Thwarting Leakage Abuse Attacks against Searchable Encryption A Formal Approach and Applications to Database Padding

Raphael Bost* Pierre-Alain Fouque†

Abstract

After the development of practical searchable encryption constructions, allowing for secure searches over an encrypted dataset outsourced to an untrusted server, at the expense of leaking some information to the server, many new attacks have recently been developed, targeting this leakage in order to break the confidentiality of the dataset or of the queries, through *leakage abuse attacks*.

These works helped to understand the importance of considering leakage when analyzing the security of searchable encryption schemes, but did not explain why these attacks were so powerful despite the existence of rigorous security definitions and proofs, or how they could be efficiently and provably mitigated.

This work addresses these questions by first proposing an analysis of existing leakage abuse attacks and a way to capture them in new security definitions. These new definitions also help us to devise a way to thwart these attacks and we apply it to the padding of datasets, in order to hide the number of queries' results, and to provide provable security of some schemes with specific leakage profile against some common classes of leakage abuse attacks.

Finally, we give experimental evidence that our countermeasures can be implemented efficiently, and easily applied to existing searchable encryption schemes.

1 Introduction

Using regular encryption to protect data outsourced to an untrusted provider provides best possible security at the expense of usability and functionality: data accesses are encumbered by the inability of the server to manipulate encrypted information as it would with plaintext database.

To get around the issue of retrieving text in an encrypted database efficiently, *searchable encryption (SE)* schemes were developed, starting with the seminal work of Song *et al.* [SWP00]. Indeed, an SE scheme encrypts documents in a structured way that allows the provider to answer to search queries from the data owner without having to decrypt the database. Yet, practical solutions, *i.e.* solutions not using generic tools like multi-party computation or fully homomorphic encryption, always leak some information about the database and/or the queries. This is the case for every recent construction [CJJ+13, PKV+14, CJJ+14, SPS14, KPR12, KP13, GMP16, Bos16]. Even ORAM-based constructions [GMP16] leak the number of results of a search query since the output size is linear in this number.

The simulation-based security model of searchable encryption, as defined by Curtmola *et al.* [CGKO06], takes into account this leakage: in the security proofs, one shows that the server cannot learn more than the leakage. Yet this formalism does not tell us if the leakage can be leveraged into decrypting the database or the queries. One needs to understand what is the acceptable level of leakage for SE schemes.

*Direction Générale de l'Armement - Maîtrise de l'Information & Université de Rennes 1, France. 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

Leakage abuse attacks. One step towards the understanding of leakage is the recent development of *leakage abuse attacks* that aim at decrypting queries and/or the database using only the leaking information [IKK12, CGPR15, KKNO16]. None of these attacks invalidate in any way the security proofs of the schemes they target: they all fall outside of the security model. Indeed, we will see that none of the possible adversarial knowledge described in [CGPR15] is captured by the security definition.

Current state-of-the-art SE schemes leak the repetition of queries, the number of documents matching each queries and, usually, for every pair of queries, the documents in common (denoted as L1 leakage in [CGPR15]). Islam *et al.* [IKK12] showed that an attacker can recover queries using this co-occurrence information, provided that she has some background knowledge on the database. Cash *et al.* [CGPR15] used the fact that the counts of queries' results can be unique in order to immediately decrypt some queries, and then leverage this knowledge to decrypt the other queries using keywords co-occurrences.

This attack is actually very powerful, as, on an email database (Enron), if one considers the 500 most common words as keywords, 63% of them have a unique result count and for the 2000 most common words, 24% of counts are unique. A large part of the queries can be immediately recovered from the knowledge of the database, and this will be of a huge help when bootstrapping from the co-occurrences information.

Recently, Kellaris *et al* [KKNO16] even showed that it was possible to recover the encrypted database from schemes leaking only the results length of range queries.

While in these previous attacks, the adversary is passive and the database is static, Zhang *et al.* [ZKP16] presented very efficient *file injection attacks* against dynamic searchable encryption, which can be only partly mitigated [SPS14, Bos16].

Our Contributions. In this paper, our goals are threefold: we want (*i*) to understand the discrepancy between the security proofs and the leakage abuse attacks, (*ii*) to give new security definitions capturing the attacks, (*iii*) to provide methods to assess the security of existing constructions, and use them to construct schemes provably secure agains such attacks. In particular, we will apply this approach to state-of-the-art schemes in order to protect them against adversary with prior knowledge of the encrypted database (*i.e.* the setting of the count attack [CGPR15]) .

For the first goal, we clearly identify why the attacks fall outside the security model defined for SSE, and thus why they are so efficient at breaking the security of SSE schemes. More formally, in the security definition, the adversary has to output two different histories, database and queries set, that have the same leakage, to be challenged upon. For the scheme to be secure, the executions should be indistinguishable.

Yet, we will see that, with the prior knowledge of the adversary – in the setting of these leakage abuse attacks – it might be impossible to find pairs of histories with the same leakage, and that both correspond to the adversary's information about the database or about the queries. In the end, the security definitions turn out to be void, because the database or the queries are uniquely defined given some leakage and adversarial knowledge.

In order to propose a new security game and definition capturing the leakage abuse attacks, we define a way to model the adversarial knowledge, and use this formalism to ensure that, for the definition to be satisfied, we have the non-uniqueness of histories given some leakage and external knowledge. This will achieve our second goal. Moreover, we will show that this non-uniqueness condition 'revives' the previous security definitions: to show the security of schemes against (some classes of) leakage abuse attack, we just have to prove the security using the previous security, and to show that the leakage function of the scheme satisfies a simple condition.

Unfortunately, this approach is not 'constructive': it does not really help us in proving the security of existing schemes against leakage abuse, nor does it hint at the tweaks necessary to fix broken schemes.

As a consequence, we present a solution based on the clustering of queries according to the leakage they induce. This clustering will allow us to precisely argue about the (in)security of some leakage in presence of some adversarial knowledge.

Also, we apply this solution to schemes leaking the result length of search queries (and possible the search pattern), and show how to make them provable secure even when the adversary fully knows the database.

Namely, we describe a database padding algorithm that will ensure that there is no query with a unique result length, while achieving a minimal overhead.

We implemented this padding algorithm, and evaluated the overhead on real databases. We show that this algorithm is quite efficient, and can be easily added to existing SE schemes, at a low computational cost. We also compared our padding methodology, with a simpler one, against the count attack of Cash *et al.* [CGPR15], and we show that, although being provably secure when the scheme only leaks the result length, this padding is not enough when co-occurrence information also leak. Finally, we argue that co-occurrence-based attacks are very powerful when the adversary knows the dataset, and that it will be very costly to protect schemes against these attacks using padding.

2 Searchable Encryption and its Security

In this section, we quickly recall what is searchable encryption, and how security is defined for SE schemes. We will show how the security definition almost directly implies leakage abuse attacks, and start devising a way to thwart these attacks.

A *database* $\mathsf{DB} = (D_1, \ldots, D_d)$ is a collection of d documents. Each document is a string of length $|D_i|$ from a character set, and can be represented as a vector W_i of keywords using an extraction procedure (which is public). We note W the set of all keywords, and we also suppose it is known to the adversary.

An SE scheme consists in a setup algorithm, a search protocol and, for dynamic schemes, an additional update protocol. The setup algorithm takes as input a database DB and outputs a secret key K and an encrypted database EDB; the client keeps K and sends EDB to the server. The Search protocol takes as client input the key K and query q and as server input the encrypted database EDB, and outputs to the client a list of documents matching the query. In this work, we will mainly focus on single-keyword queries, but this formalism covers more complex queries (*e.g.* range queries). Finally, the update protocol takes the key K and update information in from the client, the encrypted database EDB from the server, and updates EDB according to in.

Both the client and server can be stateful, but we omitted their respective inner state in the above definitions.

2.1 Security Definitions

The common security definition for searchable encryption comes from the seminal work of Curtmola *et al.* [CGKO06], and is based on an ideal-world-vs-real-world security game. It is parametrized by a *leakage function* $\mathcal{L} = (\mathcal{L}^{\mathsf{Stp}}, \mathcal{L}^{\mathsf{Srch}}, \mathcal{L}^{\mathsf{Updt}})$ describing what the protocol leaks to the adversary, acting as the server, and formalized as a stateful algorithm. The definition ensures that the scheme does not reveal any information beyond the ones that can be inferred from the leakage function.

Namely, in the real game, an adversarially chosen database DB is encrypted using the setup algorithm, and the encrypted database is given back to the adversary $\mathcal{A}$. Then she repeatedly performs search (and update queries if the scheme is dynamic) using input query q and receives the transcripts generated by running the search (and possibly update) protocol.

In the ideal game, the encrypted database and transcripts are now generated by a simulator S taking as input $\mathcal{L}^{\mathsf{Stp}}(\mathsf{DB})$ for the setup phase, $\mathcal{L}^{\mathsf{Srch}}(q)$ for search queries, and $\mathcal{L}^{\mathsf{Updt}}(\mathsf{in})$ for updates.

The scheme is said to be secure if both games are indistinguishable to any probabilistic adversary whose running time is polynomial in the security parameter λ.

Indistinguishability-based definition. The authors of [CGKO06] also define the security of searchable encryption, based on indistinguishability. For the security game of this definition, in the non-adaptive case (an adaptive version of the definition is also given in the paper), the adversary chooses two tuples of one database and search (and update) queries, also called *histories*, $H^1 = (\mathsf{DB}^1, r_1^1, \ldots, r_m^1)$ and $H^2 = (\mathsf{DB}^2, r_1^2, \ldots, r_m^2)$ such that their leakages are identical. The challenger randomly picks one of those histories and runs the setup

algorithm, and the search and update protocols with the adversary. The scheme is secure iff the adversary cannot correctly guess the picked database and queries with non-negligible advantage. This is formalized in Definition 2.1. In this paper, λ denotes the security parameter.

Definition 2.1 (Adaptive indistinguishability for SE)**.** *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be an SE scheme,* λ *the security parameter, and* $\mathcal{A}$ *a stateful algorithm. Let* $\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L}}$ *be the following game:*

$\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L}}(\lambda)$ *Game:*
- $b \xleftarrow{\$} \{0,1\}$
- $(\mathsf{DB}^0, \mathsf{DB}^1) \leftarrow \mathcal{A}(1^\lambda)$
- $(K, \mathsf{EDB}^b) \leftarrow \mathsf{Setup}(\mathsf{DB}^b)$
- $(q_1^0, q_1^1) \leftarrow \mathcal{A}(\mathsf{EDB}^b)$
- $\tau_1^b \leftarrow \mathsf{Query}(q_1^b)$
- ***for*** $i = 2$ ***to*** n ***do***
  - $(q_i^0, q_i^1) \leftarrow \mathcal{A}(q_{i-1}^b)$
  - $\tau_i^b \leftarrow \mathsf{Query}(q_i^b)$
- ***end for***
- $b' \leftarrow \mathcal{A}(\tau_n^b)$
- ***if*** $b = b'$ ***return*** 1, ***otherwise return*** 0

where $\tau_i^b \leftarrow \mathsf{Query}(q_i^b)$ *means that* τ_i^b *is the transcript of the query* q_i^b *(which can be a search or an update query), and with the restrictions that, for* $H^0 = (\mathsf{DB}^0, q_1^0, \ldots, q_n^0)$ *and* $H^1 = (\mathsf{DB}^1, q_1^1, \ldots, q_n^1)$, $\mathcal{L}(H^0) = \mathcal{L}(H^1)$.

We say that Σ *is* $\mathcal{L}$*-adaptively-indistinguishable if for all polynomial in* λ *time adversary* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathcal{L}}(\lambda) = \mathbb{P}[\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L}}(\lambda) = 1] - \frac{1}{2} \leq \mathit{negl}(\lambda).$$

It is shown in [CGKO06] that, in the non-adaptive case, both security definitions are equivalent, and that, in the adaptive case, the simulation based security implies the indistinguishability based security. Yet, the proof crucially relies on the fact that, given a database and queries, one can efficiently find a different database and queries with the same leakage. In [CGKO06] a database and a list of queries satisfying this property is called *non-singular*.

2.2 Leakage Abuse Attacks

The terminology 'leakage abuse attack' has been introduced by Cash *et al.* [CGPR15], and denotes attacks using only the leakage of schemes to break their security, rather than exploiting some particular weakness in their components. This paper extended the prior work by Islam *et al.* [IKK12] which considered a similar setting.

Both of these attacks suppose knowledge (total, partial or distributional) of the database by the adversary. In [IKK12], she knows the distribution $\mathcal{D}$ of the co-occurrence matrix of the targeted set of keywords. From the observation of the documents access pattern (the list of result indices), the adversary builds a co-occurrence matrix that follows the distribution $\mathcal{D}$, up to rows and columns permutation. Using simulated annealing, she will find this permutation, which will be the match between queries and keywords.

The count attack of [CGPR15] also uses a co-occurrence matrix, but uses it only to leverage some prior knowledge issued from the uniqueness of the number of results for some keywords: there are some keywords w such that no other keyword w' match the exact same number of documents. Hence, for the full or partial knowledge of the database, the adversary is able to decrypt the queries keyword.

On the other side, the attack of Kellaris *et al.* [KKNO16] targets range queries, and only supposes that the queries are performed uniformly at random to recover the entire dataset, using only the number of results for each query. In particular, this attack is successful even against ORAM-based constructions: even though they only leak the result count, this is sufficient to break the dataset's secrecy.

Note that all these attacks are passive and hence, non-adaptive: the adversary does not choose either the database, or the queries. Hence, the security definitions fail to prevent these attacks, even in the simpler

non-adaptive case. However, adaptive attacks do exist: some very efficient files injection attacks presented by Zhang *et al.* [ZKP16] are adaptive. In order to decrypt a previous search query, the adversary uses the update leakage of the scheme and adaptively inserts a sequence of well crafted documents in the database. Yet, these adaptive attacks can be circumvented using so-called *forward-secure* SE schemes (*cf.* [SPS14, Bos16]), and to our knowledge, except these ones, all the existing leakage abuse attacks are not adaptive.

Why does leakage abuse work? One might ask why these attacks using only the leakage are so efficient although the schemes are proven secure against adversary using this exact same leakage. In particular, we saw that the indistinguishability-based security definition states that the adversary should not be able to distinguish two executions of the SE scheme with the same leakage.

Moreover, for all the leakage targeted by the previous works, the very important observation of [CGKO06, Section 4.2] stands:

> *Note that the existence of a second history with the same trace is a necessary assumption, otherwise the trace would immediately leak all information about the history.*

Namely, it is fairly easy, given a database and queries list, to construct an *other* database and queries list with the exact same leakage, for all the common leakage used in searchable encryption (*e.g.* one could permute the database's keywords): histories are non-singular.

Yet, attacks like the one of Islam *et al.* [IKK12] or the count attack of [CGPR15] suppose some server knowledge of the database. This knowledge *pins* the database in the security proof: for the proof to be useful in this setting, one needs to find two different lists of queries generating the exact same leakage with the same (public) database. And the whole point of leakage abuse attacks is that this is impossible: knowing the database, the queries' list is uniquely defined by the leakage. For example, once the database is committed to, one cannot permute the keywords anymore to construct a different history with the same leakage. Also, the frequency of words in the English language is fixed, so if the adversary knows that a dataset stores some English-written documents, any syntactic permutation of the keywords would not hide the real keywords.

A similar point can be made with the attacks in [KKNO16]: queries are known to be uniform, and many of them are performed. And for the active attacks of Zhang *et al.* [ZKP16] too, but instead of controlling the database, the attacker controls some update queries as she injects documents she has purposely built.

In all of these examples, the adversary can decrypt queries and/or the database because they are unique given the constraints (fixed database, knowledge of the queries distribution, knowledge or control of some updates, ...) and the leakage.

An other way to see this issue with security definitions for SE is the following: the existing definitions protect the database and the queries as a whole, but once one part gets leaked (*e.g.* the database in [CGPR15]) these definitions are of no use anymore. If one only wants to protect the queries, it might be more suitable to use private information retrieval (PIR) definitions [KO97].

On the meaningfulness of security definitions. A parallel can be made with the CPA security definition for encryption: the definition states that the adversary has to give two messages of equal length to the challenger. If the message space contains a message with a unique length, this message is not protected by the security definition. If every element of the message space has a unique length, any scheme (*e.g.* a scheme whose encryption function is the identity) can be shown CPA-secure although it is trivially insecure: it will be impossible for an adversary to find a pair of same length messages on which to be challenged. In this specific setting, the CPA security definition is void, as is the indistinguishability based security definition for SE with prior knowledge.

3 Fixing the Security Definition

We saw in the previous section that SE fails against leakage abuse attacks is because there is uniqueness of histories given some constraints (the leakage plus other external constraints such as the distribution of queries

or the publicity of the dataset). To fix this problem, a searchable encryption scheme's leakage function must be so that, given a constraint, histories are no longer uniquely defined by the leakage function.

In this section, we propose new tools and definitions to capture leakage abuse attacks.

3.1 Constraints

Defining constraints is a way to formalize that the adversary knows some information about the history. In particular, we must be able to tell if a history *conforms* to the information known by the adversary. We could ask the adversary to represent this information as the known database plus the list of queries she knows, but that would be overly restrictive: we would not be able to represent partial knowledge.

A more general way to represent this knowledge is by using constraints defined by a predicate over histories: the history H satisfies the constraint C iff $C(H) = \mathsf{true}$. However, we need an adaptative way to define contraints: the adversary may want to insert a document depending on the transcript of previous queries. This is what Definition 3.1 captures.

Definition 3.1 (Constraint)**.** *A constraint $C = (C_0, C_1, \ldots, C_n)$ (with $n = poly(\lambda)$) over a database set $\mathcal{DB}$ and query set $\mathcal{Q}$ is a sequence of algorithms such that, for $\mathsf{DB} \in \mathcal{DB}$, $C_0(\mathsf{DB}) = (\mathsf{flag}_0, st_0)$ where flag_0 is true or false and st_0 captures C_0's state, and for $q \in \mathcal{Q}$, $C_i(q, \mathsf{flag}_{i-1}, st_{i-1}) = (\mathsf{flag}_i, st_i)$. The constraint is* consistent *if $C_i(., \mathsf{false}, .) = (\mathsf{false}, .)$ (once the constraint evaluates to false, it remains false).*

For a history $H = (\mathsf{DB}, q_1, \ldots, q_n)$, we note $C(H)$ the evaluation of

$$C(H) := C_n(q_n, C_{n-1}(q_{n-1}, C_{n-2}(\ldots, C_0(\mathsf{DB})))).$$

If $C(H) = \mathsf{true}$, we say that H satisfies *C. A constraint C is* valid *is there exists two different efficiently constructible histories H and H' satisfying C.*

The validity of the constraint makes sure that the adversary does not know everything about the history.

Note that this definition of constraints does not extend to distributional knowledge. We refer to Appendix A for a security definition supporting prior distributional knowledge by the adversary. In the following, we will restrict ourselves to deterministic knowledge by the server. Also, to simplify the notations, we will omit passing the states st_i. In this paper, we will only consider valid constraints.

We also want to formalize the fact that some elements of the history are completely unknown to the adversary, *i.e.* that their are left *free* from constraint.

Definition 3.2 (Free history component)**.** *Let C be a constraint.*

We say that C lets the database free *if, for every history $H = (\mathsf{DB}, q_1, \ldots, q_n)$ satisfying C, for every $\mathsf{DB}' \in \mathcal{DB}$, $H' = (\mathsf{DB}', q_1, \ldots, q_n)$ also satisfies C.*

We say the C lets the i-th query free *if for every history $H = (\mathsf{DB}, q_1, \ldots, q_n)$ satisfying C, for every search (resp. update) query q if q_i is a search (resp. update) query, $H' = (\mathsf{DB}', q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)$ also satisfies C.*

Finally, an other very important notion is the one of *acceptable* constraint, that will help us to give non-void security definition: as explained in Section 2.2, given a constraint C and a leakage function $\mathcal{L}$, for every history H we want to be able to find a different history satisfying C with the same leakage.

Definition 3.3 (Acceptable constraint)**.** *A constraint C is* $\mathcal{L}$-acceptable *for some leakage $\mathcal{L}$ if, for every efficiently computable history H satisfying C, there exists an efficiently computable $H' \neq H$ satisfying C such that $\mathcal{L}(H) = \mathcal{L}(H')$.*

A set of constraints $\mathfrak{C}$ is said to be $\mathcal{L}$-acceptable if all its elements are $\mathcal{L}$-acceptable.

Section 3.3 will give example of constraints. We first explain how we will formally use this new tool.

3.2 Constrained security

Now that we have formally defined what are the constraints, we can use them to give a new flavor of security for history satisfying these constraints.

Definition 3.4 (Constrained adaptive indistinguishability for SE)**.** *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be an SE scheme,* λ *the security parameter, and* $\mathcal{A}$ *a stateful algorithm. Let* $\mathcal{L}$ *be a leakage function and* $\mathfrak{C}$ *be a set of* $\mathcal{L}$*-acceptable constraints. We can define the notion of* constrained adaptive indistinguishability *using the following game*

$\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C}}(\lambda)$ *Game:*
$b \xleftarrow{\$} \{0,1\}$
$(C_0, \mathsf{DB}^0, \mathsf{DB}^1) \leftarrow \mathcal{A}(1^\lambda)$
$(K, \mathsf{EDB}^b) \leftarrow \mathsf{Setup}(\mathsf{DB}^b)$
$(C_1, q_1^0, q_1^1) \leftarrow \mathcal{A}(\mathsf{EDB}^b)$
$\tau_i^b \leftarrow \mathsf{Query}(q_1^b)$
for $i = 2$ ***to*** n ***do***
  $(C_i, q_i^0, q_i^1) \leftarrow \mathcal{A}(q_{i-1}^b)$
  $\tau_i^b \leftarrow \mathsf{Query}(q_i^b)$
end for
$b' \leftarrow \mathcal{A}(\tau_n^b)$
if $b = b'$ ***return*** 1, ***otherwise return*** 0

with the restrictions that, for $H^0 = (\mathsf{DB}^0, q_1^0, \ldots, q_n^0)$ *and* $H^1 = (\mathsf{DB}^1, q_1^1, \ldots, q_n^1)$,

- $C \in \mathfrak{C}$, $C(H^0) = \mathsf{true}$, *and* $C(H^1) = \mathsf{true}$ *;*
- $\mathcal{L}(H^0) = \mathcal{L}(H^1)$.

We say that Σ *is* $(\mathcal{L}, \mathfrak{C})$*-constrained-adaptively-indistinguishable if for all polynomial time* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathcal{L},\mathfrak{C}}(\lambda) = \mathbb{P}[\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C}}(\lambda)] - \frac{1}{2} \leq \mathit{negl}(\lambda).$$

Definition 3.4 is similar to the original SE security definition (Definition 2.1), with the main difference being the introduction of the condition that both histories must satisfy the constraint.

Note that a weaker, non-adaptive security notion can be easily derived from Definition 3.4 by making the adversary output the whole constraint and histories at once, at the beginning of the game.

If the leakage function is probabilistic, we replace the last restriction by the condition that the distribution of $\mathcal{L}(H^0)$ and $\mathcal{L}(H^1)$ must be computationally indistinguishable.

We underline again that the constraint can be seen as some information the server knows about the histories: the histories both have to satisfy the same constraint.

The fact that the constraints are acceptable implies that the definition is not void. Also, we can prove the following theorem, stating that we only have to prove (resp. give counter-examples of) the acceptability of some constraints given some common leakage function $\mathcal{L}$ to show the security (resp. insecurity) of existing schemes.

Theorem 1. *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be an SE scheme, and* $\mathfrak{C}$ *a set of constraints. If* Σ *is* $\mathcal{L}$*-adaptive-indistinguishability secure, and* $\mathfrak{C}$ *is* $\mathcal{L}$*-acceptable, then* Σ *is* $(\mathcal{L}, \mathfrak{C})$*-constrained-adaptive-indistinguishability secure.*

Proof. Suppose $\mathfrak{C}$ is $\mathcal{L}$-acceptable. Then, for any satisfying pair (H^0, H^1) of histories such that $\mathcal{L}(H^0) = \mathcal{L}(H^1)$, the views of the adversary will be indistinguishable, from the $\mathcal{L}$-adaptive-indistinguishability. □

3.3 Examples of constraints

Using these constraints, it is easy to model the fact that the adversary knows some information about the database or about some queries. This section gives example of constraints for existing leakage abuse attacks.

Prior knowledge of the database. Let us consider the setting of the count attack of Cash *et al.* [CGPR15]: the adversary knows the database DB and uses the leakage of the search queries to decrypt them. In the security definition, we want to capture that the adversary knows DB.

To do so, we will use the predicate C^{DB} that returns true iff the database of the input history is DB. Used in the security definition, this predicate will ensure that the both challenge histories' database is DB, and that all the queries are left free. [CGPR15, Section 4.2] shows that, for the leakage function $\mathcal{L} = L1$ (the repetition of queries, the number of documents matching each queries and, for every pair of queries, the documents in common), C^{DB} is not $L1$-acceptable: many keywords have a unique number of matching documents, and as the adversary knows the database, queries on these keywords can be decrypted just from the results count, and all the others queries from co-occurrence information.

More generally, we model the fact that the adversary knows the database by considering the set $\mathfrak{C}^{\mathcal{DB}} = \{C^{\mathsf{DB}}, \mathsf{DB} \in \mathcal{DB}\}$ where $\mathcal{DB}$ is the set of polynomially computable databases.

Known documents subset. We similarly define the partial knowledge of the dataset: if the adversary knows that the database contains documents $D_1, \ldots, D_\ell$, we will use the constraint $C_{D_1,\ldots,D_\ell}$ that returns true iff DB contains D_i for all i. Cash *et al.* also showed that for the leakage function $\mathcal{L} = L3$ (which leaks the pattern of keyword occurrences in documents – but not the occurrence count), $C_{D_1,\ldots,D_\ell}$ is not $L3$ acceptable [CGPR15, Section 5.1].

File injection attacks. This formalization can be also used to capture file injection attacks [ZKP16]. Say the attacker inserts ℓ documents $D_1, \ldots, D_\ell$, D_j being inserted during the i_j-th query. We construct C so that $C_0(.)$ always outputs true (the adversary does not know the database at the beginning), $C_{i_j}(\mathsf{flag}, q)$ outputs true iff flag is true and q is the query inserting D_j in the database (the i_j-th query is forced to be the insertion of D_{i_j}), and $C_i(\mathsf{flag}, q)$ outputs flag when $i \notin \{i_1, \ldots, i_\ell\}$ (for all the other queries, there is no constraint).

This constraint can be used both for the non-adaptive and the adaptive attacks of Zhang *et al.*. For the non-adaptive attack, the adversary will choose the D_j so that he will be able to run a binary search for all the subsequent search queries. For the adaptive attack, to break the privacy of a previous search query, the adversary will choose the successive documents to be inserted using the update leakage of the previously insertion query. We refer the reader to [ZKP16] for further details on these attacks.

As before, we can generalize all file injection attacks by considering the constraints for all the polynomially constructible lists of pairs $\{(D_j, i_j)\}_{1 \leq j \leq \ell}$. Hence, [ZKP16] shows that this set of constraints is not $\mathcal{L}$-acceptable when $\mathcal{L}$ leaks the file-access pattern (which is the case for all the existing non-ORAM-based SE schemes).

3.4 Devising new leakage abuse attacks using constraints

For now, we have used constraints as as way to formalize the security of schemes against leakage abuse attacks. But we can also use them as a way to construct new attacks.

Indeed, with leakage abuse attacks, we suppose that the targeted scheme is $\mathcal{L}$-indistinguishable for some leakage function $\mathcal{L}$, and breaking the $(\mathcal{L}, \mathfrak{C})$-constrained-indistinguishability for some set of constraints $\mathfrak{C}$ implies that $\mathfrak{C}$ is not $\mathcal{L}$-acceptable. To mount a new attack, one could then just check if there exists an history satisfying $C \in \mathfrak{C}$ such that no other history satisfying C has the same leakage. Even though this step can be (very) fastidious by hand, it can be automated using constraint programming. However, we would have to make sure that the adversarial knowledge induced by the constraint is realistic and reasonable (*e.g.* that there is not too much prior knowledge).

3.5 Extending the definition

The new constrained security definition, despite being satisfactory because it solves the definitional issue we had with leakage abuse attacks, still is not enough in terms of real-world security.

Indeed, it ensures that the adversary will not be able to distinguish between two histories. Yet, when proposed k different histories satisfying the same constraint and sharing the same leakage, she could be able to easily discard one of these. Said otherwise, Definition 3.4 guarantees us that there is some uncertainty for the adversary, but not how much: the security definition only guarantees one bit of security.

However, we can extend it by modifying the $\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C}}(\lambda)$ game: instead of adaptively outputting a pair of histories, $\mathcal{A}$ could output k of them, and the challenger randomly picks the one to be guessed. For this definition to make sense, we will have to make sure that k constrained histories can actually be found, and we also have to extend the definition of acceptable constraints.

Definition 3.5 (Extended acceptable constraint)**.** *A constraint C is* $(\mathcal{L},k)$-acceptable *for some leakage $\mathcal{L}$ and integer $k>1$ if, for every efficiently computable history H^0 satisfying C ($C(H^0)=\mathsf{true}$), there exists $k-1$ efficiently computable $\{H^i\}_{1\leq i\leq k-1}$ such that $H^i\neq H^j$ for $i\neq j$, that are all satisfying C, and $\mathcal{L}(H^0)=\cdots=\mathcal{L}(H^{k-1})$*

A set of constraints $\mathfrak{C}$ is said to be $(\mathcal{L},k)$-acceptable iff all its elements are $(\mathcal{L},k)$-acceptable.

Definition 3.6 (Extended constrained adaptive indistinguishability for SE)**.** *Let $\Sigma=(\mathsf{Setup},\mathsf{Search},\mathsf{Update})$ be an SE scheme, λ the security parameter, and $\mathcal{A}$ a stateful algorithm. Let $\mathfrak{C}$ be a set of $(\mathcal{L},k)$-acceptable constraints. We can define the notion of* constrained adaptive indistinguishability *using the following game*

$\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k}(\lambda)$ *Game:*

$\ell \xleftarrow{\$} \{0,\dots,k-1\}$
$(C_0,\mathsf{DB}^0,\dots,\mathsf{DB}^{k-1})\leftarrow\mathcal{A}(1^\lambda)$
$(K,\mathsf{EDB}^\ell)\leftarrow\mathsf{Setup}(\mathsf{DB}^\ell)$
$(C_1,q_1^0,\dots,q_1^{k-1})\leftarrow\mathcal{A}(\mathsf{EDB}^\ell)$
$\tau_i^\ell\leftarrow\mathsf{Query}(q_1^\ell)$
for $i=2$ ***to*** n ***do***
  $(C_i,q_i^0,\dots,q_i^{k-1})\leftarrow\mathcal{A}(q_{i-1}^\ell)$
  $\tau_i^\ell\leftarrow\mathsf{Query}(q_i^\ell)$
end for
$\ell'\leftarrow\mathcal{A}(\tau_n^\ell)$
if $\ell=\ell'$ ***return*** 1, ***otherwise return*** 0

with the restrictions that, for all the H^i,

- $C\in\mathfrak{C}$, *and* $\forall 0\leq i\leq k-1,\ C(H^i)=\mathsf{true}$ *;*
- $\mathcal{L}(H^0)=\cdots=\mathcal{L}(H^{k-1})$.

We say that Σ is $(\mathcal{L},\mathfrak{C},k)$-constrained-adaptively-indistinguishable if for all polynomial time $\mathcal{A}$,

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathcal{L},\mathfrak{C},k}(\lambda)=\mathbb{P}[\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k}(\lambda)]-\frac{1}{k}\leq negl(\lambda).$$

An $(\mathcal{L},\mathfrak{C},k)$-constrained-adaptively-indistinguishable scheme offers at least $\log k$ bits of security. Extended constrained indistinguishability is implied by regular indistinguishability and extended acceptability of constraints, as stated in Theorem 2, whose proof is given in Appendix B.

Theorem 2. *Let $\Sigma=(\mathsf{Setup},\mathsf{Search},\mathsf{Update})$ be an SE scheme, and $\mathfrak{C}$ a set of constraints. If Σ is $\mathcal{L}$-adaptive-indistinguishability secure, and $\mathfrak{C}$ is $(\mathcal{L},k)$-acceptable, then Σ is $(\mathcal{L},\mathfrak{C},k)$-constrained-adaptive-indistinguishability secure.*

4 Keywords Clustering

Section 3 gives us the formal tools to assess the security of a SE construction. Yet these tools are not constructive: for a given constraint and history, it is hard to tell how many other histories satisfying the constraint have the same leakage, or even if one exists.

In this section, we propose an easy way to evaluate the security of leakage functions for a usual class of constraints (partial or complete knowledge of the database): this approach will allow for queries' privacy protection.

4.1 Regrouping Keywords with Equal Leakage

We will suppose as, a starting point, that the leakage function only depends on the query itself and on the state of the database: $\mathcal{L}(q)$ can be written as a stateless function $f_{\mathcal{L}}$ of q and DB.

We make the following very simple observation: let C be a constraint, $H = (\mathsf{DB}, q_1, \ldots, q_n)$ an history satisfying C, and q, q' be two queries such that $\widetilde{H} = H||q = (\mathsf{DB}, q_1, \ldots, q_n, q)$ and $\widetilde{H}' = H||q' = (\mathsf{DB}, q_1, \ldots, q_n, q')$ are both satisfying C. Then, if $f_{\mathcal{L}}(\mathsf{DB}, q) = f_{\mathcal{L}}(\mathsf{DB}, q')$, then $\widetilde{H}$ and $\widetilde{H}'$ are two histories with the same leakage satisfying C. This observation can be iterated to easily create histories satisfying the constraint and with the same leakage, showing the acceptability of the said constraint.

More generally, a clustering $\Gamma = \{G_1, \ldots, G_m\}$ of queries induced by the leakage $\mathcal{L}$ after history H is a partition of the queries set $\mathcal{Q}$ for which, in every subset, queries share the same leakage after running the history H:

$$\bigcup_{i=1}^{m} G_i = \mathcal{Q}$$
$$\forall i \neq j \ G_i \cap G_j = \emptyset$$
$$\text{and } \forall q, q' \in G_i, \ \mathcal{L}(H, q) = \mathcal{L}(H, q'),$$

where $\mathcal{L}(H, q)$ is the output of $\mathcal{L}(q)$ after having been run on each element of H. In the following, we denote by $\Gamma_{\mathcal{L}}(H)$ the clustering induced by $\mathcal{L}$ after H, *i.e.* the clustering for which it is impossible to *merge* clusters with the same leakage. Formally, for $\Gamma_{\mathcal{L}}(H) = \{G_1, \ldots, G_m\}$, we have

$$\forall i \neq j, \forall q \in G_i, \forall q' \in G_j, \ \mathcal{L}(H, q) \neq \mathcal{L}(H, q').$$

Also, we denote by $\Gamma_{\mathcal{L},C}(H)$ the clustering of the subset $\mathcal{Q}_C(H)$ of queries q such that $C(H||q) = \mathsf{true}$. We can easily see that, in the specific case studied before, where $\mathcal{L}(q)$ is a function of DB and q only, $\Gamma_{\mathcal{L}}$ only depends on DB and, in the static case, not on previous queries.

We want that every cluster of $\Gamma_{\mathcal{L},C}(H)$ contains at least two elements. If this is not the case, one is able to construct an history H satisfying C without having any different history H' with the same leakage profile, also satisfying C: C will not be $\mathcal{L}$-acceptable, and this may lead to a new leakage abuse attack. Yet, this only makes sense if $|\mathcal{Q}_C(H)| > 1$ (there is more than one satisfying query).

In the opposite, we can show that when there is strictly more than one element in each cluster of the $\mathcal{L}$-induced clustering applied on every history satisfying C, C will be $\mathcal{L}$-acceptable, as formalized in Proposition 3.

Proposition 3. *Let C be a constraint, and $\mathcal{L}$ a leakage function. If for every history H satisfying C, the clustering $\Gamma_{\mathcal{L},C}(H) = \{G_1, \ldots, G_m\}$ is such that $|G_i| \geq k$ for all i, then C is $(\mathcal{L}, k)$-acceptable.*

Unfortunately, the condition that $|G_i| \geq k$ is very strong: constraints fixing a particular query will never verify it. On the other side, it looks very hard to give a better result. Take for example a static scheme whose leakage function $\mathcal{L}$ gives away the search pattern, *i.e.* for a search query q after the queries $(q_1, \ldots, q_n)$, the set $\mathsf{sp}(q) = \{i | q_i = q\}$, and consider the constraint $C(H)$ which return true iff $\mathsf{DB} = \overline{\mathsf{DB}}$ and $q_2 = q$ (the database and second query are fixed). Then $C^{2,q}$ will not be $\mathcal{L}$-acceptable: $H = (\overline{\mathsf{DB}}, q, q)$ has no matching history satisfying C with the same leakage.

As the search pattern is leaked for almost all practical SE schemes, we can see that generic constrained security is very hard to achieve. Hence, in the following, we will restrict ourselves to common constraints, *i.e.* common adversarial prior knowledge in leakage abuse attacks.

4.2 Applications to Common Leakage Profiles and Constraints

In this section, we will see how to use the clustering approach to assess the security of single-keyword SE schemes with simple, yet common leakage, against existing attacks. Let us first focus on static schemes.

Result length leakage. All (reasonable) schemes leak at least the number of matches of a search query, and even this leakage can break the queries' privacy, as shown by Cash *et al.* [CGPR15] (63% of the 500 most common words of the Enron database have a unique result count). In particular, this implies that ORAM-based schemes [GMP16], whose leakage $\mathcal{L}_{len}$ is limited to the size of the database and to the number of queries' matches, are not resilient to leakage abuse attacks when the adversary knows the database: $\mathfrak{C}^{\mathcal{DB}}$ is not $\mathcal{L}_{len}$-acceptable.

Now, suppose that we add to a scheme with $\mathcal{L}_{len}$ leakage a padding mechanism such that, for every keyword, there is a different keyword with the same number of matching documents. The leakage function is now slightly modified to output the number of results, including fake documents, for a search query, giving the function $\mathcal{L}_{len}^{\alpha-pad}$, where α will be the minimum size of clusters induced by $\mathcal{L}_{len}^{\alpha-pad}$. Then, from Proposition 3, we have that $\mathfrak{C}^{\mathcal{DB}}$ is $(\mathcal{L}_{len}^{\alpha-pad}, \alpha)$-acceptable.

Yet, using the fact that $\mathcal{L}_{len}^{\alpha-pad}(q)$ is only a function of DB and q (and does not depend on previous queries), we can show a more general result, also better in terms of security.

Proposition 4. *Let C be a constraint with k free queries (*cf. *Definition 3.2), and $\mathcal{L}$ a leakage function such that $\mathcal{L}(q)$ is a stateless function of DB and q. Then the clustering $\Gamma_{\mathcal{L}}(H)$ only depends on DB. Let $\alpha = \min_i |G_i|$. Then C is $(\mathcal{L}, \alpha^k)$-acceptable.*

The idea behind Proposition 4 is that we can change any of the k free queries of an history H by picking a different query in the same cluster, without modifying the leakage nor making the C not satisfied. As the k queries are free and the leakage of each query does not depend on the other ones, we can combine all the possibilities and create α^k histories with the same leakage.

Proposition 4 gives a security level that is a lot better than the one implied by Proposition 3: $(\mathcal{L}, \alpha)$-acceptability for Proposition 3 *vs.* $(\mathcal{L}, \alpha^k)$-acceptability for Proposition 4. This last proposition actually offers $\log \alpha$ bits of security for each search query, while the first one only offered security for the whole history and not individual query.

Hence, we only have to design a padding algorithm to ensure the security of ORAM-based schemes against attackers with prior knowledge of the database. We present such an algorithm in Section 5.

Result length and search pattern. Unfortunately, $\mathcal{L}_{len}$ leakage only covers not really practical solutions. Many of the existing static schemes also leak the *search pattern*, the repetition of search queries. It is similar to the $L1$ leakage of [CGPR15], but $L1$ is not result hiding and leaks searched keywords co-occurrence. We denote this leakage function $L1_{RH}$. In particular, we are no longer in the setting of Proposition 4 where the leakage is independent of the past queries. Also, even if we use padding to hide the results length (and end up with leakage function $L1_{RH}^{\alpha-pad}$), Proposition 3 does not apply either: it is easy to construct an history such that the clustering $\Gamma_{L1_{RH}^{\alpha-pad}, C^{\mathsf{DB}}}(H)$ has clusters containing only one query (for repeating queries).

However, we can still show $\mathfrak{C}^{\mathcal{DB}}$ is an $(L1_{RH}^{\alpha-pad}, \alpha)$-acceptable set of constraints, where α is the minimum cluster size (over all constructible databases). Indeed, as constraints in $\mathfrak{C}^{\mathcal{DB}}$ leave all queries free, for every history $H = (\mathsf{DB}, q_1, \ldots, q_n)$, we can generate a different history H' with the same leakage by choosing an other first query $q \neq q_1$ matching the same number of documents, and changing all queries $q_i = q_1$ to q. Also, if there is queries $q_j = q$ in H, we switch them to q_1. This gives us a history $H' \neq H$ with the same leakage as H, and as we have at leat $\alpha - 1$ choices for q, we infer the $(L1_{RH}^{\alpha-pad}, \alpha)$-acceptability.

Finally, it is interesting to notice that, if we force queries to be different (*e.g.* using a dedicated constraint), we can show a better result. Namely, let $\mathfrak{C}^{\mathcal{DB},k}$ be the set of constraints fixing the database and satisfied by histories with k search queries that are pairwise different. Then $\mathfrak{C}^{\mathcal{DB},k}$ is $(L1_{RH}^{\alpha-pad}, \beta(\alpha,k))$-acceptable where $\beta(\alpha,k) = \frac{a!}{(a-k-1)!}$ for $k \leq \alpha$ and $\beta(\alpha,k) = a!$ otherwise. This result might be surprising: we add more constraints, but we have a higher apparent security level. However this is consistent with our security

definitions, yet counter-intuitive: indeed, adding the pairwise distinct queries constraint reduces the space of histories satisfying the constraints, and somewhat artificially removed the histories that had only $\alpha - 1$ matching histories with the same constraint. We can also interpret this as the fact that, previously, the adversary could have learned that the queries were not repeating, but as she already knows this information when using constraints in $\mathfrak{C}^{\mathcal{DB},k}$, there is no problem in leaking it.

5 Application to Database Padding with Best Possible Security

In this section, we will show how to pad the database in order to achieve $\mathcal{L}_{len}^{\alpha-pad}$, as presented in section 4.2. In particular, we want to create clusters of minimal size α, based on the number of matches of every search query.

We want to solve this problem not only for static databases, but also for dynamic ones. Also, we present here a *black-box* construction: we apply the countermeasure to scheme with $\mathcal{L}_{len}$ (resp. $L1_{RH}$) leakage (leaking only the result length - resp. the result length and the search pattern) without needing access to the inner machinery of the scheme, to turn them into $\mathcal{L}_{len}^{\alpha-pad}$ (resp. $L1_{RH}^{\alpha-pad}$) secure schemes. We only need the client to store a table with $|W|$ entries, counting the occurrence of every keyword (note that many dynamic SE schemes already need $O(|W|)$ – or similar – permanent or transient storage [CJJ+14, SPS14, GMP16, Bos16]).

5.1 Using Frequencies Instead of Counts

To make our analysis easier and more general, we will work with keywords' frequency instead of exact result count. Namely, if the adversary knows the database (*e.g.* in a static setting), she will easily derive the frequencies, while our approach also covers an adversary with distributional knowledge of the database (typical in a dynamic setting without file injection attack). Also, we adopt a distributional approach, but, again in the case the adversary entirely knows the database, we can replace the expectancies by the actual real values.

So, let $\mathcal{DB}$ be a distribution, with keywords in the set W. For $\mathsf{DB} \leftarrow \mathcal{DB}$ and $w \in W$, we recall that $N_w = |\mathsf{DB}(w)|$, and $N = |DB|$. We note the expected frequency of w as e_w:

$$e_w := \mathbb{E}_{\mathsf{DB}\leftarrow\mathcal{DB}}\left[\frac{N_w}{N}\right].$$

N_w follows a multinomial law of N trials with event probability e_w:

$$\sum_{w\in W} N_w = N \text{ and } \mathbb{V}\mathrm{ar}(N_w) = Ne_w(1-e_w)$$

By applying Chebyshev's inequality to N_w, we have

$$\mathbb{P}[|f_w - e_w| \geq \varepsilon] \leq \frac{e_w(1-e_w)}{N\varepsilon^2}. \tag{1}$$

where $f_w := \frac{N_w}{N}$ is the real frequency of w in DB. In particular, Equation (1) tells us that the observed frequency converges towards the expected frequency as the database grows, and an adversary will be able to tell if a keyword w is a good candidate for a query matching n documents from the distance between the observed frequency f_w and the expected frequency e_w for w.

As mentioned in Section 4.2, to thwart count-based/frequency-based attacks, we want to pad the database so that, for every keyword w, there are at least $\alpha - 1$ different keywords with the same frequency in the database. Here, we are talking about the *observed* frequencies, *i.e.* the frequencies after padding, the ones the scheme will leak to the adversary. Also, once we know that the expected observed frequencies of different keywords are identical, Equation (1) ensures that the actual frequencies of these keywords will converge to the same value, and will be indistinguishable, forming a cluster with these keywords.

5.2 How to Pad

In the following, we will denote the expected real (resp. observed) frequency of keyword w by e^r_w (resp. e^o_w). The clusters will be formed by keywords with the same expected observed frequency: $\Gamma(e^o) = (G_1, \ldots, G_m)$ such that $\exists(\widetilde{e}_1, \ldots, \widetilde{e}_m)$ with $G_i = \{w | e^o_w = \widetilde{e}_i\}$.

To achieve this, the client will pad the real keyword distribution by inserting fake entries (keyword/document pairs) whose keyword is chosen according to a *padding distribution*, a multinomial distribution of parameter e^p_w. More formally, when answering a query on keyword w, the server will see that it matches N^o_w documents which can be decomposed as

$$N^o_w = N^r_w + N^p_w \tag{2}$$

where N^r_w is the number of real documents matching w and N^p_w is the number of fake entries used for padding. Similarly, the total number of entries in the padded database, N^o can be decomposed as $N^r + N^p$, with N^r being the number of real entries and N^p the number of fake entries. As previously, we can express e^o_w, e^r_w, and e^p_w – the parameter of the padding distribution – as

$$e^o_w = \mathbb{E}\left[\frac{N^o_w}{N^o}\right], e^r_w = \mathbb{E}\left[\frac{N^r_w}{N^r}\right], \text{ and } e^p_w = \mathbb{E}\left[\frac{N^p_w}{N^p}\right].$$

We denote by γ the ratio of fake entries inserted in the database:

$$N^p = \gamma N^r$$

By combining this with Equation (2), we end up with the following relationship among the expected keyword frequency:

$$e^o_w = \frac{1}{1+\gamma} e^r_w + \frac{\gamma}{1+\gamma} e^p_w \Leftrightarrow e^p_w = (1 + \frac{1}{\gamma}) e^o_w - \frac{1}{\gamma} e^r_w. \tag{3}$$

For each new real entry added to the database, the client, knowing the distribution of the database, and with expected observed frequencies of his choice, will hence create, on average, γ padding entries with keywords chosen according to a categorical law of parameter $(e^p_w)_{w \in W}$.

Also, it is extremely important to notice that, as e^p_w must be comprised between 0 and 1, Equation (3) gives us a lower bound on γ:

$$\gamma \geq \max_{w \in W} \left\{ \frac{e^r_w - e^o_w}{e^o_w}, \frac{e^r_w - e^o_w}{e^o_w - 1} \right\}. \tag{4}$$

Hence, if the client does not want to add too many fake entries (to reduce the cost of padding), he cannot choose any expected observed frequency distribution.

Practical considerations on how to insert fake entries in the database. An important point to notice about γ and the way we do padding is that the adversary must not be able to distinguish fake entries from real ones when they are inserted. Otherwise, at least for structured encryption-based searchable encryption, the adversary will be able to filter between real and fake entries.

Also, if we suppose that the number of additional fake entries inserted per real entry is not constant, the adversary could mark the updates with a low number of fake entries and restrict the count-based attack to only these. This will help her to defeat the padding counter-measure more easily as the observed keyword frequency for this reduced subset of entries will be closer to the real frequency than for the entire database.

However, this does not prevent γ to be non-integral: for example to get $\gamma = 1/3$, instead of inserting a fake entry one time out of two, we could cache updates, wait for three of them to be available, and then send 4 entries (the three real ones and a fake one) to the server. Hence, in practice, we will choose a rational value for γ.

5.3 Constructing Frequency-based Clusters

In the previous section, we showed how the client could pad the database once he chose a target observed frequencies distribution that will form clusters. In this section, we will see how to construct this distribution in a way that minimizes γ, given the minimum cluster size α.

Formally, by the end of this section, we would have described an algorithm that, on input a real keyword distribution (e^r_w) and a parameter α, outputs an expected observed keyword frequency distribution (e^o_w) such that the clustering $\Gamma(e^o)$ has clusters of size at least α, and that the padding cost γ is minimized. This algorithm will run in time $\Theta((|W| - \alpha)\alpha)$.

Cost Metrics. γ can be seen as the cost of the countermeasure. It measures the additional server storage induced by the padding. However, one could consider instead other cost metrics, such as the bandwidth overhead, or the computational overhead for search queries. In this work, we only focus on the storage cost γ.

Expected frequencies with minimal cost for a given clustering. Constructing the clustering from the frequencies and optimize the choice of these is not easy in practice because the process of computing $\Gamma(e^o)$ from e^o is highly discontinuous and it is hard to predict what will happen to the sizes of clusters of $\Gamma(e^o)$ when e is changed.

On the other hand, it is easier to construct the expected frequencies from a reasonable clustering choice. Namely, a frequency-based clustering $\Gamma = (G_1, \ldots, G_m)$ and the associated expected frequencies $(\widetilde{e}_i)$ (*cf.* Section 5.2) have to satisfy the equations

$$\sum_{i=1}^{m} |G_i| = |W| \tag{5}$$

$$\sum_{i=1}^{m} |G_i| \widetilde{e}_i = 1. \tag{6}$$

Equation (6) comes from the fact that the frequencies e^o_w must sum to 1, and is obtained by regrouping keywords by cluster. Also, as we want to minimize the padding cost, we want γ to be as small as possible, and the minimum value for γ is, from Equation (4)

$$\gamma_{\min}(\Gamma, \widetilde{e}) = \max_{w \in W} \left\{ \frac{e^r_w - e^o_w}{e^o_w}, \frac{e^r_w - e^o_w}{e^o_w - 1} \right\} = \max_{1 \le i \le m} \left\{ \max_{w \in G_i} \left\{ \frac{e^r_w - \widetilde{e}_i}{\widetilde{e}_i}, \frac{e^r_w - \widetilde{e}_i}{\widetilde{e}_i - 1} \right\} \right\} \tag{7}$$

For a given frequency-based clustering Γ, we can find the expected observed frequencies $(\widetilde{e}_i)$ of keywords in each cluster minimizing the padding cost, as presented in Theorem 5, whose proof is in Appendix C.

Theorem 5. *For a cluster G_i of Γ, we denote $e^r_{\max(i)}$ the maximum value of e^r_w for $w \in G_i$. The minimum value $\gamma_{\min}(\Gamma)$ of $\gamma_{\min}(\Gamma, \widetilde{e})$ (over all possible cluster frequencies choices) is*

$$\begin{aligned} \gamma_{\min}(\Gamma) &= \min_{(\widetilde{e}_i)_{1 \le i \le m}} \gamma_{\min}(\Gamma, \widetilde{e}) \\ &= \left(\sum_{i=1}^{m} |G_i| e^r_{\max(i)} \right) - 1 \end{aligned} \tag{8}$$

and is attained for

$$(\widetilde{e}_i)_{1 \le i \le m} = (\widetilde{e}^*_i)_{1 \le i \le m} = \left(\frac{e^r_{\max(i)}}{1 + \gamma} \right),$$

i.e. $\gamma_{\min}(\Gamma, \widetilde{e}^*_i) = \gamma_{\min}(\Gamma)$.

Theorem 5 directly gives us an algorithm, that, given a clustering Γ, finds the keyword frequencies minimizing the padding cost, together with this cost $\gamma_{\min}(\Gamma)$. As a consequence, to find the expected observed frequencies distribution that minimizes the padding cost, and that induces clusters of size at least α, we only have to find the clustering of keywords inducing the minimum cost and deriving the frequencies $(\widetilde{e}_i)$.

Finding a clustering with minimal cost. The last step is to design an algorithm that, given expected real keyword frequencies (e^r_w), and parameter α, constructs a clustering Γ with a minimum cost $\gamma_{\min}(\Gamma)$ with the constraint that every cluster has size at least α.

To simplify the problem, we can notice that, if $\Gamma = (G_1, \ldots, G_m)$ reaches the minimum cost, and that there are two clusters G_i and G_j such that $e^r_{\max(i)} = e^r_{\max(j)}$, then we can 'merge' these two clusters and construct a clustering with the same cost (from Equation (8)), whose clusters size is larger than α. Hence, we can suppose that $e^r_{\max(i)} \neq e^r_{\max(j)}$ for every pair of clusters.

Also, for such a clustering reaching the minimum cost, we can suppose that

$$\forall G_i, G_j \in \Gamma, e^r_{\max(i)} \leq e^r_{\min(j)} \text{ or } e^r_{\min(i)} \geq e^r_{\max(j)}. \tag{9}$$

An equivalent (but more verbose) definition would be

$$\forall G_i, G_j \in \Gamma,\ (\forall w \in G_i, w' \in G_j, e^r_w \leq e^r_{w'}) \text{ or } (\forall w \in G_i, w' \in G_j, e^r_w \geq e^r_{w'}),$$

justifying that we call such clusterings *monotone*. We can easily show the following proposition, whose proof is given in Appendix D.

Proposition 6. *Let Γ be a non-monotone clustering with clusters of size at least α. Then there exists a clustering Γ' with clusters of size at least α, $\gamma_{\min}(\Gamma) \geq \gamma_{\min}(\Gamma')$ and Γ' is monotone.*

We now have to simple constraints on the clustering we want to construct: first its clusters are of size at least α, then the clustering should be monotone. Also, it will be handy in the following to suppose that the keywords are numbered such that their expected frequencies are non increasing: for $i < j$, $e^r_{w_i} \leq e^r_{w_j}$. Constructing such a clustering is equivalent to fixing $m' = m - 1$ *cluster limits* $\ell_0 = 0 < \ell_1 < \cdots < \ell_m < |W| = \ell_{m+1}$ such that

$$\forall 1 \leq i \leq m+1, \ell_i - \ell_{i-1} \leq \alpha.$$

and the associated clustering is $\Gamma_\ell = (G_1, \ldots, G_m)$ with $G_i = \{w_j | \ell_{i-1} < j \leq \ell_i\}$.

We could try to enumerate all these clusterings to find the clustering with minimum cost. However, the number of such clusterings is lower bounded by the number of partitions $p(|W|)$ of $|W|$ (there is a surjective mapping between these clusterings and the values taken by $\ell_i - \ell_{i-1}$), and $p(|W|)$ grows extremely fast as, from Hardy and Ramanujan [HR18],

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right).$$

Instead, we can reduce the problem of finding the optimal clustering to the problem of finding a shortest path in an directed acyclic graph (DAG) of $|W|+1$ vertices and $\frac{(|W|-\alpha)(|W|-\alpha+1)}{2}$ edges. Finding a solution to this problem can be done very efficiently, in in time complexity $\Theta((|W|-\alpha)^2)$ and memory complexity $\Theta(|W|)$ [CLRS09].

To do the reduction, we consider the graph $G_\alpha(W) = (V, E)$ with vertices V and edges E defined as follows:

$$V = \{0, \ldots, |W|\}$$
$$E = \{(i, j) \in V^2 \mid j - i \geq \alpha\}.$$

The weight $c(i,j)$ of the edge $(i,j) \in E$ is defined as

$$c(i,j) := (j-i)e^r_{w_i}.$$

We can see that there is a mapping between path of $G_\alpha(W)$ from node 0 to node $|W|$ and monotone clusterings of minimal cluster size larger α: for such a clustering, with cluster limits $0 = \ell_0 < \ell_1 < \cdots < \ell_{m-1} < \ell_m = |W|$, we consider the path $P_\Gamma = (\ell_0, \ldots, \ell_m)$. This path goes from 0 to $|W|$ and two consecutive nodes in this path have a difference of at least α, as its edges are all in E. Also, the weight of P_Γ is exactly $\gamma_{\min}(\Gamma) + 1$ following from the definition of the edges' weight and from Equation (8):

$$\gamma_{\min}(\Gamma) + 1 = \sum_{i=1}^{m} |G_i| e^r_{\max(i)} = \sum_{i=1}^{m} |\ell_i - \ell_{i-1}| e^r_{w_{\ell_i}} = c(P_\Gamma).$$

So, minimizing the clustering cost is equivalent to finding a shortest path in $G_\alpha(W)$, which is clearly a DAG with $|W|+1$ vertices and $(|W|-\alpha)(|W|-\alpha+1)/2$ edges. We can reduce also the number of edges to consider in the graph to $|E| \leq (|W| - \alpha)\alpha$, reducing the computational complexity of the clustering algorithm to $\Theta((|W| - \alpha)\alpha)$ – the algorithm is now linear in $|W|$. This is described in Appendix D.2.

5.4 Integration to Existing Schemes

The padding algorithm described in the previous sections ensures that, for a given input parameter α, there are clusters, each of size at least α, of keywords with the same (adversarially observed) frequency. This algorithm can be integrated to any SE scheme very simply: the client, during the setup phase, as he knows the upcoming database distribution, is able to compute the clustering and the padding keyword distribution. As explained in Section 5.2, we can suppose that γ is a rational number $\gamma = p/q$. Hence, the client will keep a buffer of q entries to be pushed to the server.

When a new entry has to be inserted, the client inserts it in the buffer. If the buffer is full, he creates p fake entries by sampling p random keywords according to the padding distribution and chooses special document indices marking that these entries are fake. Finally, he pushes the q real and p fake entries to the server, without forgetting to randomly permute them beforehand.

This construction can directly be applied to searchable encryption schemes like TWORAM [GMP16], $\Pi^{\text{dyn}}_{\text{bas}}$ [CJJ+14], SPS [SPS14], $\Sigma o\phi o\varsigma$ [Bos16], and many others to transform them into schemes that are provably secure against adversaries with knowledge of the database, as a corollary of Section 4.2.

Corollary 7. *Let $\mathfrak{C}^{\mathcal{DB}}_k$ the set of database-fixing constraints (*cf. *Section 3.3) with k search queries. Used with the padding algorithm with parameter α, TWORAM is $(\mathcal{L}^{\alpha-pad}_{len}, \mathfrak{C}^{\mathcal{DB}}_k, \alpha^k)$-constrained-adaptive-indistinguishability secure.*

Corollary 8. *Used with the padding algorithm with parameter α, Π^{dyn}_{bas}, SPS, $\Sigma o\phi o\varsigma$, in a result-hiding scenario, are $(L1^{\alpha-pad}_{RH}, \mathfrak{C}^{\mathcal{DB}}, \alpha)$-constrained-adaptive-indistinguishability secure.*

6 Experiments

We implemented the frequency-based clustering algorithm. This allowed us to study the influence of the parameter α on the cost of the clustering, the computational overhead of the padding (the computation of the clustering and of the padding distribution. Finally, our experiments show that the count attack of Cash *et al.* is a lot more powerful than originally assessed in their paper [CGPR15]. In particular, when the adversary knows the database, leaking the keywords co-occurrence is devastating.

6.1 The Performance of the Clustering Algorithm

We implemented the clustering algorithm in Java and ran it on the Enron dataset [enr] for different values of α, in order to see the relation between α and γ. The obtained results are summarized in Figure 1. We

can see that the cost grows roughly as the minimum cluster size, although we can see some discontinuities. These singularities appear when the number of cluster of the optimal clustering changes. Namely, in the experiment with 5000 keywords, the algorithm with $\alpha = 1666$ generates 3 clusters, while for $\alpha = 1667$, it only generates 2.

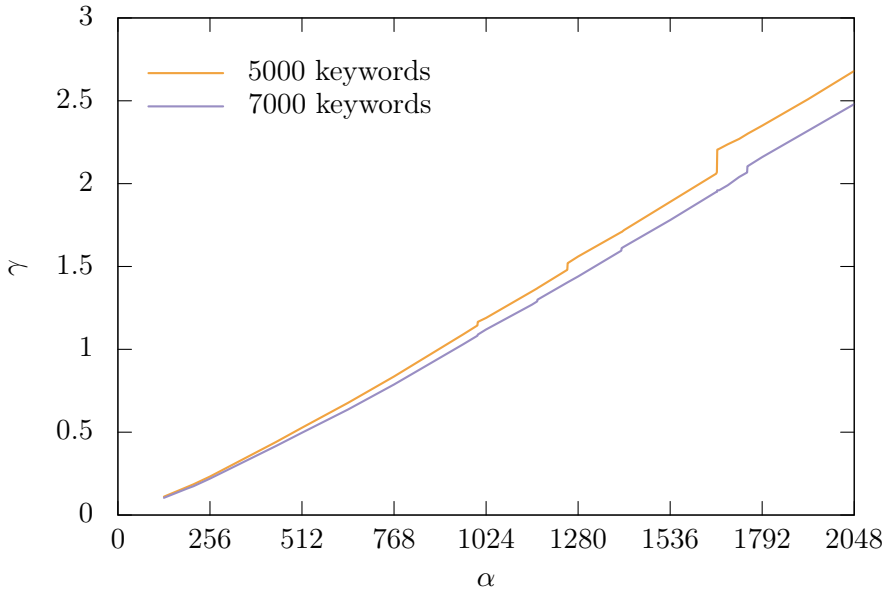

Figure 1 – Storage overhead γ due to the database padding depending on α.

As we would expect, for a given database, the cost of the clustering decreases as the number of indexed keywords increases: the clustering algorithm has more choices to construct clusters and hence will optimize this choice more easily.

Also, the clustering algorithm itself is quite fast: for 20 000 keywords and $\alpha \approx 4750$, the algorithm runs in about 170ms using a quite naïve dynamic programming implementation, on an Intel 4980HQ CPU running at 2.80GHz. In comparison, $\Sigma o\phi o\varsigma$ [Bos16] takes around 1.69ms per insertion, and if we suppose that every document contains more than 100 keywords, we can recompute a clustering every 10 (or 100) inserted documents without reducing noticeably the update throughput and prevent unnoticed changes in the database distribution. For dynamic schemes with higher throughput (*e.g.* $\Pi^{\text{dyn}}_{\text{bas}}$ [CJJ+14]), this can be done every 1000 new document insertion to avoid hindering the update throughput.

6.2 Influence of Secure Padding on the Count Attack

As an experience, we also run the count attack of Cash *et al.* against schemes with complete $L1$ leakage – in particular cooccurrence leakage, not only $L1_{RH}$ – to which we applied our padding algorithm. Remember that our frequency-based clustering provably protects schemes with $L1_{RH}$ leakage. This experiment can be seen as a way to understand what actually is the security loss due to the cooccurrence information.

We executed the count attack on the Enron email database once padded with our algorithm, with different values for α. Experimental results are stated in Table 1. For the experiments, we measured the elapsed CPU time: as the count attack is massively parallelizable, the wall clock time is not a good measurement, and the CPU time is representative of the actual cost for the attacker. We ran the attack using several randomness seeds, and we give both the average and the minimum running time over the choice of seeds. We also compared the attack running time in presence of two different padding strategies: our frequency-based clustering technique and the technique described in [CGPR15], namely the number of entries matching a keyword is padded up to the nearest multiple of an integer n, the *padding factor*.

Table 1 – Experimental results of the count attack with two types of padding. Min and average are taken over at least 20 runs. The Enron email dataset [enr] was used for this experiment, with 5000 keywords and 500 queried keywords. n is the padding factor of the padding technique of [CGPR15].

α	Clustering Runtime (h)		Runtime blowup		Up to Multiple Runtime (h)		n
	Min	Avg.	Min	Avg.	Min	Avg.	
210	0.143	0.876	1.37 ×	3.61 ×	0.105	0.243	100
440	2.15	65.2	5.63 ×	22.5 ×	0.382	2.90	200
710	16.5	1169	13.8 ×	25.6 ×	1.19	22.5	300
960	104	5040	25.1 ×	17.3 ×	4.17	292	400

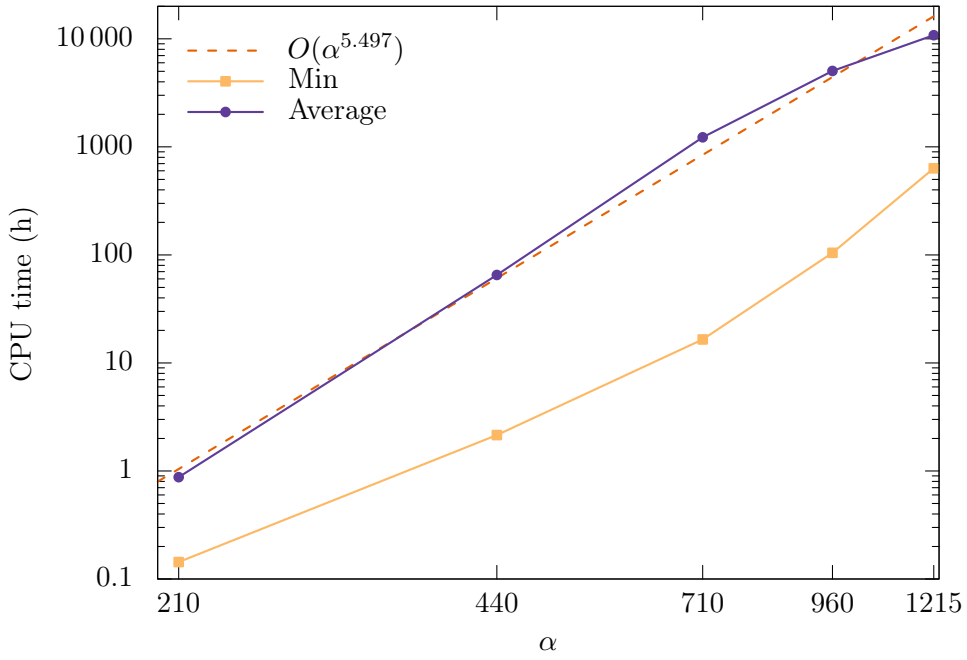

Figure 2 – Running time of the count attack depending on α, in log-log scale.

The first thing to notice is that we always were able to recover (almost) all the queries. The original [CGPR15] paper showed partial reconstruction rate due to a small bug in their original code. The count attack is much more powerful than originally stated in the original paper.

Yet, we can clearly see that the padding strategy influences quite a lot the performance of the count attack. We interpret that as a consequence of the way the count attack works. When there are keywords with unique result count, the attacker is able to recover queries on these keywords and then uses this initial information to recover the other queries by comparing the number cooccurrence between a known and an unknown keyword on one side, and the known cooccurrence information she has from the knowledge of the dataset on the other side. For a still encrypted query q, the adversary will enumerate the candidate keywords matching the same number of documents and reject keywords with incompatible cooccurrence frequency with already recovered queries.

When this is not the case, the attacker cannot leverage the fact that she already recovered queries to compare the cooccurrence. So, instead she guesses the keyword corresponding to the target query among the α and tries to proceed with the query recovery using this guess.

So, in the latter case, we have that the count attack runtime should scale quadratically with α. In practice, using a linear regression, we can see in Figure 2 that the attack scales more like $\alpha^{5.5}$ in average, but we do not explain this discrepancy. In any case, we cannot rely on a sufficiently large value of α to be out of reach of the count attack against the $L1$ leakage. Just padding the database to have many keywords with the frequency, as the algorithm described in this paper does, *cannot* result in reasonable level of security, and there is a clear security gap between schemes leaking only the result length ($\mathcal{L}_{len}$ leakage) and the one also leaking the keywords cooccurrence ($L1$ leakage). We need a different approach.

One is sketched by Islam *et al.* [IKK12, Sections 10-12] who introduced these cooccurrence attacks, and is, in spirit, similar to our clustering idea. Their idea is to pad the database so that there is a partition of the keyword set for which in each set, every keyword matches the same documents, and each set is at least of size α. Unfortunately, their experiments show that the overhead due to this countermeasure is very high ($\gamma \approx 2$ for $\alpha = 2$, and $\gamma \approx 4$ for $\alpha = 3$). Also, doing this in a dynamic setting would require up to $O(|W|^2)$ storage on the client side (in order to store the cooccurrence matrix), which would not scale for large databases.

Finally, we can see the attacks based on cooccurrence as an order 2 version of the frequency-based attack: instead of counting the number of total occurrence of a single keyword w in the database, the adversary counts the number of occurrences of pairs (w_1, w_2). Thus we can imagine higher order attacks using the cooccurrence information of 3 (or more) keywords in the same document. Provably thwarting this kind of attacks would be very costly in general, and we believe that we should actually rely on actual attack performance to evaluate the security of result-revealing SE schemes.

References

[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 16, pp. 1143–1154. ACM Press (Oct. 2016).

[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. Vimercati (eds.), ACM CCS 06, 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 15, 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).

[CJJ+14] Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., and Steiner, M. Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS 2014. The Internet Society (Feb. 2014).

[CLRS09] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. Introduction to Algorithms, Third Edition. The MIT Press, 3rd ed. (2009).

[enr] Enron email dataset. URL https://www.cs.cmu.edu/~./enron/.

[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).

[HR18] Hardy, G.H. and Ramanujan, S. Asymptotic formulæ in combinatory analysis. Proceedings of the London Mathematical Society, vol. 2(1):(1918), pp. 75–115.

[IKK12] Islam, M.S., Kuzu, M., and Kantarcioglu, M. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS 2012. The Internet Society (Feb. 2012).

[KKNO16] Kellaris, G., Kollios, G., Nissim, K., and O'Neill, A. Generic attacks on secure outsourced databases. In: E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, and S. Halevi (eds.), ACM CCS 16, pp. 1329–1340. ACM Press (Oct. 2016).

[KO97] Kushilevitz, E. and Ostrovsky, R. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th FOCS, pp. 364–373. IEEE Computer Society Press (Oct. 1997).

[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 12, pp. 965–976. ACM Press (Oct. 2012).

[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).

[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).

[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: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pp. 707–720 (2016).

A Extension of the Constrained Security Definition

We saw in Section 3.1 that the constraint-based definition did not capture distributional knowledge of the adversary on the database or on the queries. Here we propose a variation of the constrained adaptive indistinguishability security definition that will capture this kind of prior knowledge.

The idea is that the adversary, instead of a history and a constraint, will give to the challenger a pair of history distributions that will be used to sample the challenges. Before giving the actual security definition, we have to define the notion of *acceptable set of distributions*, similarly to the way we defined acceptable constraints in Section 3.1. In the following, for a distribution $\mathcal{D}$ of histories, we define $\mathcal{L}(\mathcal{D})$ as the distribution of $\{L(H) \text{ s.t. } H \xleftarrow{\$} \mathcal{D}\}$

Definition A.1 (Acceptable set of distribution)**.** *A set $\mathfrak{D}$ of history distributions is* $\mathcal{L}$-acceptable *for some leakage $\mathcal{L}$ if, for every efficiently computable distribution $\mathcal{D} \in \mathfrak{D}$, there exists an efficiently computable $\mathcal{D}' \in \mathfrak{D}$ different from $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D})$ and $\mathcal{L}(\mathcal{D}')$ are indistinguishable.*

Definition A.2 (Distribution adaptive indistinguishability for SE)**.** *Let $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ be an SE scheme, λ the security parameter, and $\mathcal{A}$ a stateful algorithm. Let $\mathfrak{D}$ be a set of $\mathcal{L}$-acceptable distribution. We can define the notion of* distribution adaptive indistinguishability *using the following game*

$\mathsf{Ind}^{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{D}}(\lambda)$ *Game:*
$b \xleftarrow{\$} \{0,1\}$
$(\mathcal{DB}^0, \mathcal{DB}^1) \leftarrow \mathcal{A}(1^\lambda)$
Sample $\mathsf{DB}^b \xleftarrow{\$} \mathcal{DB}^b$

$(K, \mathsf{EDB}^b) \leftarrow \mathsf{Setup}(\mathsf{DB}^b)$
$(\mathcal{Q}_1^0, \mathcal{Q}_1^1) \leftarrow \mathcal{A}(\mathsf{EDB}^b)$
Sample $q_1^b \xleftarrow{\$} \mathcal{Q}_1^b$
$\tau_i^b \leftarrow \mathsf{Query}(q_1^b)$
for $i = 2$ ***to*** n ***do***
  $(\mathcal{Q}_i^0, \mathcal{Q}_i^1) \leftarrow \mathcal{A}(\mathcal{Q}_{i-1}^1)$
  Sample $q_i^b \xleftarrow{\$} \mathcal{Q}_i^b$
  $\tau_i^b \leftarrow \mathsf{Query}(q_i^b)$
end for
$b' \leftarrow \mathcal{A}(\tau_n^b)$
if $b = b'$ ***return*** 1, ***otherwise return*** *0*

with the restrictions that, $\mathcal{L}(\mathcal{D}^0)$ *and* $\mathcal{L}(\mathcal{D}^1)$ *are indistinguishable.*

We say that Σ *is* $(\mathcal{L}, \mathfrak{D})$*-distribution-adaptively-indistinguishable if for all polynomial time* $\mathcal{A}$*,*

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathcal{L},\mathfrak{D}}(\lambda) = \mathbb{P}[\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{D}}(\lambda) = 1] - \frac{1}{2} \leq \mathit{negl}(\lambda).$$

If we look at Kellaris *et al.*'s paper [KKNO16], we can see that our security definition actually capture their adversarial setting for their attack on encrypted databases supporting range queries. Indeed the set of distribution to be considered is the set $\mathfrak{D}^{\mathcal{DB}} = \{\mathcal{D}^{\mathsf{DB}} | \mathsf{DB} \in \mathcal{DB}\}$ of distribution $\mathcal{D}^{\mathsf{DB}}$ such that the database distribution is reduced to 'deterministically' output only one element DB, and the queries distribution is uniform over all the range queries.

Kellaris *et al.* show that, for unbounded histories, $\mathfrak{D}^{\mathcal{DB}}$ is not $\mathcal{L}$-acceptable, where $\mathcal{L}$ is the function leaking the number of results of a query [KKNO16, Section 4].

B Proof of Theorem 2

Theorem 2. *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be an SE scheme, and* $\mathfrak{C}$ *a set of constraints. If* Σ *is* $\mathcal{L}$*-adaptive-indistinguishability secure, and* $\mathfrak{C}$ *is* $(\mathcal{L}, k)$*-acceptable, then* Σ *is* $(\mathcal{L}, \mathfrak{C}, k)$*-constrained-adaptive-indistinguishability secure.*

Proof. From Theorem 1, we know that Σ is $(\mathcal{L}, \mathfrak{C})$-constrained-adaptive-indistinguishability secure. So we must show that $(\mathcal{L}, \mathfrak{C})$-constrained-adaptive-indistinguishability implies $(\mathcal{L}, \mathfrak{C}, k)$-constrained-adaptive-indistinguishability. Note that, as $\mathfrak{C}$ is $(\mathcal{L}, k)$-acceptable, the $\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k}$ game is not void.

Let $\mathcal{A}$ be an adversary in the $\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k}$ game. We construct an adversary $\mathcal{B}$ against the $\mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C}}$ game in the following way.

$\mathcal{B}$ starts by randomly picking two integers $k_0, k_1 \in \{0, \ldots, k-1\}$. Then, $\mathcal{B}$ starts $\mathcal{A}$ and receives k databases $(\mathsf{DB}^0, \ldots, \mathsf{DB}^{k-1})$. Upon giving the pair $(\mathsf{DB}^{k_0}, \mathsf{DB}^{k_1})$ to the challenger, $\mathcal{B}$ receives the challenge EDB^* which she forwards to $\mathcal{A}$. Then $\mathcal{A}$ repeatedly outputs k queries $(q_i^0, \ldots, q_i^{k-1})$, $\mathcal{B}$ outputs $(q_i^{k_0}, q_i^{k_1})$ to the game, receives back the transcript τ_i^* and forwards it to $\mathcal{A}$. Eventually $\mathcal{A}$ outputs an integer k'. If $k' = k_0$, $\mathcal{B}$ output $b' = 0$, if $k' = k_1$, $\mathcal{B}$ outputs $b' = 1$, and otherwise outputs 0 with probability $1/2$ and 1 with probability $1/2$.

We know have to evaluate $\mathbb{P}[b = b']$, where b is the random bit picked by the challenger.

$$\begin{aligned}
\mathbb{P}[b = b'] &= \mathbb{P}[b = b' \cap k' \in \{k_0, k_1\}] \\
&\quad + \mathbb{P}[b = b' | k' \notin \{k_0, k_1\}] \cdot \mathbb{P}[k' \notin \{k_0, k_1\}] \\
&= \mathbb{P}[\mathcal{A} \text{ wins the } \mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k} \text{ game}] \\
&\quad + \frac{1}{2}(1 - \mathbb{P}[k' \in \{k_0, k_1\}])
\end{aligned}$$

We can also evaluate $\mathbb{P}[k' \in \{k_0, k_1\}]$:

$$\begin{aligned}\mathbb{P}[k' \in \{k_0, k_1\}] &= \mathbb{P}[k' = k_0] + \mathbb{P}[k' = k_1] \\ &= \frac{1}{2}\left(\mathbb{P}[k' = k_b | b = 0] + \mathbb{P}[k' = k_1]\right) \\ &\quad + \frac{1}{2}\left(\mathbb{P}[k' = k_0] + \mathbb{P}[k' = k_b | b = 1]\right)\end{aligned}$$

As k_0 and k_1 are uniformly picked in $\{0, \ldots, k-1\}$, and as $\mathbb{P}[k' = k_b]$ is the probability that $\mathcal{A}$ wins the 1-out-of-k indistinguishability game, we have

$$\mathbb{P}[k' \in \{k_0, k_1\}] = \mathbb{P}[\mathcal{A} \text{ wins the } \mathsf{Ind}_{\mathsf{SSE},\mathcal{A},\mathcal{L},\mathfrak{C},k} \text{ game}] + \frac{1}{k}$$

Finally, we can conclude that

$$\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{B},\mathcal{L},\mathfrak{C}}(\lambda) = \frac{1}{2}\mathbf{Adv}^{\mathsf{Ind}}_{\mathcal{A},\mathcal{L},\mathfrak{C},k}(\lambda).$$

□

C Proof of Theorem 5

Let us first recall and define the following useful notations. For a cluster G_i, we denote $e^r_{\max(i)}$ (resp. $e^r_{\min(i)}$) the maximum (resp. minimum) value of e^r_w for w in G_i.

$$\begin{aligned}&\delta^+_w := e^r_{\max(i)} - e^r_w, \text{ where } i \text{ is such that } w \in G_i, \\ &\Delta_i := \max_{w \in G_i} \delta^+_w, \text{ and } \Delta := \max_{1 \le i \le m} \Delta_i.\end{aligned}$$

Theorem 5. *The minimum value $\gamma_{\min}(\Gamma)$ of $\gamma_{\min}(\Gamma, \widetilde{e})$ (over all possible frequency choices) is*

$$\gamma_{\min}(\Gamma) = \left(\sum_{i=1}^{m} |G_i| e^r_{\max(i)}\right) - 1 = \sum_{i=1}^{m} \sum_{w \in G_i} \delta^+_w \tag{10}$$

and for $1 \le i \le m$, $\widetilde{e}_i = \frac{e^r_{\max(i)}}{1+\gamma}$. Moreover, we have the following bound for $\gamma_{\min}(\Gamma)$:

$$\Delta \le \gamma_{\min}(\Gamma) \le (|W| - m)\Delta. \tag{11}$$

Proof. Using the above notations, $\gamma_{\min}(\Gamma, \widetilde{e})$ can be re-written as

$$\gamma_{\min}(\Gamma, \widetilde{e}) = \max_{1 \le i \le m} \left\{ \frac{e^r_{\max(i)}}{\widetilde{e}_i}, \frac{1 - e^r_{\min(i)}}{1 - \widetilde{e}_i} \right\} - 1. \tag{12}$$

From this equation, we can easily derive a lower bound for γ: for each cluster, $\gamma_i(\Gamma, \widetilde{e}_i) = \max\left\{ \frac{e^r_{\max(i)}}{\widetilde{e}_i}, \frac{1-e^r_{\min(i)}}{1-\widetilde{e}_i} \right\} - 1$ is minimum when

$$\frac{e^r_{\max(i)}}{\widetilde{e}_i} = \frac{1 - e^r_{\min(i)}}{1 - \widetilde{e}_i} \Leftrightarrow \widetilde{e}_i = \widetilde{e}^*_i \text{ where } \widetilde{e}^*_i := \frac{e^r_{\max(i)}}{1 + e^r_{\max(i)} - e^r_{\min(i)}}.$$

and, as a consequence,

$$\gamma_{\min}(\Gamma, \widetilde{e}) \ge \gamma_0 = \max_{1 \le i \le m} \left\{ e^r_{\max(i)} - e^r_{\min(i)} \right\}. \tag{13}$$

A very important thing to notice is that, WLOG, we can suppose that either $\widetilde{e}_i \geq \widetilde{e}_i^*$ for all i, or $\widetilde{e}_i \leq \widetilde{e}_i^*$ for all i, when the optimal cost is reached. Suppose this is not the case, and that for the majority of the clusters $\widetilde{e}_i \geq \widetilde{e}_i^*$, and take j such that $\widetilde{e}_j < \widetilde{e}_j^*$. Then, by decreasing $\widetilde{e}_i > \widetilde{e}_i^*$ and increasing $\widetilde{e}_j < \widetilde{e}_j^*$ such that $|G_i|\widetilde{e}_i + |G_j|\widetilde{e}_j$ remains constant, we will decrease both γ_i and γ_j. Then if the maximum of γ_i's is reached for cluster G_i, or G_j (or both), this contradicts the cost minimality (the same argument holds when the optimal cost is reached for several clusters at the same time).

Before going on, we prove an helpful technical lemma.

Lemma 9. *The following inequalities hold:*

$$1 + \Delta \leq \sum_{i=1}^{m} |G_i| e^r_{\max(i)} \leq 1 + (|W| - m)\Delta \tag{14}$$

Proof. By definition of δ_w^+,

$$\begin{aligned}
& \sum_{i=1}^{m} \sum_{w \in G_i} e^r_w = \sum_{i=1}^{m} \sum_{w \in G_i} (e^r_{\max(i)} - \delta_w^+) \\
\Leftrightarrow \quad & \sum_{i=1}^{m} |G_i| e^r_{\max(i)} = 1 + \sum_{i=1}^{m} \sum_{w \in G_i} \delta_w^+ \\
\Rightarrow \quad & 1 + \Delta \leq \sum_{i=1}^{m} |G_i| e^r_{\max(i)} \leq 1 + (|W| - m)\Delta.
\end{aligned}$$

The $|W| - m$ factor instead of $|W|$ in the last inequality comes from the fact that in each cluster, there is at least one keyword w such that $\delta_w^+ = 0$. □

From this lemma, we can show that $\sum_{i=1}^{m} |G_i| \widetilde{e}_i^* \geq 1$:

$$\sum_{i=1}^{m} |G_i| \widetilde{e}_i^* = \sum_{i=1}^{m} |G_i| \frac{e^r_{\max(i)}}{1 + \Delta_i} \geq \sum_{i=1}^{m} |G_i| \frac{e^r_{\max(i)}}{1 + \Delta} \geq 1$$

However, we are not guaranteed that $\sum_{i=1}^{m} |G_i| \widetilde{e}_i^* = 1$, and we cannot conclude that the optimal cost will be $\gamma_{\min}$.

Suppose $\sum_{i=1}^{m} |G_i| \widetilde{e}_i^* > 1$. We need to decrease some values $\widetilde{e}_i$ to satisfy constraint (5). It is 'free' – it does not increase the overall cost – to do so for clusters such that $\gamma_i < \gamma_{\min}$, and for such clusters the minimum value that $\widetilde{e}_i$ can take is $\widetilde{e}_i^{\lim}$ such that

$$\frac{e^r_{\max(i)}}{\widetilde{e}_i^{\lim}} = 1 + \gamma_{\min} \Leftrightarrow \widetilde{e}_i^{\lim} = \frac{e^r_{\max(i)}}{1 + \gamma_{\min}} = \frac{e^r_{\max(i)}}{1 + \Delta}.$$

Indeed as, $\widetilde{e}_i^{\lim} \leq \widetilde{e}_i^*$, $\frac{e^r_{\max(i)}}{\widetilde{e}_i^{\lim}}$ is larger than $\frac{1 - e^r_{\min(i)}}{1 - \widetilde{e}_i^{\lim}}$.

Again, using Equation (14), we can show that $\sum_{i=1}^{m} |G_i| \widetilde{e}_i^{\lim} \geq 1$. Thus we will have to chose $\widetilde{e}_i < \widetilde{e}_i^{\lim}$ for some clusters. Also, for the optimal expected frequencies (the ones inducing the smallest cost), we will have $\gamma_i = \gamma$ for all clusters: if this is not the case, and that there is a cluster such that $\gamma_j < \gamma$, we can decrease $\widetilde{e}_j$ (and thus increase γ_j) while increasing the other expected frequencies (and thus decreasing γ). Hence, from the definition of γ_i, we have that

$$\gamma = \frac{e^r_{\max(i)}}{\widetilde{e}_i} - 1 \Leftrightarrow \widetilde{e}_i = \frac{e^r_{\max(i)}}{1 + \gamma}$$

and as $\widetilde{e}$ satisfies the constraint (5), we get

$$\sum_{i=1}^{m} |G_i| \frac{e^r_{\max(i)}}{1+\gamma} = 1$$

$$\Leftrightarrow \quad \gamma = \left(\sum_{i=1}^{m} |G_i| e^r_{\max(i)} \right) - 1. \tag{15}$$

Finally, in Lemma 9, we showed that

$$\sum_{i=1}^{m} |G_i| e^r_{\max(i)} = 1 + \sum_{i=1}^{m} \sum_{w \in G_i} \delta^+_w.$$

□

D Proofs and Algorithms for the Frequency-Based Clustering

D.1 Proof of Proposition 6

Proposition 6. *Let Γ be a non-monotone clustering with clusters of size at least α. Then there exists a clustering Γ' with clusters of size at least α, $\gamma_{\min}(\Gamma) \geq \gamma_{\min}(\Gamma')$ and Γ' is monotone.*

Proof. As Γ is not monotone, there are two clusters C_i and C_j such that $e^r_{\max(i)} > e^r_{\max(j)} > e^r_{\min(i)}$. Let $w \in C_i$ (resp. $w' \in C_j$) such that $e^r_w = e^r_{\min(i)}$ (resp. $e^r_{w'} = e^r_{\max(j)}$), and Γ' be the clustering obtained by exchanging w and w' in C_i and C_j. If there is only one $w' \in C_j$ reaching the maximum, The maximum expected frequency of the new cluster C'_j will decrease or stay identical if there is more than one $w' \in C_j$ reaching the maximum, while the $e'^r_{\max(i)}$ will not change. As a consequence, $\gamma(\Gamma) \geq \gamma(\Gamma')$.

We can iterate this algorithm until C_i and C_j satisfy the monotonicity condition, *i.e.* with $e^r_{\max(j)} \leq e^r_{\min(i)}$. Finally, by repeating this procedure over pairs of clusters, as in a sorting algorithm, we end up constructing a monotone clustering whose cost is less than the original one, while its min-quality remains unchanged (because the size of the clusters did not change). □

D.2 Reducing the Complexity of the Clustering Graph

In Section 5.3, we saw how to find the clustering with minimum cost using a shortest path algorithms for DAGs. The graph was defined as

$$V = \{0, \dots, |W|\}$$
$$E = \{(i,j) \in V^2 \mid j - i \geq \alpha\}.$$

and, the number of edges to consider was $|E| = \frac{(|W|-\alpha)(|W|-\alpha+1)}{2}$.

Let Γ be a clustering with minimum cost, whose clusters are larger than α, and such that no cluster contains more than 2α keywords. We can construct a clustering Γ' by splitting a cluster of size larger than 2α in two clusters each of size larger than α. Γ' will have a cost less than Γ (because the maximum expected frequency of the newly obtained cluster will be less than the one of the old split cluster). Without loss of generality, we can suppose that minimal cost clusterings have no cluster with more than 2α elements.

This implies that we can reduce the set of edges of G to

$$E = \{(i,j) \in V^2 \mid j - i \geq \alpha \text{ and } j - i < \alpha\}.$$

The number of edges is then $|E| = (|W| - 2\alpha)\alpha + \frac{\alpha(\alpha-1)}{2}$ (α incoming edges for all nodes, except the first α ones – no incoming edge – and the nodes $i \in [\alpha, 2\alpha - 1]$ which have $i - \alpha$ incoming edges).

Finally, note that as described higher, the graph G is already topologically sorted. Hence, finding the shortest path in G can be very easily done using dynamic programming.