Trusted Boolean Search on Cloud Using Searchable Symmetric Encryption

Cong Zuo
School of Computer and Information Engineering
Zhejiang Gongshang University
Zhejiang, 310018, P.R. China
Email: zuocong10@gmail.com

James Macindoe, Siyin Yang
Ron Steinfeld, Joseph K. Liu*
Faculty of Information Technology
Clayton Campus, Monash University, VIC 3800, Australia
Email: joseph.liu@monash.edu

***Abstract*—A Searchable Symmetric Encryption (SSE) scheme allows a server to search a user's data without having to decrypt the data. This provides the user with a high degree of privacy and is particularly useful when data is stored on Cloud. Numerous SSE schemes have already been proposed and while most have excellent security properties, few meet high performance requirements and most only support searching for a single keyword at a time. The SSE scheme of Cash et al. (CRYPTO 2013) is notable for its high efficiency on restricted forms of Boolean queries, but has low efficiency for, or does not support, other common forms of boolean queries. In this paper, we propose a generalization and optimization of the Cash et al. SSE scheme, which extends to support a much larger class of boolean queries, and performs no worse, and often with much higher efficiency than the Cash et al. scheme for the remaining queries.**

***Index Terms*—SSE, Boolean Query, Cloud**

I. Introduction

The privacy of confidential data is highly valued by business, government and individuals. It is becoming increasingly common for data to be hosted off-site, especially with the rise of cloud computing [12], [13], [16], [22], [21], [18]. It is very useful in many scenarios, such as healthcare [23], [15] and smart grid system [1]. However, hosting providers often cannot be trusted to respect the privacy of the data they host [7], especially in the face of malicious insiders. A simple solution is to encrypt the data before uploading it to the cloud. However, this would prevent the data from being searched [17], [14]. For example, a user may wish to use their mobile phone to search their email. The cloud server will not be able to identify which documents match the search query if the user's email data is encrypted using standard encryption techniques.

Searchable symmetric encryption (SSE) solves this problem by providing a way for encrypted data to be searched securely. However, all SSE schemes must trade off between security, performance and functionality. Most previous schemes have focused on security, to the detriment of performance and functionality, which has made many of them impractical for widespread use. Functionality includes such features as boolean search, range search and key revocation. The security of a scheme can be defined in terms of its leakage, and formal proofs of the security can be produced. Most current schemes leak very little. However, the performance of schemes varies greatly, with the best performing schemes having the same asymptotic performance as unencrypted search schemes. However, search times can still be slow in practice, especially when operating on very large amounts of data, as is becoming increasingly common with the rise of "Big Data".

If SSE schemes are to be widely deployed, they must have a feature set that is comparable with unencrypted databases. Boolean search is a very widely used feature of databases. Any query which contains an AND, OR or NOT requires boolean support to execute. Despite its importance, there are almost no existing schemes that can handle any boolean features. In fact, most existing SSE schemes can only support single keyword searches and so could not run a multi-keyword search such as "male teacher".

Recently, Cash et al. [5] (named $\texttt{CJJ}^{+}13$ afterwards) published an efficient scheme which can handle handle multi-keyword conjunctions (Computer AND Science) and a limited set of boolean queries. Namely, in $\texttt{CJJ}^{+}13$, only queries of the form $w_s \wedge \phi(w_1, \ldots, w_n)$ are supported where ϕ is any boolean expression. This is implemented in $\texttt{CJJ}^{+}13$ by the client sending to the server a 'search token' (called an stag) related to the keyword w_s (called the 's-term'), which allows the server to retrieve the set $\mathbf{DB}[w_s]$ of all database documents containing w_s. In addition, the client sends 'intersection tokens' (called xtraps) related to w_s and each of the remaining query keywords $w_1, \ldots, w_n$ (called 'x-terms'), which allow the server to *filter* the set $DB(w_s)$ to determine the subset of documents that also contains each of the keywords w_i, returning only those documents that satisfy the Boolean expression $\phi(w_1, \ldots, w_n)$. Therefore, since the query is evaluated by looking up all documents containing the 's-term' w_s, the server computation time is proportional to $|\mathbf{DB}[w_s]|$, i.e. the number of database documents containing w_s. For many typical query expressions, such as "$Male \wedge (Australian \vee Chinese)$", where $|\mathbf{DB}[w_s]|$ is potentially very large, this may be very inefficient. Similarly, to support boolean queries that do not fit the restriction form $w_s \wedge \phi(w_1, \ldots, w_n)$, such as "$Australian \vee Chinese$",

* Corresponding author. The last author has a second affiliation: Fujian Provincial Key Laboratory of Network Security and Cryptology, Fujian Normal University, Fuzhou, Fujian, China, 350117.

$\texttt{CJJ}^+13$ suggest to modify such queries into the supported form "$Person \wedge (Australian \vee Chinese)$", where "Person" is a keyword that matches all database records. Although this allows $\texttt{CJJ}^+13$ to support any booklean query, it is also very inefficient, requiring the server to scan all database records to process the query. We therefore wish to develop a more general and more efficient boolean search algorithm to deal with these problems.

Our Contributions. In this paper, we propose a trusted boolean search on cloud using searchable symmetric encryption scheme which extends the scheme of $\texttt{CJJ}^+13$ to support more general boolean formulas, while also significantly improving the efficiency over the scheme of $\texttt{CJJ}^+13$ for many of the queries that they do support. We also give a security proof for our generalized scheme that specifies the information leakage to the server. The main idea behind our generalized and more efficient $\texttt{CJJ}^+13$ scheme is an algorithm to optimize the selection of 'search tokens' and 'intersection tokens' sent by the client in order to minimize the server computation time for processing a given query. For example, for processing queries such as $Male \wedge (Australian \vee Chinese)$", instead of sending one 'search token' for 'Male' and two 'intersection tokens' for 'Australian' and 'Chinese' as in $\texttt{CJJ}^+13$, the client also considers the alternative of sending two 'search tokens': one for 'Australian' and one for 'Chinese', and 'intersection tokens' for 'Male' with respect to 'Australian' and 'Chinese'. Using this alternative choice of tokens, the server would look up all documents containing 'Australian' and filter them to those containing 'Male', and similarly look up all documents containing 'Chinese' and filter them to those containing 'Male', merging the results. With this alternative, the server computation time is reduced to be proportional to the total number of documents $|\mathbf{DB}[`Australian']|+|\mathbf{DB}[`Chinese']|$ containing 'Australian' or 'Chinese', which may be much lower than the number of documents $|\mathbf{DB}[`Male']|$ containing 'Male' needed to be scanned in the first alternative. Our scheme generalizes this idea, and gives an algorithm for the client to find the optimal choice of tokens to minimize the estimated number of documents retrieved by the server and hence server computation time, for any given boolean query. Therefore, server computation time in our scheme is expected to be no worse, and often much lower, than in the scheme of $\texttt{CJJ}^+13$. Although the alternative choice of tokens in our scheme may increase the information leakage due to the larger number of stags sent by the client, this leakage is still of a limited form similar to that in the scheme of $\texttt{CJJ}^+13$, as characterised in our security proof, and we believe is still acceptable for many applications.

A. Related Works

Many Searchable Encryption (SE) schemes have been proposed. They are reviewed here, with a focus on the schemes that are the direct predecessors of our own.

1) Linear Search: In 2000, Song et al. proposed the first SE scheme in [20]. In their scheme, the document's cipher text has a structure which allows the server to identify if a given block is the queried keyword. This means the entire document collection must be read to process a query, which is much less efficient than the index based search schemes.

2) Index Based Search: Using reverse indexes allows keyword searches to be performed in constant time and are therefore highly efficient. A number of index based schemes have been proposed, each more efficient than its predecessor. The first secure encrypted indexes were proposed by [9]. In his scheme, each document has a corresponding bloom filter. Each word in a document is inserted into the document's bloom filter. To search a document for a word, it is a simple process of checking if the document's bloom filter has the search term. This allows a single document to be searched in $O(1)$ time but still requires each document to be checked in turn, resulting in a complexity of $O(d)$ where d is the number of documents in the collection. Later, Chang and Mitzenmacher [6] proposed a scheme with similar complexity properties in which an index is stored for each file.

Curtmola et al. [8] were the first to propose using an index that operated on the whole document collection. They present two schemes, called SSE-1 and SSE-2. They have slightly different security and performance characteristics. SSE-1 makes use of an array A and a hashtable T. For each keyword, a linked list of search results is generated. This list is then stored in A such that each entry in A contains a single result and the address in A of the next result. T is used to locate the first entry in A for each keyword. This scheme is not secure against adaptive adversaries and it is difficult to parallelise its search routine. In contrast, SSE-2 is secure against adaptive adversaries and is easily parallelised. SSE-2 only requires a hash table T. The results for a keyword w are each given a label derived from w and a counter. These labels are used as the keys for T and the corresponding hash table values are encrypted versions of the document ids. [8] have the client send the complete set of hash table keys to the server e.g. $(F_k(\text{“}foo1\text{”}), F_k(\text{“}foo2\text{”}), \dots)$ where F_k is a hash function. Later, $\texttt{CJJ}^+13$ outline a similar scheme based on SSE-2 with lower communication requirements that they call Single Keyword Search (SKS). The client instead only sends the server a hashed version of the search keyword e.g. $stag = F_k(\text{“}foo\text{”})$. The hash table keys are derived from this token hashed together with the counter e.g. $(H(stag||1), H(stag||2), \dots)$ where H is a hash function. This scheme is highly efficient and so will form the basis of our own work.

3) Alternative Approaches: Oblivious RAM (ORAM) allows for searchable encryption with no leakage and a completely general set of features (including boolean search) [19], [10]. However, it is far less efficient than the SE solutions considered so far. [10] present two solutions, one with $O(\log^3 N)$ search complexity that requires $O(\log N)$ rounds of communication and one that operates in only two communication rounds but has $O(\sqrt{N})$ search complexity, where N is the size of the database. The most efficient SSE schemes take only a single round of communication and have a search complexity of $O(1)$.

SE has been studied extensively in the public key setting.

Schemes based on public key cryptography allow any user with the public key to insert data but only allow the user with the private key to search it. However, the use of public key cryptography makes the schemes less efficient than SSE. The first such scheme was proposed by [3]. A scheme which hides the access pattern was proposed by [4], but it requires $O(\sqrt{N})$ search time. Efficient searchable encryption (ESE) [2] is a public key based system which has asymptotically ideal search time but weaker security guarantees.

Private information retrieval (PIR) [11] allows search on unencrypted data without revealing the access pattern. However, this is not appropriate for our use case where the data itself is confidential. It is also generally less efficient since in order to hide the access pattern, it must touch every document when processing a query (or else the server would know only the accessed documents were of interest).

II. Definition

We use the definition of semantic security for SSE as $\texttt{CJJ}^{+}13$, as shown in **Definition II.1**.

Definition II.1. *Let* $\Pi = (\text{EDBSetup}, \text{Search})$ *be an SSE scheme and let* $\mathcal{L}$ *be an algorithm. For efficient algorithms* A *and* S*, we define experiments (algorithms)* $\mathbf{Real}_A^\Pi(\lambda)$ *and* $\mathbf{Ideal}_{A,S}^\Pi(\lambda)$ *as follows:*

$\mathbf{Real}_A^\Pi(\lambda)$ *:* $A(1^\lambda)$ *chooses* $\mathbf{DB}$ *and a list of queries* $\mathbf{q}$*. The experiment then runs* $(K, \mathbf{EDB}) \leftarrow \text{EDBSetup}(\mathbf{DB})$*. For each* $i \in \mathbf{q}$*, it runs the* Search *protocol with client input* $(K, \mathbf{q}[i])$ *and server input* $\mathbf{EDB}$ *and stores the transcript and the client's output in* $\mathbf{t}[i]$*. Finally the game gives* $\mathbf{EDB}$ *and* $\mathbf{t}$ *to* A*, which returns a bit that the game uses as its own output.*

$\mathbf{Ideal}_{A,S}^\Pi(\lambda)$ *:* $A(1^\lambda)$ *chooses* $\mathbf{DB}$ *and a list of queries* $\mathbf{q}$*. The experiment then runs* $S(\mathcal{L}(\mathbf{DB}, \mathbf{q}))$ *and gives its output to* A*, which returns a bit that the game uses as its own output.*

We say that Π *is* $\mathcal{L}$*-semantically-secure against non-adaptive attacks if for all efficient adversaries* A *there exists an algorithm* S *such that* $\Pr[\mathbf{Real}_A^\Pi(\lambda) = 1] - \Pr[\mathbf{Ideal}_{A,S}^\Pi(\lambda) = 1] \leq \text{negl}(\lambda)$.

Now we give the definition of our $\mathbf{XSet}$ Data Structure in Algorithm 1.

III. Our Construction

Our proposal aims to develop an efficient general boolean search protocol.The best existing scheme only supports queries of the form $w_1 \wedge \phi(w_2, \ldots, w_n)$ [5]. Due to lack of space, we do not review the schemes SKS,BXT and OXT of [5] here, but only describe our new generalized scheme. We refer the reader to [5] for a description of the schemes SKS, BXT, OXT. In this section, a new search scheme is presented which supports a much larger class of queries and is also often more efficient than the scheme of [5] even for the queries they do support.

Algorithm 1 Data structure setup code, including the oblivious $\mathbf{XSet}$. The singly boxed code is used in BBS and the doubly boxed code is used in OBS

function Setup($\mathbf{DB}$)
  K_S, K_{ME}, K_X $\boxed{\boxed{, K_I, K_Z}} \xleftarrow{\$} \{0,1\}^\lambda$
  $\mathbf{EDB} \leftarrow \{\}$
  $\mathbf{XSet} \leftarrow \emptyset$
  $(\text{ID}_i, W_i) \leftarrow \mathbf{DB}$
  for $w \in W$ **do**
    $\text{stag} \leftarrow F(K_S, w)$
    $K_E \leftarrow F(K_{ME}, w)$
    $\boxed{\text{xtrap} \leftarrow F(K_X, w)}$
    $c \leftarrow 1$
    $\boxed{\mathbf{EDB}[l] = e}$
    $\boxed{\mathbf{XSet} \leftarrow \mathbf{XSet} \cup \{F(\text{xtrap}, \text{ID})\}}$
    for ID $\in \mathbf{DB}[w]$ **do**
      $l \leftarrow F(\text{stag}, c)$
      $e \leftarrow \text{Enc}(K_E, \text{ID})$
      $c \leftarrow c + 1$
      $\boxed{\boxed{\begin{array}{l}\text{xind} \leftarrow F_p(K_I, \text{ID}) \\ z \leftarrow F_p(K_Z, w||c) \\ y \leftarrow \text{xind} \cdot z^{-1} \\ \text{xtag} \leftarrow g^{F_p(K_X, w) \cdot xind} \\ \mathbf{EDB}[l] = (e, y) \\ \mathbf{XSet} \leftarrow \mathbf{XSet} \cup \{\text{xtag}\}\end{array}}}$
    end for
  end for
  return K_S, K_{ME}, K_X $\boxed{\boxed{, K_I, K_Z}}$ to the client and $\mathbf{EDB}$ and $\mathbf{XSet}$ to the server
end function

A. Boolean Search

The main innovation required to implement a more general and efficient boolean algorithm is to allow multiple stags to be sent with each query. Thus, for example, with the query "$Male \wedge (Australian \vee Chinese)$", one could send the stags for both "$Australian$" and "$Chinese$" and the xtrap for "$Male$" which would then require only $|\mathbf{DB}[\text{"}Australian\text{"}]| + |\mathbf{DB}[\text{"}Chinese\text{"}]|$ look ups, which is likely less than the alternative of $|\mathbf{DB}[\text{"}Male\text{"}]|$. The issue then is choosing the optimal set of stags to send.

As boolean queries are processed, they pass through three different representations: ϕ, btree and xbtree. $\phi(\mathbf{w})$ is the boolean expression provided by the user, such as

$$\phi(IT, Clayton, Caulfield) = IT \wedge (Clayton \vee Caulfield)$$

Note that ϕ is parameterised by $\mathbf{w}$ in our notation. This is important because the function ϕ is leaked to the server, but not the actual words in the query.

In order to perform computation on the boolean query, the client transforms it into a tree data structure: the btree. Since the btree contains the actual search terms, the client cannot send it to the server. Instead, it sends the xbtree which replaces

the leaves of the btree with xtraps or xtokens (for BXT/BBS and OXT/OBS respectively). A leaf which previously contained $\mathbf{w}[\alpha]$ will now contain either $\mathrm{xtrap}[\alpha]$ or $\mathrm{xtoken}[\alpha, \cdot, \cdot]$. We assume there is a function MAKE_XBTREE$(\mathrm{btree}, \mathrm{xtoken})$ which performs this transformation, but we do not specify it.

In a conjunction, any term can be chosen as the sterm since all of the documents which match the conjunction must by definition match each term in the conjunction. Thus, the stag for any of the terms will produce a superset. This superset is then filtered down using the xtraps. In BXT, since it only supports conjunction, this is done simply be checking that each document contains all of the xterms. This is can be extended to the boolean case. However, it is more. Our formulation is given as a recursive algorithm CHOOSE_STAGS in Algorithm 2. The algorithm processes the boolean expression as a tree.

Algorithm 2 Function to choose stags. Used on the client in BBS and OBS.

function CHOOSE_STAGS($K_S, K_E, node$) // Returns the keywords that should be used as stags and the heuristic score for doing so
  if $node$ is a leaf **then**
    return $node.w, h(node.w)$
  else if $node.op$ is NOT **then**
    return nil, inf
  else if $node.op$ is AND **then**
    $stags_{min} \leftarrow nil$
    $KEs_{min} \leftarrow nil$
    $cost_{min} \leftarrow \inf$
    for $child \in node.children$ **do**
      $stags, KEs,$
      $cost \leftarrow Choose\_stags(K_S, K_E, child)$
      if $cost < cost_{min}$ **then**
        $stags_{min} \leftarrow stags$
        $KEs_{min} \leftarrow KEs$
        $cost_{min} \leftarrow cost$
      end if
    end for
    return $stags_{min}, KEs_{min}, cost_{min}$
  else if $node.op$ is OR **then**
    $all\_stags \leftarrow nil$
    $all\_KEs \leftarrow nil$
    $all\_costs \leftarrow 0$
    for $c \in node.children$ **do**
      $stags, KEs, cost \leftarrow Choose\_stags(K_S, K_E, c)$
      append $stags$ to all_stags
      append KEs to all_KEs
      $all\_costs \leftarrow all\_costs + cost$
    end for
    return $all\_stags, all\_KEs, all\_costs$
  end if
end function

In a disjunction (OR), the final results could match any of the disjoined terms. So for example, for an expression such as "w1 OR w2 OR w3", where each of the children are keywords, the stags for each of the children must be sent to the server. Similarly, when the children are themselves boolean expressions, the stags allowing each child to be evaluated must be sent to the server. So for an expression such as "(w1 AND w2) OR (w3 AND w4)", the stag for either w1 or w2 must be sent, as well as the stag for either w3 or w4.

Evaluating logical negation (NOT) is not easy to do with an index, since indexes only allow non-negated terms to be looked up. However, if the NOT is a child of an AND, such as in "(NOT w1) AND w2 AND w3", then the expression as a whole can still be evaluated since the AND allows us to simply send stags for a different child. However, if the NOT is the root of the tree, or else it does not have an AND above it, such as "(NOT w1) OR w2 OR w3", then our algorithm cannot evaluate the expression using an index. This is discussed further in section III-A1.

Once a document id has been retrieved from the $\mathbf{EDB}$, the server checks if it matches the entire boolean expression using EVALUATE_EXPR, shown as algorithm 3. EVALUATE_EXPR is a simple recursive algorithm the operates over the xbtree. When it reaches a leaf node, it evaluates if the document contains that term using the $\mathbf{XSet}$. This is done differently in the basic and oblivious versions of the protocol.

Algorithm 3 Server side function to evaluate if an expression is true for a document. The singly boxed code is included in BBS and the doubly boxed code is included in OBS.

function EVALUATE_EXPR($node$ [, ID] [[, i, c, y]])
  if $node$ is a leaf **then**
    [**return** $F(node.xtrap, \mathrm{ID}) \in \mathbf{XSet}$]
    [[**return** $node.xtoken[i, c]^y \in \mathbf{XSet}$]]
  else if $node.op$ is NOT **then**
    return NOT Evaluate_Expr($node.children[1]$, ID)
  else if $node.op$ is AND **then**
    for $child \in node.children$ **do**
      if not Evaluate_Expr($child$, ID) **then**
        return False
      end if
    end for
    return True
  else if $node.op$ is OR **then**
    for $child \in node.children$ **do**
      if Evaluate_Expr($child$, ID) **then**
        return True
      end if
    end for
    return False
  end if
end function

1) Appropriate Heuristics: A heuristic function $h(w)$ must be defined for every keyword user may include in a search. It should provide a score of the estimated cost of looking that term up in the index. There are many approaches that could be

taken for designing such a heuristic. A simple one would be to preprocess the $\mathbf{DB}$ during SETUP by counting the number of occurrences of each word and then storing this data with the client. We would then have h defined as $h(w) = |\mathbf{DB}[w]|$. Note that this definition still works if $w \notin W$, since this would give $|\mathbf{DB}[w]| = 0$ which is an appropriate score. Another option would be to base the heuristic on the frequency of words in English, or whichever language is relevant.

B. Basic Boolean Search (BBS) protocol

Basic Boolean Search (BBS) is an extension of BXT to allow boolean search, following the design of section III-A. It is specified in algorithm 4, with calls to algorithms 1, 2 and 3. The client sends the server multiple stags and an xbtree based on the xtrap values, as discussed in section III-A. The SERVER_SEARCH function then proceeds as per BXT except that EVALUATE_EXPR is used to check if a document matches the expression instead of performing the check with the xtraps directly. As with BXT, the client must send the server the K_E values, allowing them to decrypt the document ids. This is resolved in OBS.

Algorithm 4 Basic Boolean Search (BBS) Protocol

```
function CLIENT_SEARCH(btree)
    s, cost ← CHOOSE_STAGS(btree)
    stags ← ∅
    for w ∈ s do stags ← stags ∪ {F(K_S, w)} end for
    K_E s ← ∅
    for w ∈ s do K_E s ← K_E s ∪ {F(K_ME, w)} end for
    w ← leaves of btree
    for α ∈ [|w|] do
        xtrap[α] ← F(K_X, w[α])
    end for
    xbtree ← MAKE_XBTREE(btree, xtrap)
    return SERVER_SEARCH(stags, K_E s, xbtree)
end function

function SERVER_SEARCH(stags, K_E s, xbtree)
    for stag, K_E ∈ stags, K_E s do
        c ← 0
        l ← F(stag, c)
        while l ∈ EDB do
            e ← EDB[l]
            ID ← Dec(K_E, e)
            if EVALUATE_EXPR(xbtree, ID) then
                output ID
            end if
            c ← c + 1
            l ← F(stag, c)
        end while
    end for
end function
```

Unlike in BXT, in BBS, xtraps are sent for every term, including those for which stags are sent. Since the server can retrieve every document matching one of the stags, it doesn't strictly require the xtrap to check which documents match that term. However this would require it to keep all the stag results in memory and perform checks against them. For ease of implementation, we send the additional xtrap, despite it slightly increasing leakage. This problem is entirely resolved in LLOBS, where the minimum amount is leaked.

C. Oblivious Boolean Search (OBS) protocol

Oblivious Boolean Search (OBS) is an extension of OXT to allow boolean search, following the design of section III-A. It is specified in algorithm 5. The client calculates the xtoken for every combination of stag and other search keyword in the document. Again, as in BBS, this includes xtokens for terms that have an stag sent. These unnecessary xtoken values are eliminated in LLOBS. The rest of the search proceeds as in BBS, except the server does not need to decrypt the document ids to run EVALUATE_EXPR and so returns them to the client still encrypted. As in our presentation of OXT, we use have the client send an upper bound T_c number of xtokens. Again, we could instead have the server tell the client when to stop sending xtokens.

D. Low Leakage Oblivious Boolean Search (LLOBS) protocol

For simplicity of implementation, OBS sends xtokens for every combination of sterm and keyword. However, often many of these xtokens will never be used and so constitute unnecessary leakage to the server. For example, say in the expression "$IT \wedge (Clayton \vee Caulfield)$", stags were sent for "$Clayton$" and "$Caulfield$". Then, for every document id retrieved from the $\mathbf{EDB}$ using the stag for "$Clayton$", xtokens are only needed for "IT", since we already know the document matches "$Clayton \vee Caulfield$". This similarly applies to documents retrieved using "$Caulfield$". In cases like these, the unnecessary xtokens could be replaced with random group elements without compromising functionality or performance. Indeed, the performance would be improved since we have decreased the number of $\mathbf{XSet}$ look ups that must be performed.

The general rule for determining when an xtoken is not needed is to observe a large expression tree. If the tree was evaluated by starting at the sterm node and gradually propagating upwards, checking xterms only as needed, then the other children of any OR nodes would not need to be evaluated. If execution reaches an AND node, all its children need to be evaluated for it to be judged TRUE. For an OR node, if the child with the sterm is true, execution can immediately proceed upwards without evaluating other children.

Say the child of an OR that contained the sterm evaluated as false, such as in an expression of the form "(sterm AND FALSE) OR TRUE". Then it would seem that the xtokens for its other children would be necessary in order to continue execution. However, in this case, execution can stop once it is determined the OR is false. This can be seen by considering the class of trees which do not contain any NOTs. If, for a given document, an OR node is found to be false, either the document does not match the whole expression, or if it does,

Algorithm 5 Oblivious Boolean Search (OBS) Protocol. The boxed code differs from BBS.

function CLIENT_SEARCH(btree)
  $\mathbf{s}, cost \leftarrow$ Choose_Stags(btree)
  stags $\leftarrow \emptyset$
  for $w \in \mathbf{s}$ **do** stags $\leftarrow$ stags $\cup \{F(K_S, w)\}$ **end for**
  $\mathbf{w} \leftarrow$ leaves of btree
  for $\alpha \in [|\mathbf{w}|]$ **do**
    for $i \in [|\mathbf{s}|]$ **do**
      for $c \in [T_c]$ **do**
        $\text{xtoken}[\alpha, i, c]$
        $\leftarrow g^{F_p(K_Z, \mathbf{s}[i]||c) \cdot F_p(K_X, \mathbf{w}[\alpha])}$
      end for
    end for
  end for
  xbtree $\leftarrow$ MAKE_XBTREE(btree, xtoken)
  for $e \in$ SERVER_SEARCH(stags, xbtree) **do**
    output $\text{Dec}(K_E, e)$
  end for
end function

function SERVER_SEARCH(stags, xbtree)
  $i \leftarrow 1$
  for stag $\in$ stags **do**
    $c \leftarrow 0$
    $l \leftarrow F(stag, c)$
    while $l \in$ **EDB do**
      $(y, e) \leftarrow \mathbf{EDB}[l]$
      if EVALUATE_EXPR(xbtree, i, c, y) **then**
        output e
      end if
      $c \leftarrow c + 1$
      $l \leftarrow F(stag, c)$
    end while
    $i \leftarrow i + 1$
  end for
end function

this will be found by retrieving it again using a different stag. This may not hold if there are NOTs, but since stags are never sent for subtrees below NOTs, this is not a problem.

Thus, in general, xtokens for xterm α are not needed for sterm i if α and i have an OR as their common ancestor. Conversely, xtokens are only needed when the common ancestor of the xterm and sterm is an AND node. This can be implemented in CHOOSE_STAGS by replacing xtokens with random elements of G when they are not needed. It also requires modifications to be made to EVALUATE_EXPR to stop execution in the above stated conditions. Namely, when an OR with the sterm within its children is evaluated as FALSE.

IV. SECURITY ANALYSIS

A. Leakage Analysis

1) Basic Boolean Search (BBS) Protocol: The leakage of BBS is a generalisation of the leakage of BXT. Most components have additional dimensions added to account for there being multiple search terms and sterms in each query. A number of new leakage components are also added to allow for boolean search.

- $N = \sum_{i=1}^{d} |W_i|$ as defined for SKS
- $\bar{\mathbf{s}}[t, i] \in \mathbb{N}^{T \times I}$ is the same as in SKS, except we generalize to two dimensions to match the generalisation of $\mathbf{s}$.
- $\bar{\mathbf{x}}[t, i] \in \mathbb{N}^{T \times A}$ is the same as in BXT, except we generalize to two dimensions to match the generalisation of $\mathbf{w}$.
- $\text{SX}[t_1, i, t_2, \alpha] = \mathbf{DB}[s[t_1, i]] \cap \mathbf{DB}[x[t_2, \alpha]]$ is the same as in BXT, but has increased dimensions to match $\mathbf{s}$ and $\mathbf{w}$.
- $\text{SRP}[t, i] = \mathbf{DB}[s[t, i]]$ is the same as in BXT, except we generalize to two dimensions to match the generalisation of $\mathbf{s}$.
- $\text{XT}[t] = |\mathbf{w}[t]|$ is the number of terms in each query
- $\text{ST}[t] = |\mathbf{s}[t]|$ is the number of sterms in each query
- $\phi[t]$ is the form of the boolean expression, as discussed in section III-A

2) Oblivious Boolean Search (OBS) Protocol: As in OXT, this leakage function is larger than the true leakage for the same reasons. In the actual protocol, the server never has access to the unencrypted indices, so RP, SRP and IP overstate the leakage that they represent. As for OXT, this does not render the proof based on this leakage function invalid.

- $N = \sum_{i=1}^{d} |W_i|$ is the same as in SKS.
- $\bar{\mathbf{s}}[t, i] \in \mathbb{N}^{T \times I}$ is the same as in SKS, except we generalize to two dimensions to match the generalisation of $\mathbf{s}$.
- $\text{SP}[\sigma]$ is the same as in SKS, except we index it by the values of $\bar{\mathbf{s}}$, instead of the query number t. So $\text{SP}[\bar{\mathbf{s}}[t, i]] = |\mathbf{DB}[\mathbf{s}[t, i]]|$.
- $\text{RP}[t, i, \alpha] = \mathbf{DB}[\mathbf{s}[t, i]] \cap \mathbf{DB}[\mathbf{w}[t, \alpha]]$ where we require $\mathbf{s}[t, i] \neq \mathbf{w}[t, \alpha]$. This is a generalisation of the Results Pattern in OXT. It reveals the intersection of any sterm with any other term in the same query, since xtokens are sent for every stag.
- $\text{SRP}[t, i] = \mathbf{DB}[\mathbf{s}[t, i]]$ represents the results corresponding to any stag.
- $\text{IP}[t_1, t_2, i, j, \alpha, \beta] = \mathbf{DB}[\mathbf{s}[t_1, i]] \cap \mathbf{DB}[\mathbf{s}[t_2, j]]$ if $\mathbf{s}[t_1, i] \neq \mathbf{s}[t_2, j]$ and $\mathbf{w}[t_1, \alpha] = \mathbf{w}[t_2, \beta]$. Otherwise, $\text{IP}[t_1, t_2, i, j, \alpha, \beta] = \emptyset$ is a generalisation of the OXT IP
- $\text{XT}[t] = |\mathbf{w}[t]|$ is the number of terms in each query
- $\text{ST}[t] = |\mathbf{s}[t]|$ is the number of sterms in each query
- $\phi[t]$ is the form of the boolean expression, as discussed in section III-A

3) Low Leakage Oblivious Boolean Search (LLOBS) Protocol: The leakage of $N, \bar{\mathbf{s}}, \text{SP}, \text{SRP}, \text{XT}, \text{ST}$ and ϕ are exactly as in OBS. However, the leakage captured by RP and IP is decreased. No information is leaked in the cases where the xtoken values are replaced with random group elements. This is specified precisely below.

- $RP[t,i,\alpha] = \mathbf{DB}[\mathbf{s}[t,i]] \cap DB[\mathbf{w}[t,\alpha]]$ if LOWEST_ANCESTOR$(\mathbf{s}[t,i], \mathbf{w}[t,\alpha]) = AND$
- $IP[t_1,t_2,i,j,\alpha,\beta] = \mathbf{DB}[\mathbf{s}[t_1,i]] \cap \mathbf{DB}[\mathbf{s}[t_2,j]]$ if $\mathbf{s}[t_1,i] \neq \mathbf{s}[t_2,j]$ and $\mathbf{w}[t_1,\alpha] = \mathbf{w}[t_2,\beta]$ and LOWEST_ANCESTOR$(\mathbf{s}[t_1,i], \mathbf{w}[t_1,\alpha]) = AND$ and LOWEST_ANCESTOR$(\mathbf{s}[t_2,j], \mathbf{w}[t_2,\beta]) = AND$. Otherwise, $IP[t_1,t_2,i,j,\alpha,\beta] = \emptyset$.

B. Security Proof

Theorem OBS is $\mathcal{L}_{obs}$-semantically-secure against non-adaptive attacks where $\mathcal{L}_{obs}$ is defined in section IV-A2, assuming that the DDH assumption holds in G, that F and F_p are secure PRFs, that $(\mathrm{Enc}, \mathrm{Dec})$ is an IND-CPA secure symmetric encryption scheme.

Our proof is structured in a similar way to the one in [5]. A series of games $G_0, \ldots, G_7$ and a simulator S are described. The first game G_0 is designed to have the same distribution as $\mathbf{Real}^{\Pi}_A(\lambda)$ (assuming no false positives). The simulator S produces an output consistent with G_7. By showing that the distribution of each game is negligibly different from the previous game, it can be shown that the simulator S meets the requirements of Definition II.1, proving the theorem. Our proof differs from that of [5] by not including a game equivalent to their G_6. This is because we use a dictionary instead of their TSet data structure, which we handle in earlier games. Their TSet is a specification of a hash table with an API designed for their purposes. We use a standard dictionary instead for familiarity to the reader and to save specifying additional data structures and leakages.

Game 0: G_0 is a slightly modified version of the real game. Its code is derived from the actual OBS protocol, as shown in Algorithms 1. INITIALIZE first runs code identical to SETUP in Algorithm 1, except the $\mathbf{XSet}$ calculation is moved to a separate function, XSET_SETUP, to assist in the presentation of changes in later games. INITIALIZE then generates the transcript using the GEN_TRANS function, which calculates the client output in the same way as CLIENT_SEARCH in Algorithm 5 and then calls SERVER_SEARCH to get the server output. Finally, it directly calculates the results to the query by looking up each each word using $\mathbf{DB}$ and taking set unions and intersections according to ϕ (we use notation such that $\phi(\mathbf{DB}[\mathbf{w}]) = \phi(\mathbf{DB}[w_1], \ldots, \mathbf{DB}[w_n])$). Since the input vectors to GEN_TRANS consist of data for only a single query, we denote them as $\mathbf{s_t} = \mathbf{s}[t,\cdot]$ and $\mathbf{w_t} = \mathbf{w}[t,\cdot]$.

Two minor bookkeeping changes are also made. First, the order in which the document ids are used for each keyword w are recorded in an array $\mathrm{WPerms}[w]$. The order is chosen as a random permutation, which matches the real game. The second change is to record the stag values after they are first computed and look them up on subsequent uses, rather than recomputing them.

None of these changes alter the distribution of the game from that of $\mathbf{Real}^{\Pi}_A(\lambda)$. However, we assume no false positives occur. Thus, $\Pr[G_0 = 1] \leq \Pr[\mathbf{Real}^{\Pi}_A(\lambda) = 1] + \mathrm{negl}(\lambda)$.

Game 1: Compared to G_0, G_1 replaces the PRFs F and F_p with random functions. Since $F(K_S, \cdot)$ and $F(K_{ME}, \cdot)$ are only ever called on the same input once, they can be replaced with random selections. $F_p(K_X, \cdot)$, $F_p(K_I, \cdot)$ and $F_p(K_Z, \cdot)$ are replaced by f_X, f_I and f_Z respectively. By a standard hybrid argument, we can show that there exist adversaries $B_{1,1}$, $B_{1,2}$ and $B_{1,3}$ such that $\Pr[G_1 = 1] - \Pr[G_0 = 1] \leq 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F,B_{1,1}}(\lambda) + 3 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F_p,B_{1,2}}(\lambda)$.

Game 2: Game 2 is the same with G_1 except that G_2 includes $e \leftarrow \mathrm{Enc}(K_E, 0^\lambda)$. It replaces the encryption of the document ids with the encryption of 0^λ. Since the ciphertexts are never decrypted, Enc is IND-CPA secure and encryption is run with $m = |W|$ different keys, there exists and adversary B_2 such that $\Pr[G_2 = 1] - \Pr[G_1 = 1] \leq m \cdot \mathbf{Adv}^{\mathrm{ind-cpa}}_{\Sigma,B_2}(\lambda)$.

Game 3: Game G_3 restructures the computation. The $\mathbf{XSet}$ elements have the form XS_ELEM$(w_x, \mathrm{ID}) = g^{F_p(K_X, w_x) \cdot F_p(K_I, \mathrm{ID})}$. In G_3, all possible such values are precomputed and stored in an array H. Then, when these values are required, they are read from the array, instead of being recomputed. Another array Y is used for xtoken values that do not have a corresponding ID result. The values used are the same as for Game G_2. Since the computation is only rearranged, $\Pr[G_3 = 1] = \Pr[G_2 = 1]$.

Game 4: Game G_4 is the same except that y values are now drawn randomly from Z^*_p. It is based on G_3 with $y \xleftarrow{\$} Z^*_p$. y depended on the random function f_Z and, due to the modifications made in Game G_3, is only computed in one location. So replacing it with random values does not change the distribution and $\Pr[G_4 = 1] = \Pr[G_3 = 1]$.

Game 5: Game G_5 includes those two parts based on G_4: $H[\mathrm{ID}_i, w] \xleftarrow{\$} G$ and $Y[w,u,c] \xleftarrow{\$} G$. The values used in H and Y are now drawn randomly from G. By the DDH problem, we have $\Pr[G_5 = 1] - \Pr[G_4 = 1] \leq \mathbf{Adv}^{\mathrm{ddh}}_{G,B_5}(\lambda)$.

The values of the X array are the g^a values. X values are raised to the power of xind when computing H and the power of $f_Z(w||c)$ when computing Y, so xind and $f_Z(w||c)$ will act as the b values. So in Game G_4, H and Y have values of the form g^{ab}, while in Game G_5 they are replaced with random values. Differentiating between these is the DDH problem.

Game 6 and 7: Games G_6 and G_7 reduce the use of H to cases where values will be used multiple times. Game G_6 modifies XSET_SETUP to only include the members of H that could actually be tested. xtokens are only provided for sterms and xterms in $\mathbf{s}$ and $\mathbf{w}$ respectively, so it is only possible to check for elements of the $\mathbf{XSet}$ XS_ELEM(w, ID) if $\mathrm{ID} \in \mathbf{DB}[\mathbf{s}[t,i]]$ and $w = \mathbf{w}[t,\alpha]$ for some combination of t,i and α. All other elements cannot be distinguished from random group elements. Thus, XSET_SETUP is modified to only include members of H matching the above condition. All other elements are drawn randomly from G. Therefore, we have $\Pr[G_6 = 1] = \Pr[G_5 = 1]$.

Game G_7 modifies the way GEN_TRANS computes the xtokens to only include member of H that are used multiple times. Any member of H included in the $\mathbf{XSet}$ by XSET_SETUP must be used by GEN_TRANS, and this is tested for by the first if statement. It is also possible GEN_TRANS will use the same member twice, if it is used in two different

queries t_1 and t_2. The current query number is passed in as an argument t_1 and the second if statement checks if any other query t_2 will also use the same member of H. If neither of these conditions apply, the xtoken is drawn at random from G. Since all repeating values of H are still used, we have $\Pr[G_7 = 1] = \Pr[G_6 = 1]$.

The simulator $\mathcal{S}$ will take the leakage $\mathcal{L}_{obs}(\mathbf{DB}, \mathbf{s}, \mathbf{w}) = (N, \bar{\mathbf{s}}, \mathrm{SP}, \mathrm{RP}, \mathrm{SRP}, \mathrm{IP}, \mathrm{XT}, \mathrm{ST}, \phi)$ and produce a simulated $\mathbf{EDB}, \mathbf{XSet}$ and transaction array $\mathbf{t}$. By showing that the simulator produces the same distribution as G_7 and combining the relations between the games, we can show that the simulator satisfies the theorem. Our simulator works similar to [5], due to the space restriction, we omit here. (We refer reader to [5] for more detail.)

C. Low Leakage Oblivious Boolean Search (LLOBS)

The security proof for LLOBS is a slight modification of the one for OBS above. The only change to the games is to send random group elements for certain xtoken values, as specified for LLOBS. No change is made to the simulator, except that the input leakage is slightly different as described in section IV-A3. Notably, $\hat{\mathbf{w}}$ is also still defined and calculated in the exact same way. However, the information it captures is decreased. It is still defined such that $\hat{\mathbf{w}}[t_1, \alpha] = \hat{\mathbf{w}}[t_2, \beta]$ iff $\exists i, j : IP[t_1, t_2, i, j, \alpha, \beta] \neq \emptyset$. However, in OBS, if we had $\mathbf{w}[t_1, \alpha] = \mathbf{w}[t_2, \beta]$, then it would only be possible to have $IP[t_1, t_2, i, j, \alpha, \beta] \neq \emptyset$ for all i, j if $\forall i, j : DB[s[t_1, i]] \cap DB[s[t_2, j]] \neq \emptyset$. However, in LLOBS, it is also possible to have $IP[t_1, t_2, i, j, \alpha, \beta] \neq \emptyset$ simply because valid xtokens were not included for those sterms. Thus, it can be seen, less information is revealed in IP and $\hat{\mathbf{w}}$ for LLOBS.

V. Conclusion

A novel Searchable Symmetric Encryption (SSE) scheme has been proposed which which achieves a higher level of efficiency for a large class of queries than any previous scheme. We chose to extend the work of [5], which has good security and performance properties but lacks support for most boolean queries. Our scheme differs from theirs by looking up multiple terms in the index data structure when evaluating a query. Our scheme allows any of the search terms to be looked up in the index which both increases the class of suppported queries and allows many queries that are already supported to be evaluated more efficiently, since a more optimal set of terms can be chosen to be looked up in the index. This optimal choice has been investigated thoroughly, as well as the best manner to evaluate the rest of the query. We have proposed three new search schemes: BBS, OBS and LLOBS. BBS is the most efficient but also the least secure. OBS solves the security problems of BBS but requires the use of exponentiation operations. LLOBS is a modified form of OBS which has better security and performance properties but is harder to implement.

Acknowledgement

This work is supported by National Natural Science Foundation of China (61472083).

References

[1] J. Baek, Q. H. Vu, J. K. Liu, X. Huang, and Y. Xiang. A secure cloud computing based framework for big data information management of smart grid. *IEEE Trans. Cloud Computing*, 3(2):233–244, 2015.

[2] M. Bellare, A. Boldyreva, and A. O'Neill. Efficiently-searchable and deterministic asymmetric encryption. Technical report, Citeseer, 2006.

[3] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer, 2004.

[4] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III. Public key encryption that allows pir queries. In *Advances in Cryptology-CRYPTO 2007*, pages 50–67. Springer, 2007.

[5] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[7] C. Chu, W. T. Zhu, J. Han, J. K. Liu, J. Xu, and J. Zhou. Security concerns in popular cloud storage services. *IEEE Pervasive Computing*, 12(4):50–57, 2013.

[8] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, pages 79–88. ACM, 2006.

[9] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[10] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[11] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *focs*, page 364. IEEE, 1997.

[12] K. Liang, M. H. Au, J. K. Liu, W. Susilo, D. S. Wong, G. Yang, T. V. X. Phuong, and Q. Xie. A dfa-based functional proxy re-encryption scheme for secure public cloud data sharing. *IEEE Trans. Information Forensics and Security*, 9(10):1667–1680, 2014.

[13] K. Liang, J. K. Liu, D. S. Wong, and W. Susilo. An efficient cloud-based revocable identity-based proxy re-encryption scheme for public clouds data sharing. In *ESORICS 2014, Part I*, volume 8712 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2014.

[14] K. Liang, C. Su, J. Chen, and J. K. Liu. Efficient multi-function data sharing and searching mechanism for cloud-based encrypted data. In *AsiaCCS 2016*, pages 83–94. ACM, 2016.

[15] J. Liu, X. Huang, and J. K. Liu. Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption. *Future Generation Comp. Syst.*, 52:67–76, 2015.

[16] J. K. Liu, M. H. Au, X. Huang, R. Lu, and J. Li. Fine-grained two-factor access control for web-based cloud computing services. *IEEE Trans. Information Forensics and Security*, 11(3):484–497, 2016.

[17] J. K. Liu, M. H. Au, W. Susilo, K. Liang, R. Lu, and B. Srinivasan. Secure sharing and searching for real-time video data in mobile cloud. *IEEE Network*, 29(2):46–50, 2015.

[18] J. K. Liu, K. Liang, W. Susilo, J. Liu, and Y. Xiang. Two-factor data security protection mechanism for cloud storage system. *IEEE Trans. Computers*, 65(6):1992–2004, 2016.

[19] R. Ostrovsky. Efficient computation on oblivious rams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 514–523. ACM, 1990.

[20] D. X. Song, D. Wagner, and A. 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.

[21] S. Wang, K. Liang, J. K. Liu, J. Chen, J. Yu, and W. Xie. Attribute-based data sharing scheme revisited in cloud computing. *IEEE Trans. Information Forensics and Security*, 11(8):1661–1673, 2016.

[22] S. Wang, J. Zhou, J. K. Liu, J. Yu, J. Chen, and W. Xie. An efficient file hierarchy attribute-based encryption scheme in cloud computing. *IEEE Trans. Information Forensics and Security*, 11(6):1265–1277, 2016.

[23] F. Xhafa, J. Wang, X. Chen, J. K. Liu, J. Li, and P. Krause. An efficient PHR service system supporting fuzzy keyword search and fine-grained access control. *Soft Comput.*, 18(9):1795–1802, 2014.