Private Searching for Single and Conjunctive Keywords on Streaming Data

Xun Yi
Victoria University
Melbourne, VIC 8001
Australia
Xun.Yi@vu.edu.au

Elisa Bertino
Purdue University
West Lafayette, IN 47907
USA
Bertino@cs.purdue.edu

ABSTRACT

Private searching on streaming data allows a user to collect potentially useful information from huge streaming sources of data without revealing his or her searching criteria. This technique can be used for airports, without knowing a classified "possible terrorists" list, to find if any of hundreds of passenger lists has a name from the "possible terrorists" list and if so his/hers itinerary. Current solutions for private searching on streaming data only support searching for "OR" of keywords or "AND" of two sets of keywords. In this paper, we extend the types of private queries to support searching on streaming data for an "OR" of a set of both single and conjunctive keywords, such as $S_1 \vee S_2 \vee \ldots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, where $S_1, \ldots, S_{n_1}$ are single keywords and $(A_1, B_1), \ldots, (A_{n_2}, A_{n_2})$ are unordered conjunctive keywords. Our protocol is built on Boneh et al.'s result for the evaluation of 2-DNF formulas on ciphertexts. The size of our encrypted dictionary is $O(|D|)$ only, which is much less than $|D|^2$, the size of the encrypted dictionary if conjunctive keywords (A_i, B_i) $(i = 1,2, \ldots, k)$ is treated as single keyword, where we assume $A_i, B_i \in D$ $(i = 1,2, \ldots, k)$.

1. INTRODUCTION

Private searching on streaming data has been motivated by a crucial task for the intelligence community, which is to collect potentially useful information from huge streaming data [17, 18]. For example, in airports one has to find if any of hundreds of passenger lists has a name from the "possible terrorists" list and if so his/hers itinerary. Usually, data sources are huge, and it is impractical to keep all the data for such an analysis. A different more practical approach is continuously performing on-line filtering of data streaming from multiple sources, one document/message/packet at the time. Such an approach allows one to immediately discard most of the data, while retaining only a small fraction of potentially useful data.

In almost all cases, data is categorized as potentially useful based on certain searching criteria. Keeping these criteria classified is clearly crucial, as adversaries (like terrorists) could easily prevent their data from being collected by simply making sure that their data does not match the data search criteria. A naïve solution to this problem is to collect all streaming data in a secure environment, and then filter the information according to classified search criteria. This approach adds considerable cost in terms of communication and may result in delay in the delivery of information or even in the loss of data, if the transfer to the secure environment is interrupted. Furthermore, it requires considerable cost of storage to hold this (un-filtered) data in case the transfer to the classified setting is delayed.

Obviously, a far more preferable solution is to filter all these data-streams directly at their sources. A crucial issue is how we can do this while at the same time keeping secret the searching criteria, even in the case in which the system managing the data-streams is compromised by attackers.

The first solution to this problem, referred to as *private searching on streaming data*, was proposed by Ostrovsky and Skeith [17]. Their protocol is based on the homomorphism of the cryptosystems [19, 10], which allows one to compute $E_{pk}(m_1 + m_2)$, given $E_{pk}(m_1)$ and $E_{pk}(m_2)$, where $E_{pk}(m)$ is an encryption of plaintext *m* with public key pk.

In the Ostrovsky-Skeith protocol, the public dictionary $D = \{w_1, w_2, \ldots, w_{|D|}\}$ of keywords is fixed. To construct a program for the disjunction of some classified keywords $K = \{k_1, k_2, \ldots, k_{|K|}\} \subseteq D$, the user generates a pair of public and private keys (pk, sk), and produces an array of ciphertexts $C = \{c_1, c_2, \ldots, c_{|D|}\}$, one for each keyword $w_i \in D$, such that if $w_i \in K$, then $c_i = E_{pk}(1)$; otherwise, $c_i = E_{pk}(0)$. In addition, the user constructs a buffer B with γm boxes, each of them is initialized with two ciphertexts $(E_{pk}(0), E_{pk}(0))$, where m is the upper bound on the number of matching documents the buffer can accommodate. The array of ciphertexts C and the buffer B are deployed in a server monitoring the streaming data.

To perform private searching, the data is segmented into streaming files $F = \{f_1, f_2, \ldots\}$, each of which is composed of a number of words, and filtered one at a time. To process a file f_i, the server computes a product of ciphertexts corresponding to the keywords found in the file, i.e., $d_i = \prod_{w_j \in f_i} c_j = E_{pk}(|f_i \cap K|)$, and $e_i = d_i^{f_i} = E_{pk}(f_i \cdot |f_i \cap K|)$, due to the homomorphic property of the public key cryptosystem. Then the server copies (d_i, e_i) into γ randomly chosen boxes in the buffer B by multiplying corresponding ciphertexts. If $f_i \cap K = \emptyset$, this step will add an encryption of 0 to each box, having no effect on the corresponding plaintext. If $f_i \cap K \neq \emptyset$, then the matching file can be retrieved by computing $f_i = \frac{D_{sk}(e_i)}{D_{sk}(d_i)}$, where D_{sk} stands for decryption with the private key sk.

If two different matching files are ever added to the same buffer box, a collision will result and both copies will be lost. To avoid the loss of matching files, this protocol make the buffer B sufficiently large so that each matching file can survive in at least one buffer box. After the content of buffer B is returned to the user, the user is able to retrieve all matching files.

Using results by Boneh, Goh, and Nissim [4], Ostrovsky and Skeith [17, 18] extended the type of queries from an "OR" of keywords to queries with an "AND" of two sets of keywords without increasing the program size.

Their basic idea for searching all documents M such that $(M \cap K_1 \neq \phi) \wedge (M \cap K_2 \neq \phi)$, where K_1, K_2 are two sets of "keywords", is to construct two arrays of ciphertexts $C_1 = \{c_1^1, c_2^1, \dots, c_{|D|}^1\}$, where c_i^1 is the encryption of 1 if $w_i \in K_1$ and otherwise is the encryption of 0, and $C_2 = \{c_1^2, c_2^2, \dots, c_{|D|}^2\}$, where c_i^2 is the encryption of 1 if $w_i \in K_2$ and otherwise is the encryption of 0. To process a document M, the program computes $v_1 = \prod_{w_j \in M} c_j^1 = E_{pk}(|M \cap K_1|)$, $v_2 = \prod_{w_j \in M} c_j^2 = E_{pk}(|M \cap K_2|)$ and then $v = e(v_1, v_2)$, where e is a bilinear map. If $(M \cap K_1 \neq \phi) \wedge (M \cap K_2 \neq \phi)$, then v is an encryption of 1. Otherwise, v is an encryption of 0.

In 2006, Bethencourt, Song and Waters proposed a different method for retrieving matching files from the buffer [1, 2]. Like the approach by Ostrovsky and Skeith, they use an encrypted dictionary, and no-matching files have no effect on the contents of the buffer. However, rather than using one large buffer and attempting to avoid collisions, they employ three buffers – the *data buffer* F, *c-buffer* C, and the *matching indices buffer* I, each of them has m boxes, and the matching files are then retrieved by solving a linear system.

The Bethencourt-Song-Waters protocol is able to process t files $\{f_1, f_2, \dots, f_t\}$ of streaming data. For each file f_i, the server computes d_i (e_i) as the Ostrovsky-Skeith protocol, and copies d_i (e_i) randomly over approximately half of the locations across the buffer C (F). A pseudorandom function $g(i, j)$ is used to determine with probability ½ whether d_i (e_i) is copied into a given location j. In addition, the server further copies d_i into a fixed number of locations in the matching-indices buffer. This is done by using essentially the standard procedure for updating a Bloom filter. Specifically, they use k hash functions $h_1, h_2, \dots h_k$ to select the k locations. The locations of the matching-indices buffer I that d_i is multiplied into are taken to be $h_1(i), h_2(i), \dots h_k(i)$.

After the contents of all three buffers are returned, the user decrypts all buffers at first. For each of the indices $i \in \{1,2,..,t\}$, the user computes $h_1(i), h_2(i), \dots h_k(i)$ and checks the corresponding locations in the decrypted matching-indices buffer. If all locations are non-zero, i is added into the list of potential matching indices. Given the potential matching indices $\{\alpha_1, \alpha_2, \dots, \alpha_\ell\}$, the user next determines the values of $\{n_{\alpha_1}, n_{\alpha_2}, \dots, n_{\alpha_\ell}\}$, where $n_{\alpha_i} = |f_{\alpha_\ell} \cap K|$, by solving a system of linear equations constructed with the decrypted *c-buffer*. As last step, the user determines the content of the matching files $f_{\alpha_1}, f_{\alpha_2}, \dots, f_{\alpha_\ell}$ by solving another system of linear equations constructed with the decrypted *data buffer*.

Our Contribution: Current solutions for private searching on streaming data can only search for an "OR" of keywords [17, 18, 1, 2] or for an "AND" of two sets of keywords from streaming data [17, 18]. Without loss of generality, these queries can be expressed as either $k_1 \vee k_2 \vee \dots \vee k_{|K|}$ or $(k_1 \vee k_2 \vee \dots \vee k_\lambda) \wedge (k_{\lambda+1} \vee k_{\lambda+2} \vee \dots \vee k_{|K|})$. The restricted form of queries supported by those protocols limits the applications of private searching on streaming data in practice.

For example, suppose we wish to find if any list, among hundreds of passenger lists, has a name from a list of "possible terrorists" $L = \{(F_1, S_1), (F_2, S_2), \dots, (F_n, S_n)\}$ where (F_i, S_i) denotes the first name and the last name of a terrorist. If we perform a query of the form $(F_1||S_1) \vee (F_2||S_2) \vee \dots \vee (F_n||S_n)$, the dictionary D for private searching needs to be $D \times D$, where D is the set of all possible names. Such dictionary is too large for practical use. If we perform a query of the form $(F_1 \vee F_2 \vee \dots \vee F_n) \wedge (S_1 \vee S_2 \vee \dots \vee S_n)$, the dictionary D needs to be $2D$ only, but some innocent passengers, e.g., (F_1, S_2), will incorrectly appear in the search results.

In this paper, we propose a protocol to perform a private query of the form $S_1 \vee S_2 \vee \dots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_{n_2} \wedge B_{n_2})$ where $S_1, \dots, S_{n_1}$ are single keywords and $(A_1, B_1), \dots, (A_{n_2}, A_{n_2})$ are unordered conjunctive keywords. Our algorithm is built on Boneh et al.'s result concerning the evaluation of 2-DNF formulas on ciphertexts. The size of our encrypted dictionary is $O(|D|)$ only, which is much less than $|D|^2$, the size of the encrypted dictionary if conjunctive keywords (A_i, B_i) $(i = 1,2, \dots, k)$ is treated as single keyword, where we assume $A_i, B_i \in D$ $(i = 1,2, \dots, k)$.

Following up the intuition of the Ostrovsky-Skeith protocol [17, 18], our basic idea is to create a program that conditionally and obviously performs encryptions of a document based on the matching of keyword criteria, and then writes these encryptions to random locations in a buffer, using homomorphic properties of the encryption scheme. By "conditionally", we mean that if a document matches the query, our private searching protocol will generate an encryption of the document itself. Otherwise, it will generate an encryption of the identity element. The key idea is that the encryption of the identity element that the protocol computes if the document does not match the secret criteria will be indistinguishable from the encryption of the matching document. Both matching and non-matching documents appear to be treated precisely in the same way. Any party which observes the execution is unable to learn if the search condition is satisfied, as the protocol is executed as a straight-line code (i.e., all branches that the protocol executes are independent of the search criteria), so that the conditions are never known unless the underlying encryption scheme is broken.

Like the Ostrovsky-Skeith protocol for a query with an "AND" of two sets of keywords [17, 18], our protocol is also based on the results of Boneh, Goh and Nissim [4]. Unlike their protocol, our protocol supports private searches for both single and conjunctive keywords.

2. PRELIMINARIES

In this section, we briefly review the results of Boneh, Goh and Nissim in evaluating 2-DNF formulas on ciphertext. [4].

2.1 Bilinear Group

We use the following notations:

1. G and G_1 are two (multiplicative) cyclic groups of finite order n.
2. g is a generator of G.

3. e is a bilinear map $e: G \rightarrow G_1$. In other words, for all $u, v \in G$ and $a, b \in \mathbb{Z}$, we have $e(u^a, v^b) = e(u, v)^{ab}$. We also requires that $e(g, g)$ is a generator of G_1.

We say that G is a bilinear group if a group G_1 and a bilinear map as above exist.

2.2 A Homomorphic Public Key System

The system resembles the Paillier [19] and the Okomoto-Uchiyama [16] encryption schemes. The three algorithms making up the system is described as follows:

Key Generation $KeyGen(\tau)$: Given a security parameter $\tau \in \mathbb{Z}^+$, run $\mathcal{G}(\tau)$ to obtain a tuple (q_1, q_2, G, G_1, e). Let $N = q_1 q_2$. Pick two random generators $g, u \overset{R}{\leftarrow} G$ and set $h = u^{q_2}$. Then h is a random generator of the subgroup of G of order q_1. The public key is $PK = (N, G, G_1, e, g, h)$. The private key $SK = q_1$.

Encryption $Encrypt(PK, m)$: Assume the message space consists of integers in the set $\{0,1, \dots, T\}$ with $T < q_2$. We encrypt bits in which case $T = 1$. To encrypt a message m using the public key PK, pick a random $r \overset{R}{\leftarrow} \{0,1, \dots N\}$ and compute

$$C = g^m h^r \in G$$

Output C as the ciphertext.

Decryption $Decrypt(SK, C)$: To decrypt a ciphertext C using the private key $SK = q_1$, observe that

$$C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$$

Let $\tilde{g} = g^{q_1}$. To recover m, it suffices to compute the discrete log of C^{q_1} base $\tilde{g}$. Since $0 \le m \le T$, this takes expected time $O(\sqrt{T})$ using Pollard's lambda method [14].

Note that decryption in this system takes polynomial time in the size of the message space T. Therefore, the system can only be used encrypt short messages.

2.3 Homomorphic Properties

The system is clearly additively homomorphic. Let $PK = (N, G, G_1, e, g, h)$ be a public key. Given encryptions $C_1, C_2 \in G_1$ of messages $m_1, m_2 \in \{0,1, \dots, T\}$ respectively, anyone can create a uniformly distributed encryption of $m_1 + m_2 \bmod N$ by computing the product $C = C_1 C_2 h^r$ for a random r in $\{0,1, \dots, N-1\}$.

More importantly, anyone can multiply two encrypted messages once using the bilinear map. Let $g_1 = e(g, g)$ and $h_1 = e(g, h)$, then g_1 is of order n and h_1 is of order q_1. There is some (unknown) $\alpha \in \mathbb{Z}$ such that $h = g^{\alpha q_2}$. Suppose that we are given two ciphertexts $C_1 = g^{m_1} h^{r_1} \in G$ and $C_2 = g^{m_2} h^{r_2} \in G$. To build an encryption of the product $m_1 m_2 \bmod N$, (1) pick a random $r \in \mathbb{Z}_n$, and (2) let $C = e(C_1, C_2) h_1^r \in G_1$. Then

$$\begin{aligned} C &= e(C_1, C_2) h_1^r \\ &= e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r \\ &= e(g^{m_1 + \alpha q_2 r_1}, g^{m_1 + \alpha q_2 r_2}) h_1^r \\ &= e(g, g)^{(m_1 + \alpha q_2 r_1)(m_2 + \alpha q_2 r_2)} h_1^r \\ &= e(g, g)^{m_1 m_2 + \alpha q_2 (m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2)} h_1^r \\ &= e(g, g)^{m_1 m_2} h_1^{(r + m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2)} \end{aligned}$$

where $r + m_1 r_2 + m_2 r_1 + \alpha q_2 r_1 r_2$ is distributed uniformly in $\mathbb{Z}_n$ as required. Thus C is a uniformly distributed encryption of $m_1 m_2 \bmod n$, but in G_1 rather than G. We note that the system is still additively homomorphic in G_1.

3. OUR PROTOCOLS

3.1 System Model

We consider a system model as shown in Fig. 1, where a user wants to retrieve the documents (or messages) that include a set of unordered conjunctive keywords from streaming data sources.

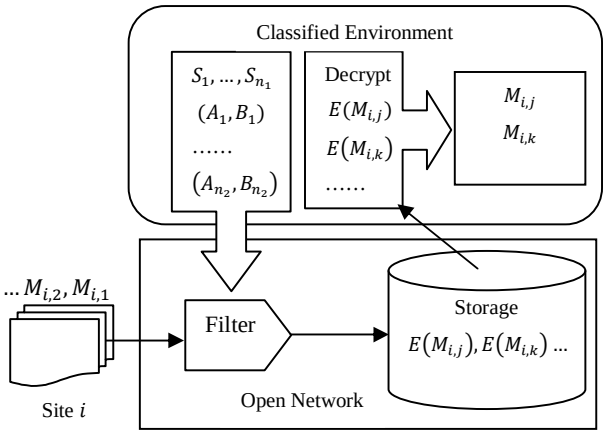

Fig. 1 System Model

First of all, the user prepares a filtering program with the set of single keywords, $S_1, \dots, S_{n_1}$ and unordered conjunctive keywords, $(A_1, B_1), \dots, (A_{n_2}, A_{n_2})$, and deploys the program at each public data source. The program processes a document at a time, and only encrypts and stores in its storage the document which satisfies the condition. $S_1 \vee S_2 \vee \dots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_{n_2} \wedge B_{n_2})$. After a certain time period, the program sends its storage content back to the user. Finally, the user decrypts the contents of the storage and obtains the matching documents.

The program is executed at a public data source and may fall into an adversary's hand. If this happens, we require that the adversary cannot obtain any classified keywords from the program.

3.2 Private Searching for Conjunctive Keywords with Space Efficiency

We first formally define our private searching protocol for conjunctive keywords. The protocol is efficient in terms of the size of the encrypted dictionary. It is composed of the algorithms: the key generation algorithm (**Key-Gen**), the filter generation

algorithm (**Filter-Gen**), the buffer decryption algorithm (**Buffer-Decrypt**) defined as follows:

Key-Gen(k)

It executes the key generation algorithm of the Boneh, Goh and Nissim system to produce the public key $PK = (N, G, G_1, e, g, h)$, where g is a generator, $N = q_1 q_2$, and h is a random element of order q_1. The private key is $SK = q_1$. We make the additional assumption that $|D| < q_2$.

Filter-Gen(D, Q, PK, m, γ)

This algorithm constructs and outputs a private filter program F for the query $Q = (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_n \wedge B_n)$, which searches for all documents M satisfying Q. Assume that the public dictionary is $D = \{w_1, w_2, \ldots, w_{|D|}\}$. F contains the following data:

- A buffer $B(\gamma)$ of size $2\gamma m$, indexed by blocks with the size of an element of G_1 times the document size, with every position initialized to the encryption of the identity element of G_1, where m is the upper bound on the number of matching documents we wish to save in the buffer $B(\gamma)$.
- n arrays of ciphertexts $C_j = \{c_1^j, c_2^j, \ldots, c_{|D|}^j\}$ $(j = 1, 2, \ldots, n)$, each corresponding to one conjunctive keyword (A_j, B_j), where c_i^j is the encryption of 1 if $w_i \in \{A_j, B_j\}$ and otherwise the encryption of 0. Each array of ciphertext contains two encryptions of 1 and $|D| - 2$ encryptions of 0.

F then proceeds with the following steps upon receiving an input document M.

1. It constructs a set of temporary collections $\tilde{C_j} = \{c_i^j \in C_j | w_i \in M \cap D\}$ for $j = 1, 2, \ldots, n$.
2. To process a word w_λ in $M \cap D$, it computes

$$v_\lambda = \prod_{j=1}^{n} e(c_\lambda^j, \prod_{i > \lambda, c_i^j \in \tilde{C_j}} c_i^j) \in G_1$$

   where e is a bilinear map. If there exists a word $w_i \in M \cap D$ and $i > \lambda$ such that $(w_\lambda \wedge w_i) \in (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots, \vee (A_n \wedge B_n)$ then v_λ is an encryption of 1 in G_1. Otherwise, v_λ is an encryption of 0.
3. It computes $v = \prod_{w_\lambda \in M \cap D} v_\lambda$. If the document M satisfies the condition $(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_n \wedge B_n)$, then v is an encryption of a positive integer in G_1. Otherwise, v is an encryption of 0.
4. It performs a bitwise encryption of M using encryption of 0 in G_1 for 0's and using v to encrypt 1's to create a vector of G_1 elements.
5. It chooses γ random locations in B, takes the encryption of Step 4, and position-wise multiplies these two vectors storing the result back in B at the same location.

Buffer-Decrypt(B, SK)

It decrypts B one block at a time using the decryption algorithm of the BGN system, interpreting the non-identity elements of G_1 as 1's and 0's as 0, outputting the non-zero, valid documents.

Correctness of the Private Filter

We show the correctness of our protocol with the following two facts:

- In our protocol, non-matching documents are stored with negligible probability. In fact, they are stored with probability 0 since clearly (i) if w_λ in Step 2 does not match with any classified keywords $A_1, B_1, \ldots, A_k, A_k$, then c_λ^j is the encryption of 0 and thus v_λ is the encryption of 0; (ii) if w_λ does match with a classified keyword, e.g., A_1, but $B_1 \notin M - \{w_\lambda\}$, then v_λ is the encryption of 0 as well. So, the buffer contents will be unaffected by the program executing on input a non-matching document M.

- In our protocol, all matching documents are saved with overwhelming probability. Clearly, if a document M satisfies $(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_k \wedge B_k)$, e.g., M satisfies $(A_1 \wedge B_1)$, when the program processes $w_\lambda = A_1$ and $B_1 \in M - \{w_\lambda\}$, then v_λ is an encryption of 1. Therefore, v is an encryption of a positive integer. Following up the "colour-survival" game [17, 18] for placing the matching document in the buffer, all documents will be saved with overwhelming probability in γ.

Remark: In [18], the color-survival game is introduced and a Lemma is proved as follows.

Color-survival game: Let $m, \gamma \in Z^+$, and suppose we have m different colors, call them $\{color_i\}_{i=1}^{m}$and γ balls of each color. We throw the γm balls uniformly at random into $2\gamma m$ bins, call them $\{bin_i\}_{i=1}^{2\gamma m}$. We say that a ball "survives" in bin_j , if no other ball (of any color) lands in bin_j. We say that $color_i$ "survives" if at least one ball of color colori survives. We say that the game succeeds if all m colors survive, otherwise we say that it fails.

Lemma. The probability that the color-survival game fails is negligible in γ.

3.3 Private Searching for Single and Conjunctive Keywords with Space Efficiency

Based on our protocol for conjunctive keywords, we now formally present our private searching protocol for both single and conjunctive keywords, which is also composed of three algorithms: the key generation algorithm (**Key-Gen**), the filter generation algorithm (**Filter-Gen**), the buffer decryption algorithm (**Buffer-Decrypt**) as follows:

Key-Gen(k)

It is the same as the algorithm (**Key-Gen**) described in section 3.2.

Filter-Gen(D, Q, PK, m, γ)

This algorithm constructs and outputs a private filter program F for the query $Q = S_1 \vee S_2 \vee \ldots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, which searches for all documents M satisfying Q.

Assume that the public dictionary is $D = \{w_1, w_2, \dots, w_{|D|}\}$. F contains the following data:

- A buffer $B(\gamma)$ of size $2\gamma m$, indexed by blocks with the size of an element of G_1 times the document size, with every position initialized to the encryption of the identity element of G_1.
- n_2 arrays of ciphertexts $C_j = \{c_1^j, c_2^j, \dots, c_{|D|}^j\}$ $(j = 1, 2, \dots, n_2)$, each corresponding to one conjunctive keyword (A_j, B_j), where c_i^j is the encryption of 1 if $w_i \in \{A_j, B_j\}$ and otherwise the encryption of 0.
- One array of ciphertexts $C_j = \{c_1^j, c_2^j, \dots, c_{|D|}^j\}$ $(j = n_2 + 1)$, corresponding to single keywords $S_1, S_2, \dots, S_{n_1}$, where c_i^j is the encryption of 1 if $w_i \in \{S_1, S_2, \dots, S_{n_1}\}$ and otherwise the encryption of 0.
- $n_2 + 1$ ciphertexts d_j $(j = 1, 2, \dots, n_2 + 1)$, each corresponding to one array of ciphertexts C_j, where d_j is the encryption of 1 if $j = n_2 + 1$ and otherwise the encryption of 0.

F then proceeds with the following steps upon receiving an input document M.

1. It construct a set of temporary collections $\tilde{C_j} = \{c_i^j \in C_j | w_i \in M \cap D\}$ for $j = 1, 2, \dots, n_2 + 1$.
2. To process a word w_λ in $M \cap D$, it computes

$$v_\lambda = \prod_{j=1}^{n_2+1} e(c_\lambda^j, d_j \prod_{i>\lambda, c_i^j \in \tilde{C}_j} c_i^j) \in G_1$$

   where e is a bilinear map. If $w_\lambda \in S_1, \dots, S_{n_1}$ or there exists a word $w_i \in M \cap D$ and $i > \lambda$ such that $(w_\lambda \wedge w_i) \in \{(A_1 \wedge B_1), (A_2 \wedge B_2), \dots, (A_{n_2} \wedge B_{n_2})$ then v_λ is an encryption of 1 in G_1. Otherwise, v_λ is an encryption of 0.
3. It computes $v = \prod_{w_\lambda \in M \cap D} v_\lambda$. If the document M satisfies the condition $S_1 \vee S_2 \vee \dots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_{n_2} \wedge B_{n_2})$, then v is an encryption of positive integer in G_1. Otherwise, v is an encryption of 0.
4. It performs s bitwise encryption of M using encryption of 0 in G_1 for 0's and using v to encrypt 1's to create a vector of G_1 elements.
5. It chooses γ random locations in B, takes the encryption of Step 4, and position-wise multiplies these two vectors storing the result back in B at the same location.

Buffer-Decrypt(B, SK)

Same as described in section 3.2.

Correctness of Private Filter

Same as described in section 3.2.

3.4 Private Searching for Single and Conjunctive Keywords with Computation Efficiency

The protocols described in sections 3.2 and 3.3 are efficient in terms of the size of the encrypted dictionary. However, they require the computation of a large number of pairing. We now formally define our private searching protocol for both single and conjunctive keywords, which requires computing of lower number of pairings. The protocol is also composed of key generation algorithm (**Key-Gen**), filter generation algorithm (**Filter-Gen**), buffer decryption algorithm (**Buffer-Decrypt**) as follows:

Key-Gen(k)

It is the same as the algorithm (**Key-Gen**) described in section 3.2.

Filter-Gen(D, Q, PK, m, γ)

This algorithm constructs and outputs a private filter program F for the query $Q = S_1 \vee S_2 \vee \dots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_{n_2} \wedge B_{n_2})$, which searches for all documents M satisfying Q. Assume that the public dictionary is $D = \{w_1, w_2, \dots, w_{|D|}\}$. F contains the following data:

- A buffer $B(\gamma)$ of size $2\gamma m$, indexed by blocks with the size of an element of G_1 times the document size, with every position initialized to the encryption of the identity element of G_1.
- n_2 arrays of ciphertexts $C_{1,j} = \{c_1^{1,j}, c_2^{1,j}, \dots, c_{|D|}^{1,j}\}$ $(j = 1, 2, \dots, n_2)$, where $c_i^{1,j}$ is the encryption of 1 if $w_i = A_j$ and otherwise the encryption of 0. Each array of ciphertext contains one encryptions of 1 and $|D| - 1$ encryptions of 0.
- n_2 arrays of ciphertexts $C_{2,j} = \{c_1^{2,j}, c_2^{2,j}, \dots, c_{|D|}^{2,j}\}$ $(j = 1, 2, \dots, n_2)$, where $c_i^{2,j}$ is the encryption of 1 if $w_i = B_j$ and otherwise the encryption of 0. Each array of ciphertext contains one encryptions of 1 and $|D| - 1$ encryptions of 0.
- One array of ciphertexts $C_{1,j} = \{c_1^{1,j}, c_2^{1,j}, \dots, c_{|D|}^{1,j}\}$ $(j = n_2 + 1)$, corresponding to the single keywords $S_1, S_2, \dots, S_{n_1}$, where $c_i^{1,j}$ is the encryption of 1 if $w_i \in \{S_1, S_2, \dots, S_{n_1}\}$ and otherwise the encryption of 0.
- One array of ciphertexts $C_{2,j} = \{c_1^{2,j}, c_2^{2,j}, \dots, c_{|D|}^{2,j}\}$ $(j = n_2 + 1)$, where $c_i^{2,j}$ is the encryption of 0.
- $n_2 + 1$ ciphertexts d_j $(j = 1, 2, \dots, n_2 + 1)$, each corresponding to two arrays of ciphertexts $C_{1,j}$ and $C_{2,j}$, where d_j is the encryption of 1 if $j = n_2 + 1$ and otherwise the encryption of 0.

F then proceeds with the following steps upon receiving an input document M.

1. It constructs a set of temporary collections $\tilde{C}_{1,j} = \{c_i^{1,j} \in C_{1,j} | w_i \in M \cap D\}$ for $j = 1, 2, \dots, n_2 + 1$ and $\tilde{C}_{2,j} = \{c_i^{2,j} \in C_{2,j} | w_i \in M \cap D\}$ for $j = 1, 2, \dots, n_2 + 1$.
2. It computes

$$v = \prod_{j=1}^{n_2+1} e\left(\prod_{c_i^{1,j} \in \tilde{C}_{1,j}} c_i^{1,j}, d_j \prod_{c_i^{2,j} \in \tilde{C}_{2,j}} c_i^{2,j}\right) \in G_1$$

where e is a bilinear map. If the document M satisfies the condition $S_1 \vee S_2 \vee \ldots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, then v is an encryption of positive integer in G_1. Otherwise, v is an encryption of 0.

3. It bitwise encrypts M using encryption of 0 in G_1 for 0's and using v to encrypt 1's to create a vector of G_1 elements.
4. It chooses γ random locations in B, takes the encryption of Step 4, and position-wise multiplies these two vectors storing the result back in B at the same location.

Buffer-Decrypt(B, SK)

Same as described in section 3.2.

Correctness of Private Filter

Same as described in section 3.2.

Remark: Our protocol with computation efficiency can be modified to support more general queries. For example, $Q = S_1 \vee S_2 \vee [(A_{11} \vee A_{12}) \wedge (B_{11} \vee B_{12} \vee B_{13})] \vee (A_2 \wedge B_2)$. For this query, the protocol constructs the array of ciphertexts $C_{1,1} = \{c_1^{1,1}, c_2^{1,1}, \ldots, c_{|D|}^{1,1}\}$, where $c_i^{1,1}$ is the encryption of 1 if $w_i \in \{A_{11}, A_{12}\}$ and otherwise the encryption of 0; and the array of ciphertexts $C_{2,1} = \{c_1^{2,1}, c_2^{2,1}, \ldots, c_{|D|}^{2,1}\}$, where $c_i^{2,1}$ is the encryption of 1 if $w_i \in \{B_{11}, B_{12}, B_{13}\}$ and otherwise the encryption of 0.

4. SECURITY ANALYSIS

4.1 Security Model

A security model for private searching on streaming data has been built by Ostrovsky and Skeith in [17, 18] as follows.

We consider a universe of words $W = \{0,1\}^*$, and a dictionary $D \subset W$ with $|D| < \infty$. We think of a document as an ordered, finite sequence of words in W, however, it will often be convenient to look at the set of distinct words in a document. We define a set of keywords to be any subset $K \subset D$. Finally, we define a stream of documents S simply to be any sequence of documents.

We think of a query type, Q as a class of logic expression in $\wedge, \vee, \sim$ with a number of binary variables. Given a query type, one can input keywords $K \subset D$ where $K = \{k_i\}_{i=1}^{\alpha}$ and create a function, call $Q_K: S \to \{0,1\}$, that takes the documents, and returns 1 if and only if a document matches the criteria. $Q_K(M)$ is computed simply by evaluating Q on inputs of the form $k_i \in M$, We call $Q_K(M)$ a query over keyword K.

Definition 4.1 For a query Q_K on a set of keywords K, and for a document M, we say that M matches query Q_K if and only if $Q_K(M)$=1.

Definition 4.2 For a fixed query type Q, a private filter consists of the following three probabilistic polynomial time algorithms:

1. $\boldsymbol{KeyGen}(k)$: It takes a security parameter k and generate public key PK and private key SK.
2. $\boldsymbol{FilterGen}(D, Q_K, PK, m, \gamma)$: It takes a dictionary D, a query $Q_K \in Q$ for the set of keywords K, and generate a search program F. F searches any document stream S (processing one document at a time and updating a buffer B) and collects up to m documents that match Q_K in B, outputting an encrypted buffer B that contains the query results, where $|B| = O(\gamma)$ throughout the execution.
3. $\boldsymbol{FilterDecrypt}(B, SK)$: It decrypts an encrypted buffer B, produced by F as above, using the private key SK and produces output B^*, a collection of the matching documents from S.

Definition 4.3 (Correctness of a Private Filter) Let $F = \boldsymbol{FilterGen}(D, Q_K, PK, m, \gamma)$ and $(PK, SK) = \boldsymbol{KeyGen}(k)$, $B = F(S)$ and $B^* = \boldsymbol{FilterDecrypt}(B, SK)$. We say a private filter is correct if the following condition holds:

- If $|\{M \in S | Q_K(M) = 1\}| \leq m$, then

$$Pr[B^* = \{M \in S | Q_K(M) = 1\}] > 1 - neg(\gamma)$$

- If $|\{M \in S | Q_K(M) = 1\}| > m$, then

$$Pr[(B^* \subset \{M \in S | Q_K(M) = 1) \vee (B^* = \perp)\}] > 1 - neg(\gamma)$$

where $\perp$ is a special symbol denoting buffer overflow, and the probabilities are taken over all coin-tosses of F, $\boldsymbol{FilterGen}$ and $\boldsymbol{KeyGen}$.

Definition 4.4 (Privacy) Fix a dictionary D. Consider the following game between an adversary A, and a challenger C. The game consists of the following steps.

1. C first runs $\boldsymbol{KeyGen}(k)$ to obtain PK and SK and then sends PK to A.
2. A chooses two queries for two sets of keywords, Q_{0K_0}, Q_{1K_1}, with $K_0, K_1 \subset D$ and sends them to C.
3. C chooses a random bit $b \in \{0,1\}$ and executes $\boldsymbol{FilterGen}(D, Q_{bK_b}, PK, m, \gamma)$ to create F_b, the filtering program for the query Q_{bK_b}, and then sends F_b back to A.
4. $A(F_b)$ can experiment with the code of $F_b$in an arbitrary way, and finally outputs $b' \in \{0,1\}$.

The adversary wins the game if $b' = b$ and loses otherwise. We define the adversary A's advantage in this game to be

$$Adv_A(k) = \left|\Pr(b = b') - \frac{1}{2}\right|$$

We say that a private filter is semantically secure if for any adversary PPT A, we have that $Adv_A(k)$ is a negligible function, where the probability is taken over coin-tosses of the challenger and the adversary.

4.2 Security Analysis

Our protocol is based on the Boneh-Goh-Nissim public key system [4], which builds its security on a subgroup

indistinguishability assumption, related to the difficulty of computing discrete logs in the groups G, G_1.

Theorem 4.5 Assume the Boneh-Goh-Nissim public key system is semantically secure, then our private searching protocols for single and conjunctive keywords is semantically secure according to Definition 4.4.

Proof: Denote by $\mathcal{E}$ the encryption algorithm of the Boneh-Goh-Nissim public key system. Suppose that there exists an adversary A that can gain a non-negligible advantage ϵ in our semantic game from Definition 4.4. Then A could be used to gain an advantage in breaking the semantic security of the Boneh-Goh-Nissim public key system as follows:

At first, we initiate the semantic security game for the Boneh-Goh-Nissim public key system with a challenger C. C will send us with the public key $PK = (n, G, G_1, e, g, h)$, where $n = q_1 q_2$, g is a random generator of G, $h = u^{q_2}$, and e is bilinear map.

Next, we initiate the private filter semantic security game with an adversary A. A will give us two queries Q_0, Q_1 in Q for some sets of single and conjunctive keywords K_0, K_1, respectively.

Assume the private filters for Q_0, Q_1 are F_0, F_1, respectively, and the sequences of the Boolean plaintexts corresponding to the ciphertexts in F_0, F_1 are M_0, M_1, respectively. If the lengths of the two sequences are not equal, 0s are appended.

After sending M_0 and M_1 to the challenger C, C replies us with the encryptions of all Boolean plaintexts in one of these two sequences, denoted as $c_b = \mathcal{E}(M_b)$, where $b \in \{0,1\}$.

Now we give this private filter composed by c_b back to A. The private filter is equivalent to either F_0 or F_1. A returns a guess b'. We use A's guess as our guess. As our behavior is indistinguishable from an actual challenger, A will guess b correctly with probability ½+ ϵ, and hence we have obtained a non-negligible advantage in the semantic security game for the Boneh-Goh-Nissim public key system, a contradiction to our assumption. Therefore, our system is secure according to Definition 4.4.

5. PERFORMANCE ANALYSIS

Three protocols are described in section 3. The first protocol described in section 3.2, denoted as Protocol I, can be used to search conjunctive keywords only. The protocols described in sections 3.3 and 3.3, denoted as Protocol II and Protocol III, respectively, are able to search for both single and conjunctive keywords.

Protocol I is a special case of Protocol II. Both of them aim to achieve space efficiency in terms of the size of encrypted dictionary.

For a query $Q = (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, the size of encrypted dictionary required in Protocol I is $n_2|D|$. For a query $Q = S_1 \vee S_2 \vee \ldots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, the size of encrypted dictionary in Protocol II is $(n_2 + 1)|D|$, independent of the number of single keywords n_1.

However, both Protocol I and Protocol II require to compute a large number of pairings e, which is even more expensive than computing modular exponentiation for large modulo. To process a document M, the number of pairing computation required in Protocol I is $n_2|M|$, and the number of pairing computation required in Protocol II is $(n_2 + 1)|M|$.

Protocol III aims to achieve computation efficiency by reducing the number of pairing computation required. However, it requires a longer encrypted dictionary. For a query $Q = S_1 \vee S_2 \vee \ldots \vee S_{n_1} \vee (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_{n_2} \wedge B_{n_2})$, the number of pairing computation required to process a document M in Protocol III is $n_2 + 1$ only, independent of both the size of the document $|M|$ and the number of single keywords n_1. But the size of the encrypted dictionary required in Protocol III is $2(n_2 + 1)|D|$.

A performance comparison of the three protocols is shown as in Tab. 1.

Protocols	Size of Encrypted Dictionary	Number of Pairing Computation
Protocol I	$n_2\|D\|$	$n_2\|M \cap D\|$
Protocol II	$(n_2 + 1)\|D\|$	$(n_2 + 1)\|M \cap D\|$
Protocol III	$2(n_2 + 1)\|D\|$	$n_2 + 1$

Tab. 1 Performance Comparison

If the size of the document M is small (e.g., the document is a list of keywords only) and the size of dictionary is large, Protocol II will be a better option.

6. RELATED WORK

Private searching on streaming data is related to searching on encrypted data [20, 3, 11, 4, 5], where the original file is encrypted by a public key of the user. Searching on encrypted data requires that given a keyword by the user, the server is able to tell whether an encrypted file contains the keyword or not, but learns nothing else about the original file. Private searching on streaming data requires that the original data is in the clear, but the output of searching is encrypted. Essentially, private searching on streaming data and searching on encrypted data are different.

Private searching on streaming data is also closely related to Single-database Private Information Retrieval (PIR) [7, 12, 6, 9, 8] and oblivious transfer [15,13], which allows a user to retrieve a record from a database without the owner of that database being able to determine which record was selected, and with the communication cost less than the database size. There are important differences between private searching on streaming data and single-database PIR. In the streaming model, the size of the query must be independent of the stream, as the stream is assumed to be an arbitrarily large set of data and we do not know the size of the stream when compiling the query. In contrast, in all PIR protocols, when creating the PIR query, the user of the PIR protocol must know the upper bound on the database size. In addition, the PIR protocol allows one to search a single keyword in the database and return a single result. If one wants to query

data based on an "OR" of several keywords, then several PIR queries must be created and sent to the database. Private searching on streaming data allows us efficiently to query the data based on an "OR" of a set of keywords.

7. CONCLUSION

In this paper, we have presented three protocols, which support private searches of single and conjunctive keywords on streaming data. Our approach adds a new type of query into private searching on streaming data.

A problem with our solution is that the number of conjunctive keywords in the private filter is closely related to the number of arrays of ciphertexts in the encrypted dictionary. Our future work will investigate how to hide the number of conjunctive keywords in the private filter.

8. REFERENCES

[1] Bethencourt J., Song D. and Water B. 2006. New construction and practical applications for private streaming searching, in *Proc. IEEE Symposium on Security and Privacy (SP'06).*

[2] Bethencourt J., Song D., and Water B. 2009. New techniques for private stream searching, *ACM Transactions on Information and System Security*, 12(3), 16:1-32.

[3] Boneh D., Crescenzo G., Ostrovsky R. and Persiano G. 2004. Public encryption with keyword search, in *Proc. Eurocrypt'04*, 506-522.

[4] Boneh D., Goh. E and Nissim K. 2005. Evaluating 2-DNF formulas on ciphertext, in *Proc. TCC'05*, 325-341.

[5] Boneh D., Waters B. 2007. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Proc. TCC'07*, 535-554

[6] Cachin C., Micali S., Stadler M. 1999. Computationally private information retrieval with polylogarithmic communication. *In Proc of EUROCRYPT'99*, 402-414

[7] Chor B., Coldreich O., Kushilevitz E. and Sudan M. 1995. Private information retrieval, in *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS'95).*

[8] Chang Y. C. 2004. Single database private information retrieval with logarithmic communication, in *Proc. ACISP'04*, 50-61.

[9] Crescenzo G. D., Malkin T., Ostrovsky R. 2000. Single-database private information retrieval implies oblivious transfer. In *Proc Eurocrypt'00*.

[10] Damgård I. and Jurik M. 2001. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system, in *Proc. PKC'01*, 119-136

[11] Golle P., Staddon J., and Waters B. 2004. Secure conjunctive keyword search over encrypted data. In *Proc. ACNS'04*, 31-45.

[12] Kushilevitz E. and Ostrovsky R. 1997. Replication is not needed: single database, computational-private information retrieval, in *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS'97)*, 364-373.

[13] Lipmaa H. 2005. An oblivious transfer protocol with log-squared communication, in *Proc. ISC'05*.

[14] Menezes A., Van Oorschot P. and Vanstone S. 1997. *Handbook of Applied Cryptography*, CRC Press.

[15] Naor M., and Pinkas. B. 1999. Oblivious transfer and polynomial evaluation, in *Proc. STOC'99*.

[16] Okamoto T. and Uchiyama S. 1998. A new public-key cryptosystem as secure as factoring. In *Proc Eurocrypt'98*, 308-318.

[17] Ostrovsky R. and Skeith W. 2005. Private searching on streaming data, in *Proc. Crypto'05*, 223-240.

[18] Ostrovsky R. and Skeith W. 2007. Private searching on streaming data, *Journal of Cryptology, 20(4),* 397-430.

[19] Paillier P. 1999. Public key cryptosystems based on composite degree residue classes, in *Proc. Eurocrypt'99*, 223-238.

[20] Song D. X., Wagner D., Perrig A. 2000. Practical techniques for searches on encrypted data. *In Proc. IEEE Symposium on Security and Privacy 2000*, 44-55.