Security-Efficiency Tradeoffs in Searchable Encryption – Lower Bounds and Optimal Constructions

Raphael Bost * Pierre-Alain Fouque †

Abstract

Besides their security, the efficiency of searchable encryption schemes is a major criteria when it comes to their adoption: in order to replace an unencrypted database by a more secure construction, it must scale to the systems which rely on it. Unfortunately, the relationship between the efficiency and the security of searchable encryption has not been widely studied, and the minimum cost of some crucial security properties is still unclear.

In this paper, we present new lower bounds on the tradeoffs between the size of the client state, the efficiency and the security for searchable encryption schemes. These lower bounds target two kinds of schemes: schemes hiding the repetition of search queries, and forward-private dynamic schemes, for which updates are oblivious.

We also show that these lower bounds are tight, by either constructing schemes matching them, or by showing that even a small increase in the amount of leaked information allows for constructing schemes breaking the lower bounds.

1 Introduction

Searchable encryption aims at making efficient a seemingly easy task: outsourcing the storage of a database to an untrusted server, while keeping search features. With the development of Cloud storage services, for both private individuals and businesses, efficiency of searchable encryption is crucial: inefficient constructions would not be deployed at a large scale because they would not be usable. The key problem with searchable encryption is that any construction achieving 'perfect security' induces a computational or a communication overhead that is unacceptable for the cloud providers or for the cloud users — at least with current techniques and by today's standards.

Indeed, constructions based on Fully Homomorphic Encryption (FHE) [Gen09], or on Multi-Party Computations (MPC) have a huge computational overhead as their asymptotic complexity is linear in the size of the database, as the result of a search is the output of the evaluation of a circuit that uses the whole database as input. This has to be compared to unencrypted databases, which are able to resolve any query in time that is linear in the number of results of the query. Using Oblivious RAM (ORAM) [GO96], or Garbled RAM programs [LO13] would help, but these tools also suffer from an inherent cost, either in terms of computation or of communication, poly-logarithmic in the size of the database.

On the other hand, 'legacy compatible' schemes, which use deterministic encryption, or order preserving encryption (OPE) such as CryptDB [PRZB11], are very efficient (they often have the same asymptotic efficiency as an unencrypted database), but offer a poor level of security, as demonstrated by recent attacks [NKW15, CGPR15, GSB+17].

In between those two extrema, there exist constructions that do leak information, but not as much as the legacy compatible schemes, and that are sufficiently efficient to be used in practice, *e.g.* [CGKO06, KO12,

*Direction Générale de l'Armement - Maîtrise de l'Information. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DGA or the French Government. email: raphael_bost@alumni.brown.edu

†Université de Rennes 1, France. email: pa.fouque@gmail.com

KPR12, KP13, CJJ+13, PKV+14, SPS14, Bos16, BMO17, EKPE18]. This raises the question of a tradeoff between the efficiency of searchable encryption schemes and the amount of information they leak. Are there lower bounds on the computational complexity of searchable encryption schemes given a certain leakage profile, or a certain security feature?

This work studies this question and gives a positive answer for two types of searchable encryption schemes: (1) schemes that only reveal the size of the database and the number of results of a query, and (2) forward-private dynamic schemes — schemes that do not leak information about updated keywords during database modifications [Bos16].

Besides stating and proving these lower bounds, we also show that they are essentially tight by presenting constructions that match the bounds. Also, we show that, by slightly increasing the leakage compared to what is prescribed by the lower bounds, we can often drastically improve the asymptotics of schemes.

Our Contributions. We start by studying static schemes that only leak the number of entries N in the database, and the number n_w of entries matching the queried keyword w. In that case, as a corollary of a generic lower bound result, we show that the computational complexity of the first query is $\Omega\left(\frac{\log\binom{N}{n_w}}{\log|\sigma|}\right)$, where $|\sigma|$ is the size of the client's state. To do so, we proceed with similar ideas to the ORAM lower bound of Goldreich and Ostrovsky [GO96]. This result unfortunately precludes any scalable searchable encryption scheme that completely hides the search queries. We then show that we can construct a scheme with less overhead, but at the expense of a little bit more leakage during the setup phase.

In a second part, we show that any forward-private scheme has a minimal update overhead: either the computational complexity of an update is $\Omega(\frac{\log K}{\log|\sigma|})$, where K is the number of distinct keywords in the database, or the search complexity is $\Omega(\frac{t\log K}{\log|\sigma|})$, where t is the number of updates prior to the search query. Again, to prove this lower bound, we use a canvas similar to the one used in [GO96], but we adapt the details to forward-private SSE. We explain that this lower bound is tight, as there exist forward-private schemes with constant-time updates and $\Theta(K)$ client state, and which can be easily transformed in schemes with constant-size client state and $\Theta\left(\log^2 K\right)$ update complexity with current techniques.

2 Related Work

Symmetric Searchable Encryption (SSE) was first introduced in [SWP00] by Song *et al.*. The security definitions that we use today, and in this paper, for SSE, and which brought the formal notion of leakage into use, were described by Curtmola *et al.* in [CGKO06]. In that paper, the authors also presented the first efficient SSE construction, whose search query running time is linear in the number of results matching the query. Note that SSE is an instance of structured encryption [CK10], in which the data structure to encrypt is a multi-map (*a.k.a.* T-Set or reversed index).

Subsequently, many constructions arose, with various security features, and efficiency properties, such as database dynamism [KPR12], secure database updates [SPS14, BMO17], advanced searches [CJJ+13, PKV+14, KM17], reduced leakage [Nav15, GMP16], security against malicious adversaries [KO12, KO13] (a.k.a. verifiable SSE).

SSE can also be implemented using property preserving encryption (PPE), such as deterministic encryption, order-preserving encryption (OPE) [BCLO09], or order-revealing encryption (ORE) [PLZ13]. Such systems based on PPE, *e.g.* CryptDB, are very efficient, but extremely sensitive to attacks based on frequency analysis [NKW15, CGPR15, GSB+17].

Some works focus on public-key searchable encryption [BDOP04, BKOS07], on multi-user searchable encryption [PZ13], or on more complex settings [JJK+13]. These schemes consider a setting where more than one user can update the database and/or query it (*e.g.* [BBO07]). Here, we focus on *symmetric* searchable encryption, where the data owner and the querier are the same party, but many of the lower bounds we present in this paper can be ported to the aforementioned cases.

Related work on lower bounds. A recent line of work studies the tradeoffs between the locality of the memory accesses, the amount of memory read during a search and the server storage size, from lower bounds [CT14, ANSS16, ASS18] to tight constructions [ASS18, DPP18].

Lower bounds on the efficiency of searchable encryption has previously been studied in the case of verifiable SSE, *i.e.* when an active and malicious adversary returns invalid search results to the client. Bost *et al.* showed in [BFP16] that a verifiable SSE scheme with a client state of size $|\sigma| = \omega(K)$ has a computational overhead of $\Omega(\frac{\log K}{\log \log K})$ either for search or update queries, and proposed a tight construction based on incremental hashing and authenticated dictionaries.

On a related topic, Goldreich and Ostrovsky showed in [GO96] that any secure ORAM protocol between a server (supporting only read and write calls) and a client with local storage $|\sigma|$ has a bandwith overhead $\Omega\left(\frac{\log N}{\log |\sigma|}\right)$. We will actually use a very similar technique to the one employed for the ORAM lower bound to show two of the lower bounds in this paper.

The lower bound of Goldreich and Ostrovsky is actually limited by a very specific setting: it applies to a *balls and bin* memory model where the data is modeled as balls and server-side and client-side storage are bins, and moving data between bins is the only allowed operations (in particular, the server cannot perform any computation). As such, the result in [GO96] only lower bounds the communication overhead between the client and the server.

However, as mentioned by Devadas *et al.* in [DDF+16], even in the case where server computations are allowed, the Goldreich-Ostrovsky lower bound is in the number of operations that have to be performed: the overhead of the communication between the server and the client can be reduced below the lower bounds, but the computational overhead cannot. In this paper, we are actually going to take a similar approach.

Recently, a new line of work tried to fully understand the limits of the Goldreich and Ostrovsky lower bound, starting with the paper by Boyle and Naor [BN16], showing that, in the offline setting (where the queries are given ahead of time), in the RAM computation model (server-side computation is not allowed), by using specific data encodings (outside of the balls and bin model), the lower bouds can be overcome. This result was extended by Weiss and Wichs [WW18] to settings like read-only ORAM. Finally, Larsen and Nielsen proved in [LN18] a lower bound for online ORAM constructions, very similar to the original Goldreich-Ostrovsky lower bound, that applies to any data representation, even in the case where we only require computational complexity.

3 Background and Definitions

3.1 Preliminaries

In this paper, we use the following common notations. The security parameter is denoted λ and $\mathsf{negl}(\lambda)$ denotes a function that is negligible in the security parameter. We only consider (probabilistic) algorithms and protocols running in time polynomial in the security parameter λ. Adversaries are probabilistic polynomial-time algorithms. For a finite set X, $x \xleftarrow{\$} X$ means that x is sampled uniformly from X.

3.1.1 Games.

Our security and correctness notions are defined using the code-based games introduced in [BR06]. A game G is a set of oracle procedures – including an initialization Init procedure and a finalization Final procedure – that is executed with an adversary A, *i.e.* A has access to the procedures, with some possible restrictions. For instance, the Init oracle is always the first one to be called and Final the last one, once A halted, taking A's output as input. The output of Final is called the output of the game and is denoted $G^A(\lambda)$. When Final is omitted, it just forwards the adversary's output.

At startup, the boolean variables are initialized to false and the integer variables to 0. When the variable T is a dictionary, $T[v]$ denotes the item associated to v, if there is one, whereas $\perp$ denotes the absence of this item.

3.1.2 Protocols.

In the paper, we will construct and use some two-party protocols, involving a client C and a server S. We will denote a protocol P as

$$P(\mathtt{input}_C; \mathtt{input}_S) = (P_C(\mathtt{input}_C), P_S(\mathtt{input}_S))$$

meaning that P_C (resp. P_S) is executed by the client (resp. the server) with input $\mathtt{input}_C$ (resp. $\mathtt{input}_S$). We write

$$(\mathtt{out}_C; \mathtt{out}_S) \xleftarrow{\$} C(\mathtt{input}_C) \leftrightarrow S(\mathtt{input}_S)$$

to mean that $\mathtt{out}_C$ and $\mathtt{out}_S$ are the outputs of the interaction between C on input $\mathtt{input}_C$ and S on input $\mathtt{input}_S$. When C and S run a protocol P, we simplify the notation, and we denote the result of P as

$$(\mathtt{out}_C; \mathtt{out}_S) \xleftarrow{\$} P(\mathtt{input}_C; \mathtt{input}_S).$$

In this formalism, we consider the messages $\tau_{C \to S}$ (resp. $\tau_{C \leftarrow S}$) sent by C to S (resp. S to C) as part of the output $\mathtt{out}_C$ (resp. $\mathtt{out}_S$). These messages are called the *transcript* of C (resp. S). Transcripts might be omitted from the output of the protocol for simplicity.

3.2 Formalism of Symmetric Searchable Encryption

A *database* DB is defined as:

$$\mathsf{DB} = \{(\mathsf{ind}_i, \mathsf{W}_i) : 1 \leq i \leq D\},$$

with $\mathsf{ind}_i \in \{0,1\}^\ell, \mathsf{W}_i \in \{\{0,1\}^*\}^*$ and where ind_i are distinct *document indices*, represented by ℓ-bit strings, and W_i is a finite set of *keywords* matching document ind_i, represented by binary strings of arbitrary finite length. Note that, in this paper, a document is identified with its index: indeed we focus on index-based searchable encryption, where the documents are encrypted (and stored) separately from the data structure used to search among these. In addition, let us define:

- $D = |\mathsf{DB}|$ the number of documents;
- $\mathsf{DB}(w) = \{\mathsf{ind}_i | w \in \mathsf{W}_i\}$ the set of documents matching w;
- $\mathsf{W} = \bigcup_{i=1}^{D} \mathsf{W}_i$ the set of keywords;
- $K = |\mathsf{W}|$ the number of keywords;
- $K_n = |\{w | \mathsf{DB}(w) = n\}|$ the number of keywords matching n documents;
- $N = \sum_{i=1}^{D} |\mathsf{W}_i|$ the number of document/keyword pairs, also referred to as the *size* of the database;
- $\mathsf{DB}(w) = \{\mathsf{ind}_i | w \in \mathsf{W}_i \text{ and } (\mathsf{ind}_i, \mathsf{W}_i) \in \mathsf{DB}\}$ the set of documents containing the keyword w;
- $n_w = |\mathsf{DB}(w)|$ the number of documents matching w.

We consider structure-only, symmetric SSE schemes supporting single-keyword search. The dynamic schemes we consider support addition and deletion of keywords. Formally, a *dynamic symmetric searchable encryption (SSE) scheme* is a triple $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ consisting of one algorithm and two protocols between a client and a server:

- $\mathsf{Setup}(\mathsf{DB})$ is a probabilistic algorithm that is run by the client, and takes as input the initial database DB. It outputs a triple $(\mathsf{EDB}, K_\Sigma, \sigma)$, where K_Σ is the master secret key, EDB is an encrypted database, and σ is the client's state (his permanent memory).

- $\mathsf{Search}(K_\Sigma, q, \sigma; \mathsf{EDB})$ is a protocol between the client with input the master secret key K_Σ, the client's internal state σ, and a search query q; and the server with input the encrypted database EDB.

  After completing the Search protocol, the client outputs a list R of results and a new internal state σ'. Both R and σ' can take the special value $\perp$ to signify an error or a failure in the execution of the protocol. The server possibly outputs an updated encrypted database EDB'.

  The query q can be of any kind, but in this paper, we will focus on search queries restricted to a single-keyword w (and hence often identify q with w). More generally, $\mathsf{DB}(q)$ denotes the set of documents matching the query q. The transcript τ of the client (the messages sent by the client) is also included in its output when needed in the formal definitions.

- $\mathsf{Update}(K_\Sigma, \sigma, \mathsf{op}, \mathsf{in}; \mathsf{EDB})$ is a protocol between the client with input the key K_Σ and internal state σ, an operation op, and an input in for the operation; and the server with input EDB. Again, this formalism covers a wide range of different update operations, such as merges, duplications, *etc.* Yet, as we will see in the next section, this paper focuses on simpler operations: the update operations are taken from the set $\{\mathsf{add}, \mathsf{del}\}$, meaning, respectively, the addition and the deletion of a keyword to a document. The input in is thus parsed as an index ind, pointing to the modified document, a keyword w to insert or delete. Insertion of a new document is modeled by using a completely new and previously unused index ind. At the end of the execution of the protocol, the client outputs a new state σ', which can take the special value $\perp$, and the server outputs a new encrypted database EDB'. In the security definitions, as mentioned in Section 3.1, we also include the transcript τ of the client in its output.

These searchable encryption schemes are called *symmetric* because the same key K_Σ is used for both the updates and the search queries. One could extend the definition to support two different keys: a private key for search and a public key for updates, so that anyone can enrich the encrypted database (*e.g.* by sending an email encrypted with the public key). This setting, called 'Public key encryption with keyword search' (PEKS) has been well defined [BDOP04, BKOS07], but in this paper, we will focus on the symmetric setting. Similarly, some works looked at how to let anyone with a public key issue some search queries [BBO07].

In this formalism, we explicitly separate the client's state and the key. Informally, we want the key to be fixed while the state is mutable. We do not require a scheme to have a state (beyond the key), and when it is empty, we may omit it from the protocol's signature. Also, we do not require the size of the state to be upper bounded, *e.g.* by the number of keywords. For example, we could imagine a scheme working only on the client side, with no storage on the server. This would be a perfectly valid (and secure), but very costly scheme. Hence, the size of the client's state is an important parameter regarding the tradeoff between security and performance.

An important restriction of this definition is *static symmetric searchable encryption*. Such schemes do not support update requests, and hence do not implement the Update protocol.

Finally, in the rest of the paper, when we look at the computational complexity of a query, we are considering the sum of the complexities one the client side and on the server side.

Database structure. In our formalism, the document/keyword pairs (w, ind) such that $\mathsf{ind} \in \mathsf{DB}(w)$, also named *entries*, are the atomic elements of the databases. Entries are the elements that are manipulated by the protocols, and the encrypted database can be modeled as N atomic entries, each encoding a document/keyword pair of the dataset. This database representation is common to the entire SSE literature to our knowledge (*e.g.* [CGKO06, CK10, KO12, CJJ+13, CT14] and many more).

Our lower bounds rely on this representation, as it is closely related to the balls and bin model in use in the Goldreich-Ostrovky lower bound. Similarly to the work of Boyle and Naor [BN16], by using more complex data encoding, our lower bounds might be overcome.

3.3 Searchable Encryption Security Definitions

Modern definitions to formalize the secrecy of an SSE scheme come from [CGKO06]. In this paper, the authors give two definitions, one based on indistinguishability, the other based on simulatability, both using

the notion of leakage (a.k.a *trace* in [CGKO06]). Indeed, the leakage is formally taken into account and plays an important role in the security definitions: the indistinguishability-based definition states that two executions of SSE protocols with the same leakage are indistinguishable, while the simulation-based definition states that an execution of the SSE protocol can be simulated using the leakage. Informally, both definitions ensure that the server should not learn any information beyond the leakage (which is a parameter of the definitions).

In [CGKO06], the authors show that the simulation-based definition implies the indistinguishability-based definition (and that they are actually equivalent in their non-adaptive version). As, in our paper, we will only need the indistinguishability-based definition, only this version will be formally stated. Also, as the simulation-based definition is stronger than the indistinguishability-based one, it means that our results are entirely applicable to the more common former definition.

Leakage Function. Before going further, it is essential to clearly formalize this leakage. First let us define a history.

Definition 1 (Database and queries history)**.** *An* history H *is a tuple* $H = (\mathsf{DB}, r_1, \ldots, r_m)$ *consisting of a database* DB *and* m *queries* $r_1, \ldots, r_m$*. Each query* r_i *can either be a search query* $r_i = q_i$*, or an update query* $r_i = (\mathsf{op}_i, \mathsf{in}_i)$*.*

To do so, as explained before, the definition will be parametrized using a *leakage function* $\mathcal{L}$, more exactly a triple of stateful algorithms $(\mathcal{L}^{\mathsf{Stp}}, \mathcal{L}^{\mathsf{Srch}}, \mathcal{L}^{\mathsf{Updt}})$, capturing what is leaked by, respectively, the setup algorithm, the search protocol and the update protocol.

This notation, introduced by Chase and Kamara for the generic case of structured encryption in [CK10], generalizes the trace definition of Curtmola *et al.* [CGKO06]. Because the leakage function is stateful, it will not be necessary to pass the whole history as an argument of the leakage function every time, as in [CJJ+13].

A very common leakage pattern in searchable encryption is the repetition of search queries: often, the tokens sent by the client to the server are deterministically generated, *e.g.* with a PRF. As a consequence, if a token sent to the server during a search query is generated using only the queried keyword, he will immediately detect when this query is repeated. This leakage, call the *search pattern*, is formally defined as follows: the leakage function $\mathcal{L}$ keeps in its state the *query list* Q, the list of all queries issued so far, and whose entries are (i, w) for a search query on keyword w, or $(i, \mathsf{op}, \mathsf{in})$ for an op update query with input in. The integer i is a timestamp, initially set to 0, and is incremented at each query. The search pattern of a search query r is then defined as $\mathsf{sp}(x) = \{j \mid (j, x) \in Q\}$ (this only matches search queries). Phrased differently, the search pattern of the keyword w corresponds to the list of search queries' timestamps whose searched keyword was w.

In the following, we will slightly overload the notations, and, for a history $H = (\mathsf{DB}, r_1, \ldots, r_m)$, use $\mathcal{L}(H)$ to denote $(\mathcal{L}^{\mathsf{Stp}}(\mathsf{DB}), \mathcal{L}(r_1), \ldots, \mathcal{L}(r_m))$ where $\mathcal{L}(r_i) = \mathcal{L}^{\mathsf{Srch}}(q_i)$ if r_i is a search query, and $\mathcal{L}(r_i) = \mathcal{L}^{\mathsf{Updt}}(\mathsf{op}_i, \mathsf{in}_i)$ if r_i is an update query.

Security definition. The indistinguishability-based security definition of Curtmola *et al.* [CGKO06] can be reformulated (equivalently) using a security game, $\mathrm{SSEIND}_{\Sigma,\mathcal{L}}$, parametrized by the scheme Σ and the leakage function $\mathcal{L}$, which picks a random bit b, and to which the adversary's goal is to guess b. To do so, the adversary (adaptively) submits two histories H_0 and H_1, and receives the encrypted database and runs the server-side part of the Search and Update protocols (the client-side part being run by the game). To be valid, the histories submitted by the adversary must satisfy a very important constraint: the leakage must be the same for both histories. Otherwise, the game aborts as soon as the adversary submits two queries (or two databases) having a different leakage. Finally, the adversary outputs a bit b', and wins the game if he successfully guesses b.

In the 'honest-but-curious' (or 'passive') adversarial setting, the adversary has to follow the specification of the Search and Update protocols to the letter: this can be seen as the adversary only receiving the resulting transcript of the execution of the protocols.

$\underline{\mathsf{Init}(\mathsf{DB}_0, \mathsf{DB}_1)}$
if $\mathcal{L}^{\mathsf{Stp}}(\mathsf{DB}_0) \neq \mathcal{L}^{\mathsf{Stp}}(\mathsf{DB}_1)$
Abort game
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{EDB}, K_\Sigma, \sigma) \xleftarrow{\$} \mathsf{Setup}(\mathsf{DB}_b)$
return EDB
$\underline{\mathsf{Final}(b')}$
return $b = b'$

$\underline{\mathsf{Search}(q_0, q_1)}$
if $\mathcal{L}^{\mathsf{Srch}}(q_0) \neq \mathcal{L}^{\mathsf{Srch}}(q_1)$
Abort game
$(R, \sigma, \tau; \mathsf{EDB}) \xleftarrow{\$} \mathsf{Search}_C(K_\Sigma, \sigma, q_b) \leftrightarrow A$
return τ
$\underline{\mathsf{Update}((\mathsf{op}_0, \mathsf{in}_0), (\mathsf{op}_1, \mathsf{in}_1))}$
if $\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}_0, \mathsf{in}_0) \neq \mathcal{L}^{\mathsf{Updt}}(\mathsf{op}_1, \mathsf{in}_1)$
Abort game
$(\sigma, \tau; \mathsf{EDB}) \xleftarrow{\$} \mathsf{Update}_C(K_\Sigma, \sigma, \mathsf{op}_b, \mathsf{in}_b) \leftrightarrow A$
return τ

Figure 1 – $\text{SSEIND}_{\Sigma,\mathcal{L}}$: Indistinguishability game for the SSE scheme $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$, with the leakage function $\mathcal{L}$. The notation $\leftrightarrow A$ represents interactions with the adversary.

On the other hand, in the 'malicious' (or 'active') setting, the adversary can try to get additional information by deviating from the protocols, and is allowed to do so in the definition. This is particularly important for schemes whose protocols use more than a single round-trip, *e.g.* ORAM-inspired constructions such as the ones described in [SPS14, GMP16]. A single round-trip scheme secure against honest adversaries is secure against malicious ones because no adversary can, by construction, influence the messages sent by the client (*i.e.* the client's transcript) by providing incorrect or incomplete responses. To the contrary, when multiple round-trips are involved, the adversary can trick the client by sending an incorrect message so that the client reveals sensitive information later in the search or update protocol.

Both these settings can be formalized using the security game of Figure 1, leading to Definition 2. Note that, we introduce the case of an unbounded adversary in our definition, as our lower bounds will be proven against such adversaries.

Definition 2. *Let Σ be an SSE scheme. For an adversary A, the advantage $\mathsf{Adv}^{\text{SSE-ind}}_{\Sigma,\mathcal{L},A}(\lambda)$ of A in the indistinguishability-based confidentiality game is*

$$\mathsf{Adv}^{\text{SSE-ind}}_{\Sigma,\mathcal{L},A}(\lambda) = \left| \frac{1}{2} - \mathbb{P}[\text{SSEIND}^{A}_{\Sigma,\mathcal{L}}(1^\lambda) = 1] \right|.$$

An SSE scheme Σ is $\mathcal{L}$-adaptive-indistinguishability secure *if for any polynomial-time adversary A, $\mathsf{Adv}^{\text{SSE-ind}}_{\Sigma,\mathcal{L},A}(\lambda)$ is negligible in λ. If the adversary A is unbounded, we say that the scheme is* $\mathcal{L}$-unconditionally secure.

3.4 An Order Relation over Leakage Functions

To formally state our lower bounds, we have to give a formal definition to the proposition "$\mathcal{L}_1$ leaks less than $\mathcal{L}_2$". By that, we mean that any information given by $\mathcal{L}_1$ can be inferred from $\mathcal{L}_2$. To do so, we use the order relation on leakage functions introduced in [Bos18] (similar to the leakage upper bounds defined in [KMO18]).

Definition 3 (The order $\preceq$ on leakage functions)**.** *Let $\mathcal{L}_1 = (\mathcal{L}_1^{\mathsf{Stp}}, \mathcal{L}_1^{\mathsf{Srch}}, \mathcal{L}_1^{\mathsf{Updt}})$ and $\mathcal{L}_2 = (\mathcal{L}_2^{\mathsf{Stp}}, \mathcal{L}_2^{\mathsf{Srch}}, \mathcal{L}_2^{\mathsf{Updt}})$ be two leakage functions. We say that $\mathcal{L}_1$* leaks less than *$\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$ if and only if, there exists a triple of* stateful polynomial-time *algorithms $\mathcal{T} = (\mathcal{T}^{\mathsf{Stp}}, \mathcal{T}^{\mathsf{Srch}}, \mathcal{T}^{\mathsf{Updt}})$, such that, for any database* DB *and sequence of queries $(r_1, \ldots, r_n)$,*

$$\begin{aligned} \mathcal{L}_1^{\mathsf{Stp}}(\mathsf{DB}) &= \mathcal{T}^{\mathsf{Stp}} \circ \mathcal{L}_2^{\mathsf{Stp}}(\mathsf{DB}), \\ \forall 1 \leq i \leq n, \mathcal{L}_1^{\mathsf{Srch}}(r_i) &= \mathcal{T}^{\mathsf{Srch}} \circ \mathcal{L}_2^{\mathsf{Srch}}(r_i) \text{ for search,} \\ \text{and } \mathcal{L}_1^{\mathsf{Updt}}(r_i) &= \mathcal{T}^{\mathsf{Updt}} \circ \mathcal{L}_2^{\mathsf{Updt}}(r_i) \text{ for update.} \end{aligned}$$

In other words, $\mathcal{L}_1$'s output is *simulatable* from $\mathcal{L}_2$'s output. Also note that $\preceq$ is a partial order. An important (and easy) application of this relation in our paper, is that if a scheme is $\mathcal{L}$-adaptively-secure, then it is $\mathcal{L}'$-secure for every $\mathcal{L} \preceq \mathcal{L}'$.

Proposition 1. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two leakage functions such that $\mathcal{L}_1 \preceq \mathcal{L}_2$. If Σ is a $\mathcal{L}_1$-adaptively-secure SSE scheme, then Σ is $\mathcal{L}_2$-adaptively-secure.*

4 Searchable Encryption Hiding the Search Pattern

One crucially important question about searchable encryption is the one of the tradeoff between the security of constructions and their performance. We would like to design efficient schemes with the best possible security, or, symmetrically, design highly secure schemes with the best performance. In particular, legacy-compatible constructions are not really secure (see [CGPR15, NKW15]), although being very efficient (asymptotically as efficient as unencrypted databases). In this section, we will see that some minimal leakage is absolutely necessary to achieve even reasonable performance.

4.1 An Efficiency Lower Bound on Search Pattern Hiding SSE

In almost every construction, except for the ones based on ORAM, the search pattern is leaked: the server learns the repetition of search queries. One can wonder if the cost of ORAM is necessary to hide the search pattern, or if there is another means to achieve this security level without paying the high cost of ORAM.

A first step towards this goal is to study a simple reduction from ORAM to SSE (*i.e.* implement an ORAM protocol from a search-pattern-hiding SSE scheme): we use the address of ORAM blocks as keywords for the SSE scheme, and the blocks' data as the documents indices. In this artificial encrypted database, each keyword and each document index only appears once, and we have $K = D = N$.

As a consequence, by applying the ORAM lower bounds [GO96, LN18], this reduction would only offer us a $\Omega(\log K)$ lower bound on the search complexity of a search-pattern hiding of SSE. On the other hand, by using a naive ORAM-based implementation of SSE, we expect having a search complexity of the order of $n_w \log N$.

Differences between ORAM and search-pattern-hiding SSE Indeed, there is an important difference between ORAM and search-pattern-hiding SSE that has an impact on the way to construct such schemes. ORAM only supports access to fixed-size blocks, and each access fetches a single block. As a consequence, in the above reduction, the underlying SSE construction only has to support a database with a single match per document, while the specificity of SSE is to be able to return a set of results, a set which can be of arbitrary size (but revealed to the adversary). This difference is crucial when studying what are the information the adversary is allowed to learn, and the one which will have to be hidden from him.

Namely, in the case of two search queries to an SSE scheme, suppose that the first query matches n_1 entries. If the second query matches $n_2 \neq n_1$ entries, the adversary will immediately learn that the queries are different, and hence that the result entries are different. Yet, if $n_2 = n_1$, this is not true anymore, and we have to hide the possible repetition of the first query in this case.

Similarly, the server already knows that he will have to access at least n_w distinct entries during a search query on w: we do not have to hide this information, and we can use this fact to speed the search algorithm up. Also, as we are only interested in the result set, not in the result order, we might be able to chose the order that minimizes number of operations. We will see that both of these points play a huge role in the proof of the SSE lower bound.

Summary of the Goldreich-Ostrovsky lower bound for ORAM In order to sketch the proof for the SSE lower bound, we have to give an insight of the proof given in [GO96] for ORAM. In that paper, Goldreich and Ostrovsky model the ORAM security using a balls-and-bins game involving a (probabilistic) player, who can hold at most b balls representing the data, and which impersonates the client, and an

observer (the server/adversary). The (external) memory is represented as an array of m non-transparent cells, each capable of storing a single ball, the total number of balls being m. The player is allowed to access any cell and perform some actions on these (get the ball in the cell, place a ball in the cell, or do nothing). The observer sees which cell is accessed, but cannot see the action, nor the content of the cell.

For each round of the game, the player has to answer to a (secret) access request r for one of the m balls: his goal is to hold the r-th ball at the end of the round. Yet, he must not leak any information to the observer about the value of r: the observer must not learn anything about the request sequence. Hence, any action sequence (*i.e.* a sequence of memory accesses by the player and their associated action) must be 'compatible' with any possible request.

The [GO96] proof hence proceeds using a counting argument, by lower bounding the number of request sequences that can be satisfied by a given action sequence of length q in t rounds. The resulting inequality gives a relation between the number of actions q, the number of access t, the size of the client memory b, and the size of external memory m: this gives us the lower bound on the number of actions, *i.e.* the overhead, necessary to obliviously access a memory value (a ball).

Our result The main difference between our result and the ORAM lower bound lies in the fact that counting the number of possible search queries at each step is much more complicated than in the original proof, due to the functional differences between ORAM and SSE, as we studied earlier. For example, because the number of matches between queries might differ and is leaked to the observer, we have to take it into account: for the $i+1$-st search query, the query being done on keyword w, the observer will know that the searched keyword is not in the set $\{w_j | 1 \leq j \leq i \text{ and } n_{w_j} \neq n_w\}$, *i.e.* the set of previously searched keywords matching a number of documents different from the one of w. More precisely, if there has been a previous request w_j with $n_w \neq n_{w_j}$, the observer knows that n_{w_j} entries will not be 'touched' by the search algorithm for the $i+1$-st query, making the number of entries 'available' be $N - n_{w_j}$ (for a database of size N).

This reasoning is applicable to the other previous queries whose number of match is different from n_w: if $n_{w_j} \neq n_{w_k} \neq n_w$, we do not have to consider $n_{w_j} + n_{w_k}$ entries. Yet, if there are two queries w_j and w_k such that $n_{w_j} = n_{w_k}$, we cannot remove from consideration $2n_{w_j}$ entries: we can be in a case where $w_j = w_k$.

If we do this with every query preceding w, for a partial history $H_i = (\mathsf{DB}, w_1, \ldots, w_i)$, the number entries that can be considered is

$$\overline{N}(H_i, w) = N - \sum_{n \in \{|\mathsf{DB}(w_j)| \neq |\mathsf{DB}(w)|\}} n.$$

In the proof, we then have to consider how many search requests having n_w results can be satisfied with $\overline{N}(H_i, w)$ candidate entries. This is done using simple combinatorics: there are $\binom{\overline{N}(H,w)}{n_w}$ such possibilities. We end up the proof by simplifying the inequality we obtained.

Our whole proof heavily relies on the fact that the studied scheme only leaks the number of entries and the number of distinct keyword in the database, and that the search protocols only reveals the number of matches of a query. Formally, we use the $\mathcal{L}_{\overline{\mathsf{sp}}}$ leakage function, defined as

$$\mathcal{L}^{\mathsf{Stp}}_{\overline{\mathsf{sp}}}(\mathsf{DB}) = (N, K), \quad \mathcal{L}^{\mathsf{Srch}}_{\overline{\mathsf{sp}}}(w) = n_w.$$

This leads to the formal statement of our theorem:

Theorem 1. *Let Σ be a (static) SSE scheme that is (non-adaptively) $\mathcal{L}$-unconditionally secure (* i.e. *secure against unbounded adversaries), with $\mathcal{L} \preceq \mathcal{L}_{\overline{\mathsf{sp}}}$, with a client state (permanent memory) σ of size $|\sigma|$, and an encrypted database containing N entries exactly. Then the overall computational complexity of the* Search *protocol (the sum of the computational complexity for the client and the server) for a keyword w matching n_w documents, after the execution of the history H is*

$$\Omega\left(\frac{\log\binom{\overline{N}(H,w)}{n_w}}{\log|\sigma| \cdot \log\log\binom{\overline{N}(H,w)}{n_w}}\right).$$

Proof. To start the proof, we are going to introduction that the client uses at most b (transient) memory blocks to run the query on w (b can depend on the history). Without loss of generality, we can suppose that the adversary only gets to see the read (and write) accesses to the encrypted database generated by the search protocol: this will only reduce his capabilities. We also only consider a non-adaptive attacker who chooses two fixed histories. In this setting, the SSE indistinguishability game can be modeled as follows:

- a player, who can hold at most b balls, makes (probabilistic) accesses to the encrypted database to answer the sequence of t search queries corresponding to the keywords $(w_1, \ldots, w_t)$ on the database DB;
- an observer, who gets to see the accesses.

The observer/adversary will win the game if he is able to distinguish the execution of two search sequences $(w_1, \ldots, w_t)$ and $(w'_1, \ldots, w'_t)$, respectively on the databases DB and DB' such that $|\mathsf{DB}| = |\mathsf{DB}'|$ and $|\mathsf{DB}(w_i)| = |\mathsf{DB}'(w'_i)|$ for $i = 1, \ldots, t$.

The encrypted database is modeled as N atomic entries, each encoding a document/keyword pair of the dataset, equivalent to the balls in the ORAM lower bound proof of Goldreich and Ostrovsky [GO96]. These entries are stored in non-transparent cells holding a single entry. At any time the player accesses a cell, he can either fetch the entry residing in this cell, place an entry in it, or do nothing. The observer will see that the player accessed this cell, but not what he just did with it. It is important to note that it is not because the scheme is static that the encrypted database cannot be modified by the Search algorithm: the fact that the scheme is static only implies that it does not support Update operations.

To answer these t search queries, the player will make a sequence of q *visible* accesses $V = (v_1, \ldots, v_q)$, observable by the adversary. For each access v_i, the player will perform a *hidden action* h_i, which the observer cannot see. As mentioned before, the player can take an entry from the cell, place an entry in the cell or do nothing. In particular, there are $b + 2$ possible actions (b 'placing' actions, the 'taking' action, and the 'nothing' action). This action sequence $(v_1, h_1), \ldots, (v_q, h_q)$ satisfies the search queries sequence $(w_1, \ldots, w_t)$ if and only if there exists a sequence $1 = j_0 \leq j_1 \leq \cdots \leq j_t \leq q$ such that for every i-th search request, the player has held every entry corresponding to the entries (w_i, ind), for $\mathsf{ind} \in \mathsf{DB}(w_i)$, after the actions $(v_{j_{i-1}+1}, h_{j_{i-1}+1}), \ldots, (v_{j_i}, h_{j_i})$. We note $\delta q_i = j_i - j_{i-1} + 1$.

Let us focus on the i-th search request. As the player holds at most b entries, a fixed sequence $(v_j, h_j), \ldots, (v_{j+\delta q}, h_{j+\delta q})$ can satisfy at most $(b+1)^{\delta q}$ different search queries: at every step one can pick one or zero ball to form the result (this is a very large upper bound on the number of entries that the player can fetch in δq actions). Also, each δq-long visible accesses sub-sequence $(v_j, \ldots, v_{j+\delta q})$ may be coupled with $(b+2)^{\delta q}$ hidden action sequences. In the end, we can say that each sub-sequence δV_i of length δq_i can satisfy at most $(b+1)^{\delta q_i}(b+2)^{\delta q_i}$ search queries.

Finally, as the search pattern is hidden, but the size pattern is revealed, the number of search queries matching n documents in the database DB after the search queries $w_1, \ldots, w_i$ is $\binom{\overline{N}(H,w)}{n}$, where $H = (\mathsf{DB}, w_1, \ldots, w_i)$ is the history of previous executions. The reason for this non-trivial expression is first that the order in which the entries are fetched does not matter (we are only interested in the result set), then that the player does not have to hide that he does not access the entries that were previously fetched for searches matching a different number of results. Namely, after the execution of the history H, there are at most $\overline{N}(H, w)$ non-touched entries matching $n \neq n_w$ entries: the encrypted database consists, by hypothesis, of N entries, and there have been $\sum_{j=1, |\mathsf{DB}(w_j)| \neq |\mathsf{DB}(w)|}^{i} |\mathsf{DB}(w_j)|$ entries touched by previous search queries. As the i-th sub-sequence V_i, of length δq_i must satisfy $\binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}}$ possible search queries, we have

$$(b+1)^{\delta q_i}(b+2)^{\delta q_i} > \binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}}$$
$$\implies \quad \delta q_i = \Omega\left(\frac{\log \binom{\overline{N}(H_{i-1}, w_i)}{n_{w_i}}}{\log(b+1)(b+2)}\right).$$

The computational complexity of the search protocol is

$$\Omega\left(\frac{\log\binom{\overline{N}(H,w)}{n_w}}{\log(b+1)(b+2)}\right).$$

The memory b used during the execution of the Search protocol can be separated into two parts. First, there is the client's state, his permanent memory, and then the additional transient memory b' used by either the client or the server to execute the protocol itself: $b = |\sigma| + b'$. Without loss of generality, we can suppose that the Search protocol touches each of the b' memory blocks at least once: these memory blocks are local to the client, and not observed by the adversary, so if one of these blocks is not touched, then the protocol does not need it (there is no need for dummy accesses). Hence, the complexity of the protocol is at least $\Omega(b')$.

To conclude, we use the previous bounds on the computational complexity and apply the following lemma, whose proof is given in Appendix B.

Lemma 2. *Let $C > 8$, $D \geq 0$ and $f : [D+1, +\infty) \to [1, +\infty)$ defined as $f(x) = \max\left\{x - D, \frac{C}{\log(x+1)(x+2)}\right\}$. Then $\frac{C}{4\log(D+2)\cdot\log C} - 1 < f(x)$ for all x in $[D+1, +\infty)$.*

From all what precedes, and by applying the lemma with $C = \log\binom{\overline{N}(H,w)}{n_w}$ and $D = |\sigma|$, we have that the computational complexity of a search query is

$$\Omega\left(\frac{\log\binom{\overline{N}(H,w)}{n_w}}{\log|\sigma| \cdot \log\log\binom{\overline{N}(H,w)}{n_w}}\right).$$

□

The $\log\log$ factor, which is not present in the ORAM lower bound of [GO96], comes from the fact that we consider separately the transient and the non-transient (*i.e.* the state) memory of the client, although it is, in some sense an artifact of the proof. As explained at the end of Appendix B, we can improve the result of Theorem 1, asymptotically get rid of the artificial $\log\log$ factor in the denominator, and end up with the following lower bound on the computational efficiency of search-pattern hiding schemes: $\Omega\left(\frac{\log\binom{\overline{N}(H,w)}{n_w}}{\log|\sigma|}\right)$.

The binomial coefficient in the lower bound also has a practical interpretation: it leverages the fact that, when the protocol fetches the matching entries, a single entry cannot be accessed twice, and that the order in which they are accessed does not matter. If it had, we would have a bound in $\frac{\overline{N}(H,w)!}{n_w!}$. This leaves space for practical optimizations, as the entries corresponding to the documents matching the search query will have to be (randomly) relocated only once they have all been accessed (leaking the search pattern otherwise), the player can optimize the number of visible accesses he makes, something he cannot do with ORAM: it is not a problem to leak that different entries/balls were accessed during a given search query as this information is obvious.

It is interesting to see that, when all the keywords only have a single matching document, *i.e.* in a setting similar to the ORAM one (see the discussion at the beginning of the section), our lower bound matches exactly the ones of [GO96] and [LN18].

Note that the lower bound was proven in a non-adaptive setting: we did not rely on the capacity of the adversary to adaptively query the scheme. It is unclear whether adaptivity would actually have an impact on this lower bound. Also, Theorem 1 only targets static SSE schemes (*i.e.* without updates), but a similar result can be given for dynamic schemes hiding both the search and update patterns. Indeed, Section 5 treats the case of forward-private schemes, *i.e.* schemes that do not leak information about the updated keywords.

4.2 Discussion about the Lower Bound

Here, we want to quickly discuss about limitations of the above result, and possible ways to lift them.

About the restriction on the encrypted database. The previous result heavily relies on the fact that the encrypted database consists of N entries exactly. One way to overcome the lower bound would indeed be to duplicate the encrypted database, to make non-oblivious accesses to the first database for the first query, and then oblivious accesses to the second database for the other queries.

Hence, the restriction on the number of database entries is crucial in the proof. However, all existing SE schemes (*e.g.* [CGKO06, KO12, CJJ+13, Bos16, BMO17]) fall under this restriction. Also, it does not preclude an encrypted database containing more than the entries strictly speaking: it could also store additional information that is used during the search protocol by the client and/or the server. Accesses to this part of the memory are not considered in our proof, we only look at the memory blocks containing the entries of the database. Similarly, adding dummy entries to the database might help designing an $\mathcal{L}_{\overline{\mathsf{sp}}}$-secure scheme, as it is when constructing ORAM schemes, without impacting the lower bound, as in the ORAM case. Indeed, in both proofs, what really matters in the number of useful balls, *i.e.* the balls with the actual content. The other balls, such as the ones containing meta-data and dummy entries, are not accounted for.

The data representation As mentioned in Section 3.2 and described in the previous section, the lower bound is proven in some kind of balls and bins model, inheriting its weaknesses such as the simplicity of the data representation. Although Larsen and Nielsen overcame these limits for ORAM in [LN18] by using a different model, called the oblivious cell probe model, applying this model to static SSE to obtain a lower bound similar to the one of Theorem 1 looks far from trivial due to the differences between ORAM and SSE (especially the variable number of results). We leave that as an intriguing open question.

Postponing result delivery. The way we defined the search functionality of SSE encryption schemes is very important in the proof : we require that, at the end of the execution of the Search algorithm, all the results of the search query must be returned to the client. This actually plays an important role in the proof. This constraint is satisfied by all the existing SSE schemes, except the piggyback scheme (PBS) of Kamara *et al.* [KMO18].

Indeed, the PBS transforms the data so that a fixed number of entries are fetched during each query (called a batch), and it queues incoming queries so they can be processed at a later time. As stated by the authors, this trades leakage for latency: to be sure to get all the results of a query, the client has to wait for the execution of subsequent queries. As such, PBS does not satisfy our hypothesis, and might overcome our lower bound (although computing the exact efficiency of the scheme is not trivial as is heavily depends on the keyword and on the query distribution).

Practicality of the lower bound Although it is unclear if leaking the search pattern can lead to a real attack in general, in settings where the distribution of search queries is known, it can definitely be an issue. Studying the $\mathcal{L}_{\overline{\mathsf{sp}}}$ leakage is interesting though, as this is essentially the minimal leakage for an "efficient" SSE scheme running is time less than $\mathcal{O}(N)$. As such, our result is a theoretical refinement to Naveed's study of the composition of ORAM and SSE [Nav15]: it shows that targeting the the $\mathcal{L}_{\overline{\mathsf{sp}}}$ leakage (called $\mathcal{LC}1$ in [Nav15]) cannot be practical, even without relying directly on ORAM.

Also, a search-pattern-hiding scheme does not protect against all attacks: the count attack of Cash *et al.* [CGPR15] uses the number of matches of a query to recover the queried keyword when the adversary knows the database, for keywords that have a unique number of matching documents in the data set. However, such leakage-abuse attacks are beyond the scope of this paper.

4.3 On the Tightness of the Lower Bound

It is interesting to see that this lower bound does not hold anymore when the leakage is slightly increased: suppose that besides N, the setup also leaks, for each $n \in \mathbb{N}$, the number K_n of keywords w such that $|\mathsf{DB}(w)| = n$. We can then design a scheme Σ whose search complexity for a keyword w is $\mathcal{O}\left(n_w \cdot \log^2 K_{n_w}\right)$. To do so, the construction uses a derivative from ORAM, an oblivious map (or oblivious dictionary) data structure, as defined by Wang *et al.* in [WNL+14], which has the same functionalities as a regular associative

map with the additional security property that a query, whether is it a read or a write query, does not reveal any information about the accessed key, nor about its associated value.

Formally, an oblivious map (OMap) consists of one Setup algorithm, and two protocols, Read and Write. Setup takes a (regular) map as input and outputs the oblivious map data structure OMap *per se*, that will be given to the server, and the corresponding client state OMapState. To run Read, the client gives a key and the OMap state, and the server takes the OMap structure, and the client is returned the matching value to the key (or $\perp$ if the key matches nothing). Finally, $\mathsf{Write}(k, v, \mathsf{OMapState}; \mathsf{OMap})$ modifies the structure and the state so that subsequent calls to Read on key k return v. Note that OMap and OMapState can be modified during the execution of both protocols. The obliviousness security property required for an OMap states that the execution of two same-size sequences of Read and Write queries are indistinguishable to the server. A formal description of oblivious maps and their security properties is given in Appendix A.

Concretely, for a map storing N elements of size B, the oblivious map constructed in [WNL+14] has a client state of $\mathcal{O}(B \cdot \log N)$ bits, and the computational complexity of an access is $\mathcal{O}\left(B \cdot \log^2 N\right)$. Note that the client state can also be stored on the server, at the expense of downloading, re-encrypting and re-uploading the state after each access. This does not change the asymptotic complexity of the oblivious map, and this is the oblivious map instantiation that we will use here.

Algorithm 1 Description of Melinoe.

Setup(DB)

$\mathsf{OMapState}, \mathsf{OMap}, \mathbf{W} \leftarrow$ empty maps
for all n such that $\exists w, |\mathsf{DB}(w)| = n$ **do**
  $\mathsf{Tmp} \leftarrow$ empty map
  for all w such that $|\mathsf{DB}(w)| = n$ **do**
    $\mathsf{Tmp}[w] \leftarrow \mathsf{DB}(w)$
    $\mathbf{W}[w] \leftarrow n$
  end for
  $(\mathsf{OMapState}[n], \mathsf{OMap}[n]) \leftarrow \Theta.\mathsf{Setup}(\mathsf{Tmp})$
end for
return $(\mathsf{OMap}, \varnothing, (\mathbf{W}, \mathsf{OMapState}))$

Search(K_Σ, w, σ; EDB)

Client:
$n \leftarrow \mathbf{W}[w]$
Run $w \leftarrow \Theta.\mathsf{Read}(w, \mathsf{OMapState}[n]; \mathsf{OMap}[n])$ with the server. ▷ *Access w in the n-th OMap: run the oblivious map data structure protocol between the client and the server.* $\mathsf{OMapState}[n]$ *gets updated in the process.*
Parse the value as $(\mathsf{ind}_1, \ldots, \mathsf{ind}_n)$.
return $(\mathsf{ind}_1, \ldots, \mathsf{ind}_n)$.

The SSE scheme matching the complexity claimed above, Melinoe, is described in Algorithm 1. For each n such that $K_n \neq 0$, Σ initializes an oblivious map (OMap) of size K_n where each block corresponds to a keyword w such that $|\mathsf{DB}(w)| = n$ and stores $\mathsf{DB}(w)$. Also, for each keyword $w \in K$, the client stores n_w. Then a search query only consists in reading the block corresponding to the search keyword w in the OMap for keywords with n_w results. With the OMap described in [WNL+14], this only requires $\mathcal{O}(n_w \log K_{n_w})$ operations as we use K_{n_w} blocks of n_w keywords.

Melinoe can be shown $\mathcal{L}_{\text{size}}$-adaptively-secure with $\mathcal{L}_{\text{size}}$ defined as

$$\mathcal{L}^{\mathsf{Stp}}_{\text{size}}(\mathsf{DB}) = \{(n, K_n)\} = \Big\{\big(n, |\{w | DB(w) = n\}|\big)\Big\},$$
$$\mathcal{L}^{\mathsf{Srch}}_{\text{size}}(w) = n_w.$$

If we use the OMap described earlier, our construction has a $\mathcal{O}(K)$ client state (we have to store at most one OMap state and one integer per keyword in the worst case), and the search complexity for the keyword

w is $\mathcal{O}\left(n_w \cdot \log^2 K_{n_w}\right)$, as the OMap associated with w has K_{n_w} elements of size n_w. We can also outsource $\mathbf{W}$ and OMapState using an OMap, without altering the leakage profile (accesses to w are hidden by the use of an OMap). In this variant, which is formally described in Algorithm 3, the client has a constant-sized state, but the search complexity increases to $\mathcal{O}\left(\log^2 K + n_w \cdot \log^2 K_{n_w}\right) = \mathcal{O}\left(n_w \cdot \log^2 K\right)$.

Let us now consider a database such that $N = K^\alpha$, and with a set of keywords w such that $n_w = L$ (constant). We suppose that we want to repeatedly want to search for one of these keyword. In the Melinoe construction, the search complexity is $\mathcal{O}\left(L \log^2 K\right)$, while the lower bound gives us $\Omega\left(\log \binom{K^\alpha}{L}\right)$. From Stirling's formula, $\binom{K^\alpha}{L} \sim \frac{1}{\sqrt{2\pi L}} \cdot \frac{K^{\alpha L}}{L^L}$, and the lower bound ends up being $\Omega(L\alpha \log K)$. By taking $\alpha = \log^2 K$, we can see that the lower bound is completely broken by the Melinoe scheme.

4.4 Communication vs Computational Complexity

Theorem 1 states a lower bound on the total computational complexity of the client and the server during a search query, but does not specify how the work is shared between the two. This is similar to the ORAM lower bound of [GO96], as explained in [DDF+16]. Indeed, in the setting of [GO96], the server is not allowed to do computations, so we end up with a lower bound on the bandwidth overhead. But once the server is allowed to 'help' the client, this bandwidth overhead does not hold anymore, but the total number of atomic operations still has to satisfy the lower bound.

We are exactly in the same case with our SSE lower bound: if the server is supposed to be completely passive (*i.e.* act like some basic storage device), our computational lower bound will also be a communication lower bound. On the other hand, the searchable encryption scheme could be implemented as a garbled RAM program (see [LO13]) executed by the server on its own: the size of the search token (*i.e.* the garbled program) would only depend on the security parameter, and there would not be any bandwidth overhead, whereas the computational overhead is very high.

This shows that even a small relaxation in the hypothesis of Theorem 1 invalidates the lower bound.

5 Forward Privacy

In this section, we study a very important security property for dynamic searchable encryption schemes, called *forward privacy*. Informally, it states that updates do not leak information about the updated keywords. We start with a motivation and the formal definition of forward privacy, continue by presenting a lower bound on the efficiency of the update protocol of a forward-private searchable encryption scheme, and finish this section by presenting constructions that tightly match this lower bound.

5.1 File Injection Attacks and Forward Privacy

File injection attacks aim at breaking the confidentiality of the user's queries by injecting adversarially-controlled documents in the database (think of an encrypted email service attacked using spam). The first of these attacks was presented by Cash *et al.* in [CGPR15], but only targeted legacy-compatible encrypted databases, with much more leakage than what we usually consider in this paper.

Zhang *et al.* [ZKP16] improved this attack against any dynamic scheme leaking, during a search query, the results of the search or when the matching documents have been inserted in the database. They also presented a devastating adaptive variant of this attack that uses the update leakage, namely which are the previous search queries that the inserted document matches.

More specifically, their attack reveals the keyword w associated to a past search query q by inserting $\log 2T$ documents if the adversary knows the keyword distribution in the database, or $K/T + \log T$ new documents if he does not. Here, T is the *threshold parameter*, a public parameter used to thwart the non-adaptative version of the file injection attack, and that needs to be kept small to efficiently counter it (but not too small, as it has an impact on the efficiency of the schemes). The value $T = 200$ is used in the experiments

of [ZKP16]: inserting 8 fake documents is sufficient for the adversary to break the confidentiality of any search query.

Hence, to avoid this attack, it is necessary that dynamic searchable encryption schemes are *forward-private*, *i.e.* leak no information about the updated keywords during an insertion. This notion, that is now a *de facto* standard for dynamic searchable encryption schemes [GMP16, KKL+17, BMO17, EKPE18], was introduced by Stefanov *et al.* in [SPS14], and formalized by Bost in [Bos16]. We restate this formalization in the following definition.

Definition 4 (Forward privacy)**.** *An $\mathcal{L}$-adaptively-secure SSE scheme Σ is* forward private *if the update leakage function $\mathcal{L}^{\mathsf{Updt}}$ can be written as*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{op}, \mathsf{in}) = \mathcal{L}'(\mathsf{op}, \{(\mathsf{ind}_i, \mu_i)\})$$

*where $\{(\mathsf{ind}_i, \mu_i)\}$ is the set of modified documents paired with the number μ_i of modified keywords for the updated document ind_i, and $\mathcal{L}'$ is a stateless function (*i.e. *does not depend on previous queries).*

5.2 Lower Bounding the Search and Update Complexities of Forward-Private Searchable Encryption

Unfortunately, forward privacy has a non-negligible performance impact: when compared to non-forward private SSE schemes such as the one in [CJJ+13], which has constant client size and constant update complexity, every existing forward-private construction has an important overhead on the client storage (*e.g.* Σοφος [Bos16] and subsequent works [KKL+17, EKPE18] have a $\Theta(K)$ client storage), on the query complexity (*e.g.* $\tilde{\mathcal{O}}\left(\log^3 N\right)$ overhead for TWORAM [GMP16]), or on both ($\Theta\left(\log^2 N\right)$ update overhead and $\mathcal{O}(N^\alpha)$ client storage for SPS [SPS14]). This leads to the question of the minimal cost of forward-private SSE.

General ideas To study the computational complexity of such construction, we proceed similarly to the lower bound of Theorem 1. Unfortunately, because we have to consider both search and update queries, the analysis is slightly more involved. Indeed, we have to consider the case where the Update protocol does essentially nothing, just appending an encrypted entry to a log, and the Search protocol does all the work: during a search, the client downloads the log, decrypts the entries and sees which ones match the searched keyword. This sketched scheme has constant update complexity, constant client size (the client can stream the log, and not download it at once), but extremely high search complexity as the update work is piggy-backed to the Search protocol.

Also, as the forward-privacy security notion only makes sense against an adaptive adversary, we have to account for this, contrary to the previous bound where the considered adversary was non-adaptive, adding a bit of complexity in the proof.

Here, for simplicity, we suppose that the scheme only supports atomic insertions, *i.e.* additions of single-keyword/document pairs. Note that we can do that without loss of generality as more general updates can be expressed as a sequence of atomic updates, and as all the previously mentioned schemes are indeed inserting documents by processing them entry-by-entry.

The choice of the leakage function Although we want our lower bound to be as general as possible, we need some restrictions on the leakage to make the result easier to state and to prove. For example, we could have a leakage function which reveals everything about the database and the previously issued queries during a search, and nothing during an update. This will still be a forward-private construction, although not very secure in practice. Instead, we will consider a leakage function that only reveals information about the queried keyword and its matches during a search.

This can be formalized as follows: remember that, in Section 3.3, to define the search pattern, we used the query list Q, that is the list of all issued queries, marked as (i, w) for a search query on keyword w and

$(i, \mathsf{op}, w, \mathsf{ind})$ for an update on the (w, ind) entry, i being the timestamp of the query. From this notation, we can define the list $Q(w)$ of queries involving w as

$$Q(w) = \{q \in Q | q = (i, w) \text{ or } q = (i, \mathsf{op}, w, \mathsf{ind})\}.$$

From this, we can define a generic leakage function $\mathcal{L}_{FP}$, that achieves forward-privacy and only reveals information about the searched keywords, as:

$$\mathcal{L}_{FP}^{\mathsf{Stp}}(\mathsf{DB}) = \mathsf{DB},\ \mathcal{L}_{FP}^{\mathsf{Srch}}(w) = Q(w)$$
$$\text{and } \mathcal{L}_{FP}^{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{ind}) = \mathcal{L}'(\mathsf{op}, \mathsf{ind})$$

where $\mathcal{L}'$ is a stateless function. It easy to see that any of the forward-private schemes from [SPS14, GMP16, Bos16, KKL+17, EKPE18] have a leakage profile that leaks less than $\mathcal{L}_{FP}$ (in the sense of Definition 3).

We can now formally state our lower bound theorem on the computational complexity of forward-private SSE schemes:

Theorem 3. *Let Σ be an $\mathcal{L}$-unconditionally-secure forward private SSE scheme supporting insertion and deletion of entries in the database, with $\mathcal{L} \preceq \mathcal{L}_{FP}$. Suppose Σ has a client state (permanent memory) of size $|\sigma|$. Then, either the average computational complexity of the* Update *protocol (when summing the contributions of the client and of the server) is $\Omega\left(\frac{\log K}{\log|\sigma|\cdot\log\log K}\right)$ or the complexity of the* Search *protocol is $\Omega\left(\frac{t\log K}{\log|\sigma|\cdot\log\log K}\right)$ where t is the number of updates since the last search query.*

Proof. We proceed in a similar spirit to the proof of Theorem 1, with a player, able to hold at most b entries (atomic document/keyword pairs), making accesses to the encrypted database in order to answer a sequence of t update queries, and an observer who sees these accesses. As we supposed that the updates are atomic (*i.e.* are insertions or deletions of single keyword/document pairs), a sequence of t updates following by a search can be written as $[(\mathsf{op}_1, w_1, \mathsf{ind}_1), \ldots, (\mathsf{op}_t, w_t, \mathsf{ind}_t), w^*]$. Suppose that there exists $1 \leq k \leq t$ such that $w^* = w_k$, and that $\forall i \neq k, w_i \neq w_k$. The observer will win the game if he is able to distinguish the execution of this query sequence with the execution of a query sequence $[(\mathsf{op}'_1, w'_1, \mathsf{ind}'_1), \ldots, (\mathsf{op}'_t, w'_t, \mathsf{ind}'_t), w'_k]$ such that, for all $1 \leq i \leq t$, $\mathcal{L}'(\mathsf{op}_i, \mathsf{ind}_i) = \mathcal{L}'(\mathsf{op}'_i, \mathsf{ind}'_i)$, and $\forall i \neq k, w'_i \neq w'_k$. Note that if $(\mathsf{op}_i, \mathsf{ind}_i) = (\mathsf{op}'_i, \mathsf{ind}'_i)$ for $1 \leq i \leq t$, then the first condition clearly holds.

To answer these $t+1$ queries, the player will make a sequence of q *visible* accesses $V = (v_1, \ldots, v_q)$, observable by the adversary, and a sequence $(h_1, \ldots, h_q)$ of *hidden actions* h_i, which the observer cannot see. As before, the player can do one of the $b+2$ actions among taking an entry from the cell, placing an entry in the cell or doing nothing.

As in the case of the proof of Theorem 1, the action sequence can answer at most $(b+1)^q(b+2)^q$ update queries followed by a search query. On the other hand, the number of query sequences with the same $\mathcal{L}_{FP}$-leakage as $[(\mathsf{op}_1, w_1, \mathsf{ind}_1), \ldots, (\mathsf{op}_t, w_t, \mathsf{ind}_t), w_k]$, and such that $\forall i \neq k, w_i \neq w_k$, is at least $(K-1)^{t-1}$: also, w'_k must be fixed to w_k, we can generate such sequence by choosing and w'_i in $\mathsf{W} \setminus \{w_k\}$ for $\forall i \neq k$. Indeed, w_k cannot be used in the other update queries as it will be searched by the search query. As a consequence, we must have that

$$(b+1)^q(b+2)^q \geq (K-1)^{t-1} \Leftrightarrow q \geq \frac{(t-1)\log(K-1)}{\log(b+1)(b+2)},$$

which implies that we cannot have both that the amortized complexity of an update is less than be $\frac{\log K - 1}{\log(b+1)(b+2)}$, and that the complexity of a search is less than $\frac{(t-1)\log(K-1)}{\log b(b+2)}$. Also, we know that the computational complexity of a query has to be larger than $b - |\sigma|$ (every memory cell used during the execution of the protocol has to be used unless it is useless). In the end, we are in one of these two cases: either the amortized complexity of an update is $\Omega\left(\frac{\log K}{\log|\sigma|\cdot\log\log K}\right)$ (from Lemma 2), or the complexity of a search is $\Omega\left(\frac{(t-1)\log K}{\log|\sigma|\cdot\log\log K}\right)$. □

5.3 On the Tightness of the Lower Bound

It happens that the lower bound of Theorem 3 is essentially tight. Indeed, there exist three schemes — $\Sigma o\phi o\varsigma$ [Bos16], KKLPK [KKL+17], and EKPE [EKPE18] — that match the lower bound. For all these schemes, the client storage complexity is $\Theta(K)$ (at least one counter is stored for each keyword on the client side), and the updates have a constant time complexity.

Also, we can modify these schemes so as to reach the other side of the tradeoff: forward-private schemes with constant client storage. It is very important that the outsourcing of these schemes' keyword-to-counter map does not leak that the same counter was accessed when a search query on w is followed by an update query on w. A natural way to avoid that is to put the counter map in an ORAM, as every access would be hidden.

More precisely, we can use oblivious maps, as in Section 4.3, to outsource the storage of the counters to the server. The OMap of Wang *et al.* [WNL+14] will produce a scheme with update complexity $\mathcal{O}\left(\log^2 K\right)$ and $\mathcal{O}(\log K)$ client storage. The client storage, as in the Melinoe construction, be reduced to $\mathcal{O}(1)$ by storing the OMap state on the server, fetching it and re-encrypting it for every access.

Also, one can construct a scheme, described in Algorithm 2, with constant update complexity, constant client size and $\mathcal{O}\left(\log^2 K + n_w\right)$ amortized search complexity. It uses an underlying forward-private scheme Σ with a constant-size client state and $\mathcal{O}\left(\log^2 K\right)$ search and update complexity (such as the ones described just before). The search complexity of the first two steps is $\mathcal{O}(t \log K)$, as the client must first decrypt the entries in L, each of size $\mathcal{O}(\log K)$ at least (there is at least one entry per keyword, and there are K keywords). The total complexity of the Search protocol is $\mathcal{O}\left(t \log^2 K + n_w\right)$ as the client makes t insertions, of complexity $\mathcal{O}\left(\log^2 K\right)$ and one search of complexity $\mathcal{O}\left(\log^2 K + n_w\right)$. This scheme is hence optimal as the complexity of the search algorithm has to be $\Omega(n_w)$ anyways, which, combined with our lower bound, means that the search complexity is at least $\Omega\left(\log^2 K + n_w\right)$.

Algorithm 2 Description of a forward private scheme with constant update complexity and constant client size. Σ is a forward-private scheme with $\mathcal{O}\left(n_w + \log^2 K\right)$ search and $\mathcal{O}\left(\log^2 K\right)$ update complexity.

$\underline{\mathsf{Setup}(\mathsf{DB})}$

1: $K_E \xleftarrow{\$} \{0,1\}^\lambda$
2: $L \leftarrow$ empty list
3: $(\mathsf{EDB}_\Sigma, K_\Sigma, \sigma_\Sigma) \leftarrow \Sigma.\mathsf{Setup}(\mathsf{DB})$
4: **return** $((\mathsf{EDB}_\Sigma, L), (K_\Sigma, K_E), \sigma_\Sigma)$

$\underline{\mathsf{Update}((K_\Sigma, K_E), \sigma_\Sigma, w, \mathsf{ind}; \mathsf{EDB})}$

Client:
1: $e \leftarrow \mathsf{Enc}(K_E, (w, \mathsf{ind}))$
2: Send e to the server.
Server:
3: Appends e to L.

$\underline{\mathsf{Search}((K_\Sigma, K_E), w, \sigma; \mathsf{EDB})}$

1: For each $e \in L$, the client downloads and decrypts e, getting the pair (w', ind).
2: The client and server run $\Sigma.\mathsf{Update}(K_\Sigma, \sigma, w', \mathsf{ind})$.
3: The server resets L to an empty list.
4: Run $\Sigma.\mathsf{Search}(w)$, and return the result.

Yet, it is unclear if we can construct schemes whose efficiency is on the trade-off curve, between the extrema, *e.g.* a scheme with $\mathcal{O}(\sqrt{K})$ client storage and $\mathcal{O}(1)$ update complexity. We think this is an interesting question for future work.

Also, it would be interesting to see if we can transform the $\log^2 K$ term into a $\log K$ term in both of the previous tight schemes with constant-size client state. The squaring comes from the use of OMaps/tree-based ORAM to outsource the client's state, and it might be asymptotically overcome using more recent ORAM

constructions, such as OptORAMa [AKL+18].

Variants of the $\mathcal{L}_{FP}$ leakage function One could wonder what is the impact of slightly changing the leakage function. First, reducing the leakage would essentially mean that the scheme hides the 'history' of the searched keyword, *i.e.* when the entries matching the keyword have been inserted. A scheme with such leakage would be very close to ORAM, and the only existing construction achieving this leakage is actually ORAM-based [GMP16]. As such, we believe it is unlikely to construct such a scheme that would be significantly better than ORAM, as shown in Section 4.

On the other side, increasing the leakage would mean leaking information about keywords different from the one being searched. In some sense, this would break the intention behind the forward privacy requirement: leaking information only about the query itself (in the case of an update, the nature of the operation and its size). Hence, we did not really study this case, although we can easily see that, in this setting, we can achieve a better complexity, for example by revealing *all* the updates since the last search during a search query, and not just the updates related to the searched keyword.

6 Conclusion

We studied two lower bounds on searchable encryption schemes, targeting static schemes hiding the search pattern, and forward-private dynamic schemes. We showed that these lower bounds are tight in the last case, and allow different efficiency tradeoffs, *e.g.* between the size of the client's state and the queries' computational complexity.

Although, as explained in Section 4.2, the chosen models for the lower bounds might have their limits, we think that our results provide a better understanding of what is possible, and what is not, to do with searchable encryption schemes. Improving the strength of results, *e.g.* by using a more general model for the database representation, is a very interesting question left for future work.

Acknowledgments

The authors thanks Tarik Moataz for spotting an issue with a previous version of Theorem 3, David Pointcheval for the help and all the useful comments, the PoPETs shepherds, Wei-Kai Lin and Peter Schwabe for their help improving the paper, and the anonymous reviewers. They also thank the Agence Nationale de Recherche (ANR), which partially funded this research by means of the SafeTLS project.

References

[AKL+18] Asharov, G., Komargodski, I., Lin, W.K., Nayak, K., Peserico, E., and Shi, E. OptORAMa: Optimal oblivious ram. Cryptology ePrint Archive, Report 2018/892 (2018). https://eprint.iacr.org/2018/892.

[ANSS16] Asharov, G., Naor, M., Segev, G., and Shahaf, I. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In: D. Wichs and Y. Mansour (eds.), 48th ACM STOC, pp. 1101–1114. ACM Press (Jun. 2016).

[ASS18] Asharov, G., Segev, G., and Shahaf, I. Tight tradeoffs in searchable symmetric encryption. In: H. Shacham and A. Boldyreva (eds.), CRYPTO 2018, Part I, *LNCS*, vol. 10991, pp. 407–436. Springer, Heidelberg (Aug. 2018).

[BBO07] Bellare, M., Boldyreva, A., and O'Neill, A. Deterministic and efficiently searchable encryption. In: A. Menezes (ed.), CRYPTO 2007, *LNCS*, vol. 4622, pp. 535–552. Springer, Heidelberg (Aug. 2007).

[BCLO09] Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. Order-preserving symmetric encryption. In: A. Joux (ed.), EUROCRYPT 2009, *LNCS*, vol. 5479, pp. 224–241. Springer, Heidelberg (Apr. 2009).

[BDOP04] Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G. Public key encryption with keyword search. In: C. Cachin and J. Camenisch (eds.), EUROCRYPT 2004, *LNCS*, vol. 3027, pp. 506–522. Springer, Heidelberg (May 2004).

[BFP16] Bost, R., Fouque, P.A., and Pointcheval, D. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. Cryptology ePrint Archive, Report 2016/062 (2016). http://eprint.iacr.org/2016/062.

[BKOS07] Boneh, D., Kushilevitz, E., Ostrovsky, R., and Skeith III, W.E. Public key encryption that allows PIR queries. In: A. Menezes (ed.), CRYPTO 2007, *LNCS*, vol. 4622, pp. 50–67. Springer, Heidelberg (Aug. 2007).

[BMO17] Bost, R., Minaud, B., and Ohrimenko, O. Forward and backward private searchable encryption from constrained cryptographic primitives. In: B.M. Thuraisingham, D. Evans, T. Malkin, and D. Xu (eds.), ACM CCS 2017, pp. 1465–1482. ACM Press (Oct. / Nov. 2017).

[BN16] Boyle, E. and Naor, M. Is there an oblivious RAM lower bound? In: M. Sudan (ed.), ITCS 2016, pp. 357–368. ACM (Jan. 2016).

[Bos16] Bost, R. $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In: E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, and S. Halevi (eds.), ACM CCS 2016, pp. 1143–1154. ACM Press (Oct. 2016).

[Bos18] Bost, R. Searchable Encryption – New Constructions of Encrypted Databases. Ph.D. thesis, Université de Rennes 1 (January 2018). URL https://www.theses.fr/2018REN1S001.

[BR06] Bellare, M. and Rogaway, P. The security of triple encryption and a framework for code-based game-playing proofs. In: S. Vaudenay (ed.), EUROCRYPT 2006, *LNCS*, vol. 4004, pp. 409–426. Springer, Heidelberg (May / Jun. 2006).

[CGKO06] Curtmola, R., Garay, J.A., Kamara, S., and Ostrovsky, R. Searchable symmetric encryption: improved definitions and efficient constructions. In: A. Juels, R.N. Wright, and S. De Capitani di Vimercati (eds.), ACM CCS 2006, pp. 79–88. ACM Press (Oct. / Nov. 2006).

[CGPR15] Cash, D., Grubbs, P., Perry, J., and Ristenpart, T. Leakage-abuse attacks against searchable encryption. In: I. Ray, N. Li, and C. Kruegel (eds.), ACM CCS 2015, pp. 668–679. ACM Press (Oct. 2015).

[CJJ+13] Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., and Steiner, M. Highly-scalable searchable symmetric encryption with support for Boolean queries. In: R. Canetti and J.A. Garay (eds.), CRYPTO 2013, Part I, *LNCS*, vol. 8042, pp. 353–373. Springer, Heidelberg (Aug. 2013).

[CK10] Chase, M. and Kamara, S. Structured encryption and controlled disclosure. In: M. Abe (ed.), ASIACRYPT 2010, *LNCS*, vol. 6477, pp. 577–594. Springer, Heidelberg (Dec. 2010).

[CT14] Cash, D. and Tessaro, S. The locality of searchable symmetric encryption. In: P.Q. Nguyen and E. Oswald (eds.), EUROCRYPT 2014, *LNCS*, vol. 8441, pp. 351–368. Springer, Heidelberg (May 2014).

[DDF+16] Devadas, S., Dijk, M., Fletcher, C.W., Ren, L., Shi, E., and Wichs, D. Onion ORAM: A constant bandwidth blowup oblivious RAM. In: E. Kushilevitz and T. Malkin (eds.), TCC 2016-A, Part II, *LNCS*, vol. 9563, pp. 145–174. Springer, Heidelberg (Jan. 2016).

[DPP18] Demertzis, I., Papadopoulos, D., and Papamanthou, C. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In: H. Shacham and A. Boldyreva (eds.), CRYPTO 2018, Part I, *LNCS*, vol. 10991, pp. 371–406. Springer, Heidelberg (Aug. 2018).

[EKPE18] Etemad, M., Küpçü, A., Papamanthou, C., and Evans, D. Efficient dynamic searchable encryption with forward privacy. PoPETs, vol. 2018(1):(2018), pp. 5–20. URL https://doi.org/10.1515/popets-2018-0002.

[Gen09] Gentry, C. A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009). crypto.stanford.edu/craig.

[GMP16] Garg, S., Mohassel, P., and Papamanthou, C. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In: M. Robshaw and J. Katz (eds.), CRYPTO 2016, Part III, *LNCS*, vol. 9816, pp. 563–592. Springer, Heidelberg (Aug. 2016).

[GO96] Goldreich, O. and Ostrovsky, R. Software protection and simulation on oblivious RAMs. Journal of the ACM, vol. 43(3):(1996), pp. 431–473.

[GSB+17] Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., and Ristenpart, T. Leakage-abuse attacks against order-revealing encryption. In: 2017 IEEE Symposium on Security and Privacy, pp. 655–672. IEEE Computer Society Press (May 2017).

[JJK+13] Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., and Steiner, M. Outsourced symmetric private information retrieval. In: A.R. Sadeghi, V.D. Gligor, and M. Yung (eds.), ACM CCS 2013, pp. 875–888. ACM Press (Nov. 2013).

[KKL+17] Kim, K.S., Kim, M., Lee, D., Park, J.H., and Kim, W.H. Forward secure dynamic searchable symmetric encryption with efficient updates. In: B.M. Thuraisingham, D. Evans, T. Malkin, and D. Xu (eds.), ACM CCS 2017, pp. 1449–1463. ACM Press (Oct. / Nov. 2017).

[KM17] Kamara, S. and Moataz, T. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In: J. Coron and J.B. Nielsen (eds.), EUROCRYPT 2017, Part III, *LNCS*, vol. 10212, pp. 94–124. Springer, Heidelberg (Apr. / May 2017).

[KMO18] Kamara, S., Moataz, T., and Ohrimenko, O. Structured encryption and leakage suppression. In: H. Shacham and A. Boldyreva (eds.), CRYPTO 2018, Part I, *LNCS*, vol. 10991, pp. 339–370. Springer, Heidelberg (Aug. 2018).

[KO12] Kurosawa, K. and Ohtaki, Y. UC-secure searchable symmetric encryption. In: A.D. Keromytis (ed.), FC 2012, *LNCS*, vol. 7397, pp. 285–298. Springer, Heidelberg (Feb. / Mar. 2012).

[KO13] Kurosawa, K. and Ohtaki, Y. How to update documents verifiably in searchable symmetric encryption. In: M. Abdalla, C. Nita-Rotaru, and R. Dahab (eds.), CANS 13, *LNCS*, vol. 8257, pp. 309–328. Springer, Heidelberg (Nov. 2013).

[KP13] Kamara, S. and Papamanthou, C. Parallel and dynamic searchable symmetric encryption. In: A.R. Sadeghi (ed.), FC 2013, *LNCS*, vol. 7859, pp. 258–274. Springer, Heidelberg (Apr. 2013).

[KPR12] Kamara, S., Papamanthou, C., and Roeder, T. Dynamic searchable symmetric encryption. In: T. Yu, G. Danezis, and V.D. Gligor (eds.), ACM CCS 2012, pp. 965–976. ACM Press (Oct. 2012).

[LN18] Larsen, K.G. and Nielsen, J.B. Yes, there is an oblivious RAM lower bound! In: H. Shacham and A. Boldyreva (eds.), CRYPTO 2018, Part II, *LNCS*, vol. 10992, pp. 523–542. Springer, Heidelberg (Aug. 2018).

[LO13] Lu, S. and Ostrovsky, R. How to garble RAM programs. In: T. Johansson and P.Q. Nguyen (eds.), EUROCRYPT 2013, *LNCS*, vol. 7881, pp. 719–734. Springer, Heidelberg (May 2013).

[Nav15] Naveed, M. The fallacy of composition of oblivious RAM and searchable encryption. Cryptology ePrint Archive, Report 2015/668 (2015). http://eprint.iacr.org/2015/668.

[NKW15] Naveed, M., Kamara, S., and Wright, C.V. Inference attacks on property-preserving encrypted databases. In: I. Ray, N. Li, and C. Kruegel (eds.), ACM CCS 2015, pp. 644–655. ACM Press (Oct. 2015).

[PKV+14] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A.D., and Bellovin, S. Blind seer: A scalable private DBMS. In: 2014 IEEE Symposium on Security and Privacy, pp. 359–374. IEEE Computer Society Press (May 2014).

[PLZ13] Popa, R.A., Li, F.H., and Zeldovich, N. An ideal-security protocol for order-preserving encoding. In: 2013 IEEE Symposium on Security and Privacy, pp. 463–477. IEEE Computer Society Press (May 2013).

[PRZB11] Popa, R.A., Redfield, C., Zeldovich, N., and Balakrishnan, H. Cryptdb: protecting confidentiality with encrypted query processing. In: ACM SOSP 11, pp. 85–100. ACM (2011).

[PZ13] Popa, R.A. and Zeldovich, N. Multi-key searchable encryption. Cryptology ePrint Archive, Report 2013/508 (2013). http://eprint.iacr.org/2013/508.

[SPS14] Stefanov, E., Papamanthou, C., and Shi, E. Practical dynamic searchable encryption with small leakage. In: NDSS 2014. The Internet Society (Feb. 2014).

[SWP00] Song, D.X., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy, pp. 44–55. IEEE Computer Society Press (May 2000).

[WNL+14] Wang, X.S., Nayak, K., Liu, C., Chan, T.H.H., Shi, E., Stefanov, E., and Huang, Y. Oblivious data structures. In: G.J. Ahn, M. Yung, and N. Li (eds.), ACM CCS 2014, pp. 215–226. ACM Press (Nov. 2014).

[WW18] Weiss, M. and Wichs, D. Is there an oblivious RAM lower bound for online reads? Cryptology ePrint Archive, Report 2018/619 (2018). https://eprint.iacr.org/2018/619.

[ZKP16] Zhang, Y., Katz, J., and Papamanthou, C. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: T. Holz and S. Savage (eds.), USENIX Security 2016, pp. 707–720. USENIX Association (Aug. 2016).

A Oblivious Maps

A.1 Definition

As we explained in Section 4.3, an oblivious map is composed of one algorithm and two protocols between a client and a server:

- $\mathsf{Setup}(T)$ is a probabilistic algorithm that takes as input the initial content of the OMap T. It outputs a couple $(\mathsf{OMapState}, \mathsf{OMap})$, OMap is the oblivious data structure stored on the server, and $\mathsf{OMapState}$ is the client's state.

$\underline{\mathsf{Init}(T_0, T_1)}$
if $|T_0| \neq |T_0|$
Abort game
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{OMapState}, \mathsf{OMap}) \xleftarrow{\$} \mathsf{Setup}(T_b)$
return OMap

$\underline{\mathsf{Final}(b')}$
return $b = b'$

$\underline{\mathsf{Access}(q_0, q_1)}$
if q_b is a Read access **then**
Parse q_b as k
$(v, \mathsf{OMapState}, \tau; \mathsf{OMap}) \xleftarrow{\$} \mathsf{Read}_C(k, \mathsf{OMapState}; \mathsf{OMap}) \leftrightarrow A$
else
Parse q_b as (k, v)
$(\mathsf{OMapState}, \tau; \mathsf{OMap}) \xleftarrow{\$} \mathsf{Write}_C(k, v, \mathsf{OMapState}; \mathsf{OMap}) \leftrightarrow A$
end if
return τ

Figure 2 – Security game OMapInd capturing malicious adversaries. The notation $\leftrightarrow A$ represents interactions with the adversary.

- $\mathsf{Read}(k, \mathsf{OMapState}; \mathsf{OMap})$ is a protocol between the client with input the (non cryptographic) key k (the query), and the client's internal state OMapState; and the server with input the OMap data structure OMap.

  After completing the Search protocol, the client outputs a value v and a new state $\mathsf{OMapState}'$. The value v can be $\top$ to signify that k does not match any value in the map. Both v and $\mathsf{OMapState}'$ can take the special value $\perp$ to signify an error or a failure in the execution of the protocol. The server possibly outputs an updated structure OMap'.

- $\mathsf{Write}(k, v, \mathsf{OMapState}; \mathsf{OMap})$ is a protocol between the client with input the key k, the value v, and the client's internal state OMapState; and the server with input the OMap data structure OMap. At the end of the execution of the protocol, the client outputs a new state $\mathsf{OMapState}'$, which can take the special value $\perp$, and the server outputs a new data structure OMap'.

In this paper, we assumed that the OMap instantiations we were using are correct, namely that $\mathsf{Read}(k)$ return the latest value v for key k, *i.e.* the value v of the latest call to $\mathsf{Write}(k, v)$ or $T[k]$ where T is the initial map used during setup.

A.2 Security

The security requirement of oblivious maps state that two sequences of access (read or writes) are indistinguishable. It can be formalized using the security game of Figure 2.

Definition 5. *Let Θ be an OMap scheme. For an adversary A, the advantage $\mathsf{Adv}^{\text{OVM-ind}}_{\Theta,A}(\lambda)$ of A in the indistinguishability-based confidentiality game is defined as*

$$\mathsf{Adv}^{\text{OVM-ind}}_{\Theta,A}(\lambda) = \left| \frac{1}{2} - \mathbb{P}[\text{OMapInd}^A_\Theta(1^\lambda) = 1] \right|.$$

An OMap scheme Θ is adaptively-indistinguishability secure *if for any polynomial-time adversary A, $\mathsf{Adv}^{\text{OVM-ind}}_{\Theta,A}(\lambda)$ is negligible in λ.*

As in the SSE case, this definition covers both passive and active adversaries (depending on whether A respects or not the protocols).

B Additional Proofs

We are giving here the formal proofs of Lemma 2.

Proof. This lemma can be shown using simple analysis. We start by a simple variable change:

$$\begin{aligned}&\max_{x\in[D+1,+\infty)}\left\{x-D,\frac{C}{\log(x+1)(x+2)}\right\}\\&\quad=\max_{x\in[1,+\infty)}\left\{x,\frac{C}{\log(x+D+1)(x+D+2)}\right\}\end{aligned}$$

Then, notice that, that, as log is an increasing function,

$$\begin{aligned}\frac{C}{\log(x+D+1)(x+D+2)}&\geq\frac{C}{2\log(x+D+2)}\\&=\frac{C}{2(\log(x/D'+1)+\log(D'))}\end{aligned}$$

where $D'=D+2$. Also, by denoting $C'=\frac{C}{2\log D'}$, we have that

$$\begin{aligned}\frac{C}{\log(x+D+1)(x+D+2)}&\geq\frac{C'}{\log_{D'}(x/D'+1)+1}\\&\geq\frac{C'}{\log_{D'}(x+1)+1}\end{aligned}$$

again, because log is an increasing function. As a consequence

$$\begin{aligned}&\max_{x\in[D+1,+\infty)}\left\{x-D,\frac{C}{\log x(x+2)}\right\}\\&\quad\geq\max_{x\in[1,+\infty)}\left\{x,\frac{C'}{\log_{D'}(x+1)+1}\right\}\end{aligned}$$

Let $g(x)=\max\left\{x,\frac{C'}{\log_{D'}(x+1)+1}\right\}$. Lower bounding f on $[D+1,+\infty)$ is equivalent to lower bounding g on $[1,+\infty)$. Let us separate two cases, whether $x\geq D'-1$ or $x<D'-1$.

If $x<D'-1$, then $\log_{D'}(x+1)+1\leq 2$ and

$$g(x)\geq\max\{x,C'/2\}\geq\frac{C}{4\log D'}.$$

If $x\geq D'-1$, then $\log_{D'}(x+1)+1\leq 2\log_{D'}(x+1)$ and

$$g(x)\geq\max\left\{x,\frac{C'/2}{\log_{D'}(x+1)}\right\}.$$

Also, as $x\mapsto x$ is an increasing function and $x\mapsto\frac{C'}{2\log_{D'}(x+1)}$ is decreasing, the minimum is reached when $x=\frac{C'}{2\log_{D'}(x+1)}$. Finding this minimum is equivalent to finding x^* such that $h(x^*)=0$ where

$$h(x)=x-\frac{C'}{2\log_{D'}(x+1)}=x-\frac{C}{4\log(x+1)}$$

as $C'=\frac{C}{2\log D'}$. In particular, $f(x)\geq x^*$. Note that we are actually only looking for a lower bound of x^*, not its exact value. Hence, as h is a continuous and strictly increasing function on $[1,+\infty)$, x^* is the only value on which h annihilates and $h(x)\leq 0\Leftrightarrow x\leq x^*$. Also, we have that

$$(x+1)-\frac{C}{4\log(x+1)}\leq 0\Rightarrow x\leq x^*,$$

Let $C'' = C/4$ and suppose that $\log C'' > \log\log C'' \geq 0$. Then, we directly have that

$$\frac{1}{\log C''} - \frac{1}{\log \frac{C''}{\log C''}} = \frac{1}{\log C''} - \frac{1}{\log C'' - \log\log C''} \leq 0,$$

and for $x_0 = \frac{C''}{\log C''} - 1$, we have that $(x_0 + 1) - \frac{C}{4\log(x_0+1)} \leq 0$, and then that $x_0 \leq x^*$.

To conclude, we just have to check that $\log C'' > \log\log C'' \geq 0$. The last inequality is verified for $C \geq 8$. For the first inequality, we have to study $t : x \mapsto x - \log x$ on the interval $[1, +\infty)$. Its derivative $x \mapsto 1 - \frac{1}{x \cdot \ln 2}$ (where ln is the natural logarithm) is negative on $[1, \frac{1}{\ln 2}]$ and positive on $[\frac{1}{\ln 2}, +\infty[$. Thus, t reaches its minimum for $x = \frac{1}{\ln 2} \approx 1.443$, and

$$t(x) \geq t\left(\frac{1}{\ln 2}\right) \approx 0.914 > 0.$$

As a consequence, and because we supposed that $C > 8$, $\log C'' > 0$, and $\log C'' > \log\log C''$, we have that $x_0 \geq \frac{C}{4(\log C)-2} - 1 \geq \frac{C}{4\log C} - 1$.

We can conclude the proof of this lemma by stating that

$$\begin{aligned}
&\max_{x \in [D+1, +\infty)} \left\{ x - D, \frac{C}{\log x(x+2)} \right\} \\
&\quad \geq \min\left\{ \frac{C}{4\log D'}, \frac{C}{4\log C} - 1 \right\} \\
&\quad \geq \frac{C}{4\log(D+2) \cdot \log C} - 1
\end{aligned}$$

□

Note that we can show a variant of Lemma 2, that allows for better lower bounds.

Lemma 4. *For all $a \in \mathbb{N}^*$, if $C > 4 \cdot 2^{2^{\cdot^{\cdot^{\cdot^{2^0}}}}}$ (a times), then*

$$\max\left\{ x - D, \frac{C}{\log(x+1)(x+2)} \right\} > \frac{C}{4\log(D+2) \cdot \log^{(a)} C}.$$

where $\log^{(1)} x = \log x$ and $\log^{(a)} x = \log\log^{(a-1)} x$ for $a > 1$.

Hence, in the proof of Theorems 1 (resp. Theorem 3), we can apply Lemma 4 instead of Lemma 2 with a well chosen value $a \in \mathbb{N}^*$ such that $1 \leq \log^{(a)}\left(\overline{N}(H_{i-1}, w_i) \atop n_{w_i}\right) < 2$ (resp. $1 \leq \log^{(a)} K < 2$), and end up with a larger lower bound, namely $\Omega\left(\frac{\log\left(\overline{N}(H,w) \atop n_w\right)}{\log|\sigma|}\right)$ (resp. $\Omega\left(\frac{\log K}{\log|\sigma|}\right)$), than stated originally.

Algorithm 3 Description of the Melinoe variant with constant size state.

Setup(DB)

$K_E \xleftarrow{\$} \{0,1\}^\lambda$
$\mathbf{W} \leftarrow$ empty maps
$\mathsf{OMapState}, \mathsf{EncOMapState}, \mathsf{OMap}, \leftarrow$ empty array
for all n such that $\exists w, |\mathsf{DB}(w)| = n$ **do**
  $\mathsf{Tmp} \leftarrow$ empty map
  for all w such that $|\mathsf{DB}(w)| = n$ **do**
    $\mathsf{Tmp}[w] \leftarrow \mathsf{DB}(w)$
    $\mathbf{W}[w] \leftarrow n$
  end for
  $(\mathsf{OMapState}[n], \mathsf{OMap}[n]) \leftarrow \Theta.\mathsf{Setup}(\mathsf{Tmp})$
  $\mathsf{EncOMapState}[n] \leftarrow \mathsf{Enc}(K_E, \mathsf{OMapState}[n])$
end for
$(\mathsf{OMapState\text{-}}\mathbf{W}, \mathsf{OMap\text{-}}\mathbf{W}) \leftarrow \Theta.\mathsf{Setup}(\mathbf{W})$
$\mathsf{EncWState} \leftarrow \mathsf{Enc}(K_E, \mathsf{OMapState\text{-}}\mathbf{W})$
return $((\mathsf{OMap}, \mathsf{EncOMapState}, \mathsf{OMap\text{-}}\mathbf{W}, \mathsf{EncWState}), K_E, \varnothing)$

Search(K_Σ, w, σ; EDB)

Client:
Fetch EncWState from the server.
$\mathsf{OMapState\text{-}}\mathbf{W} \leftarrow \mathsf{Dec}(K_E, \mathsf{EncWState})$
Run $n \leftarrow \Theta.\mathsf{Read}(w, \mathsf{OMapState\text{-}}\mathbf{W}; \mathsf{OMap\text{-}}\mathbf{W})$ with the server. ▷ $n \leftarrow \mathbf{W}[w]$*: run the oblivious map data structure protocol between the client and the server.* $\mathsf{OMapState\text{-}}\mathbf{W}$ *gets updated in the process.*
Fetch $\mathsf{EncOMapState}[n]$ from the server.
$\mathsf{OMapState}[n] \leftarrow \mathsf{Dec}(K_E, \mathsf{EncOMapState}[n])$
Run $w \leftarrow \Theta.\mathsf{Read}(w, \mathsf{OMapState}[n]; \mathsf{OMap}[n])$ with the server. ▷ *Access w in the n-th OMap: run the oblivious map data structure protocol between the client and the server.* $\mathsf{OMapState}[n]$ *gets updated in the process.*
Parse the value as $(\mathsf{ind}_1, \ldots, \mathsf{ind}_n)$.
$\mathsf{EncWState} \leftarrow \mathsf{Enc}(K_E, \mathsf{OMapState\text{-}}\mathbf{W})$
$\mathsf{EncOMapState}[n] \leftarrow \mathsf{Enc}(K_E, \mathsf{OMapState}[n])$
Send EncWState and $\mathsf{EncOMapState}[n]$ to the server.
return $(\mathsf{ind}_1, \ldots, \mathsf{ind}_n)$.