bit. Then, it verifies that $\mathsf{tk} = H(d)$. If it is the case, then return 1, 0 otherwise.

For any subset $V = (e_1, e_2, \cdots, e_{|V|})$, we define a predicate $P_\Omega(V)$, such that

$$P_\Omega(V) = \prod_{1 \leq i \leq |V|} \Big(\prod_{1 \leq j \leq k} \mathsf{BF}(H_j(e_i)) \Big).$$

Correctness: Let $V = (e_1, e_2, \cdots, e_{|V|})$. If $V \subseteq \Omega$, then $P_\Omega(V) = 1$, which implies that $\mathsf{BF}[u_{ij}] = 1$ and $\mathsf{EBF}[u_{ij}] = 1 \bigoplus F_0(K_0, u_{ij})$, for each $u_{ij} \leftarrow H_j(e_i)$, where $i \in [1, |V|]$, $j \in [1, k]$. Obviously, the string computed in token is the same as that computed in Check, and hence $b = 1$. Otherwise, $V \nsubseteq \Omega$, then $P_\Omega(V) = 0$ happens with probability $\geq (1 - P_e)^{|V|}$, where P_e is the false positive of BF. There must exist $\ell \in \{H_j(e_i)\}_{1 \leq i \leq |V|, 1 \leq j \leq k}$ such that $\mathsf{BF}[\ell] = 0$, which implies that the string computed in token is different from that computed in Check. So $b = 1$ holds with negligible probability.

Privacy: For a sequence of queried subsets $\mathbf{V}$, we define two leakage functions of HSMC: $\mathcal{L}_{\mathsf{HSMC}}(\Omega) = (N)$, $\mathcal{L}_{\mathsf{HSMC}}(\Omega, \mathbf{V}) = (CP, \mathbf{b})$, where the tuples N, CP, and $\mathbf{b}$ are detailed as follows.

- $N = |\Omega|$ is the total number of elements included in the set Ω.
- CP is the check pattern of subsets $\mathbf{V}$, recording locations of each subset $\mathbf{V}[i]$ derived from the BF hash functions, i.e., $CP[i, j, k] \leftarrow H_k(\mathbf{V}[i, j])$, where $\mathbf{V}[i, j]$ denotes the j-th element of the i-th queried subset.
- $\mathbf{b}$ is the result pattern, where $\mathbf{b}[i]$ indicates that $\mathbf{V}[i] \subseteq \Omega$ or not.

Theorem 1: Our hash-based subset membership check scheme HSMC is $\mathcal{L}_{\mathsf{HSMC}}$-semantically-secure against adaptive attacks, assuming that F_0 is a secure PRF, and that H is a collision-resistant hash function.

Proof: The proof is detailed in Appendix -A. ■

We note that HSMC has holistic security since its leakage functions described above only reveal nothing but the trivial information.

V. PHXT CONSTRUCTION

In this section, we propose a practical SSE protocol, the 'Practical Hidden Cross-Tags' (PHXT) protocol, which supports conjunctive queries and prevents KPRP leakage. A detailed description of it is as follows.

We observe that it is the ability of the server to check whether a keyword is contained in a document of s-term [3] that leads to the KPRP leakage. For a set of keywords KW and a document D, if there is a holistic check whether $KW \subseteq D$, then the KPRP leakage is prevented intuitively. The HXT protocol uses HVE to achieve a holistic check which circumvents the requirement that the server has to check the relation between keyword and document one by one, while resulting in a large representation of XSet. In PHXT, we use subset membership check to achieve holistic check of the relation between a keyword set and a document. Moreover, the operation of a holistic check of PHXT is different from HXT. In PHXT, the holistic check is performed by the client rather than by the server as in HXT.

A. PHXT*: PRACTICAL HIDDEN CROSS-TAGS PROTOCOL*

Let $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption (SE) scheme. The PHXT protocol consists of two algorithms: EDBSetup and Search.

Given the security parameter λ and the database DB, the algorithm EDBSetup (Algorithm 2) outputs the secret key-set mk kept on the client and the encrypted database EDB stored on the server. EDB consists of TSet and the encrypted vector $\mathbf{c}$. TSet is generated exactly the same as in OXT and HXT, which is an encrypted inverted index of DB. The encrypted vector $\mathbf{c}$ is derived from the algorithm HSMC . Setup, taking as input the set XSet. The secret key K_0 of the subset membership check scheme is kept by the client. In PHXT, each bit of BF corresponding to XSet is encrypted into one-bit ciphertext rather than a block of at least 128 bits as in HXT.

The protocol Search is shown in Algorithm 2, where first 16 lines generate stag and xtoken similar to OXT and HXT. The main difference between PHXT and HXT is the approach to check the XSet subset membership for conjunctions. In our PHXT scheme, the XSet subset membership check for conjunctions is processed by the client, while it is processed by

Algorithm 2 PHXT

```
EDBSetup
Input: $1^\lambda$, DB
Output: $mk$, EDB
 1: Select key $K_S$ for PRF $F_S$
 2: Select keys $K_X, K_I, K_Z$ for PRF $F_p$
 3: Initialize $\mathbf{T} \leftarrow \emptyset$ indexed by W
 4: Initialize XSet $\leftarrow \emptyset$
 5: Initialize BF $\leftarrow 0^m$
 6: for each $w \in$ W do
 7:     Initialize $\mathbf{t} \leftarrow \emptyset$
 8:     $K_e \leftarrow F_S(K_S, w)$
 9:     for $id \in$ DB[$w$] do
10:         $xid \leftarrow F_I(K_I, id)$
11:         $z \leftarrow F_Z(K_Z, w||c)$
12:         $y \leftarrow xid \cdot z^{-1}$
13:         $e \leftarrow$ Enc($K_e, id$)
14:         Append $(y, e)$ to $\mathbf{t}$
15:         $xtag \leftarrow g^{F_p(K_X,w)\cdot xid}$
16:         XSet $\leftarrow$ XSet $\cup \{xtag\}$
17:     end for
18:     $\mathbf{T}[w] \leftarrow \mathbf{t}$
19: end for
20: (TSet, $K_T$) $\leftarrow$ TSet . Setup($\mathbf{T}$)
21: $(\mathbf{c}, K_0) \leftarrow$ HSMC . Setup(XSet)
22: $mk \leftarrow (K_X, K_I, K_Z, K_S, K_T, K_0)$
23: EDB $\leftarrow$ (TSet, $\mathbf{c}$)
24: return $mk$ and EDB

Protocol: Search
Input: Client's inputs are $mk$ and the query $\phi(\bar{w}) = w_1 \wedge \cdots \wedge w_n$ with sterm $w_1$. Server's input is EDB.
Output: Result $R$
 1: Client computes stag $\leftarrow$ TSet . GetTag($K_T, w_1$).
 2: Client sends stag to server.
 3: Server computes $\mathbf{t} \leftarrow$ TSet . Retrieve(TSet, stag).
 4: Server sends $|\mathbf{t}|$ to client.
 5: for $c = 1 : |\mathbf{t}|$ do
 6:     Client computes $z \leftarrow F_p(K_Z, w_1||c)$.
 7:     for $\ell = 2 : n$ do
 8:         Client computes xtoken[$c, \ell$] $\leftarrow g^{z \cdot F_p(K_X, w_\ell)}$.
 9:     end for
10: end for
11: Client sends xtoken to server.
12: for $c = 1 : |\mathbf{t}|$ do
13:     Server retrieves $(y, e)$ from the $c$-th tuple of $\mathbf{t}$.
14:     for $\ell = 2 : n$ do
15:         Server computes xr[$c, \ell$] $\leftarrow$ xtoken[$c, \ell$]$^y$.
16:     end for
17:     Server computes tk[$c$] $\leftarrow$ HSMC . token($\mathbf{c}$, xr[$c$]).
18: end for
19: Server sends xr and tk to client.
20: Client sets $\mathcal{E}' \leftarrow \emptyset$
21: for $c = 1 : |\mathbf{t}|$ do
22:     Client computes $b_c \leftarrow$ HSMC . Check($K_0$, tk[$c$], xr[$c$]).
23:     if $b_c = 1$ then
24:         Client adds $c$ to $\mathcal{E}'$. (i.e. $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{c\}$)
25:     end if
26: end for
27: Client sends $\mathcal{E}'$ to server.
28: Server sets $\mathcal{E} \leftarrow \emptyset$.
29: for $c = 1, \cdots, |\mathbf{t}|$ do
30:     if $c \in \mathcal{E}'$ then
31:         Server retrieves $(y, e)$ from the $c$-th tuple of $\mathbf{t}$.
32:         Server adds $e$ to $\mathcal{E}$. (i.e. $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$)
33:     end if
34: end for
35: Server sends $\mathcal{E}$ to client.
36: Client sets $R \leftarrow \emptyset$.
37: Client computes $K_e \leftarrow F_S(K_S, w_1)$.
38: for $e \in \mathcal{E}$ do
39:     Client computes $id \leftarrow$ Dec($K_e, e$).
40:     Client adds $id$ to $R$. (i.e. $R \leftarrow R \cup \{id\}$)
41: end for
42: return $R$
```

the server in HXT. More specifically, to check whether all the keywords are contained in the c-th ($c \in [1, |\mathbf{t}|]$) document, the server generates the relation set xr[c] whose elements are $xtag$s and invokes HSMC . token($\mathbf{c}$, xr[c]) to get the token tk[c]. Then xr and tk are delivered to the client which invokes HSMC . Check(K_0, tk[c], xr[c]) to check whether all the elements of xr[c] are contained in XSet or not. If it is the case, the number c is sent to the server, which retrieves e_c to be decrypted to id_c. The communication is summarized in Table 1. The following theorem shows the correctness of PHXT.

Theorem 2: For any efficient adversary $\mathcal{A}$, if both the underlying T-Set instance and HSMC instance are correct, the PRFs F_p and F_S are secure, and the false positive P_e of the underlying bloom filter is negligible in λ, then we have

$$Pr[Cor_{\mathcal{A}}^{\text{PHXT}} = 1] \leq neg(\lambda).$$

In other words, PHXT is a correct SSE scheme.

Proof: The proof of the theorem is detailed in Appendix -B. ■

B. SECURITY OF PHXT

Throughout the paper, we represent a vector of Q $n-$term conjunctive queries by $\mathbf{q} = (\mathbf{s}, \mathbf{x_2}, \cdots, \mathbf{x_n})$ where $\mathbf{q}[i] = (\mathbf{s}[i], \mathbf{x_2}[i], \cdots, \mathbf{x_n}[i])$, $i \in [Q]$, represents the $i-$th query $\mathbf{s}[i] \wedge \mathbf{x_2}[i] \wedge \cdots \wedge \mathbf{x_n}[i]$ with s-term $\mathbf{s}[i]$ and xterms [3] $\mathbf{x_2}[i], \cdots, \mathbf{x_n}[i]$. For a database DB and a vector of queries $\mathbf{q}$,

TABLE 1. Communication between the client and the server during the search for $\phi(\bar{w}) = w_1 \wedge \cdots \wedge w_n$.

	HXT(three rounds)	PHXT(three rounds)
Comm. components	$stag$	$stag$
	$\lvert\mathbf{t}\rvert$	$\lvert\mathbf{t}\rvert$
	$\mathsf{xtoken}[i,j](i \in [1,\lvert\mathbf{t}\rvert], j \in [2,n])$	$\mathsf{xtoken}[i,j](i \in [1,\lvert\mathbf{t}\rvert], j \in [2,n])$
	$\mathbf{v}[i,j](i \in [1,\lvert\mathbf{t}\rvert], j \in [2,n])$	$\mathsf{xr}[i,j](i \in [1,\lvert\mathbf{t}\rvert], j \in [2,n])$; $\mathsf{tk}[i](i \in [1,\lvert\mathbf{t}\rvert])$
	$\mathsf{token}[i](i \in [1,\lvert\mathbf{t}\rvert])$	$\mathcal{E}' = \{c \mid e_c \in \mathsf{DB}[\phi(\bar{w})]\}$
	$\mathcal{E} = \{e \mid e \in \mathsf{DB}[\phi(\bar{w})]\}$	$\mathcal{E} = \{e \mid e \in \mathsf{DB}[\phi(\bar{w})]\}$
#(Comm.)	820 KB	291 KB

$|stag| = |e| = |\mathsf{tk}[i,j]|$=128 bits, $|\mathsf{xtoken}[i,j]| = |\mathsf{xr}[i,j]| = |\mathsf{token}[i]|$=256 bits, $|\mathbf{v}[i,j]|$=1280 bits, $|c|$=32 bits, using the parameters chosen in section 6. #(Comm).:total communication overhead, assuming that $|\mathbf{t}| = 2000$, $|\mathcal{E}|$=500, $n = 3$.

we define the leakage function $\mathcal{L}_{\mathsf{PHXT}}(\mathsf{DB}, \mathbf{q})$ of PHXT as the tuple (N, $\bar{\mathbf{s}}$, SP, WRP, IP) detailed as follows.

- $N = \Sigma_{i=1}^{d} |W_i|$ is the total number of keyword-identify pairs in DB.
- $\bar{\mathbf{s}} \in [m]^Q$ is the search pattern of $\mathbf{s} \in W^Q$. $\bar{\mathbf{s}}$ reveals the equality pattern of $\mathbf{s}$ because deterministic stag's corresponding to $\mathbf{s}$ are provided to the server. $\bar{\mathbf{s}}[i] = |\{\bar{\mathbf{s}}[1], \cdots, \bar{\mathbf{s}}[j]\}|$, where j is the least integer such that $\bar{\mathbf{s}}[j] = \bar{\mathbf{s}}[i]$, which is the number of unique s-terms have been appeared less than or equal to the $j - th$ query.
- $SP \in [d]^Q$ is the size pattern of the queries, recording the number of documents matched by the s-term of each query, i.e, $SP[i] = |\,\mathsf{DB}[\mathbf{s}[i]]|$.
- $WRP \in [ID]^{|Q|}$ is the Whole Result Pattern of $\mathbf{q}$. Formally, $WRP[i] = \mathsf{DB}[\mathbf{s}[i]] \cap \bigcap_{j=2}^{n} \mathsf{DB}[\mathbf{x}_j[i]]$.
- IP is the conditional intersection pattern, which is a four-dimensional table. Formally,

$$IP[i,j,\alpha,\beta] = \begin{cases} \mathsf{DB}[\mathbf{s}[i]] \cap \mathsf{DB}[\mathbf{s}[j]], & \text{if } i \neq j \text{, and } \mathbf{x}_\alpha[i] = \mathbf{x}_\beta[j] \\ \emptyset, & \text{otherwise} \end{cases}$$

Similar to that of [8], we first show the non-adaptive security of PHXT, where all the queries are chosen non-adaptively. The formal statement is as follows.

Theorem 3: Our scheme PHXT is $\mathcal{L}_{\mathsf{PHXT}}$-semantically secure against non-adaptive attacks where all of the queries are conjunctions and $\mathcal{L}_{\mathsf{PHXT}}$ is the leakage function defined as above, assuming that F_p and F_S are secure PRFs, that SE $=$ (Gen, Enc, Dec) is an IND-CPA secure SE scheme, that the DDH assumption holds in $\mathbb{G}$, that the TSet implementation invoked is (non-adaptively) $\mathcal{L}_T$-secure and computationally correct, and that HSMC is $\mathcal{L}_{\mathsf{HSMC}}$-semantically-secure against adaptive attacks and computationally correct.

Proof: The proof is detailed in Appendix -C. ∎

We next show that the theorem is also established against adaptively attack for any conjunctive queries.

Theorem 4: Our scheme PHXT is $\mathcal{L}_{\mathsf{PHXT}}$-semantically secure against adaptive attacks where all of the queries are conjunctions and $\mathcal{L}_{\mathsf{PHXT}}$ is the leakage function defined as before, assuming that F_p and F_S are secure PRFs, that SE $=$ (Gen, Enc, Dec) is an IND-CPA secure SE scheme, that the DDH assumption holds in $\mathbb{G}$, and that the TSet implementation invoked is adaptive $\mathcal{L}_T$-secure and computationally correct, that HSMC is $\mathcal{L}_{\mathsf{HSMC}}$-semantically-secure against adaptive attacks and computationally correct.

Proof: The proof is detailed in Appendix -D. ∎

VI. PERFORMANCE COMPARISON

We compare our scheme PHXT with HXT [8] in terms of computational costs (of the set-up and search phases), storage size, and communication overheads (including interaction rounds and bandwidth between the client and the server) for a conjunctive query $w_1 \wedge w_2 \wedge \cdots \wedge w_n$ with s-term w_1. Assume that all hash functions of BF have 64-bit outputs. A detailed comparison between our scheme and HXT is summarized in Table 2 whose notations are given in Table 3.

A. SET-UP COMPUTATIONAL COSTS

PHXT and HXT share the same way to compute TSet, whose costs sum up to $N \cdot (T_F + T_E)$ since TSet has N components each filled up through a PRF and a symmetric encryption (SE). Although each scheme does not literally output XSet, it still computes every component of XSet to initiate the corresponding BF vector, which costs $N \cdot T_e$ for PHXT and HXT. To generate $\mathbf{c}$, each scheme costs $Nk \cdot T_h$ for the plain BF vector and $m \cdot T_F$ for the encrypted one, although HXT operates differently with PHXT.

B. SEARCH COMPUTATIONAL COSTS

During the search for $w_1 \wedge \cdots \wedge w_n$, the common computational costs between PHXT and HXT include the costs to generate stag and xtoken's and to decrypt all matched document identifies for the client, which sum up to $T_F + \alpha \cdot (n \cdot T_F + (n-1) \cdot T_e) + |\mathcal{E}| T_D$, and the costs to retrieve $\mathbf{t}$ and to generate xtag's for the server, which sum up to $\alpha \cdot (n-1) \cdot T_e$. Whereas, the holistic check costs of PHXT are different from those of HXT. The computation of the holistic check in HXT includes the generation of vector $\mathbf{v}[c]$ and the operation of Query of HVE by the server, which costs $\alpha \cdot ((n-1)k \cdot T_h) + \alpha \cdot (m' \cdot T_X + T_D)$, and the operation of KeyGen of HVE by the client, which costs $\alpha \cdot (m' \cdot (T_F + T_X) + T_E)$. The holistic check in PHXT includes the operation of token of SMC by the server and the operation of Check of SMC by the client, which costs $\alpha \cdot ((n-1)k \cdot T_h + T_h)$ and $\alpha \cdot ((n-1)k \cdot (T_h + T_F) + T_h)$, respectively.

TABLE 2. Computational costs, storage sizes, and communication overhead between the client and the server for HXT and PHXT.

Conjunctive query q= $(w_1 \wedge w_2 \wedge \cdots \wedge w_n)$			
		HXT	PHXT
Comp.	**setup**	$N(T_E + T_F + kT_h)$ $+mT_F + NT_e$	$N(T_E + T_F + kT_h)$ $+mT_F + NT_e$
	search(server)	$+\alpha((n-1)(kT_h + T_e))$ $+\alpha(m'T_X + T_D)$	$+\alpha((n-1)(kT_h + T_e))$ $+\alpha T_h$
	search(client)	$T_F + \|\mathcal{E}\|T_D + \alpha T_E$ $+\alpha(nT_F + (n-1)T_e)$ $+\alpha(m'(T_X + T_F))$	$T_F + \|\mathcal{E}\|T_D + \alpha T_h$ $+\alpha(nT_F + (n-1)T_e)$ $+\alpha((n-1)k(T_h + T_F))$
Stor.	**storage size**	$N(\ell_E + \ell_p) + m\ell_F$	$N(\ell_E + \ell_p) + m$
Comm.	**rounds**	3	3
	bandwidth	$\ell_F + \|\mathbf{t}\| + \|\mathcal{E}\|\ell_E$ $+\alpha((n-1)(\ell_D + k \cdot 64) + \ell_F + \ell_E)$	$\ell_F + \|\mathbf{t}\| + \|\mathcal{E}\|(\|\mathbf{t}\| + \ell_E)$ $+\alpha(2(n-1)\ell_D + \ell_h)$

Comp.: Computation; Stor.: Storage; Comm.: Communication

TABLE 3. Notations used in Performance Comparison.

Notation	Meaning
ℓ_p	size of an element from Z_p
ℓ_D	size of an element from the DH group
ℓ_F	size of the output of a PRF
ℓ_E	size of the block of SE
ℓ_h	size of the output of the hash function H
m	length of the BF
m'	number of non-wildcard elements in a BF
T_e	time taken to compute an exponentiation
T_F	time taken to compute a PRF
T_h	time taken to compute a hash
T_X	time of an exclusive-or operation over λ
T_E	time taken to encrypt a block in SE
T_D	time taken to decrypt a block in SE
$\|\mathcal{E}\|$	number of documents matching a query
α	number of identifiers matching w_1

C. STORAGE SIZE

Each scheme needs to store EDB, which consists of TSet and **c** derived from the m-bit BF. HXT and PHXT share the same size of TSet, which equals $N \cdot \ell_p + N \cdot \ell_E$ since each of its components contains an element from Z_p and one block of ciphertext of SE. The bloom filter BF in PHXT is encrypted into a ciphertext with length unchanged. Whereas, each bit of BF in HXT is encrypted into a block of ℓ_F bit length, resulting in significant size **c**. Indeed, its storage costs $m \cdot \ell_F$.

D. INTERACTION ROUNDS AND BANDWIDTH

The search protocol can be divided into three phases. In the first phase, both schemes spend their first three interactions for the server in collecting enough information to retrieve $D[w_1]$ and to generate all holistic check tokens tk. During these interactions, both PHXT and HXT need to transmit stag, $|\mathbf{t}|$, and xtoken, which result in $\ell_F + |\mathbf{t}| + \alpha(n-1) \cdot \ell_D$ communication overheads. In the second phase, the fourth interaction is required for PHXT to accomplish holistic subset membership check with communication overheads $\alpha((n-1) \cdot \ell_D + \ell_h)$. Whereas, HXT needs the fourth and the fifth interactions to operate holistic check with communication overheads summing up to $\alpha \cdot ((n-1)k \cdot 64 + \ell_F + \ell_E)$. In the third phase, HXT requires the sixth interaction with bandwidth $|\mathcal{E}| \cdot \ell_E$ to send the query result to the client. PHXT needs the fifth and the sixth interactions with bandwidth $|\mathcal{E}|(|\mathbf{t}| + \ell_E)$ to let the client send the holistic check result to the server and receive the query result from it.

To give intuitive comparisons between PHXT and HXT, we use the percentages of overheads between them to illustrate the improvements of our schemes (see Table 4). For example, let $P_{\mathsf{PHXT}\,/\,\mathsf{HXT}}$ denote the overhead percentage between PHXT and HXT. The overhead ranges from computation to communication and storage.

TABLE 4. Evaluations of the percentages of the overheads between PHXT and HXT.

	Comp. client	Comp. server	Stor. size	Comm. bandwidth
$P_{\mathsf{PHXT}\,/\,\mathsf{HXT}}$	98.85%	98.34%	8.71%	35.71%

To evaluate $P_{\mathsf{PHXT}\,/\,\mathsf{HXT}}$, we set $\ell_p = 192$, $\ell_E = 128$, $\ell_F = 128$, $\ell_h = 128$, and $\ell_D = 256$ for a 512-bit DH-type element. As described in HXT [8], the computation time of an xor operation over 128 bits is 3 times faster than hash, 3900 times than exponentiation, 50 times than Dec, 100 times than Enc, 43 times than PRF. Note that m is approximately $1.44kN$ to attain a negligible probability of false positives. Let $P_e = 10^{-6}$, we get $k = 20$. Assuming that $n = 2$, evaluations of the percentages are summarized in Table 4.

VII. EVALUATIONS

We implement PHXT, OXT, and HXT with C++, using NTL [22] for the implementations of cryptographic primitives needed. The key length of symmetric encryption is set to 128 in all schemes, and the false positive P_e is set to 10^{-6}. The server is deployed on a desktop PC having 4 cores CPU (Intel i3-8350K, 4.00GHz), 32GB of DDR4 RAM, and 500 GB solid-state drive, and the client is in a notebook PC with Intel i7-6700K CPU (4.00GHz×8), 15.6GB of DDR4 RAM,

TABLE 5. Statistics of datasets used in the evaluations.

Size	#(D)	Dist. keywords	Dist. (w,id) pairs
2.11GB	$7.4 \cdot 10^5$	$2.6 \cdot 10^6$	$5.8 \cdot 10^7$
7.61GB	$3.1 \cdot 10^6$	$6.5 \cdot 10^6$	$2.2 \cdot 10^8$
56.1GB	$5.8 \cdot 10^6$	$2.7 \cdot 10^7$	$1.56 \cdot 10^9$

#(D): # of documents; Dist.: distinct

TABLE 6. Storage overhead comparisons among HXT, OXT, PHXT in 2.11GB, 7.61GB and 56.1GB datasets, respectively.

	TSet	c
HXT	2.17GB, 8.17GB, 58.05GB	24.43GB, 91.48GB, 650.18GB
OXT		0.19GB, 0.71GB, 5.08GB
PHXT		

and 983.7GB hard disk drive. They use gRPC[2] to communicate each other in a local area network. We conduct our evaluations on three datasets from Wikimedia Downloads [23], having the original sizes as 2.11GB,[3] 7.61GB,[4] and 56.1GB,[5] respectively, whose statistical features are summarized in Table 5.

In PHXT and HXT, each DH-type *xtag* is replaced with $H(xtag)$ and sent from the server to the client during the holistic subset membership check to reduce the communication costs. Note that these changes obliviously affect the computational costs for the efficiency of hash computations.

A. STORAGE COMPARISON

The TSet of PHXT, HXT, and OXT corresponding to the above three datasets are 2.6GB, 9.8GB, and 69.66GB, respectively. The encrypted vector **c** of OXT and PHXT are 0.19GB, 0.71GB, and 5.08GB, respectively, while **c** of HXT are 24.43GB, 91.48GB, and 650.18GB, respectively. A detailed comparison is listed in Table 6.

B. SEARCH PERFORMANCE COMPARISON

To conduct a fair comparison, we evaluate the performance of all the search protocols on the 2.11GB dataset, which enables **c** of HXT to be RAM-resident in our desktop PC. We choose a variable term **v**, with the selectivity varying from 1 to 298318, and a fixed-term **a** to perform two types of conjunctive queries. The first query uses the **v** as s-term and the **a** as x-term, tested on OXT, PHXT, and HXT, while the other query uses the **a** as s-term and the **v** as x-term.

The time comparisons between the three schemes on the server-side and on the client-side are depicted in Fig. 1 and 2, respectively. The overall query delay comparison is shown in Fig. 3.

As shown by Fig. 1, PHXT and OXT are better than HXT and have almost the same time consumption for the server. Concretely, for the first query $\mathbf{v} \wedge \mathbf{a}$, the server's time of each scheme is linear to the selectivity of **v**. However, the server's time of PHXT is only $5.37\% - 24.1\%$ of that of HXT. It is because that in PHXT the holistic check is done in the client-side rather than in the server. For the query $\mathbf{a} \wedge \mathbf{v}$,

[2]http://www.grpc.io
[3]enwiki-20190220-pages-articles22.xml
[4]enwiki-20190220-pages-articles27.xml
[5]enwiki-20190220-pages-articles.xml

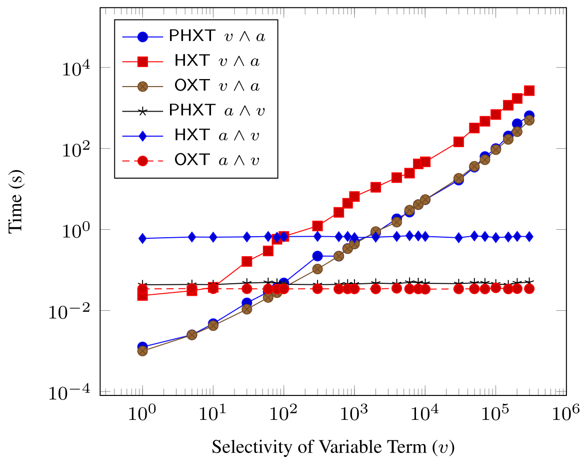


FIGURE 1. Server's time comparison between PHXT, HXT, and OXT in 2.11GB dataset.

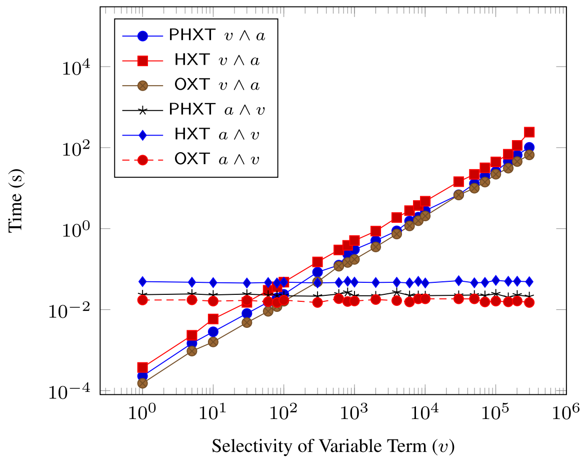


FIGURE 2. Client's time comparison between PHXT, HXT, and OXT in 2.11GB dataset.

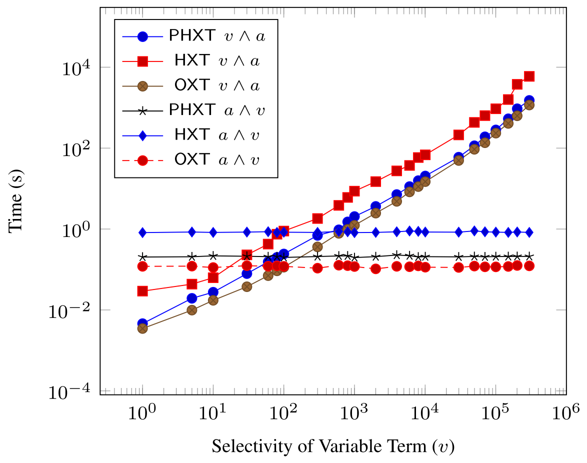


FIGURE 3. Overall query delay comparison between PHXT, HXT, and OXT in 2.11GB dataset.

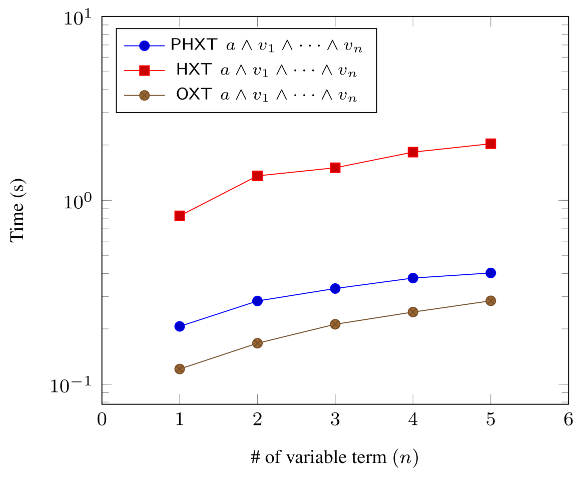


FIGURE 4. **Overall query delay comparison between PHXT, HXT, and OXT in 2.11GB dataset in the multi-keyword setting.**

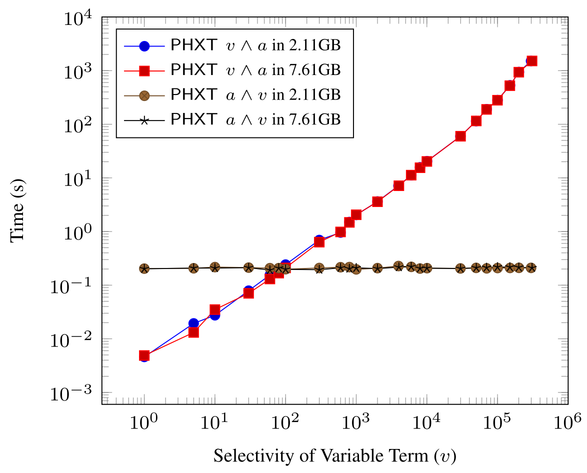


FIGURE 5. **Overall query delay comparison of PHXT for datasets of different sizes.**

the time cost on the server-side is constant regardless of the selectively of $\mathbf{v}$. In this case, the server's time of PHXT is only $6.42\% - 7.47\%$ of that of HXT.

According to Fig. 2, the time of client-side purely reflects the computation costs and does not have any load of I/O. The client's time cost of PHXT is $42.12\% - 66.13\%$ of that of HXT, which is closer to the theoretical analysis.

According to Fig. 3, for the query $\mathbf{v} \wedge \mathbf{a}$, the overall query time of PHXT is $15.74\% - 44.58\%$ of that of HXT, while it is $23.38\% - 25.45\%$ for the query $\mathbf{a} \wedge \mathbf{v}$.

We further give an evaluation to compare the performances of the three schemes for querying multiple keywords, where s-term is identical to the fixed term $\mathbf{a}$ in the previous two-keyword evaluations, while introducing more variable $\{\mathbf{v}_n\}$, $n \in [1, 5]$, as x-terms in conjunctive queries. As shown in Fig. 4, PHXT outperforms HXT although the query delay of all schemes increases as n increases. The underlying reason is that the number of *xtag*s increases when n increases.

From the above performance evaluation discussions, we can conclude that the performance of PHXT is better than that of HXT and close to that of OXT.

C. SCALABILITY OF PHXT

To illustrate the scalability of our scheme, we present the query delay comparison on two datasets of different sizes for PHXT. Figures 5 shows that there are negligible differences between the query delay of PHXT on datasets of different sizes, implying that PHXT is highly scalable.

VIII. CONCLUSION

In this work, we introduce the new primitive subset membership check which enables a modular design of SSE schemes to prevent KPRP leakage. We also give an efficient hash-based SMC implementation. Based on SMC, we propose a practical SSE protocol PHXT that support conjunctive queries without KPRP leakage. Our protocol PHXT maintains the same storage overhead as the OXT while preserving the same security and functionality as the HXT.

To evaluate and compare the performance of our schemes with others, we implemented OXT, HXT, and PHXT using C++ and NTL library. The experimental results demonstrate that our schemes are more efficient than HXT and close to OXT.

To extend our PHXT to the dynamic setting while attaining forward/backward security [4], the underlying building blocks TSet and SMC need to be dynamic with forward/backward security. Recently, many studies have focused on dynamic inverted indexing [6], [24]–[28]. However, extending an SMC implementation to a dynamic one is a challenging attempt.

APPENDIX A
HARDNESS ASSUMPTIONS

Definition 5: (DDH Assumption.) Let $\mathbb{G}$ be a cyclic group generated by g with prime order p. The DDH problem is to distinguish the distribution $\{(g, g^a, g^b, g^{ab})\}$ from the distribution $\{(g, g^a, g^b, g^c)\}$, where a, b, c are chosen uniformly at random from $\mathbb{G}$. For a distinguisher $\mathcal{D}$, the advantage $Adv^{DDH}_{\mathcal{D},\mathbb{G}}(\lambda)$ for $\mathcal{D}$ to solve the DDH problem is defined as $|Pr[\mathcal{D}(g, g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{D}(g, g^a, g^b, g^c) = 1]|$. The DDH Assumption asserts that $Adv^{DDH}_{\mathcal{D},\mathbb{G}}(\lambda)$ is negligible in λ for any probabilistic polynomial-time (PPT) distinguisher $\mathcal{D}$.

Definition 6 (Hash Function): For a security parameter λ, a hash function H is collision resistant if for any PPT adversary $\mathcal{A}$ there is a negligible function neg in λ such that $Pr[(a, b) \leftarrow \mathcal{A}(H) : H(a) = H(b)] \leq neg(\lambda)$.

Definition 7 (PRF): Let $F : \{0, 1\}^\lambda \times \{0, 1\}^n \to \{0, 1\}^m$ be a keyed function. We say that F is a pseudorandom function (PRF) if for any PPT adversary $\mathcal{A}$, its advantage

$$Adv^{prf}_{F,\mathcal{A}}(\lambda) = Pr[\mathcal{A}^{F(K,\cdot)}(1^\lambda) = 1] - Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]$$

is negligible, where the first probability of the right part is taken over the uniform choice of $K \in \{0, 1\}^\lambda$ and the randomness of $\mathcal{A}$, and the second probability is taken over the uniform choice of $f \in Fun(\{0, 1\}^n, \{0, 1\}^m)$ and the randomness of $\mathcal{A}$.

Definition 8 (IND-CPA Security): A symmetric encryption scheme SE =(Gen, Enc, Dec) is a triple of algorithms. The algorithm Gen takes as input a parameter λ and outputs a key $K \in \{0,1\}^\lambda$. The algorithm Enc takes as input a key $K \in \{0,1\}^\lambda$ and a message M and outputs a ciphertext C. The algorithm Dec takes as input a key $K \in \{0,1\}^\lambda$ and a ciphertext C and outputs a message M. We require that the scheme is with computationally correctness for all possible keys K and all messages M.

We say that SE =(Gen, Enc, Dec) is indistinguishable under chosen-plaintext attacks (IND-CPA), if the advantage of any PPT adversary $\mathcal{A}$ in the following experiment is a negligible function $neg(\lambda)$. The experiment first selects a key $k \leftarrow$ Gen and a bit $b \overset{\$}{\leftarrow} \{0,1\}$. Then, the adversary $\mathcal{A}$ outputs two messages m_0 and m_1, from which a challenge ciphertext is generated through $c^* \leftarrow \mathsf{Enc}(k, m_b)$. Finally, the adversary takes as input c^* and outputs a bit b'. The advantage of $\mathcal{A}$ is defined as $Adv^{\text{IND-CPA}}_{\mathsf{SE},\mathcal{A}} = |Pr[b = b'] - \frac{1}{2}|$, where the probability is taken over the randomness used in the experiment and the randomness of $\mathcal{A}$.

APPENDIX B
THEOREM PROOFS
A. PROOF OF THEOREM 1

We first present a simulator $\mathcal{S}_{\mathsf{HSMC}}$ in the ideal experiment, which takes as input the corresponding leakage profiles. The operations of $\mathcal{S}_{\mathsf{HSMC}}$ are divided into two phases, the setup phase and the check phase.

- Setup phase: The adversary $\mathcal{A}$ chooses a set Ω. Taking as input $\mathcal{L}_{\mathsf{HSMC}}(\Omega) = N$, the simulator $\mathcal{S}_{\mathsf{HSMC}}$ randomly chooses EBF $\overset{\$}{\leftarrow} \{0,1\}^m$, which is delivered to $\mathcal{A}$.
- Check phase: The adversary $\mathcal{A}$ chooses the subset $\mathbf{V}_i$ adaptively. Taking as input $\mathcal{L}_{\mathsf{HSMC}}(\Omega, \mathbf{V}) = (CP, \mathbf{b})$ for the i-th query, the simulator $\mathcal{S}_{\mathsf{SMC}}$ returns to $\mathcal{A}$ the bit $b \leftarrow \mathbf{b}[i]$.

The outputs of the PRF F_0 are indistinguishable from random for any PPT algorithm. Hence, the EBF is indistinguishable between $\mathcal{S}_{\mathsf{HSMC}}$ and the real experiment. Furthermore, all transcripts are generated in the same way as in the real experiment. We can finally conclude that

$$|Pr[Real^{\mathsf{HSMC}}_{\mathcal{L}}(\lambda) = 1] - Pr[Ideal^{\mathsf{HSMC}}_{\mathcal{A},\mathcal{S}}(\lambda)] = 1| \le neg(\lambda).$$

B. PROOF OF THEOREM 2

Let $G_0 = Cor^{\mathsf{PHXT}}_{\mathcal{A}}$, and we want to show that $Pr[G_0 = 1] \le neg(\lambda)$. We firstly modify G_0 to get G_1 such that the TSet of G_1 retrieves the correct lists of tuples for all legal queries. According to the correctness of the underlying TSet instance, it implies straightly that there exists an adversary $\mathcal{B}_1$ such that $Pr[G_1 = 1] - Pr[G_0 = 1] \le AdvCor^{\mathsf{TSet}}_{\mathcal{B}_1}(\lambda)$. We then modify G_1 to get G_2 by replaying the PRFs F_X and F_I with random functions f_X and f_I with the same domains and ranges, respectively. The security of PRF implies that there exists an adversary $\mathcal{B}_2$ such that $Pr[G_2 = 1] - Pr[G_1 = 1] \le 2Adv^{prf}_{F_p,\mathcal{B}_2}(\lambda)$. Let $\phi(\bar{w}) = w_1 \wedge \cdots \wedge w_n$ with s-term w_1 be the query. The game G_2 outputs 1 only if the simulated search protocol causes the output of the client to be different from the result $\mathsf{DB}[\phi(\bar{w})]$. According to the correctness of TSet, the simulated server will retrieve the correct list $T[w_1]$. Combining with xtoken received from the client, the server is able to compute $\mathsf{xr}[c]$ for each $c \in [1, |\mathsf{DB}[w_1]|]$, where $\mathsf{xr}[c][i] = xtag(w_i, id_c) = g^{f_X(w_i)\cdot f_I(id_c)}$. For each $id_c \in \mathsf{DB}[\phi(\bar{w})]$, it will be returned as a part of the result of the client since $b_c = 1$, where b_c is derived from HSMC . Check. Hence, the event $G_2 = 1$ will happen only due to false positives, which implies there exists an index c such that $id_c \in \mathsf{DB}[w_1]$ and $id_c \notin \mathsf{DB}[\phi(\bar{w})]$ (i.e. $id_c \notin \mathsf{DB}[w_i]$ for some $2 \le i \le n$), but $b_c = 1$. There are two subcases. The first subcase is that $g^{f_X(w_i)\cdot f_I(ind_c)} = g^{f_X(w')\cdot f_I(ind')}$ for some other $(w', id') \ne (w_i, id_c)$ and $id' \in \mathsf{DB}[w']$. Assume that $f_I(id) \ne 0$ and $f_X(w) \ne 0$ for every id and every w. It is the case with all but probability $\frac{(d+m)}{p}$. Then for certain pairs (w_i, id_c) and (w', id'), the probability that the equation described above holds is $\frac{1}{p-1}$, which implies that the first subcase happens with probability less than $\frac{|\mathsf{DB}[w_1]|(n-1)N}{p-1}$ over all pairs (w_i, id_c) in the query and over all pairs (w', id') in XSet. The second subcase is that there are some mistakes made in the scheme SMC, which happens with probability $\le |\mathsf{DB}[w_1]|neg(\lambda)$ over all $c \in [|\mathsf{DB}[w_1]|]$. Hence, $Pr[G_2 = 1] \le \frac{(d+m)}{p} + \frac{|\mathsf{DB}[w_1]|(n-1)N}{p-1} + |\mathsf{DB}[w_1]|neg(\lambda)$.

We conclude that $Pr[Cor^{\mathsf{PHXT}}_{\mathcal{A}} = 1] \le neg(\lambda)$ through a simple hybrid argument for games G_0, G_1, G_2.

C. PROOF OF THEOREM 3

The following process of proof is similar to that of OXT [3] and HXT [8] except for the use of an HSMC simulator.

We establish our proof through a sequence of games G_0, $G_1, G_2, \cdots, G_8$ which take as input DB, $\mathbf{q} = (\mathbf{s}, \mathbf{x_2}, \cdots, \mathbf{x_n})$, where the output of G_0 has the same distribution as that of $\mathbf{Real}^{\mathsf{PHXT}}_{\mathcal{A}}(\lambda)$ and the output of G_8 has the same distribution as that of the simulator. Then, we proceed to show that $|Pr[G_i = 1] - Pr[G_j = 1]| \le neg(\lambda)$, for $0 \le i < j \le 8$, such that we can obtain an inequality $|Pr[\mathbf{Real}^{\mathsf{PHXT}}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Ideal}^{\mathsf{PHXT}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \le neg(\lambda)$ by the standard hybrid argument, which completes our proof. The process is similar to that of OXT and HXT except for the use of HSMC simulator.

Game G_0: As described in Algorithm 3, G_0 is an implementation of the real game $Real^{\mathsf{PHXT}}_{\mathcal{A}}(\lambda)$ but with some bookkeeping changes which would be useful in the following games. G_0 firstly generates the array $\mathbf{T}$ almost the same as that of $\mathbf{Real}^{\mathsf{PHXT}}_{\mathcal{A}}(\lambda)$ except that it records the permutation σ in the array *Wperms* indexed by keywords, which will make the following analysis clearer. It invokes TSet.Setup with input $\mathbf{T}$. Then it generates the encrypted vector $\mathbf{c}$ by invoking HSMC . Setup. The output of G_0 is EDB and a transcript $\mathbf{tr}$, where for each $i \in [Q]$, $\mathbf{tr}[i]$ comprises $(ResInds[i], (\mathsf{STags}[i], \mathsf{xtoken}[i]))$, where $\mathsf{STags}[i]$ contains the stag used for TSet.Retrieve in the i-th query,

Algorithm 3 Game G_0 of PHXT (DB, **q**)

```
$(id_i, W_i)_{i=1}^d \leftarrow$ DB
$K_X, K_I, K_Z, K_S \xleftarrow{\$} \{0,1\}^\lambda$
for $w \in W$ do
    $(\bar{id}_1, \cdots, \bar{id}_{T_w}) \leftarrow$ DB$[w]$
    $\sigma \xleftarrow{\$} Perm([T_w])$, $WPerms[w] \leftarrow \sigma$
    $K_e \leftarrow F_S(K_S, w)$; $\mathbf{t} \leftarrow \emptyset$
    for $c = 1 : T_w$ do
        $xid \leftarrow F_p(K_I, \bar{id}_{\sigma(c)})$
        $z \leftarrow F_p(K_Z, w||c)$
        $y \leftarrow xid \cdot z^{-1}$
        $e \leftarrow \mathsf{Enc}(K_e, \bar{id}_{\sigma(c)})$
        $\mathbf{t}[c] \leftarrow (y, e)$
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
(TSet, $K_T$) $\leftarrow$ TSet.Setup($\mathbf{T}$)
for $i = 1, \cdots, Q$ do
    STags$[i] \leftarrow$ TSet.GetTag($K_T$, $\mathbf{s}[i]$)
end for
Initialize XSet $\leftarrow \emptyset$
for each $w \in W$ and $id \in$ DB$[w]$ do
    $xw \leftarrow F_p(K_X, w)$
    $xid \leftarrow F_p(K_I, id)$
    $xtag \leftarrow g^{xw \cdot xid}$
    XSet $\leftarrow$ XSet $\cup \{xtag\}$
end for
$(\mathbf{c}, K_0) \leftarrow$ HSMC.Setup(XSet)
EDB $\leftarrow$ (TSet, $\mathbf{c}$)
for $i = 1, \cdots, Q$ do
    for $c = 1 : |$DB$[\mathbf{s}[i]]|$ do
        $z_c \leftarrow F_p(K_Z, \mathbf{s}[i]||c)$
        for $\ell = 2 : n$ do
            $xw_\ell \leftarrow F_p(K_X, \mathbf{x}_\ell[i])$
            xtoken$[i, c, \ell] \leftarrow g^{z_c \cdot xw_\ell}$
        end for
    end for
    $ResInds[i] \leftarrow$ DB$[\mathbf{s}[i]] \cap \bigcap_{\ell=2}^n$ DB$[\mathbf{x}_\ell[i]]$
    $\mathbf{tr}[i] \leftarrow (ResInds[i]$, (STags$[i]$, xtoken$[i]$))
end for
return (EDB, $\mathbf{tr}$)
```

xtoken$[i]$ comprises the sets used for XSet membership test, and $RedInds[i]$ records the id values matching the i-th query, i.e. $ResInds[i] \leftarrow$ DB$[\mathbf{s}[i]] \cap \bigcap_{\ell=2}^n$ DB$[\mathbf{x}_\ell[i]]$, which is indistinguishable from the output of the search protocol for the correctness of PHXT. All of them are the same as the view of the server in the practical setting and are sent to $\mathcal{A}$. Assuming that no errors have occurred, the distribution of the output of G_0 is the same as that of $\mathbf{Real}_{\mathcal{A}}^{\mathsf{PHXT}}(\lambda)$. With the correctness of PHXT, we have:

$$|Pr[G_0 = 1] - Pr[Real_{\mathcal{A}}^{\mathsf{PHXT}}(\lambda) = 1]| \leq neg(\lambda)$$

Algorithm 4 Games G_1, $\boxed{G_2}$, $\boxed{\boxed{G_3}}$ of PHXT (DB, **q**)

```
$(id_i, W_i)_{i=1}^d \leftarrow$ DB
$f_I, f_X, f_Z \xleftarrow{\$} Func(\{0,1\}^\lambda, Z_p^*)$
for $w \in W$ do
    $(\bar{id}_1, \cdots, \bar{id}_{T_w}) \leftarrow$ DB$[w]$
    $\sigma \xleftarrow{\$} Perm([T_w])$, $WPerms[w] \leftarrow \sigma$
    $K_e \xleftarrow{\$} \{0,1\}^\lambda$; $\mathbf{t} \leftarrow \emptyset$
    for $c = 1 : T_w$ do
        $xid \leftarrow f_I(\bar{id}_{\sigma(c)})$
        $z \leftarrow f_Z(w||c)$
        $y \leftarrow xid \cdot z^{-1}$
        $e \leftarrow \mathsf{Enc}(K_e, \bar{id}_{\sigma(c)})$
        $\boxed{e \leftarrow \mathsf{Enc}(K_e, 0^\lambda)}$
        $\mathbf{t}[c] \leftarrow (y, e)$
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
(TSet, $K_T$) $\leftarrow$ TSet.Setup($\mathbf{T}$)
for $i = 1, \cdots, Q$ do
    STags$[i] \leftarrow$ TSet.GetTag($K_T$, $\mathbf{s}[i]$)
end for
Initialize XSet $\leftarrow \emptyset$
for each $w \in W$ and $id \in$ DB$[w]$ do
    $xw \leftarrow f_X(w)$
    $xid \leftarrow f_I(id)$
    $xtag \leftarrow g^{xw \cdot xid}$
    XSet $\leftarrow$ XSet $\cup \{xtag\}$
end for
$(\mathbf{c}, K_0) \leftarrow$ HSMC.Setup(XSet)
$\boxed{\boxed{\mathbf{c} \leftarrow \mathcal{S}_{\mathsf{HSMC}}(N)}}$
EDB $\leftarrow$ (TSet, $\mathbf{c}$)
for $i = 1, \cdots, Q$ do
    for $c = 1 : |$DB$[\mathbf{s}[i]]|$ do
        $z_c \leftarrow f_Z(\mathbf{s}[i]||c)$
        for $\ell = 2 : n$ do
            $xw_\ell \leftarrow f_X(\mathbf{x}_\ell[i])$
            xtoken$[i, c, \ell] \leftarrow g^{z_c \cdot xw_\ell}$
        end for
    end for
    $ResInds[i] \leftarrow$ DB$[\mathbf{s}[i]] \cap \bigcap_{\ell=2}^n$ DB$[\mathbf{x}_\ell[i]]$
    $\mathbf{tr}[i] \leftarrow (ResInds[i]$, (STags$[i]$, xtoken$[i]$))
end for
return (EDB, $\mathbf{tr}$)
```

Game G_1: As described in Algorithm 4, G_1 is almost the same as G_0 except that we replace the PRFs F_p, F_S with random functions. More specifically, we replace $F_p(K_X, \cdot)$, $F_p(K_I, \cdot)$, $F_p(K_Z, \cdot)$ with random functions f_X, f_I, f_Z, respectively. As for $F_S(K_S, \cdot)$, it will not evaluate twice at the same input, so for the sake of conciseness, we will not record its evaluation value at any point. That is to say, we just need to replace the evaluation $F_S(K_S, \cdot)$ at every point with random number.

As stated in Theorem 3, we assume that F_p, F_S are secure PRFs, so there exists efficient adversaries $\mathcal{B}_{1,1}, \mathcal{B}_{1,2}$ such that

$$|Pr[G_1 = 1] - Pr[G_1 = 1]| \leq 3Adv^{prf}_{F_p,\mathcal{B}_{1,1}}(\lambda) + Adv^{prf}_{F,\mathcal{B}_{1,2}}(\lambda).$$

The details of the reductions are omitted here.

Game G_2: This game described in Algorithm 4 is almost the same as G_1 except that it replaces the row above the boxed code with the boxed code. That is to say, for $w \in W$, each e is an encryption of 0^λ instead of the corresponding id with the same key.

Since SE is an IND-CPA secure symmetric scheme, and the encryption of 0^λ is executed for polynomial, say $poly(\lambda)$, times, there exists an efficient adversaries B_2 such that

$$|Pr[G_2 = 1] - Pr[G_1 = 1]| \leq poly(\lambda) \cdot Adv^{\text{IND-CPA}}_{\text{SE},\mathcal{B}_2}(\lambda).$$

Game G_3: AS described in Algorithm 4, G_3 changes the evaluations of the encrypted vector $\mathbf{c}$. $\mathbf{c}$ is generated by invoking an HSMC simulator $\mathcal{S}_{\text{HSMC}}$ rather than HSMC . Setup. Such an efficient simulator $\mathcal{S}_{\text{HSMC}}$ exists for the security of HSMC scheme. Hence, there exists an efficient adversary $\mathcal{B}_3$ perfectly simulating G_2 with HSMC real game, and perfectly simulating G_3 with HSMC ideal game. For the security notion of HSMC, we conclude that

$$|Pr[G_3 = 1] - Pr[G_2 = 1]| \leq Adv^{\text{HSMC}}_{\mathcal{B}_3}(\lambda).$$

Game G_4: As described in Algorithm 5, each elements of xtoken of G_4 are computed in an alternative but equivalent way. Roughly speaking, all values $g^{f_X(w) f_I(id)}$ for each pair $(w, id) \in W \times ID$ are precomputed and recorded in an array A, and all elements of xtoken's are produced via the corresponding values from A.

In the $i-th$ query, retrieve $\mathbf{t}$ corresponding to the s-term $\mathbf{s}[i]$, and look up $\text{DB}[\mathbf{s}[i]] = (\bar{id}_1, \cdots, \bar{id}_{T_S})$ and $WPerms[\mathbf{s}[i]] = \sigma$, where $\mathbf{t} = \{(f_I(\bar{id}_{\sigma(c)})/f_Z(\mathbf{s}[i]||c), e_c)_{c \in [T_S]}\}$. Then for $c \in [T_S]$ and $\ell \in [2, n]$, it computes xtoken$[c, \ell]$ as $A[\mathbf{x}_\ell[i], \bar{id}_{\sigma(c)}]^{1/y_c}$ rather than $g^{f_Z(\mathbf{s}[i]||c) \cdot f_X(\mathbf{x}_\ell[i])}$. But $A[\mathbf{x}_\ell[i], \bar{id}_{\sigma(c)}]^{1/y_c} = (g^{f_X(\mathbf{x}_\ell[i]) f_I(\bar{id}_{\sigma(c)})})^{f_Z(\mathbf{s}[i]||c)/f_I(\bar{id}_{\sigma(c)})} = g^{f_X(\mathbf{x}_\ell[i]) \cdot f_Z(\mathbf{s}[i]||c)}$. Hence, xtoken in G_4 is computed in the same way as that in G_3 and we conclude that

$$Pr[G_4 = 1] = Pr[G_3 = 1].$$

Game G_5: As described in Algorithm 5, G_5 is almost the same as G_4 except that all y's are drawn uniformly and independently from $\{0, 1\}^\lambda$. Thanks to the modification made in G_4, no repeated computation is required for f_Z of any input. Hence, z distributes uniformly and independently, which implies that $y \leftarrow xid \cdot z^{-1}$ also distributes uniformly and independently. So replaying y with random values does not change anything about the distribution of the output of G_5 from that of G_4 and we have $Pr[G_5 = 1] = Pr[G_4 = 1]$.

Game G_6: As described in Algorithm 5, G_6 is almost the same as G_5 except that we set every $A[w, id]$ value to be an

Algorithm 5 Games G_4, $\boxed{G_5}$, $\boxed{\boxed{G_6}}$, $\boxed{\boxed{\boxed{G_7}}}$ of PHXT (DB, $\mathbf{q}$)

```
$(id_i, W_i)_{i=1}^d \leftarrow \text{DB}$
$f_I, f_X, f_Z \xleftarrow{\$} Func(\{0,1\}^\lambda, Z_p^*)$
for $w \in W$ do
    for $id \in ID$ do
        $xw \leftarrow f_X(w)$; $X[w] \leftarrow g^{xw}$; $xid \leftarrow f_I(id)$
        $A[w, id] \leftarrow X[w]^{xid}$
        $\boxed{\boxed{A[w, id] \xleftarrow{\$} \mathbb{G}}}$
    end for
end for
for $w \in W$ do
    $(\bar{id}_1, \cdots, \bar{id}_{T_w}) \leftarrow \text{DB}[w]$
    $\sigma \xleftarrow{\$} Perm([T_w])$, $WPerms[w] \leftarrow \sigma$
    $K_e \xleftarrow{\$} \{0,1\}^\lambda$; $\mathbf{t} \leftarrow \emptyset$
    for $c = 1 : T_w$ do
        $xid \leftarrow f_I(\bar{id}_{\sigma(c)})$
        $z \leftarrow f_Z(w||c)$
        $y \leftarrow xid \cdot z^{-1}$
        $\boxed{y \xleftarrow{\$} Z_p^*}$
        $e \leftarrow \text{Enc}(K_e, 0^\lambda)$
        $\mathbf{t}[c] \leftarrow (y, e)$
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
$(\text{TSet}, K_T) \leftarrow \text{TSet.Setup}(\mathbf{T})$
for $i = 1, \cdots, Q$ do
    $\text{STags}[i] \leftarrow \text{TSet.GetTag}(K_T, \mathbf{s}[i])$
end for
$\boxed{\boxed{\boxed{(\text{TSet}, \text{STags}) \leftarrow \mathcal{S}_T(\mathcal{L}(\text{DB}, \mathbf{s}), T[\mathbf{s}])}}}$
$\mathbf{c} \leftarrow \mathcal{S}_{\text{HSMC}}(N)$
$\text{EDB} \leftarrow (\text{TSet}, \mathbf{c})$
for $i = 1, \cdots, Q$ do
    $\mathbf{t} \leftarrow \text{TSet.Retrieve}(\text{TSet}, \text{STags}[i])$
    $(\bar{id}_1, \cdots, \bar{id}_{T_s}) \leftarrow \text{DB}[\mathbf{s}[i]]$; $\sigma \leftarrow WPerms[\mathbf{s}[i]]$
    for $c = 1 : |\text{DB}[\mathbf{s}[i]]|$ do
        Retrieves $(y_c, e_c)$ from the $c-th$ tuple in $\mathbf{t}$
        for $\ell = 2 : n$ do
            $\text{xtoken}[i, c, \ell] \leftarrow A[\mathbf{x}_\ell[i], \bar{id}_{\sigma(c)}]^{1/y_c}$
        end for
    end for
    $ResInds[i] \leftarrow \text{DB}[\mathbf{s}[i]] \cap \bigcap_{\ell=2}^n \text{DB}[\mathbf{x}_\ell[i]]$
    $\mathbf{tr}[i] \leftarrow (ResInds[i], (\text{STags}[i], \text{xtoken}[i]))$
end for
return $(\text{EDB}, \mathbf{tr})$
```

element chosen uniformly and independently from $\mathbb{G}$. Under the DDH assumption, there exists a PPT adversary $\mathcal{B}_6$ simply simulating the values of array X with g^a values and simulating the values of A with g^c values, the process of which is similar to that as in [3]. Thus, $\mathcal{B}_6$ simulates G_5 while $c = ab$ and simulates G_6 while $c \overset{\$}{\leftarrow} Z_p^*$. And we conclude that

$$|Pr[G_6 = 1] - Pr[G_5 = 1]| \leq Adv^{DDH}_{\mathbb{G},\mathcal{B}_6}(\lambda).$$

Game G_7: As described in Algorithm 5, G_7 generates TSet and STags by using a TSet simulator $\mathcal{S}_T$ rather than by invoking TSet.setup and TSet.GetTag. Such an efficient simulator $\mathcal{S}_T$ can be guaranteed because TSet is (non-adaptively) $\mathcal{L}_T$ secure. And there exists an efficient adversary $\mathcal{B}_7$ can perfectly simulates G_6 with TSet real game, and can perfectly simulates G_7 with TSet ideal game. For the security notion of TSet and the discussion above, we have

$$|Pr[G_7 = 1] - Pr[G_6 = 1]| \leq Adv^{\mathsf{TSet}}_{\mathcal{B}_7}(\lambda).$$

Game G_8: As described in Algorithm 6, G_8 modifies the way of accessing the array A to enable the simulator to work with its given leakage profile. Roughly speaking, the game only accesses array A at an index (w, id) if the game accesses this index in A repeatedly. Note that only the generation of xtoken will access the array A. And a value in array A indexed by $[w, id]$ will be repeatedly accessed only if there exists $i, j \in [Q]$ and $\ell, v \in [2, n]$ such that $j \neq i$ and $id \in (\mathsf{DB}[\mathbf{s}[i]] \cap \mathsf{DB}[\mathbf{s}[j]]) \wedge \mathbf{x}_\ell[i] = \mathbf{x}_v[j]$. This condition is exactly what the last 'If' statement in G_8 tests for. And if this condition statement is not satisfied, the xtoken value is randomly and independently selected from $\mathbb{G}$, which does not change anything about the distribution of the output of G_8 from that of G_7. So we have

$$Pr[G_8 = 1] = Pr[G_7 = 1].$$

Simulator: Next, we present a simulator $\mathcal{S}_{PHXT}$, described in Algorithm 7, which takes as input the leakage profile $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ described in section V-B, which consists of $(N, \bar{\mathbf{s}}, SP, WRP, IP)$. By showing that the distribution of the simulator $\mathcal{S}_{\mathsf{PHXT}}$ is the same as that of G_8 and combining the relations between the games showed above, we will conclude that $\mathcal{S}_{\mathsf{PHXT}}$ satisfies all the requirements detailed in definition 2 in the non-adaptive version.

We first define a restricted equality pattern of $\mathbf{x}$, which is denoted as $\hat{\mathbf{x}}$ and is not exactly the equality pattern of $\mathbf{x}$ but the 'known' equality pattern derived from the leakage IP, which is similar to that of [3]. More specifically, we firstly define that $\hat{\mathbf{x}}[i, \alpha] = \hat{\mathbf{x}}[j, \beta]$ iff $IP[i, j, \alpha, \beta] \neq \emptyset$. Then, we take transitive closure on $\hat{\mathbf{x}}$ to get an equivalence relation. In particular, for all $i, j \in [Q]$, we have that

$$\hat{\mathbf{x}}[i, \alpha] = \hat{\mathbf{x}}[j, \beta] \Rightarrow \mathbf{x}_\alpha[i] = \mathbf{x}_\beta[j], \tag{1}$$

and

$$(\mathbf{x}_\alpha[i] = \mathbf{x}_\beta[j]) \wedge (\mathsf{DB}[\bar{\mathbf{s}}[i]] \cap \mathsf{DB}[\bar{\mathbf{s}}[j]] \neq \emptyset) \Rightarrow \hat{\mathbf{x}}[i, \alpha] = \hat{\mathbf{x}}[j, \beta]. \tag{2}$$

With the leakage profile $\mathcal{L}_{\mathsf{PHXT}}(\mathsf{DB}, \mathbf{q})$ and $\hat{\mathbf{x}}$ derived from the leakage IP, the work of simulator $\mathcal{S}_{\mathsf{PHXT}}$ is detailed

Algorithm 6 Games G_8 of PHXT (DB, $\mathbf{q}$)

```
$(id_i, W_i)_{i=1}^d \leftarrow$ DB
for $w \in W$ and $id \in ID$ do
    $A[w, id] \overset{\$}{\leftarrow} \mathbb{G}$
end for
for $w \in W$ do
    $(\bar{id}_1, \cdots, \bar{id}_{T_w}) \leftarrow \mathsf{DB}[w]$
    $\sigma \overset{\$}{\leftarrow} Perm([T_w])$, $WPerms[w] \leftarrow \sigma$
    $K_e \overset{\$}{\leftarrow} \{0, 1\}^\lambda$; $\mathbf{t} \leftarrow \emptyset$
    for $c = 1 : T_w$ do
        $y \overset{\$}{\leftarrow} Z_p^*$
        $e \leftarrow \mathsf{Enc}(K_e, 0^\lambda)$
        $\mathbf{t}[c] \leftarrow (y, e)$
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
$(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}_T(\mathcal{L}(\mathbf{T}, \mathbf{s}), \mathbf{T}[\mathbf{s}])$
$\mathbf{c} \leftarrow \mathcal{S}_{\mathsf{HSMC}}(N)$
$\mathsf{EDB} \leftarrow (\mathsf{TSet}, \mathbf{c})$
for $i = 1, \cdots, Q$ do
    $\mathbf{t} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{STags}[i])$
    $(\bar{id}_1, \cdots, \bar{id}_{T_s}) \leftarrow \mathsf{DB}[\mathbf{s}[i]]$; $\sigma \leftarrow WPerms[\mathbf{s}[i]]$
    for $c = 1 : |\mathsf{DB}[\mathbf{s}[i]]|$ do
        Retrieves $(y_c, e_c)$ from the $c - th$ tuple in $\mathbf{t}$
        for $\ell = 2 : n$ do
            if $\exists j \neq i$ and $v \in [2, n]$: $\bar{id}_{\sigma(c)} \in \mathsf{DB}[\mathbf{s}[j]] \wedge \mathbf{x}_\ell[i] = \mathbf{x}_v[j]$ then
                $\mathsf{xtoken}[i, c, \ell] \leftarrow A[\mathbf{x}_\ell[i], \bar{id}_{\sigma(c)}]^{1/y_c}$
            else
                $\mathsf{xtoken}[i, c, \ell] \overset{\$}{\leftarrow} \mathbb{G}$
            end if
        end for
    end for
    $ResInds[i] \leftarrow \mathsf{DB}[\mathbf{s}[i]] \cap \bigcap_{\ell=2}^n \mathsf{DB}[\mathbf{x}_\ell[i]]$
    $\mathbf{tr}[i] \leftarrow (ResInds[i], (\mathsf{STags}[i], \mathsf{xtoken}[i]))$
end for
return $(\mathsf{EDB}, \mathbf{tr})$
```

in Algorithm 7. We claim that the distribution of the output of $\mathcal{S}_{\mathsf{PHXT}}$ is the same as that of G_8.

We first observe that TSet and STags are computed in the same way as in G_8, obviously having the same distribution. When construct $\mathbf{c}$, both $\mathcal{S}_{\mathsf{PHXT}}$ and G_8 invoke a simulator $\mathcal{S}_{\mathsf{HSMC}}$. When computing xtoken, $\mathcal{S}_{\mathsf{PHXT}}$ first initializes an array A, which is only filled out for the indexes (w, id) where $w \in \hat{\mathbf{x}}$ and $id \in \bigcup_{i=1}^{Q} \bigcup_{j \neq i, \alpha, \beta} IP[i, j, \alpha, \beta]$, used to keep the reaccessed pattern of array A during the generation of xtoken. The values of $\mathbf{t}$ are exactly identically distributed. For each $w \in \bar{\mathbf{s}}$, the permutation σ is chosen uniformly and independently for w with respect to $SP[w]$, except when the same permutation is reused from $WPerms$. $WPerms$ catches both the repetition pattern and the size pattern of s-terms and has the same pattern of permutations with that of G_8.

Algorithm 7 Simulator $\mathcal{S}_{\mathsf{PHXT}}$ $(N, \bar{\mathbf{s}}, \hat{\mathbf{x}}, SP, WRP, IP)$

```
for $w \in \hat{\mathbf{x}}$ and $id \in \bigcup_{i=1}^{Q} \bigcup_{j\neq i,\alpha,\beta} IP[i,j,\alpha,\beta]$ do
  $A[w, id] \xleftarrow{\$} \mathbb{G}$
end for
for $w \in \bar{\mathbf{s}}$ do
  $i \xleftarrow{\$} \bar{\mathbf{s}}^{-1}[w]$
  $WPerms[w] \xleftarrow{\$} Perms([SP[i]])$
end for
for $w \in \bar{\mathbf{s}}$ do
  $K_e \xleftarrow{\$} \{0,1\}^{\lambda}$, $\mathbf{t} \leftarrow \emptyset$
  $i \xleftarrow{\$} \bar{\mathbf{s}}^{-1}[w]$
  for $c = 1 : SP[i]$ do
    $y \xleftarrow{\$} \{0,1\}^{\lambda}$
    $e \leftarrow \mathsf{Enc}(K_e, 0^{\lambda})$, $\mathbf{t}[c] \leftarrow (y, e)$
  end for
  $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
$\mathbf{T}[\mathbf{s}] \leftarrow (\mathbf{T}[\bar{\mathbf{s}}[1]], \cdots, \mathbf{T}[\bar{\mathbf{s}}[Q]])$
$(\mathsf{TSet}, \mathsf{STags}) \leftarrow \mathcal{S}_T(\mathcal{L}_T(\mathbf{T}, \mathbf{s}), \mathbf{T}[\mathbf{s}])$
$\mathbf{c} \leftarrow \mathcal{S}_{\mathsf{HSMC}}(N)$
$\mathsf{EDB} \leftarrow (\mathsf{TSet}, \mathbf{c})$
for $i = 1, \cdots, Q$ do
  $\mathbf{t} \leftarrow \mathsf{TSet.Retrieve}(\mathsf{TSet}, \mathsf{STags}[i])$
  $\sigma \leftarrow WPerms[\bar{\mathbf{s}}[i]]$
  $R_i \leftarrow \bigcup_{j\neq i,\alpha,\beta} IP[i,j,\alpha,\beta]$
  $T' \leftarrow |R_i|$
  $(\bar{id}_1, \cdots, \bar{id}_{T'}, \underbrace{\perp, \cdots, \perp}_{SP[i]-T'}) \leftarrow \mathsf{DB}[\bar{\mathbf{s}}[i]]$
  for $c = 1, \cdots, SP[i]$ do
    for $\ell = 2 : n$ do
      $(y_c, e_c) \leftarrow \mathbf{t}[c]$
      if $\bar{id}_{\sigma(c)} \neq \perp$ then
        $\mathsf{xtoken}[i, c, \ell] \leftarrow A[\hat{\mathbf{x}}[i, \ell], \bar{id}_{\sigma(c)}]^{1/y_c}$
      else
        $\mathsf{xtoken}[i, c, \ell] \xleftarrow{\$} G$
      end if
    end for
  end for
  $ResInds[i] \leftarrow WRP[i]$
  $\mathbf{tr}[i] \leftarrow (ResInds[i], (\mathsf{STags}[i], \mathsf{xtoken}[i]))$
end for
Return $(\mathsf{EDB}, \mathbf{tr})$
```

The 'revealed' identifiers $R_i \leftarrow \bigcup_{j\neq i,\alpha,\beta} IP[i,j,\alpha,\beta]$ are denoted as $\bar{id}_1, \cdots, \bar{id}_{|R_i|}$, and then R_i are padded up to size $SP[i]$ by dummy symbols $\perp$, which is different from that of G_8, but there is no effect on constructing $\mathsf{xtoken}[i]$ for $\perp$, which are just used to remind $\mathcal{S}$ to set the corresponding positions of xtoken to values chosen randomly and independently rather than from array A. The logic of $\mathcal{S}$ is identical to that of G_8. And we finally need to show that array A in $\mathcal{S}_{\mathsf{PHXT}}$ follows the reaccessed pattern identical to that of G_8.

For any two indexes $(\mathbf{x}_\ell[i_1], id_1)$ and $(\mathbf{x}_v[i_2], id_2)$ accessed from array A in G_8, $\mathcal{S}_{\mathsf{PHXT}}$ will read the indexes $(\hat{\mathbf{x}}[i_1, \ell], id_1)$ and $(\hat{\mathbf{x}}[i_2, v], id_2)$ instead. To show that $\mathcal{S}_{\mathsf{PHXT}}$ is identical to G_8, we claim that

$$(\hat{\mathbf{x}}[i_1, \ell], id_1) = (\hat{\mathbf{x}}[i_2, v], id_2) \Leftrightarrow (\mathbf{x}_\ell[i_1], id_1) = (\mathbf{x}_v[i_2], id_2).$$

If $(\hat{\mathbf{x}}[i_1, \ell], id_1) = (\hat{\mathbf{x}}[i_2, v], id_2)$, that $\hat{\mathbf{x}}[i_1, \ell] = \hat{\mathbf{x}}[i_2, v]$ and $id_1 = id_2$, $\hat{\mathbf{x}}[i_1, \ell] = \hat{\mathbf{x}}[i_2, v]$ implies that $\mathbf{x}_\ell[i_1] = \mathbf{x}_v[i_2]$ from (1), and we can conclude that $(\mathbf{x}_\ell[i_1], id_1) = (\mathbf{x}_v[i_2], id_2)$. If $(\mathbf{x}_\ell[i_1], id_1) = (\mathbf{x}_v[i_2], id_2)$, that $\mathbf{x}_\ell[i_1] = \mathbf{x}_v[i_2]$ and $id_1 = id_2$. As has been known, an index (w, id) is accessed in array A iff there exists an i such that $id \in \bigcup_{j\neq i,\alpha,\beta} IP[i,j,\alpha,\beta]$. So $id_1 = id_2$ are members of the set

$$\bigcup_{j\neq i_1,\alpha,\beta} IP[i_1, j, \alpha, \beta] \cap \bigcup_{j\neq i_2,\alpha,\beta} IP[i_2, j, \alpha, \beta].$$

As $\bigcup_{j\neq i,\alpha,\beta} IP[i,j,\alpha,\beta] \in \mathsf{DB}[\mathbf{s}[i]]$, we can get that $id_1 = id_2$ are members in the set $\mathsf{DB}[\mathbf{s}[i_1]] \cap \mathsf{DB}[\mathbf{s}[i_2]]$, which implies that $\mathsf{DB}[\mathbf{s}[i_1]] \cap \mathsf{DB}[\mathbf{s}[i_2]] \neq \emptyset$, from which we can conclude that $\hat{\mathbf{x}}[i_1, \ell] = \hat{\mathbf{x}}[i_2, v]$ from (2). Now we can conclude that $(\hat{\mathbf{x}}[i_1, \ell], id_1) = (\hat{\mathbf{x}}[i_2, v], id_2)$.

D. PROOF OF THEOREM 4

The main idea of proving Theorem 4 is similar to that of Theorem 3, except that the TSet and all queries are chosen adaptively. To respond to these queries, the simulator initializes an array A adaptively by choosing elements independently from the group with proper repetitions due to the leakage profiles given adaptively.

REFERENCES

[1] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur. CCS*, 2006, pp. 79–88, doi: 10.1145/1180405.1180417.

[2] S.-F. Sun, J. K. Liu, A. Sakzad, R. Steinfeld, and T. H. Yuen, "An efficient non-interactive multi-client searchable encryption with support for Boolean queries," in *Proc. Eur. Symp. Res. Comput. Secur.* Berlin, Germany: Springer, 2016, pp. 154–172.

[3] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Proc. Annu. Cryptol. Conf.* Berlin, Germany: Springer, 2013, pp. 353–373.

[4] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 72–75.

[5] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 496–510, May 2018.

[6] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1038–1055.

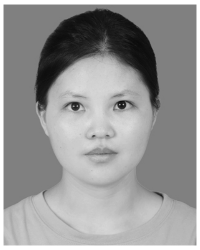
[7] D. Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy. S&P*, May 2000, pp. 44–55.

[8] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 745–762.

[9] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. NDSS*, vol. 20, 2012, p. 12.

[10] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Secur. Symp. USENIX Secur.*, 2016, pp. 707–720.
[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. CCS*, 2015, pp. 668–679.
[12] C. Bösch, P. H. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 18:1–18:51, 2014.
[13] G. S. Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad, "Searchable Symmetric Encryption: Designs and Challenges," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 40:1–40:37, 2017.
[14] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2004, pp. 31–45.
[15] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. Int. Conf. Inf. Commun. Secur.* Berlin, Germany: Springer, 2005, pp. 414–426.
[16] J. W. Byun, D. H. Lee, and J. Lim, "Efficient conjunctive keyword search on encrypted data storage system," in *Proc. Eur. Public Key Infrastruct. Workshop*. Berlin, Germany: Springer, 2006, pp. 184–196.
[17] E.-K. Ryu and T. Takagi, "Efficient conjunctive keyword-searchable encryption," in *Proc. 21st Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINAW)*, May 2007, pp. 409–414.
[18] P. Wang, H. Wang, and J. Pieprzyk, "Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Berlin, Germany: Springer, 2008, pp. 178–195.
[19] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2010, pp. 577–594.
[20] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2014, pp. 23–26.
[21] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, Jan. 2004.
[22] V. Shoup. *NTL: A Library for Doing Number Theory (Version 11.3.4)*. Accessed: 2019. [Online]. Available: https://www.shoup.net/ntl/
[23] W. Foundation. (2019). *Wikimedia Downloads*. [Online]. Available: https://dumps.wikimedia.org/enwiki/
[24] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Depend. Sec. Comput.*, early access, Apr. 2, 2019, doi: 10.1109/TDSC.2018.2822294.
[25] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 763–780.
[26] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1465–1482.
[27] J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu, C. Dong, and W. Lou, "Searchable symmetric encryption with forward search privacy," *IEEE Trans. Depend. Sec. Comput.*, early access, Jan. 22, 2019, doi: 10.1109/TDSC.2019.2894411.
[28] C. Ma and M. Jiang, "Practical lattice-based multisignature schemes for blockchains," *IEEE Access*, vol. 7, pp. 179765–179778, 2019, doi: 10.1109/ACCESS.2019.2958816.

CHANGSHE MA received the Ph.D. degree in computer science from Shanghai Jiao Tong University, in 2002. He is currently a Professor with the School of Computer Science, South China Normal University. His current research interests include lattice-based cryptography, privacy of data outsourcing, and the IoT security.



YIPING GU is currently pursuing the master's degree with the School of Computer Science, South China Normal University.

HONGFEI LI is currently pursuing the master's degree with the School of Computer Science, South China Normal University.