# Symmetric Searchable Encryption for Exact Pattern Matching using Directed Acyclic Word Graphs[*]

Rolf Haynberg[1], Jochen Rill[2], Dirk Achenbach[2] and Jörn Müller-Quade[2]
[1]*1&1 Internet AG, Karlsruhe, Germany*
[2]*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

Keywords: Searchable Encryption, SSE, Exact Pattern Matching, Directed Acyclic Word Graphs.

Abstract: Searchable Encryption schemes allow searching within encrypted data without prior decryption. Various index-based schemes have been proposed in the past, which are only adequate for certain use cases. There is a lack of schemes with exact pattern matching capabilities. We introduce Symmetric Searchable Encryption for Exact Pattern Matching, a new class of searchable encryption schemes. To this end, we define the XPM-SSE primitive and two privacy notions for the new primitive. Our own construction, SEDAWG, is a XPM-SSE scheme which uses Directed Acyclic Word Graphs. We discuss and prove its properties.

## 1 INTRODUCTION

Cloud computing is one of the most promising trends in the IT industry. In terms of data security however, cloud computing brings a new threat: Users lose control over their data. Cloud providers can access their customer's data at will.

To challenge this drawback technologically, one needs to seek methods that enhance the privacy of the user's data without negating the advantages of cloud computing. Notably, computational and storage overhead should be handled in the cloud, not on the client.

Consider for example the scenario of an outsourced e-mail archive: Users wish to employ a cloud provider to store their e-mails in order to be able to access them with a mobile device and without letting the provider gain knowledge of the archive's content. To conserve bandwidth, they want to perform searches on their archived e-mail online instead of downloading each and every message individually and searching locally.

In this paper, we address the problem of secure exact pattern matching: A user encrypts a long string he later wishes to query for the occurrence of certain patterns. This encrypted string is then uploaded to a server and to perform a search, the user can interact with the server. The server should never learn neither the string itself, nor the patterns searched for.

[*]This work has been partially funded by the Federal Ministry of Education and Research, Germany (BMBF, Contract No. 16BY1172). The responsibility for the content of this article lies solely with the authors.

### 1.1 Our Contribution

We introduce a primitive for a symmetric searchable encryption for exact pattern matching and two security notions for the primitive. To our knowledge, this is the first such primitive. Further, we offer a construction that realizes this primitive. Our construction involves precomputation in order to ensure a particularly efficient search performance.

### 1.2 Related Work

In the literature, there are two general approaches to searchable encrypted data: Symmetric Searchable Encryption (SSE) (Goh, 2003; Curtmola et al., 2006; Golle et al., 2004) and Public Key Encryption with Keyword Search (PEKS) (Chang and Mitzenmacher, 2005; Abdalla et al., 2008). In almost all cases these approaches are keyword-based. Keyword-based schemes generally allow arbitrary keywords to be stored in an index and not only keywords contained in the actual document. Furthermore, they allow indices to be created for arbitrary documents, not just strings. On the other hand, keyword-based schemes don't support substring search, exact pattern match-

Haynberg R., Rill J., Achenbach D. and Müller-Quade J..
Symmetric Searchable Encryption for Exact Pattern Matching using Directed Acyclic Word Graphs.
DOI: 10.5220/0004530004030410
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 403-410
ISBN: 978-989-8565-73-0
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

ing or return the number of occurrences of a substring. Also, keywords must be defined when constructing the index, whereas our approach only requires the pattern at search time. The work of D. Xiaodong Song et al. (Song et al., 2000) presents a keywordless approach based on stream ciphers and supports pattern matching. However, the time required for performing a search scales linearly with the size of the document.

There is a rich body of literature on exact pattern matching algorithms, going back to the late seventies. The general method for improving performance is precomputation. Approaches can be separated by where the precomputation occurs: There are algorithms that perform precomputation on the search pattern (Boyer and Moore, 1977; Knuth et al., 1977; Karp and Rabin, 1987; Baeza-Yates and Gonnet, 1992) as well as algorithms that perform precomputation on the string (Manber and Myers, 1990; Ukkonen, 1995; Blumer et al., 1985; Blumer et al., 1987; Crochemore and Vérin, 1997). Our scheme can be assigned to the latter. The aforementioned algorithms have been engineered for performance and efficiency alone and were not conceived in an adversarial scenario, however. Our scenario involves an honest-but-curious adversary and we therefore seek to hide as much information as possible from him. Our goal of hiding access patterns is very similar to that of Private Information Retrieval (Chor et al., 1998; Kushilevitz and Ostrovsky, 1997; Di Crescenzo et al., 2000) and that of the work on Oblivious RAMs (Goldreich and Ostrovsky, 1996). In fact, we can use a PIR construction in our scheme to improve its privacy. Moreover, an ideal solution to our problem statement can be used to construct a PIR scheme.

This paper is organized as follows: In the remainder of this section, we define a new primitive XPM-SSE. We then define the optional property of *pattern privacy* and show that Private Information Retrieval can be reduced to a XPM-SSE scheme with pattern privacy. In Section 3, we present our construction SEDAWG (Searchable Encryption using Directed Acyclic Word Graphs) and discuss its properties. Section 4 concludes.

## 2 DEFINITION OF XPM-SSE

In this section we define a searchable encryption scheme which allows to perform a full text search within encrypted data.

**Definition 1** (Exact Pattern Matching *XPM*)**.** *An algorithm $\mathcal{A}$ is a technique for exact pattern matching (XPM) over the alphabet $\Sigma$ if it returns upon input $S \in \Sigma^*$ and $m \in \Sigma^*$ the exact number of occurrences of $m$ in $S$.*

Based on the exact pattern matching algorithm, we are defining *Symmetric Searchable Encryption for Exact Pattern Matching* (XPM-SSE). Note that pre-existing notions for encryption schemes don't suffice in this scenario: Assume a protocol in which, to perform a query, the client uploads the decryption key to the server and lets the server decrypt the ciphertext and return the result. Such a protocol does not exhibit data privacy and therefore should not be considered a XPM-SSE scheme. Also, it is easy to provide a security notion that can only be met with inefficient solutions, simply by storing an encrypted file on the server which is downloaded and decrypted prior to searching. Hence, any meaningful notion must also account for the protocol messages that are exchanged in the evaluation of a query.

**Definition 2** (XPM-SSE Scheme)**.** *Let $S \in \Sigma^n$ be an arbitrary but finite string over the encoding $\Sigma$. A tuple $((\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}), I = (\mathcal{S}, \mathcal{C}))$ is a Symmetric Searchable Encryption scheme for Exact Pattern Matching (XPM-SSE scheme), if*

- $\mathsf{Gen} : 1^k \rightarrow \{0,1\}^k$ *is a PPT algorithm which, given a key $K$, the plaintext $S$ and a security parameter $k$, generates a key $K \leftarrow \mathsf{Gen}(1^k)$.*
- $\mathsf{Enc} : \{0,1\}^k \times \Sigma^* \rightarrow \{0,1\}^*$ *is a PPT algorithm which generates a ciphertext $D \leftarrow \mathsf{Enc}_K(S)$.*
- $\mathsf{Dec} : \{0,1\}^* \times \{0,1\}^k \rightarrow \Sigma^*$ *is a polynomially bounded algorithm which outputs the plaintext $S \leftarrow \mathsf{Dec}_K(\mathsf{Enc}_K(S))$ given a key $K$ and the cipher text $\mathsf{Enc}_K(S)$.*
- $I = (\mathcal{S}, \mathcal{C})$ *is a protocol for a pair of machines (server and client) which perform an algorithm for Exact Pattern Matching. The server $\mathcal{S}$ is provided with an encryption of the string $\mathsf{Enc}_K(S)$, the client $\mathcal{C}$ is supplied with the search pattern $w$, the encryption key $K$ and outputs the exact number of occurrences of $w$ in $S$. With $\mathrm{view}_K(S, w)$ we denote all state transitions of the server $\mathcal{S}$ and all its received received and sent messages.*
- *It has* data privacy*, that is, the advantage of the adversary in* $\mathrm{PrivK}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ *(Security Game 1) is negligible, that is*

$$\forall \mathrm{PPT} \mathcal{A}, c \in \mathbb{N} \exists k \in \mathbb{N} :$$

$$\Pr[\mathrm{PrivK}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k) = 1] \leq \frac{1}{2} + k^{-c}$$

*The notion of data privacy captures the goal of hiding the string itself from the adversary.*

Note that by this definition, any XPM-SSE scheme also has *result privacy*: If an adversary could learn the result of a query she supplies, she can use that information to distinguish two encryptions of plain texts which does not fulfill the above definition.

Next, we describe Security Game 1 which is used in Definition 2.

**Security Game 1** ($\text{PrivK}^{\text{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$)**.**

1. *The experiment chooses a key* $K \leftarrow \mathsf{Gen}(1^k)$ *and a random bit* $b \leftarrow \{0,1\}$.
2. *The adversary* $\mathcal{A}$ *is given input* $1^n$*, oracle access to* $\mathsf{Enc}_K(\cdot)$ *and to a view oracle* $\mathsf{T}_K(\cdot,\cdot)$. $\mathsf{T}_K(S,x)$ *returns the server's view to any query x, given a plaintext S:* $\text{view}_K(S,x)$.
3. $\mathcal{A}$ *outputs two plaintexts* $m_0$ *and* $m_1$ *of the same length to the experiment.*
4. $\mathcal{A}$ *is given* $\mathsf{Enc}_K(m_b)$.
5. $\mathcal{A}$ *outputs a number of queries* $x_0,\ldots,x_q$ *and an integer* $i \in \{0,\ldots,q\}$ *to the experiment.*
6. *The queries* $x_0,\cdots,x_q$ *are evaluated in that order.*
7. $\mathcal{A}$ *is given the view on the challenge ciphertext to query i:* $\text{view}_K(m_b,x_i)$.
8. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_K(\cdot)$ *and* $\mathsf{T}_K(\cdot,\cdot)$.
9. $\mathcal{A}$ *submits a guess* $b'$ *for b.*

*The result of the experiment is defined to be* 1 *if* $b' = b$ *and* 0 *else.*

The complementary property to data privacy is that of pattern privacy: A scheme that has pattern privacy hides the contents of queries from the server. We define pattern privacy as (computational) indistinguishability of transcripts.

**Definition 3** (Pattern Privacy)**.** *A scheme* $((\mathit{Gen},\mathit{Enc},\mathit{Dec}),(\mathcal{S},\mathcal{C}))$ *has pattern privacy if for all* $c \in \mathbb{N}$*, PPT algorithms* $\mathcal{A}$*, search patterns* $x,x' \in \Sigma^*$ *with* $|x| = |x'|$ *and* $S \in \Sigma^*$ *there is a* $k \in \mathbb{N}$*, so that*

$$|\Pr[\mathcal{A}(1^k,\text{view}_K(S,x) \to 1] - \Pr[\mathcal{A}(1^k,\text{view}_K(S,x') \to 1]| \leq k^{-c}$$

If a XPM-SSE scheme has *pattern privacy* (Definition 3), we call the scheme *XPM-SSE with pattern privacy*. In the remainder of this section, we show that Private Information Retrieval can be reduced to a XPM-SSE scheme with pattern privacy.

First we give a definition of a Private Information Retrieval scheme here. The definition follows that of Kenan (Kenan, 2005) closely, but uses our notation. Informally, a PIR scheme is a scheme that allows a client to retrieve bits from a remote database in such a way that the database doesn't learn which bit has been retrieved.

**Definition 4** (PIR)**.** *Let* $D \in \{0,1\}^n$ *be a database of n bits. The* $(i+1)$*th bit shall be denoted as* $D[i]$*. A scheme for Computational Single-Server Private Information Retrieval (PIR) is a pair of machines* $(\mathcal{S},\mathcal{C})$ *for which the following properties hold:*

1. *Correctness:* $\forall x \in \{0,\ldots,n-1\}, D \in \{0,1\}^n$, $c \in \mathbb{N} \exists k \in \mathbb{N}:$
$$\Pr[\mathcal{C}(1^k,\text{view}(D,x)) = D[x]] \geq 1 - k^c$$
2. *User Privacy:* $\forall x,x' \in \mathbb{N}_0, D \in \{0,1\}^n$, $\text{PPT}\mathcal{A}, c \in \mathbb{N} \exists k \in \mathbb{N}:$
$$|\Pr[\mathcal{A}(1^k,\text{view}(D,x)) = 1] - \Pr[\mathcal{A}(1^k,\text{view}(D,x')) = 1]| \leq k^{-c}$$

We can now state our claim.

**Proposition 1.** *Computational Single-Server Private Information Retrieval can be reduced to XPM-SSE with pattern privacy.*

*Proof.* To prove the proposition, we give a construction using a XPM-SSE scheme and show that it achieves PIR. We show how to construct a string from a binary database with *n* entries. We then explain how to perform a database query and how to interpret the result.

Table 1: Example database with the corresponding string to be used for a database retrieval with a XPM-SSE scheme.

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $D[i]$ | 1 | 0 | 1 | 0 | 1 | 0 |

$$\Downarrow$$

`!000$ !010$ !100$`

The alphabet for the XPM-SSE construction is $\Sigma = \{0,1,!,\$\}$. Construct *S* as follows: Start with an empty string *S*. Iterate *i* through $\{0,\ldots,n-1\}$, and, if $D[i] = 1$, append the binary encoding of *i* (with leading zeroes to hide the size of *i*), enclosed in the delimiters ! and $, to S. Output S. (See Table 1 for an example.)

To retrieve bit *i* from *S* (stored on the server), run the search protocol for the binary encoding of *i*, enclosed in the delimiters ! and $. If the search is a success, the retrieved bit is 1 and 0 otherwise.

By construction, the binary encoding of *i* is $\in S$ iff $D[i] = 1$. It follows that the above construction delivers correct results.

In the above construction, the bit ID *i* that is to be retrieved is transmitted in the form of a search query. This information is, kept secret from the adversary because of the required pattern privacy. If the above construction didn't provide Private Information Retrieval, it would leak information about $i$ to the adversary and the XPM-SSE scheme wouldn't have pattern privacy which contradicts the proposition. □

## 3 SEDAWG: A XPM-SSE CONSTRUCTION

Our Searchable Encryption Scheme is based on the idea to store the encrypted data on the server and to perform the searching on the client. The encryption algorithm uses Directed Acyclic Word Graphs (DAWGs) to prepare the encrypted data for searching. They are described in Section 3.1. We then present our XPM-SSE scheme in Section 3.2. In Section 3.3, we discuss its performance and proof its security.

#### 3.0.1 Notation and Conventions

If not mentioned otherwise, $S$ will be an arbitrary string of length $n$ over the encoding $\Sigma$, that is $S \in \Sigma^*$. The length of a string $S$ will be denoted as $|S|$. $\varepsilon$ is the empty string and has a length of 0. $S[i]$ denotes the $(i+1)$th character of $S$ for $i \in \mathbb{N}_0$ with $i < |S|$. $S[i..j]$ denotes the string $S[i] \cdot \ldots \cdot S[j]$ for $0 \le i \le j < n$. If $i = j$ let $S[i..j] = S[i]$. $S[i..j]$ is called a *substring*. The set of all substrings of a string $S$ is written as

$$\mathit{substrings}_S = \{S[i..j] \mid 0 \le i \le j < n\} \cup \{\varepsilon\}$$

### 3.1 The Underlying Data Structure DAWG

The *Directed Acyclic Word Graph* (DAWG) is a data structure derived from a string. It is similar to the Suffix Tree data structure and reduces the time complexity for several string analysis algorithms, such as pattern matching, by making use of pre-computation. DAWGs were first introduced by Blumer et al. in (Blumer et al., 1985). Our definition follows theirs.

**Definition 5** (Endpoint Set). *Let x be a substring of a string S. Then*

$$E_S(x) := \{j \mid \exists i : x = S[i..j]\}$$

*is called the* Endpoint Set *for x with respect to S, i. e. the endpoints of all occurrences of x in S.*

The substrings that specify the same Endpoint Set are important for the data structure. An equivalence class encompasses them.

**Definition 6** ($\equiv_S^E$, $[x]_S^E$). *Two substrings x and y of a string S are equivalent with respect to $\equiv_S^E$, if they specify the same Endpoint Set:*

$$x \equiv_S^E y \Leftrightarrow E_S(x) = E_S(y)$$

*The equivalence class of x with respect to $\equiv_S^E$ is denoted as $[x]_S^E$. The representative of an equivalence class is defined as the longest substring of that class and is denoted as $\overleftarrow{x}$.*

**Definition 7** ($DAWG(S)$). *The* Directed Acyclic Word Graph *of a prefix-free string S, in symbols $DAWG(S)$ is the directed graph $(V,E)$ with*

$$\begin{aligned} V =& \{[x]_S^E \mid x \in \mathit{substrings}_S\} \\ E =& \{([x]_S^E, [xa]_S^E) \mid a \in \Sigma, \\ & x, xa \in \mathit{substrings}_S, \overleftarrow{x} \ne \overleftarrow{xa}\} \end{aligned}$$

The equivalence class of the empty string $[\varepsilon]_S^E$ plays a special role. The corresponding node will be called the *root node* in the following chapters as it does not have any incoming edges.

Moreover, a label is associated with every edge of the graph.

**Definition 8** (Edge Label $\mathit{edgeLabel}(e)$). *Let $([x]_S^E, [xa]_S^E) = e$ be an edge of the $DAWG(S)$ data structure. Then $\mathit{edgeLabel}(e) = a$ is called the* edge label *of e.*

The edge labels are defined in such a way that every path in the graph $DAWG(S)$ corresponds to a substring of $S$. The path that corresponds to $S$ itself is important for the decryption algorithm of our scheme. We will refer to the edges of this path as *natural edges*.

**Definition 9** (Natural Edge). *Let $([x]_S^E, [xa]_S^E) = e$ be an edge of the $DAWG(S)$ data structure. If there exists a string w in $[xa]_S^E$ such that w is a prefix of S, e is called a* natural edge *for this graph.*

### 3.2 Pattern Matching using DAWGs

We now sketch how we utilize $DAWG(S)$ to decide whether $w \in \mathit{substrings}_S$ with $O(|w|)$ character comparisons. The algorithm is based on a central property of the DAWG: $w \in \mathit{substrings}_S$ if and only if there is a path $[\varepsilon]_S^E, v_1, \ldots, v_{|w|}$ with $\mathit{edgeLabel}(v_0, v_1) \cdot \ldots \cdot \mathit{edgeLabel}(v_{|w|-1}, v_{|w|}) = w$. Since, for any pattern $w$, this path starts at the root node, we can tell if it exists by matching the edge labels of the path along the pattern: We traverse the DAWG by comparing the labels on the outgoing edges of a node to the corresponding character in $w$. The edge that has a matching label leads to the next node on the path. If all characters of the pattern have been matched, $w \in \mathit{substrings}_S$ and the search ends. Otherwise, $w$ does not occur in $S$.

In our XPM-SSE construction, we store the graph using adjacency lists. Also, with every node $v$ we

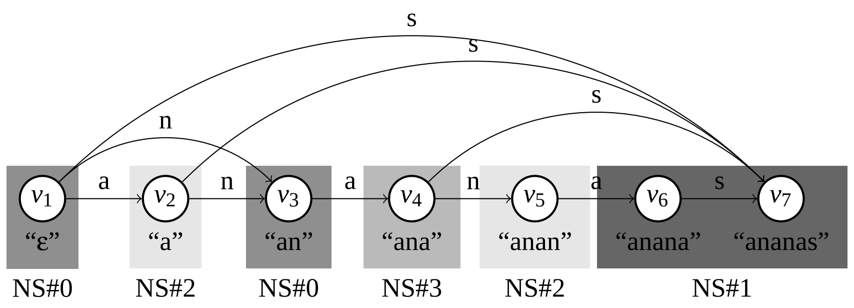
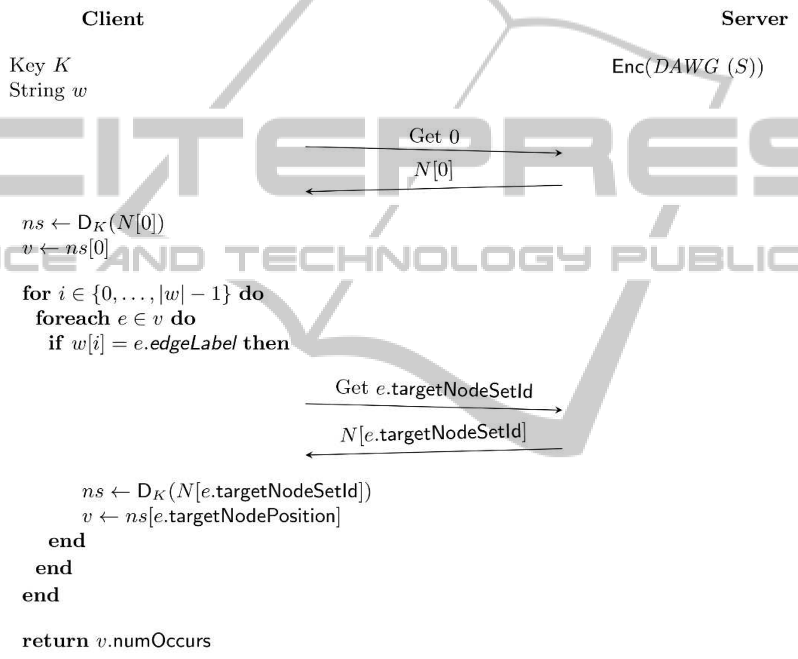


Figure 1: The DAWG for the input string *S*=`ananas`. It consists of seven nodes ($v_1$ to $v_7$) which are mapped to four node sets (NS#0 to NS#3). The node representative is written underneath each node. The transition edges are drawn as directed arrows which are labeled with their respective edge labels. Every outgoing path from $v_1$ is a subword of *S*.



```
$ns \leftarrow \mathsf{D}_K(N[0])$
$v \leftarrow ns[0]$

for $i \in \{0,\ldots,|w|-1\}$ do
  foreach $e \in v$ do
    if $w[i] = e.edgeLabel$ then
      $ns \leftarrow \mathsf{D}_K(N[e.\text{targetNodeSetId}])$
      $v \leftarrow ns[e.\text{targetNodePosition}]$
    end
  end
end

return $v$.numOccurs
```

Figure 2: General communication pattern for searching a string *w* in *S*. We assume that $\mathsf{Enc}(DAWG\ (S))$ is available on the server. The case of $w \notin substrings_S$ is not covered here. Also note that for privacy reasons, we explicitly do not employ caching mechanisms.

store a property *v*.numOccurs. It represents the number of occurences of the representative of *v* in *S* and allows to determine the number of occurences of the search pattern when reading *v*.

To store the DAWG, we randomly distribute all of its nodes into a fixed number of disjoint node sets which depends on the size of the text. Then, we augment every edge in the DAWG with a reference to the node set that contains the target node. To ensure the indistinguishability of the encryptions of different strings of the same size, we pad the node sets individually to their maximum size. Finally, we create files from the node sets and encrypt them individually using the symmetric encryption scheme $(\mathsf{E},\mathsf{D},\mathsf{G})$. See Figure 1 for a visualization of how the DAWG is stored.

For a schematic description of the search procedure with pseudocode, see Figure 2. The detailed descriptions of the algorithms for encryption and decryption can be found in Algorithms 1 and 2, respectively.

## 3.3 Properties of SEDAWG

In this subsection, we discuss the security and performance properties of our scheme.

### 3.3.1 Performance and Space Complexity

The space complexity of the output of $\mathsf{Enc}_K(S)$, i. e. the ciphertext size, is in $O(|S|)$ because the size of $DAWG(S)$ lies in $O(|S|)$ and the encryption algo-

**Algorithm 1:** $\mathsf{Enc}_K(S)$.

```
Input: String $S$ with encoding $\Sigma$, Key $K$
Output: A set $B$ of files encrypted under $K$
Data: Set of Nodes $V$, Set of Edges $E$, Set of Node Sets
      $N$, $s^\star$
(V, E) = DAWG(S)                                  // Step 1
$position(\circ) \leftarrow 0$
$nodeset(\circ) \leftarrow 0$
getNodeSetById($0$) $\leftarrow \{\circ\}$   // Add root node to
node set 0
foreach $v \in V$ do                              // Step 2
    $ns \leftarrow_r \{ns \in N \mid s^\star \geq size(ns \cup v)\}$
    $position(v) \leftarrow size(ns)$
    $nodeset(v) \leftarrow id(ns)$
    $ns \leftarrow ns \cup v$
end
foreach $ns \in N$ do                             // Step 3
    foreach $v \in ns$ do
        foreach $(v,w) \in edges(v)$ do
            $v[w]$.targetNodePosition $\leftarrow position(w)$
            $v[w]$.targetNodeSetId $\leftarrow nodeset(w)$
        end
    end
end
foreach $ns \in N$ do                             // Step 4
    $b \leftarrow$ ””
    foreach $v \in ns$ do
        foreach $(v,w) \in edges(v)$ do
            $b \leftarrow$ concatenate($b$,
            $v[w]$.isNaturalEdge,
            $v[w]$.edgeLabel,
            $v[w]$.targetNodePosition,
            $v[w]$.targetNodeSetId)
        end
        $b$.append(#)
    end
    $b \leftarrow$ pad $(b, s^\star)$
    $b \leftarrow \mathsf{E}_K(b)$
    $B \leftarrow B \cup \{b\}$
end
return $B$
```

**Algorithm 2:** $\mathsf{Dec}_K(B)$.

```
Input: Set of encrypted files $B$, Key $K$
Output: Plaintext $S$
Data: Set of Nodes $V$, Set of Edges $E$, Set of Node Sets
      $N$
foreach $b \in B$ do                 // Decrypt all files
    $N \leftarrow N \cup \mathsf{D}_K(b)$
end
nsid $\leftarrow 0$
npos $\leftarrow 0$
while $v \leftarrow \mathsf{D}_K$(getFileFromServer($nsid$))[$npos$] do
// For node ...
    foreach AdjacencyListEntry $e \in v$ do  // ...find
    the natural edge
        if $e$.isNaturalEdge then
            nsid $\leftarrow e$.targetNodeSetId
            npos $\leftarrow e$.targetNodePosition
            S.append($e$.edgeLabel)
            break
        end
    end
end
return $S$
```

rithm Enc only adds information which size is linear in $S$ (see Algorithm 1). Also the time complexity, i. e. the execution time of $\mathsf{Enc}_K(S)$, is in $O(|S|)$ for similar reasons. The communication complexity of the search protocol $I_K(\mathsf{Enc}_K(S), w)$ is asymptotically optimal and depends only linearly on $|w|$.

We performed preliminary benchmarks with an unoptimized implementation of our scheme on Amazon's S3 cloud storage. Results show an advantage of our scheme regarding search times in comparison to a trivial approach (which is: downloading the full encrypted text, decrypting it and searching with the Boyer-Moore algorithm (Boyer and Moore, 1977)) if the bandwidth is limited. In our tests we used a 3G mobile network connection and the search was up to 10-15 times faster for short words (4 to 8 characters) and 2-4 times for long words (32 to 64 characters) using an email archive with an unencrypted size of 60 MiB. The precomputational overhead is unpractical for many use cases however, since the encrypted text was about 100 times larger than the original text and the memory demand for the precomputation grew linerally with the size of the input text with a factor of approximately 1300. This must be improved upon in future work.

### 3.3.2 Security

We can now present our result that SEDAWG is a XPM-SSE scheme. Given the ability to hide access patterns from the adversary, it also exhibits pattern privacy. With Proposition 1 from Section 2, this implies that pattern privacy and PIR are equivalent.

**Proposition 2.** *SEDAWG is a XPM-SSE scheme if* $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ *is a IND-CPA secure scheme.*

*Proof.* We must prove that the data privacy property holds for our construction. Suppose $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ is a IND-CPA secure encryption scheme and $\mathcal{A}$ is a PPT adversary who can win game $\mathrm{PrivK}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ (Security Game 1) with nonnegligible probability. In $\mathrm{PrivK}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, $\mathcal{A}$ receives one encryption $\mathsf{Enc}_K(m_b)$ and one server's view to a protocol run $\mathrm{view}_K(m_b, x_i)$. She then outputs a guess $b'$ for $b$.

The server's view of a protocol run consists of the IDs of the requested files and the files themselves. Consider a modification of the game, $\mathrm{PrivK'}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$: Instead of $\mathsf{Enc}_K(m_b)$ and $\mathrm{view}_K(m_b, x_i)$, the adversary is sent the encryption of a zero string with the same length as $m_0$[2]: $\mathsf{Enc}_K(0^{|m_0|})$. Also, $\mathrm{view}_K(m_b, x_i)$ is altered in such a way that the transmitted files are taken from $\mathsf{Enc}_K(0^{|m_0|})$ instead of $\mathsf{Enc}_K(m_b)$ ("so the story fits"). Call $\mathcal{A}$'s output from the modified game $b''$. Because $(\mathsf{E}, \mathsf{D}, \mathsf{G})$ is IND-CPA secure, $\mathcal{A}$ cannot distinguish $\mathsf{Enc}_K(m_b)$ from $\mathsf{Enc}_K(0^{|m_0|})$. Hence, $b''$ is statistically close to $b'$.

In a second modification $\mathrm{PrivK}''^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, we replace the IDs from the server's view with random IDs (chosen uniformly at random from the set of available file IDs, without replacement), except for the first request—the file ID that is first requested is always 0. Call $\mathcal{A}$'s output from this modified game $b'''$. Because the original file IDs have been chosen in the same manner as the IDs in our modified game, and the adversary is only supplied with one view, her output $b'''$ is again statistically close to $b''$.

Following our argument, if the adversary's output in $\mathrm{PrivK}^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ is correlated to $b$, its output in $\mathrm{PrivK}''^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$ is also correlated to $b$. But in $\mathrm{PrivK}''^{\mathrm{cppa}}_{\mathcal{A},\mathsf{Enc}}(k)$, $\mathcal{A}$ receives no input that correlates with $b$. This is a contradiction. $\square$

In Proposition 1 we showed that PIR is at least as hard as XPM-SSE. We now show that they are in fact equivalent. Considering that no practical scheme for Single Server PIR is known, this implies that achieving XPM-SSE with pattern privacy is a difficult task.

**Proposition 3.** *SEDAWG with PIR is a XPM-SSE scheme with pattern privacy if* $\mathsf{E}$ *is a IND-CPA secure encryption.*

*Proof.* Assume an adversary $\mathcal{A}$ who can distinguish two search patterns from their transcripts of SEDAWG with nonnegligible probability. Requested file IDs depend on the search pattern in a deterministic manner. Because, by definition, $\mathcal{A}$ cannot learn any information from $\mathsf{Enc}_K(S)$, she solely uses the requested IDs for the distinction. Hence, she can distinguish two series of server requests and violate the PIR assumption. $\square$

## 4 SUMMARY AND CONCLUSIONS

In this paper, we introduced Symmetric Searchable Encryption for Exact Pattern Matching, a new class of searchable encryption schemes, with which a client can privately search an encrypted string stored on a server. We defined the new primitive XPM-SSE and two security notions for this primitive, data privacy and pattern privacy. Data privacy captures the idea that the data stored on the server should be kept hidden from the server. Pattern privacy ensures the server can learn nothing from search logs except the pattern length. We showed that pattern privacy is equivalent to Computational Single-Server Private Information Retrieval.

We provided our construction SEDAWG for XPM-SSE. It uses directed acyclic word graphs (DAWGs) to ensure good performance for the cost of precomputational overhead. During precomputation, the DAWG for the string is computed and split into files. These files are then encrypted with a symmetric IND-CPA secure cipher. The search protocol navigates the DAWG, successively downloading required files.

There is a preliminary implementation that shows the practicality of our approach. However, while the search operation performs very efficiently, the precomputation is memory intense. Algorithm engineering might improve the overall performance of our implementation.

Further research can be directed at extending the scheme to allow modifications or extensions of the encrypted text without the need for a complete reencryption.

[2] $|m_0| = |m_1|$.

## REFERENCES

Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., and Shi, H. (2008). Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *Journal of Cryptology*, 21:350–391.

Baeza-Yates, R. and Gonnet, G. H. (1992). A new approach to text searching. *Commun. ACM*, 35(10):74–82.

Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M. T., and Seiferas, J. (1985). The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31 – 55. Eleventh International Colloquium on Automata, Languages and Programming.

Blumer, A., Blumer, J., Haussler, D., McConnell, R., and Ehrenfeucht, A. (1987). Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595.

Boyer, R. S. and Moore, J. S. (1977). A fast string searching algorithm. *Commun. ACM*, 20(10):762–772.

Chang, Y.-C. and Mitzenmacher, M. (2005). Privacy preserving keyword searches on remote encrypted data. In Ioannidis, J., Keromytis, A., and Yung, M., editors, *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer Berlin / Heidelberg.

Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. (1998). Private information retrieval. *J. ACM*, 45(6):965–981.

Crochemore, M. and Vérin, R. (1997). On compact directed acyclic word graphs. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, pages 192–211, London, UK. Springer-Verlag.

Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2006). Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, New York, NY, USA. ACM.

Di Crescenzo, G., Malkin, T., and Ostrovsky, R. (2000). Single database private information retrieval implies oblivious transfer. In Preneel, B., editor, *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin / Heidelberg.

Goh, E.-J. (2003). Secure indexes. http://eprint.iacr.org/2003/216/.

Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473.

Golle, P., Staddon, J., and Waters, B. (2004). Secure conjunctive keyword search over encrypted data.

Karp, R. M. and Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249 –260.

Kenan, K. (2005). *Cryptography in the Database : The Last Line of Defense*. Addison-Wesley, Upper Saddle River, NJ.

Knuth, D. E., James H. Morris, J., and Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350.

Kushilevitz, E. and Ostrovsky, R. (1997). Replication is not needed: Single database, computationally-private information retrieval. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 364, Washington, DC, USA. IEEE Computer Society.

Manber, U. and Myers, G. (1990). Suffix arrays: A new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on discrete algorithms*, pages 319–327, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. *IEEE Symposium on Security and Privacy*, pages 44–55. http://citeseer.nj.nec.com/song00practical.html.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3):249–260.