Revisiting Leakage Abuse Attacks

Laura Blackstone* Brown University
Seny Kamara† Brown University
Tarik Moataz‡ Aroki Systems

Abstract

Encrypted search algorithms (ESA) are cryptographic algorithms that support search over encrypted data. ESAs can be designed with various primitives including searchable/structured symmetric encryption (SSE/STE) and oblivious RAM (ORAM). Leakage abuse attacks attempt to recover client queries *using knowledge of the client's data*. An important parameter for any leakage-abuse attack is its *known-data rate*; that is, the fraction of client data that must be known to the adversary.

In this work, we revisit leakage abuse attacks in several ways. We first highlight some practical limitations and assumptions underlying the well-known IKK (Islam et al. *NDSS '12*) and Count (Cash et al., *CCS '15*) attacks. We then design four new leakage-abuse attacks that rely on much weaker assumptions. Three of these attacks are *volumetric* in the sense that they only exploit leakage related to document sizes. In particular, this means that they work not only on SSE/STE-based ESAs but also against ORAM-based solutions. We also introduce two volumetric *injection* attack which use adversarial file additions to recover queries even from ORAM-based solutions. As far as we know, these are the first attacks of their kind.

We evaluated all our attacks empirically and considered many experimental settings including different data collections, query selectivities, known-data rates, query space size and composition. From our experiments, we observed that the only setting that resulted in reasonable recovery rates under practical assumptions was the case of high-selectivity queries with a leakage profile that includes the response identity pattern (i.e., the identifiers of the matching documents) and the volume pattern (i.e., the size of the matching documents). All other attack scenarios either failed or relied on unrealistic assumptions (e.g., very high known-data rates). For this specific setting, we propose several suggestions and countermeasures including the use of schemes like PBS (Kamara et al, *CRYPTO '18*), VLH/AVLH (Kamara and Moataz, *Eurocrypt '19*), or the use of padding techniques like the ones recently proposed by Bost and Fouque (Bost and Fouque, *IACR ePrint 2017/1060*).

*laura_blackstone@alumni.brown.edu.
†seny@brown.edu.
‡tarik@aroki.com. Work done while at Brown University.

Contents

1 Introduction

The area of encrypted search is concerned with the design and analysis of cryptographic techniques to search over encrypted data. There are many ways to design encrypted search algorithms (ESA) including using fully-homomorphic encryption (FHE) [28], oblivious RAM (ORAM) [31], functional encryption [9], property-preserving encryption [2, 7, 8] and searchable/structured symmetric encryption (SSE/STE) [68, 19, 18]. All these approaches achieve different tradeoffs between leakage, expressiveness and efficiency.

At a high level, a static ESA is composed of two algorithms: a setup algorithm and a search algorithm. Setup encrypts a data collection in such a way that it can later be queried using the search algorithm. The most basic form of search is single-keyword search which, given a keyword w, returns the documents in the collection that contain w. If the solution is dynamic, there is an additional update algorithm to modify the encrypted data collection. In this work, we will focus on SSE/STE- and ORAM-based ESAs, which we sometimes refer to as *structured ESAs* and *oblivious ESAs*, respectively.

Leakage. While it is possible to search on encrypted data with essentially no leakage with time and communication that is linear in the size of the document collection, this is however not a viable approach for datasets of practical interest. Because of this, all known ORAM-, STE- and PPE-based ESAs leak some information. This leakage comes in two forms: setup leakage, which is revealed at setup time by the encrypted dataset, and query leakage, which is revealed at query time by the encrypted dataset and the query operation. To better understand the real-world impact of this leakage, an important research direction in encrypted search has been to design leakage attacks. This line of work was initiated by Islam, Kuzu and Kantarcioglu in [38] in the context of SSE and was expanded to PPE by Naveed, Kamara and Wright [56] and to ORAM by Kellaris, Kollios, Nissim and O'Neill [46]. Since then, several works have further explored leakage attacks including [13, 74, 48, 33] in the SSE setting and [13, 22, 34] in the PPE setting.

Leakage attacks. Leakage attacks come in different forms depending on the leakage profiles they exploit, the adversarial models in which they work, the information they recover, the auxiliary information they need and the assumptions they rely on. We can categorize attacks along the following dimensions:

- adversarial model: *snapshot* attacks only require access to the encrypted document collection; *persistent* attacks require access to the encrypted data collection and to the transcripts of the query operations.
- target: *data-recovery* attacks recover information about the data collection whereas *query-recovery* attacks recover data about queries;
- auxiliary data: *sampled-data* attacks require a sample from a distribution that is close (e.g., in statistical distance) to the distribution of the data collection; *known-query* attacks require knowledge of a subset of the queries; *known-data* attacks require knowledge of a subset of the data collection. An important parameter for known-data attacks is the *known-data rate* which is defined as the fraction δ of documents in the client's data collection that are known to the adversary.

- passive or active: passive attacks do not require the adversary to choose any part of the client's data; active, or *chosen-data* attacks, require the adversary to choose some of the data in the data collection.

In this work, we focus on the persistent model since it has recently been shown that solutions with little to no leakage can be achieved in the snapshot setting using both PPE [50] and STE [3] (this doesn't take into account possible systems-level pitfalls as pointed out in [32]). We recall that sampled-data attacks are commonly referred to as *inference* attacks, that known-data attacks are commonly referred to as *leakage abuse* attacks and that chosen-data attacks are commonly referred to as *injection* attacks.

Leakage profiles. Of course, an important characteristic of any leakage attack is what kind of leakage it exploits. For example, the IKK attack exploits the *co-occurrence pattern* which reveals, for every pair of queries, the number of times they appear in the same document. The Count attack rely on the co-occurrence pattern and the *response length pattern*; the latter of which reveals, for every query, the number of documents that contain it. The injection attacks of [74] exploit the *response identity* (also known as the *access pattern*) which reveals, for every query, the identifiers of the documents that contain it. Note that the response identity reveals the response length and the co-occurrence pattern so the IKK and Count attack can apply to any construction that leaks the response identity. An important class of leakage patterns for our purposes will be what we call *volumetric* patterns. By this we mean any leakage pattern that reveals the size of documents. Here, we will focus specifically on the *volume pattern* which reveals, for each query, the volumes of the documents that contain it; and the *total volume pattern* which reveals, for each query, the sum of the volumes of the documents that contain it.

Known-data attacks against structured ESAs. Known-data attacks were introduced by Cash, Grubbs, Perry and Ristenpart in [13]. In that work, they described several attacks against both SSE/STE- and PPE-based ESAs. They also introduced injection (or *chosen-data*) attacks against PPE-based ESAs. Injection attacks were later demonstrated against structured ESAs by Zhang, Katz and Papamanthou [74].

The IKK attack was first described in [38] as an *inference* attack that exploits the co-occurrence pattern. [38] reported high recovery rates but the experiments conducted had several methodological flaws. The most salient ones were that: (1) they were run on a small query space (of size 2500 out of a total of 77000 after stemming and removing stop words); and (2) the training and test data collections were not independent. Motivated by this, Cash et al. re-evaluated the IKK attack with independent testing and training data and found that IKK could not recover any queries. IKK was then re-evaluated as a known-data attack and it was found that it could achieve reasonable recovery rates if it was given 95% or more of the client's data. Effectively, [13] showed that IKK failed as an inference attack but worked as a known-data attack when $\delta \geq .95$. [13] then introduced a new attack called the Count attack. This attack relies on co-occurrence and response length leakage and was shown to perform better than the IKK attack. Recently, the ePrint version of [16] was updated to include a new attack that performs better than the one originally published in [13]. Throughout this work, we will refer to the first Count attack as Count v.1 and to the new attack as Count v.2.

Discussion and overview of our contributions. The IKK and Count attacks have received a lot of attention and are commonly used to draw conclusions about various ESAs. As an example, they are often cited as a reason to prefer oblivious ESAs over structured ESAs [70, 71, 73, 52, 20, 63, 27, 25, 29, 64, 58, 53, 51, 66, 65, 72, 26, 5, 37]. The results in this work underscore that the study of and, especially, the *interpretation* of known-data attacks is more nuanced.

To address this, we present in Section 3 several new known-data attacks that do not have these limitations and that achieve higher recovery rates for much lower known-data rates. What is perhaps surprising about our attacks is that they work not only against structured ESAs but also against *oblivious* ESAs which contradicts the conventional wisdom that ORAM-based search is resistant to leakage-abuse attacks.

Another contribution of our work is that we demonstrate, for the first time, that injection attacks apply not only to PPE-based ESAs [13] and structured ESAs [74] but also to oblivious ESAs. In particular, we describe in Section 4 two volumetric injection attacks, which contradicts the conventional wisdom that ORAM-based search is resistant to injection attacks.

In Section 5, we report on a thorough empirical analysis of our attacks (the details are in Appendix 5.1). Specifically, we evaluate the attacks in a host of different settings including different keyword selectivities, query space compositions, query space sizes and datasets. This extensive evaluation shows that the success rates of known-data attacks is very sensitive to various parameters that were not considered in previous work. This highlights the importance of common but often implicit assumptions made in the leakage attack literature.

Finally, in Section 7 we propose several countermeasures to mitigate all known-data attacks, including our own.

1.1 Theory vs. Practice of Known-Data Attacks

Here, we revisit the state-of-the art in known-data attacks highlighting some of the practical limitations and assumptions of the currently-known attacks.

Reliance on co-occurrence. Recall that all IKK, Count v.1 and Count v.2 attacks all rely on the co-occurrence pattern. This particular leakage pattern, however, can be hidden using standard SSE/STE techniques. In fact, we describe a construction in Appendix A, OPQ, that does not reveal the co-occurrence pattern. As far as we know, this construction has not appeared in previous work and may be of independent interest.

High known-data rates. The experimental results in [13] (cf., Figure 6) and [16] (cf., Figure 6) show that the IKK, Count v.1 and Count v.2 attacks achieve non-trivial recovery rates only with very high known-data rates: IKK needs to know 70% of the data to recover 5% of the queries; Count v.1 needs to know 80% of the data to recover 40% of the queries (note that knowing even up to 75% of the data results in a query recovery rate of 0%); and Count v.2 needs to know 75% of the data to recover 40% of the queries. Given how high these known-data rates are, it is not clear whether these attacks should be considered practical. An instructive question to ask here is how exactly an adversary could, in practice, get up to, say 75%, of a client's data? Recall that in the encrypted search setting the client encrypts its data and *outsources* it to an untrusted server. In particular, this means the client deletes the data from its system after setup which leaves the only copy on the server and in encrypted form. In such a setting, there are a few scenarios in which an

adversary could recover 75% of the client's data. One is that the client, for some reason, chooses to encrypt public data which the adversary later recovers. A second is that the client decides to release a large percentage of its data (after downloading and decrypting it from the server). A third is that the client queries its data and over time caches enough of the results to amount to a large percentage of the data. At this stage, a data breach occurs on the client and the cached data is revealed to the adversary. The first two scenarios are relatively contrived and have more to do with a misuse of the primitive: in such settings one should use private information retrieval. The third scenario is perhaps less contrived but caching 75% of one's data locally seems to defeat the purpose of outsourced storage. Indeed, if a client is willing to store 75% of its data locally then they might as well do search locally on the 75% and use encrypted search only for the remaining 25%.

High- vs. low-selectivity keywords. An important consideration when evaluating a query-recovery attack is how exactly client queries are chosen. Most leakage attack papers make implicit assumptions about this but the query distribution has a large impact on accuracy rates. For example, the experiments in [13] assume the client queries high-selectivity keywords. It is not clear, however, if this is realistic. In fact, we ran the IKK and Count attacks on low-selectivity keywords over the Enron dataset and neither attack worked; even when the adversary had a complete knowledge of the client's data. More precisely, the IKK and Count attacks had recovery rate 0 even when $\delta = 1$ for keyword selectivities lower than 20.

Known queries. The IKK and Count v.1 attacks are described in [13] as known-data attacks that do not require knowledge of any client queries. While it is true that these attacks can achieve high recovery rates without knowledge of client queries, this only holds if the adversary has complete knowledge of the data; that is, if $\delta = 1$. If the adversary has less than full knowledge (i.e., $\delta < 1$) and no knowledge of any queries, then Count v.1 does not work. [1]

Theoretical vs. practical attacks. In cryptanalysis it is common to distinguish between theoretical attacks and practical attacks. [2] The former are attacks that work in strong adversarial models; often relying on assumptions about the adversary's capabilities which rarely occur in practice. Examples include related-key attacks where the adversary is allowed to make chosen-plaintext and chosen-ciphertext queries under related keys (e.g., through modification of keys). Another is the known-key model where the adversary is assumed to know the key and its goal is to distinguish ciphertexts from random. It is our belief that the IKK and Count attacks are mostly of theoretical interest since they rely on strong assumptions like known-queries and high known-data rates and have only been shown to work on high-selectivity keywords.

Should we discount theoretical attacks? Though these attacks are of theoretical interest it does not mean we should dismiss them. Even theoretical cryptanalytic results have something to

[1] This may seem to contradict the results presented in [13] but the experimental evaluation of Count v.1 presented in Figure 6 of [13] was incomplete. For $\delta = 1$ (i.e., complete knowledge) the attack does not need knowledge of queries but for $\delta < 1$ it does. We confirmed this with the authors who updated their manuscript with the new Count v.2 attack which does not require knowledge of queries.

[2] We highly recommend the paper of Aumasson [6] for an insightful discussion of these issues.

Attack	Type			Leakage pattern	Known queries	δ for high selectivity	δ for p-low selectivity	δ for low selectivity
	Sampled	Known	Injection					
IKK [38]	✗	✓	✗	co	✓	$\geq$95%	⊙	⊙
Count [13]	✗	✓	✗	co, rlen	✓	$\geq$80%	⊙	⊙
Zhang et al. [74]	✗	✗	✓	rid	✗	–	–	–
Subgraph$^{\mathsf{ID}}$	✗	✓	✗	rid	✗	$\geq$5%	$\geq$50%	$\geq$60%
Subgraph$^{\mathsf{VL}}$	✗	✓	✗	vol	✗	$\geq$5%	$\geq$50%	⊘
VolAn	✗	✓	✗	tvol	✗	$\geq$85%	$\geq$85%	⊘
SelVolAn	✗	✓	✗	tvol, rlen	✗	$\geq$80%	$\geq$85%	⊘
Decoding & Binary	✗	✗	✓	tvol	✗	–	–	–

Table 1: Comparison of existing leakage abuse (known-data) and injection (chosen-data) attacks. The last three columns give the known-data rate δ needed for a recovery rate of at least 20% against low-, pseudo-low- and high-selectivity keywords, respectively. ⊙ means the experiment was not conducted in previous work. ⊘ means that even a known-data rate of $\delta = 1$ does not achieve at least 20% recovery rate. All experiments were based on the Enron dataset [62] with 150 queries and a keyword space of 500 keywords.

teach us about the security of our constructions. It is important, however, to be clear and explicit about the limitations of cryptanalytic results.

1.2 Our Attacks

Motivated by the discussion above, we revisit known-data attacks (i.e., leakage abuse attacks) against SSE. We introduce four new attacks and two new injection attacks in Sections 3 and 4. We also perform a thorough evaluation which we report on in Section 5 and provide in detail in Appendix 5.1. Our attacks achieve high recovery rates with low known-data rates and do not rely on known queries. We summarize the characteristics of our attacks in Table 1. Most surprisingly, all but one of our attacks are volumetric and, therefore, apply not only to SSE/STE-based solutions but also to ORAM-based constructions. As far as we know, these are the first known-data and chosen-data attacks against ORAM. We now summarize our attacks and their performance:

- Volume analysis (VolAn): a known-data attack that exploits the *total volume pattern*. It has high recovery rates when $\delta \geq .8$ and the client queries keywords with high-selectivity (i.e., 10-13) or pseudo-low-selectivity (i.e., 1-2).
- Selective volume analysis (SelVolAn): an extension of volume analysis that relies on the *total volume* and *response length* patterns. This attack has slightly higher recovery rates under the same conditions as volume analysis.
- Subgraph attacks: a framework to design known-data attacks against *atomic* leakage patterns, i.e., leakage pattern that reveals information about each matching document. We give two concrete instantiations of our framework. The first is Subgraph$^{\mathsf{ID}}$ which exploits the *response identity* and the second is Subgraph$^{\mathsf{VL}}$ which exploits the *volume pattern*. Both attacks achieve high recovery rates with very low known-data rates (i.e., with δ as low as .05) when the client queries high-selectivity keywords. The recovery rate drops significantly and reaches 0% when the client queries keywords with low selectivity (i.e., 1-2) and pseudo-low selectivity.
- the Decoding attack (Decoding): an injection attack that exploits the *total volume pattern*. This attack always recovers its target query if the adversary can inject between 4 to 16KBytes depending on the query's selectivity.

- the Binary Search attack (Binary): an injection attack that also exploits the *total volume pattern*. The attack requires logarithmic number of (adaptive) injections. The attack recovers its target query if the adversary can inject around 8KBytes.

1.3 Countermeasures

We propose several countermeasures and guidelines against both our new attacks and previously-known attacks.

As discussed earlier, our empirical evaluation found only a single practical setting where our attacks are successful: querying high-selectivity keywords using a scheme that leaks both the response identity and the volume patterns. The simplest countermeasure to this is to use a scheme that does not leak these patterns like the PBS construction of Kamara, Moataz and Ohrimenko [45]. In Section 7, we demonstrate empirically that PBS is resistant to all known-data attacks (even ours) as long as the client makes at least 4 queries. Specifically, we show that under this condition, the best possible attack has recovery rate 0.02% recovery rate.

To mitigate theoretical attacks, like the IKK or Count attack which require high known-data rates and exploit the co-occurrence pattern, we design a new scheme called OPQ that does not leak this pattern. We also point out that, in [12], Bost and Fouque introduced padding techniques that efficiently mitigate these attacks.

Finally, to protect against purely volumetric attacks one can use the recent constructions of Kamara and Moataz [44] which are volume-hiding.

1.4 Related Work

We already mentioned work on leakage attacks and, more specifically, on leakage abuse attacks so we focus here on work related to SSE/STE- and ORAM-based ESAs.

SSE/STE. SSE was introduced by Song, Wagner and Perrig [68]. Curtmola, Garay, Kamara and Ostrovsky formalized SSE in [19] and described the first sub-linear and optimal-time constructions. STE was introduced by Chase and Kamara in [18] as a generalization of SSE. Many works have explored various aspects of SSE including dynamism [30, 42, 41, 69, 10, 17, 11, 23], locality [17, 15, 4, 55, 21], expressiveness [18, 14, 59, 24, 54, 40, 43], multiple clients [19, 39, 60, 35], and leakage [27, 45, 44, 3].

ORAM. Goldreich and Ostrovsky introduced ORAM in [31], where they described constructions with amortized square-root and polylog overheads. Shi, Chan, Stefanov and Li introduced tree-based ORAMs which achieved worst-case polylog overhead in [67]. Since then, many works have improved ORAM along many dimensions including communication complexity, round complexity, client and server storage [47, 70, 63, 27].

2 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. $[n]$ is the set of integers $\{1,\ldots,n\}$, and $2^{[n]}$ is the corresponding power set. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{q}$ of n elements, we

refer to its ith element as q_i or $\mathbf{q}[i]$. If S is a set then $\#S$ refers to its cardinality. If s is a string then $|s|_2$ refers to its bit length. Throughout, k will denote the security parameter.

The word RAM. Our model of computation is the word RAM. In this model, we assume memory holds an infinite number of w-bit words and that arithmetic, logic, read and write operations can all be done in $O(1)$ time. We denote by $|x|_{\mathsf{w}}$ the word-length of an item x; that is, $|x|_{\mathsf{w}} = |x|_2/\mathsf{w}$. Here, we assume that $\mathsf{w} = \Omega(\log k)$.

Document collections. Let $\mathbb{W}$ be a keyword space. A document collection $\mathbf{D} = (\mathrm{D}_1, \ldots, \mathrm{D}_n)$ over $\mathbb{W}$ consists of n documents, each of which is a subset of $\mathbb{W}$. We denote by $\mathbb{D}$ the space of all document collections over $\mathbb{W}$. We assume each document $\mathrm{D} \in \mathbf{D}$ has a unique identifier that is independent from its contents. For ease of exposition, we assume these identifiers are the integers 1 through n and that they are assigned to documents uniformly at random. For ease of exposition, it will be helpful to consider the following functions. The identifier function $\mathsf{id} : \mathbf{D} \to [n]$ that maps a document D_i to its identifier i. The function $\mathbf{D} : \mathbb{W} \to 2^{\mathbf{D}}$ that maps a keyword w to the documents that contain it. The identifiers function $\mathsf{ids} : \mathbb{W} \to 2^{[n]}$ that maps a keyword w to the identifiers of the documents that contain it. We refer to the word-length of a document $|\mathrm{D}|_{\mathsf{w}}$ as its *volume*.

Throughout this work, we denote the adversary's known dataset by $\widetilde{\mathbf{D}}$ and assume it is a subset of the client's document collection $\mathbf{D}$ chosen uniformly at random.

Structured ESAs. A static structured encrypted search algorithm $\mathsf{ESA} = (\mathsf{Setup}, \mathsf{Search})$ consists of two efficient algorithms. Setup takes as input a security parameter 1^k and a document collection $\mathbf{D} = (\mathrm{D}_1, \ldots, \mathrm{D}_n)$ and outputs a secret key K and an encrypted data collection $(\mathsf{EDB}, \mathrm{ct}_1, \ldots, \mathrm{ct}_n)$. Search is a two-party protocol between a client and a server. The client inputs its secret key K and a keyword w and the server inputs an encrypted collection $(\mathsf{EDB}, \mathrm{ct}_1, \ldots, \mathrm{ct}_n)$. The client receives a set of encrypted documents $\{\mathrm{ct}_i\}_{i \in \mathsf{ids}(w)}$ and the server receives $\perp$. Structured ESAs are constructed using structured encryption (STE) [18] and, in particular, using a multi-map or dictionary encryption schemes. We describe some examples in Appendix A.

Oblivious ESAs. ESAs can also be designed using ORAM. The simplest approach is similar to the structured ESA construction described above but where EDB is replaced with an oblivious RAM ORAM that stores a search structure (e.g., a multi-map). The Search algorithm then executes the structure's query algorithm and replaces each read operation with a call to the ORAM's access protocol. We describe some examples of oblivious ESA constructions in Appendix A.

Modeling leakage. Each ESA operation is associated with leakage which itself can be composed of multiple *leakage patterns*. The collection of all these leakage patterns forms the scheme's *leakage profile*. Leakage patterns are (families of) functions over the various spaces associated with the underlying data collection. For concreteness, we recall some well-known leakage patterns:

- the *query equality pattern* is the function family $\mathsf{qeq} = \{\mathsf{qeq}_{k,t}\}_{k,t \in \mathbb{N}}$ with $\mathsf{qeq}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \{0,1\}^{t \times t}$ such that $\mathsf{qeq}_{k,t}(\mathbf{D}, w_1, \ldots, w_t) = M$, where M is a binary $t \times t$ matrix such that $M[i,j] = 1$ if $w_i = w_j$ and $M[i,j] = 0$ if $w_i \neq w_j$. The query equality pattern is referred to as the search pattern in the SSE literature;

- the *identifier pattern* is the function family $\mathsf{rid} = \{\mathsf{rid}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{rid}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to [2^{[n]}]^t$ such that $\mathsf{rid}_{k,t}\big(\mathbf{D}, w_1, \ldots, w_t\big) = (\mathsf{ids}(w_1), \ldots, \mathsf{ids}(w_t))$. The identifier pattern is referred to as the access pattern in the SSE literature;

- the *response length pattern* is the function family $\mathsf{rlen} = \{\mathsf{rlen}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{rlen}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}$ such that $\mathsf{rlen}_{k,t}(\mathbf{D}, w_1, \ldots, w_t) = \big(\#\mathbf{D}(w_1), \ldots, \#\mathbf{D}(w_t)\big)$;

- the *volume pattern* is the function family $\mathsf{vol} = \{\mathsf{vol}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{vol}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}^t$ such that
$$\mathsf{vol}_{k,t}(\mathbf{D}, w_1, \ldots, w_t) = \left(\left(|\mathrm{D}|_{\mathsf{w}}\right)_{\mathrm{D}\in\mathbf{D}(w_1)}, \ldots, \left(|\mathrm{D}|_{\mathsf{w}}\right)_{\mathrm{D}\in\mathbf{D}(w_t)}\right).$$

- the *total volume pattern* is the function family $\mathsf{tvol} = \{\mathsf{tvol}_{k,t}\}_{k,t\in\mathbb{N}}$ with $\mathsf{tvol}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \to \mathbb{N}^t$ such that
$$\mathsf{tvol}_{k,t}(\mathbf{D}, w_1, \ldots, w_t) = \left(\sum_{\mathrm{D}\in\mathbf{D}(w_1)} |\mathrm{D}|_{\mathsf{w}}, \ldots, \sum_{\mathrm{D}\in\mathbf{D}(w_t)} |\mathrm{D}|_{\mathsf{w}}\right).$$

Each operation of an ESA (e.g., setup, query) generates leakage which is the *direct product* of one or more leakage patterns.

We say that a leakage pattern is *atomic* if it reveals information about each individual matching document. For example, rid and vol are atomic whereas tvol is not. We say that a leakage pattern is *volumetric* if it reveals size information about the matching documents. For example, vol and tvol are volumetric. Attacks that rely on volumetric leakage are particularly interesting because they apply to almost all constructions, including ORAM-based constructions.[3]

Security. In encrypted search we consider *persistent* and *snapshot* adversaries. A persistent adversary receives: (1) the encrypted data; and (2) the transcripts of the interaction between the client and the server when a query is made. A snapshot adversary, on the other hand, only receives the encrypted data after a query has been executed.

The security of ESAs can be formalized using "leakage-parameterized" definitions following [19, 18]. In this framework, a construction is proven secure with respect to a security definition that is parameterized with a specific leakage profile. Leakage-parameterized definitions for persistent adversaries were given in [19, 18] and for snapshot adversaries in [3]. [4] We recall these definitions here informally and refer the reader to [19, 18, 3] for the formal definitions.

Definition 2.1 (Security vs. persistent adversary (Informal))**.** *Let* $\Lambda = \big(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}\big) = \big(\mathsf{patt}_1, \mathsf{patt}_2\big)$ *be a leakage profile. An encrypted search algorithm* ESA *is* Λ*-secure if there exists a* PPT *simulator that, given* $\mathsf{patt}_1(\mathbf{D})$ *for an adversarially-chosen document collection* $\mathbf{D}$ *and* $\mathsf{patt}_2(\mathbf{D}, q_1, \ldots, q_t)$ *for adaptively-chosen queries* q_i*, can simulate the view of any* PPT *adversary. Here, the view includes the encrypted data collection and the transcript of the queries.*

[3] Note that different schemes have different leakage profiles but, until recently, all known constructions leaked one of the patterns described above. The total volume pattern in particular is very difficult to supress and is leakaed by all known constructions except for the constructions recently proposed in [44].

[4] Even though parameterized definitions were introduced in the context of SSE and STE, they can be (and have been) applied to other primitives, including to FHE-, PPE-, ORAM- and FE-based solutions.

Known leakage profiles. There are many ways to design ESAs and each one provides a tradeoff between leakage and efficiency. Here, we summarize some of the most common leakage profiles and refer the reader to Appendix A for an overview of how these profiles can be achieved (and their cost). For ease exposition, we will ignore setup leakage in this work and just denote it by $\star$. This is justified since none of our attacks rely on it and, moreover, there are no known attacks that leverage it. As we show in Appendix A, there are structured ESAs with leakage profiles,

$$\Lambda_{\mathsf{BSL}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\star, \big(\mathsf{qeq}, \mathsf{rid}, \mathsf{vol}\big)\big)$$

and

$$\Lambda_{\mathsf{OPQ}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\star, \big(\mathsf{qeq}, \mathsf{tvol}\big)\big),$$

and oblivious ESAs with leakage profiles,

$$\Lambda_{\mathsf{SMI}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\star, \big(\mathsf{rlen}, \mathsf{rid}, \mathsf{vol}\big)\big)$$

and

$$\Lambda_{\mathsf{FLL}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\star, \big(\mathsf{rlen}, \mathsf{tvol}\big)\big).$$

Adversarial model. As discussed in Section 1, we consider two kinds of attacks each of which is carried out by different adversaries. Known-data attacks (i.e., leakage-abuse attacks) are carried out by a passive adversary that: (1) observes all query operations and therefore sees the query leakage; (2) knows a fraction of the client's data; and (3) knows the universe of keywords from which the queries are drawn. Chosen-data attacks (i.e., injection attacks) are carried out by an active adversary that can add arbitrary documents either adaptively (i.e., as a function of previously-observed search results and/or leakage) or non-adaptively. [5]

3 Volumetric Known-Data Attacks

In this section we present four new known-data attacks, three of which are volumetric. The first is *volume analysis* which exploits the total volume pattern tvol. The second attack is *selective volume analysis* which exploits the total volume and response length patterns $(\mathsf{tvol}, \mathsf{rlen})$. The last two are concrete instantiations of an attack framework we refer to as *subgraph* attacks. The first instantiation, the *volumetric subgraph attack*, exploits the volume pattern vol. The second instantiation, the *identifier subgraph attack* exploits the identifier pattern rid.

Remark on queries vs. keywords. For ease of exposition, we will use the term *query* when referring to a keyword that is unknown to the attacker. In particular, given leakage $\mathsf{patt}(\mathbf{D}, q_1, \ldots, q_t)$ on a sequence of t *queries*, the adversary's goal will be to match each query q_i to a keyword $w \in \mathbb{W}$.

3.1 Volume Analysis

Volume analysis exploits the total volume pattern. It can be viewed as a volume-based analogue of frequency analysis. It takes as auxiliary input a known dataset $\widetilde{\mathbf{D}}$ and as leakage

$$\mathsf{tvol}(\mathbf{D}, q_1, \ldots, q_t) = (v_1, \ldots, v_t)$$

[5] Note that for chosen-data attacks, one assumes a trivial client that accepts all injected documents without any filtering.

Consider the attack VolAn defined as follows:

- $\mathsf{VolAn}\big(\widetilde{\mathbf{D}}, (v_1, \ldots, v_t)\big)$:
  1. initialize a map $\alpha : [t] \to \mathbb{W}$;
  2. let $f(w) = \sum_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}(w)} |\widetilde{\mathrm{D}}|_{\mathsf{w}}$;
  3. for all $v_i \in (v_1, \ldots, v_t)$, set $\alpha(i) = \mathsf{argmax}_{w \in \mathbb{W}}\big\{f(w) : f(w) \leq v_i\big\}$,
  4. output α.

Figure 1: Volume analysis.

where $v_i = \sum_{\mathrm{D} \in \mathbf{D}(q_i)} |\mathrm{D}|_{\mathsf{w}}$. The attack then maps the ith query q_i to the keyword $w \in \mathbb{W}$ that has the closest *known* volume to v_i. More precisely, the attack maps q_i to

$$\mathsf{argmax}_{w \in \mathbb{W}}\left\{ f(w) : f(w) \leq v_i \right\},$$

where f is the function

$$f(w) = \sum_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}(w)} |\widetilde{\mathrm{D}}|_{\mathsf{w}},$$

that maps each keyword to its known volume. The pseudo-code of the attack is detailed in Figure 1.

Efficiency. The attack runs in $O(t \cdot \#\mathbb{W})$ time.

3.2 Selective Volume Analysis

We now describe an extension of volume analysis we call *selective volume analysis*, that exploits the total volume tvol and the response length rlen. It takes as input known data $\widetilde{\mathbf{D}}$ and leakage

$$\mathsf{tvol} \times \mathsf{rlen}(\mathbf{D}, q_1, \ldots, q_t) = \Big(\big(v_1, r_1\big), \ldots, \big(v_t, r_t\big) \Big)$$

where $v_i = \sum_{\mathrm{D} \in \mathbf{D}(q_i)} |\mathrm{D}|_{\mathsf{w}}$ and $r_i = \#\mathbf{D}(q_i)$. The attack proceeds in two steps: window matching and selectivity filtering which we detail below.

Window matching. The first step is similar to volume analysis in that we map queries to keywords with known volumes that are close to the observed volumes. The main difference, however, is that instead of mapping to a single keyword we now map to a set of keywords. Specifically, we map each query q_i to all keywords with known volumes that are within a specific distance from the query's observed volume. The idea behind this approach can be seen with the following concrete example. Suppose there are two keywords w_1 and w_2 with close volumes (with respect to the original dataset) and recall that the adversary knows only a δ fraction of $\mathbf{D}$. When $\delta < 1$, it is possible that the known volume of one keyword, say w_1, is different than its real volume while the other keyword's known volume is the same as its real volume. In such a setting, volume analysis might map the query to w_2 instead of w_1. Otherwise, if there is no potential volume in the window $[\delta \cdot v_i, v_i]$, for $i \in [t]$, we just search for the closest volume to $\delta \cdot v_i$ in the known dataset such that

$$\mathsf{argmax}_{w \in \mathbb{W}}\left\{ f(w) : f(w) \leq \delta \cdot v_i \right\},$$

where f is the function $f(w) = \sum_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}(w)} |\widetilde{\mathrm{D}}|_{\mathsf{w}}$ that maps each known keyword to its known volume.

Selectivity filtering. At this stage, the attack has a set S_i of potential keyword matches. It then starts filtering out keywords by removing any keyword that has a known selectivity that is too far away from the observed selectivity (response length pattern). For this, we determine a range of acceptable selectivities that are computed based on: (1) an error parameter ε; (2) the known data rate δ; (3) the observed selectivity r_i; (4) the observed volume of the keyword under consideration; and (5) the average document size (over documents in $\widetilde{\mathbf{D}}$) θ.

The attack proceeds by first computing an *estimated selectivity* that will be more or less close to the selectivities that the adversary knows of. In particular, for each keyword $w \in S_i$, we compute the estimated selectivity, $\hat{r}_i$ as

$$\hat{r}_i = r_i - \big(v_i - f(w)\big)/\theta,$$

where $f(w) = \sum_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}(w)} |\widetilde{\mathrm{D}}|_{\mathsf{w}}$. This equation can be explained as follows: we first calculate the difference of the observed volume v_i and the known volume of the selected keyword $f(w)$; then given the average document size θ, the quantity $\big(v_i - f(w)\big)/\theta$ is an estimated number of documents that the adversary does not know of. Finally, subtracting the latter from r_i will give an estimated selectivity for the keyword w.

Now given this estimated selectivity, our goal is to find out whether such a quantity is close enough to the selectivity of keyword w. For this we simply check if $\hat{r}_i$ is larger than $\#\widetilde{\mathbf{D}}(w)$ by at most λ– a linear error that depends of the known-data rate δ, the selectivity r_i, and an error parameter ε. Note that the computation of λ can vary depending on the dataset. Finally, if the selectivity is not close enough or if the observed selectivity is smaller than the selectivity of w, we remove w from potential matches S_i. We repeat these steps for all keywords $w \in S_i$ and we simply map the ith query to the first keyword remaining in S_i, for all $i \in [t]$. The pseudo-code of the attack is detailed in Figure 2.

Efficiency. The attack runs in $O(t \cdot \#\mathbb{W})$ time.

3.3 Subgraph Attacks

In this section, we present our subgraph attack framework. Subgraph attacks are query-recovery attacks with known-data that can exploit any *atomic* leakage pattern; that is, any pattern that reveals a function of each matching document. Formally, a pattern patt is atomic if there exists a function $h : \mathbf{D} \to Y$ such that

$$\mathsf{patt}(\mathbf{D}, q_1, \ldots, q_t) = \big(L_1, \ldots, L_t\big)$$

where, for all $i \in [t]$, L_i is a tuple $(h(\mathrm{D}))_{\mathrm{D} \in \mathbf{D}(q_i)}$. Atomic patterns are relatively common and include the volume pattern vol and the identifier pattern rid. In the case of the volume pattern, h is the function $|\cdot|_{\mathsf{w}}$. In the case of the identifier pattern, h is the function ids. In the following, we refer to the value $h(\mathrm{D})$ as D's *handle*. We stress that just because a subgraph attack can be defined with respect to any atomic leakage pattern, it does not necessarily mean that it will be successful against that pattern. Its accuracy has to be verified experimentally, as we do in Section 5. The attack is described in detail in Figure 3 and works as follows.

Let $\varepsilon \in \mathbb{N}$ be a public parameter, $\theta \in \mathbb{N}$ be the average document size, and δ be the known-data rate. Consider the SelVolAn attack defined as follows:

- $\mathsf{SelVolAn}\big(\widetilde{\mathbf{D}}, \big((v_1, r_1), \ldots, (v_t, r_t)\big), \delta, \theta, \varepsilon\big)$:
  1. initialize t empty sets $S_1, \ldots, S_t$ and a map $\alpha : [t] \to \mathbb{W}$;
  2. let $f(w) = \sum_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}(w)} |\widetilde{\mathrm{D}}|_{\mathsf{w}}$;
  3. for all $i \in [t]$,
     (a) for all $v \in [\delta \cdot v_i, v_i]$,
        – if there exists $w \in \mathbb{W}$ such that $f(w) = v$, then set $S_i = S_i \bigcup \{w\}$;
     (b) if $\#S_i = 0$, set
     $$S_i = \left\{ \mathsf{argmax}_{w \in \mathbb{W}} \big\{ f(w) : f(w) \leq \delta \cdot v_i \big\} \right\},$$
     (c) for all $w \in S_i$,
        i. set $\lambda = (1 - \delta) \cdot r_i / \varepsilon$;
        ii. compute
        $$\hat{r}_i = r_i - \big(v_i - f(w)\big)/\theta;$$
        iii. if $\hat{r}_i - \lambda > \#\widetilde{\mathbf{D}}(w)$ or $\#\widetilde{\mathbf{D}}(w) > r_i$, then set $S_i = S_i \setminus \{w\}$;
     (d) set $\alpha(i) = w$ where w is the first keyword in S_i;
  4. output α.

Figure 2: Selective volume analysis.

Bipartite graphs. The attack takes as input an auxiliary data set $\widetilde{\mathbf{D}} \subseteq \mathbf{D}$ and query leakage $(L_1, \ldots, L_t)$ defined as above. It starts by creating two bipartite graphs $\widetilde{\mathrm{G}} = \big((\widetilde{\mathrm{L}}, \mathbb{W}), \widetilde{\mathrm{E}}\big)$ and $G = \big((\mathrm{L}, \mathrm{Q}), \mathrm{E}\big)$ from the auxiliary data and the leakage, respectively. The vertex set $\widetilde{\mathrm{L}}$ is composed of the handles of the known documents; that is,

$$\widetilde{\mathrm{L}} = \left\{ h(\widetilde{\mathrm{D}}) \right\}_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}}.$$

For all keywords $w \in \mathbb{W}$ and documents $\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}$, $\widetilde{\mathrm{E}}$ includes an edge $\big(w, h(\widetilde{\mathrm{D}})\big)$ if $w \in \widetilde{\mathrm{D}}$. The second bipartite graph $G = ((\mathrm{L}, \mathrm{Q}), \mathrm{E})$ is constructed as follows. The vertex set L is composed of the observed document handles; that is,

$$\mathrm{L} = \left\{ h(\mathrm{D}) \right\}_{\mathrm{D} \in \bigcup_{i=1}^{t} \mathbf{D}(q_i)}.$$

The vertex set $\mathrm{Q} = (q_1, \ldots, q_t)$ is composed of the queries q_1 through q_t. The edges E are created using the observed leakage by adding an edge $(q_j, h(\mathrm{D}))$ if $h(D) \in L_j$, for all $j \in [t]$ and $\mathrm{D} \in \bigcup_{i=1}^{t} \mathbf{D}(q_i)$.

Refinement. Given these two graphs, the algorithm's goal is to match each q_i in Q to some w in $\mathbb{W}$. For each q_i, the attack will build a set of *potential* keyword matches $S_i \subseteq \mathbb{W}$ and keep refining it using several filtering steps. We will denote by $S_i^{(j)}$ the set of q_i's potential keyword matches after the jth refinement step. Let $S_i^{(0)} = \mathbb{W}$ and let w_i^* be q_i's correct match. The first filtering

Let $\varepsilon \in \mathbb{N}$ be a public error parameter and δ be the known-data rate. Let $h : \mathbf{D} \to Y$ be the function associated to patt. Consider the subgraph attack Subgraph defined as follows:

- $\mathsf{Subgraph}\big(\widetilde{\mathbf{D}}, (L_1, \cdots, L_t), \epsilon, \delta\big)$:
  1. initialize t empty sets $S_1, \cdots, S_t$ and a map $\alpha : [\ell] \to \mathbb{W}$;
  2. create a bipartite graph $\widetilde{\mathrm{G}} = \big((\widetilde{\mathrm{L}}, \mathbb{W}), \widetilde{\mathrm{E}}\big)$ where
  $$\widetilde{\mathrm{L}} = \big\{h(\widetilde{\mathrm{D}})\big\}_{\widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}} \quad \text{and} \quad \widetilde{\mathrm{E}} = \big\{\big(w, h(\widetilde{\mathrm{D}})\big) : \forall w \in \mathbb{W}, \forall \widetilde{\mathrm{D}} \in \widetilde{\mathbf{D}}, w \in \widetilde{\mathrm{D}}\big\}$$
  3. create a bipartite graph $G = \big((\mathrm{L}, \mathrm{Q}), \mathrm{E}\big)$ where
  $$\mathrm{L} = \big\{h(\mathrm{D})\big\}_{\mathrm{D} \in \bigcup_{i=1}^{t} \mathbf{D}(q_i)} \quad \text{and} \quad \mathrm{Q} = (q_1, \cdots, q_t) \quad \text{and}$$
  $$\mathrm{E} = \big\{\big(q_j, h(\mathrm{D})\big) : \forall j \in [t], \forall \mathrm{D} \in \bigcup_{i=1}^{t} \mathbf{D}(q_i), h(D) \in L_j\big\}$$
  4. for all $i \in [t]$,
     (a) for all $w \in \mathbb{W}$, if $N_{\widetilde{\mathrm{G}}}(w) \subseteq N_{\mathrm{G}}(q_i)$, then set $S_i^{(1)} = S_i^{(1)} \bigcup \{w\}$;
     (b) for all $w \in S_i^{(1)}$, if $\#N_{\widetilde{\mathrm{G}}}(w) \geq \delta \cdot \#N_G(q_i) - \varepsilon$, then set $S_i^{(2)} = S_i^{(2)} \bigcup \{w\}$;
     (c) compute
     $$S_i^{(3)} = S_i^{(2)} \bigcap \Big(\bigcap_{\alpha \in \widetilde{\mathrm{L}} \cap L_i} \widetilde{\mathrm{D}}_\alpha \Big);$$
     (d) if $\#S_i^{(3)} = 1$, then set $\alpha(i) = w$ where $S_i^{(3)} = \{w\}$;
  5. let $A \subseteq [t]$ be the set of all indices for which $\#S_i^{(3)} = 1$;
  6. while $\#A$ is increasing,
     (a) for all $S_i^{(\ell)}$ such that $\#S_i^{(\ell)} > 1$, then set
     $$S_i^{(\ell+1)} = S_i^{(\ell)} \setminus \Big(\bigcap_{j \in A} \alpha(j) \Big);$$
     (b) if $\#S_i^{(\ell+1)} = 1$, then set $\alpha(i) = w$ where $S_i^{(\ell+1)} = \{w\}$;
     (c) update A;
  7. output α.

Figure 3: The Subgraph framework.

step is based on the observation that w_i^*'s matching documents in $\widetilde{\mathbf{D}}$ have to be a subset of q_i's matching documents in $\mathbf{D}$. More formally, we have

$$S_i^{(1)} = \Big\{ w \in \mathbb{W} : N_{\widetilde{\mathrm{G}}}(w) \subseteq N_{\mathrm{G}}(q_i) \Big\},$$

where $N_{\widetilde{\mathrm{G}}}(w)$ and $N_{\mathrm{G}}(w)$ are the neighbors of w and q_i in $\widetilde{\mathrm{G}}$ and G, respectively. The second filtering step is based on the observation that the selectivity of w^* in $\widetilde{\mathbf{D}}$ should be a δ fraction of its selectivity in $\mathbf{D}$, where δ is the known-data rate. Based on this we have

$$S_i^{(2)} = \Big\{ w \in S_i^{(1)} : \#N_{\widetilde{\mathrm{G}}}(w) \geq \delta \cdot \#N_G(q_i) - \varepsilon \Big\},$$

where ε is an error parameter we set experimentally.

Cross filtering. The next filtering step is optional and can be used only if the function $h : \mathbf{D} \to Y$ is a bijection and if the adversary knows a large enough fraction of $\mathbf{D}$. The observation we rely on is that the correct keyword in a potential set $S_i^{(2)}$ must be contained in all the documents in the set $h^{-1}(L_i)$. As a concrete example, consider the case where h is the document ID function id. In this case, the observation above translates to the fact that the correct keyword w_i^* in $S_i^{(2)}$ must be contained in all the documents $(\mathrm{D}_\alpha)_{\alpha \in L_i}$. This follows from the correctness of the ESA scheme. More formally, we have that

$$w_i^* \in S_i^{(2)} \bigcap \Big(\bigcap_{\mathrm{D} \in h^{-1}(L_i)} \mathrm{D} \Big).$$

Notice, however, that we may not be able to compute the above subset because it requires us to invert h and recover all the documents that matched the query. In particular, we can only invert h if our auxiliary dataset is complete, i.e., $\delta = 1$. Nevertheless, we can approximate the set h^{-1} even when $\delta < 1$ as follows. We use the set $\widetilde{\mathrm{L}} \cap L_i$ which is the subset of observed handles of documents that we know. We then compute

$$S_i^{(3)} = S_i^{(2)} \bigcap \Big(\bigcap_{\alpha \in \widetilde{\mathrm{L}} \cap L_i} \widetilde{\mathrm{D}}_\alpha \Big).$$

At the end of this step, if the size of the set $S_i^{(3)}$ is equal to 1, then this means that we found a match for the ith query q_i, and therefore update the map α accordingly.

Iterative elimination. The final step of the attack relies on the observation that if some w is the correct match for a query q_i then it cannot be the correct match for another query q_j, where $i \neq j$. In other words, if w is the unique element of some potential set $S_i^{(\ell)}$, for $\ell \in [4, t+3]$, then w cannot be the matching keyword in some other potential set $S_j^{(\ell)}$ and, therefore, we can remove it from $S_j^{(\ell)}$. If this removal leads to $S_j^{(\ell)}$ having a single element, then we can in turn remove that element from other potential sets. This process keeps going until the potential sets stabilize. Note that while the algorithm will terminate it may not find matches for all q_i.

Efficiency. The first filtering step is $O(t \cdot \#\mathbb{W})$, the cross filtering step is

$$O\Big(t + \sum_{i=1}^{t} \sum_{\mathrm{D} \in \mathbf{D}(q_i)} \#\mathrm{D}\Big)$$

and the iterative elimination step is $O(t^2)$. In total, the algorithm runs in time

$$O\Big(t \cdot \#\mathbb{W} + \sum_{i=1}^{t} \sum_{\mathrm{D} \in \mathbf{D}(q_i)} \#\mathrm{D}\Big).$$

The volumetric subgraph attack. As discussed above, subgraph attacks can exploit any atomic leakage pattern by properly instantiating the handle function h. The volumetric subgraph attack results from instantiating h with the function $|\cdot|_{\mathsf{w}}$ which maps each document to its volume. Note that $|\cdot|_{\mathsf{w}}$ is not bijective so this instantiation cannot use the cross filtering step.

The identifier subgraph attack. Our subgraph framework can also be used to exploit the identifier pattern by instantiating h with the function ids that maps keywords to the identifiers of the documents that contain it. Note that because ids is bijective, we can use the cross filtering step.

4 Volumetric Injection Attacks

Injection attacks were first proposed by Cash et al. [13] in the context of the PPE-based ShadowCrypt and Mimesis [36, 49] systems. The first injection attacks on structured ESAs were described by Zhang, Katz and Papamanthou in [74]. In that work, two attacks are described, each of which exploits the identifier pattern. In this section, we describe new volumetric injection attacks. In particular, our attacks exploit the *total volume pattern* and, therefore, can even be used against oblivious ESAs.

4.1 The Decoding Attack

The decoding attack is described in detail in Figure 4 and works as follows. It works in two phases: baseline and recovery. In the *baseline* phase, the adversary waits until it has observed the total volumes for all keywords in $\mathbb{W}$. During the recovery phase, the adversary observes an additional sequence of $t \geq 1$ client queries $\mathbf{q} = (q_1, \ldots, q_t)$ with total volumes $\mathbf{v} = (v_1, \ldots, v_t)$. The attack will recover all queries in $\mathbf{q}$. We now describe each phase in more detail.

Baseline. During the baseline phase, the adversary observes queries until it holds the volumes $\mathbf{b} = (b_1 \ldots, b_m)$ of all the keywords $w_1, \ldots, w_\ell \in \mathbb{W}$. It then creates a set of documents to inject as follows. It first computes an *offset* γ defined as

$$\gamma = \min\Big\{\gamma \in \mathbb{N} : \forall i, j \in [m], \gamma \not| \, b_i - b_j\Big\}.$$

For all keywords $w_i \in \mathbb{W}$, the adversary injects a document with volume $i \cdot \gamma$ filled with w_i. Intuitively, this step increases w_i's volume by $i \cdot \gamma$.

Consider the attack Decoding defined as follows:

- Decoding$(\cdot)$:
  1. gather the baseline volumes $\mathbf{b} = (b_1 \ldots, b_m)$ for all keywords in $\mathbb{W}$ and initialize a mapping $\alpha : [\ell] \to \mathbb{W}$;
  2. compute
  $$\gamma = \min \left\{ \gamma \in \mathbb{N} : \forall\ i, j \in [t], \gamma \not| v_i - v_j \right\};$$
  3. for all $w_i \in \mathbb{W}$, the adversary injects a document with volume $i \cdot \gamma$ containing w_i;
  4. gather the observed volumes $\mathbf{v} = (v_1, \ldots, v_t)$;
  5. for all $i \in [t]$,
     (a) find $u \in [\ell]$ such that $v_i - u \cdot \gamma = b_j$, for some $j \in [m]$;
     (b) set $\alpha(j) = w_u$;
  6. output α.

Figure 4: The decoding attack.

Recovery. During the recovery phase, the adversary will observe volumes $\mathbf{v} = (v_1, \ldots, v_t)$ on queries $\mathbf{q} = (q_1, \ldots, q_t)$. Note that for all $i \in [t]$, the volume of q_i can be written as

$$v_i = b_j + u \cdot \gamma$$

for some $j \in [m]$ and $u \in [\ell]$. The adversary's goal now is to map q_i to some keyword $w \in \mathbb{W} = (w_1, \ldots, w_\ell)$. It does this as follows. It checks if there exists a $u \in [\ell]$ such that $v_i - u \cdot \gamma$ is equal to one of the baseline volumes $(b_1, \ldots, b_m)$. If this is the case, then the adversary maps q_i to keyword w_u. Note that there can be at most a single baseline volume that satisfies this condition. To see why, suppose there exists two baseline volumes $\beta_1 \neq \beta_2$ and two values $z \neq z'$ in $[\ell]$ such that

$$v_i - z \cdot \gamma = \beta_1 \qquad \text{and} \qquad v_i - z' \cdot \gamma = \beta_2.$$

But this implies that $\gamma \cdot (z' - z) = \beta_1 - \beta_2$ which is a contradiction since $\gamma \not| b_i - b_j$, for all $i, j \in [m]$.

Correctness & efficiency. The decoding attack recovers all queries in $\mathbf{q}$ as long as the user did not add any documents of its own. As described, the attack recovers all queries in $\mathbf{q}$ by injecting $\#\mathbb{W}$ documents with a total volume of $O(\gamma \cdot \#\mathbb{W}^2)$. Note, however, that the attack can be made a lot more efficient if the adversary is only interested in recovering queries within some target set $T \subset \mathbb{W}$. In this case, in the baseline phase the adversary only needs to gather the baseline volumes for queries in T. Similarly, it only needs to inject documents for keywords in T. With this modification, the attack recovers queries in $\mathbf{q} \cap T$ by injecting T documents with a total volume of $O(\gamma \cdot \#T^2)$. Our evaluation demonstrates that the offset γ for different subsets of the `Enron` dataset is between 4 and 16 KBytes depending on the query selectivity (see Appendix D).

4.2 The Binary Search Attack

The binary search attack is described in detail in Figure 5 and works as follows. It works in three phases: baseline, targeting and recovery. In the *baseline* phase, the adversary waits until it

has observed the total volumes for all keywords in $\mathbb{W}$. During the *targeting* phase the adversary observes more client queries until it decides on a query q_0, with total volume v_0, that it wishes to target. In the *recovery* phase, the adversary observes an additional sequence of $t > \log \#\mathbb{W}$ client queries $\mathbf{q} = (q_1, \ldots, q_t)$ with volumes $\mathbf{v} = (v_1, \ldots, v_t)$ that it will use to recover its target q_0. We now describe each phase in more detail.

Baseline. During the baseline phase, the adversary observes queries until it holds the volumes $\mathbf{b}^{(0)} = (b_1, \ldots, b_m)$ for each keyword $w_1, \ldots, w_\ell \in \mathbb{W}^{(0)}$, where $\mathbb{W}^{(0)} = \mathbb{W}$.

Targeting. During the targeting phase, the adversary observes queries until it decides on a target query q_0 with total volume v_0. It then partitions $\mathbb{W}^{(0)}$ into two equal-sized sets, $\mathbb{W}_0^{(0)}$ and $\mathbb{W}_1^{(0)}$ and computes an offset

$$\gamma = \min \left\{ \gamma \in \mathbb{N} : \forall j \in [m], \gamma \neq |v_0 - b_j^{(0)}| \wedge \gamma \geq |\mathbb{W}_1^{(0)}|_{\mathsf{w}} \right\}.$$

The adversary then injects a document with volume γ that contains all the keywords in $\mathbb{W}_1^{(0)}$.

Recovery. In the first round of the recovery phase, the adversary observes the total volume v_1 for query q_1. It then uses γ to decide on one of three cases:

- if $v_1 = v_0 + \gamma$ then the adversary concludes that $q_1 = q_0$ and that $q_0 \in \mathbb{W}_1^{(0)}$;
- if $v_1 = v_0$ then the adversary concludes that $q_1 = q_0$ and that $q_0 \in \mathbb{W}_0^{(0)}$;
- if $v_1 \neq v_0$ and $v_1 \neq v_0 + \gamma$ then the adversary concludes that $q_1 \neq q_0$.

If $q_0 \in \mathbb{W}_1^{(0)}$ the adversary sets $\mathbb{W}^{(1)} = \mathbb{W}_1^{(0)}$. If $q_0 \in \mathbb{W}_0^{(0)}$ it sets $\mathbb{W}^{(1)} = \mathbb{W}_0^{(0)}$. For both these cases, before moving to the next round, the adversary re-injects a document with volume γ such that

$$\gamma = \min \left\{ \gamma \in \mathbb{N} : \forall j \in [m], \gamma \neq |v_1 - b_j^{(1)}| \wedge \gamma \geq |\mathbb{W}_1^{(1)}|_{\mathsf{w}} \right\},$$

where $\mathbf{b}^{(1)} = (\mathbf{b}^{(0)}, b_1^{(0)} + \gamma, \cdots, b_m^{(0)} + \gamma)$, $\mathbb{W}_0^{(1)}$ and $\mathbb{W}_1^{(1)}$ are the two partitions of $\mathbb{W}^{(1)}$. If otherwise $q_1 \neq q_0$, the adversary moves to the next round without changing $\mathbb{W}^{(0)}$ nor injecting a new file.

Correctness & efficiency. The binary search attack will recover q_0 as long as: (1) it appears $\log \#\mathbb{W}$ times in $\mathbf{q} = (q_1, \ldots, q_t)$; (2) it has a unique volume in the baseline volumes; and (3) the user did not add any document of its own. The attack needs to inject $\log \#\mathbb{W}$ files with a total volume of $\Omega(\#\mathbb{W})$. Our evaluation demonstrates that the size of the query space was the main factor that determines the total injected volume (i.e., $\gamma \simeq \#\mathbb{W}/2^u$ for all $u \leq \log \#\mathbb{W}$ for different subsets of the Enron dataset). The total injected volume was around 8KBytes (see Appendix D).

5 Empirical Evaluation

To evaluate the effectiveness of our attacks, we implemented and evaluated them under different conditions. The results are perhaps surprising and provide a new and more nuanced perspective on the potential impact and limitations of leakage abuse attacks.

Consider the attack Binary defined as follows:

- $\mathsf{Binary}(\,\cdot\,)$:

  1. gather the baseline volumes $\mathbf{b}^{(0)} = (b_1^{(0)} \ldots, b_m^{(0)})$ for all keywords in $\mathbb{W}^{(0)} = \mathbb{W}$;
  2. partition $\mathbb{W}^{(0)}$ into two equal-sized sets, $\mathbb{W}_0^{(0)}$ and $\mathbb{W}_1^{(0)}$;
  3. observe queries and determine a target query q_0 with total volume $v^{(0)} = v_0$;
  4. compute
  $$\gamma = \min\left\{\gamma \in \mathbb{N} : \forall j \in [m], \gamma \neq |v^{(0)} - b_j^{(0)}| \wedge \gamma \geq |\mathbb{W}_1^{(0)}|_{\mathsf{w}}\right\};$$
  5. injects a document with volume γ that contains all the keywords in $\mathbb{W}_1^{(0)}$;
  6. while $\#\mathbb{W}^{(u)} > 1$ and for every new observed volume v_i,
     (a) if $v_i = v^{(u)} + \gamma$ then set $\mathbb{W}^{(u+1)} = \mathbb{W}_1^{(u)}$;
     (b) if $v_i = v^{(u)}$ then set $\mathbb{W}^{(u+1)} = \mathbb{W}_0^{(u)}$;
     (c) if $v_i \neq v^{(u)} + \gamma$ and $v_i \neq v^{(u)}$, then move to the next round;
     (d) if $\mathbb{W}^{(u)}$ has been updated, then,
        i. partition $\mathbb{W}^{(u+1)}$ into two equal-sized sets, $\mathbb{W}_0^{(u+1)}$ and $\mathbb{W}_1^{(u+1)}$;
        ii. set $v^{(u+1)} = v_i$;
        iii. compute
        $$\gamma = \min\left\{\gamma \in \mathbb{N} : \forall j \in [m], \gamma \neq |v^{(u+1)} - b_j^{(u+1)}| \wedge \gamma \geq |\mathbb{W}_1^{(u+1)}|_{\mathsf{w}}\right\},$$
     where $\mathbf{b}^{(u+1)} = (\mathbf{b}^{(1)}, \cdots, \mathbf{b}^{(u)}, b_1^{(u)} + \gamma, \cdots, b_m^{(u)} + \gamma)$;
  7. output $\mathbb{W}^{(\log \#\mathbb{W})} = \{w\}$.

Figure 5: Binary Search Attack.

Document collections. We build our document collections using the Enron Email dataset [62]. This dataset is composed of 150 folders with a total of 520,901 files. Each folder corresponds to the email account of a single individual and is itself composed of several folders including, for example, `inbox`, `sent`, `contacts`, `discussion threads`, etc. Starting from the entire Enron dataset, we generated different subsets that capture different settings of interest:

- *single user* (SU): is a document collection composed of one individual's email account. For this dataset, we picked the `arnold-j` folder which is 11.6MBytes and is composed of 4,944 files. The total number of keywords is 40,363. This collection models the traditional single user ESA setting where a single client uses an ESA to encrypt and privately access its own dataset.

- *small multiple users* (S-MU): is a document collection composed of multiple email accounts. For this dataset, we picked 5 folders with a total size equal to 26Mbytes. This document collection is composed of the following email accounts: `baughman-d`, `gay-r`, `heard-m`, `hendrickson-s` and `linder-e`. The dataset is composed of 9,416 files in total. The total number of keywords is 77,762. This collection models the multi-user ESA setting in which there is one party (e.g., a company) that uses an ESA to encrypt and store multiple users' documents. Here, queries can be performed by a subset of authorized users.

- *medium multiple user* (M-MU): is the same as above except that we increase the number of folders to 10 with a total size of 49Mbytes. The data collection is composed of the same email accounts as above plus: `allen-p`, `buy-r`, `forney-j`, `hyvl-d` and `keiser-k`. The total number of keywords is 115,679.

The purpose of the first two collections is to see whether the effectiveness of our attacks will vary as a function of different data distributions. The third collection is used to understand if increasing the size of the dataset impacts the effectiveness of the attacks. We note that we performed evaluations which we do not report here in order to avoid redundancy. In particular, we evaluated the attacks on a larger document collection from `Enron` of size 106MBytes and the results were similar to the ones on M-UM. We also evaluated our attacks on a subset of the `TREC 2007 Public Corpus` dataset [57], an email dataset composed of 75,419 emails (including spam), and the results were similar.

Data indexing. We indexed each data collection using `Apache Lucene` [1]. We removed 224 stop words listed in the `SnowBall` list [61]. Furthermore, we used the `Porter Stemming` implementation of `Lucene` so all words that share the same stem are mapped to the same root.

Query frequency. We observed that previous works on leakage abuse attacks [38, 13] evaluated the effectiveness of their attacks on the most frequent keywords in the dataset. While this assumption might hold in some settings, it is far from clear if this a reasonable assumption in practice. In fact, it might seem that users would more often search for keywords that occur less frequently in their dataset rather than for keywords that occur frequently. With this in mind, we evaluated all of our attacks in three different settings:

- *high selectivity*: the queries are sampled from the set of keywords with the highest selectivities (i.e., that appear in the largest number of documents). We noticed that keyword selectivities

in `Enron` are power-law distributed (see Appendix C) which implies that high-selectivity keywords tend to have *unique* selectivities while low-selectivity keywords tend to have less unique selectivities (in our datasets it was none).

- *low selectivity*: the queries are sampled from the set of keywords with the lowest selectivities. In our datasets, all the low-selectivity keywords had selectivity 1.
- *pseudo-low selectivity*: in this case we consider low-selectivity keywords with a slightly higher selectivity than above. In particular, we consider the case where the selectivity ranges from 10 to 13.

Size and composition of the query space. We first fix the size of the query space, $\mathbb{Q}$, to be 500. We then study the impact of increasing $\#\mathbb{Q}$. In particular, we increased $\#\mathbb{Q}$ up to 5000 keywords. We also consider two ways of instantiating the query space. The first consists of populating $\mathbb{Q}$ with keywords that only exist in the known-data collection $\widetilde{\mathbf{D}}$. This guarantees that all $q \in \mathbb{Q}$ must exist in $\widetilde{\mathbf{D}}$. As a consequence, the attacker would know that any client query will match at least one document in $\widetilde{\mathbf{D}}$. The second approach consists of populating $\mathbb{Q}$ from keywords that exist in the client's collection $\mathbf{D}$.

Experimental setting. As described above, there are several variables that can impact the effectiveness of our attacks. In our evaluation, we considered many different combinations of these variables and organized them in three main categories:

- *(C1) single keyword queries*: we consider the SU dataset and fix the size of the query space to be 500. We then vary the query selectivity and query space composition; refer to Figures 6a, 6b, 6c and 6d.
- *(C2) size of the query space*: this second category is the same as the first except that we increase the size of the query space to be 5000; refer to Figures 8a, 8b, 8c and 8d.
- *(C3) varying the datasets*: the third category is similar to the first category except that we replace the SU dataset with the S-MU and M-MU datasets; refer to Figures 10a, 10b, 10c and 10d.

All our attacks are evaluated against a query sequence $\mathbf{q}$ of size $t = 150$. The queries are sampled uniformly at random from the corresponding keyword space $\mathbb{Q}$ whose composition varies depending on the chosen approach (see above). In all our experiments, we start with the adversary knowing the entire client collection and then gradually decrease this knowledge until it knows only 5% of the documents (chosen uniformly at random). For each attack, we report the *recovery rate*, i.e., the number of queries recovered correctly over the total number of queries. We run all experiments 5 times and report the minimum, median and maximum of the recovery rate. All attacks are implemented in Java and the experiments were run on a MacBook 3.1 GHz Intel Core i7 with 16GBytes of RAM.

Baseline. In all our experiments we have included the *count-only* attack which is simply the Count attack [13] without the co-occurence matrix. This depicts constructions for which we have suppressed the co-occurence leakage, refer to Appendix A.

Overview of results. We noticed that the recovery rate of our attacks is impacted the most by the *selectivity* of the queries. In fact, it seems that evaluating known-data attacks on high- vs. low-selectivity queries leads to completely opposite conclusions.[6] Moreover, we noticed that changing the datasets or increasing the size of the query space led to some fluctuations but the overall trends remained the same. This shows that our attacks work across different settings and different dataset compositions. We found, however, that if client queries are not in the adversary's known dataset then the recovery rate is always low. This simply follows from the fact that there are several queries for which the adversary does not hold any part of the response. Below, we provide more detailed comments on the results of our evaluations:

- When the query space is composed of high-selectivity keywords, both $\mathsf{Subgraph}^{\mathsf{ID}}$ and $\mathsf{Subgraph}^{\mathsf{VL}}$ have a recovery rate of about 70% even with a known-data rate of 5% (see Figure 10a). We believe that the low known-data rate makes these attacks practical for high-selectivity keywords. However, if the query space is composed of low-selectivity keywords, then the recovery rate drops significantly. In fact, we found that $\mathsf{Subgraph}^{\mathsf{VL}}$ does not work at all while $\mathsf{Subgraph}^{\mathsf{ID}}$ has a recovery rate of about 20% even when $\delta = 1$; that is, with full knowledge of the client's data. With a known-data rate of $\delta = 1/2$, $\mathsf{Subgraph}^{\mathsf{ID}}$ only has 10% recovery rate. It tends to 0 for known-data rates smaller than 10%.

- For VolAn and SelVolAn, our evaluation shows that both attacks work only when: (1) the query space consists of high-selectivity keywords; and (2) the adversary has a high known-data rate, often at least .85. We refer the reader to Figure 6a for an example. In the case where the query space consists of low-selectivity keywords, the recovery rate is very low: around 10% even when $\delta = 1/2$. It drops significantly when δ gets smaller.

- When the query space consists of pseudo-low-selectivity keywords (i.e., with selectivity between 10 and 13), both the VolAn and SelVolAn attacks have high recovery rate only when the known-data rate $\delta \geq .8$ (see Figure 7). As δ decreases, the recovery rate stabilizes at around 18% and starts to decrease again to 0 when $\delta \leq 0.15$. Note that this recovery rate is the highest among the three selectivity classes for these two attacks. The recovery rates of both $\mathsf{Subgraph}^{\mathsf{ID}}$ and $\mathsf{Subgraph}^{\mathsf{VL}}$ against pseudo-low-selectivity queries are slightly better than the recovery rates of SelVolAn and VolAn. $\mathsf{Subgraph}^{\mathsf{ID}}$ and $\mathsf{Subgraph}^{\mathsf{VL}}$ also do better on pseudo-low-selectivity keywords than they do on low-selectivity keywords. However, they do a lot worse than they do against high-selectivity keywords.[7]

Results on chosen-data attacks. Since our injection attacks always succeed we do not evaluate their success rate empirically. We report, however, that in order to succeed we set the size of the keyword universe $\mathbb{W}$ to 500. Also, for the Decoding attack, one has to inject between 4 and 16 KBytes to recover one keyword depending on the type of the document collection and the keyword selectivity. For the Binary attack, the adversary has to inject around 8 KBytes for all document collections and this holds independently of the selectivity of the keyword. We provide more details about our evaluation in Appendix D.

[6] We recall that the experiments in [13] were done exclusively on high-selectivity keywords.

[7] We also conducted experiments in which the queries were sampled uniformly at random from the entire keyword space. The results were similar to the low-selectivity case (the $\mathsf{Subgraph}^{\mathsf{ID}}$ recovery rate was slightly higher though).

5.1 Detailed Results

We provide in the following more elaborate details about each of the categories that we have introduced above.

5.1.1 C1: Single Keyword Queries

In this section, we detail the evaluation for each of the attacks described in Section (3) in the C1 setting.

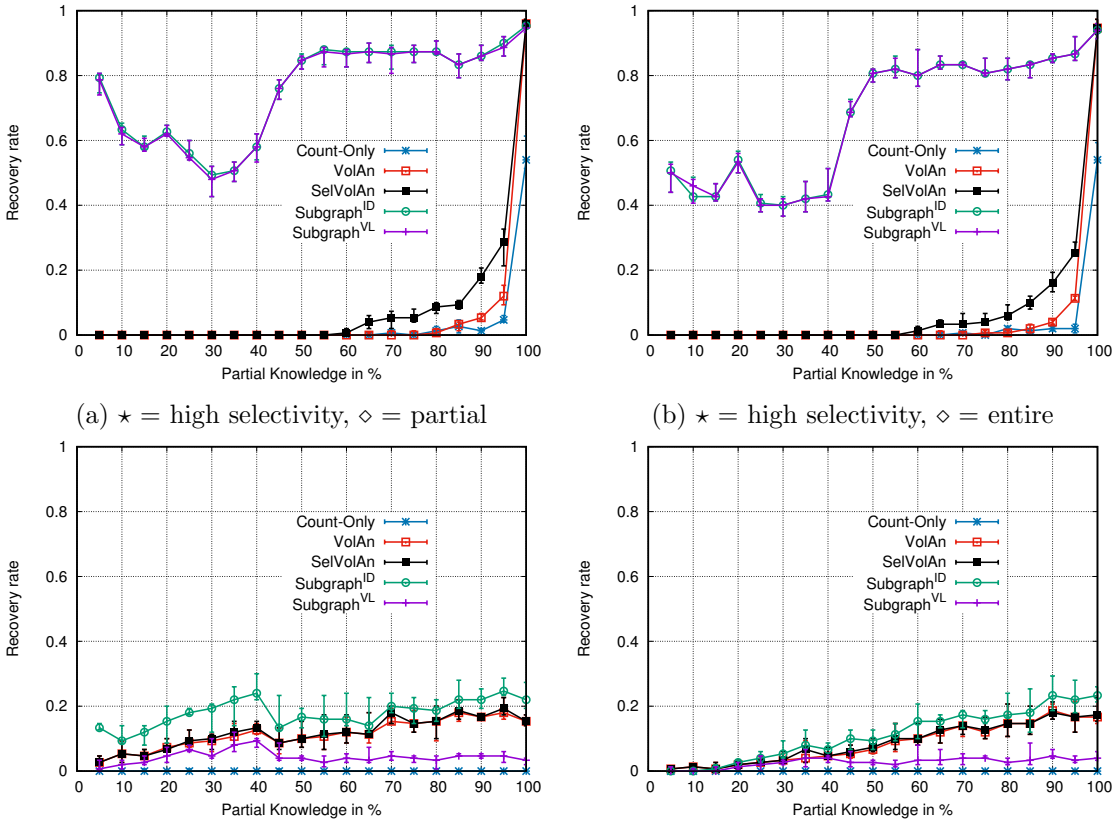
Subgraph^{ID} attack. When the query space is composed of high-selectivity keywords in the adversary known data $\widetilde{\mathbf{D}}$, the attack's recovery rate is high in general. As can be seen in Figure (6a), the recovery rate is around 90% for any known-data rate larger than 50%. The recovery rate then decreases to 50% with 30% known-data but then increases back to 80% with 5% known data. We believe that this surprising behavior is an artifact of the dataset used in the experiment as it is not repeated when the attack is run on other datasets. When the query space is composed of high-selectivity keywords, the attack also has high recovery rate. As illustrated in Figure (6b), the rate is around 90% for known-data rates larger than 50%. The recovery rate then decreases and stabilizes around 40%. On the other hand, when the query space is composed of the low-selectivity keywords, the recovery rate drops considerably. This can be seen in Figure (6c), where it decreases steadily and slightly fluctuates around 20% . Finally, in the case where the query space is composed of the low-selectivity keywords, the recovery rate decreases steadily from 23% to 0%. As can be seen in Figure (6d), for smaller known-data rates (i.e., below 50%) the recovery rate is less than 10%. Finally, as illustrated in Figure (7), when the query space is composed of pseudo-low-selectivity keywords, the recovery rate drops sharply and becomes very low for known-data rates smaller than 50%.

Subgraph^{VL} attack. The recovery rate of Subgraph^{VL} is similar to the one of Subgraph^{ID} when the query space is composed of the high-selectivity keywords. This can be seen in Figures (6a) and (6b). The recovery rate is also similar on pseudo-low-selectivity keywords which can be seen in Figure (7). On the other hand, as can be seen in Figures (6c) and (6d), the attack does poorly against low-selectivity keywords. In this case, the recovery rate is insignificant and is around 0% and 4%.

VolAn attack. As can be seen in Figures (6a) and (6b), the VolAn attack does well against high-selectivity keywords (i.e., recovery rate of 96%) when the adversary has full knowledge of the client's data but it drops own to 12% even with a known-data rate of 95% and further down to 5% with a known-data rate of 90%. For known-data rates lower than 85%, the recovery rate is almost null. When the query space is composed of pseudo-low-selectivity keywords, the recovery rate has similar behavior but with higher values. This can be seen in Figure (7). When the query space is composed of the low-selectivity keywords, the recovery rate decreases steadily from 18% to 2%. The drop-off is slightly sharper when the query space is composed of keywords from the client's dataset; refer to Figures (6c) and (6d).

SelVolAn attack. The recovery rate of SelVolAn behaves similarly to the one of VolAn except that it is slightly larger. It is around 96% when the adversary has full knowledge of the dataset, but drops

sharply to 28% with 95% known-data rate, and then to 18% with 90% known-data rate. The attack does exactly the same as VolAn when the query space is composed of the pseudo-low-selectivity keywords.



(a) $\star$ = high selectivity, $\diamond$ = partial

(b) $\star$ = high selectivity, $\diamond$ = entire

(c) $\star$ = low selectivity, $\diamond$ = partial

(d) $\star$ = low selectivity, $\diamond$ = entire

Figure 6: SU dataset. 150 keywords queried u.a.r. from 500 $\star$ keywords in the $\diamond$ dataset.

5.1.2 C2: Size of the Query Space

We now describe our evaluation for each of the attacks in Section (3) in the C2 setting. We note that the increase of keyword space does not have any impact on the recovery rate when the query space is composed of low-selectivity keywords. In fact, the results are very similar to the ones in the C1 experiments (see Figures (8c) and (8d)). This is mainly due to the fact that, in the SU dataset, there are 17745 keywords with the same selectivity (of 1), i.e., there are 17745 low-selectivity keywords. Therefore, increasing the query space size from 500 to 5000 does not have any impact because all the keywords will still have the same selectivity as in C1. In the following, we only focus on the case where the query space is composed of the high-selectivity keywords. In fact, since the selectivities are power law distributed, high-selectivity keywords tend to have unique selectivities; which leads to slightly different results as we going to explain below.

Subgraph^{ID} attack. The recovery rate of Subgraph^{ID} is similar to its recovery rate in C1 for any known-data rate larger than 50%. This can be seen in Figure (8a). The attack does slightly better

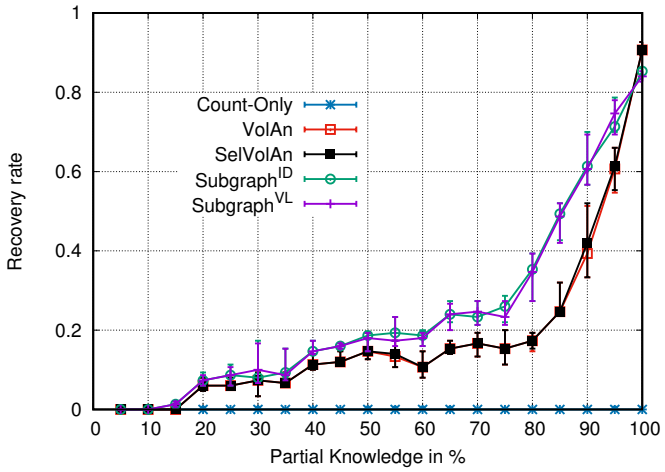


Figure 7: 150 keywords queried u.a.r. from 500 pseudo-less frequent keywords in SU.

for known-data rates between 25% and 50%. However, contrary to C1, and when the known-data rate is 20% or smaller, the recovery rate decreases considerably and reaches 35% with a known-data rate of 5%. This could be explained by the fact that the queries can include keywords with lower selectivities which negatively impact the accuracy of the attack for low known-data rates. On the other hand, when the query space is composed of keywords from the client's dataset, the recovery rate slightly decreases; especially with known-data rates between 20% and 50%. This can be seen in Figure (8b).

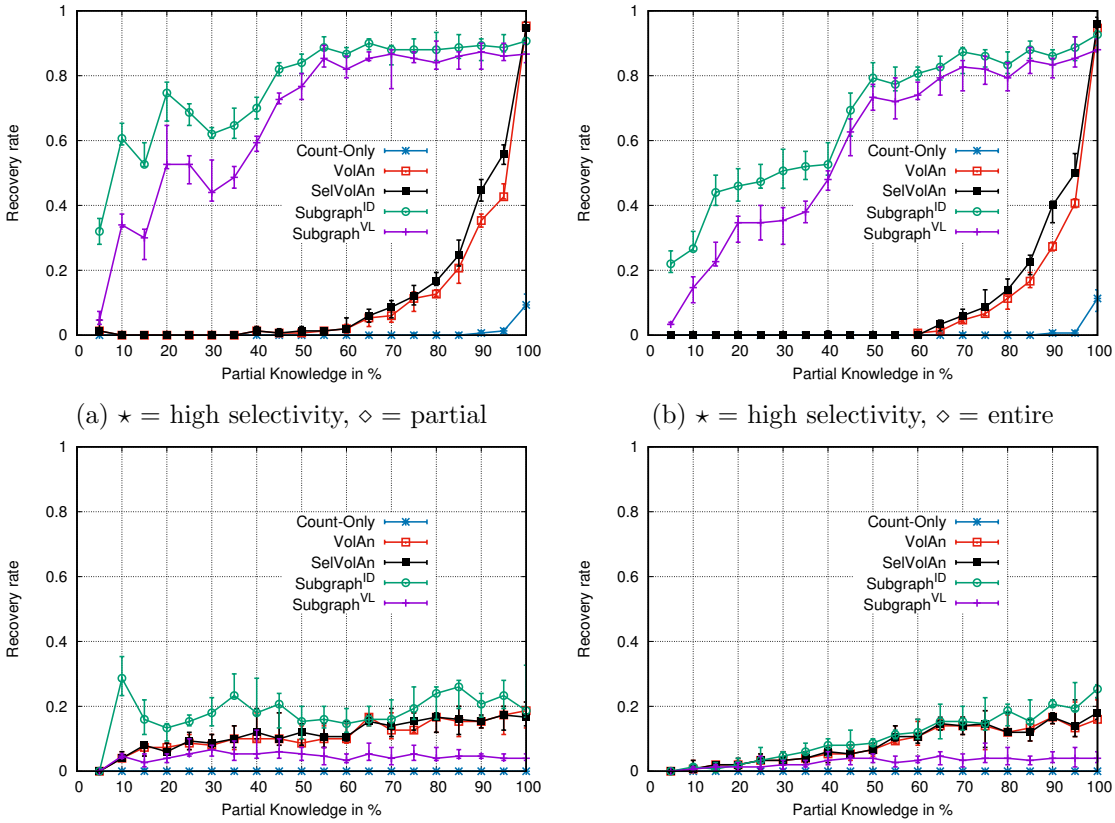
Subgraph^{VL} attack. The recovery rate of Subgraph^{VL} is similar to Subgraph^{ID} except that, starting from a 50% known-data rate, the recovery rates are 10% to 20% lower than those in C1.

VolAn and SelVolAn attacks. Both of the attacks behave similarly to how they behave in the C1 setting. However, the recovery rate of both slightly increases. In particular, instead of a sharp decrease at a 95% known-data rate, the drop-off is slightly smoother. For example, SelVolAn can still achieve 56% recovery at 95% and 16% recovery at 80%. This new behavior could stem from the fact that the total volume of keywords with lower selectivities is more identifying. Note however that with a 70% known-data rate or smaller, the recovery rate is almost null.

5.1.3 C3: Varying the Datasets

In this section, we describe the results of our evaluation for each of the attacks described in Section (3) in the C3 setting. We note that, here, varying the datasets does not impact the recovery rate when the query space is composed of low-selectivity keywords. In fact, the curves are very similar to the ones in the C1 and C2 experiments; this can be seen in Figures (9c), (9d), (10c) and (10d).

Subgraph^{ID} and Subgraph^{VL} attacks on the S-MU Dataset. When the query space is composed of keywords from adversary's known dataset $\widetilde{\mathbf{D}}$, the recovery rate is stable at around 80%. Otherwise, when the query space is composed of keywords from the client's dataset, the recovery



(a) $\star$ = high selectivity, $\diamond$ = partial

(b) $\star$ = high selectivity, $\diamond$ = entire

(c) $\star$ = low selectivity, $\diamond$ = partial

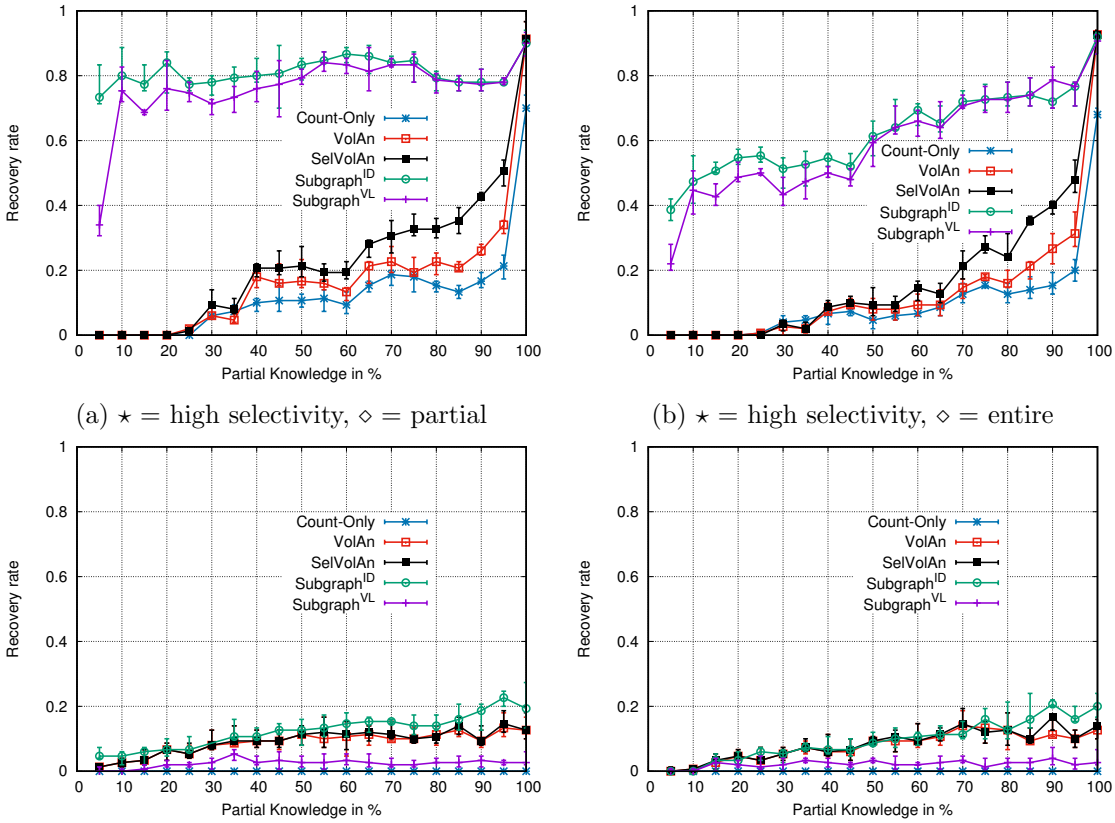
(d) $\star$ = low selectivity, $\diamond$ = entire

Figure 8: SU dataset. 150 keywords queried u.a.r. from 5000 $\star$ keywords in the $\diamond$ dataset.

rate of both attacks decreases steadily and hits 38% and 22% with a 5% known-data rate, respectively. This is illustrated in Figures (9a) and (9b). Compared to previous experiments, the recovery rate is slightly higher, especially in Figure (9a).

VolAn and SelVolAn attacks on the S-MU Dataset. Compared to the C2 setting, the recovery rate of both of these attacks is even smoother. In fact, the sharp drop-off is replaced with a smooth decrease; especially when the query space is composed of keywords from the adversary's known dataset. In fact, between a known-data rates of 95% and 40%, the recovery rate decreases between 50% and 20%. This is one of the highest recovery rates for theses two attacks that we found. This increase is smaller when the query space is composed of keywords from the client's dataset; refer to Figures (9a) and (9b).

Subgraph^{ID} and Subgraph^{VL} attacks on the M-MU Dataset. The recovery rates are similar to the ones for the S-MU dataset, except that with known-data rates between 70% and 95%, the recovery rate drops considerably—by 10% to 20% depending on the composition of the query space. This can be seen in Figures (10a) and (10b).



(a) $\star$ = high selectivity, $\diamond$ = partial

(b) $\star$ = high selectivity, $\diamond$ = entire

(c) $\star$ = low selectivity, $\diamond$ = partial

(d) $\star$ = low selectivity, $\diamond$ = entire

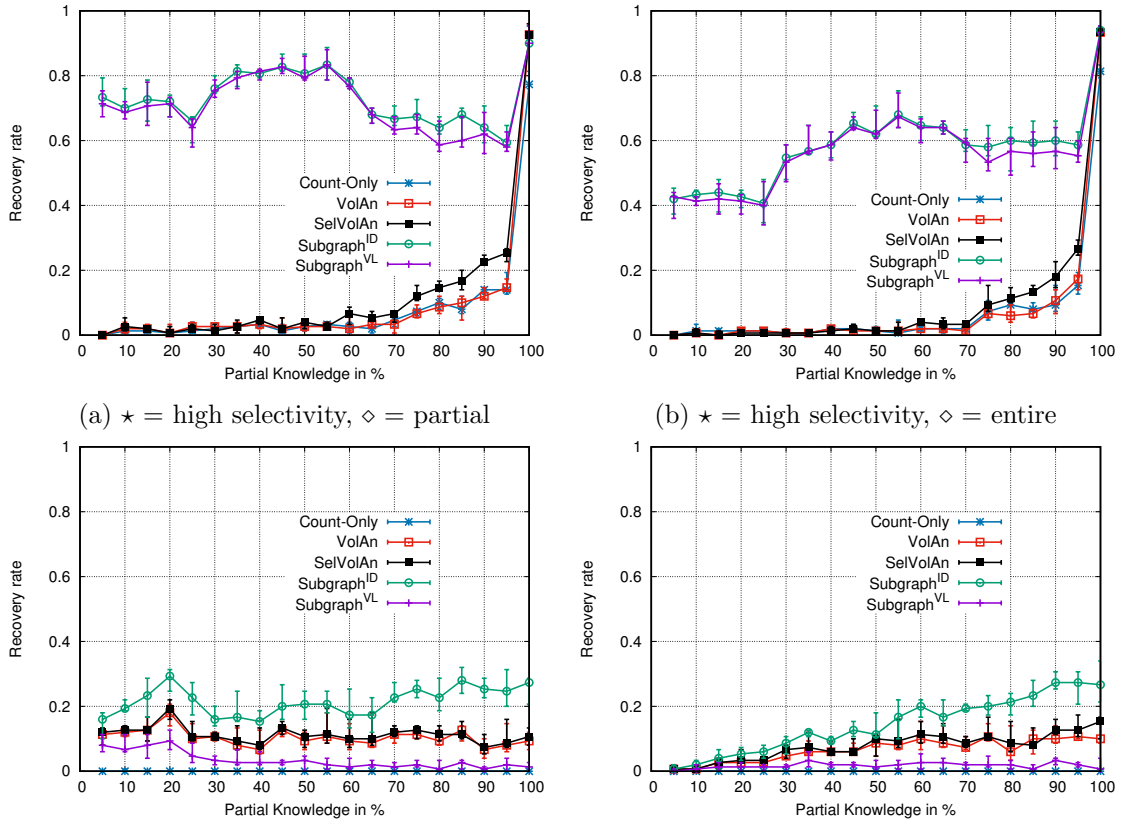
Figure 9: S-MU dataset. 150 keywords queried u.a.r. from 500 $\star$ keywords in the $\diamond$ dataset.

VolAn and SelVolAn attacks on the M-MU Dataset. The recovery rates are similar to the C1 setting.

6 Takeaways

In this work we revisited leakage abuse and injection attacks against ESAs. In particular, we argued that the often-cited IKK and Count attacks are mostly of theoretical interest due to the following limitations and assumptions:

- *high known-data rates*: both the IKK and Count attack require high known-data rates to achieve reasonable recovery rates and it is not clear whether such rates are realistic;
- *known queries*: in addition to relying on known-data, the Count v.1 attack also relies on known queries;
- *suppressable leakage*: the IKK and Count attacks rely on the co-occurrence pattern which can be easily hidden at the cost of additional storage using our OPQ construction.
- *experimental evaluation*: the experimental evaluations of the IKK and Count attack were not



(a) $\star$ = high selectivity, $\diamond$ = partial

(b) $\star$ = high selectivity, $\diamond$ = entire

(c) $\star$ = low selectivity, $\diamond$ = partial

(d) $\star$ = low selectivity, $\diamond$ = entire

Figure 10: M-MU dataset. 150 keywords queried u.a.r. from 500 $\star$ keywords in the $\diamond$ dataset.

conducted in all settings of interest. This includes, for example, querying low-selectivity keywords or keywords that are not in the adversary's known dataset.

New attacks. To address these limitations, we introduced four new known-data attacks and two new injection attacks. We believe our known-data attacks are of practical interest since they work with low known-data rates and do not rely on any known queries. Most surprisingly, our attacks make use of only volumetric leakage and therefore apply, not only to structured ESAs, but also to oblivious ESAs.

We implemented our attacks and evaluated them empirically in various settings and using different kinds of queries. We hope that our study provides useful insights that may help the community better understand the real-world impact of leakage abuse attacks. In the following, we list our main takeaways, sometimes referencing the constructions described in Appendix A:

- None of the attacks worked against low-selectivity or pseudo-low-selectivity keywords;
- When querying high-selectivity keywords, the rid and vol patterns (which are respectively leaked by the BSL and SMI constructions) can be exploited by our Subgraph attacks with very low known-data rates (as low as 5%). Note that high recovery rates are maintained across different settings. In addition, we believe our attacks would do even better on larger data collections.

For example, on collections on the order of gigabytes we estimate that relatively high recovery rates could still be achieved with known-data rates as low as 1%. We conclude that, for high-selectivity keywords, there are *practical* attacks against leakage profiles that include rid and vol.

- When querying high-selectivity keywords, the total volume pattern (which is leaked by OPQ and FLL) seems resistant to our attacks for $\delta \leq .8$.
- The total volume pattern tvol can be successfully attacked with our injection attacks (though in the case of the Binary Search attack only if the target query has a unique total volume).
- Structured and oblivious ESAs seem to provide the same level of security against both our known-data (i.e., leakage abuse) and chosen-data (i.e., injection) attacks.

7 Countermeasures

Our study revealed two settings in which our attacks could be practical. The first is using our Subgraph attacks to exploit the rid and vol patterns on high-selectivity keywords and the second is using our volumetric injection attacks to exploit the total volume pattern. For all other settings, we do not believe any countermeasures are required though they are certainly available.

High-selectivity keywords. For high-selectivity keywords, one should simply use a scheme that does not leak rid or vol like the PBS construction of [45] (see Appendix A for a brief overview of PBS) or the OPQ or FLL constructions described in Appendix A. These schemes have the following leakage profiles

$$\Lambda_{\mathsf{PBS}} = \big(\star, (\mathsf{qeq}, \mathsf{svol})\big) \quad \text{and} \quad \Lambda_{\mathsf{OPQ}} = \big(\star, (\mathsf{keq}, \mathsf{tvol})\big)$$

and,

$$\Lambda_{\mathsf{FLL}} = \big(\star, (\mathsf{rlen}, \mathsf{tvol})\big),$$

where svol is the *sequence volume pattern*,

$$\mathsf{svol}(\mathbf{D}, w_1, \ldots, w_t) = \sum_{i=1}^{t} \sum_{D \in \mathbf{D}(w)} |\mathrm{D}|_{\mathbf{w}}.$$

Our experiments suggest that for $\delta \leq .8$ either OPQ or FLL can be used but that for $\delta > .8$ one should use PBS. Note that the sequence volume pattern seems to be a very "low leakage" pattern in the sense that even a "brute-force" attack that simply tries to match keywords to the sequence volume leakage does not work on our dataset. We provide more details below.

Brute force. We assume the adversary has full knowledge of the client's data, i.e., $\delta = 1$. Given the sequence volume pattern of a query sequence of length λ drawn uniformly at random from a set of 500 either low- or high-selectivity keywords, the attack finds all possible sequences of λ keywords that have sequence volume leakage equal to the given/observed leakage. If there is only a single such sequence, the attack returns it as its output otherwise it fails.

Note that this is the best possible attack against the sequence volume pattern (not taking efficiency into account). We define the attack's success rate as the fraction of its output that is

correct; that is, the number of keywords in its output sequence (assuming it outputs a sequence) that are indeed in the client sequence over the size of the sequence. We ran the attack for high- and low-selectivity keywords, with λ ranging from 1 to 6 and found that when $\lambda \geq 4$ the attack stopped working. More precisely, in the case of low-selectivity keywords its success rate was 0 and in the case of high-selectivity keywords it was 0.02. Figure 11 describes these results in detail.

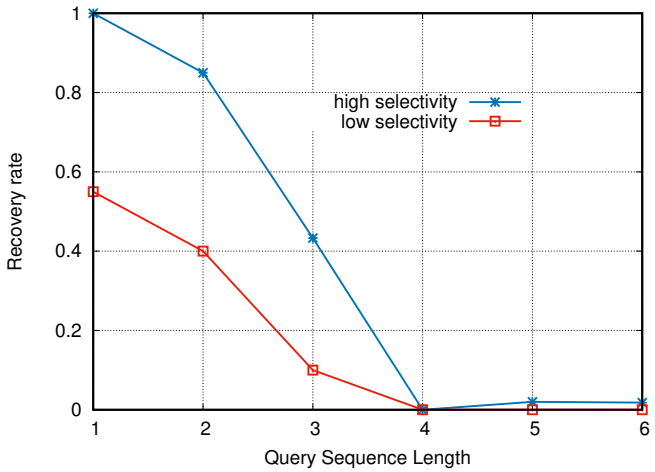


Figure 11: PBS brute-force attack.

Note that the success rate of the brute-force attack does not capture partial knowledge since it only accounts for the case where the attack finds a single "matching" sequence. For example, the success rate could be 0 even though the attack found just 2 matching sequences. To address this, we ran an additional experiment that computes the number of matching sequences found by the attack. The results are described in Figure 12 which shows that the average number of matches grows exponentially as a function of the sequence length (this holds independently of the selectivity of the keywords in the sequence). For example, for $\lambda = 5$, there are 385 matching sequences even when the keyword space space is as small as 100.[8]. Notice that increasing the number of keywords will significantly increase the number of matches.

Volumetric attacks. Recently, Kamara and Moataz [44] proposed the first volume-hiding encrypted multi-maps that do not rely on naíve padding. The two constructions, VLH and AVLH, achieve different trade-offs between storage efficiency, query efficiency and lossiness and can be used to protect against volumetric attacks. We refer the reader to Appendix A.

Another approach to protecting against volumetric attacks is to use padding techniques. Naive padding (adding dummy values to ensure the volume of every query response is of the same size) will protect against volumetric attacks but incurs a large storage overhead. More efficient padding techniques were proposed by Bost and Fouque [12]. While these techniques seem to make attacks harder, it is not clear if they can completely mitigate them.

[8] For this experiment we had to reduce the size of the keyword space from 500 to 100 keywords, because the former results in an extremely large number of sequences to check, i.e., $\binom{500}{6} \simeq 2 \cdot 10^{13}$

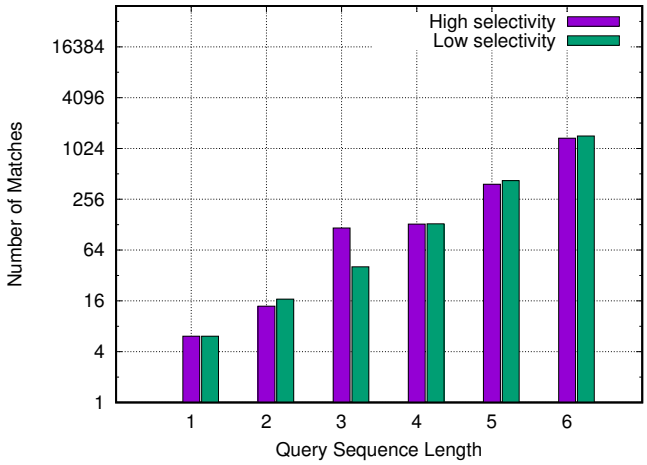


Figure 12: Number of matches in PBS brute-force attack.

Acknowledgment. We would like to thank the authors of [13] for answering our questions about the Count attack, for sharing the code of the Count attack with us, and for useful feedback on this work.

References

[1] Apache lucene. http://lucene.apache.org, 1999.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004.

[3] Ghous Amjad, Seny Kamara, and Tarik Moataz. Breach-resistant structured encryption. *Proceedings on Privacy Enhancing Technologies*, 2019(1):245–265, 2019.

[4] G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *ACM Symposium on Theory of Computing (STOC '16)*, STOC '16, pages 1101–1114, New York, NY, USA, 2016. ACM.

[5] Gilad Asharov, TH Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Oblivious computation with data locality. *ePrint IACR*, 2017.

[6] Jean-Philippe Aumasson. Cryptanalysis vs. reality. In *Black Hat (Abu Dhabi)*, 2011.

[7] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, Lecture Notes in Computer Science, pages 535–552. Springer, 2007.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, 2009.

[9] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference (TCC '11)*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[10] R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.

[11] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*, 2017.

[12] Raphael Bost and Pierre-Alain Fouque. Thwarting leakage abuse attacks against searchable encryption – a formal approach and applicaitons to database padding. Technical Report 2017/1060, IACR Cryptology ePrint Archive, 2017.

[13] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM Conference on Communications and Computer Security (CCS '15)*, pages 668–679. ACM, 2015.

[14] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[15] D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2014*, 2014.

[16] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. *IACR Cryptology ePrint Archive*, 2016:718, 2016.

[17] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[18] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[19] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[20] Jonathan L Dautrich Jr, Emil Stefanov, and Elaine Shi. Burst oram: Minimizing oram response times for bursty access patterns. In *USENIX Security Symposium*, pages 749–764, 2014.

[21] I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *ACM International Conference on Management of Data (SIGMOD '17)*, SIGMOD '17, pages 1053–1067, New York, NY, USA, 2017. ACM.

[22] F. Betül Durak, Thomas M. DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1155–1166. ACM, 2016.

[23] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(1):5–20, 2018.

[24] Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 395–410. IEEE Computer Society, 2015.

[25] Christopher W Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Freecursive oram:[nearly] free recursion and integrity verification for position-based oblivious ram. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 103–116. ACM, 2015.

[26] Christopher W Fletcher, Ling Ren, Albert Kwon, Marten Van Dijk, Emil Stefanov, Dimitrios Serpanos, and Srinivas Devadas. A low-latency, low-area hardware oblivious ram controller. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 215–222. IEEE, 2015.

[27] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Advances in Cryptology - CRYPTO 2016*, pages 563–592, 2016.

[28] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[29] Craig Gentry, Shai Halevi, Charanjit Jutla, and Mariana Raykova. Private database access with he-over-oram architecture. In *International Conference on Applied Cryptography and Network Security*, pages 172–191. Springer, 2015.

[30] E-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

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

[32] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why your encrypted database is not secure. In *Workshop on Hot Topics in Operating Systems (HotOS '17)*, pages 162–168, New York, NY, USA, 2017. ACM.

[33] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 315–331. ACM, 2018.

[34] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 655–672. IEEE Computer Society, 2017.

[35] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. Multi-key searchable encryption, revisited. In *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, pages 95–124, 2018.

[36] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Song. Shadowcrypt: Encrypted web applications for everyone. In *ACM Conference on Computer and Communications Security (CCS '14)*, 2014.

[37] Thang Hoang, Attila Altay Yavuz, and Jorge Guajardo. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 302–313. ACM, 2016.

[38] M. Saiful Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS '12)*, 2012.

[39] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security (CCS '13)*, pages 875–888, 2013.

[40] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sublinear complexity. In *Advances in Cryptology - EUROCRYPT '17*, 2017.

[41] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.

[42] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

[43] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, pages 149–180, 2018.

[44] Seny Kamara and Tarik Moataz. Encrypted multi-maps with computationally-secure leakage. In *EUROCRYPT '19*, 2019.

[45] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakae suppression. In *Advances in Cryptology - CRYPTO '18*, 2018.

[46] G. Kellaris, G. Kollios, K. Nissim, and A. O' Neill. Generic attacks on secure outsourced databases. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.

[47] E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in) security of hash-based oblivious ram and a new balancing scheme. In *ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 143–156, 2012.

[48] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[49] Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. Mimesis aegis: A mimicry privacy shield–a system's approach to data privacy on public cloud. In *USENIX Security Symposium (USENIX Security 14)*, pages 33–48, 2014.

[50] K. Lewi and D. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.

[51] Jacob R Lorch, James W Mickens, Bryan Parno, Mariana Raykova, and Joshua Schiffman. Toward practical private access to data centers via parallel oram. *IACR Cryptology ePrint Archive*, 2012:133, 2012.

[52] Jacob R Lorch, Bryan Parno, James W Mickens, Mariana Raykova, and Joshua Schiffman. Shroud: ensuring private access to large-scale data in the data center. In *FAST*, volume 2013, pages 199–213, 2013.

[53] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy and access control for outsourced personal records. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 341–358. IEEE, 2015.

[54] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM Conference on Computer and Communications Security (CCS 15)*, 2015.

[55] Ian Miers and Payman Mohassel. IO-DSSE: scaling dynamic searchable encryption to millions of indexes by improving locality. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[56] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM Conference on Computer and Communications Security (CCS)*, CCS '15, pages 644–655. ACM, 2015.

[57] NIST. Trec 2007 public corpus. https://plg.uwaterloo.ca/~gvcormac/treccorpus07/, 2018.

[58] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. The melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*, pages 556–567. Springer, 2014.

[59] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S.-G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[60] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. *IACR Cryptology ePrint Archive*, 2017:973, 2017.

[61] Martin Porter. Stop word - snowball. http://snowball.tartarus.org/, 2018.

[62] CALO Project. Enron email dataset. https://www.cs.cmu.edu/~enron, 2018.

[63] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten Van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious ram. In *USENIX Security Symposium*, pages 415–430, 2015.

[64] Ling Ren, Christopher W Fletcher, Xiangyao Yu, Marten Van Dijk, and Srinivas Devadas. Integrity verification for path oblivious-ram. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[65] Daniel S Roche, Adam Aviv, and Seung Geol Choi. A practical oblivious map data structure with secure deletion and history independence. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 178–197. IEEE, 2016.

[66] Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. Taostore: Overcoming asynchronicity in oblivious data storage. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 198–217. IEEE, 2016.

[67] E. Shi, T.-H. Chan, E. Stefanov, and M. Li. Oblivious ram with o((logn)¡sup¿3¡/sup¿) worst-case cost. In *Advances in Cryptology - ASIACRYPT '11*, pages 197–214. Springer-Verlag, 2011.

[68] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[69] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[70] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *ACM Conference on Computer and Communications Security (CCS '13)*, 2013.

[71] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 247–258. ACM, 2013.

[72] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious distributed cloud data store. In *NDSS*, 2013.

[73] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226. ACM, 2014.

[74] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium*, 2016.

A Overview of ESA Constructions

We recall some common ESA constructions based on both STE and ORAM.

Baseline (BSL). Let $\Sigma_{\mathsf{mm}} = (\mathsf{Setup}, \mathsf{Get})$ be a response-revealing multi-map encryption scheme and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme. Consider the ESA scheme $\mathsf{BSL} = (\mathsf{Setup}, \mathsf{Search})$ where each algorithm works as follows:

- $\mathsf{BSL.Setup}(1^k, \mathbf{D})$: it builds a multi-map MM that maps each keyword $w \in \mathbb{W}$ to the tuple $\mathbf{D}(w)$. It then computes $(K_1, \mathsf{EMM}) \leftarrow \Sigma_{\mathsf{mm}}.\mathsf{Setup}(1^k, \mathsf{MM})$ and, for all $i \in [n]$, $\mathrm{ct}_i \leftarrow \mathsf{Enc}(K_2, \mathrm{D}_i)$, where $K_2 \leftarrow \mathsf{SKE.Gen}(1^k)$. It outputs $(K, \mathbf{ED})$, where $K = (K_1, K_2)$ and $\mathbf{ED} = (\mathsf{EMM}, \mathrm{ct}_1, \ldots, \mathrm{ct}_n)$.
- $\mathsf{BSL.Search}\big(K, w; \mathbf{ED}\big)$: the parties execute $(\bot; \mathbf{I}) \leftarrow \Sigma_{\mathsf{mm}}.\mathsf{Get}(K_1, w; \mathsf{EMM})$ after which the server returns $(\mathrm{ct}_i)_{i \in \mathbf{I}}$ to the client. For all $i \in \mathbf{I}$, the client computes $\mathrm{D}_i := \mathsf{SKE.Dec}(K_2, \mathrm{ct}_i)$.

If the multi-map encryption scheme Σ_{mm} is instantiated with any of the standard constructions [19, 18, 17, 10, 11, 3], the SSE scheme will have leakage profile

$$\Lambda_{\mathsf{BSL}} = \big(\star, \big(\mathsf{qeq}, \mathsf{rid}, \mathsf{vol}\big)\big).$$

Its storage complexity will be

$$O\bigg(\sum_{w \in \mathbb{W}} \#\mathbf{D}(w) + \sum_{i=1}^{n} |\mathrm{D}_i|_{\mathsf{w}} \bigg),$$

and its search and communication complexity will be

$$O\bigg(\#\mathbf{D}(w) + \sum_{\mathrm{D} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}} \bigg),$$

which is optimal.

Opaque (OPQ). As far as we know, the construction we now describe has not appeared in prior work. It has a relatively low leakage profile and optimal search and communication complexity at the cost of additional storage. Let $\Sigma_{\mathsf{mm}} = (\mathsf{Setup}, \mathsf{Get})$ be a response-hiding multi-map encryption scheme and let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme. Consider the structured ESA scheme $\mathsf{OPQ} = (\mathsf{Setup}, \mathsf{Search}$ where each algorithm works as follows:

- $\mathsf{OPQ.Setup}(1^k, \mathbf{D})$: it builds a multi-map MM that maps each keyword $w \in \mathbb{W}$ to a tuple $\mathbf{t} = (t_1, \cdots, t_{a/B})$ composed of B-sized blocks, where t_i is the ith block of the concatenation of the documents (not identifiers) that contain w, and where

$$a = \sum_{\widetilde{\mathrm{D}} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}}.$$

  The algorithm then computes $(K, \mathsf{EMM}) \leftarrow \Sigma_{\mathsf{mm}}.\mathsf{Setup}(1^k, \mathsf{MM})$ and outputs $(K, \mathbf{ED})$ where $\mathbf{ED} = \mathsf{EMM}$.

- $\mathsf{OPQ.Search}\big(K, w; \mathbf{ED}\big)$: the parties execute

$$(\mathbf{t}; \perp) \leftarrow \Sigma_{\mathsf{mm}}.\mathsf{Get}(K, w; \mathsf{EMM})$$

and the client parses $\mathbf{t}$ as $(\mathrm{D})_{\mathrm{D} \in \mathbf{D}(w)}$.

If the multi-map encryption scheme Σ_{mm} is instantiated with a standard response-hiding encrypted multi-map [19, 18, 17, 10, 11][9] the ESA will have leakage profile

$$\Lambda_{\mathsf{OPQ}} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}) = \big(\star, \big(\mathsf{qeq}, \mathsf{tvol}\big)\big).$$

Its storage complexity will be

$$O\bigg(\sum_{w \in \mathbb{W}} \sum_{\mathrm{D} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}}\bigg),$$

and the search and communication complexity will be

$$O\bigg(\#\mathbf{D}(w) + \sum_{\mathrm{D} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}}\bigg),$$

which is optimal.

Semi-ORAM (SMI). Let $\mathsf{ORAM} = (\mathsf{Setup}, \mathsf{Read})$ be an ORAM scheme and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme. Consider the scheme $\mathsf{SMI} = (\mathsf{Setup}, \mathsf{Search})$ where each algorithm works as follows:

- $\mathsf{SMI.Setup}(1^k, \mathbf{D})$: it builds a multi-map MM that maps each keyword $w \in \mathbb{W}$ to the tuple $\mathbf{D}(w)$. It then computes $(K_1, \mathsf{OMM}) \leftarrow \mathsf{ORAM.Setup}(1^k, \mathsf{MM})$ and, for all $i \in [n]$, $\mathrm{ct}_i \leftarrow \mathsf{Enc}(K_2, \mathrm{D}_i)$, where $K_2 \leftarrow \mathsf{SKE.Gen}(1^k)$. It outputs $(K, \mathbf{ED})$, where $K = (K_1, K_2)$ and $\mathbf{OD} = (\mathsf{OMM}, \mathrm{ct}_1, \ldots, \mathrm{ct}_n)$.
- $\mathsf{SMI.Search}\big(K, w; \mathbf{OD}\big)$: the client uses ORAM to simulate an execution of $\mathsf{Get}(\mathsf{MM}, w)$; that is, it runs $\mathsf{Get}(\mathsf{MM}, w)$ locally but replaces every read operation to location i with an execution of $\mathsf{ORAM.Read}(K_1, i; \mathsf{OMM})$. At the end of this simulation, the client holds a set of indices $\mathbf{I}$ which it sends to the server. The server returns $(\mathrm{ct}_i)_{i \in \mathbf{I}}$ which the client decrypts.

If the ORAM scheme is instantiated with any standard construction [31, 47, 70], the search scheme will have leakage profile

$$\Lambda_{\mathsf{SMI}} = \big(\star, \big(\mathsf{rlen}, \mathsf{rid}, \mathsf{vol}\big)\big).$$

Using Path ORAM [70], the storage complexity is

$$O\bigg(\sum_{w \in \mathbb{W}} \#\mathbf{D}(w) + \sum_{i=1}^{n} |\mathrm{D}_i|_{\mathsf{w}}\bigg),$$

and the search and communication complexity are

$$O\bigg(\frac{\#\mathbf{D}(w)}{B} \cdot \log^2\bigg(\sum_{w \in \mathbb{W}} \frac{\#\mathbf{D}(w)}{B}\bigg) + \sum_{\mathrm{D} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}}\bigg),$$

where B is the block size in bits.

[9] We note that while most of these constructions are described as response-revealing constructions it is trivial to convert them to response-hiding schemes.

Full ORAM (FLL). Let $\mathsf{ORAM} = (\mathsf{Setup}, \mathsf{Read})$ be an ORAM scheme and consider the scheme $\mathsf{FLL} = (\mathsf{Setup}, \mathsf{Search})$ where each algorithm works as follows:

- $\mathsf{FLL.Setup}(1^k, \mathbf{D})$: it builds an array RAM that stores all the documents in $\mathbf{D}$. It then builds a multi-map MM that maps each keyword $w \in \mathbb{W}$ to the locations of the blocks in RAM that store the documents in $\mathbf{D}(w)$. It then computes $(K_1, \mathsf{OMM}) \leftarrow \mathsf{ORAM.Setup}(1^k, \mathsf{MM})$ and $(K_2, \mathsf{ORAM}) \leftarrow \mathsf{ORAM.Setup}(1^k, \mathsf{RAM})$. It outputs $(K, \mathbf{OD})$, where $K = (K_1, K_2)$ and $\mathbf{OD} = (\mathsf{OMM}, \mathsf{ORAM})$.
- $\mathsf{FLL.Search}\big(K, w; \mathbf{OD}\big)$: the client uses ORAM to simulate an execution of $\mathsf{Get}(\mathsf{MM}, w)$. At the end of this simulation, the client holds a set of indices $\mathbf{I}$. It then uses ORAM again to simulate, for all $i \in \mathbf{I}$, an execution of $\mathsf{Read}(\mathsf{RAM}, i)$ to recover the documents.

If the ORAM scheme is instantiated with any standard construction [31, 47, 70], the search scheme will have leakage profile

$$\Lambda_{\mathsf{FLL}} = \big(\star, \big(\mathsf{rlen}, \mathsf{tvol}\big)\big).$$

The storage complexity is

$$O\Big(\sum_{w \in \mathbb{W}} \#\mathbf{D}(w) + \sum_{\mathrm{D} \in \mathbf{D}(w)} |\mathrm{D}|_{\mathsf{w}}\Big),$$

and the search and communication complexity are

$$O\Big(\frac{\#\mathbf{D}(w)}{B_1} \cdot \log^2\Big(\sum_{w \in \mathbb{W}} \frac{\#\mathbf{D}(w)}{B_1}\Big) + \sum_{i \in \mathbf{D}(w)} \frac{|\mathrm{D}_i|_{\mathsf{w}}}{B_2} \cdot \log^2\Big(\sum_{i=1}^{n} \frac{|\mathrm{D}_i|_{\mathsf{w}}}{B_2}\Big)\Big)$$

where B_1 and B_2 are the block sizes in bits of the first and second ORAM, respectively. Note that this construction has leakage profile Λ_{FLL} only if the client retrieves all of the matching documents from the second ORAM at once. If, on the other hand, the client retrieves them one by one then it will have leakage profile $\big(\star, \big(\mathsf{rlen}, \mathsf{vol}\big)\big)$.

Additional ORAM-based constructions. We note that there are alternative ORAM-based designs in addition to the ones we described above. One could, for example, merge the two ORAMs used in the full ORAM simulation into a single ORAM with the same block size. This would have leakage pattern $\big(\star, \mathsf{tvol}\big)$.

The Piggyback scheme (PBS). PBS is an STE scheme recently introduced in [45] that partially hides the volume pattern. It comes in two variants. The first reveals only the sequence volume pattern (i.e., the sum of the volume associated to a query sequence) on non-repeating query sequences. The second variant reveals nothing (beyond a public parameter independent of the volume) on non-repeating query sequences. At a high level, the scheme leverages a new trade-off in STE design; specifically, it trades latency for an improved leakage profile. At a high level, the scheme processes the input data structure such that the query responses are divided into smaller chunks of equal size. These chunks are then stored and encrypted so that, on each query, the client only retrieves a fixed number of chunks. If the whole response is not retrieved at that moment,

then the query is added to a queue and the remaining chunks are retrieved on the next query. The responses can therefore be delayed but the authors show that the delay can be minimal for standard query distributions.

Volume-hiding constructions. VLH and AVLH are volume-hiding encrypted multi-map constructions recently introduced by Kamara and Moataz [44]. These schemes are the first volume-hiding STE constructions that do not rely on naïve padding. VLH makes use of a pseudo-random function F and an optimal multi-map encryption scheme. It is parameterized with a public parameter $\lambda \geq 1$ that affects correctness. Given a multi-map MM, the scheme determines a new response length for each label ℓ in MM which is computed by evaluating F on ℓ's original response length and adding λ. If the new response length is larger than the original, then ℓ's tuple is padded. If the new response length is smaller than the original, then ℓ's tuple is truncated. AVLH is a more advanced construction based on a new design paradigm based on bi-partite graphs. More precisely, AVLH transforms its input multi-map as a bi-partite graph where top vertices correspond to the multi-map's labels and the bottom vertices correspond to bins. Each label's tuple values are then stored in its associated bin in a specific way. The bins are then padded to have the same size. At query time, the user always retrieves the same number of bins. AVLH does not improve on the query complexity of encrypted multi-map schemes but does improve on the storage efficiency of naive padding. In [44] it is then shown that the storage can be further reduced by relying on the conjectured hardness of the planted densest subgraph problem.

B Count v.1 with $\delta < 1$

The Count v.1 attack was shown in [13] to have high recovery rate when $\delta = 1$; that is, when the adversary has full knowledge of the data. For $\delta < 1$, however, the attack seems to only work if $\delta \leq .8$. We found that the experimental results for $\delta < 1$ that are reported in [13], however, are for an unpublished variant of the count attack that relies on knowledge of client queries. To better understand how known queries impact the recovery rate of Count v.1, we evaluated the attack with a varying fraction of known queries. The results are shown in Figure 13.[10] When the adversary knows 5% of the queries, recovery rates are similar to the ones reported in [13]. When the adversary knows 2% known queries, however, the attack ceases to work even with $\delta = .9$.

C Keyword Selectivity

Our empirical evaluation (see Section 5) clearly shows that the selectivity of the queries is by far the most important factor on the recovery rate of all the attacks. Understanding the selectivity of keywords in our dataset is therefore important. In Figures 14 (a) and (b) we plot the selectivity of 1000 and $10,000$ most selective keywords, respectively, in our datasets after stemming and removing stop words. We can see in these Figures that keyword selectivity in `Enron` is power law distributed. In other words, only a few keywords have high and unique selectivities whereas the overwhelming majority of keywords have low and common selectivities (at most 3).

[10] This experiment was performed using the implementation and dataset of [13]. We thank the authors for promptly sharing their implementation and data with us.

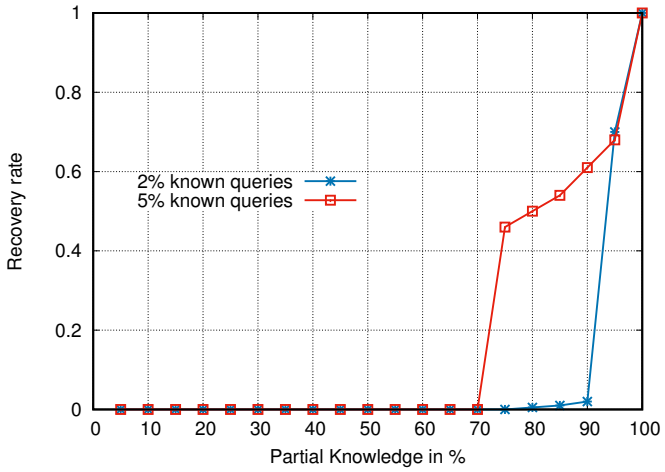
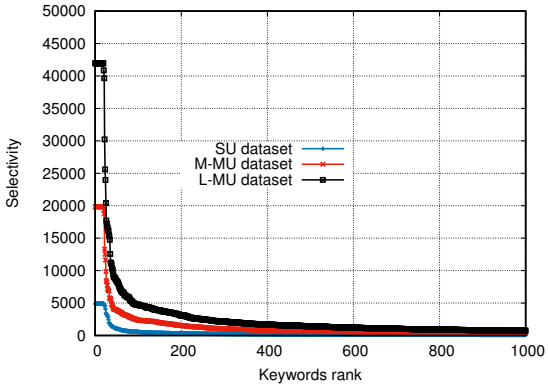
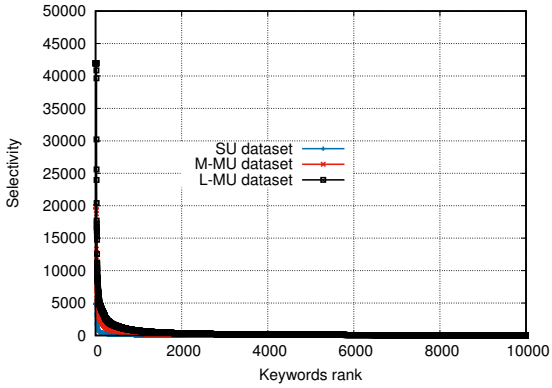


Figure 13: Count v.1 with varying fractions of known queries (on 150 queries out of a keywords space of size 500).



(a) Distribution of the most selective 1000 keywords.

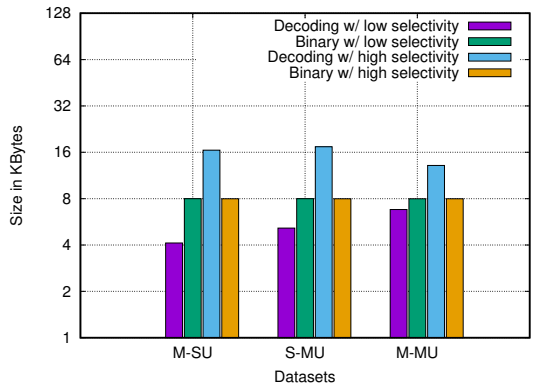


(b) Distribution of the most selective $10,000$ keywords.

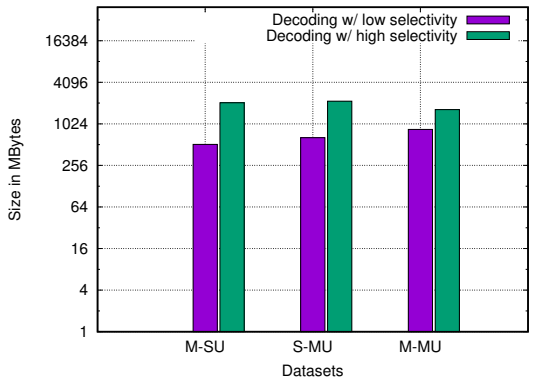
Figure 14: Keyword selectivity.

D Quantifying the Offset for Injection

The total number of files injected by both the Decoding and Binary Search attacks depend on an offset γ which is determined by characteristics of the data collection. Here, we study the values of these offsets on three different collections: SU, S-MU and M-MU as defined in Section 5. Our results are described in Figure 15. We found that querying on high- or low-selectivity keywords did not have any impact on the Binary Search attack. However, as can be seen from its description, the size of the keyword space did have an impact. For the Decoding attack, the amount of injected data did depend on the selectivity of the queries: the amount for high-selectivity queries was about twice as much as for low-selectivity queries. This held for both the SU and S-MU datasets. We believe that this is inherent to the way the offset is computed. In fact, on high-selectivity queries, we noticed that the total volumes tend to have a higher gap between them. This is not the case for low-selectivity queries.



(a) Amount of injected data to recover one keyword.



(b) Amount of injected data to recover 500 keywords.

Figure 15: Amount of injected data for both the Decoding and Binary Search attacks (with $\#\mathbb{W} = 500$).