Executing Boolean Queries on an Encrypted Bitmap Index

Mohamed Ahmed Abdelraheem
SICS Swedish ICT AB
Lund, Sweden
mohamed.abdelraheem@sics.se

Christian Gehrmann
SICS Swedish ICT AB
LTH, Lund University
Lund, Sweden
chrisg@sics.se

Malin Lindström
Blekinge Institute of Technology
Karlskrona, Sweden
mkli10@student.bth.se

Christian Nordahl
Blekinge Institute of Technology
Karlskrona, Sweden
cno@bth.se

ABSTRACT

We propose a simple and efficient searchable symmetric encryption scheme based on a Bitmap index that evaluates Boolean queries. Our scheme provides a practical solution in settings where communications and computations are very constrained as it offers a suitable trade-off between privacy and performance.

Keywords

Searchable symmetric encryption; conjunctive search; Boolean query; Bitmap index.

1. INTRODUCTION

Cloud computing is attractive to a broad range of users as well as private and public organizations by offering massive processing and storage at reduced cost. In addition, the cloud computing paradigm enables for providers to offer a range of software services such as email, spreadsheet and document management applications. One of the main obstacles for even higher adoption rates, is the lack of privacy. In most current cloud services, cloud providers are able to access the data stored at their servers. Encrypting the data before outsourcing it to the cloud can be a solution to this problem, but then processing the encrypted data by the cloud is possible through currently impractical solutions such as fully homomorphic encryption schemes [10]. One trivial but impractical solution is to download all the encrypted stored data and decrypt it locally on the client machine and then search for the desired keyword or execute SQL queries in case the data is stored in a relational database.

However, some practical solutions for searching on encrypted data do exist at the cost of leaking some information about the encrypted data. One of these schemes is searchable symmetric encryption (SSE). The very first SSE scheme was proposed by Song et al. in [18] where searches are made possible by symmetrically encrypting each word separately and then embedding its deterministic encryption value (trapdoor) inside the encrypted document. When a user is searching for a keyword, the client application will ask the server to match the deterministic encryption value of the keyword (trapdoor) with the existing deterministic encryption values on the encrypted documents in the server. When a match is found, the server retrieves the corresponding document(s). This scheme is an in-place searchable encryption scheme where searches are performed per document by iterating over all the encrypted keywords within the document. Song et. al scheme's showed that their scheme is *indistinguishable against chosen plaintext attacks* IND-CPA which ensures that the ciphertext does not leak any information about the plaintext. However the IND-CPA security definition does not address the main leakage of searchable encryption schemes coming from the trapdoors and queries. Two security definitions addressing the main leakage of searchable encryption schemes were introduced by Goh in [11] and Chang and Mitzenmacher in [7]. Goh introduced the concept of indistinguishably against adaptive chosen keyword attacks and Chang and Mitzenmacher introduced a simulation-based security definition. Curt-

CCSW'16, October 28 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4572-9/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2996429.2996436

mola et al. [9] showed that these security definitions are insufficient and do not imply that keywords cannot be recovered. Moreover, they proposed two security definitions which have become since then the standard security definitions for searchable symmetric encryption schemes. The first security definition addresses security against non-adaptive adversaries whose current search queries are not based on the outcome of their previous queries while the second definition addresses security against an adaptive adversary whose current queries are based on the outcome of their previous queries. Regarding efficiency Curtmola et al. [9] proposed the use of an inverted index which is an index per keyword whose search is linear in the number of documents containing the keyword. This is optimal for search as opposed to the forward index which is an index per document used in previous schemes such as Goh's [11] scheme and Chang and Mitzenmacher's scheme [7].

Curtmola et al.'s security model for SSE together with their inverted index has been used by many subsequent SSE schemes. Chase and Kamara [8] proposed an adaptively secure SSE based on an inverted index data structure where padding is used to hide the length of the list corresponding to each keyword. Liesdonk et al. [19] proposed the first dynamic adaptively secure SSE scheme. They used a different index data structure, namely, the Bitmap index which is a keyword per Bitmap index. Also Kamara et al. [13] proposed a dynamic SSE scheme supporting update operations such as addition and deletion of documents.

All the above schemes support only single keyword search where conjunctive or a general Boolean search can only be supported using the naïve approach which basically means performing a single keyword search on each keyword's trapdoor and then let either the server or the client to compute the intersections of the search results. Beside being inefficient, the naïve approach obviously leaks a lot of information to the server beyond the search result. A practical solution to this problem has been presented recently at Crypto 2013 by Cash et al. [6] where they proposed searchable encryption schemes executing arbitrary Boolean queries of the form "$w_1 \wedge f(w_2, \cdots, w_{n_t})$" with a remarkable combination of a minimal security leakage and an efficient sublinear search time. They proposed two schemes, the first scheme is called the Basic Cross-Tags (BXT) protocol. The second protocol is called the Oblivious Cross-Tags (OXT) protocol which achieves security against a stronger attacker model compared to the BXT protocol.

Our Contribution. Beside providing a practical trade-off between privacy and performance where the leakage is reduced to its minimal and the search complexity is sublinear, Cash et al. SSE schemes [6] suffer from communications and computations overhead. Moreover, their schemes use an auxiliary data structure called the XSet beside the main index data structure the TSet. This adds a storage overhead that might not be suitable for very large databases. To address the communications, computations and storage limitations, we propose a searchable symmetric encryption scheme similar to the basic Boolean query scheme proposed by Cash et al. [6] as it evaluates arbitrary Boolean queries of the form "$w_1 \wedge f(w_2, \cdots, w_{n_t})$" with minimal leakage and low communication overhead. Our scheme uses a Bitmap index as its main index data structure which allows to perform Boolean queries without needing an auxiliary data structure such as the XSet used in Cash et al. searchable symmetric encryption schemes.

Related work. Several searchable encryption schemes proposed solutions [12, 2, 4, 14] addressing only conjunctive queries $w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$. These solutions leak only the set of matching documents but they are not suitable for environments where computations and communications are constrained due to the pairings operations. Also, all these solutions handle only conjunctive queries and none of them handles general Boolean queries. However, our solution is similar to Cash et al.'s BXT protocol [6] which handles arbitrary Boolean queries of the form "$w_1 \wedge f(w_2, \cdots, w_{n_t})$" but achieves better security and a smaller communication overhead. Moreover our solution does not need an additional auxiliary data structure which would be an improvement when dealing with very large databases.

Organization of the paper. In Section 2, we introduce searchable symmetric encryption schemes. In Section 3, we describe the BXT protocol [6] which is the protocol that we want to improve. In Section 4, we describe an efficient BXT-like searchable symmetric encryption whose encrypted index is a Bitmap data structure and demonstrate its advantages over the BXT protocol [6]. In Section 5, we give some experimental results demostrating the advantage of our approach for some very large databases compared to previous work in [6, 5]. Finally, in Section 6 we conclude our results.

2. BACKGROUND AND DEFINITIONS

Notation. Throughout the paper, we use the following notation. The security parameter is denoted by λ. Let DB denotes a database of n documents and m unique keywords. Let $\mathsf{ID} = (\mathsf{id}_i)_{i=1}^{n}$ denotes the set of all document identifiers in DB and let $\mathsf{W} = (w_j)_{j=1}^{m}$ denotes the set of all unique keywords in DB. The database can be represented as an inverted index where each keyword is associated with a list of identifiers, i.e. $\mathsf{DB} = (w_i, \mathsf{DB}(w_i))_{i=1}^{m}$ where $\mathsf{DB}(w_i)$ denotes the set of documents containing the keyword w_i. Let EDB denotes an encrypted database index.

Bitmap Index. Our searchable symmetric encryption scheme is based on the Bitmap index. Let $\mathsf{DB} = (w_j, \mathsf{DB}(w_j))_{j=1}^{m}$ be a database index. Then for each keyword w_j, allocate an n-bit vector $\mathcal{B}_j$ where the ith bit $\mathcal{B}_j[i]$ is set to 1 if $\mathsf{id}_i \in \mathsf{DB}(w_j)$ and otherwise is set to 0. So the plaintext Bitmap index is $(w_j, \mathcal{B}_j)_{j=1}^{m}$.

Denote by $B_j[s_1, \cdots, s_2]$ a substring of the Bitmap B_j where s_1 is the begin index and s_2 is the end index.

Hash Tables. Our conjunctive search protocol on the Bitmap relies on hash tables. A Hash table data structure has two functions. The first function, $\mathsf{HTConstruct}(L)$ takes as an input a list of m elements $L = (l_i, d_i)_{i=1}^m$ where l_i is an l-bit string label and d_i is an r-bit data and m is the number of unique keywords in the database as mentioned above, and outputs a hash table HT.

The second function, $\mathsf{HTRetrieve}(\mathsf{HT}, l_i)$ outputs d_i if $(l_i, d_i) \in L$ and outputs $\emptyset$ otherwise. The total size of the hash is $O(m(l+r))$. For more info about hash tables implementations, we refer the reader to [17].

2.1 Searchable Symmetric Encryption.

Definition. A searchable symmetric encryption (SSE) scheme enables a server to search on an encrypted database consisting of documents (or records) without learning the plaintext of the encrypted database. SSE achieves good performance at the cost of leaking some information about the plaintext according to a *leakage function* (defined below). All SSE schemes leak the *access pattern*: the result of the query or the document IDs corresponding to the queried keyword and the *search pattern*: the fact that whether two searches are the same or not. An SSE scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TrapGen}, \mathsf{Search})$, consists of a random key generation algorithm KeyGen that generates the secret keys to be used in the scheme, a randomized encryption algorithm $\mathsf{EDBSetup}$ that takes as input a database DB and the secret keys and outputs an encrypted database EDB, a trapdoor generation algorithm $\mathsf{TrapGen}$ and a search protocol Search between a client holding the secret keys and a server holding the encrypted database EDB.

The leakage function. The minimal leakage in an SSE scheme $\mathcal{L}_{\min}(\mathsf{DB})$ is the number of keyword occurrences in the plaintext database DB, $N = \sum_{w \in W} |\mathsf{DB}(w)|$ and $\{\mathsf{DB}(\mathsf{w_i}) : \mathsf{w_i} \in \mathsf{W}\}$.

However, an SSE scheme can leak more information such as the unique number of keywords m, the number of documents n or $M = \max_w |\mathsf{DB}(\mathsf{w})|$. The initial leakage $\mathcal{L}(\mathsf{DB})$ given to the simulator S_0 does not include any $\mathsf{DB}(w)$ since no query has been issued. Depending on the SSE scheme, the initial leakage can be the minimal size leakage N as in [6]. Specifically, in our scheme, the initial leakage will be m and n. The leakage given to the simulator S_i where $i > 0$ is $\mathcal{L}(\mathsf{DB})$ and the access pattern of w_i, $\mathsf{DB}(w_i)$ in addition to the access pattern leakage from the previous queries $\mathsf{DB}(w_j)$ where $j < i$.

Adaptive Security Definition. We recall the adaptive semantic security [9] definition of an SSE scheme with respect to a leakage function $\mathcal{L}(\mathsf{DB})$ capturing the information that an adversary (honest-but-curious server) $\mathcal{A} = (\mathcal{A}_0, \cdots, \mathcal{A}_{q+1})$ is allowed to learn about the DB and the search queries during its interaction with a secure SSE scheme. Formally, the security definition requires that the view of the adversary (i.e. encrypted index, trapdoors) generated during an adaptive attack can be simulated by a simulator $\mathcal{S} = (\mathcal{S}_0, \cdots, \mathcal{S}_q)$ given only the defined leakage profile $\mathcal{L}$. More precisely, let $\mathsf{REAL}^{\mathsf{adap}}_{\Sigma,\mathcal{A}}(\lambda)$ denote the output of the real execution where the adversary will attempt to compute $f(\mathsf{DB})$ given the encryption of DB and let $\mathsf{IDEAL}^{\mathsf{adap}}_{\Sigma,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)$ denote the output of the ideal world execution where the adversary will attempt to compute $f(\mathsf{DB})$ given the encryption of the database created by the simulator given the leakage profile $\mathcal{L}$. Note that f is a function or more precisely a distinguisher that the adversary tries to find in order to predict the value $f(\mathsf{DB})$ with a probability that is significantly better than predicting it when given no information about the real DB other than its defined leakage $\mathcal{L}(\mathsf{DB})$.

DEFINITION 1. *(Adaptive security) Let* $\Sigma = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TrapGen}, \mathsf{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be a leakage function. We say that the scheme* Σ *is* $\mathcal{L}$*-adaptively-secure SSE scheme if for every efficient adversary* $\mathcal{A}$ *there is an efficient simulator* $\mathcal{S}$ *such that*

$$|\Pr[\mathsf{REAL}^{\mathsf{adap}}_{\Sigma,\mathcal{A}}(\lambda)] - \Pr[\mathsf{IDEAL}^{\mathsf{adap}}_{\Sigma,\mathcal{L},\mathcal{A},\mathcal{S}}(\lambda)]| < \mathsf{neg}(\lambda).$$

In the following, we describe the single keyword Bitmap SSE scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TrapGen}, \mathsf{Search})$. The scheme relies on a pseudo random functions (PRF) $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{K}$, pseudo random permutation (PRP) $\pi : \mathcal{K} \times \{0,1\}^l \to \{0,1\}^l$ assuming that the keywords are represented as l-bit strings and a hash function H with output size b bits which for adaptive security is modeled as a random oracle.

$\underline{\mathsf{KeyGen}(1^\lambda):}$

Sample $K_I, K_\pi \xleftarrow{\$} \{0,1\}^\lambda$ and output $K = (K_I, K_\pi)$. K_I is used to generate unique subkeys to encrypt b-bit blocks in each keyword Bitmap B_j. K_π will be used to permute the Bitmaps $(B_j)_{j=1}^m$.

$\underline{\mathsf{EDBSetup}(\mathsf{DB}):}$

- For each $w_j \in \mathsf{W}$ build an n-bit vector B_j as follows:
  - For each $\mathsf{id}_i \in \mathsf{DB}(w_j)$: set $B_j[i] = 1$ and zero otherwise.
  - Set $K_{w_j} \leftarrow F(K_I, w)$.
  - Divide B_j in to b-bit blocks and encrypt each block $l_j[t]$, $1 \le t \le \lceil \frac{n}{b} \rceil$ as follows (if n is not a multiple of b we pad the last block to satisfy this by adding some random bits):
    - $l_j[t] \leftarrow B_j[(t-1)*b+1, \cdots, b*t]$
    - $l_j[t] \leftarrow l_j[t] \oplus H(K_{w_j}||t)$
    - $B_j[(t-1)*b+1, \cdots, b*t] \leftarrow l_j[t]$
- Permute the encrypted Bitmap $B = (B_j)_{j=1}^m$ by applying π to each row B_j: $B = B_{\pi(K_\pi, j)} \leftarrow B_j$.

- Output $\mathsf{EDB} = (B)$.

$\underline{\mathsf{TrapGen}(K, w_j)}$:

- The client takes as input the key $K = (K_I, K_\pi)$ and a keyword w_j.
- It computes $K_j \leftarrow \pi(K_\pi, j)$ and $K_w \leftarrow F(K_I, w_j)$
- Output $T_{w_j} = (K_j, K_{w_j})$.

$\underline{\mathsf{Search}}$:

- The client takes as input the key $K = (K_I, K_\pi)$ and a keyword w_j to query.
- The client computes $(K_j, K_{w_j}) \leftarrow \mathsf{TrapGen}(K, w_j)$.
- The server divides B_{K_j} into b-bit blocks and decrypt each block $B_{K_j}[t]$, $1 \leq t \leq \lceil \frac{n}{b} \rceil$ as follows:
  - $l_j[t] \leftarrow B_{K_j}[(t-1)*b+1,\ \cdots, b*t]$
  - $l_j[t] \leftarrow l_j[t] \oplus H(K_{w_j}||t)$
  - $B_{K_j}[(t-1)*b+1,\ \cdots, b*t] \leftarrow l_j[t]$
  - Find the active bits in block $B_{K_j}[(t-1)*b+1,\ \cdots, b*t]$ and return their corresponding document identifiers id_i.

In Section 4, we show how to slightly modify the above single keyword search scheme to handle Boolean queries. The single keyword SSE scheme has leakage $\mathcal{L} = \{n, m\}$ and it is very similar to [3, 19] which have the same leakage and are proved to be $\mathcal{L}$-adaptively secure. So we have the following theorem. We do not provide a proof here as it will be redundant. However, one difference between the proofs in [3, 19] is our use of a random oracle. We do this mainly for adaptive security and to avoid the lower bound on the search tokens given by Chase and Kamara in [8].

THEOREM 1. *The single keyword SSE scheme Σ described above is adaptively secure under the assumption that F is pseudorandom function and π is a pseudorandom permutation and the hash function H is modeled as a random oracle.*

3. CASH ET AL.'S SSE SCHEME FOR BOOLEAN QUERIES

Recently, Cash et al. [6] have proposed two schemes (the BXT and the OXT SSE protocols) that execute Boolean queries of the form "$w_1 \wedge f(w_2, \cdots, w_{n_t})$" efficiently and have lesser leakage compared to the naïve approach. Their main encrypted searchable index and length-hiding data structure, TSet, is basically a dictionary data structure which is slightly different than the one proposed by Curtmola et al. in [9]. Different and efficient alternatives for the TSet are proposed in [5]. In Appendix A, we briefly describe the TSet data structure. For more details, we refer the reader to [6]. Next, we describe Cash et al.'s BXT protocol [6].

3.1 The Basic Cross-Tags (BXT) Protocol

Assuming that w_1 is the least frequent keyword in the conjunctive query "$w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$", the BXT protocol [6] computes the conjunctive query by using the single keyword search (SKS) scheme described in [6] to allow the server to retrieve the document IDs corresponding to w_1, $\mathsf{TSet}(w_1)$. Then the protocol enables the server to perform the intersections with the other terms $w_2, \cdots, w_{n_t}$ and only returns the document IDs matching the full conjunctive query "$w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$". As mentioned in [6], the least frequent keyword w_1 is called the s-term where the 's' refers to the use of the "SKS" scheme or to w_1 being the keyword whose frequency is "small". Also, the other terms in the conjunction are called the x-terms where 'x' refers to "cross".

Besides the TSet data structure, the BXT protocol uses another data structure called the XSet which is basically a set data structure. It can be implemented as a Bloom filter data structure (or any set representation that is history-independent as noted in [6]). By adding $f(\mathsf{xtrap}_i, \mathsf{ind})$ to the Bloom filter data structure, XSet, the server will be able to decide whether or not the keyword w_i corresponding to xtrap_i exists in the document with ID equivalent to ind.
The encrypted database in the BXT protocol $\mathsf{EDBSetup}$ is setup as follows:

1. Choose keys K_S and K_X for the PRF F.
2. Initialize $\mathbf{T}$ to an empty array indexed by keywords from the set of all keywords W.
3. Initialize XSet to an empty set.
4. For each $w \in \mathsf{W}$, construct the tuple list $\mathbf{T}[w]$ as follows:
   - Initialize $\mathbf{t}$ to an empty list, and set $K_e \leftarrow F(K_S, w)$.
   - Compute $\mathsf{xtrap} \leftarrow F(K_X, w)$.
   - For all document IDs $\mathsf{ind} \in \mathsf{DB}(w)$ in random order:
     - $e \leftarrow \mathsf{Enc}(K_e, \mathsf{ind})$; append e to $\mathbf{t}$.
     - $\mathsf{xtag} \leftarrow f(\mathsf{xtrap}, \mathsf{ind})$ and insert xtag to XSet.
   - $\mathbf{T}[w] \leftarrow \mathbf{t}$.
5. $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathbf{T})$.
6. $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$.

The Search protocol in the BXT protocol is as follows:

1. The client takes as input the keys K_S, K_X and K_T and $w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$ as a query. Assuming that w_1 is the least frequent keyword in the query, it evaluates:
   - $K_e \leftarrow F(K_S, w_1)$
   - $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w_1)$

   - For each $i = 2, \cdots, n_t$, it evaluates $\mathsf{xtrap}_i \leftarrow F(K_X, w_i)$.
   - It sends $(\mathsf{stag}, K_e, \mathsf{xtrap}_2, \cdots, \mathsf{xtrap}_{n_t})$ to the server.

2. The server takes as input $(\mathsf{TSet}, \mathsf{XSet})$ and replies as follows:
   - $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$
   - For each entry e in $\mathbf{t}$:
     - $\mathsf{ind} \leftarrow \mathsf{Dec}(K_e, e)$.
     - if $f(\mathsf{xtrap}_i, \mathsf{ind}) \in \mathsf{XSet}\ \forall i = 2, \cdots, n_t$, then send ind to the client.

3. The client outputs all of the received inds.

Analysis of the BXT protocol:

One can see that Cash et al.'s BXT protocol described above is very efficient. The server takes time in the order of $n_t \cdot |\mathsf{D}(\mathsf{w}_1)|$ where w_1 is the s-term in the conjunctive query and n_t is the number of terms or keywords in the query. This represents a significant improvement over previous schemes which are linear in the database and also over the naïve solution whenever there is a term with less number of occurrences in the database.

In terms of security, the BXT protocol has lesser leakage compared to the naïve solution as it only leaks the document IDs corresponding to the s-term and not the x-terms where the naïve solution leaks the document IDs corresponding to each term involved in the conjunctive query. However, the above described solution allows the server to correlate information from different queries about the x-terms. One can see that once the server receives the xtrap_i corresponding to an x-term w_i at at one query, then the server can later use it against an ind found through an s-term in another query. In other words, once the server gets xtrap_i during the query "$w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$", it can save xtrap_i in order to decide if an old or a newly revealed document ID ind contains w_i by testing the membership of $f(\mathsf{xtrap}_i, \mathsf{ind})$ in the set or Bloom filter XSet.

As pointed in [6], one can prevent the above attack at the cost of having multiple communication rounds, by evaluating the function $f(\mathsf{xtrap}_i, \mathsf{ind})$ at the client side and sending its result to the server rather than evaluating it at the server side. Henceforth, we will call the BXT protocol after this fix, the *interactive BXT protocol*. However, as pointed in [6], this fix comes at the cost of extra communication between the client and the server in which the client needs to send the results of $(n_t - 1)$ evaluations of the f function for each document ID ind. Assuming that the output of f has size s bits, then the size of transmitted bits for all the x-terms w_i is $(n_t - 1) \cdot s \cdot |\mathsf{DB}(w_1)|$ bits. To avoid this extra communication cost, the authors propose the Oblivious Cross Tags (OXT) protocol [1]. To address this communication issue, we propose an efficient single round of interaction BXT-like protocol as opposed to the above multi-round interactive BXT protocol. Moreover, the total transmission cost of our protocol is reduced by a factor of s, namely, $(n_t - 1) \cdot |\mathsf{DB}(w_1)|$ bits.

In terms of storage efficiency, both the BXT and the OXT protocols suffer from using an additional auxiliary data structure (XSet) beside the main index data structure (TSet). The XSet data structure is employed to enable the execution of Boolean queries. It holds the secret values computed from each pair of keyword and document identifier. In [6], the XSet is realized as a RAM-resident Bloom filter which might be suitable for some databases. However, as mentioned in [6], for very large databases an efficient disk-resident XSet is needed. In fact, it is mentioned as part of their future work (See page 28, [6]). Our protocol addresses this storage issue by processing the Boolean queries without using an auxiliary set such as Cash et al.'s XSet.

4. AN EFFICIENT SSE SCHEME: THE BXT-BITMAP PROTOCOL

Liesdonk et al. [19] described a dynamic and adaptively secure SSE scheme whose index is a Bitmap index data structure. Later, Bösch et al. also used a Bitmap index in their distributed searchable scheme [3]. In order to reduce the storage space of the Bitmap index, Kuzu et al. [15] also used a Bitmap index combined with an inverted index in their distributed search over encrypted big data. All these schemes are single keyword Bitmap-based searchable encryption schemes that are adaptively secure.

In this section, we propose an SSE scheme that performs Boolean queries on the Bitmap index. Our scheme realizes the BXT protocol described above on the Bitmap index as opposed to the TSet and the XSet data structures used in [6]. Using only the Bitmap index with two small hash tables, we are able to perform conjunctive search without an auxiliary huge set such as the XSet which is not suitable for very large databases. We focus here on conjunctive queries which can easily be extended to the arbitrary Boolean queries of the form $w_1 \wedge f(w_2, \cdots, w_{n_t})$ without changing the protocol. For more details about this, we refer the reader to [6].

We begin by describing how to setup an encrypted database based on the Bitmap index. Then we describe how to perform the conjunctive search efficiently as done in the BXT protocol which was specifically designed to enable search on a secure inverted index or dictionary data structure. However, we show that using a Bitmap index instead we can perform conjunctive queries with better communication efficiency without compromising security at the price of a linear search process that can be easily parellelized.

4.1 Description of the BXT-Bitmap Scheme

We show here how to slightly modify the single key-

[1]For more details about the OXT protocol, we refer the reader to [6].

word SSE scheme described above to handle Boolean queries. More specifically, we protect the trapdoor K_j generated for the keyword w_j by encrypting it with two different keys and storing the resulting encryption in two different hash tables where one is accessed when the keyword is used as an s-term and the other is accessed when the keyword is used as an x-term. So unlike the single keyword scheme where K_j is part of the trapdoor, our new scheme generates a trapdoor for the s-term and the x-terms to enable the server to retrieve their corresponding encrypted K_j pointer from the hash tables and decrypt them.

Our conjunctive Bitmap-based search protocol is based on the *interactive BXT protocol*. However, due to the Bitmap data structure, we achieve conjunctive search without the need for an XSet and the client will be able to decide itself whether an x-term w_i exists in the documents returned by the s-term w_1 or not. Therefore, there is no interaction in our protocol and it only consists of two rounds, in the first round, the client sends the secret token stag of w_1 together with its corresponding secret key and the secret token xtag of each x-term w_i. In the second round, the server responds to the client by sending document identifiers id_s corresponding to the s-term w_1 together with encrypted bits corresponding to each x-term with each document identifier id_s.

Our protocol relies on two hash tables $\mathsf{HT}_{\mathsf{stag}}$ and $\mathsf{HT}_{\mathsf{xtag}}$. The $\mathsf{HT}_{\mathsf{stag}}$ hash table is used to process the s-term w_1. In each entry (l, d), the label l is a hash of the secret token generated using a PRF F' with a secret key K_S and the data d is the encryption of the pointer K_j to the Bitmap corresponding to the keyword w_j concatenated with its Bitmap encryption key K_{w_j}. The $\mathsf{HT}_{\mathsf{xtag}}$ hash table is used to process the x-terms. In each entry (l, d), the label l is a hash of the secret token generated using a PRF F' with a secret key K_X and the data d is only the encryption of the pointer K_j corresponding to the keyword w_j in order to enable the server to access the Bitmaps corresponding to the x-term keywords without the ability to decrypt since K_{w_j} is only stored encrypted on the $\mathsf{HT}_{\mathsf{stag}}$ and it is only revealed when processing the s-term which needs the decryption key to return the corresponding document identifiers id_s.

We now proceed to describe our conjunctive search BXT-Bitmap SSE $\Sigma_{\mathsf{BXT-BIT}} = (\mathsf{KeyGen}, \mathsf{EDBSetup}, \mathsf{TrapGen}, \mathsf{Search})$ protocol based on the Bitmap index. Similar to the single keyword scheme our scheme relies on two pseudo random functions $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{K} \times \mathcal{K}$ and $F' : \mathcal{K} \times \{0,1\}^* \to \mathcal{K}$, and a pseudo random permutation $\pi : \mathcal{K} \times \{0,1\}^l \to \{0,1\}^l$ assuming that the keywords are represented as l-bit strings. We also use a hash function H with output size b bits to generate the block keys for the n-bit Bitmap and which for adaptive security is modeled as a random oracle in the security proof. Also for adaptive security we use an encryption scheme $(\mathcal{E}, \mathcal{D})$ with a keyspace $\mathcal{K}$ where $\mathcal{E}(K, m) = (r, H'(K||r) \oplus m)$ where r is a random value and H' is a random oracle with output size λ-bit. H' is also used to generate the l-bit labels for the hash tables and in this case its λ-bit output is truncated to l-bit.

$\underline{\mathsf{KeyGen}(1^\lambda):}$

Sample $K_I, K_\pi, K_S, K_X \xleftarrow{\$} \{0,1\}^\lambda$ and output $K = (K_I, K_\pi, K_S, K_X)$. K_I is used to generate unique subkeys to encrypt b-bit blocks in each keyword Bitmap B_j. K_π will be used to permute the Bitmaps $(B_j)_{j=1}^m$. K_S is used to generate a secret trapdoor for a keyword when it is used as an s-term in a conjunctive query while K_X is used to generate a secret trapdoor for the same keyword when it is used as an x-term in a conjunctive query.

$\underline{\mathsf{EDBSetup}(\mathsf{DB}):}$

- Initialize an $m \times n$ Bitmap index $(B_j)_{j=1}^m$ where B_j is an n-bit vector.
- Initialize two empty lists L_{stag} and L_{xtag}.
- For each $w_j \in \mathsf{W}$ process B_j as follows:
  - For each $\mathsf{id}_i \in \mathsf{DB}(w_j)$: set $B_j[i] = 1$ and zero otherwise.
  - Compute $K_{w_j} \leftarrow F'(K_I, w_j)$.
  - Divide B_j in to b-bit blocks and encrypt each block $l_j[t]$, $1 \le t \le \lceil \frac{n}{b} \rceil$ as follows (if n is not a multiple of b we pad the last block to satisfy this by adding some random bits):
    - $l_j[t] \leftarrow B_j[(t-1)*b+1, \cdots, b*t]$
    - $l_j[t] \leftarrow l_j[t] \oplus H(K_{w_j}||t)$
    - $B_j[(t-1)*b+1, \cdots, b*t] \leftarrow l_j[t]$
  - Compute $(K_{s_1}, K_{s_2}) \leftarrow F(K_S, w_j)$.
  - Compute $K_j \leftarrow \pi(K_\pi, j)$
  - Set label $l \leftarrow H'(K_{s_1})$ and data $d \leftarrow \mathcal{E}(K_{s_2}, K_j||K_{w_j})$ and add (l, d) to L_{stag}.
  - Compute $(K_{x_1}, K_{x_2}) \leftarrow F(K_X, w_j)$.
  - Set label $l \leftarrow H'(K_{x_1})$ and data $d \leftarrow \mathcal{E}(K_{x_2}, K_j)$ and add (l, d) to L_{xtag}.
- Randomly permute the encrypted Bitmap $B = (B_j)_{j=1}^m$ by applying π to each row B_j: $B = B_{\pi(K_\pi, j)} \leftarrow B_j$.
- Compute $\mathsf{HT}_{\mathsf{stag}} \leftarrow \mathsf{HTConstruct}(L_{\mathsf{stag}})$ and $\mathsf{HT}_{\mathsf{xtag}} \leftarrow \mathsf{HTConstruct}(L_{\mathsf{xtag}})$
- Output $\mathsf{EDB} = (B, \mathsf{HT}_{\mathsf{stag}}, \mathsf{HT}_{\mathsf{xtag}})$.

$\underline{\mathsf{TrapGen}(K_S, K_X, w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}):}$

- The client takes as input the keys K_S,K_X and a query $w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$.
- Compute $(K_{s_1}, K_{s_2}) \leftarrow F(K_S, w_1)$. Set $\mathsf{stag} \leftarrow (K_{s_1}, K_{s_2})$.

- For each $i = 2, \cdots, n_t$, compute $(K_{x_{i_1}}, K_{x_{i_2}}) \leftarrow F(K_X, w_i)$. Set $\mathsf{xtag}_i \leftarrow (K_{x_{i_1}}, K_{x_{i_2}})$.
- Output $(\mathsf{stag}, \mathsf{xtag}_2, \cdots, \mathsf{xtag}_{n_t})$.

<u>**Conjunctive Keyword** Search **Protocol**:</u>

- The client takes as input the secret keys K_S and K_X and the query $w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$. It computes:
  - $(\mathsf{stag}, \mathsf{xtag}_2, \cdots, \mathsf{xtag}_{n_t}) \leftarrow \mathsf{TrapGen}(K_S, K_X, w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t})$
  - It sends $(\mathsf{stag}, \mathsf{xtag}_2, \cdots, \mathsf{xtag}_{n_t})$ to the server.
- The server takes as input $\mathsf{EDB} = (B, \mathsf{HT}_{\mathsf{stag}}, \mathsf{HT}_{\mathsf{xtag}})$ and replies as follows:
  - Initialize an empty result set RS.
  - It extracts (K_{s_1}, K_{s_2}) from stag.
  - For each $i = 2, \cdots, n_t$, it extracts $(K_{x_{i_1}}, K_{x_{i_2}})$ from xtag_i.
  - It computes $d \leftarrow \mathsf{HTRetrieve}(\mathsf{HT}_{\mathsf{stag}}, H'(K_{s_1}))$ and $(K_j || K_{w_j}) \leftarrow \mathcal{D}(Ks_2, d)$.
  - It divides B_{K_j} into b-bit blocks and decrypt each block $B_{K_j}[t]$, $1 \le t \le \lceil \frac{n}{b} \rceil$ as follows:
    - $*$ $l_j[t] \leftarrow B_{K_j}[(t-1) * b + 1, \ \cdots, b * t]$
    - $*$ $l_j[t] \leftarrow l_j[t] \oplus H(K_{w_j} || t)$
    - $*$ $B_{K_j}[(t-1) * b + 1, \ \cdots, b * t] \leftarrow l_j[t]$
    - $*$ Find the active bits in block $B_{K_j}[(t-1) * b + 1, \ \cdots, b * t]$ and return their corresponding document identifiers id_s and add them to RS.
  - For each $\mathsf{id}_s \in \mathsf{RS}$, the server:
    - $*$ Sends id_s to the client.
    - $*$ For each $i = 2, \cdots, n_t$:
      - $\cdot$ $d \leftarrow \mathsf{HTRetrieve}(\mathsf{HT}_{\mathsf{xtag}}, H'(Kx_{i_1}))$ and $K_j = \mathcal{D}(Kx_{i_2}, d)$.
      - $\cdot$ $l_i \leftarrow B_{K_j}[\mathsf{id}_s]$ and sends (id_s, l_i) to the client.
- For each $(\mathsf{id}_s, (l_i)_{i=2}^{n_t})$ received from the server, the client computes:
  - $K_{w_i} \leftarrow F'(K_I, w_i)$ and $k \leftarrow H(K_{w_i} || \lceil \frac{\mathsf{id}_s}{b} \rceil)$
  - $v_i \leftarrow l_i \oplus k[\mathsf{id}_s(\mod b)]$.
  - if $\forall i = 2, \cdots, n_t$, $v_i = 1$, the client outputs id_s and retrieve the document whose identifier is id_s.

4.2 Analysis of the BXT-Bitmap Protocol

Efficiency-wise, our BXT-Bitmap search protocol has only a single round of interaction which is similar to Cash et al.'s OXT search protocol but better than Cash et al.'s *interactive BXT protocol* discussed above which has many rounds between the client and server. More precisely, it has one round for each $\mathsf{ind} \in \mathsf{DB}(w_1)$ and each x-term, that is $(n_t - 1) \cdot |\mathsf{DB}(w_1)|$ rounds of interactions. Moreover, the size of the transmitted bits per round in our protocol is much smaller than Cash et al.'s OXT and BXT protocols. This confirms that our search protocol is more suitable for communication constrained environments. Here we note that our $\mathsf{EDBSetup}$ needs to upload an encrypted n-bit per each keyword but that is done only once whereas the search protocol will be running throughout the lifetime of the client's application.

Table 1 shows the communication and computational advantages of our protocol compared to Cash et al.'s interactive BXT and OXT protocols [6]. Regarding the storage, we have a clear advantage when it comes to very large and dense databases since we make use of no auxiliary data structure which such as the XSet employed by Cash et al.'s protocols [6]. In Section 5, we show that for some large and dense databases such as the Census data[16], our Bitmap index consumes lesser storage compared to the TSet data structure when it is used with Cash et al.'s OXT protocol [6]. All these communication, computational and storage advantages come at the cost of more search time for identifying the corresponding document identifiers for the s-term keyword. The search cost is linear in the number of documents $O(n)$ but fortunately the search procedure is embarrassingly parallel.

Security-wise, our protocol achieves the same level of security achieved by the *interactive BXT protocol* which we will explain in what follows.

THE ROLE OF THE HASH TABLES. If there are no hash tables, then the only way for the s-term and x-term to access the Bitmaps is by sending to the server the secret Bitmap pointer K_j corresponding to the s-term and also the secret Bitmap pointer K_j corresponding to each x-term. This would enable the server to partly respond to new queries whose s-terms had been used as x-terms in previous queries. So one solution is to store the pointers K_j on two different hash tables, one to be accessed by the s-terms and another to be accessed by the x-terms.

INTERSECTION BETWEEN S-TERMS. Similar to the *interactive BXT protocol*, our protocol reveals the intersection between the s-terms of any two queries but direct intersections between x-terms of different queries are not possible. However, one can fix the s-term intersection issue at the cost of extra computation at the client side as follows. During the database setup the client permutes the document identifiers for each keyword using a pseudorandom permutation $\pi_w : \mathcal{K} \times [1, n] \rightarrow [1, n]$ with a unique secret key specifically generated for each keyword K_{π_w}. Applying this will yield a single keyword search protocol where the intersection between any two queries is not known at the index server. Now, we propose two solutions to adjust the conjunctive search according to the permutation applied for each keyword since the document identifier id_i

Protocol	**Communication Size**	**No of Rounds**	**Computations**	**Search**
BXT-Bitmap	$(n_t - 1) \cdot \mathsf{DB}(w_1)$	1	xor and hash function	$O(n)$
Interactive BXT	$(n_t - 1) \cdot \mathsf{DB}(w_1) \cdot \lvert f \rvert$	$(n_t - 1) \cdot \mathsf{DB}(w_1)$	$\mathsf{DB}(w_1)$ decryptions	$\mathsf{DB}(w_1)$
OXT	$(n_t - 1) \cdot \mathsf{DB}(w_1) \cdot \lvert G \rvert$	1	$(n_t - 1) \cdot \mathsf{DB}(w_1)$ exponentiations	$\mathsf{DB}(w_1)$

Table 1: The table draws comparisons between our protocol (BXT-Bitmap) and Cash et al.'s Interactive BXT and OXT protocols regarding the communication size in bits, the number of interaction rounds between the client and the server, and the computational operations done at the client and the server and the search complexity spent for retrieving the document identifiers corresponding to the s-term keyword w_1. $|f| \equiv$ the output size of $f(\mathsf{xtrap}_i, \mathsf{ind})$ in bits. $|G| \equiv$ the size of the group elements in bits where the group G satisfies the decision Diffie-Hellman (DDH) assumption).

represented by the i-th bit in the encrypted Bitmap corresponding to the s-term w_1 will no longer be represented at the i-th bit of the Bitmaps corresponding to the x-terms. The *first solution* comes at the cost of extra communication between the client and the server but no computational effort at the server as it only needs to retrieve either the Bitmap corresponding to the s-term keyword or certain individual bits corresponding to an x-term keyword. So it is a suitable option for cloud servers with limited computational resources. Firstly, the server sends the encrypted n-bit Bitmap corresponding to the s-term w_1 to the client so that it can decrypt it and permute the identifiers back to their initial permutations using π_w with secret key K_{π_w}. Afterwards, for each $\mathsf{id}_s \in \mathsf{DB}(w_1)$ and for each x-term w_j, the client asks the server to send the encrypted bit at the location $\pi_{w_j}(K_{\pi_w}, \mathsf{id}_s)$ in the encrypted Bitmap corresponding to the x-term w_j. The server sends the corresponding bit for each x-term per each $\mathsf{id}_s \in \mathsf{DB}(w_1)$ and the client decrypts it using the corresponding secret key bit and decides whether to output the document identifier id_s in the result set or not. The *second solution* is similar but with lower communication and with giving the server the ability to decrypt the whole Bitmap corresponding to an s-term and sending the result set back to the client which permutes back the result set to its initial state using π_w and the secret key $K_{\pi_{w_1}}$ and afterwards sends back per each $\mathsf{id}_s \in \mathsf{DB}(w_1)$, only one key bit corresponding to each x-term after applying π_w to enable the server to decrypt the corresponding bit and at the same time decide whether to include id_s in the conjunctive search result set or not.

It is worthy to note that while the OXT protocol and the two solutions described above prevents intersection between the s-terms of any two queries at the index server, during the document retrieval process the index server will be able to know the intersection between the s-terms unless the document access pattern is hidden or the index server is completely isolated from the document retrieval server and has no communication with it.

No Intra-query leakage. Our protocol has no intra-query leakage – which is the leakage from the intersection between an s-term w_1 and any subset of the x-terms $w_2, \cdots, w_{n_t}$ within the same conjunctive query $w_1 \wedge w_2 \wedge \cdots \wedge w_{n_t}$. Compared to the OXT and BXT protocols which suffer from this kind of leakage. This is possible since our protocol does not decide at the server side but at the client side whether an x-term is contained within a document whose identifier is $\mathsf{id}_s \in \mathsf{DB}(w_1)$ where w_1 is the s-term.

Security of two-term conjunctions. Let $\mathcal{L}_{bxt-bit}$ be the leakage function describing the leakage of the BXT-Bitmap protocol. We focus on the simple case where we have two keywords. Let $q[i] = w_1[i] \wedge w_2[i]$ be the ith conjunctive query where $w_1[i]$ is the s-term and $w_2[i]$ is the x-term. As pointed above, our protocol leaks the intersection between the s-term of any two queries, i.e $\mathsf{DB}(w_1[i]) \cap \mathsf{DB}(w_1[j])$ where $i \neq j$ but does not leak $\mathsf{DB}(w_2[i]) \cap \mathsf{DB}(w_2[j])$. The server access the Bitmaps corresponding to the x-terms via the hash table $\mathsf{HT}_{\mathsf{xtag}}$ which enables the server to obtain the encrypted pointer to the corresponding x-term Bitmap and decrypt it. After obtaining the pointer K_j of the x-term's Bitmap, the server accesses it and sends to the client the bits corresponding to the identifiers id_s of the s-term. Now since the server does not know the secret key, it will not be able to decrypt any single bit in the Bitmap. This ensures that the server does not learn anything about the x-terms except for what can be learned from the intersections of their corresponding s-terms. Also we note that when the client issues a query whose s-term had been used before in a previous query as an x-term, the server will not be able to notice this since the s-terms' Bitmaps are accessed through a different secret token stag stored in the $\mathsf{HT}_{\mathsf{stag}}$ hash table which is not linked to the secret token of x-terms xtag stored in the $\mathsf{HT}_{\mathsf{xtag}}$ hash table.

The leakage of our scheme $\mathcal{L}_{\mathsf{bxt-bit}}$. Our SSE scheme $\Sigma_{\mathsf{BXT-BIT}}$ leaks the number of documents n, the number of unique keywords m, the search pattern, the size of the s-term w_1, $|\mathsf{DB}(w_1)|$ and the intersection between the s-terms of any two queries. The above analysis confirms that the leakage $\mathcal{L}_{\mathsf{bxt-bit}}$ is necessary for a correct simulation. We also model the hash function as a random oracle and define our security within the random oracle model for the same reasons mentioned above for the security of the single keyword SSE scheme.

We claim in the following theorem that our protocol achieves at least the same level of security achieved by the interactive BXT protocol.

THEOREM 2. *Let $\mathcal{L}_{\mathsf{bxt-bit}}$ be as defined above. Then the above described searchable encryption scheme $\Sigma_{\mathsf{BXT-BIT}}$ is $\mathcal{L}_{\mathsf{bxt-bit}}$-adaptively-secure SSE scheme and its security is at least equivalent to that of Cash et al.'s interactive BXT protocol under the assumption that F and F' are secure pseudorandom functions, π is a secure pseudorandom permutation, H and H' are hash functions modeled as a random oracle and $(\mathcal{E}, \mathcal{D})$ is an encryption scheme as defined above.*

5. EXPERIMENTAL RESULTS

In this section, we focus on showing the practical viability of our solution. We first show that our protocol is more suitable for very large and dense datasets such as the Census dataset [16]. We then explain the results of our experiments on the Census data comparing the latency between the single-term keyword search vs the two-term conjunctive queries and how they are affected by varying the number of documents, the number of x-terms and the frequency of the involved keywords. In Appendix B, we present the results of running these experiments on the Enron data set to show that our solution is practical, in terms of search time, not only for the Census data set.

Data Sets. To show that our solution is viable, tests were run with two data sets: (1) a census data set [16] with 299,285 rows, 3,993 distinct keywords and 11,347,304 keyword-document pairs, and (2) a subset of the Enron corpus [1] with 500,000 emails, 520,218 distinct keywords and 75,062,313 keyword-document pairs. Each row in the census data set is considered to be a document d, and each attribute-value pair is treated as a keyword w. Common words, such as *an, the, for*, were removed from the Enron data set. Stemming was performed as well, to remove word-endings.

Experimental Results. Table 2 and Table 3 show the storage required for each subset of the Census and Enron data sets. The tables also show the estimated size of what the TSet would be, when used on these data sets. Comparing the two tables, it is shown that our proposed scheme occupies less space than the TSet when used on dense data sets, such as the Census. Our estimation for the TSet's storage is based on the TSet instantiation procedure described in [6]. But, when used on sparse sets, i.e. the Enron corpus, the BXT are OXT protocols are a better choice (Note that the BXT protocol is not secure and its interactive version has considerable communication overhead). Figure 1 shows that conjunctive queries introduce some computational overhead, which is expected since the identifiers of the s-term need to be compared to the identifiers of the x-term. We can also see that the search time grows as the number of documents increase, which is also expected since the

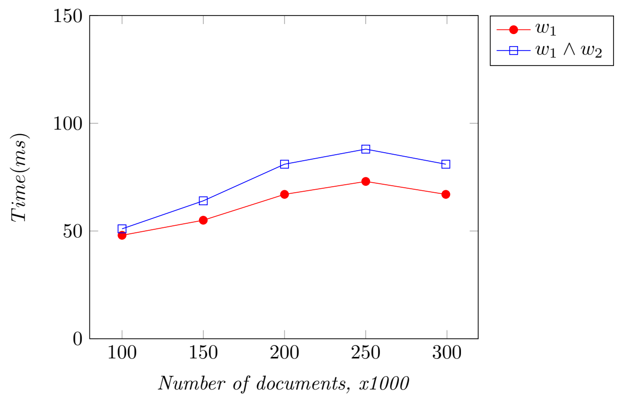

Figure 1: End-to-end execution time for the single keyword and non-interactive 2-term conjunctive search on the Census data set. Note that this figure is produced using a slightly different previous variant of the BXT-Bitmap protocol described above. The previous variant used to send the short key K_{w_j} rather than storing its encryption in the hash tables and then decrypting it during the s-term keyword search.

key generation of the bitmap block keys requires $\lceil \frac{n}{b} \rceil$ hash computations. With the conjunctive search, there is an additional Bitmap subkey generation related to the x-term at the client side, as opposed to the single keyword search where only the Bitmap key for the s-term's Bitmap is generated. An important note is that the key generation can be parallelized and would thus reduce the computational overhead.

In Figure 2, we present results from single keyword and conjunctive searches where the keywords vary in frequency. The red line shows that increasing the frequency of the x-term does not affect the search, which is expected since the s-term determines how many bits to fetch from the x-terms. Increasing the frequency of the s-term during a single keyword search (black line) does not affect the search either, which was also expected, as the dominant computation during execution of a single keyword search is the bitmap key generation. Decrypting the s-term's Bitmap is dependent on the number of documents n. However, the blue line shows that the conjunctive search is highly dependent on choosing a good s-term, i.e. a keyword with a low frequency. When the frequency of the s-term increases, the search time dramatically increases. The increase is due to the fact that more key blocks need to be generated during decryption, and that more bits need to be fetched from the x-terms' Bitmaps. We emphasize that our scheme is intended to be used with, and requires, a low frequency s-term.

We also performed conjunctive searches on the Census data set with a varying number of x-terms, which can be found in Figure 3. It was demonstrated in Figure 2 that the frequency of the x-term does not affect the search and thus we let the x-terms in Figure 3 have

n	**50,000**	**100,000**	**150,000**	**200,000**	**250,000**	**299,285**
N	1,896,214	3,792,048	5,686,907	7,582,582	9,478,817	11,347,304
$\mathsf{HT}_{\mathsf{stag}}/\mathsf{HT}_{\mathsf{xtag}}$	0.12	0.16	0.19	0.20	0.21	0.22
Bitmaps	14	36	62	90	119	149
Est. TSet (BXT)	61	122	183	244	305	365

Table 2: Storage results in MB for the Census data set, for subsets from 50,000 to 299,285 rows, and the estimated storage of the TSet. Note that our estimation for the TSet's storage is based on the TSet instantiation procedure described in [6] where each entry in the TSet has a value (encryption of a document ID and encryption of a single bit β) and a label L of size equal to the security parameter λ. Thus the minimal size of each TSet entry is 257 bits. For more details, we refer the reader to the Appendix. Note that n denotes the number of documents while $N = \sum_i \mathsf{DB}(w_i)$ denotes the size (total number of keyword occurrences) of the Census database DB.

n	**100,000**	**200,000**	**300,000**	**400,000**	**500,000**
N	14,352,245	26,509,765	43,529,924	57,516,347	75,062,313
$\mathsf{HT}_{\mathsf{stag}}/\mathsf{HT}_{\mathsf{xtag}}$	7	10	17	25	29
Bitmaps	1582	4414	11446	22124	32513
Est. TSet (BXT)	461	851	1398	1848	2411

Table 3: Storage results in MB for the Enron data set, for subsets from 100,000 to 500,000 documents, and the estimated storage of the TSet. Note that our estimation for the TSet's storage is based on the TSet instantiation procedure described in [6] where each entry in the TSet has a value (encryption of a document ID and encryption of a single bit β) and a label L of size equal to the security parameter λ. Thus the minimal size of each TSet entry is 257 bits. For more details, we refer the reader to the Appendix. Note that n denotes the number of documents while $N = \sum_i \mathsf{DB}(w_i)$ denotes the size (total number of keyword occurrences) of the Enron database DB.

a varying frequency between roughly 4,000 and 100,000. This experiment shows that the number of x-terms in a conjunctive query has very little influence on the search time. As long as the s-term is well chosen, i.e. has low frequency, we conclude with these experiments that neither the frequency of the x-terms nor the number of x-terms affect the performance in a negative manner. We would like to emphasize that the aim of our experiments is to give an indication of how well our solution performs in terms of search time and more importantly demonstrate how it scales and that x-terms do not affect the performance. The latency experiments were run on a Lenovo T450 machine with a Intel i7-5600U CPU with 4 cores, each running at speed 3.2 GHz, and with a RAM size of 12 GB, and the storage experiments were run on a VPS with 20 virtual cores of Intel Xeon e5-2650lv3 at 1.80GHz and 60 GB of RAM. Worth to note is that the latency experiments only measured the time from that the client starts to process the query until it has the decrypted documents identifiers.

6. CONCLUSION

In this paper, we proposed a searchable encryption scheme that executes arbitrary Boolean queries of the form $w_1 \wedge f(w_2, \cdots, w_{n_t})$. Our scheme realizes Cash et al.'s BXT protocol [6] on the Bitmap index data structure. This is suitable for environments where computations and communications are very constrained. Moreover, for some large and dense datasets such as the Census dataset [16], our solution occupies lesser storage compared to Cash et al.'s BXT and OXT protocols [6].

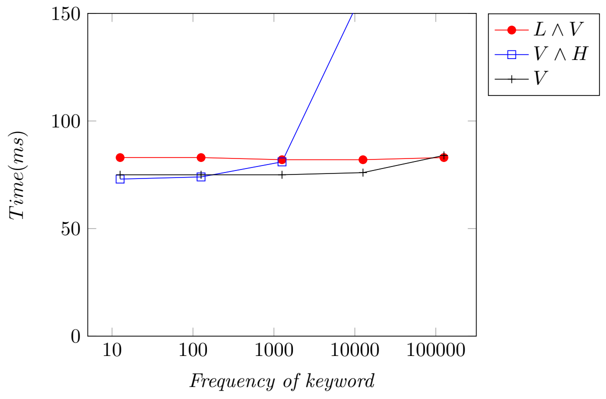

Figure 2: End-to-end execution time for the Census data set, 299,285 documents. H denotes a term with high frequency, L denotes a term with low frequency and V denotes the term that varies in frequency.

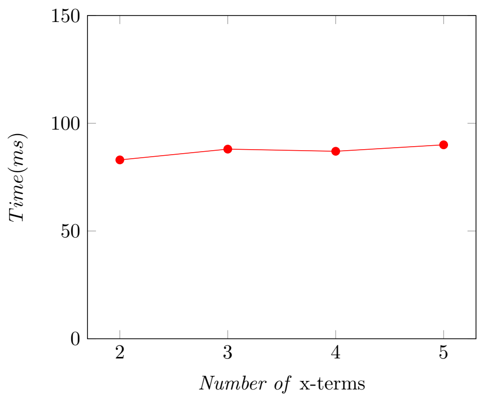

Figure 3: End-to-end execution time for the non-interactive conjunctive search on the complete Census data set with a growing number of x-terms. The s-term has low frequency and the frequency of the x-terms varies between 4,177 and 100,955.

Acknowledgments

We would like to thank the anonymous reviewers of SAC 2016 and CCSW 2016 for their valuable comments. This work was supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 644814, the PaaSword project within the ICT Programme ICT- 07-2014: Advanced Cloud Infrastructures and Services.

7. REFERENCES

[1] Enron e-mail data set. Available at https://www.cs.cmu.edu/~./enron/, [Last Accessed 2016-03-02].

[2] Lucas Ballard, Seny Kamara, and Fabian Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security*, pages 414–426. Springer, 2005.

[3] Christoph Bosch, Adrian Peter, Bram Leenders, Hoon Wei Lim, Qiang Tang, Huaxiong Wang, Pieter Hartel, and Willem Jonker. Distributed searchable symmetric encryption. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 330–337. IEEE, 2014.

[4] Jin Wook Byun, Dong Hoon Lee, and Jongin Lim. Efficient conjunctive keyword search on encrypted data storage system. In *Public Key Infrastructure*, pages 184–196. Springer, 2006.

[5] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptology ePrint Archive*, 2014:853, 2014.

[6] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[7] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[8] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010*, pages 577–594. Springer, 2010.

[9] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM, 2006.

[10] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[11] Eu-Jin Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[12] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*, pages 31–45. Springer, 2004.

[13] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[14] Florian Kerschbaum. Secure conjunctive keyword searches for unstructured text. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 285–289. IEEE, 2011.

[15] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. Distributed search over encrypted big data. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 271–278. ACM, 2015.

[16] Terran Lane and Ronny Kohavi. Census-income (kdd) data set, 2000. Available at http://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD), [Last Accessed 2016-03-02].

[17] Rasmus Pagh and Flemming Friche Rodler. *Cuckoo hashing*. Springer, 2001.

[18] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[19] Peter van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. *Secure Data Management: 7th VLDB Workshop, SDM 2010, Singapore, September 17, 2010.*

Proceedings, chapter Computationally Efficient Searchable Symmetric Encryption, pages 87–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

APPENDIX

A. THE TSET INDEX

The TSet is a new primitive index data structure presented in [6]. It is called a tuple set or TSet. It allows the association of a list of fixed size data tuples with each keyword in the database. The server retrieves the corresponding keyword list if the client provides the right token for the keyword. It has three functions which we describe in the following.

The first function, TSetSetup, processes an array $\mathbf{T}$ of lists of equal-length indexed by the set of unique keywords W. For each $w \in \mathsf{W}$, $\mathbf{T}[w]$ is a linked list $\mathbf{t}$ containing the encryptions of the corresponding document IDs. Note that when the TSet is used with BXT, only e, the encryption of a document ID ind is added to the linked lists but when it is used with OXT, e and another integer $y \in \mathbf{Z_p}$ are added to the linked lists. Note that in both schemes (BXT and OXT), each element added to the linked list has a random label, L, generated using a hash function. For more details about this we refer the reader to [6]. Thus, the length of $\mathbf{T}[w]$ is equivalent to the number of occurrences of w in the database. The security goal of T-Set is to hide the length of $\mathbf{T}[w]$ except for those $\mathbf{T}[w_i]$'s previously retrieved by the server as a reply to the client's queried w_i's. For more details, we refer the reader to [6] where it is shown that the only leakage of the T-Set instantiation as described in [6] is the total number of keyword occurrences in the database, namely, $N = \sum_{w \in \mathsf{W}} |\mathbf{T}[w]|$. The output of this function is the TSet data structure and the randomly chosen secret key K_T used by the TSetGetTag function to generate the secret token or trapdoor stag.

The second function, TSetGetTag, takes a secret key K_T and a keyword w as inputs and outputs stag $\to \bar{F}(K_T, w)$ where $\bar{F}$ is a PRF. It is used by the TSetSetup during the setup of the encrypted database at upload time and also by the client to generate the corresponding key-dependent token for each keyword.

The third function, TSetRetrieve, takes as input the TSet and stag and returns the data associated with the stag corresponding to the client's queried keyword.

B. ENRON RESULTS

Note that all the figures in this Section are produced using a slightly different previous variant of the BXT-Bitmap protocol described above. The previous variant used to send the short key K_{w_j} rather than storing its encryption in the hash tables and then decrypting it during the s-term keyword search. Figure 4 and Figure 5 shows the experimental results on the Enron dataset of the same experiments on the Census dataset shown in Figure 1 and Figure 2 respectively. We see similar

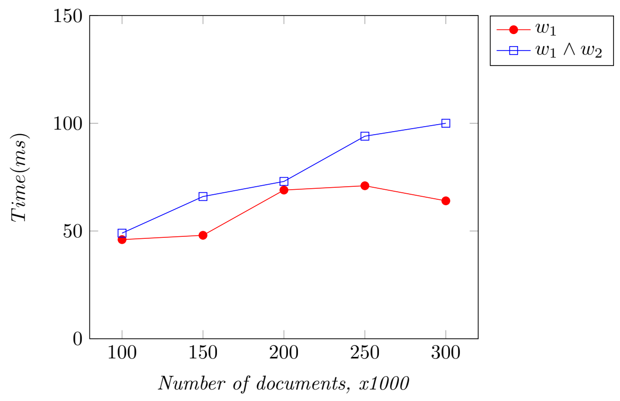

Figure 4: End-to-end execution time for the single keyword search and non-interactive 2-term conjunctive search on the Enron data set.

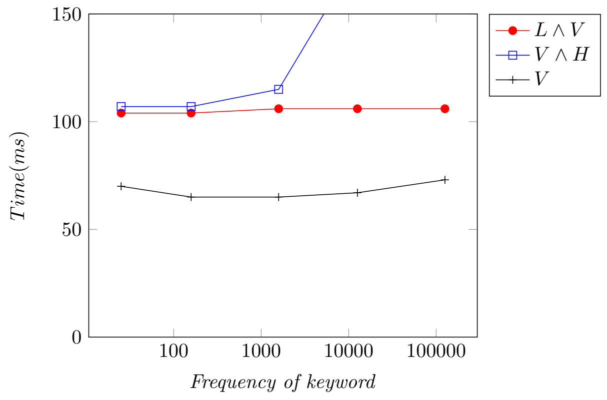

Figure 5: End-to-end execution time for the Enron data set, 300,000 documents. H denotes a term with high frequency, L denotes a term with low frequency and V denotes the term that varies in frequency.

results for the Enron corpus, which show that our proposed scheme's latency does not depend on the data set. However, as previously stated, our scheme is optimal for dense data sets such as the Census data set.