Private Searching on Streaming Data Based on Keyword Frequency

Xun Yi, Elisa Bertino, *Fellow*, *IEEE*, Jaideep Vaidya, and Chaoping Xing

Abstract—Private searching on streaming data is a process to dispatch to a public server a program, which searches streaming sources of data without revealing searching criteria and then sends back a buffer containing the findings. From an Abelian group homomorphic encryption, the searching criteria can be constructed by only simple combinations of keywords, for example, disjunction of keywords. The recent breakthrough in fully homomorphic encryption has allowed us to construct arbitrary searching criteria theoretically. In this paper, we consider a new private query, which searches for documents from streaming data on the basis of keyword frequency, such that the frequency of a keyword is required to be higher or lower than a given threshold. This form of query can help us in finding more relevant documents. Based on the state of the art fully homomorphic encryption techniques, we give disjunctive, conjunctive, and complement constructions for private threshold queries based on keyword frequency. Combining the basic constructions, we further present a generic construction for arbitrary private threshold queries based on keyword frequency. Our protocols are semantically secure as long as the underlying fully homomorphic encryption scheme is semantically secure.

Index Terms—Private searching on streaming data, fully homomorphic encryption, binary linear code

1 Introduction

The problem of private searching on streaming data was first introduced by Ostrovsky and Skeith [17]. It was motivated by one of the tasks of the intelligence community, that is, how to collect potentially useful information from huge volumes of streaming data flowing through a public server. However, that data which is potentially useful and raises a red flag is often classified and satisfies secret search criteria. The challenge is thus how to keep the search criteria classified even if the program residing in the public server falls into adversary's hands. This problem has many applications for the purpose of intelligence gathering. For example, in airports one can use this technique to find if any of hundreds of passenger lists has a name from a possible list of terrorists and, if so, to find his/hers itinerary without revealing the secret terrorists list.

The first solution for private searching on streaming data was given by Ostrovsky and Skeith [17], [18]. It was built on the concept of public-key program obfuscation, where an obfuscator compiles a given program f from a complexity class $\mathcal{C}$ into a pair of algorithms (F, Dec), such that $Dec(F(x)) = f(x)$ for any input x and it is impossible to distinguish for any polynomial time adversary which f from $\mathcal{C}$ was used to produce a given code for F. The basic idea can be briefly described as follows.

Assume that the public dictionary of potential keywords is $D = \{w_1, w_2, \ldots, w_{|D|}\}$. To search for documents containing one or more of classified keywords $K = \{k_1, k_2, \ldots, k_{|K|}\} \subset D$, the client generates a public/private key pair of a public key cryptosystem and constructs a program F, composed of an encrypted dictionary $\mathcal{E}(D)$ from K and a buffer $\mathbb{B}$ which will store matching documents. Then the client dispatches the program F to a public server, where F filters a streaming documents and stores the encryptions of matching documents in the buffer $\mathbb{B}$. After the buffer $\mathbb{B}$ returns, the client decrypts the buffer and retrieves the matching documents. Because both the keywords and the buffer are encrypted, the search criteria are kept classified to the public.

On the basis of this idea, several solutions for private searching on streaming data have been proposed in literature as follows:

1. Ostrovsky and Skeith [17], [18] gave two solutions for private searching on streaming data. One is based on the Paillier cryptosystem [20] and allows to search for documents satisfying a disjunctive condition $k_1 \vee k_2 \vee \cdots \vee k_{|K|}$, i.e., containing one or more classified keywords. Another is based on the Boneh et al. cryptosystem [3] and can search for documents satisfying $(k_{11} \vee k_{12} \vee \cdots \vee k_{1|K_1|}) \wedge (k_{21} \vee k_{22} \vee \cdots \vee k_{2|K_2|})$, an AND of two sets of keywords.
2. Bethencourt et al. [1], [2] also gave a solution to search for documents satisfying a condition $k_1 \vee k_2 \vee \cdots \vee k_{|K|}$. Like the idea of [17], an encrypted dictionary is used. However, rather than using one large buffer and attempting to avoid collisions like [17], Bethencourt et al. stored the matching documents in three buffers and retrieved them by solving linear systems.

- *X. Yi is with the College of Engineering and Science, Victoria University, Melbourne, Victoria 8001, Australia. E-mail: xun.yi@vu.edu.au.*
- *E. Bertino is with the Department of Computer Science, Cyber Center and CERIAS, Purdue University, West Lafayette, IN 47907. E-mail: bertino@cs.purdue.edu.*
- *J. Vaidya is with the Management Science and Information Systems Department, Rutgers University, 1 Washington Park, Newark, NJ 07102. E-mail: jsvaidya@business.rutgers.edu.*
- *C. Xing is with the School of Physical and Mathematical Science, Nanyang Technological University, Singapore 637371. E-mail: xingcp@ntu.edu.sg.*

Manuscript received 16 May 2013; revised 12 Aug. 2013; accepted 20 Aug. 2013; published online 29 Aug. 2013.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2013-05-0117. Digital Object Identifier no. 10.1109/TDSC.2013.36.

3. Yi et al. [25] proposed a solution to search for documents containing more than t out of n keywords, so-called (t, n) threshold searching, without increasing the dictionary size. The solution is built on the state of the art fully homomorphic encryption (FHE) technique and the buffer keeps at most m matching documents without collisions. Searching for documents containing one or more classified keywords like [17], [18], [1], [2] can be achieved by $(1, n)$ threshold searching.

The existing solutions for private searching on streaming data have not considered keyword frequency, the number of times that keyword is used in a document. Search engines like Google, Yahoo, and AltaVista display results based on secret algorithms. Although we do not know the equations, we believe that these are based mainly on keyword frequency and link popularity.

Our contributions. In this paper, we consider a new private query, which searches for documents from streaming data based on keyword frequency, such that a number of times that a keyword appears in a matching document is required to be higher or lower than a given threshold. For example, find documents containing keywords $k_1, k_2, \ldots, k_n$ such that the frequency of the keyword $k_i (i = 1, 2, \ldots, n)$ in the document is higher (or lower) than t_i. We take the lower case into account because terms that appear too frequently are often not very useful as they may not allow one to retrieve a small subset of documents from the streaming data.

This form of query can help us in finding more relevant documents, but it cannot be implemented with traditional homomorphic encryption schemes. Based on FHE, we give disjunctive, conjunctive, and complement constructions for private threshold queries based on keyword frequency: 1) Our disjunctive construction allows to search for documents satisfying a condition such as $(f(k_1) \geq t_1) \vee (f(k_2) \geq t_2) \vee \cdots \vee (f(k_n) \geq t_n)$, where $f(k_i)$ denotes the frequency of the keyword k_i and t_i is a given threshold; 2) Our conjunctive construction allows to search for documents satisfying a condition such as $(f(k_1) \geq t_1) \wedge (f(k_2) \geq t_2) \wedge \cdots \wedge (f(k_n) \geq t_n)$; 3) We have two complement constructions. Our disjunctive complement construction allows us to search for documents satisfying a condition such as $(f(k_{i_1}) \geq t_{i_1}) \vee \cdots \vee (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \vee \neg(f(k_{j_1}) \geq t_{j_1}) \vee \cdots \vee \neg(f(k_{j_{n_2}}) \geq t_{j_{n_2}})$, i.e., $(f(k_{i_1}) \geq t_{i_1}) \vee \cdots \vee (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \vee (f(k_{j_1}) < t_{j_1}) \vee \cdots \vee (f(k_{j_{n_2}}) < t_{j_{n_2}})$, where $\neg$ stands for complement and $n_1 + n_2 = n$. Our conjunctive complement construction allows to search for documents satisfying a condition such as $(f(k_{i_1}) \geq t_{i_1}) \wedge \cdots \wedge (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \wedge \neg(f(k_{j_1}) \geq t_{j_1}) \wedge \cdots \wedge (f(k_{j_{n_2}}) \geq t_{j_{n_2}})$, i.e., $(f(k_{i_1}) \geq t_{i_1}) \wedge \cdots \wedge (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \wedge f(k_{j_1}) < t_{j_1}) \wedge \cdots \wedge (f(k_{j_{n_2}}) < t_{j_{n_2}})$.

Furthermore, by combining the above basic constructions, we present a generic construction for arbitrary threshold query based on keyword frequency.

Like Yi et al.'s solution for the (t, n) threshold query [25], our solutions encrypt the thresholds, compare them with the ciphertexts and store a matching document into the buffer by constructing an encryption of (L, ℓ) linear code of the document. Unlike the (t, n) threshold query solution where only one threshold t is encrypted and enclosed to the searching program, our solutions encrypt the frequency threshold for each keyword because different keywords may have different frequency thresholds.

In summary, our main contribution is a new type of private threshold query based on keyword frequency, which can help us in finding more relevant documents from streaming data.

Organization of the rest of this paper. In the rest of this paper, we will introduce the related work and the background necessary to understand our solutions in Sections 2 and 3, define the formal security model for private query in Section 4, describe our two basic constructions for private queries based on keyword frequency in Sections 5 and 6, our complement construction in Section 7, and our generic construction in Section 8. Security and performance analysis is performed in Sections 9 and 10. Conclusions are outlined in the last section.

2 Related Work

In 2005, Ostrovsky and Skeith [17], [18] gave the first solution for private searching on streaming data as follows.

Assume that the public dictionary of potential keywords is $D = \{w_1, w_2, \ldots, w_{|D|}\}$. To construct a program searching for documents containing one or more of classified keywords $K = \{k_1, k_2, \ldots, k_{|K|}\} \subset D$, the client generates a pair of public and private keys (pk, sk) for a homomorphic encryption scheme $\mathcal{E}$, such as the Paillier cryptosystem [20], and produces an array of ciphertexts $\mathcal{E}(D) = \{c_1, c_2, \ldots, c_{|D|}\}$, one for each keyword $w_i \in D$, such that if $w_i \in K$, then $c_i = \mathcal{E}_{pk}(1)$ and $c_i = \mathcal{E}_{pk}(0)$ otherwise. In addition, the client constructs a buffer $\mathbb{B}$ with γm boxes, each of them is initialized with two ciphertexts $(\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(0))$, where m is the upper bound on the number of matching documents the buffer can accommodate and $m/2^{\gamma}$ should be negligible.

To perform private searching for keywords, Ostrovsky and Skeith segmented the streaming data S into streaming documents $\{M_1, M_2, \ldots\}$, each of which is composed of a number of words, and filtered one at a time. To process a document M_i, the server, which is provided with $D, \mathcal{E}(D), \mathbb{B}$, computes $d_i = \prod_{w_j \in M_i} c_j = \mathcal{E}_{pk}(|M_i \cap K|)$ and $e_i = d_i^{M_i} = \mathcal{E}_{pk}(M_i \cdot \mid M_i \cap K \mid)$, then copies (d_i, e_i) into γ randomly chosen boxes in the buffer $\mathbb{B}$ by multiplying corresponding ciphertexts. If $M_i \cap K = \emptyset$, this step will add an encryption of 0 to each box, having no effect on the corresponding plaintext. If $M_i \cap K \neq \emptyset$, the matching document can be retrieved by computing $M_i = \mathcal{D}_{sk}(e_i)/\mathcal{D}_{sk}(d_i)$ after the buffer returns. If two different matching documents are ever added to the same buffer box, a collision will occur and both copies will be lost. To avoid the loss of matching documents, the buffer size has to be sufficiently large so that each matching document can survive in at least one buffer box with overwhelming probability.

In 2009, Bethencourt et al. [1], [2] proposed a different approach for retrieving matching documents from the buffer. Like the idea of [17], an encrypted dictionary is used, and no-matching documents have no effect on the contents of the buffer. However, rather than using one large buffer and attempting to avoid collisions, Bethencourt et al. stored the matching documents in three buffers—the data buffer $\mathbb{F}$, the count buffer $\mathbb{C}$, and the matching indices buffer $\mathbb{I}$, and retrieved them by solving linear systems.

Bethencourt et al.'s solution is able to process t documents $\{M_1, M_2, \ldots, M_t\}$ of streaming data. For each document M_i, the server computes d_i and e_i as the Ostrovsky-Skeith protocol, and copies d_i and e_i randomly over approximately half of the locations across the buffers $\mathbb{C}$ and $\mathbb{F}$, respectively. A pseudorandom function $g(i, j)$ is used to determine with probability 1/2 whether d_i (or e_i) is copied into a given location j. In addition, the server copies d_i into a fixed number of locations in the buffer $\mathbb{I}$. This is done by using essentially the standard procedure for updating a Bloom filter. Specifically, k hash functions $h_1, h_2, \ldots, h_k$ are used to select the k locations. The locations of $\mathbb{I}$ that d_i is multiplied into are taken to be $h_1(i), h_2(i), \ldots, h_k(i)$.

To retrieve the matching documents, Bethencourt et al. decrypted three buffers $\mathbb{F}, \mathbb{C}, \mathbb{I}$ to $\mathbb{F}', \mathbb{C}', \mathbb{I}'$ at first. For each of the indices $i \in \{1, 2, \ldots, t\}$, $h_1(i), h_2(i), \ldots, h_k(i)$ are computed and the corresponding locations in $\mathbb{I}'$ are checked. If all these locations are nonzero, i is added into the list of potential matching indices, denoted as $\{i_1, i_2, \ldots, i_\ell\}$. The values of $c = \{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_\ell}\}$, where $\alpha_{i_j} = |M_{i_j} \cap K|$, are then determined by solving the system of linear equations $A \cdot c^T = \mathbb{C}'^T$, where $A = (g(j, i))$ is an $|\mathbb{C}| \times i_\ell$ matrix. As the last step, the content of the matching documents $M' = \{M_{i_1}, M_{i_2}, \ldots, M_{i_\ell}\}$ are determined by solving the system of linear equations $A \cdot diag(c) \cdot M'^T = \mathbb{F}'^T$.

The advantage of Bethencourt et al.'s approach [1], [2], compared to Ostrovsky and Skeith solution [17], is that buffer collisions do not matter because matching documents can be retrieved by solving linear systems. Consequently, the buffer size does not need to be sufficiently large in order to maintain a high probability of recovering all matching documents. In fact, the buffer size becomes optimal, i.e., $O(m)$. However, Bethencourt et al.'s approach has a drawback as well. To determine the ordinal numbers of potential matching documents in the decrypted buffer $\mathbb{I}'$, Bethencourt et al. had to check each of the indices $i \in \{1, 2, \ldots, t\}$ of the data stream. Therefore, the buffer recovering has a running-time proportional to the size of the data stream, i.e., $O(m^{2.376} + t \log(t/m))$. This does not fit the model given by Ostrovsky and Skeith in [17] and [18], in which the buffer is decrypted at the cost which is independent of the stream size.

The idea of private searching for documents containing one or more of keywords can be modified to construct more complicated queries. For example, a query composed of at most a λ AND operations can be performed simply by changing the dictionary D to a dictionary D' containing all $|D|^\lambda$ λ-tuples of words in D, which of course comes at a polynomial blow-up of program size.

Using results by Boneh et al. [3], Ostrovsky and Skeith [17], [18] gave a solution for private queries involving an AND of two sets of keywords without increasing the program size. Their basic idea of searching for documents M such that $(M \cap K_1 \neq \emptyset) \wedge (M \cap K_2 \neq \emptyset)$, where K_1, K_2 are two sets of potential keywords, is to construct two arrays of ciphertexts $C_\ell = \{c_1^\ell, c_2^\ell, \ldots, c_{|D|}^\ell\} (\ell = 1, 2)$, where c_i^ℓ is the encryption of 1 if $w_i \in K_\ell$ and 0 otherwise. To process a document M, the program computes $v_\ell = \prod_{w_j \cap M} c_j^\ell = \mathcal{E}_{pk}(|M \cap K_\ell|)$ $(\ell = 1, 2)$ and then $v = e(v_1, v_2)$, where e is a bilinear map. If $(M \cap K_1 \neq \emptyset) \wedge (M \cap K_2 \neq \emptyset)$ is true, v is an encryption of a nonzero element and 0 otherwise. Then, M is encrypted by replacing 1 with v and 0 with an encryption of 0 and the ciphertext is copied into γ randomly chosen boxes in the buffer $\mathbb{B}$.

Ostrovsky and Skeith [19] showed that the general methods used here to create protocols for searching on streaming data (which are based essentially upon manipulating homomorphic encryption) cannot be extended to perform conjunctive queries beyond what has been accomplished as above. More specifically, if one builds a protocol based on an Abelian group homomorphic encryption, then no conjunctive (of more than one term) can be performed without increasing (superlinearly) the dictionary size. It seems that to make progress in significantly extending the query semantics will likely require fundamentally different approaches to the problem, unless major developments are made in the design of homomorphic encryption scheme.

Gentry [9], [10], [11], [12] using lattice-based cryptography constructed the first FHE scheme. In the same year, Dijk et al. [7] presented a second FHE scheme. In 2010, Smart et al. [22] presented a refinement of Gentry's scheme giving smaller key and ciphertext sizes. Recent breakthrough in FHE makes it possible to perform more complicated private queries on streaming data.

In 2012, based on FHE technique, Yi et al. [25] provided a construction of the searching criteria for private (t, n) threshold query on streaming data, which searches for documents containing more than t out of n keywords, without increasing the dictionary size. Like the idea of [17], an encrypted dictionary $\mathcal{E}(D) = \{c_1, c_2, \ldots, c_{|D|}\}$, where correspondences to n keywords are encryptions of 1 and 0 otherwise, is used. Besides it, an encryption of the threshold t $(\leq |D|)$, denoted as $\mathcal{E}_{pk}(t)$, is attached to the program. To process a document M_i, the program computes $d_i = \sum_{w_j \in M_i} c_j = \mathcal{E}_{pk}(|M_i \cap K|)$ and compares $|M_i \cap K|$ with t, using d_i and $\mathcal{E}_{pk}(t)$ on the basis of the fully homomorphic property. It outputs a ciphertext α, which is an encryption of 0 if $|M_i \cap K| \geq t$ and an encryption of 1 otherwise. Then M_i is encrypted by replacing 1 with $\alpha + 1$ and 0 with an encryption of 0. The encryption of a matching document is stored into the buffer by constructing an encryption of (L, ℓ) linear code of the document, where ℓ and L are the plain document size and the plain buffer size, respectively, and then positionwise adding the code into the buffer. To keep up to m matching documents, the buffer size only needs to be $m\ell k$ $(= Lk)$, where k is a security parameter. In addition, the computational decoding cost is $O(m\ell k^2)$ independent of the streaming size. Furthermore, the buffer can keep at most m matching documents. In case there are more than m matching documents in the streaming data, the buffer stores the first m matching documents and throws the rest away. Thus, the buffer collision is no longer an issue.

3 Preliminaries

3.1 Fully Homomorphic Encryption

Previous homomorphic encryption schemes, such as [8], [20], [6], allow homomorphic computation of only one operation (either addition or multiplication) on plaintexts. Recently, FHE schemes, such as [9], [10], [11], [12], [7], [22], which can support evaluation of arbitrary depth circuits, were successfully constructed. Among them, the somewhat fully homomorphic encryption scheme proposed by

Dijk et al. [7] is relatively easy to understand and can be described as follows:

1. KeyGen(k). Takes a security parameter k and determines parameters γ, ρ, η, τ satisfying certain conditions. Chooses a random odd η-bit integer p from $(2\mathbb{Z}+1)\cap(2^{\eta-1},2^{\eta})$ as the secret key sk. Randomly chooses $q_0,q_1,\ldots,q_\tau$ from $[1,2^\gamma/p)$ subject to the condition that the largest q_i is odd and relabel $q_0,q_1,\ldots,q_\tau$ so that q_0 is the largest. Randomly chooses $r_0,r_1,\ldots,r_\tau$ from $\mathbb{Z}\cap(2^{-\rho},2^{\rho})$ and sets $x_0=q_0p+2r_0$ and $x_i=q_ip+2r_i\ mod\ x_0$. The public key is $pk=<x_0,x_1,\ldots,x_\tau>$.
2. Encrypt(pk,m). To encrypt $m\in\{0,1\}$, chooses a random subset $S\subset\{1,2,\ldots,\tau\}$ and a random integer r from $(2^{-\rho},2^{\rho})$ and outputs
$$c=\mathcal{E}(m)=m+2r+\sum_{i\in S}x_i(mod\ x_0).$$
3. Decrypt(sk,c). To decrypts c, outputs
$$(c\ mod\ p)mod\ 2.$$

3.2 Fully Homomorphic Properties

In general, a fully homomorphic encryption scheme has the following properties:

$$\mathcal{E}(m_1)+\mathcal{E}(m_2)=\mathcal{E}(m_1\oplus m_2),$$

$$\mathcal{E}(m_1)\mathcal{E}(m_2)=\mathcal{E}(m_1m_2),$$

for any $m_1,m_2\in\{0,1\}$.

Based on the above two properties, given $\mathcal{E}(m_1)$ and $\mathcal{E}(m_2)$, we can construct

$$\mathcal{E}(m_1\wedge m_2)=\mathcal{E}(m_1)\mathcal{E}(m_2),$$
$$\mathcal{E}(m_1\vee m_2)=\mathcal{E}(m_1)+\mathcal{E}(m_2)+\mathcal{E}(m_1)\mathcal{E}(m_2),$$

for any $m_1,m_2\in\{0,1\}$.

For a positive integer $M=(m_1m_2\ldots m_\ell)_b$ (a binary expression), we write $\mathcal{E}(M)=(\mathcal{E}(m_1),\mathcal{E}(m_2),\ldots,\mathcal{E}(m_\ell))$. Given $\mathcal{E}(M_1)=(\mathcal{E}(x_1),\mathcal{E}(x_2),\ldots,\mathcal{E}(x_\ell))$ and $\mathcal{E}(M_2)=(\mathcal{E}(y_1),\mathcal{E}(y_2),\ldots,\mathcal{E}(y_\ell))$, we can construct $\mathcal{E}(M_1+M_2)$ as follows.

Assume that $(x_1x_2\ldots x_\ell)_b+(y_1y_2\ldots y_\ell)_b=(z_0z_1\ldots z_\ell)_b$, where z_0 is the carry bit. On the basis of the digital circuit for binary integer addition [21], we have

$$c_{i-1}=x_iy_i\vee(x_i\oplus y_i)c_i,$$
$$z_i=x_i\oplus y_i\oplus c_i,$$

for $i=\ell,\ldots,2,1$, where $c_\ell=0$ and $z_0=c_0$. Due to $\alpha\vee\beta=(\alpha\oplus\beta)\oplus(\alpha\beta)$, one can compute

$$\begin{aligned}\mathcal{E}(a_{i-1})&=\mathcal{E}(x_i)\mathcal{E}(y_i)=\mathcal{E}(x_i\oplus y_i),\\ \mathcal{E}(b_{i-1})&=(\mathcal{E}(x_i)+\mathcal{E}(y_i))\mathcal{E}(c_i)=\mathcal{E}((x_i\oplus y_i)c_i),\\ \mathcal{E}(c_{i-1})&=(\mathcal{E}(a_{i-1})+\mathcal{E}(b_{i-1}))+\mathcal{E}(a_{i-1})\mathcal{E}(b_{i-1})\\ &=\mathcal{E}((a_{i-1}\oplus b_{i-1})\oplus a_{i-1}b_{i-1})\\ \mathcal{E}(z_i)&=\mathcal{E}(x_i)+\mathcal{E}(y_i)+\mathcal{E}(c_i)=\mathcal{E}(x_i\oplus y_i\oplus c_i),\end{aligned}$$

for $i=\ell,\ldots,2,1$, then let $\mathcal{E}(z_0)=\mathcal{E}(c_0)$ and $\mathcal{E}(M_1+M_2)=(\mathcal{E}(z_0),\mathcal{E}(z_1),\ldots,\mathcal{E}(z_\ell))$. We define $\mathcal{E}(M_1)\boxplus\mathcal{E}(M_2)=\mathcal{E}(M_1+M_2)$.

3.3 Integer Comparison

In particular, given $\mathcal{E}(M_1)$ and $\mathcal{E}(M_2)$ where M_1 and M_2 are two positive integers, we can compare M_1 with M_2 by computing

$$\mathcal{E}(\overline{M}_1)\boxplus\mathcal{E}(\overline{-M}_2)=\mathcal{E}(\overline{M}_1+\overline{-M}_2),$$

where $\overline{M}_1$ and $\overline{-M}_2$ are two's complements of M_1 and $-M_2$, respectively. Two's complement system is the most common method of representing signed integers on computers (please refer to [14], [15], [24]).

If $M_1\geq M_2$, the most significant bit of $\overline{M}_1+\overline{-M}_2$ is 0 and 1 otherwise.

Given $\mathcal{E}(M)=(\mathcal{E}(m_1),\mathcal{E}(m_2),\ldots,\mathcal{E}(m_\ell))$, we have

$$\mathcal{E}(\overline{M})=(\mathcal{E}(0),\mathcal{E}(m_1),\mathcal{E}(m_2),\ldots,\mathcal{E}(m_\ell)),$$
$$\mathcal{E}(\overline{-M})=(\mathcal{E}(1),\mathcal{E}(m_1)+1,\mathcal{E}(m_2)+1,\ldots,\mathcal{E}(m_\ell)+1)\boxplus\mathcal{E}(1).$$

3.4 Binary Linear Codes

An $[n,k]$ binary linear code C of length n and dimension k is a k-dimensional subspace of F_2^n according to [16]. A generator matrix for C is a $k\times n$ matrix

$$G=\begin{pmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\ a_{21}&a_{22}&\cdots&a_{2n}\\ \cdots&\cdots&\cdots&\cdots\\ a_{k1}&a_{k2}&\cdots&a_{kn}\end{pmatrix},$$

where $a_{ij}\in F_2$, such that $C=\{(b_1,b_2,\ldots,b_k)G\mid b_i\in F_2\}$. The matrix G corresponds to a map $F_2^k\rightarrow F_2^n$ expanding a message $(b_1,b_2,\ldots,b_k)$ of length k to an n-bit string.

We say that binary linear codes $C_1,C_2,\ldots,C_m$ are orthogonal if $C_i\cap C_j=\emptyset$ and $c_i\cdot c_j=0$ for any two codewords $c_i\in C_i$ and $c_j\in C_j(i,j=1,2,\ldots,m,i\neq j)$, where "$\cdot$" stands for the dot product operation. In case where $m=n/k$, there exist m simple orthogonal binary linear codes $C_1,C_2,\ldots,C_m$. The generator matrix of C_i is

$$G_i=\begin{pmatrix}\cdots&1&0&\cdots&0&\cdots&0&0&\cdots&0\\ \cdots&0&1&\cdots&0&\cdots&0&0&\cdots&0\\ \cdots&\cdot&\cdot&\cdots&\cdot&\cdots&\cdot&\cdot&\cdots&\cdot\\ \cdots&0&0&\cdots&1&\cdots&0&0&\cdots&0\end{pmatrix},$$

where the element at $(j,(i-1)k+j)$ (for $i=1,2,\ldots,m$ and $j=1,2,\ldots,k$) is 1 and otherwise 0.

4 Definitions

Definitions for general private queries were given in [17] and [18]. In this paper, slightly different definitions are given.

Like the streaming model given in [17] and [18], we consider a universe of words $W=\{0,1\}^*$, and a dictionary $D\subset W$ with $|D|<\infty$. We think of a document M just to be an ordered, finite sequence of words in W, and a stream of documents S just to be any sequence of documents. We define a set of keywords to be any subset $K\subset D$.

Definition 1. *A query $\mathcal{Q}$ over a set of keywords K, denoted as $\mathcal{Q}_K$, is a logical expression of keywords in K.*

Definition 2. *Given a document M and a query $\mathcal{Q}_K$, we define $\mathcal{Q}_K(M)=1$ if M matches the query $\mathcal{Q}_K$ and $\mathcal{Q}_K(M)=0$ otherwise.*

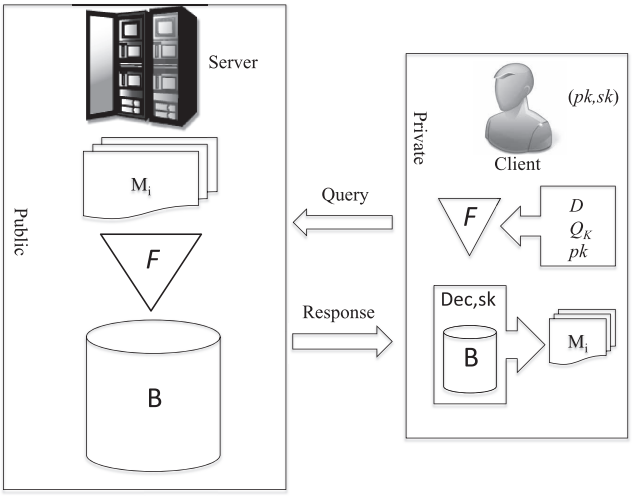

Fig. 1. Model for private searching on streaming data.

Definition 3. *For a query $\mathcal{Q}_K$, a private query protocol is composed of the following probabilistic polynomial time algorithms:*

1. ***KeyGen***(k). *Takes a security parameter k and generates a pair of public and secret keys (pk, sk).*
2. ***FilterGen***$(D, \mathcal{Q}_K, pk)$. *Takes a dictionary D, a query $\mathcal{Q}_K$, the public key pk, and generates a program F.*
3. ***FilterExec***(S, F, pk, m). *Takes a stream of documents S, F searches for any document $M \in S$ such that $\mathcal{Q}_K(M) = 1$ (processing one document at a time), and encrypts each matching document with the public key pk, and keeps up to m encrypted matching document in a buffer $\mathbb{B}$, and finally outputs an encrypted buffer $\mathbb{B}$.*
4. ***BufferDec***$(\mathbb{B}, sk)$. *Decrypts the encrypted buffer $\mathbb{B}$, produced by F as above, using the private key sk and outputs a plain buffer $\mathbb{B}^*$, a collection of the matching documents from S.*

Based on Definition 3, the model for privacy searching on stream data can be illustrated in Fig. 1.

Definition 4 (Correctness of Private Query Protocol). *Let $F = \mathrm{FilterExec}(S, F, pk, m)$, where D is a dictionary, $\mathcal{Q}_K$ is a query over keywords K, $(pk, sk) = \mathrm{KeyGen}(k)$, m is an upper bound on the number of matching documents, we say that a private query protocol is correct if the following holds: Let F run on any document stream S, $\mathbb{B} = F(S)$, $\mathbb{B}^* = \mathrm{BufferDec}(\mathbb{B}, sk)$.*

1. *(Compiled Program Conciseness) $|F| = O(|D|)$.*
2. *(Output Conciseness) $|\mathbb{B}| = O(m)$.*
3. *(Search Completeness) If $|\{M \in S \mid \mathcal{Q}_K(M) = 1\}| \leq m$, then*

$$\mathbb{B}^* = \{M \in S \mid \mathcal{Q}_K(M) = 1\}.$$

4. *(Collision Freeness) If $|\{M \in S \mid \mathcal{Q}_K(M) = 1\}| > m$, then*

$$|\mathbb{B}^* \cap \{M \in S \mid \mathcal{Q}_K(M) = 1\}| = m,$$

*where the probabilities are taken over all coin-tosses of F, **FilterGen**, and **KeyGen**.*

TABLE 1
Notations

Symbol	Explanation
D	dictionary of possible keywords
$\|D\|$	number of possible keywords in D
w_i	word in the dictionary and documents
K	set of classified keywords
k_i	classified keyword
n	number of classified keywords
Q_K	logical expression of keywords in K
F	filter program
M, M_i	document in the streaming data
d	maximal number of words in a document
$\mathbb{B}$	buffer to store matching documents
m	maximal number of matching documents in $\mathbb{B}$
(pk, sk)	public/private key pair
$\mathcal{E}_{pk}(b)$	encryption of a bit b using pk
$\mathcal{D}_{sk}(c)$	decryption of a ciphertext c using sk
$\hat{0}, \hat{1}$	encryptions of 0 and 1 using pk
$\|C\|$	size of the ciphertext
$f(k_i)$	frequency of keyword k_i in a document
t_i	frequency threshold of keyword k_i
$\hat{w}_i$	encryption of frequency threshold t_i
$\bar{t}$	two's complement of an integer t
$\boxplus$	homomorphic addition of integers
$\neg()$	complement of a condition

Definition 5 (Privacy). *Fix a dictionary D. Consider the following game between an adversary $\mathcal{A}$, and a challenger $\mathcal{C}$. The game consists of the following steps:*

1. *The challenger $\mathcal{C}$ first runs $\mathrm{KeyGen}(k)$ to obtain a pair of public and secret keys (pk, sk), and then sends pk and m, the upper bound on the number of matching documents, to $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ chooses two queries for two sets of keywords, $\mathcal{Q}_{0K_0}, \mathcal{Q}_{1K_1}$, with $K_0, K_1 \subset D$ and sends them to $\mathcal{C}$.*
3. *The challenger $\mathcal{C}$ chooses a random bit $b \in \{0, 1\}$ and executes $\mathrm{FilterGen}(D, \mathcal{Q}_{bK_b}, pk)$ to create F_b, the filtering program for the query $\mathcal{Q}_{bK_b}$, and then sends F_b back to $\mathcal{A}$.*
4. *The adversary $\mathcal{A}(F_b, pk, m)$ can experiment with code of F_b in an arbitrary nonblack-box way, and finally output $b' \in \{0, 1\}$.*

The adversary wins the game if $b' = b$ and loses otherwise. We define the adversary $\mathcal{A}$'s advantage in this game to be $\mathrm{Adv}_{\mathcal{A}}(k) = |\Pr(b' = b) - 1/2|$. We say that a private query protocol is semantically secure if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, we have that $\mathrm{Adv}_{\mathcal{A}}(k)$ is a negligible function, where the probability is taken over coin-tosses of the challenger and the adversary.

In the rest of this paper, we will use the notations as listed in Table 1.

5 Disjunctive Threshold Query Based on Keyword Frequency

Formally, a disjunctive threshold query over keywords $K = \{k_1, k_2, \ldots, k_n\}$ can be expressed as

$$\mathcal{Q}_K = (f(k_1) \geq t_1) \vee (f(k_2) \geq t_2) \vee \cdots \vee (f(k_n) \geq t_n),$$

where $f(k_i)(1 \le i \le n)$ is the frequency of the keyword k_i in the document and t_i is the given threshold. It is easy to see the following lemma.

Lemma 1. *Given a document M, a disjunctive threshold query $\mathcal{Q}_K(M) = 1$ if and only if there exists i such that $f(k_i) \ge t_i$.*

5.1 Construction

Following the model described in Section 4, our protocol for disjunctive threshold queries is composed of four algorithms KeyGen, FilterGen, FilterExec, BufferDec. Our construction is based on a fully homomorphic encryption scheme and can be formally presented as follows.

Key Generation. $\mathrm{KeyGen}(k)$. Run the key generation algorithm for the underlying fully homomorphic encryption scheme to produce the private key sk and the public key pk.

Filter Program Generation. $\mathrm{FilterGen}(D, \mathcal{Q}_K, pk)$. This algorithm outputs a filter program F for disjunctive threshold query $\mathcal{Q}_K$ based on keyword frequency.

Assume that the public dictionary $D = \{w_1, w_2, \ldots, w_{|D|}\}$, keywords $K = \{k_1, k_2, \ldots, k_n\} \subset D, d = \lceil \log_2 |M| \rceil$ where $|M|$ stands for the maximal number of words the document M may contain, then F consists of the dictionary D, disjunctive query sign (denoted as 00), and an array of ciphertexts

$$\hat{D} = \{\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{|D|}\},$$

where $\hat{w}_i = \mathcal{E}_{pk}(t_i)$ and

$$t_i = \begin{cases} frequency\ threshold\ for\ k_j & \text{if } w_i = k_j \in K \\ 2^d - 1 & \text{if } w_i \notin K. \end{cases}$$

Remark. Because the document M contains at most $2^d - 1$ words, the frequency of any word in M is less than $2^d - 1$. In practice, a document which repeats a word for $2^d - 1$ times is unusual. We do not consider this special case in our paper. We set the frequent threshold of a nonkeyword as $2^d - 1$ so that its frequency in M is never more than the threshold.

Assume $t_i = (a_{i1}a_{i2}\ldots a_{id})_b$, where $a_{ij} \in \{0, 1\}$, then $\hat{w}_i = \mathcal{E}_{pk}(t_i) = (\mathcal{E}_{pk}(a_{i1}), \mathcal{E}_{pk}(a_{i2}), \ldots, \mathcal{E}_{pk}(a_{id}))$. The array of ciphertexts $\hat{D}$ contains n encryptions of frequency thresholds and $|D| - n$ encryptions of $2^d - 1$.

Filter Program Execution. $\mathrm{FilterExec}(S, F, pk, m)$. This algorithm outputs an encrypted buffer $\mathbb{B}$ keeping up to m matching documents.

First of all, the program F constructs a data buffer $\mathbb{B}$ with $m\ell$ boxes, each of them is initialized with $\mathcal{E}_{pk}(0)$, where ℓ is the size of the document. Next, F constructs a base buffer $\mathbb{G}$ with m boxes, which are initialized with $(\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0), \mathcal{E}_{pk}(1))$.

Remark. The data buffer $\mathbb{B}$ is used to store the matching documents and the base buffer $\mathbb{G}$ is used to ensure the first m matching documents are stored in $\mathbb{B}$ without collision.

In addition, the program F constructs the encryption of the two's complement of $-t_i$ (denoted as $\overline{-t_i}$) with $\hat{w}_i = \mathcal{E}_{pk}(t_i)$, that is,

$$\mathcal{E}_{pk}(\overline{-t_i}) = (\mathcal{E}_{pk}(1), \mathcal{E}_{pk}(a_{i1}) + 1, \ldots, \mathcal{E}_{pk}(a_{id}) + 1) \boxplus \mathcal{E}_{pk}.(1)$$

The leftmost bit of the two's complement of an negative integer is 1 and otherwise 0.

Upon receiving an input document $M = (m_1 m_2 \ldots m_\ell)_b$ from the stream S, to determine if M is a matching document or not, the program F homomorphically compute a ciphertext $\mathcal{E}_{pk}(c_0)$ such that M is a matching document if $c_0 = 1$ and 0 otherwise. It proceeds with the following steps:

1. (*Word Collection*). The program F first collects

   $$\hat{H} = \{w_i, f(w_i) \mid w_i \in M \cap D\},$$

   where $f(w_i)$ is the frequency of w_i in the document M.

   Remark. $\hat{H}$ is the set of common words in the document M and the dictionary D and their frequencies in M.

   Next, F constructs the encryption of the two's complement of $f(w_i) = (b_{i1}b_{i2}\cdots b_{id})_b$, denoted as $\overline{f(w_i)}$, for each $w_i \in \hat{H}$, that is,

   $$\mathcal{E}_{pk}(\overline{f(w_i)}) = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(b_{i1}), \mathcal{E}_{pk}(b_{i2}), \ldots, \mathcal{E}_{pk}(b_{id})).$$

   Remark. Because $f(w_i) < 2^d - 1$, we only consider the encryptions of the d bits and one sign bit.
2. (*Frequency Comparison*). For each $w_i \in \hat{H}$, the program F homomorphically compares the frequency $f(w_i)$ and the frequency threshold t_i by computing

   $$\begin{aligned} &\mathcal{E}_{pk}(\overline{f(w_i)} + \overline{-t_i}) \\ &\quad = \mathcal{E}_{pk}(\overline{f(w_i)}) \boxplus \mathcal{E}_{pk}(\overline{-t_i}) \\ &\quad = (\mathcal{E}_{pk}(c_{i0}), \mathcal{E}_{pk}(c_{i1}), \mathcal{E}_{pk}(c_{i2}), \ldots, \mathcal{E}_{pk}(c_{id})), \end{aligned}$$

   from which only $\mathcal{E}_{pk}(c_{i0})$ is extracted. In two's complement system, if $c_{i0} = 0$, then $f(w_i) \ge t_i$ and otherwise $f(w_i) < t_i$.

   Next, the program F computes

   $$\mathcal{E}_{pk}(c_0) = \mathcal{E}_{pk}\left(\bigvee_{w_i \in \hat{H}} (c_{i0} \oplus 1)\right) \tag{1}$$

   by repeatedly using $\mathcal{E}_{pk}(c_{i0} \vee s) = \mathcal{E}_{pk}(c_{i0}) + \mathcal{E}_{pk}(s) + \mathcal{E}_{pk}(c_{i0})\mathcal{E}_{pk}(s)$.

   If $c_0 = 1$, then there exists i such that $c_{i0} \oplus 1 = 1$ (i.e., $c_{i0} = 0$ and $f(w_i) \ge t_i$). If $w_i \notin K$, then $t_i = 2^d - 1$ and it is impossible that $f(w_i) \ge 2^d - 1$. This means that $w_i \in K$ and $f(w_i) \ge t_i$. According to Lemma 1, M is a matching document.

   If $c_0 = 0$, then $c_{i0} \oplus 1 = 0$ (i.e., $c_{i0} = 1$ and $f(w_i) < t_i$) for all $w_i \in M \cap D$. According to Lemma 1, M is not a matching document.
3. (*Document Storing*). Assume that the state of the base buffer $\mathbb{G}$ is $(\hat{g}_m, \hat{g}_{m-1}, \ldots, \hat{g}_1)$, where $\hat{g}_i$ is an encryption of either 0 or 1, the program F constructs an encrypted $\ell \times L$ generator matrix G for an $[L, \ell]$ binary linear code as follows:

   $$G = \begin{pmatrix} \hat{g}_1 & \hat{0} & \cdots & \hat{0} & \cdots & \hat{g}_m & \hat{0} & \cdots & \hat{0} \\ \hat{0} & \hat{g}_1 & \cdots & \hat{0} & \cdots & \hat{0} & \hat{g}_m & \cdots & \hat{0} \\ \cdot & \cdot & \cdots & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ \hat{0} & \hat{0} & \cdots & \hat{g}_1 & \cdots & \hat{0} & \hat{0} & \cdots & \hat{g}_m \end{pmatrix},$$

where $L = m\ell$ and the element at $(i, (j-1)\ell + i)$ (for $i = 1, 2, \ldots, \ell$ and $j = 1, 2, \ldots, m$) is $\hat{g}_j$ and otherwise $\hat{0}$ (an encryption of 0).

To store the encryption of the document M into the data buffer $\mathbb{B}$, the program F computes

$$\begin{aligned}\hat{M} &= \mathcal{E}_{pk}(c_0)\mathcal{E}_{pk}(M)G \\ &= (\mathcal{E}_{pk}(c_0 m_1), \ldots, \mathcal{E}_{pk}(c_0 m_\ell))G,\end{aligned}$$

and positionwise adds the result into the data buffer $\mathbb{B}$, denoted as

$$\mathbb{B} = \mathbb{B} + \hat{M}.$$

If $c_0 = 1$, then $\hat{M}$, the encryption of the binary linear code of the matching document M, is kept in the data buffer $\mathbb{B}$. If $c_0 = 0$, then $\hat{M}$ is the encryption of 0, which has no effect on the data buffer $\mathbb{B}$.

4. To avoid collision when storing next matching document into the data buffer $\mathbb{B}$, the program F updates the base buffer $\mathbb{G}$ by homomorphically shifting $\mathcal{E}_{pk}(1)$ in the base buffer $\mathbb{G}$ to the left one position if M is a matching document and 0 position otherwise. This is done by computing

$$\mathbb{G}' = \mathbb{G} \boxplus \mathcal{E}_{pk}(c_0)\mathbb{G},$$

where $\mathbb{G}$ is treated as the encryption of an m-bit integer, and replacing $\mathbb{G}$ with $\mathbb{G}'$.

Remark. Initially, $\mathbb{G} = (\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0), \mathcal{E}_{pk}(1))$. If $c_0 = 0$, the buffer does not change. Only when $c_0 = 1$, the buffer is updated by shifting $\mathcal{E}_{pk}(1)$ one position to the left. We only consider the encryptions of the right m bits. After shifting m times, the buffer becomes the encryptions of all zeros. The buffer contains at most one encryption of 1 all the time.

Buffer Decryption. $\mathrm{BufferDec}(\mathbb{B}, sk)$. Using the secret key sk, the algorithm decrypts the encrypted data buffer $\mathbb{B}$, sent back by the filter program F, one box at a time. Assume that the decrypted data buffer is $(m'_1 m'_2 \ldots m'_L)_b$ where $L = m\ell$, then the set of matching documents is

$$\mathbb{B}^* = \{M = (m'_{i\ell+1} \ldots m'_{i\ell+\ell})_b \mid M \neq 0, i = 0, 1, \ldots, m-1\}.$$

5.2 Correctness

The filter program F is composed of D (the dictionary) and $\hat{D}$ (the encryption of the frequency thresholds). The size of $\hat{D}$ is $|D|dk$, where k is the security parameter. Therefore, the size of the filter program $|F| = O(|D|)$.

The data buffer $\mathbb{B}$ has $m\ell$ boxes (where ℓ is the size of the document), each keeps a ciphertext of one bit. The size of the buffer $|\mathbb{B}| = m\ell k = O(m)$.

We need to show that if the number of matching documents is less than or equal to m, then $\mathbb{B}^* = \{M \in S \mid \mathcal{Q}_K(M) = 1\}$ (search completeness) and otherwise we have $|\mathbb{B}^* \cap \{M \in S \mid \mathcal{Q}_K(M) = 1\}| = m$ (collision freeness).

Assume that the matching documents in the stream $S = \{M_1, M_2, \ldots,\}$ are $\{M_{i_1}, M_{i_2}, \ldots\}$. Initially, the data buffer $\mathbb{B} = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0))$, the base buffer $\mathbb{G} = (\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0), \mathcal{E}_{pk}(1))$ and the generator matrix

$$G = \begin{pmatrix} \hat{1} & \hat{0} & \cdots & \hat{0} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \\ \hat{0} & \hat{1} & \cdots & \hat{0} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdot & \cdots & \cdot \\ \hat{0} & \hat{0} & \cdots & \hat{1} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \end{pmatrix},$$

where $\hat{1}$ and $\hat{0}$ are encryptions of 1 and 0, respectively.

For a nonmatching document M, we have $c_0 = 0$ and thus $\hat{M} = \mathcal{E}_{pk}(c_0)\mathcal{E}_{pk}(M)G = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0))$, the data buffer $\mathbb{B} = \mathbb{B} + \hat{M} = \mathbb{B}$ and the base buffer $\mathbb{G}' = \mathbb{G} \boxplus \mathcal{E}_{pk}(c_0)\mathbb{G} = \mathbb{G}$, which means that the content of $\mathbb{B}$ and $\mathbb{G}$ do not change.

When the filter program F deals with the matching document M_{i_j} $(1 \leq j \leq m)$, we have $c_0 = 1$ and the state of the base buffer $\mathbb{G}$ is evolved from $(\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0), \mathcal{E}_{pk}(1))$ by shifting $\mathcal{E}_{pk}(1)$ to the left $j-1$ positions because there are $j-1$ matching documents before M_{i_j}. Therefore, the generator matrix

$$G = \begin{pmatrix} \cdots & \hat{1} & \hat{0} & \cdots & \hat{0} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \\ \cdots & \hat{0} & \hat{1} & \cdots & \hat{0} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \\ \cdots & \cdot & \cdot & \cdots & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ \cdots & \hat{0} & \hat{0} & \cdots & \hat{1} & \cdots & \hat{0} & \hat{0} & \cdots & \hat{0} \end{pmatrix},$$

and $\hat{M}_{i_j} = \mathcal{E}_{pk}(c_0)\mathcal{E}_{pk}(M_{i_j})G = (\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(M_{i_j}), \ldots, \mathcal{E}_{pk}(0))$ and $\mathbb{B} = \mathbb{B} + \hat{M}_{i_j} = (\mathcal{E}_{pk}(M_{i_1}), \ldots, \mathcal{E}_{pk}(M_{i_{j-1}}), \mathcal{E}_{pk}(M_{i_j}), \mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0))$. After that, the ba se buffer $\mathbb{G}$ is updated to $\mathbb{G} \boxplus \mathcal{E}_{pk}(c_0)\mathbb{G} = \mathbb{G} \boxplus \mathbb{G}$, i.e., shifting $\mathcal{E}_{pk}(1)$ further to the left one position.

In case when the filter program F deals with the matching document M_{i_j} $(j > m)$, although $c_0 = 1$, the base buffer $\mathbb{G}$ contains the encryptions of all zeros and so does the generator matrix G. Therefore, $\hat{M}_{i_j} = \mathcal{E}_{pk}(c_0)\mathcal{E}_{pk}(M_{i_j})G = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0))$ and $\mathbb{B} = \mathbb{B} + \hat{M}_{i_j} = \mathbb{B}$. This means the matching document $M_{i_j} (j > m)$ has no effect on the data buffer $\mathbb{B}$.

In summary, both search completeness and collision freeness are true.

6 Conjunctive Threshold Query Based on Keyword Frequency

Formally, a conjunctive threshold query over keywords $K = \{k_1, k_2, \ldots, k_n\}$ can be expressed as

$$\mathcal{Q}_K = (f(k_1) \geq t_1) \wedge (f(k_2) \geq t_2) \wedge \cdots \wedge (f(k_n) \geq t_n),$$

where $f(k_i) (1 \leq i \leq n)$ is the frequency of the keyword k_i in the document and t_i is the given threshold. It is easy to see

Lemma 2. *Given a document M, a conjunctive threshold query $\mathcal{Q}_K(M) = 1$ if and only if $f(k_i) \geq t_i$ for $1 \leq i \leq n$.*

6.1 Construction

Following the model described in Section 4, our protocol of conjunctive threshold query is composed of four algorithms KeyGen, FilterGen, FilterExec, BufferDec. Our conjunctive construction can be formally presented as follows.

Key Generation. $\mathrm{KeyGen}(k)$. Run the key generation algorithm for the underlying fully homomorphic encryption scheme to produce the private key sk and the public key pk.

Filter program generation. $\mathrm{FilterGen}(D, \mathcal{Q}_K, pk)$. This algorithm outputs a filter program F for conjunctive threshold query $\mathcal{Q}_K$ based on keyword frequency.

Assume that the public dictionary $D = \{w_1, w_2, \ldots, w_{|D|}\}$, keywords $K = \{k_1, k_2, \ldots, k_n\} \subset D$, $d = \lceil \log_2 |M| \rceil$ where $|M|$ stands for the maximal number of words the document M can contain, then F consists of the dictionary D, conjunctive query sign (denoted as 01), and an array of ciphertexts

$$\hat{D} = \{\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{|D|}\},$$

where $\hat{w}_i = \mathcal{E}_{pk}(t_i)$ and

$$t_i = \begin{cases} frequency\ threshold\ for\ k_j & \text{if } w_i = k_j \in K \\ 0 & \text{if } w_i \notin K. \end{cases}$$

Remark. Because the document M contains at most $2^d - 1$ words, both t_i and $t = \sum_{w_i \in K} t_i$ must be less than $2^d - 1$. We set the frequent threshold of a nonkeyword as 0 so that its frequency is always more than the threshold.

Assume $t_i = (a_{i1} a_{i2} \ldots a_{id})_b$ where $a_{ij} \in \{0, 1\}$, then $\hat{w}_i = \mathcal{E}_{pk}(t_i) = (\mathcal{E}_{pk}(a_{i1}), \mathcal{E}_{pk}(a_{i2}), \ldots, \mathcal{E}_{pk}(a_{id}))$. The array of ciphertexts contains n encryptions of frequency thresholds and $|D| - n$ encryptions of 0.

Filter program execution $\mathrm{FilterExec}(S, F, pk, m)$. This algorithm outputs an encrypted buffer $\mathbb{B}$ keeping up to m matching documents.

First of all, the program F constructs a data buffer $\mathbb{B}$ with $m\ell$ boxes, each of them is initialized with $\mathcal{E}_{pk}(0)$. Next, F constructs a base buffer $\mathbb{G}$ with m boxes, which are initialized with $(\mathcal{E}_{pk}(0), \ldots, \mathcal{E}_{pk}(0), \mathcal{E}_{pk}(1))$. In addition, the program F constructs the encryption of the two's complement of $-t_i$ (denoted as $\overline{-t_i}$) with $\hat{w}_i = \mathcal{E}_{pk}(t_i)$, that is,

$$\mathcal{E}_{pk}(\overline{-t_i}) = (\mathcal{E}_{pk}(1), \mathcal{E}_{pk}(a_{i1}), \ldots, \mathcal{E}_{pk}(a_{id})) \boxplus \mathcal{E}_{pk}(1),$$

and the encryption of $t = \sum_{w_i \in K} t_i$ with $\hat{D}$ (please note that $t_i = 0$ when $w_i \notin K$), that is,

$$\boxplus_{i=1}^{|D|} \hat{w}_i = \hat{w}_1 \boxplus \hat{w}_2 \boxplus \cdots \boxplus \hat{w}_{|D|},$$

and the encryption of the two's complement of $-t$ (denoted as $\overline{-t}$) with $\mathcal{E}_{pk}(t)$, that is,

$$\mathcal{E}_{pk}(\overline{-t}) = (\mathcal{E}_{pk}(1), \mathcal{E}_{pk}(\alpha_1), \ldots, \mathcal{E}_{pk}(\alpha_d)) \boxplus \mathcal{E}_{pk}(1).$$

Upon receiving an input document $M = (m_1 m_2 \ldots m_\ell)_b$ from the stream S, to determine if M is a matching document or not, the program F homomorphically compute a ciphertext $\mathcal{E}_{pk}(c_0)$ such that M is a matching document if $c_0 = 1$ and 0 otherwise. It proceeds with the following steps:

1. (*Word Collection*). The program F first collects

$$\hat{H} = \{w_i, f(w_i) \mid w_i \in M \cap D\},$$

where $f(w_i)$ is the frequency of w_i in the document M. Next, F constructs the encryption of the two's complement of $f(w_i) = (b_{i1} b_{i2} \ldots b_{id})_b$, denoted as $\overline{f(w_i)}$, for each $w_i \in \hat{H}$, that is,

$$\mathcal{E}_{pk}(\overline{f(w_i)}) = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(b_{i1}), \mathcal{E}_{pk}(b_{i2}), \ldots, \mathcal{E}_{pk}(b_{id})),$$

and the encryption of the two's complement of $t' = \sum_{w_i \in \hat{H}} t_i = (\beta_1, \beta_2, \ldots, \beta_d)$, denoted as $\overline{t'}$, that is

$$\mathcal{E}_{pk}(\overline{t'}) = (\mathcal{E}_{pk}(0), \mathcal{E}_{pk}(\beta_1), \mathcal{E}_{pk}(\beta_2), \ldots, \mathcal{E}_{pk}(\beta_d)).$$

Remark: $\mathcal{E}_{pk}(t') = (\mathcal{E}_{pk}(\beta_1), \mathcal{E}_{pk}(\beta_2), \ldots, \mathcal{E}_{pk}(\beta_d))$ can be obtained with $\boxplus_{w_i \in \hat{H}} \hat{w}_i$. Because the sum t' is never more than $2^d - 1$, we consider d bits of t' only.

2. (*Frequency Comparison*). For each $w_i \in \hat{H}$, the program F homomorphically compares $f(w_i)$ and t_i by computing

$$\begin{aligned} &\mathcal{E}_{pk}(\overline{f(w_i)} + \overline{-t_i}) \\ &\quad = \mathcal{E}_{pk}(\overline{f(w_i)}) \boxplus \mathcal{E}_{pk}(\overline{-t_i}) \\ &\quad = (\mathcal{E}_{pk}(c_{i0}), \mathcal{E}_{pk}(c_{i1}), \mathcal{E}_{pk}(c_{i2}), \ldots, \mathcal{E}_{pk}(c_{id})), \end{aligned}$$

from which only $\mathcal{E}_{pk}(c_{i0})$ is extracted. If $c_{i0} = 0$, then $f(w_i) \geq t_i$ and otherwise $f(w_i) < t_i$.

In addition, the program F homomorphically checks if the document M contains all keywords in K by computing

$$\begin{aligned} &\mathcal{E}_{pk}(\overline{t'} + \overline{-t}) \\ &\quad = \mathcal{E}_{pk}(\overline{t'}) \boxplus \mathcal{E}_{pk}(\overline{-t}) \\ &\quad = (\mathcal{E}_{pk}(\gamma_0), \mathcal{E}_{pk}(\gamma_1), \mathcal{E}_{pk}(\gamma_2), \ldots, \mathcal{E}_{pk}(\gamma_d)), \end{aligned}$$

from which only $\mathcal{E}_{pk}(\gamma_0)$ is extracted. If $\gamma_0 = 0$, then $t' \geq t$ and thus $t' = t$ and the document contains all keywords in K. If $\gamma_0 = 1$, then $t' < t$ and the document does not contain all keywords in K.

Remark. Because $t' = \sum_{w_i \in \hat{H}} t_i = \sum_{w_i \in \hat{H} \cap K} t_i \leq \sum_{w_i \in K} t_i = t$, the inequality $t' \geq t$ means that $t' = t$, $\hat{H} \cap K = K$, and the document contains all keywords in K. Reversely, the inequality $t' < t$ means that $\hat{H} \cap K \subset K$ and the document does not contain all keywords in K.

Next, the program F computes

$$\begin{aligned} \mathcal{E}_{pk}(c_0) &= \mathcal{E}_{pk}\left((\gamma_0 \oplus 1) \bigwedge_{w_i \in \hat{H}} (c_{i0} \oplus 1)\right) \\ &= (\mathcal{E}_{pk}(\gamma_0) + \mathcal{E}_{pk}(1)) \prod_{w_i \in \hat{H}} (\mathcal{E}_{pk}(c_{i0}) + \mathcal{E}_{pk}(1)). \end{aligned} \tag{2}$$

If $c_0 = 1$, then $\gamma_0 = 0$ and $c_{i0} = 0$ for all $w_i \in \hat{H}$. As discussed above, $\gamma_0 = 0$ means $\hat{H} \cap K = K$ while $c_{i0} = 0$ for all $w_i \in \hat{H}$ means $f(w_i) \geq t_i$ for all $w_i \in \hat{H}$. It is obvious that $f(w_i) \geq 0$ for all $w_i \notin K$. According to Lemma 2, M is a matching document.

If $c_0 = 0$ and $\gamma_0 = 1$, M does not contain all keywords in K. According to Lemma 2, M is not a matching document. If $c_0 = 0$ and $\gamma_0 = 0$, M does contain all keywords in K, but there exists i such that $f(w_i) < t_i$. According to Lemma 2, M is not a matching document.

The rest of the algorithm and the buffer decryption algorithm are the same as our disjunction threshold query.

The correctness of our conjunctive threshold query can be proved in the same way as we prove the correctness of our disjunctive threshold query.

7 Complement Threshold Query Based on Keyword Frequency

We have two complement constructions for private threshold queries based on keyword frequency. They are disjunctive complement and conjunctive complement.

7.1 Disjunctive Complement

Formally, a disjunctive complement threshold query over keywords $K = \{k_1, k_2, \ldots, k_n\}$ can be expressed as

$$\begin{aligned}\mathcal{Q}_K &= (f(k_{i_1}) \geq t_{i_1}) \vee \cdots \vee (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \\ &\quad \vee \neg(f(k_{j_1}) \geq t_{j_1}) \vee \cdots \vee \neg(f(k_{j_{n_2}}) \geq t_{j_{n_2}}) \\ &= (f(k_{i_1}) \geq t_{i_1}) \vee \cdots \vee (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \\ &\quad \vee (f(k_{j_1}) < t_{j_1}) \vee \cdots \vee (f(k_{j_{n_2}}) < t_{j_{n_2}}),\end{aligned}$$

where $\neg$ stands for complement (i.e., negation), $\{k_{i_1}, \ldots, k_{i_{n_1}}, k_{j_1}, \ldots, k_{j_{n_2}}\} = K$ and $n_1 \geq 0$, $n_2 \geq 0$. It is easy to see

Lemma 3. *Given a document M, a conjunctive complement query $\mathcal{Q}_K(M) = 1$ if and only if there exists l such that $f(k_{i_l}) \geq t_{i_l}$ or $f(k_{j_l}) < t_{j_l}$.*

Our construction for the conjunctive complement query is composed of KeyGen, FilterGen, FilterExec, and BufferDec, where KeyGen and BufferDec are the same as the disjunctive threshold query described in Section 5.

Filter program generation. $\text{FilterGen}(D, \mathcal{Q}_K, pk)$. This algorithm outputs a filter program F, which consists of the public dictionary $D = \{w_1, w_2, \ldots, w_{|D|}\}$, disjunctive complement sign (denoted as 10), an array of ciphertexts $\hat{D} = \{\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{|D|}\}$, where $\hat{w}_i = \mathcal{E}_{pk}(t_i)$ and

$$t_i = \begin{cases} frequency\ threshold\ for\ k_j & \text{if } w_i = k_j \in K \\ 2^d - 1 & \text{if } w_i \notin K, \end{cases}$$

and an array of ciphertexts $\hat{D}' = \{\hat{w}'_1, \hat{w}'_2, \ldots, \hat{w}'_{|D|}\}$, where $\hat{w}'_i = \mathcal{E}_{pk}(s_i)$ and

$$s_i = \begin{cases} 1 & \text{if } w_i \in \{k_{j_1}, \ldots, k_{j_{n_2}}\} \\ 0 & \text{otherwise.} \end{cases}$$

Remark. The encryptions of $s_1, s_2, \ldots, s_n$ are used to indicate the complement positions in Q_K in private.

Filter program execution. $\text{FilterExec}(S, F, pk, m)$. This algorithm outputs an encrypted buffer $\mathbb{B}$ keeping up to m matching documents.

The algorithm is the same as the filter program execution in the disjunctive threshold query described in Section 5 except that F computes

$$\mathcal{E}_{pk}(c_0) = \mathcal{E}_{pk}\Big(\bigvee_{w_i \in \hat{H}} (c_{i0} \oplus 1 \oplus s_i)\Big), \tag{3}$$

on the basis of homomorphic properties described in Section 3.

If $c_0 = 1$, then there exists l such that $c_{l0} \oplus 1 \oplus s_l = 1$ (i.e., $c_{l0} \oplus s_l = 0$). If $w_l \in \{k_{i_1}, \ldots, k_{i_{n_1}}\}$, then $s_l = 0$ and thus $c_{l0} = 0$, which means that $f(w_l) \geq t_l$. If $w_l \in \{k_{j_1}, \ldots, k_{j_{n_2}}\}$, then $s_l = 1$ and thus $c_{l0} = 1$, which means that $f(w_l) < t_l$. If $w_l \notin K$, then $s_l = 0$ and thus $c_{l0} = 0$, which means that $f(w_l) \geq t_l = 2^d - 1$. It is impossible and this event never occurs when $c_0 = 1$. According to Lemma 3, M is a matching document when $c_0 = 1$.

If $c_0 = 0$, then $c_{l0} \oplus 1 \oplus s_l = 0$ (i.e., $c_{l0} \oplus s_l = 1$) for all $w_l \in M \cap D$. If $w_l \in \{k_{i_1}, \ldots, k_{i_{n_1}}\}$, then $s_l = 0$ and thus $c_{l0} = 1$, which means that $f(w_l) < t_l$. If $w_l \in \{k_{j_1}, \ldots, k_{j_{n_2}}\}$, then $s_l = 1$ and thus $c_{l0} = 0$, which means that $f(w_l) \geq t_l$. According to Lemma 3, M is not a matching document when $c_0 = 0$.

Remark. A disjunctive complement query becomes a disjunctive query if letting $s_i = 0$ for all i. In addition, if letting $s_i = 1$ for all i, a disjunctive complement query becomes

$$\mathcal{Q}_K = (f(k_1) < t_1) \vee (f(k_2) < t_2) \vee \cdots \vee (f(k_n) < t_n).$$

7.2 Conjunctive Complement

Formally, a conjunctive complement threshold query over keywords $K = \{k_1, k_2, \ldots, k_n\}$ can be expressed as

$$\begin{aligned}\mathcal{Q}_K &= (f(k_{i_1}) \geq t_{i_1}) \wedge \cdots \wedge (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \\ &\quad \wedge \neg(f(k_{j_1}) \wedge t_{j_1}) \vee \cdots \wedge \neg(f(k_{j_{n_2}}) \geq t_{j_{n_2}}) \\ &= (f(k_{i_1}) \geq t_{i_1}) \wedge \cdots \vee (f(k_{i_{n_1}}) \geq t_{i_{n_1}}) \\ &\quad \wedge (f(k_{j_1}) < t_{j_1}) \wedge \cdots \vee (f(k_{j_{n_2}}) < t_{j_{n_2}}),\end{aligned}$$

where $\neg$ stands for complement (i.e., negation), $\{k_{i_1}, \ldots, k_{i_{n_1}}, k_{j_1}, \ldots, k_{j_{n_2}}\} = K$ and $n_1 \geq 0$, $n_2 \geq 0$. It is easy to see

Lemma 4. *Given a document M, a conjunctive complement query $\mathcal{Q}_K(M) = 1$ if and only if, for any $k_l \in \{k_{i_1}, k_{i_2}, \ldots, k_{i_{n_1}}\}$, $f(k_l) \geq t_l$, and for any $k_l \in \{k_{j_1}, k_{j_2}, \ldots, k_{j_{n_2}}\}$, $f(k_l) < t_l$.*

Our construction for the conjunctive complement query is composed of KeyGen, FilterGen, FilterExec, and BufferDec, where KeyGen and BufferDec are the same as the disjunctive threshold query described in Section 5.

Filter program generation. $\text{FilterGen}(D, \mathcal{Q}_K, pk)$. This algorithm outputs a filter program F, which consists of the public dictionary $D = \{w_1, w_2, \ldots, w_{|D|}\}$, conjunctive complement sign (denoted as 11), an array of ciphertexts $\hat{D} = \{\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{|D|}\}$, where $\hat{w}_i = \mathcal{E}_{pk}(t_i)$ and

$$t_i = \begin{cases} frequency\ threshold\ for\ k_j & \text{if } w_i = k_j \in K \\ 0 & \text{if } w_i \notin K, \end{cases}$$

and an array of ciphertexts $\hat{D}' = \{\hat{w}'_1, \hat{w}'_2, \ldots, \hat{w}'_{|D|}\}$, where $\hat{w}'_i = \mathcal{E}_{pk}(s_i)$ and

$$s_i = \begin{cases} 1 & \text{if } w_i \in \{k_{j_1}, \ldots, k_{j_{n_2}}\} \\ 0 & \text{otherwise.} \end{cases}$$

Filter program execution. $\text{FilterExec}(S, F, pk, m)$. This algorithm outputs an encrypted buffer $\mathbb{B}$ keeping up to m matching documents.

The algorithm is the same as the filter program execution in the conjunctive threshold query described in Section 6 except that F computes

$$\mathcal{E}_{pk}(c_0) = \mathcal{E}_{pk}\Big((\gamma_0 \oplus 1) \bigwedge_{w_i \in \hat{H}} (c_{i0} \oplus 1 \oplus s_i)\Big) = (\mathcal{E}_{pk}(\gamma_0) + \mathcal{E}_{pk}(1)) \prod_{w_i \in \hat{H}} (\mathcal{E}_{pk}(c_{i0}) + \mathcal{E}_{pk}(1) + \hat{w}'_i), \tag{4}$$

according to homomorphic properties described in Section 3.

If $c_0 = 1$, then $\gamma_0 = 0$ and $c_{l0} \oplus 1 \oplus s_l = 1$ (i.e., $c_{l0} + s_l = 0$) for all $w_l \in \hat{H}$. $\gamma_0 = 0$ means $\hat{H} \cap K = K$. If $w_l \in \{k_{i_1}, \ldots, k_{i_{n_1}}\}$, then $s_l = 0$ and thus $c_{l0} = 0$, which means that $f(w_l) \geq t_l$. If $w_l \in \{k_{j_1}, \ldots, k_{j_{n_2}}\}$, then $s_l = 1$ and thus $c_{l0} = 1$, which means that $f(w_l) < t_l$. According to Lemma 4, M is a matching document when $c_0 = 1$.

If $c_0 = 0$ and $\gamma_0 = 1$, M does not contain all keywords in K. According to Lemma 4, M is not a matching document. If $c_0 = 0$ and $\gamma_0 = 0$, M does contain all keywords in K, but there exists l such that $c_{l0} \oplus 1 \oplus s_l = 0$ (i.e., $c_{l0} \oplus s_l = 1$). If $w_l \in \{k_{i_1}, \ldots, k_{i_{n_1}}\}$, then $s_l = 0$ and thus $c_{l0} = 1$, which means that $f(w_l) < t_l$. If $w_l \in \{k_{j_1}, \ldots, k_{j_{n_2}}\}$, then $s_l = 1$ and thus $c_{l0} = 0$, which means that $f(w_l) \geq t_l$. According to Lemma 4, M is a matching document when $c_0 = 0$.

Remark. A conjunctive complement query becomes a conjunctive query if letting $s_i = 0$ for all i. In addition, if letting $s_i = 1$ for all i, a disjunctive complement query becomes

$$\mathcal{Q}_K = (f(k_1) < t_1) \wedge (f(k_2) < t_2) \wedge \cdots \wedge (f(k_n) < t_n).$$

8 Generic Threshold Query Based on Keyword Frequency

By combining the above basic constructions for private threshold queries based on keyword frequency, we present a construction for a generic threshold query without asymptotically increasing the program size as follows.

Assume that D is the public dictionary of potential keywords and $\mathcal{Q}^{(i)}_{K_i}(i = 1, 2, \ldots, \lambda)$ stands for a disjunctive, or conjunctive, or complement query over keywords $K_i \subset D$, we consider a generic threshold query

$$\Phi\big(Q^{(1)}_{K_1}, Q^{(2)}_{K_2}, \ldots, Q^{(\lambda)}_{K_\lambda}\big),$$

where operators in Φ belong to $\{\vee, \wedge, \oplus\}$ and $K_i \cap K_j$ for any i and j is not necessary to be empty.

Our construction for the generic threshold query over keywords K_i $(i = 1, 2, \ldots, \lambda)$ is composed of KeyGen, FilterGen, FilterExec, and BufferDec, where KeyGen and BufferDec are the same as the threshold queries described in Section 5.

Filter program generation. $\mathrm{FilterGen}(D, \mathcal{Q}^{(1)}_{K_1}, \mathcal{Q}^{(2)}_{K_2}, \ldots, \mathcal{Q}^{\lambda}_{K_\lambda}, pk)$. This algorithm outputs a filter program F, which consists of $\{F_1, F_2, \ldots, F_\lambda\}$, where $F_i = \mathrm{FilterGen}(D, \mathcal{Q}^{(i)}_{K_i}, pk)$.

Filter program execution. $\mathrm{FilterExec}(S, F, pk, m)$. This algorithm outputs an encrypted buffer $\mathbb{B}$ keeping up to m matching documents. Upon receiving an input document $M = (m_1 m_2 \ldots m_\ell)_b$ from the stream S, the program F proceeds with the following steps:

1. The program F runs the programs F_i to compute $\mathcal{E}_{pk}(c^{(i)}_0)$ based on (1)-(4).
2. The program F computes

$$\mathcal{E}_{pk}(c_0) = \mathcal{E}_{pk}\big(\Phi\big(c^{(1)}_0, c^{(2)}_0, \ldots, c^{(\lambda)}_0\big)\big),$$

according to homomorphic properties described in Section 3.

If $c_0 = 1$, M is a matching document. If $c_0 = 0$, M is not a matching document.

The rest of the construction is the same as FilterExec of the disjunction threshold query described in Section 5.

Remark. All kind of private threshold queries based on keyword frequency can be expressed as $\Phi(Q^{(1)}_{K_1}, Q^{(2)}_{K_2}, \ldots, Q^{(\lambda)}_{K_\lambda})$, where $\mathcal{Q}^{(i)}_{K_i}$ is either disjunctive, conjunctive, or complement threshold query, and operators in Φ belong to $\{\vee, \wedge, \oplus\}$. Therefore, our solution supports arbitrary private threshold queries.

9 Privacy

The privacy of our threshold query protocols is built on the underlying fully homomorphic encryption scheme. We have:

Theorem 1. *Assume that the underlying fully homomorphic encryption scheme is semantically secure, then our threshold query protocols based on keyword frequency are semantically secure according to Definition 5.*

Proof. We consider the privacy of the disjunctive threshold query based on keyword frequency in the proof. The privacy of other threshold queries can be proved in the same way.

Denote by $\mathcal{E}$ the underlying fully homomorphic encryption scheme. Suppose that there exists an adversary $\mathcal{A}$ that can gain a nonnegligible advantage ϵ in our semantic security game from Definition 5. Then, $\mathcal{A}$ can be used to gain a nonnegligible advantage in breaking the semantic security of the underlying fully homomorphic encryption scheme as follows:

Initiate the semantic security game for the encryption scheme with some challenger $\mathcal{C}$, which will send us the public key pk for the challenge. For messages m_0 and m_1, we choose $m_0 = 0 \in \{0, 1\}$ and $m_1 = 1 \in \{0, 1\}$. After sending m_0, m_1 back to the challenger $\mathcal{C}$, we will receive $e_b = \mathcal{E}(m_b)$, an encryption of one of these two values. Next, we initiate the private query game with the adversary $\mathcal{A}$, who will give us two disjunctive threshold queries $\mathcal{Q}_{K_0}, \mathcal{Q}_{K_1}$ for two sets of keywords $K_0 \subset D$ and $K_1 \subset D$ with frequency thresholds $\{t_{i0}\}(i = 1, 2, \ldots, |D|)$ for Q_{K_0} and frequency threshold $\{t_{i1}\}(i = 1, 2, \ldots, |D|)$ for Q_{K_1}. We pick a random bit q, and construct a private filter program F for $\mathcal{Q}_{K_q}$, i.e., $\hat{D} = \{\hat{w}_{1q}, \hat{w}_{2q}, \ldots, \hat{w}_{|D|q}\}$ as follows.

For a word w_i in $D - K_q$, $t_{iq} = 2^d - 1 = (11 \ldots 1)_b$, we construct $\hat{w}_{iq} = (\mathcal{E}_{pk}(1), \mathcal{E}_{pk}(1), \ldots, \mathcal{E}_{pk}(1))$ with $\mathcal{E}_{pk}(1)$.

Note that $\mathcal{E}_{pk}(1)$ is the encryption of 1 with the public key pk and different randomness are chosen in $\mathcal{E}(1)$ for different words and bits.

For a word w_i in K_q, $t_{iq} = (b_1 b_2 \dots b_d)_b$, we construct $\hat{w}_{iq} = (\mathcal{E}_{pk}(b_1), \mathcal{E}_{pk}(b_2), \dots, \mathcal{E}_{pk}(b_d))$ by replacing $\mathcal{E}_{pk}(b_j)$ with $\mathcal{E}_{pk}(1)$ when $b_j = 1$ and with $\mathcal{E}(0) + e_b$ when $b_j = 0$.

Now we give the filter program F to the adversary $\mathcal{A}$, who then returns a guess q'. With probability 1/2, e_b is the encryption of 1, and hence $\hat{D}$ are the encryption of all $2^b - 1$ and no documents can meet the search criterion, and in this event $\mathcal{A}$'s guess is independent of q, and hence the probability $q' = q$ is 1/2. However, with probability 1/2, $e_b = \mathcal{E}(0)$, hence F is the filter program that searches for $\mathcal{Q}_{K_q}$, constructed exactly as in FilterGen algorithm, and hence in this case with probability $1/2 + \epsilon$, $\mathcal{A}$ will guess q correctly, as our behavior was indistinguishable for an actual challenger. We determine our guess b' as follows: If $\mathcal{A}$ guesses $q' = q$ correctly, then we will set $b' = 1$, and otherwise we will set $b' = 0$.

Putting it all together, we can now compute the probability that our guess is correct:

$$\Pr(b' = b) = \frac{1}{2}\left(\frac{1}{2}\right) + \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}.$$

Therefore, we have obtained a nonnegligible advantage in the semantic security game for the underlying fully homomorphic encryption scheme, a contradiction to our assumption. Thus, our protocol is semantically secure according to Definition 5. □

10 Performance Analysis

We have presented two basic constructions for threshold query based on keyword frequency. They are disjunctive and conjunctive.

In our disjunctive construction (Section 5), the client can pregenerates the public/private key pair. In addition, the client needs to encrypt the frequency of each classified keyword in the phase of the filter program generation and to decrypt the buffer $\mathbb{B}$ to retrieve the matching documents after the buffer returns. If we do not consider the key generation, the total computation complexity of the client is $O(d|D|)$ encryptions to generate the program F and $O(m\ell)$ decryptions to retrieve the matching documents from the buffer, where $|D|$ is the number of words in the dictionary D, 2^d is the maximal number of words contained in each document, ℓ is the number of bits of each document, and m is the maximal number of matching documents in the buffer.

After receiving the filter program F, the server processes each document M_i in three steps. We assume $\mu = |M_i \cap D|$. At first, the server needs to compute $\mathcal{E}_{pk}(c_0)$. The computation complexity of the first step is $O(\mu d)$ encryptions to encrypt μ frequencies with d bits, $O(\mu)$ homomorphic additions of integers with d bits, $O(\mu)$ homomorphic multiplications of bits, and $O(\mu)$ homomorphic additions of bits (please refer to (1)). Then, the server needs to add M_i into the buffer $\mathbb{B}$ if M_i is a matching document or add 0 into the buffer otherwise. The computation complexity of the second step is $O(m\ell^2)$ homomorphic multiplications of bits and $O(m\ell^2)$ homomorphic addition of bits. At last, the server needs to update the buffer base $\mathbb{G}$. The computation complexity of the third step is $O(m)$ homomorphic multiplications of bits and $O(1)$ homomorphic addition of integers with m bits.

TABLE 2
Performance Comparison

Protocols	Comp. Complexity (Client)	Comp. Complexity (Sever, M_i)	Comm. Complexity
Disjunctive	O($d\|D\|$) enc. +O($m\ell$) dec.	O(μd) enc. +O(μ) ADD. +O($m\ell^2 + \mu$) multi. +O($m\ell^2 + \mu$) add.	O($d\|D\|\|C\|$) +O($m\ell$)
Conjunctive	Same as Disjunctive	Disjunctive +O($\|D\|$) ADD.	Same as Disjunctive
Disjunctive Complement	Disjunctive +O($\|D\|$) enc.	Same as Disjunctive	Disjunctive +O($\|D\|\|C\|$)
Conjunctive Complement	Disjunctive +O($\|D\|$) enc.	Same as Conjunctive	Disjunctive +O($\|D\|\|C\|$)

The communication complexity of our disjunctive construction is $O(d|D\|C|)$ bits for the query and $O(m\ell|C|)$ bits for response, where $|C|$ is the size of the ciphertext.

Unlike our disjunctive construction, our conjunctive construction (Section 6) needs to compute $\mathcal{E}_{pk}(\gamma_0)$ and then $\mathcal{E}_{pk}(c_0)$. The computation complexity for the server to compute $\mathcal{E}_{pk}(\gamma_0)$ is $O(|D|)$ homomorphic additions of integers. Although the two constructions computes $\mathcal{E}_{pk}(c_0)$ with two different equations (please refer to (1) and (2)), their complexities for this computation are almost the same.

Our disjunctive complement construction (Section 7) is different from our disjunctive construction in two ways. The query contains an extra array of ciphertexts to indicate the complement positions in private and the server computes $\mathcal{E}_{pk}(c_0)$ with (3), which is different from (1). The differences do not affect the computation complexity of the server, but the computation complexity of the client is increased by $O(|D|)$ encryptions of bits and the communication complexity is increased by $O(|D\|C|)$ bits on the basis of the performance of our disjunctive construction.

Similarly, our conjunctive complement construction (Section 8) is different from our conjunctive construction in two ways. The differences do not change the computation complexity of the server, but the computation complexity of the client is increased by $O(|D|)$ encryptions of bits and the communication complexity is increased by $O(|D\|C|)$ bits on the basis of our conjunctive complement construction.

The performance of our generic construction (Section 9) depends on the performance of the underlying basic constructions.

The performance comparison of our threshold query protocols can be summarized in Table 2, where enc. and dec. stand for encryption and decryption of bit, add. and multi. denote the homomorphic addition and multiplication of bits, and ADD. represents the homomorphic addition of integers.

11 Conclusion and Discussion

On the basis of the state of the art fully homomorphic encryption techniques, we have presented constructions

for disjunctive, conjunctive, and complement threshold queries based on keyword frequency and then a construction for a generic threshold query based on keyword frequency. Our protocols are semantically secure as long as the underlying fully homomorphic encryption scheme is semantically secure.

Our construction for disjunctive threshold query is able to search for documents containing at least one of a set of keywords as [17], [18], [1], [2] by letting the threshold $t_i = 1$ for keyword $k_i \in K$. Our construction for generic threshold query can search for documents M such that $(M \cap K_1 \neq \emptyset) \wedge (M \cap K_1 \neq \emptyset)$ as [17], [18] by letting $\mathcal{Q}_{K_1}^{(1)}$ and $\mathcal{Q}_{K_2}^{(2)}$ be two disjunctive threshold queries with the threshold $t_i = 1$ for keyword $k_i \in K$ and $\Phi(\mathcal{Q}_{K_1}^{(1)}, \mathcal{Q}_{K_2}^{(2)}) = \mathcal{Q}_{K_1}^{(1)} \wedge \mathcal{Q}_{K_2}^{(2)}$. Therefore, their solutions are special cases of ours.

To improve the performance of our constructions, we can compress or postprocess the ciphertext of a bit in the final stage of filter program execution as [7]. In this case, the ciphertext of a bit can have the same size as an RSA modulus asymptotically.

Theoretically, any search criteria can be constructed with fully homomorphic encryption scheme in private searching on streaming data. Even if so, different queries will need different constructions. As long as the underlying fully homomorphic encryption scheme is practical, our protocols will be practical. So far, fully homomorphic encryption schemes are impractical for many applications according to [13], because ciphertext size and computation time increase sharply as one increases the security level. Recently, many research efforts have been devoted to construct efficient fully homomorphic encryption schemes, such as the ones by [23], [4], [5]. We believe that our protocols for private threshold queries based on keyword frequency will be made practical with the performance improvement of fully homomorphic encryption techniques in the future.

Privacy is gaining increasingly higher attention, and future computing paradigms, for example, cloud computing, will only become viable if privacy of users is thoroughly protected. For example, Google Alerts is a service offered by Google that notifies its users by e-mail, or as a feed, about the latest Web and news pages of their choice. As in the case of the AOL search data leak, it is not hard to imagine queries which could be privacy sensitive. With our private searching solutions, it is possible for a user to make a filtering program according to the frequencies of some classified keywords and submit it to Google, which executes the program on all latest Web and news pages. The program can notify to the user its discovery according to the search criteria specified by the user. While the program is executed by Google, the search criteria of the user can be kept confidential to Google.

Acknowledgments

This work was supported by the ARC Discovery Project (DP0988411) "Private Data Warehouse Query" and the US National Science Foundation Award (1016722) "TC: Small: Collaborative: Protocols for Privacy-Preserving Scalable Record Matching and Ontology Alignment." The authors would like to thank blind reviewers for their constructive comments, which helped them to improve this paper.

References

[1] J. Bethencourt, D. Song, and B. Water, "New Construction and Practical Applications for Private Streaming Searching," *Proc. IEEE Symp. Security and Privacy,* 2006.

[2] J. Bethencourt, D. Song, and B. Water, "New Techniques for Private Stream Searching," *ACM Trans. Information and System Security,* vol. 12, no. 3, pp. 1-32, 2009.

[3] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2-DNF Formulas on Ciphertext," *Proc. Second Int'l Conf. Theory of Cryptography,* pp. 325-341, 2005.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping," http://eprint.iacr.org/2011/277, 2011.

[5] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," http://eprint.iacr.org/2011/344, 2011.

[6] I. Damgard and M. Jurik, "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System," *Proc. Fourth Int'l Workshop Practice and Theory in Public Key Cryptography (PKC '01),* pp. 119-136, 2001.

[7] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," *Proc. Advances in Cryptology (EUROCRYPT '10),* pp. 24-43, 2010.

[8] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans. Information Theory,* vol. 31, no. 4, pp. 469-472, July 1985.

[9] C. Gentry, "Fully Homomorphic Encryption Scheme," PhD thesis, Stanford Univ., http://crypto.stanford.edu/craig, 2009.

[10] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," *Proc. 41st Ann. ACM Symp. Theory of Computing (STOC '09),* pp. 169-178, 2009.

[11] C. Gentry, "Computing Arbitrary Functions of Encrypted Data," *Comm. ACM,* vol. 53, no. 3, pp. 97-105, 2010.

[12] C. Gentry, "Toward Basing Fully Homomorphic Encryption on Worst-Case Hardness," *Proc. Advances in Cryptology (CRYPTO '10),* pp. 116-137, 2010.

[13] C. Gentry and S. Halevi, "Implementing Gentry's Fully-Homomorphic Encryption Scheme," *Proc. 30th Ann. Int'l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT '11),* pp. 129-148, 2011.

[14] D. Harris, D.M. Harris, and S.L. Harris, *Digital Design and Computer Architecture.* Morgan Kaufmann, 2007.

[15] D.J. Lilja and S.S. Sapatnekar, *Designing Digital Computer Systems with Verilog.* Cambridge Univ. Press, 2005.

[16] S. Ling and C.P. Xing, *Coding Theory: A First Course.* Cambridge Press, 2004.

[17] R. Ostrovsky and W. Skeith, "Private Searching on Streaming Data," *Proc. Advances in Cryptology (CRYPTO '05),* pp. 223-240, 2005.

[18] R. Ostrovsky and W. Skeith, "Private Searching on Streaming Data," *J. Cryptology,* vol. 20, no. 4, pp. 397-430, 2007.

[19] R. Ostrovsky and W. Skeith, "Algebraic Lower Bounds for Computing on Encrypted Data," *Proc. Electronic Colloquium on Computational Complexity (ECCC '07),* 2007.

[20] P. Paillier, "Public Key Cryptosystems Based on Composite Degree Residue Classes," *Proc. 17th Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT '99),* pp. 223-238, 1999.

[21] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs,* second ed. Oxford Univ. Press, 2010.

[22] N. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," *Proc. 13th Int'l Conf. Practice and Theory in Public Key Cryptography (PKC '10),* pp. 420-443, 2010.

[23] D. Stehle and R. Steinfeld, "Faster Fully Homomorphic Encryption," *Proc. Advances in Cryptology (ASIACRYPT '10),* pp. 377-394, 2010.

[24] J.F. Wakerly, *Digital Design Principles and Practices,* third ed. Prentice Hall, 2000.

[25] X. Yi and C.P. Xing, "Private (t, n) Threshold Searching on Streaming Data," *Proc. Int'l Conf. Social Computing Privacy, Security, Risk and Trust (PASSAT '12),* pp. 676-683, 2012.

Xun Yi is a professor at the College of Engineering and Science, Victoria University, Australia. His research interests include applied cryptography, computer and network security, mobile and wireless communication security, and privacy-preserving data mining. He has published more than 100 research papers in conference proceedings and international journals, such as the *IEEE Transactions on Knowledge and Data Engineering*, the *IEEE Transactions on Wireless Communication*, the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Transactions on Circuit and Systems*, the *IEEE Transactions on Technologies*, the *IEEE Communication Letters*, and the *IEEE Electronic Letters*. He has undertaken program committee members for more than 20 international conferences. Recently, he has led a few Australia Research Council Discovery Projects.

Elisa Bertino is currently a professor in the Computer Science Department, Purdue University, and serves as research director of CERIAS and director of the Cyber Center, Purdue University. Her main research interests include security, privacy, digital identity management systems, database systems, distributed systems, and multimedia systems. She received the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems and database security and advanced data management systems and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems. She is a fellow of both the IEEE and the ACM.

Jaideep Vaidya is an associate professor in the Management Science and Information Systems Department, Rutgers University. His primary research interests include at the intersection of privacy, security, data analysis, and data management. As such, he is very interested in the field of secure information sharing and its various applications, as also the application of secure computation technologies to business processes such as supply chain management and optimization. He is also interested in security and privacy issues raised by data mining, and the use of data mining techniques to enhance security, such as in role engineering.

Chaoping Xing is a professor at the School of Physical and Mathematical Science, Nanyang Technological University, Singapore. He is the deputy head (research) of the Division of Mathematical Sciences, Nanyang Technological University. His research interests include algebraic curves over finite fields, application of algebraic geometry, and number theory to block coding, quantum coding, space-time coding, cryptography, quasi-Monte Carlo methods, and lattice packings.