Subliminal Encrypted Multi-Maps and Black-Box Leakage Absorption

Amine Bahi*
École Normale Supérieure

Seny Kamara†
MongoDB & Brown University

Tarik Moataz‡
MongoDB

Guevara Noubir§
Northeastern University

Abstract

We propose a dynamic, low-latency encrypted multi-map (EMM) that operates in two modes: *low-leakage mode*, which reveals minimal information such as data size, expected response length, and query arrival rate; and *subliminal mode*, which reveals only the data size while hiding metadata including query and update times, the number of operations executed, and even whether an operation was executed at all—albeit at the cost of full correctness. We achieve this by exploiting a tradeoff between leakage and latency, a previously underexplored resource in EMM design. In low-leakage mode, our construction improves upon existing work both asymptotically and empirically: it achieves optimal server-side storage, as well as communication and computational complexity that is independent of the maximum response length. In subliminal mode, it is the first construction to hide metadata.

To analyze the latency and client-side storage of our construction, we utilize queuing theory and introduce a new queuing model, which may be of independent interest. To examine its metadata-hiding properties, we extend standard security definitions to account for metadata and prove a surprising result: if a scheme is subliminal in that it hides the execution of its operations, then it absorbs the leakage of any scheme that makes black-box use of it without sending additional messages. In other words, if a scheme is subliminal, then any scheme that makes black-box use of it will also be subliminal.

We implement and evaluate our construction, demonstrating that our empirical results align with our theoretical analysis and that the scheme achieves a median query latency below 10 milliseconds, making it practical for some applications.

*amine.bahi@ens.psl.eu. Work done in part while at Northeastern University and Mohammed VI Polytechnic University.

†seny.kamara@mongodb.com.

‡tarik.moataz@mongodb.com.

§g.noubir@northeastern.edu .

Contents

1 Introduction

Encrypted search algorithms (ESA) are cryptographic primitives that allow one to encrypt their data while preserving the ability to search/query over it. ESAs have several applications but the most important is the design of end-to-end encrypted databases. Encrypted databases increase the security and privacy of data by providing "encryption in use"; that is, data remains encrypted even while the database management system processes it. ESAs can be designed using a variety of (sometimes overlapping) cryptographic primitives each leading to solutions that achieve different tradeoffs between efficiency, security and expressiveness. The security of an ESA is captured by its leakage profile which describes the information it reveals about the encrypted data and queries. A leakage profile is itself composed of leakage patterns which are functions that map the data and queries to some observed leakage. Common leakage patterns include size (i.e., the size of the dataset), frequency (i.e., the number of occurrences of a data item), query equality (i.e., whether two queries are for the same item) and response length or volume (i.e., the length of a query's response).

Leakage regimes. To help our discussion, we will categorize ESAs into the four following regimes:

- *high-leakage*: solutions that reveal non-trivial information given only access to the encrypted data;
- *mid-leakage*: solutions that reveal minimal information (e.g., size) given access to the encrypted data but can reveal non-trivial information about queries and updates given access to query and update transcripts. Some notable examples include the Z-IDX construction of [30], the scheme presented in [22], the SSE-1 and SSE-2 constructions of [24] and the π_{bas} construction of [19]. Mid-leakage ESAs are usually considered practical if their communication complexity is $O(d)$ and their computational complexity is either $O(d)$ or $O(d + \log(N))$, where d is the number of data items that need to be returned for a query and N is the size of the entire dataset.
- *low-leakage*: solutions that reveal minimal information given both the encrypted data and query and update transcripts. Notable examples include the construction that underlies the TWORAM scheme of [28] which we refer to as twoEMM, the AZL scheme of [45] and the zeroSSE construction of [7]. [1] All currently-known low-leakage ESAs, with the exception of AZL, have $\Omega(m)$ communication and computational complexities, where m is the maximum response length over all queries. Intuitively speaking, this seems to stem from the fact that low-leakage solutions need to hide the query equality and volume patterns.
- *zero-leakage*: solutions that hide everything about the data and queries beyond the size of the data structure. The only example of a zero-leakage scheme we are aware of is the FZL construction of [45] which is only *partially* correct in the sense that it cannot guarantee that the full response to a query will be returned within a bounded amount of time.

[1] Though the scheme is called zeroSSE it is a multi-map encryption scheme and not an SSE scheme; though, of course, it can be used as a building block to construct an SSE scheme. It is also not zero-leakage in the sense of [45] since it leaks the maximum response length.

We stress that these categories are an over-simplification of ESA security and that whether a leakage pattern is minimal or not is depends on the context in which the solution is used.[2] But this rough categorization approximately captures how ESA designers think about constructions and will be helpful for our broader discussion. Typically, mid-leakage ESAs are based on structured encryption and low- and zero-leakage ESAs use a combination of structured encryption and oblivious RAM (ORAM) techniques. We note, however, that the distinction between structured encryption and ORAM is not clearly defined. As mentioned in [45], ORAM can also be viewed as a zero-leakage array encryption scheme from the perspective of structured encryption. Similarly, a zero-leakage encrypted multi-map (EMM) or even a zero-leakage encrypted dicitonary (EDX) can be trivially used as an ORAM. This, in part, explains why ORAM and STE techniques seem to combine naturally as illustrated in several works [28, 45, 26, 29, 7].

Encrypted multi-maps. While there are several kinds of structured encryption schemes, multi-map encryption schemes are the most studied because EMMs are a core building block for encrypted relational databases [42, 46, 20, 35], encrypted NoSQL databases [44] and optimal-time searchable symmetric encryption (SSE) schemes [24, 17, 19]. A multi-map data structure stores a set of label/tuple pairs and supports get and possibly put operations. A get on a label returns its associated tuple and a put adds a new label/tuple pair. Multi-map encryption schemes are structured encryption schemes customized to encrypt multi-maps and to efficiently support get and put operations.

Latency as a resource. The first low- and zero-leakage EMMs that did not rely on black-box ORAM simulation and that achieve better asymptotic performance than existing low-leakage EMMs for certain natural workloads were presented in [45]. This was achieved in part by taking advantage of a new tradeoff between leakage and query *latency*, a resource that had not been explicitly considered before. Two static constructions are proposed, AZL and FZL, that leak at query time the total response length (i.e., the sum of query response lengths) and nothing, respectively. Their worst-case latency is $q \cdot (m/\alpha - 1)$, where q is the number of queries made so far and $\alpha > 1$ is a parameter of the scheme that affects the latency. In addition, their worst-case communication and computation complexity is $O((\mathsf{opt} + \alpha) \cdot \log^2 N)$.[3] While the authors show that the latency improves for specific query and multi-map distributions, in [45], latency is quantified as the number of queries that must be processed before the query under consideration can be executed. Here, we will refer to this as the *countable* latency and note that it is related to but different than the more standard notion of latency quantified by time which we will refer to as *chronometric* latency.

Chronometric latency in the stochastic setting. Countable latency is an interesting metric but it does not capture all the intuition and subtleties of chronometric latency. Furthermore, the relationship between countable and chronometric latency could be non-linear. For example, if a scheme is used to process queries that are sampled at regular intervals then the chronometric latency will be a linear function of the scheme's countable latency. If, however, the queries are sampled at arbitrary time intervals then the relationship could be non-linear. To better understand

[2] Whether a leakage profile is exploitable or not depends on several factors including the distributions from which the data and queries are sampled from, the auxiliary information the adversary has about them and the number of queries executed. For a summary of practical leakage attacks see [41] and for a theoretical perspective see [43].

[3] Note that while FZL is zero-leakage, unlike the other constructions, it is correct only under restricted parameter choices.

chronometric latency we extend the setting in which encrypted search algorithms are normally studied. In particular, we assume that queries are sampled from a stochastic process rather than from a simple query distribution. A stochastic process $\{X(t) : t \in T\}$ is a collection of random variables indexed by a set T. If $T \subseteq \mathbb{R}_{\geq 0}$ then it is a continuous-time stochastic process and the index set T can be interpreted as time. Based on the query process, we can define other useful processes including the arrival and interarrival processes, which determine the number of queries per time unit and the time elapsed between queries, respectively. Throughout, we will refer to this analytical framework as the *stochastic setting*.

With this framework in place, we can now revisit our discussion and restate things more precisely (but still informally). In particular, the chronometric latency of a scheme is a function of both its countable latency *and* of the query *interarrival* times. This explains why the chronometric latency can be a non-linear function of the countable latency and also points to a limitation of the AZL and FZL constructions of [45] which is that their chronometric latency is *always* a function of the query interarrival time. To see why, we first recall how the schemes work at a high level.[4] Roughly speaking, they transform the input multi-map to a dictionary by replacing every multi-map label/tuple pair $(\ell, \mathbf{v})$, where $\mathbf{v} = (v_1, \ldots, v_n)$, by a set of label/value pairs $\{(\ell \| i, v_i) : i \in [n + p]\}$, where $p = \alpha - (n \mod \alpha)$ and, for all $n < i \leq n + p$, v_i is a dummy element. They then store the label/value pairs in a dictionary and encrypt it using a dictionary encryption scheme. To query for a multi-map label ℓ_1, they query the encrypted dictionary for labels $\ell_1 \| 1, \ldots, \ell \| \alpha$. They then wait until the next multi-map query ℓ_2 (which could be such that $\ell_2 \neq \ell_1$), add ℓ_2 to a local queue and query the encrypted dictionary on $\ell_1 \| \alpha + 1, \ldots, \ell \| 2\alpha$. And so on $(n + p)/\alpha$ times after which it dequeues a label from its queue and processes it in the same manner. One can see that these schemes are *event-driven* in the sense that they can only make progress on responding to a query (with response length larger than α) when future queries occur. In particular, if the query interarrival times are large, then the chronometric latency could be large even if the countable latency is small.

A practical low-leakage EMM. Based on this discussion, it follows that [45] left open the problem of designing low- and zero-leakage EMMs with good *chronometric* latency in the sense that, under natural conditions, it does not depend on the query interarrival time. In this work, we address this problem but, in doing so, also uncover and achieve significantly stronger security guarantees than previously known. Specifically, we propose a new dynamic construction we refer to as EXH that can be used in two modes: a low-leakage mode that achieves full correctness; and a new security mode we refer to as *subliminal mode* that offers considerably stronger security than zero-leakage, also at the expense of full correctness. In low-leakage mode, EXH provides good chronometric latency, optimal server-side storage, small client-side storage and small communication and computational complexity. EXH achieves this not only asymptotically, but empirically as well. In fact, as we show in Section 7, in low-leakage mode the scheme can be considered practical.

Subliminal vs. zero-leakage. As mentioned above, our construction can also be used in a stronger subliminal mode. Intuitively, the difference between zero-leakage solutions like FZL and subliminal solutions like EXH is that the former leak metadata such as the number of operations

[4] For the sake of simplicity, we only describe the piggy-back scheme (PBS) that underlies the AZL and FZL constructions; and we disregard describing the cache-based and rebuild compilers.

executed and their execution times[5] whereas the latter do not. In fact, we will show that when used in subliminal mode, our EXH construction even hides the execution of the operations themselves, though it reveals some setup or initiation information.

This leads to two natural questions: (1) how can metadata be hidden, given that the server needs to receive and respond to operations? and (2) how can the hiding of metadata be formalized? To address the first question, at a high level, EXH hides metadata by sending a sequence of real and dummy operations on a schedule that is independent of the real operations. One way to view this is that the scheme creates its own steganographic covertext, allowing it to conceal operations within it. The second question is also non-trivial because, as we will see in Section 4, standard definitions in the real/ideal-world paradigm do not capture metadata hiding. Therefore, to properly analyze the security of our scheme, we will need to extend these definitions.

1.1 Our Contributions

In this work, we consider the problem of designing low-leakage and subliminal EMMs with chronometric latency that does not depend on query interarrival times. Towards this end, we make several contributions.

Defining subliminal executions. To study the security of our construction and, in particular, how it hides metadata, we extend the standard notions of security for structured encryption in several respects. First, we consider adversaries that have access to a global clock. This alone, however, does not allow us to capture the subliminal properties of a scheme. To see why, recall that the ideal/real-world security definition for structured encryption requires the existence of a simulator that can simulate the encrypted structure and the operation executions in such a way that the adversary cannot distinguish them from real ones, even for adaptively-chosen operations.

The issue with this is that even the simple activation of the simulator can reveal to it metadata like the time and rank of the operation which, as we will see in Section 4, means that a simulator could make use of this information in a proof for a scheme that is metadata revealing. To addresss this, we need to carefully design new security experiments that are sound with respect to metadata hiding protocols. We show how to do this for: execution time, execution count and whether an operation was executed at all. We then show that if a scheme is execution-hiding then it also hides execution times and execution counts and in such a case we say that it is *subliminal*.

Absorption and black-box design. With our new definitions and models in place, we also show an interesting and powerful leakage suppression property of subliminal constructions we refer to as *absorption*. Specifically, we show that if a scheme is subliminal, then any scheme that makes black-box use of it (without sending additional messages) becomes subliminal. We believe our Theorem has interesting consequences for the design of ESAs since it provides a relatively simple way to design metadata-hiding—and, consequently, zero-leakage—schemes by repeatedly building on top of simpler subliminal building blocks. In fact, our absorption Theorem justifies the practice advocated by certain works [36, 42, 45, 29] of designing and analyzing ESAs in a black-box manner so that the resulting schemes can benefit from improvements in leakage suppression techniques and low-leakage constructions.

[5] To see why FZL reveals this information, consider the first multi-map query ℓ_1 and note that the time at which the server receives the first α dictionary queries corresponds to when ℓ_1 was issued. Similarly, the time it receives the second batch of α dictionary queries corresponds to the time at which the second multi-map query was issued.

Our construction. Our EXH construction is simple and relies on a query-equality-hiding encrypted dictionary which we can build using ORAM. Given a multi-map, the client transforms it into label/value pairs of the form $(\ell \| i, v_i)$, where v_i is the ith element of ℓ's tuple. It then adds a dummy label/value pair, creates a local structure that maps labels to their tuple size and encrypts the dictionary before sending it to the server. At this stage, the client creates two local queues, one for queries and another for updates, and starts two parallel processes that do the following in a continuous loop: the query process samples a value δ from a distribution $\mathcal{D}_{\mathsf{Q}}$; waits δ steps; if the query queue is empty, it queries the dummy label; otherwise it queries the dequeued label. Similarly, the update process samples a value δ from a distribution $\mathcal{D}_{\mathsf{U}}$; waits δ steps; if the update queue is empty, it puts a dummy label/value pair. To get a label ℓ, the client retrieves the size of ℓ's tuple, n_ℓ, from the local structure and adds $\ell \| 1, \ldots, \ell \| n_\ell$ to the queue. To add or delete a label/tuple pair $(\ell, \mathbf{v})$, EXH adds $\mathsf{op} \| \ell \| v_1, \ldots, \mathsf{op} \| \ell \| v_{\#\mathbf{v}}$ to the update queue where $\mathsf{op} \in \{\mathsf{pairAdd}, \mathsf{pairDel}\}$. The scheme's communication and computational complexities for both queries and updates are

$$O\left(\frac{\mu}{\lambda} \cdot \log N\right),$$

where $\lambda \in \mathbb{R}_{>0}$ is the arrival rate that controls the rate at which operation are generated and $\mu \in \mathbb{R}_{>0}$ is the service rate that controls the rate at which the operations are processed. Furthermore, the server-side storage is $O(N)$, which is optimal. Note that while one can use EXH with arbitrary values of λ and μ in subliminal mode, full correctness can only be obtained when the size of the queues is bounded and does not diverge. For get operations, for instance, this occurs when $\lambda \cdot \mathrm{E}[R] < \mu$, where R is a random variable that determines the tuple lengths of the input multi-map and μ satisfies the stability condition which is a function of the arrival rate λ. Because of this, the scheme cannot be subliminal and fully correct. The leakage in low-leakage mode, however, is minimal and consists of the arrival rate λ and the expected response length $\mathrm{E}[R]$. The communication and computational complexity are $O\left(\mathrm{E}[R] \cdot \log N\right)$, where $\mathrm{E}[R] \cdot \log N = o(m)$ in most practical settings.[6] This is a non-trivial improvement over all previously-known dynamic low-leakage EMMs which all achieve $\Omega(m)$, where m is the maximum tuple length.

Latency & stash analysis. To analyze the latency and client-side storage of our construction, we make use of techniques from queuing theory. We model the client's queue as a variant of a batch arrival queuing system where the arrival process is a compound Poisson process with rate λ and the servicing process is exponentially distributed with rate μ and bound the scheme's chronometric latency, client-side storage and communication complexity as a function of λ and μ. Batch arrival systems model situations where items can arrive in batches which is the case for both EXH's queues since the client either inserts n_ℓ dictionary labels per multi-map query label ℓ, or, inserts $\#\mathbf{v}$ dictionary pairs per multi-map label/tuple pair $(\ell, \mathbf{v})$. An important technical difference between traditional batch arrival systems and EXH, however, is that with our scheme the server always processes something; even if the client's queue is empty (in which case it will process a dummy query or a dummy put). This seemingly minor difference changes the analysis of the system considerably. To address this, we introduce a new queuing model we call *batch arrival with dummies* and analyze it under different conditions. With this new model, we are able to show that

[6] The maximum response length of the `Enron` dataset behaves asymptotically as $O(N^{0.58})$ whereas the expected response length behaves as $O(N^{0.18})$, refer to Section 6.5 for more details.

EXH's chronometric query latency, is upper bounded by

$$O\Big(\frac{\mathrm{E}[R^2]}{\mathrm{E}[R]\cdot(\mu-\lambda\cdot\mathrm{E}[R])}\Big).$$

At this stage, the key thing to notice is that if the arrival rate λ is constant and the servicing rate μ tends to ∞ then the chronometric latency tends to 0. More intuitively, this guarantees that if the servicing rate is large enough to handle the arriving queries, then the chronometric latency will be small even if the arrival rate is large. The chronometric update latency has the same form except that we replace the random variable R by U which determines the length of the update tuples. In addition, we show that the size of EXH's query queue is

$$O\Big(\frac{\lambda\cdot\mathrm{E}[R^2]}{\mu-\lambda\cdot\mathrm{E}[R])}\Big),$$

which can be made very small with large enough servicing rates. In particular, with high probability, the queue does not diverge as long as $\lambda\cdot\mathrm{E}[R]<\mu$ which is the *stability condition*. Note that the stability condition is always satisfied when EXH is used in low-leakage mode, but not necessarily when it is used in subliminalmode since the system parameters are picked arbitrarily. The size of the update queue has the same form except that we replace the random variable R by U. As a concrete example, if the arrival rate is set to $\lambda=10$, the service rate to $\mu=40$ and if the tuple length distribution R is geometrically-distributed with parameter $p=0.1$, then the client's queue will hold 2 elements.

Application to ORAM. We note that multi-maps can simulate arrays with no query amplification so an EMM results trivially in an encrypted array. As observed in [45], an oblivious RAM (ORAM) is an encrypted array that is zero-leakage in the sense that it guarantees that reads and writes leak nothing about the read index and the write value and index, respectively. Since EXH can trivially simulate an array, it follows that it results in a subliminal ORAM.

Experimental evaluation. We implement and empirically evaluate our construction and show that our empirical results are aligned with our theoretical analysis and that it scales. We run EXH on multi-maps of sizes from 2^{12} to 2^{20} and evaluate the communication complexity, storage, latency and queue size after performing 10^5 queries. We also varied the arrival rates to 32, 64 and 128 queries per second following a Poisson process. For $N=2^{20}$, the communication complexity of a query is 958KB and 243KB for $\lambda=32$ and $\lambda=128$, respectively. The median latency is below 1 millisecond for both $\lambda=32$ and $\lambda=64$, and below 10 milliseconds for $\lambda=128$. The median queue size was zero for both $\lambda=32$ and $\lambda=64$ and 6 when $\lambda=128$. The worst-case latency and queue length for all arrival rates and all sizes of the multi-map was 1.82 seconds and $9,890$ elements, respectively.

A note on the Poisson process. Our use of the Poisson process to model arrival distributions is not arbitrary. Poisson processes are known to capture many naturally occurring events, including, the number of times a web server is accessed, the number of calls arriving at a call center, the number of emails sent per client, or the number of insurance claims filed during a fixed period of time, among many others [57]. Since it is challenging to get access to real query logs, there are numerous papers that model, e.g., web traffic as a Poisson process [15, 47]. Poisson processes

are also a good model for aggregate traffic of the kind that results from multiple clients querying the same encrypted database. More precisely, when n independent and identically distributed proccesses are aggregated then they result in a Poisson process with rate equal to the sum of the rates of the individual processes [8]. Note, however, that modeling query arrival times as Poisson processes results in queries being independent. We believe, however, that this is a natural and necessary first step since, in order to move towards more complex stochastic processes, one has to first understand how the system behaves in the independent setting. Finally, it is important to note that EXH is agnostic to the query, servicing and multi-map distributions and can be analyzed in any queuing model.

New directions and future work. Our work introduces several new directions of inquiry not only in encrypted search but also in cryptography in general. These include:

- *(closure for black-box constructions)* the absorption property we demonstrate for subliminal constructions represents a form of security closure under black-box use. However, subliminal execution is an extremely strong security guarantee that may not be necessary in many settings. This raises the question of whether there are other security properties that are closed under black-box use.

- *(composition)* our absorption theorem holds when the calling construction uses a single execution-hiding building block. Extending our result to constructions that use multiple execution-hiding components would be interesting. Additionally, constructing execution-hiding schemes that achieve standard composition notions like Universal Composability (UC) [14] would also be valuable.

- *(correctness)* all the zero-leakage and subliminal constructions we know of so far do not achieve full correctness; that is, they cannot guarantee that every query will be answered within a bounded amount of time. Demonstrating the existence of fully correct zero-leakage and subliminal constructions or proving that such constructions cannot exist would be of great interest.

- *(latency)* EXH, along with FZL and AZL, demonstrates that latency can be a useful resource to trade off for decreased leakage. Exploring this trade-off further by designing new low-latency constructions and by establishing latency versus leakage lower bounds would be very interesting.

- *(subliminal protocols)* while we demonstrate the possibility of metadata hiding in the context of encrypted multi-maps, the notion of subliminal protocols extends naturally to private information retrieval (PIR), Secure Multi-Party Computation (MPC). A subliminal PIR scheme would protect, in addition to the query, the time, number, or even the execution of the queries. Subliminal (reactive) MPC protocols could work as follows: in cases where only honest parties receive outputs, the protocol would hide—in addition to all partial information about the inputs beyond what can be inferred from the output—the time, number, or even the execution of the protocol. In the case where corrupted parties receive outputs, the protocol would reveal, e.g., at most that an execution occurred in the past. We stress that these are preliminary ideas meant to motivate new directions and leave it to future work to explore them in detail.

2 Related Work

Structured encryption (STE) was proposed by Chase and Kamara [23] as a generalization of index-based searchable symmetric encryption (SSE) [59, 24]. While SSE schemes encrypt unstructured data collections in such a way that they can support keyword search (i.e., return all the documents that contain a given keyword), STE schemes encrypt data structures in such a way that they can be privately queried. Special types of STE schemes include dictionary and multi-map encryption schemes which constitute the main building blocks in the design of optimal-time and sub-linear SSE schemes [24, 39, 19], efficient encrypted relational databases [42, 46, 66, 65, 20, 35], and efficient encrypted non-relational databases [43]. STE schemes have been improved along various dimensions including expressiveness [17, 36, 55, 27], efficiency [24, 19], dynamism [39, 38], security guarantees [60, 12, 37, 56, 9, 4], I/O efficiency, locality and more recently page efficiency [18, 5, 25, 11].

Leakage (crypt)analysis. The leakage of STE schemes has been studied empirically through cryptanalysis and, more recently, theoretically through various analytical frameworks. The leakage patterns that have been cryptanalyzed include the query equality pattern [52, 54, 40], the response identity pattern [34, 16, 64, 49, 10, 33] and the volume pattern [32, 10] under a variety of assumptions and distributions. Attempts to formalize and study leakage have also been made including the Bayesian leakage analysis framework of [44], the leakage inversion framework of [48] and the biased coin framework of [63].

Leakage suppression and low-leakage constructions. In [45], Kamara, Moataz and Ohrimenko introduced leakage suppression, which focuses on (general) techniques to reduce the leakage of STE constructions. In this work, they showed how Goldreich and Ostrovsky's square-root ORAM construction [31] can be interpreted as an instance of query equality suppression and generalized and applied the techniques to suppress the query equality leakage of static encrypted multi-maps. The authors also propose the AZL and FZL constructions discussed above. George, Kamara and Moataz [29] extended this initial framework to work in the dynamic setting. In addition to the schemes that result from general leakage suppression techniques, several concrete low-leakage constructions are known [28, 7] which we discussed in the introduction.

Covert computation. In [62], Von Ahn, Hopper and Langford introduce the notion of covert (two-party) computation. Roughly speaking, a protocol is covert if the output and execution of the protocol is only revealed to the participants if they all indeed participated and if the result of the computation is favorable according to some predicate. This was further extended in [21] by Chandran, Goyal, Ostrovsky and Sahai to the multi-party setting. Our notion of subliminality is similar but distinct from covertness in the following ways. First, whereas covertness is natural in the context of secure function evaluation (SFE), subliminality is more suited to reactive functionalities and, specifically, functionalities that deliver outputs to a single party (e.g., structured encryption or private function evaluation). With a covert protocol, if the output is unfavorable or if some parties choose not to participate then the execution is completely hidden even from the parties themselves. In other words, third parties as well as the participants themselves will not know that the protocol ever took place. If the output is favorable, however, the participants learn that the protocol was executed and will receive their output. With a subliminal protocol, on the other hand, the corrupted party learns that the first round (i.e., the setup/initialization round) occurred but

hides the execution of every following round. Note that the security of subliminal protocols is not defined with respect to favorable outputs.

3 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. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{v}$ of n elements, we refer to its ith element as $\mathbf{v}_i$. If T is a set then $\#T$ refers to its cardinality. Given strings x and y, we refer to their concatenation as $x\|y$.

Model of computation. Our model of computation is the concurrent read exclusive write (CREW) word parallel RAM (PRAM). In this model, the machine has an infinite number of processors that can ready memory cells at the same time but can only write to a memory cell one at a time. We also 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 $|x|_w$ the word-length of an item x; that is, $|x_w| = |x_2|/w$. We assume that $w = \Omega(k)$ where k is the security parameter. We also assume that the processors on a machine have access to a global clock; that is, when any two processors (on possibly different machines) read from the global clock at the same moment, they receive the same time. We refer to such machines as *clocked*. We also consider machines whose processors have access to a timer; that is, each processor has the ability to tell how many time units have elapsed between two events.

Dictionaries & Multi-maps. A dynamic dictionary DX with capacity n is a collection of n label/value pairs $\{(\ell_i, v_i)_i\}_{i\leq n}$ that supports Get and Put operations. We denote the label space of a dictionary by $\mathbb{L}$ and the set of labels stored in a dictionary DX by $\mathbb{L}_{\mathsf{DX}}$. We write $v_i = \mathsf{DX}[\ell_i]$ to denote getting the value associated with label ℓ_i; and $\mathsf{DX}[\ell_i] := v_i$ to denote putting the the label/value pair (ℓ_i, v_i) in the dictionary. A dynamic multi-map MM with capacity n is a collection of n label/tuple pairs $\{(\ell_i, \mathbf{v}_i)_i\}_{i\leq n}$ that supports Get and Update operations. We denote the label space of a multi-map by $\mathbb{L}$ and the set of labels stored in a multi-map MM by $\mathbb{L}_{\mathsf{MM}}$. We write $\mathbf{v}_i = \mathsf{MM}[\ell_i]$ to denote getting the tuple associated with label ℓ_i. We consider multi-maps that support both additions and deletions as their update operation. An addition operation adds a label/tuple pair $(\ell, \mathbf{v})$ to the multi-map where the label may or may not already exist in the multi-map. A delete operation removes a set of values $\mathbf{v}'$ from the tuple $\mathbf{v}$ of a given label ℓ. It will be useful for us to make use of a multi-map-to-dictionary transformation. More precisely, we write $\mathsf{DX} := \tau(\mathsf{MM})$ to denote the process of transforming a multi-map MM with label space $\mathbb{L}$ and value space $\mathbb{V}$ into a dictionary DX over the same spaces. Such transformations are implicitly used in many constructions including the schemes in [24, 23, 19].

Stochastic processes. A stochastic process $\{X(t) : t \in T\}$ is a set of random variables over a common probability space that is indexed by an index set T. Here, we will consider ensembles of stochastic processes which are sets $\{X_k(t) : t \in T, k \in \mathbb{N}_{\geq 1}\}$ indexed by both a set T and a security

parameter k. For notational convenience we omit the security parameter and refer to stochastic process ensembles simply as stochastic processes.

Probability generating function (PGF). A PGF of a distribution $\mathcal{D}$ is the generating function P such that $P(z) = \mathrm{E}[z^X] = \sum_{n=0}^{\infty} p_n \cdot z^n$, where, for all $n \in \mathbb{N}$, $p_n = \Pr[X = n]$, when the random variable $X \sim \mathcal{D}$.

Exponential distribution. We say that a random variable X is *exponentially* distributed with parameter $\lambda \in \mathbb{R}_{>0}$ when its probability density function $f_X(t) = \lambda \cdot e^{-\lambda \cdot t}$, for $t \geq 0$ and 0 otherwise. And we usually write $X \sim \mathsf{Exp}(\lambda)$. An important property of the exponential distribution is that it is *memoryless* which means that for $s, t > 0$, $\Pr[X > t + s \mid X > t] = \Pr[X > s]$.

Poisson distribution. We say that a random variable X is Poisson distributed with parameter $\lambda \in \mathbb{R}_{>0}$, and write $X \sim \mathsf{Pois}(\lambda)$, if its probability mass function, is $\Pr[X = n] = \lambda^n \cdot e^{-\lambda}/n!$, for all $n \in \mathbb{N}$.

3.1 Queueing Theory

The analysis of our scheme relies on fundamental concepts of the theory of queues. In this section, we recall some necessary background, including standard queuing systems our model relies on. For more details on queueing theory we recommend [58].

The Poisson process. A Poisson process is a stochastic counting process that is widely used to model the number of events that occur in an interval of time. More formally, a stochastic process $\{X(t) \mid t \geq 0\}$ is a *Poisson process* with rate $\lambda \in \mathbb{R}_{>0}$ if it verifies the following:

1. $X(t) \in \mathbb{N}$ counts the number of occurrences of an event between time 0 and time t. In particular, for all $s < t$, the quantity $X(t) - X(s)$ represents the number of occurrences between time s and time t.

2. for any two disjoint time intervals, the number of occurrences during the two intervals is independent.

3. the number of occurrences in any time interval of length τ is Poisson distributed with parameter $\lambda \cdot \tau$. In particular, for all $t, \tau > 0$, and all $n \in \mathbb{N}$, we have $\Pr[X(t + \tau) - X(t) = n] = e^{\lambda \cdot \tau} \cdot (\lambda \cdot \tau)^n / n!$.

Note that the expected number of occurrences within an interval of size τ is equal to $\lambda \cdot \tau$ which follows from the fact that the distribution is Poisson. And if we set the interval of time τ to 1, then the expected number of occurrences is λ. Because of this, λ is usually viewed as the arrival rate of the Poisson process. An important property of the Poisson process is that the inter-arrival times are exponentially distributed with parameter λ.

The compound Poisson process. As described above, the Poisson process counts the number of occurrences of a specific event; for example, a customer entering a mall, an individual making a phone call, or an application sending a query to a database. In the following, we are interested in a generalization of the Poisson process where the events have weights assigned to them. This

models situations where, for example, we would like to keep track not only of the number of customers that enter a mall but the amount of time each customer spent in the mall. In particular, the compound Poisson process, also called the *stuttering* Poisson process, counts the number of occurrences of an event along with the corresponding weights. We are going to assume that the weights are independent and identically distributed. More formally, we denote by $W_i \sim \mathcal{W}$ the random variable that corresponds to the weight of the ith occurrence of the event and by $\mathcal{W}$ the underlying distribution which can be either continuous or discrete. Given a Poisson process $\{X(t) \mid t \geq 0\}$ with rate $\lambda \in \mathbb{R}_{>0}$ and a sequence of independent and identically distributed random variables $(W_i)_{i\in\mathbb{N}}$, a compound Poisson process is the stochastic process $\{C(t) \mid t \geq 0\}$ where $C(t) = \sum_{i=0}^{X(t)} W_i$.

Continuous Markov process. A continuous-time Markov chain (MTMC) $\{X(t) \mid t > 0\}$ is a stochastic process that has two components: a jump chain $\mathcal{X}$ and a set of holding time parameters $\lambda_i \in \mathbb{R}_{>0}$. The jump chain is a discrete time Markov chain defined with a transition matrix $\mathsf{T} = (\mathsf{T}_{i,j})$ and an initial distribution μ over a countable set $\mathbb{S} \subset \mathbb{N}$ where $\mathsf{T}_{i,i} = 0$ and $\sum_{j\in\mathbb{S}} \mathsf{T}_{i,j} = 1$ for all $i \in \mathbb{S}$, and which verifies the following:

- if $X(t) = i$, then the time until the process changes to a state $j \neq i$ is exponentially distributed with parameter λ_i;
- if $X(t) = i$, then the probability that the next jump will land at the jth state is equal to $\mathsf{T}_{i,j}$.

The process verifies the *Markov property* which is that, for all $0 \leq t_1 < \ldots t_n < t_{n+1}$, the following holds $\Pr[\, X(t_{n+1}) = i_{n+1} \mid X(t_n) = i_n, \cdots, X(t_1) = i_1 \,] = \Pr[\, X(t_{n+1}) = i_{n+1} \mid X(t_n) = i_n \,]$.

Transition rate matrix. A continuous-time Markov chain $(\mathsf{T}, \mu, (\lambda_i)_i)$ can also be characterized by the transition rate matrix $\mathsf{R} = (\mathsf{R}_{i,j})$ defined as

$$\mathsf{R}_{i,j} = \begin{cases} \lambda_i \cdot \mathsf{T}_{i,j} & \text{if } i \neq j \\ \lambda_i & \text{otherwise.} \end{cases}$$

We can similarly compute the transition matrix T of the jump chain when given the transition rate matrix R, refer to [9] for more details.

Balance equations. Given a continuous time Markov chain $(\mathsf{T}, \mu, (\lambda_i)_{i\in\mathbb{S}})$ over a countable set $S \subset \mathbb{N}$, if a stationary distribution $\pi = (\pi_i)_{i\in\mathbb{S}}$ can be found, then the balance equations are given by

$$\pi_i \cdot \sum_{j\in\mathbb{S}\setminus\{i\}} \mathsf{R}_{i,j} = \sum_{j\in\mathbb{S}\setminus\{i\}} \pi_j \cdot \mathsf{R}_{j,i}, \tag{1}$$

where R is the transition rate matrix. Balance equations represent the probability *flow* between the Markov chain states which is fixed and independent of the time.

3.2 Queuing Systems

A queuing system models a queue and allows us to study its behavior with respect to a specific arrival and servicing processes. The arrival process inserts items into the queue and the servicing

process dequeues items one at a time in a first in first out (FIFO) basis.[7]

Kendall's notations. Queuing systems can be described using Kendall's notation, which is a series of letters and numbers, most commonly a triplet $\mathsf{A/S/c}$, that characterizes the arrival process, the servicing process, and the number of processing entities.[8] The first letter A denotes the stochastic process of arrivals, the second letter C denotes the servicing distribution, and the last number c refers to the number of processing servers.

Common queueing systems. One of the most widely used and studied queueing system is the $\mathsf{M/M/1}$ system where M is a shorthand for *memoryless* and refers to a Poisson process. The memoryless property is a result of the inter-arrival times being exponentially distributed. Similarly, the second letter M is shorthand for memoryless but refers to a servicing distribution that is exponentially distributed, and finally, the number 1 means that there is a single server processing the queue. In a batch arrival queueing system, $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$, the elements arrive in batches following a compound Poisson process such that the weights are $\mathcal{R}$-dsitributed. The servicing distribution is exponentially distributed and the number of servers is equal to 1.

Little's law. The number of elements in a queueing system, L, as well as the waiting time of an element in the queue, W, are related based on Little's law. In particular, for the $\mathsf{M/M/1}$ system, $L = \lambda \cdot W$ where λ is the arrival rate, whereas $L = \lambda \cdot \mathrm{E}[R] \cdot W$ in the case of the $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ system where $R \sim \mathcal{R}$.

4 Definitions

STE schemes can be interactive or non-interactive. Interactive schemes produce encrypted structures that are queried or updated through an interactive two-party protocol between a client and a server, whereas non-interactive schemes produce structures that can be queried or updated by sending a single token. These schemes can also be response-hiding or response-revealing where the former reveal the response to queries to the server whereas the latter do not. We recall here the syntax of an interactive response-hiding dynamic structured encryption scheme.

Definition 4.1 (Dynamic structured encryption)**.** *An interactive response-hiding dynamic structured encryption scheme* $\Sigma_{\mathsf{DS}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Update})$ *for data type* DS *consists of the following polynomial time algorithms:*

1. $(K, \mathsf{st}) \leftarrow \mathsf{Setup}_{\mathbf{C}}(1^k, \mathsf{DS})$ *is an algorithm that takes as input the security parameter* k *and a data structure* DS *and outputs a secret key* K *and an (optional) state* st*.*

2. $(r, \mathsf{st}'; \mathsf{EDS}') \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}(K, q, \mathsf{st}; \mathsf{EDS})$ *is an interactive protocol executed between a client* $\mathbf{C}$ *and server* $\mathbf{S}$*.* $\mathbf{C}$ *inputs the secret key* K*, a query* q *and state* st*.* $\mathbf{S}$ *inputs the encrypted data*

[7] Note that while we are limiting our analysis to the FIFO setting, other servicing disciplines such as *last-in first-out* or *shortest job first* are also possible and we leave it as future work to assess the impact of other servicing disciplines on the efficiency of EXH.

[8] A more detailed description of a queuing system includes, in addition, the size of the processing unit's buffer, the maximum number of elements to process, as well as the service discipline which are all omitted from our notations for clarity purposes.

structure EDS. *The protocol outputs a response* r *and an updated state* st' *to the client and an updated encrypted structure* EDS' *to the server.*

3. $(\mathsf{st}', \mathsf{EDS}') \leftarrow \mathsf{Update}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}, u; \mathsf{EDS})$ *is an interactive protocol executed between a client* $\mathbf{C}$ *and server* $\mathbf{S}$. *The client inputs a secret key* K, *a state* st *and an update operation* u.*The server inputs an encrypted structure* EDS. *The protocol outputs an updated state* st' *to the client and an updated encrypted structure* EDS' *to the server.*

Standard security. The standard notion of security for STE guarantees that: (1) an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$; and (2) the various operations that are supported (e.g., query, add, delete) reveal no information about the structure and the operations beyond some stateful operation leakage $\mathcal{L}_{\mathsf{O}}$. If this holds for non-adaptively chosen operations then the scheme is said to be non-adaptively secure. If, on the other hand, the operations can be chosen adaptively, the scheme is said to be adaptively-secure [24, 23]. Note that the operation leakage is usually broken down into separate leakage functions—one for each supported operation—but here we consider a single stateful leakage function $\mathcal{L}_{\mathsf{O}}$ for all operations. The advantage of this formulation is that it allows us to more easily capture leakage that is a function of different operations. We note that leakage functions for operations can also dependent on the structure which is why typically query and, e.g., add leakages are usually take the structure as input. Here, we will assume that $\mathcal{L}_{\mathsf{S}}$ and $\mathcal{L}_{\mathsf{O}}$ share state so when we write, e.g., $\mathcal{L}_{\mathsf{O}}(\mathsf{op}_i)$ it should be understood that $\mathcal{L}_{\mathsf{O}}$ can depend on both the structure and the operations $(\mathsf{op}_1, \ldots, \mathsf{op}_{i-1})$. We now define two experiments that capture adaptive security for structured encryption:

- *Real-world experiment*: the experiment takes place between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ that is given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses an operation op_i, where op_1 is required to be a setup operation with an adversarially-chosen data structure. For all $i \in [m]$, $\mathcal{C}$ and $\mathcal{A}$ execute the appropriate protocol for op_i with $\mathcal{A}$ playing the role of the server and the challenger playing the role of the client. Finally, the adversary outputs a bit $b \in \{0,1\}$. We refer to the random variable that outputs this bit as $\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$
- *Ideal-world experiment*: the experiment takes place between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ that are both given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses an operation op_i, where op_1 is required to be a setup operation with an adversarially-chosen data structure. For all $i \in [m]$, $\mathcal{A}$ and $\mathcal{S}$ execute the appropriate protocol for op_i with $\mathcal{A}$ playing the role of the server and $\mathcal{S}$ playing the role of the client and being given $\mathcal{L}_S(\mathsf{DS})$ if $i = 1$ and $\mathcal{L}_{\mathsf{O}}(\mathsf{op}_i)$ if $i \neq 1$. Finally, the adversary outputs a bit $b \in \{0,1\}$. We refer to the random variable that outputs this bit as $\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathbb{S}}(k)$

Definition 4.2 (Standard security)**.** *We say that* Σ *is* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})$*-secure if there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT adversaries* $\mathcal{A}$*, for all* $z \in \{0,1\}^*$*:*

$$|\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(k) = 1] - \Pr[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1]| \leq \mathsf{negl}(k).$$

Hiding metadata. Extending the standard security definition to capture time, count and execution-hiding schemes is non-trivial; at least with respect to simulation-based definitions. At a high level, there are two definitional problems that need to be addressed. The first is formally ensuring that

a leakage function be time, count or execution-hiding. [9] The second, more challenging problem, is ensuring that the security *experiment* itself does not reveal time, count or execution information. Towards addressing this, we need to introduce a few conceptual tools. An *execution schedule* is a list of times at which a protocol should be executed in the real-world and an *activation schedule* is a list of times at which a simulator should be activated in the ideal-world. An *execution selection* is a subset of operations that should be executed in the real-world experiment and an *activation selection* is a subset of operations for which a simulator should be activated in an ideal-world experiment. We represent a selection over a sequence of m operations as its indicator vector; that is, an m-bit string with a 1 at the ith position if the ith operation in the sequence is selected and 0 otherwise.

Hiding time. We start by discussing the case of time hiding. Suppose we defined time-hiding by requiring that a scheme be adaptively-secure in the standard sense but with respect to a time-hiding leakage function. As we will show, such an approach is not enough because the *activation time* of the simulator could reveal information about the operation's arrival time even if the leakage function does not. And if this is the case, a protocol that reveals arrival times could be proven time-hiding with respect to this definition. As a concrete example, consider an arbitrary static structured encryption scheme Σ with a query protocol that executes instantly and leakage $\mathcal{L}_{\mathsf{Q}}$ that we assume is time-independent (for example, one can assume it has no query leakage). Now suppose that the client executes the query protocol as soon as a query arrives. It is clear that this scheme reveals query arrival times since the time of execution is equal to the arrival time of the queries and the latter is observable to the adversary. This protocol, however, can be proven time-hiding with respect to the above definition by simply using its simulator with its time-independent leakage: since the simulator is activated as soon as the adversary's operation is received, the simulated protocol will be executed at the same time the real protocol would be.

To address this, in addition to time-hiding leakage, we will require that the scheme be simulatable even when the simulator's activation times are adversarially-chosen. More precisely, we let the adversary adaptively choose a sequence of operations and both an execution and an activation schedule, with the restriction that the first execution and activation times are equal. In the real-world experiment, the protocols will be executed at the times dictated by the execution schedule whereas in the ideal-world experiment the simulator will be activated at the times dictated by the activation schedule. Note that the scheme mentioned above cannot be proven time-hiding with respect to this definition. To see why, consider an adversary that outputs a setup operation and two queries (s_1, q_2) with execution times $(\mathsf{et}_1, \mathsf{et}_2)$ and activation times $(\mathsf{at}_1, \mathsf{at}_2)$ such that $\mathsf{et}_1 = \mathsf{at}_1$ and $\mathsf{et}_2 \neq \mathsf{at}_2$. In the real-world experiment, the query protocol will be executed at time et_2 whereas in the ideal-world experiment the scheme's simulator will be activated at time at_2. And since the adversary is clocked, it can trivially distinguish between these cases.

Hiding counts. Similar definitional problems occur when we try to define count hiding. Suppose we defined count hiding by requiring that a scheme be adaptively-secure in the standard sense but with respect to a count-hiding leakage function. As we will show, such a definition is not enough because the *number of activations* could reveal information about the operation count even if the leakage does not. And, as above, if this is the case then a protocol that reveals counts could

[9] Since our construction has no query and update leakage in zero-leakage mode, we won't need to formalize this but we note that it is relatively easy to do.

be proven count-hiding. As an example, consider, as above, an arbitrary static scheme but with a count-hiding leakage function $\mathcal{L}_{\mathsf{Q}}$. It is clear that if we execute the query protocol for each operation, this scheme will reveal the total number of operations to the server since each execution is observable by the server. This scheme, however, can be proven count-hiding in the sense above by simply using its simulator with its count-hiding leakage. Note that since the simulator is activated once for each of the adversary's operations, the simulated protocol will be executed the same number of times the real protocol would be executed.

To address this, in addition to count-hiding leakage, we will require that the scheme be simulatable even if the simulator activations are adversarially-chosen. More precisely, we let the adversary adaptively choose a sequence of operations, an execution selection es and an activation selection as such that the first execution and activation is 1 (which corresponds to the setup operation being executed in both cases) and that $\mathsf{hw}_{-1}(\mathsf{es}) \neq \mathsf{hw}_{-1}(\mathsf{as})$, where $\mathsf{hw}_{-1}(\cdot)$ is the Hamming weight, ignoring the first bit. In the real-world experiment, only the operations in es are executed whereas in the ideal-world experiment, the simulator will be activated only for the operations in as. Note that the scheme described above cannot be proven count-hiding with respect to this definition. To see why, consider an adversary that outputs a sequence of two queries (q_1, q_2), a selection of executions $\mathsf{es} = (1, 1)$ and a selection of activations $\mathsf{as} = (1, 0)$. In the real-world experiment, the query protocol will be executed for both operations, whereas in the ideal-world experiment the scheme's simulator will only be activated for q_1. Note that, technically speaking, this hides more than the counts, it hides the selections.

Hiding time and count. To formalize both time-hiding and count-hiding together we require that the scheme be simulatable even if the time and number of simulator activations are adversarially-chosen. More precisely, we let the adversary adaptively choose m tuples $(\mathsf{op}_1, \mathsf{et}_1, \mathsf{es}_1, \mathsf{at}_1, \mathsf{as}_1), \ldots, (\mathsf{op}_m, \mathsf{et}_m, \mathsf{es}_m, \mathsf{at}_m, \mathsf{as}_m)$. In the real-world experiment, the selected operations will be executed using the appropriate protocol at their scheduled time whereas in the ideal-world experiment the simulator will be activated at the activation times for the selected operations. We define both experiments more formally as follows:

- *Real-world experiment*: the experiment takes place between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ that is given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses a tuple $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i, \mathsf{at}_i, \mathsf{as}_i)$, where op_1 is required to be a setup operation with an adversarially-chosen data structure, $\mathsf{et}_1 = \mathsf{at}_1$ and $\mathsf{es}_1 = \mathsf{as}_1 = 1$. For all $i \in [m]$, if $\mathsf{es}_i = 1$, $\mathcal{C}$ and $\mathcal{A}$ execute the appropriate protocol at time et_i with $\mathcal{C}$ playing the role of the client and $\mathcal{A}$ playing the role of the server. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$. We refer to the random variable that outputs this bit as $\mathbf{Real}^{\infty}_{\Sigma,\mathcal{A}}(k)$.

- *Ideal-world experiment*: the experiment takes place between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ that are both given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses a tuple $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i, \mathsf{at}_i, \mathsf{as}_i)$, where op_1 is required to be a setup operation with an adversarially-chosen data structure, $\mathsf{et}_1 = \mathsf{at}_1$ and $\mathsf{es}_1 = \mathsf{as}_1 = 1$. For all $i \in [m]$, if $\mathsf{as}_i = 1$, $\mathcal{A}$ and $\mathcal{S}$ execute the appropriate protocol at time at_i with $\mathcal{A}$ playing the role of the server and $\mathcal{S}$ playing the role of the client and being given $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ if $i = 1$ and $\mathcal{L}_{\mathsf{O}}(\mathsf{op}_i)$ if $i \neq 1$. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$. We refer to the random variable that outputs this bit as $\mathbf{Ideal}^{\infty}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$.

Definition 4.3. *We say that Σ is $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{O}})^{\infty}$-secure if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, for all $z \in \{0,1\}^*$:*

$$\Big|\Pr[\mathbf{Real}^{\infty}_{\Sigma,\mathcal{A}}(k) = 1] - \Pr[\mathbf{Ideal}^{\infty}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1]\Big| \leq \mathsf{negl}(k).$$

Hiding executions. As in the previous two cases, defining subliminality requires the design of a new experiment. The concern here is that, if a simulator is activated at all, it reveals to the simulator that an execution took place and this could allow one to prove that an execution-revealing scheme is execution-hiding. To address this, we will require the scheme to be simulatable even if the simulator is only activated once at setup and never again.

- *Real-world experiment*: the experiment takes place between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ that is given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses a tuple $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i)$, where op_1 is required to be a setup operation with an adversarially-chosen data structure and $\mathsf{es}_1 = 1$. For all $i \in [m]$, if $\mathsf{es}_i = 1$, $\mathcal{C}$ and $\mathcal{A}$ execute the appropriate protocol at time et_i with $\mathcal{C}$ playing the role of the client and $\mathcal{A}$ playing the role of the server. Finally, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. We refer to the random variable that outputs this bit as $\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{A}}(k)$.
- *Ideal-world experiment*: the experiment takes place between an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$ that are both given z. For all $1 \leq i \leq m = \mathsf{poly}(k)$, $\mathcal{A}$ adaptively chooses a tuple $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i)$, where op_1 is required to be a setup operation with an adversarially-chosen data structure and $\mathsf{es}_1 = 1$. When $i = 1$, $\mathcal{A}$ and $\mathcal{S}$ execute the setup protocol with $\mathcal{A}$ playing the role of the server and $\mathcal{S}$ playing the role of the client and being given $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$. Finally, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. We refer to the random variable that outputs this bit as $\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$.

Definition 4.4 (Subliminality)**.** *We say that Σ is $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-secure if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, for all $z \in \{0,1\}^*$:*

$$\Big|\Pr[\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{A}}(k) = 1] - \Pr[\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1]\Big| \leq \mathsf{negl}(k).$$

Relationship between notions. In the Theorem below, we show that subliminality implies time and count hiding. Intuitively, this is straightforward since if an adversary does not know whether an operation was executed or not it cannot learn at what time it was executed.

Theorem 4.5. *If Σ is $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-secure then Σ is $(\mathcal{L}_{\mathsf{S}}, \perp)^{\infty}$-secure.*

Proof. Let $\mathcal{S}_{\varnothing}$ be the simulator guaranteed to exist by the $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-security of Σ and consider the simulator $\mathcal{S}_{\infty}$ that runs $\mathcal{S}_{\varnothing}$ when it is activated at setup time and does nothing whenever it is activated again.

We show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}_{\infty}$ such that

$$\Big|\Pr\Big[\,\mathbf{Real}^{\infty}_{\Sigma,\mathcal{A}_{\infty}}(k) = 1\,\Big] - \Pr\Big[\,\mathbf{Ideal}^{\infty}_{\Sigma,\mathcal{A}_{\infty},\mathcal{S}_{\infty}}(k) = 1\,\Big]\Big| \geq \varepsilon(k)$$

where $\varepsilon(k)$ is non-negligible, then there exists a probabilistic polynomial-time adversary $\mathcal{B}_{\varnothing}$ such that

$$\Big|\Pr\Big[\,\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_{\varnothing}}(k) = 1\,\Big] - \Pr\Big[\,\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_{\varnothing},\mathcal{S}_{\varnothing}}(k) = 1\,\Big]\Big| \geq \varepsilon(k).$$

$\mathcal{B}_{\varnothing}$ starts by simulating $\mathcal{A}_{\infty}$. For all $i \in [m]$, when $\mathcal{A}_{\infty}$ outputs $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i, \mathsf{at}_i, \mathsf{as}_i)$, $\mathcal{B}_{\varnothing}$ outputs $(\mathsf{op}_i, \mathsf{et}_i, \mathsf{es}_i)$. Recall that in $\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_{\varnothing},\mathcal{S}_{\varnothing}}(k)$, the simulator $\mathcal{S}_{\varnothing}$ is activated for the first/setup

operation op_1. Finally, $\mathcal{B}_\varnothing$ outputs whatever $\mathcal{A}_\infty$ outputs. Notice that if $\mathcal{B}_\varnothing$ is in a $\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_\varnothing}(k)$ experiment, then $\mathcal{A}_\infty$'s view is the same as in a $\mathbf{Real}^{\infty}_{\Sigma,\mathcal{A}_\infty}(k)$ experiment. On the other hand, if $\mathcal{B}_\varnothing$ is an $\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_\varnothing,\mathcal{S}_\varnothing}(k)$ experiment, then $\mathcal{A}_\infty$'s view is the same as in an $\mathbf{Ideal}^{\infty}_{\Sigma,\mathcal{A}_\infty,\mathcal{S}_\infty}(k)$ experiment. It follows then that

$$\Pr\left[\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_\varnothing}(k)=1\right]=\Pr\left[\mathbf{Real}^{\infty}_{\Sigma,\mathcal{A}_\infty}(k)=1\right]$$

and

$$\Pr\left[\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_\varnothing,\mathcal{S}_\varnothing}(k)=1\right]=\Pr\left[\mathbf{Ideal}^{\infty}_{\Sigma,\mathcal{A}_\infty,\mathcal{S}_\infty}(k)=1\right]$$

from which the Theorem follows.

■

Absorption. We now show an interesting property of subliminal constructions we refer to as *absorption*. Namely, that if a scheme is subliminal then any scheme that makes black-box use of it as a building block becomes subliminal. To formalize this, we first need to introduce some notation. If Ω is a structured encryption scheme that makes black-box use of another scheme Σ, we write $\Omega[\Sigma]$ to refer to the construction that results from instantiating Ω with Σ. We say that Ω makes *singular* use of Σ if it does not send any messages other than the messages generated by Σ. Finally, for any scheme Ω that makes black-box use of a scheme Σ, we denote by $(\mathsf{op}_1,\ldots,\mathsf{op}_m)\leftarrow\Theta^{\Sigma}_{\Omega}(\mathsf{op})$ the process of generating the Σ operations $(\mathsf{op}_1,\ldots,\mathsf{op}_m)$ produced by the Ω operation op. We are now ready to state our Theorem.

Theorem 4.6 (Absorption)**.** *If Σ is $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-secure and if Ω makes singular and black-box use of Σ, then $\Omega[\Sigma]$ is $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-secure.*

Proof. Let $\mathcal{S}_\Sigma$ be the simulator guaranteed to exist by the $(\mathcal{L}_{\mathsf{S}})^{\varnothing}$-security of Σ and consider the simulator $\mathcal{S}_\Omega$ that runs $\mathcal{S}_\Sigma$ when it is activated at setup time. We show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}_\Omega$ such that

$$\left|\Pr\left[\mathbf{Real}^{\varnothing}_{\Omega,\mathcal{A}_\Omega}(k)=1\right]-\Pr\left[\mathbf{Ideal}^{\varnothing}_{\Omega,\mathcal{A}_\Omega,\mathcal{S}_\Omega}(k)=1\right]\right|\geq\varepsilon(k),$$

where $\varepsilon(k)$ is non-negligible, then there exists a probabilistic polynomial-time adversary $\mathcal{B}_\Sigma$ such that

$$\left|\Pr\left[\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma}(k)=1\right]-\Pr\left[\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma,\mathcal{S}_\Sigma}(k)=1\right]\right|\geq\varepsilon(k).$$

$\mathcal{B}_\Sigma$ starts by simulating $\mathcal{A}_\Omega$. For all $i\in[m]$, when $\mathcal{A}_\Omega$ outputs $(\mathsf{op}_i,\mathsf{et}_i,\mathsf{es}_i)$, $\mathcal{B}_\Sigma$ computes $(\mathsf{op}_{i,1},\ldots,\mathsf{op}_{i,m_i})\leftarrow\Theta^{\Sigma}_{\Omega}(\mathsf{op}_i)$ and, for all $j\in[m]$, outputs $(\mathsf{op}_{i,j},\mathsf{et}_i+\Delta_j,\mathsf{es}_i)$, where Δ_j is the amount of time needed to execute $(\mathsf{op}_{i,1},\ldots,\mathsf{op}_{i,j-1})$ and $\Delta_1=0$. Finally, $\mathcal{B}_\Sigma$ outputs whatever $\mathcal{A}_\Omega$ outputs. Notice that if $\mathcal{B}_\Sigma$ is in a $\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma}(k)$ experiment, then $\mathcal{A}_\Omega$'s view is the same as in a $\mathbf{Real}^{\varnothing}_{\Omega,\mathcal{A}_\Omega}(k)$ experiment. On the other hand, if $\mathcal{B}_\Sigma$ is an $\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma,\mathcal{S}_\Sigma}(k)$ experiment, then $\mathcal{A}_\Omega$'s view is the same as in an $\mathbf{Ideal}^{\varnothing}_{\Omega,\mathcal{A}_\Omega,\mathcal{S}_\Omega}(k)$ experiment. It follows then that,

$$\Pr\left[\mathbf{Real}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma}(k)=1\right]=\Pr\left[\mathbf{Real}^{\varnothing}_{\Omega,\mathcal{A}_\Omega}(k)=1\right]$$

and

$$\Pr\left[\mathbf{Ideal}^{\varnothing}_{\Sigma,\mathcal{B}_\Sigma,\mathcal{S}_\Sigma}(k)=1\right]=\Pr\left[\mathbf{Ideal}^{\varnothing}_{\Omega,\mathcal{A}_\Omega,\mathcal{S}_\Omega}(k)=1\right]$$

from which the Theorem follows.

■

Modeling leakage. We recall some common leakage patterns:

- the *data size*, dsize, returns the size of the multi-map which is its number of label/value pairs.
- the *label size*, lsize, returns the number of labels in the multi-map.
- the *query equality*, qeq, returns an $n \times n$ binary matrix with a 1 at location (i, j) if the ith query label was the same as the jth query label.
- the *response length*, rlen, returns the length of the tuple associated to the query label.
- the *maximum response length*, mrlen, returns the length of the largest tuple in the multi-map.
- the *expected response length*, erlen, returns the expected response length of the queries.
- the *time*, time, returns a timestamp of when the query was issued.
- the *query and update rate*, rate, returns the rate at which the queries and updates arrive.

It is easy to see that the time pattern leaks more than the rate pattern. In particular, one can derive the latter from the former. However, it is not clear how to compare the expected response length erlen and the maximum response length mrlen.

5 EXH: A Subliminal Multi-Map Encryption Scheme

In this section, we give a high-level description of EXH, starting from a baseline construction and highlighting various technical challenges.

Baseline scheme. In the Setup algorithm, the client initializes an empty dictionary DX in which it will store the label/value pairs of the input multi-map MM, two empty queues $\mathbf{Q}_{\mathsf{Q}}$ and $\mathbf{Q}_{\mathsf{U}}$, and an auxiliary dictionary $\mathsf{DX}_{\mathsf{st}}$ for bookkeeping purposes. More precisely, for every label $\ell \in \mathbb{L}_{\mathsf{MM}}$, it parses $\mathsf{MM}[\ell]$ as $(v_1, \ldots, v_{n_\ell})$ and adds $(\ell \| i, v_i)$ to DX. It also keeps track of the length of every tuple and adds the pair (ℓ, n_ℓ) to $\mathsf{DX}_{\mathsf{st}}$. The client then encrypts the dictionary using any standard dynamic dictionary encryption scheme Σ_{DX}, sends the encrypted dictionary EDX to the server and locally stores the secret key K and the state $\mathsf{st} := (\mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{DX}_{\mathsf{st}})$. In parallel, it starts two *servicing processes*, a query servicing process and an update servicing process, that will execute a dictionary-level get or put with the server in order to process the pairs in the query and update queues. To query for a label ℓ, it first retrieves n_ℓ from $\mathsf{DX}_{\mathsf{st}}$, and then adds $\ell \| 1, \ldots, \ell \| n_\ell$ to the query queue $\mathbf{Q}_{\mathsf{Q}}$. To add or delete a label/tuple pair $(\ell, \mathbf{v})$, it adds $\mathsf{pairAdd} \| \ell \| (n_\ell{+}1) \| v_1, \ldots, \mathsf{pairAdd} \| \ell \| (n_\ell{+}\#\mathbf{v}) \| v_{\#\mathbf{v}}$ or $\mathsf{pairDel} \| \ell \| (n_\ell + 1) \| v_1, \ldots, \mathsf{pairDel} \| \ell \| (n_\ell + \#\mathbf{v}) \| v_{\#\mathbf{v}}$ to the update queue $\mathbf{Q}_{\mathsf{U}}$, respectively. Note that our approach to deletion is based on *lazy* deletion where the deleted values persist in the encrypted structure and are treated similarly to additions. This is a standard approach used by most dynamic EMMs [19, 12, 13, 4]

The need for independent servicing. It is crucial that the servicing process be independent of the operation arrivals. To see why, consider a servicing process that waits for the client to enqueue a query and only then starts to process the queue in its entirety. Such a process would reveal the time of the query and the response length for the following reason. The former is revealed because

the servicing process only executes when a query is added to the queue. For the latter, suppose only one query occurred. In such a case, the process will start when the query is issued and will only end when all of its dictionary-level queries are made; revealing exactly the number of values in the label's tuple. To address this, the servicing process needs to be *independent* of the query arrivals. The same reasoning applies to update operations. In our case, we will make the servicing processes wait a number of time steps determined by fixed distributions $\mathcal{D}_{\mathsf{Q}}$ and $\mathcal{D}_{\mathsf{U}}$ that are independent of operation arrivals.

The need for dummies. The servicing processes above can lead to empty queues which can also reveal information. To see why, suppose the client added γ pairs to the query queue $\mathbf{Q}_{\mathsf{Q}}$ while querying for one or more labels but the queue is currently empty. If the server notices that the queue is empty, then it can infer that the number of pairs that it processed is the sum of the response lengths of the client's queries. That is, it learns the total response length. To address this, we modify the construction by adding a dummy pair $(\alpha, \mathbf{0})$ to the dictionary at setup time and having the query servicing process query it whenever the query queue is empty. Note that this already requires us to use a dictionary encryption scheme for which the dummy queries are indistinguishable from real ones. A similar problem also arises in the case of updates where a server can learn the sum of the lengths of the deleted/added tuples to the multi-map. Similarly, to tackle this issue, we modify the construction by having the update servicing process perform a dummy put operation whenever the update queue is empty. Here, a dummy put consists of adding a dummy pair to the dictionary.[10]

The need for a operation-equality-hiding EDX. Consider the following example where the client queries for ℓ_1, ℓ_2, and then ℓ_1 again, where $n_{\ell_1} = 1$ and $n_{\ell_2} = 2$. At this stage, $\mathbf{Q}_{\mathsf{Q}}$ holds $(\ell_1 \| 1, \ell_2 \| 1, \ell_2 \| 2, \ell_1 \| 1)$ and the server receives four dictionary get tokens $(\mathsf{gtk}_1, \mathsf{gtk}_2, \mathsf{gtk}_3, \mathsf{gtk}_4)$. If the dictionary encryption scheme leaks the query equality, the server learns that gtk_1 and gtk_4 are for the same label from which it can infer that ℓ_1 has a response length of 1 and that there were either two labels with response length 1 that were processed in between, or a single label with response length 2. A similar example could be described for update operations where the server can leverage the update equality to learn the length of the added/deleted tuples. To address this issue, we use a dynamic dictionary encryption scheme that hides the operation equality pattern.

The need to ignore. When a queuing system's stability conditions are violated, the queue can diverge which means that its size can grow to infinity. Stability conditions usually come in the form of a relationship between the arrival rate and the servicing rate of the system so we could avoid divergence by setting the rate of the servicing process appropriately. Note that this would leak information, however, since the server could use its observation of the servicing rate and knowledge of the system's stability condition to infer the query arrival rate or the update arrival rate. To address this, we set the servicing rate of the queues independently of the arrival rate and require the client to ignore incoming operations if the queue diverges. Note that in this case, the scheme will not be fully correct.

[10] Note that trivially adding dummies can lead to non-trivial storage overhead. Instead, a dummy put will consist of *overwriting* the same dummy label α. That is, the cost of a dummy put, from a storage standpoint, is almost inexistent.

Low-leakage mode. Note that the scheme can achieve full correctness at the cost of leaking information about the arrival rates of both queries and updates as well as the expected response and update lengths. For this, the user can set the service rates appropriately so that the stability conditions holds for both queues. We refer the reader to Section 6.5 for more details.

5.1 The Construction

The EXH construction makes black-box use of an operation-equality-hiding and response-hiding dynamic dictionary encryption scheme and is parametrized with two servicing distributions, $\mathcal{D}_{\mathsf{Q}}$ and $\mathcal{D}_{\mathsf{U}}$, two bounds mx_{Q} and mx_{U} set to the maximum amount of memory available for the query queue and the update queue, and an upper bound ub set to the maximum number of added or deleted label/value pairs that the client wants to store in the structure. The construction is described below and its detailed pseudo-code is in Figures 1 and 2.

Setup. The setup protocol is an interactive protocol between the client and the server. The client inputs the security parameter 1^k and the multi-map MM whereas the server has no input. The client first initializes a data dictionary $\mathsf{DX} : \{0,1\}^{\gamma} \to \{0,1\}^{\theta+1}$, two empty queues $\mathbf{Q}_{\mathsf{Q}}$ and $\mathbf{Q}_{\mathsf{U}}$, an empty multi-map $\mathsf{MM}_{\mathsf{st}} : \{0,1\}^k \to \{0,1\}^{\star}$, an empty dictionary $\mathsf{DX}_{\mathsf{st}} : \{0,1\}^{\gamma} \to \{0,1\}^{\log(\#\mathsf{MM}+\mathsf{ub})}$, and a counter cnt. For every label $\ell \in \mathbb{L}_{\mathsf{MM}}$, the client parses $\mathsf{MM}[\ell]$ as $(v_1, \ldots, v_{n_\ell})$, adds $(\ell \| i, \mathsf{pairAdd} \| v_i)$ to DX, for all $i \in [n_\ell]$, and adds the pair (ℓ, n_ℓ) to $\mathsf{DX}_{\mathsf{st}}$. The client also adds a dummy pair $(\alpha, \mathsf{pairAdd} \| \mathbf{0}^{\theta})$ to $\mathsf{DX}_{\mathsf{st}}$. The client and server execute the setup protocol of the underlying dictionary encryption scheme Σ_{DX},

$$(K, \mathsf{st}_{\mathsf{DX}}; \mathsf{EMM}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX}; \perp),$$

which outputs a key K and a state $\mathsf{st} := (\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$ to the client and an encrypted multi-map $\mathsf{EMM} := \mathsf{EDX}$ to the server. Finally, the client starts both the query and update servicing processes (in parallel), which each takes as input the key K and the state st from the client and the encrypted multi-map EMM from the server.

Get. The client inputs a key K, a label ℓ and a state st. The client first initializes an empty set Result and parses the state st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$. It then checks if the size of Q_{Q} exceeds mx_{Q} and if so aborts. It then increments the counter cnt and sets $\mathsf{id} := \mathsf{cnt}$. This id is a unique identifier for the query which will help map the query responses back to the query. This is useful because responses can be delayed due to our use of queues and the queue discipline.[11] The client then retrieves the response length $n_\ell := \mathsf{DX}_{\mathsf{st}}[\ell]$ and adds the triplets $(\mathsf{id}, \ell \| 1, \mathsf{false}), \ldots, (\mathsf{id}, \ell \| n_\ell, \mathsf{true})$ to $\mathbf{Q}_{\mathsf{Q}}$. The boolean flag added at the end of every triplet helps identify the final element to process for a given query. The client then waits until the server processes all of the triplets in $\mathbf{Q}_{\mathsf{Q}}$ with the identifier id. Note that the responses to each triplet gets added gradually to the $\mathsf{MM}_{\mathsf{st}}[\ell]$. The client then parses every element in $\mathsf{MM}_{\mathsf{st}}[\ell]$ as $\mathsf{op} \| v$; if op is an addition, it adds the value v to Result; otherwise, it removes the value v from Result. Finally, the client sets $\mathsf{MM}_{\mathsf{st}} := \mathsf{MM}_{\mathsf{st}} - \mathsf{id}$, and outputs Result.

[11] We consider a first-come, first-serve (FCFS) discipline so the queries will be processed in the order of their arrivals. In this case, the multi-map $\mathsf{MM}_{\mathsf{st}}$ stores one entry at most. However, our scheme is designed to support other queueing disciplines such as the *shortest job* queue discipline where queries with the shortest response are prioritized, and in this case, the multi-map $\mathsf{MM}_{\mathsf{st}}$ stores several responses which correspond to different queries.

Update. The client inputs a key K, an update u, and a state st. The client first parses the update u as $(\mathsf{op}, \ell, \mathbf{v})$ and the state st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$. It then checks if the size of Q_{U} exceeds mx_{U} and if so aborts. The client then retrieves the response length $n_\ell := \mathsf{DX}_{\mathsf{st}}[\ell]$ and adds the triplets $(\mathsf{op}, \ell \| (n_\ell + 1), v_1), \cdots, (\mathsf{op}, \ell \| (n_\ell + \#\mathbf{v}), v_{\#\mathbf{v}})$ to the queue $\mathbf{Q}_{\mathsf{U}}$. Finally, the client updates the new tuple length of ℓ by setting $\mathsf{DX}_{\mathsf{st}}[\ell] := n_\ell + \#\mathbf{v}$.

The query servicing protocol. The query servicing protocol is an interactive protocol between the client and the server. It is parametrized with a distribution $\mathcal{D}_{\mathsf{Q}}$ that specifies how the queue is serviced and takes as input the key K and the state st from the client and the encrypted multi-map EMM from the server. It starts an infinite loop and samples $\delta \leftarrow \mathcal{D}_{\mathsf{Q}}$ and waits δ time steps. If the queue is non-empty, it dequeues an element $(\mathsf{id}, \ell^\star, \mathsf{flag})$ and executes

$$(r, \mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \ell^\star; \mathsf{EDX}).$$

It then sets $\mathsf{MM}_{\mathsf{st}}[\mathsf{id}] := \mathsf{MM}_{\mathsf{st}}[\mathsf{id}] \cup \{r\}$. Otherwise, if the queue is empty, it executes

$$(\mathsf{pairAdd} \| \mathbf{0}^\theta, \mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \alpha; \mathsf{EDX}),$$

where α is a dummy element. Moreover, if $\mathsf{flag} = \mathsf{true}$, the protocol outputs $(\mathsf{done}, \mathsf{id})$ which implies that the processing of the query with identifier id has been completed.

The update servicing protocol. The update servicing protocol is an interactive protocol between the client and the server. It is parameterized with a distribution $\mathcal{D}_{\mathsf{U}}$ that specifies how the queue is serviced and takes as input the key K and the state st from the client and the encrypted multi-map EMM from the server. It starts an infinite loop, samples $\delta \leftarrow \mathcal{D}_{\mathsf{U}}$ and waits δ time steps. If the queue is non-empty, it dequeues an element $(\mathsf{op}^\star, \ell^\star, v^\star)$ and executes

$$(\mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \ell^\star, \mathsf{op}^\star \| v^\star; \mathsf{EDX}).$$

Otherwise, if the queue is empty, it executes

$$(\mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \alpha, \mathsf{pairAdd} \| \mathbf{0}^\theta; \mathsf{EDX}),$$

where α is a dummy element.

5.2 Security Analysis

We now prove the security of EXH.

Theorem 5.1. *If Σ_{DX} is $(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}, \perp)$-secure then EXH is $(\mathcal{L}_{\mathsf{S}} \circ \tau)^{\varnothing}$-secure.*

Proof. Let $\mathcal{S}_{\mathsf{DX}}$ be the simulator guaranteed to exist by the adaptive security of Σ_{DX} and consider the simulator $\mathcal{S}_{\mathsf{EXH}}$ that, given leakage $\left(\mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}(\tau(\mathsf{MM})), \mathcal{D}_{\mathsf{Q}}, \mathcal{D}_{\mathsf{U}}, \mathsf{ub}\right)$, outputs $\mathsf{EMM} := \mathsf{EDX}$ where $\mathsf{EDX} \leftarrow \mathcal{S}_{\mathsf{DX}}\left(\mathsf{setup}, \mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}\left(\tau\left(\mathsf{MM}\right)\right), \mathsf{ub}\right)$ where τ is the mapping that takes as input a multi-map MM and outputs a dictionary DX as described in Line 5 and Line 6 in Figure 1. The simulator then starts two parallel processes that do the following:

- *query process*: while true, it samples $\delta_{\mathsf{Q}} \leftarrow \mathcal{D}_{\mathsf{Q}}$, uses its clock to wait for δ_{Q} steps and uses $\mathcal{S}_{\mathsf{DX}}(\mathsf{get}, \perp)$ to simulate a get operation.

Let $\gamma, \theta, \mathsf{mx_Q}, \mathsf{mx_U}, \mathsf{ub} \in \mathbb{N}_{\geq 1}$, $\mathcal{D}_\mathsf{Q}$ and $\mathcal{D}_\mathsf{U}$ be probability distributions and $\Sigma_\mathsf{DX} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Put})$ be a response-hiding dynamic dictionary encryption scheme. Consider the scheme $\mathsf{EXH} = (\mathsf{Setup}, \mathsf{Get}, \mathsf{Update})$ with label space $\mathbb{L}_\mathsf{MM} = \{0,1\}^\gamma$ and value space $\mathbb{V}_\mathsf{MM} = \{0,1\}^\theta$ defined as follows:

- $\mathsf{Setup}_{\mathbf{C},\mathbf{S}}(1^k, \mathsf{MM})$:
  1. $\mathbf{C}$ initializes a dictionary $\mathsf{DX} : \{0,1\}^\gamma \to \{0,1\}^{\theta+1}$;
  2. $\mathbf{C}$ initializes a dictionary $\mathsf{DX_{st}} : \{0,1\}^\gamma \to \{0,1\}^{\log(\#\mathsf{MM}+\mathsf{ub})}$;
  3. $\mathbf{C}$ initializes two empty first-in first-out queues $\mathbf{Q}_\mathsf{Q}$ and $\mathbf{Q}_\mathsf{U}$ and a counter $\mathsf{cnt} := 0$;
  4. $\mathbf{C}$ initializes an empty multi-map $\mathsf{MM_{st}} : \{0,1\}^k \to \{0,1\}^\star$;
  5. for all labels $\ell \in \mathbb{L}_\mathsf{MM}$,
     (a) $\mathbf{C}$ parses $\mathsf{MM}[\ell]$ as $(v_1, \cdots, v_{n_\ell})$;
     (b) for all $i \in [n_\ell]$, $\mathbf{C}$ sets $\mathsf{DX}[\ell \| i] := \mathsf{pairAdd} \| v_i$;
     (c) $\mathbf{C}$ sets $\mathsf{DX_{st}}[\ell] := n_\ell$;
  6. $\mathbf{C}$ sets $\mathsf{DX}[\alpha] := \mathbf{0}^\theta$;
  7. $\mathbf{C}$ computes $(K, \mathsf{st_{DX}}, \mathsf{EDX}) \leftarrow \Sigma_\mathsf{DX}.\mathsf{Setup}(1^k, \mathsf{ub}, \mathsf{DX})$;
  8. output the key K and the state $\mathsf{st} := (\mathsf{DX_{st}}, \mathsf{st_{DX}}, \mathsf{MM_{st}}, \mathbf{Q}_\mathsf{Q}, \mathbf{Q}_\mathsf{U}, \mathsf{cnt})$ to $\mathbf{C}$ and $\mathsf{EMM} := \mathsf{EDX}$ to $\mathbf{S}$;
  9. $\mathbf{C}$ and $\mathbf{S}$ start the query and update queue processes

$$\mathsf{QServicing}^{\mathcal{D}_\mathsf{Q}}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st}; \mathsf{EMM}\Big) \quad \text{and} \quad \mathsf{UServicing}^{\mathcal{D}_\mathsf{U}}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st}; \mathsf{EMM}\Big).$$

- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}, \ell)$:
  1. $\mathbf{C}$ initializes an empty set Result;
  2. $\mathbf{C}$ parses st as $(\mathsf{DX_{st}}, \mathsf{st_{DX}}, \mathsf{MM_{st}}, \mathbf{Q}_\mathsf{Q}, \mathbf{Q}_\mathsf{U}, \mathsf{cnt})$ and $\mathsf{DX_{st}}[\ell]$ as n_ℓ;
  3. $\mathbf{C}$ checks if the size of $\mathbf{Q}_\mathsf{Q}$ is larger than mx and if so aborts;
  4. $\mathbf{C}$ computes $\mathsf{id} := \mathsf{cnt}$ and set $\mathsf{cnt} := \mathsf{cnt} + 1$;
  5. for $i \in [n_\ell - 1]$, $\mathbf{C}$ computes $\mathbf{Q}_\mathsf{Q}.\mathsf{enqueue}\big((\mathsf{id}, \ell \| i, \mathsf{false})\big)$;
  6. $\mathbf{C}$ computes $\mathbf{Q}_\mathsf{Q}.\mathsf{enqueue}\big((\mathsf{id}, \ell \| n_\ell, \mathsf{true})\big)$;
  7. $\mathbf{C}$ holds until it receives $(\mathsf{done}, \mathsf{id})$ from the queue process $\mathsf{QServicing}^{\mathcal{D}_\mathsf{Q}}_{\mathbf{C},\mathbf{S}}$;
  8. for all $e \in \mathsf{MM_{st}}[\mathsf{id}]$
     (a) parse e as $\mathsf{op} \| v$;
     (b) if $\mathsf{op} = \mathsf{pairAdd}$, set $\mathsf{Result} := \mathsf{Result} \cup \{v\}$, otherwise set $\mathsf{Result} := \mathsf{Result} \setminus \{v\}$;
  9. $\mathbf{C}$ computes $\mathsf{MM_{st}} := \mathsf{MM_{st}} - \mathsf{id}$ and outputs Result.

Figure 1: Our EXH construction (part 1).

- $\mathsf{Update}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}, u; \mathsf{EMM})$:
  1. $\mathbf{C}$ parses u as $(\mathsf{op}, \ell, \mathbf{v})$, st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$ and $\mathsf{DX}_{\mathsf{st}}[\ell]$ as n_ℓ;
  2. $\mathbf{C}$ checks if the size of $\mathbf{Q}_{\mathsf{U}}$ is larger than mx_{U} and if so aborts;
  3. for $i \in [\#\mathbf{v}]$, $\mathbf{C}$ computes $\mathbf{Q}_{\mathsf{U}}.\mathsf{enqueue}\big((\mathsf{op}, \ell \| (n_\ell + i), v_i)\big)$;
  4. $\mathbf{C}$ set $\mathsf{DX}_{\mathsf{st}}[\ell] := n_\ell + \#\mathbf{v}$ and outputs the updated state.
- $\mathsf{QServicing}^{\mathcal{D}_{\mathsf{Q}}}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}; \mathsf{EMM})$:
  1. while true,
     (a) sample $\delta \leftarrow \mathcal{D}_{\mathsf{Q}}$ and hold δ time steps;
     (b) parse EMM as EDX and st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$;
     (c) if $\mathbf{Q}_{\mathsf{Q}}.\mathsf{peek} \neq \perp$,
        i. compute $e \leftarrow \mathbf{Q}_{\mathsf{Q}}.\mathsf{dequeue}$;
        ii. parse e as $(\mathsf{id}, \ell^\star, \mathsf{flag})$;
     (d) otherwise, set $\ell^\star := \alpha$;
     (e) compute $(r, \mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Get}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \ell^\star; \mathsf{EDX})$;
     (f) if $\ell^\star \neq \alpha$, compute $\mathsf{MM}_{\mathsf{st}}[\ell^\star] := \mathsf{MM}_{\mathsf{st}}[\ell^\star] \cup r$;
     (g) if $\mathsf{flag} = \mathsf{true}$, send $(\mathsf{done}, \mathsf{id})$ to $\mathbf{C}$.
- $\mathsf{UServicing}^{\mathcal{D}_{\mathsf{U}}}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}; \mathsf{EMM})$:
  1. while true,
     (a) sample $\delta \leftarrow \mathcal{D}_{\mathsf{U}}$ and hold δ time steps;
     (b) parse EMM as EDX and st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{DX}}, \mathsf{MM}_{\mathsf{st}}, \mathbf{Q}_{\mathsf{Q}}, \mathbf{Q}_{\mathsf{U}}, \mathsf{cnt})$;
     (c) if $\mathbf{Q}_{\mathsf{U}}.\mathsf{peek} \neq \perp$,
        i. compute $e \leftarrow \mathbf{Q}_{\mathsf{U}}.\mathsf{dequeue}$;
        ii. parse e as $(\mathsf{op}^\star, \ell^\star, v^\star)$;
     (d) otherwise, set $\mathsf{op}^\star := \mathsf{pairAdd}$, $\ell^\star := \alpha$ and $v^\star := \mathbf{0}^\theta$;
     (e) compute $(\mathsf{st}_{\mathsf{DX}}; \perp) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Put}_{\mathbf{C},\mathbf{S}}(\mathsf{st}_{\mathsf{DX}}, \ell^\star, \mathsf{op}^\star \| v^\star; \mathsf{EDX})$.

Figure 2: Our EXH construction (part 2).

- *update process*: while true, it samples $\delta_{\mathsf{U}} \leftarrow \mathcal{D}_{\mathsf{U}}$, uses its clock to wait for δ_{U} steps and uses $\mathcal{S}_{\mathsf{DX}}(\mathsf{put}, \perp)$ to simulate a put operation.

It remains to show that for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathbf{Real}_{\mathsf{EXH},\mathcal{A}}(k)$ outputs 1 is negligibly close to the probability that $\mathbf{Ideal}_{\mathsf{EXH},\mathcal{A},\mathcal{S}_{\mathsf{EXH}}}(k)$ outputs 1. Notice that $\mathbf{Real}_{\mathsf{EXH},\mathcal{A}}(k)$ is the same as $\mathbf{Ideal}_{\mathsf{EXH},\mathcal{A},\mathcal{S}_{\mathsf{EXH}}}(k)$, except that in the latter EDX is replaced with the output of

$$\mathcal{S}_{\mathsf{DX}}\left(\mathsf{setup}, \mathcal{L}_{\mathsf{S}}^{\mathsf{dx}}\left(\tau\left(\mathsf{MM}\right)\right)\right)$$

and step 5 of $\mathsf{QServicing}$ and $\mathsf{UServicing}$ is replaced with a simulated execution of Get with $\mathcal{S}_{\mathsf{DX}}(\mathsf{get}, \perp)$ and with a simulated execution of Put with $\mathcal{S}_{\mathsf{DX}}(\mathsf{Put}, \perp)$, respectively.

We do this by showing that if there exists a probabilistic polynomial-time adversary $\mathcal{A}_{\mathsf{EXH}}$ for which

$$\left|\Pr\left[\,\mathbf{Real}^{\varnothing}_{\mathsf{EXH},\mathcal{A}_{\mathsf{EXH}}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}^{\varnothing}_{\mathsf{EXH},\mathcal{A}_{\mathsf{EXH}},\mathcal{S}_{\mathsf{EXH}}}(k) = 1\,\right]\right| \geq \varepsilon(k),$$

where $\varepsilon(k)$ is non-negligible, then there exists a probabilistic polynomial-time adversary $\mathcal{B}_{\mathsf{DX}}$ such that

$$|\Pr\left[\,\mathbf{Real}_{\Sigma_{\mathsf{DX}},\mathcal{B}_{\mathsf{DX}}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}_{\Sigma_{\mathsf{DX}},\mathcal{B}_{\mathsf{DX}},\mathcal{S}_{\mathsf{DX}}}(k) = 1\,\right]| \geq \varepsilon(k)$$

which violates the security of Σ_{DX}.

$\mathcal{B}_{\mathsf{DX}}$ starts by simulating $\mathcal{A}_{\mathsf{EXH}}$. When $\mathcal{A}_{\mathsf{EXH}}$ outputs $(\mathsf{op}_1, \mathsf{et}_1, \mathsf{at}_1)$, where $\mathsf{op}_1 = (\mathsf{setup}, \mathsf{MM})$, $\mathcal{B}_{\mathsf{DX}}$ outputs $\mathsf{op}_1 = (\mathsf{setup}, \tau(\mathsf{MM}))$ at time et_1 and passes the messages it receives back and forth to $\mathcal{A}_{\mathsf{EXH}}$. From then on, $\mathcal{B}_{\mathsf{DX}}$ ignores $\mathcal{A}_{\mathsf{EXH}}$'s operations and starts two processes that work as follows. The first repeatedly samples $\delta_{\mathsf{Q}} \leftarrow \mathcal{D}_{\mathsf{Q}}$, waits δ_{Q} steps and outputs an operation $\mathsf{op} = (\mathsf{query}, \ell)$, where ℓ is an arbitrary label. The second, repeatedly samples $\delta_{\mathsf{U}} \leftarrow \mathcal{D}_{\mathsf{U}}$, waits δ_{U} steps and outputs an operation $\mathsf{op} = (\mathsf{update}, u)$, where ℓ and $\mathbf{v}$ are arbtirary elements of the label and value space, respectively.

Notice that if $\mathcal{B}_{\mathsf{DX}}$ is in a $\mathbf{Real}_{\Sigma_{\mathsf{DX}},\mathcal{B}_{\mathsf{DX}}}(k)$ experiment, then $\mathcal{A}_{\mathsf{EXH}}$'s view is the same as in a $\mathbf{Real}^{\varnothing}_{\mathsf{EXH},\mathcal{A}_{\mathsf{EXH}},\mathcal{S}_{\mathsf{EXH}}}(k)$ experiment. On the other hand, if $\mathcal{B}_{\mathsf{DX}}$ is in an $\mathbf{Ideal}_{\Sigma_{d^x},\mathcal{B}_{\mathsf{DX}}}(k)$ experiment then $\mathcal{A}_{\mathsf{EXH}}$'s view is as in an $\mathbf{Ideal}^{\varnothing}_{\mathsf{EXH},\mathcal{A}_{\mathsf{EXH}},\mathcal{S}_{\mathsf{EXH}}}(k)$ experiment.

■

6 Efficiency Analysis of EXH

In this section, we analyze EXH with respect to query latency, update latency, client and server storage, and communication complexity. At the end of this section, we will compare the asymptotic behavior of EXH to state of the art low-leakage schemes. Note that EXH is the only construction that is subliminal not only in the chronometric setting but even in the standard setting. To make the presentation of this section simpler, we describe our model as well as state all our theorems for the case of queries. The results for updates can be derived in the exact same way where we only change the response length, query arrival and query service distributions with their analogous distributions.

6.1 A New Queuing Model

To analyze the efficiency of EXH, it is necessary to define and analyze the queueing model under which EXH operates. In particular, we introduce a new queueing system we call the *batch arrival model with dummies* and analyze it with respect to latency and queue size.

The need for a new queuing model. For our analysis, we assume that queries arrive following a Poisson process with rate $\lambda \in \mathbb{R}_{>0}$ and that the servicing distribution is an exponential distribution with parameter $\mu \in \mathbb{R}_{>0}$. In EXH, a query for ℓ gets transformed into n_ℓ elements that get added simultaneously to the queue $\mathbf{Q}$. This is a compound Poisson process that captures a *batch* arrival behavior (refer to Section 3.1 for more details). However, contrary to a traditional batch arrival model, our system always processes an element; either a dummy element if the queue is empty or a real element otherwise. While this may seem like an insignificant change, it makes the analysis of the queue size much more complex than anticipated.

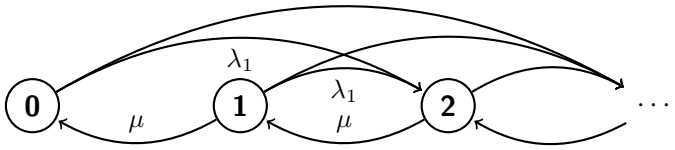
The batch arrival model with dummies. We write $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ to represent our queueing system. Here, the response lengths of the multi-map follow a discrete distribution $\mathcal{R}$, and $\overline{M}$ refers to the servicing being exponentially distributed but where the server either processes a real or a dummy element. We denote by R_i the discrete random variables with a countable support $\mathbb{S} \subset \mathbb{N}$ that represent the response lengths of the queries and we write $\Pr[\, R_i = k \,] = c_k$, for all $k \in \mathbb{S}$. Based on the description of the queueing system, and by modeling the behavior of the system as a continuous Markov chain process, we can generate the transition rate matrix R which is equal to

$$\mathsf{R} = \begin{pmatrix} -\lambda & 0 & \lambda \cdot c_1 & \lambda \cdot c_2 & \lambda \cdot c_3 & \cdots \\ \mu & -(\mu+\lambda) & \lambda \cdot c_1 & \lambda \cdot c_2 & \lambda \cdot c_3 & \cdots \\ 0 & \mu & -(\mu+\lambda) & \lambda \cdot c_1 & \lambda \cdot c_2 & \cdots \\ 0 & 0 & \mu & -(\mu+\lambda) & \lambda \cdot c_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots \end{pmatrix}$$

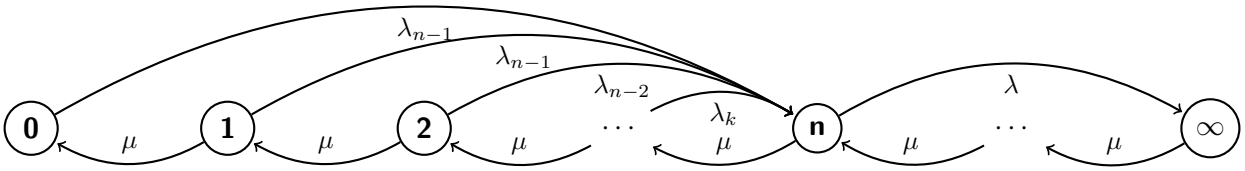
Concretely, the main difference between our model and $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ queuing system is that there is no *real* zero state in our model, as our system will always either process a dummy or a real element. From a modeling standpoint, we set our *zero* state to represent the setting where the system is processing a dummy element, and we set the *one* state to represent the setting where the system is processing a real element. For states larger than *two*, the matrix is the same as the transition rate matrix of a $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ system.

Analysis. For our analysis, we make use of the concept of continuous Markov chain processes. Assuming that the queuing system reaches a steady state, which is a function of the system parameters, we denote by p_i, for $i \in \mathbb{S}$, the ith element of the stationary distribution of the continuous Markov chain. In particular, p_i denotes the probability that the system has size i, for $i \in \mathbb{S}$, when time $t \to \infty$, i.e., after a large number of queries.[12] We also denote by $P(z)$ and $C(z)$ the probability generating functions of the stationary distribution $(p_n)_{n\geq 0}$ and the response length distribution $(c_n)_{n\geq 0}$, respectively. In Theorem 6.1, we compute a closed form of the probability generating

[12]Recall that when the system has i elements, the queue length is at least equal to $i-1$ as the server processes at most one element at a time.



(a) Transition rate diagram for the three initial states.



(b) Transition rate diagram for the nth state when $n \geq 2$.

Figure 3: Transition rate diagram for the $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ queueing system.

function P of the $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ queueing system as well as the probability that the system exceeds size $\nu \in \mathbb{N}$ in the steady phase. Based on this main result, we will derive the expected size of the system in Corollary 6.3 and the expected latency in Corollary 6.4. Due to space constraints, all of the proofs of the theorems and corollaries are in the appendix.

Theorem 6.1. *Given a $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ queueing system with arrival rate $\lambda \in \mathbb{R}_{>0}$, servicing rate $\mu \in \mathbb{R}_{>0}$ and discrete response length distribution $\mathcal{R}$, the probability that the system exceeds $\nu \in \mathbb{N}$ elements is $\sum_{n=\nu}^{\infty} P^{(n)}(0)/n!$, where $P^{(n)}(0)$ is the nth derivative of P at point 0 when it exists, and for $|z| \leq 1$,*

$$P(z) = \sum_{n=0}^{\infty} p_n \cdot z^n = \left(\frac{1-\rho}{1+\lambda\cdot\mu^{-1}}\right)\cdot\left(\frac{\mu+\lambda z C(z)}{\mu-\lambda z\overline{C}(z)}\right),$$

where $\overline{C}(z) = (C(z)-1)\cdot(z-1)^{-1}$, $\rho = \lambda\cdot E[R]\cdot\mu^{-1}$ and $R \sim \mathcal{R}$.

Proof. The transition rate matrix is equivalent to the following set of balance equations which can also be extracted from the transition rate diagram in Figure 3.

$$\begin{aligned} -\lambda p_0 + \mu p_1 &= 0 \quad \text{for } n = 0 \\ -\lambda p_1 - \mu p_1 + \mu p_2 &= 0 \quad \text{for } n = 1 \\ -(\lambda+\mu)p_n + \mu p_{n+1} + \lambda\sum_{k=1}^{n-1} p_{n-k}c_k + \lambda c_{n-1}p_0 &= 0 \quad \text{for } n \geq 2 \end{aligned}$$

Recall that for $|z| \leq 1$, we have

$$P(z) = \sum_{n=0}^{\infty} p_n z^n \quad \text{and} \quad C(z) = \sum_{n=1}^{\infty} c_n z^n$$

Multiplying the balance equations above with z^n and summing all the terms we obtain

$$\begin{aligned}
&-(\lambda+\mu)\sum_{n=1}^{\infty} p_n z^n + \mu\sum_{n=0} p_{n+1} z^n + \lambda\sum_{n=2}^{\infty}\sum_{k=1}^{n-1} p_{n-k}c_k z^n + \lambda p_0\big(\sum_{n=2}^{\infty} c_{n-1}z^n - 1\big) = 0 \\
\Leftrightarrow\ &-(\lambda+\mu)\big(P(z)-p_0\big) + \frac{\mu}{z}\big(P(z)-p_0\big) + \lambda\big(P(z)\cdot C(z) - p_0 C(z)\big) + \lambda p_0\big(zC(z)-1\big) = 0 \\
\Leftrightarrow\ &P(z)\bigg(-(\lambda+\mu) + \frac{\mu}{z} + \lambda C(z)\bigg) + \mu p_0 - \frac{\mu}{z}p_0 + \lambda p_0 C(z)(z-1) = 0,
\end{aligned}$$

where the second equality holds as the double sum represents a convolution

$$\sum_{n=2}^{\infty}\sum_{k=1}^{n-1} p_{n-k}c_k z^n = \sum_{n=1}^{\infty}\sum_{k=1}^{n} p_{n-k}c_k z^n - \sum_{n=1}^{\infty} p_0 c_n z^n = P(z)\cdot C(z) - p_0 C(z).$$

Given the equality above, we obtain

$$\begin{aligned}
P(z) &= \frac{-\mu\cdot z^{-1}\cdot p_0(z-1) - \lambda p_0 C(z)(z-1)}{\lambda(C(z)-1) - \mu\cdot z^{-1}\cdot(z-1)} \\
&= \frac{-\mu\cdot z^{-1}\cdot p_0 - \lambda p_0 C(z)}{\lambda\cdot(C(z)-1)\cdot(z-1)^{-1} - \mu\cdot z^{-1}} \\
&= p_0\cdot\frac{\mu+\lambda z C(z)}{\mu - \lambda z\overline{C}(z)},
\end{aligned}$$

where $\overline{C}(z) = (C(z)-1)\cdot(z-1)^{-1}$. To compute the value of p_0, we solve the equation above for $z=1$. First, notice that

$$P(1) = \sum_{n=0}^{\infty} p_n = 1 \quad \text{and} \quad C(1) = \sum_{n=0}^{\infty} c_n = 1.$$

Using the l'Hopital Rule, we also obtain that

$$\overline{C}(1) = \lim_{z\to 1}\overline{C}(z) = \lim_{z\to 1} C'(z) = \lim_{z\to 1}\sum_{n=1}^{\infty} n c_n z^{n-1} = \lim_{z\to 1}\sum_{n=1}^{\infty} n c_n = \mathrm{E}[R],$$

where $X\sim\mathcal{R}$. Plugging $C(1)$, $\overline{C}(1)$, and $P(1)$ above, we obtain

$$p_0 = \frac{1-\rho}{1+\lambda\cdot\mu^{-1}}$$

where ρ represents the rate of the batch and is equal to $\lambda\cdot\mathrm{E}[R]\cdot\mu^{-1}$. Finally, using Taylor's expansion on P, we obtain that

$$P(z) = \sum_{n=0}^{\infty}\frac{P^{(n)}(0)}{n!}z^n = \sum_{n=0}^{\infty} p_n z^n,$$

which holds true when the nth derivative of P at point 0 exists. The probability that the system exceeds ν elements is therefore

$$\sum_{n=\nu}^{\infty} p_n = \sum_{n=\nu}^{\infty}\frac{P^{(j)}(0)}{j!}.$$

■

Remark. Theorem 6.1 gives us the probability distribution of the number of elements (or sub-tokens) in the system—recall that a sub-token is in the system either when waiting in the queue or when being processed by the server. Deriving the concrete quantity can be achieved for certain response-length distributions as we are going to show below in the case of the geometric distribution, but can be impossible for others.

The geometric case. Consider the case where the response length distribution is a geometric distribution with parameter $\gamma \in \mathbb{R}_{>0}$ such that $c_n = \Pr[R_i = n] = (1-\gamma) \cdot \gamma^{n-1}$, for all $n \geq 1$, where $R_i \sim \mathsf{Geo}_\gamma$ for all $i \in \mathbb{N}$.

Corollary 6.2. *Given a* $\mathsf{M}^{\mathsf{Geo}}/\overline{\mathsf{M}}/1$ *queueing system with arrival rate* $\lambda \in \mathbb{R}_{>0}$*, servicing rate* $\mu \in \mathbb{R}_{>0}$ *and geometric response length distribution* Geo_γ *for* $\gamma \in \mathbb{R}_{>0}$*, the probability that the system exceeds* $\nu \in \mathbb{N}$ *elements is equal to*

$$\frac{\lambda \cdot \mu^{-1} \cdot (1-\rho)}{1-\gamma-\lambda \cdot \mu^{-1}} \cdot \left(\gamma + \lambda \cdot \mu^{-1}\right)^{\nu-2},$$

where $\rho = \lambda \cdot E[R] \cdot \mu^{-1}$ *and* $R \sim \mathsf{Geo}_\gamma$*.*

Proof. First notice that the probability generating function of the response length C can be concretely calculated such that

$$C(z) = \sum_{n=1}^{\infty} c_n \cdot z^n = \sum_{n=1}^{\infty} (1-\gamma) \cdot \gamma^{n-1} z^n = \frac{z \cdot (1-\gamma)}{1-\gamma z}.$$

We can also calculate $\overline{C}$ such that

$$\overline{C}(z) = \frac{C(z)-1}{z-1} = \frac{z \cdot (1-\gamma) - 1 + \gamma z}{(z-1) \cdot (1-\gamma z)} = \frac{1}{1-\gamma z}.$$

Given these quantities and based on the result of Theorem 6.1, we can now concretely calculate P such that

$$\begin{aligned}
P(z) &= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \left(\frac{\mu + \lambda z C(z)}{\mu - \lambda z \overline{C}(z)}\right) \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \frac{\mu + \lambda z^2 \cdot (1-\gamma) \cdot (1-\gamma z)^{-1}}{\mu - \lambda z \cdot (1-\gamma z)^{-1}} \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \frac{\mu \cdot (1-\gamma z) + \lambda z^2 \cdot (1-\gamma)}{\mu \cdot (1-\gamma z) - \lambda z} \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \frac{1 - \gamma z + \lambda \cdot \mu^{-1} \cdot z^2 \cdot (1-\gamma)}{1 - (\gamma + \lambda \mu^{-1}) z} \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \left(\frac{1}{1 - (\gamma + \lambda \mu^{-1}) z} - \frac{\gamma z}{1 - (\gamma + \lambda \mu^{-1}) z} + \frac{\lambda \cdot \mu^{-1} \cdot z^2 \cdot (1-\gamma)}{1 - (\gamma + \lambda \mu^{-1}) z}\right) \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \left(\sum_{n=0}^{\infty} (\gamma + \lambda \mu^{-1})^n z^n - \gamma \sum_{n=0}^{\infty} (\gamma + \lambda \mu^{-1})^n z^{n+1} + \lambda \mu^{-1} (1-\gamma) \sum_{n=0}^{\infty} (\gamma + \lambda \mu^{-1})^n z^{n+2}\right) \\
&= \left(\frac{1-\rho}{1+\lambda \cdot \mu^{-1}}\right) \cdot \left(\sum_{n=0}^{\infty} (\gamma + \lambda \mu^{-1})^n z^n - \gamma \sum_{n=1}^{\infty} (\gamma + \lambda \mu^{-1})^{n-1} z^n + \lambda \mu^{-1} (1-\gamma) \sum_{n=0}^{\infty} (\gamma + \lambda \mu^{-1})^{n-2} z^n\right)
\end{aligned}$$

From the above equality, we can rewrite the nth term of P, p_n, for $n \geq 2$, such that

$$\begin{aligned}
p_n &= \left(\frac{1-\rho}{1+\lambda\cdot\mu^{-1}}\right)\cdot\left((\gamma+\lambda\mu^{-1})^n - \gamma(\gamma+\lambda\mu^{-1})^{n-1} + \lambda\mu^{-1}(1-\gamma)(\gamma+\lambda\mu^{-1})^{n-2}\right) \\
&= \left(\frac{1-\rho}{1+\lambda\cdot\mu^{-1}}\right)\cdot\left(\gamma+\lambda\mu^{-1}\right)^{n-2}\cdot\left((\gamma+\lambda\mu^{-1})^2 - \gamma(\gamma+\lambda\mu^{-1}) + \lambda\mu^{-1}(1-\gamma)\right) \\
&= \left(\frac{1-\rho}{1+\lambda\cdot\mu^{-1}}\right)\cdot\left(\gamma+\lambda\mu^{-1}\right)^{n-2}\cdot\left((\lambda\mu^{-1})^2 + \lambda\mu^{-1}\right) \\
&= \frac{\lambda}{\mu}\cdot\left(1-\rho\right)\cdot\left(\gamma+\lambda\cdot\mu^{-1}\right)^{n-2}.
\end{aligned}$$

Finally, computing the probability to exceed $\nu \geq 2$ elements in the system is equal to

$$\sum_{n=\nu}^{\infty} p_n = \frac{\lambda}{\mu}\cdot\left(1-\rho\right)\cdot\sum_{n=\nu}^{\infty}\left(\gamma+\lambda\cdot\mu^{-1}\right)^{n-2} = \frac{\lambda\cdot\mu^{-1}\cdot(1-\rho)}{1-\gamma-\lambda\cdot\mu^{-1}}\cdot\left(\gamma+\lambda\cdot\mu^{-1}\right)^{\nu-2}$$

■

Based on the result of Corollary 6.2, one can note that as long as $\gamma+\lambda\cdot\mu^{-1} < 1$, the probability of exceeding ν elements in the system exponentially decreases as function of ν. In the remainder of this section, we focus on deriving the expected behavior of the system (and therefore of the queue) which only depends on the existence of the first and second moments of the response length distribution $\mathcal{R}$.

6.2 Client-Side Storage

Using Theorem 6.1, we compute the number of elements (or sub-tokens) in the system, L, in Corollary 6.3.

Corollary 6.3. *Given a* $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ *queueing system with arrival rate* $\lambda \in \mathbb{R}_{>0}$*, servicing rate* $\mu \in \mathbb{R}_{>0}$ *and discrete response length distribution* $\mathcal{R}$*, then*

$$L = \frac{1+E[R]}{1+\mu/\lambda} + \frac{\lambda}{\mu}\cdot\frac{E[R]+E[R^2]}{2(1-\rho)},$$

where $\rho = \lambda\cdot E[R]\cdot\mu^{-1}$ *and* $R \sim \mathcal{R}$*.*

Proof. Based on Theorem 6.1, we know the exact value of the probability generating function P. To calculate the expected number of sub-tokens in the system, it suffices to calculate the first derivative of P and evaluate it at 1 since $L = P^{(1)}(1)$.

$$\begin{aligned}
P^{(1)}(z) &= p_0\cdot\frac{d}{dz}\left(\frac{\mu+\lambda z C(z)}{\mu-\lambda z\overline{C}(z)}\right) \\
&= p_0\cdot\frac{\lambda\cdot(C(z)+zC^{(1)}(z))\cdot(\mu-\lambda z\overline{C}(z)) + \lambda\cdot(\overline{C}(z)+z\overline{C}^{(1)}(z))\cdot(\mu+\lambda z C(z))}{\left(\mu-\lambda z\overline{C}(z)\right)^2}. \qquad (2)
\end{aligned}$$

On the other hand, we have

$$C(1) = \sum_{n=0}^{\infty} c_n = 1 \quad \text{and} \quad C^{(1)}(1) = \sum_{n=0}^{\infty} n \cdot c_n = \mathrm{E}[R] \quad \text{and} \quad \overline{C}(1) = \mathrm{E}[R],$$

where the third equality was derived in the proof of Theorem 6.1. To obtain the value of $\overline{C}^{(1)}(1)$, we apply the l'Hopital Rule twice such that

$$\lim_{z\to 1} \overline{C}^{(1)}(z) = \lim_{z\to 1} \frac{(z-1)C^{(2)}(z)}{2(z-1)} = \lim_{z\to 1} \frac{(z-1)C^{(3)}(z) + C^{(2)}(z)}{2} = \lim_{z\to 1} \sum_{n=1}^{\infty} \frac{n(n-1)}{2} \cdot c_n = \frac{\mathrm{E}[X(X-1)]}{2}.$$

where $\overline{C}^{(1)}(z) = \Big(C^{(1)}(z)(z-1) - (C(z)-1)\Big) \cdot (z-1)^{-2}$ and $R \sim \mathcal{R}$. By evaluating P at $z = 1$ in Equation 2 and by replacing the values of $C(1)$, $C^{(1)}(1)$, $\overline{C}(1)$, and $\overline{C}^{(1)}(1)$ by their concrete quantities derived above, we obtain

$$\begin{aligned}
P^{(1)}(1) &= p_0 \cdot \frac{\lambda(1+\mathrm{E}[R])(\mu - \lambda\mathrm{E}[R]) + \lambda \cdot 2^{-1}(\mathrm{E}[R] + \mathrm{E}[R^2])(\mu+\lambda)}{(\mu - \lambda \cdot \mathrm{E}[R])^2} \\
&= p_0 \cdot \frac{\lambda}{\mu} \frac{(1+\mathrm{E}[R])(1-\rho) + 2^{-1}(\mathrm{E}[R] + \mathrm{E}[R^2])(1+\lambda/\mu)}{(1-\rho)^2} \\
&= \frac{\lambda}{\mu} \frac{(1+\mathrm{E}[R])(1-\rho) + 2^{-1}(\mathrm{E}[R] + \mathrm{E}[R^2])(1+\lambda/\mu)}{(1-\rho)\cdot(1+\lambda/\mu)} \\
&= \frac{1+\mathrm{E}[R]}{1+\mu/\lambda} + \frac{\lambda}{\mu} \cdot \frac{\mathrm{E}[R] + \mathrm{E}[R^2]}{2(1-\rho)}.
\end{aligned}$$

■

Relationship to $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ systems. Note that the expected number of sub-tokens is very close to the standard $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ queueing system. In particular, the expected number of sub-tokens in $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ is

$$\overline{L} = \frac{\lambda}{\mu} \cdot \frac{\mathrm{E}[R] + \mathrm{E}[R^2]}{2(1-\rho)},$$

(refer to [58] for more details). So $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ systems like EXH incur an additive overhead of $(1+\mathrm{E}[R])/(1+\mu/\lambda)$.

Example. Consider a setting with arrival rate $\lambda = 10$, service rate $\mu = 400$, and a geometric response length distribution with parameter $p = 0.1$. Then, the expected number of sub-tokens in the system is equal to 2.

Client and server storage. The client-side storage is equal to the size of the queue, L, described above plus the size of the additional bookkeeping structures which are $O(\#\mathbb{L}_{\mathsf{MM}})$. The server-side storage complexity is equal to the server-side storage complexity of the underlying encrypted dictionary. Using the LDX construction described in Appendix A as the underlying EDX, EXH has server-side storage complexity $O(N)$.

6.3 Query Latency

Leveraging Corollary 6.3 as well as Little's Law, we can calculate the latency W of a query. We state our results in the following corollary.

Corollary 6.4. *Given a $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ queueing system with arrival rate $\lambda \in \mathbb{R}_{>0}$, servicing rate $\mu \in \mathbb{R}_{>0}$ and discrete response length distribution $\mathcal{R}$,*

$$W \leq \left(\frac{1 + E[R]}{1 + \mu/\lambda} + \overline{L}\right) \cdot (\lambda \cdot E[R])^{-1} + E[R] \cdot \mu^{-1},$$

where $\overline{L}$ is the expected number of sub-tokens in the $\mathsf{M}^{\mathcal{R}}/\mathsf{M}/1$ queueing system and $R \sim \mathcal{R}$.

Proof. Using Little's Law, the latency of a sub-token is $P^{(1)}(1) \cdot (\lambda \cdot \mathrm{E}[R])^{-1}$. Assuming the worst-case, where the sub-token was the first sub-token in a query, we obtain

$$\begin{aligned} W &\leq P^{(1)}(1) \cdot (\lambda \cdot \mathrm{E}[R])^{-1} + \mathrm{E}[R] \cdot \mu^{-1} \\ &\leq \left(\frac{1 + \mathrm{E}[R]}{1 + \mu/\lambda} + \overline{L}\right) \cdot (\lambda \cdot \mathrm{E}[R])^{-1} + \mathrm{E}[R] \cdot \mu^{-1}. \end{aligned}$$

■

Example. For arrival rate $\lambda = 10$, service rate $\mu = 400$, and geometric response length distribution with parameter $p = 0.1$, the expected latency per query is at most 0.045 units of time. A unit of time can be thought of as a second if the arrival and service rates are per second as well.

6.4 Communication Complexity

The communication complexity of EXH depends on the parameters of the queueing model but also on the complexity of the underlying encrypted dictionary scheme Σ_{DX}. In particular, if Σ_{DX} is instantiated with LDX (described in Appendix A) using OptORAMa [6] or the non-recursive variant of Path ORAM [61] as the underlying encrypred array scheme, its communication complexity is $O(\log N)$, where N is the number of pairs in the dictionary DX.

Corollary 6.5. *Given a $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ queueing system with arrival rate $\lambda \in \mathbb{R}_{>0}$, servicing rate $\mu \in \mathbb{R}_{>0}$ and discrete response length distribution $\mathcal{R}$, the expected communication complexity per query is $O(\frac{\mu}{\lambda} \cdot \log N)$.*

Remark. While Corollary 6.6 holds for any choice of parameters λ and μ, the latency and the expected size of the system do not. From Theorem 6.1 and its subsequent corollaries, we know that the rate ρ should be strictly smaller than 1 to achieve a stable system which then implies that $\rho = \lambda \cdot \mathrm{E}[R]/\mu < 1$ for $R \sim \mathcal{R}$. In other words, the expected communication complexity of EXH when its queue is stable is lower bounded by $O(\mathrm{E}[R] \cdot \log N)$.

Corollary 6.6. *If $\rho \to 1$, the worst-case communication complexity per query is*

$$O\Big(\mathsf{opt} \cdot \log N\Big).$$

Scheme	Comm. & Comp.	Rounds	Client State	Storage	Correctness	Dynamism	Leakage Patterns
π_{bas} [19]	$O(\mathsf{opt})$	$O(1)$	$O(1)$	$O(N)$	✓	X	dsize, qeq, rlen, time
twoEMM [28]	$O\left(m \cdot k \cdot \log^2 s\right)$	O(1)	O(1)	$O(N)$	✓	✓	lsize, mrlen, time
AZL [45]	$O\left((\alpha + \mathsf{opt}) \cdot \log^2 s\right)$	$O(\mathsf{opt}/\alpha)$	$\Omega(s)$	$O(\alpha \cdot s + N)$	✓	X	dsize, trlen, time
FZL [45]	$O\left((\alpha + \mathsf{opt}) \cdot \log^2 s\right)$	$O(\mathsf{opt}/\alpha)$	$\Omega(s)$	$O(\alpha \cdot s + N)$	X	X	dsize, time
zeroSSE [7]	$O(m \cdot \log(N/m))$	O(1)	$O(s)$	$O(N)$	✓	✓	dsize, mrlen, time
EXH *(low-leakage)*	$O(\mathrm{E}[R] \cdot \log N)$	$O(\mathrm{E}[R])$	$O(s + \mathrm{E}[R^2])$	$O(N)$	✓	✓	dsize, erlen, rate
EXH *(subliminal)*	$O(\mu \cdot \lambda^{-1} \cdot \log N)$	$O(\mu \cdot \lambda^{-1})$	$O\left(s + \frac{\lambda \cdot \mathrm{E}[R^2]}{\mu - \lambda \cdot \mathrm{E}[R]}\right)$	$O(N)$	X	✓	dsize

Table 1: Asymptotic comparison between EXH and other low-leakage constructions. N denotes the number of label/value pairs in the multi-map, s denotes the number of labels, m denotes the maximum response length and k denotes the security parameter. We use opt to denote the optimal time to retrieve the response of any label ℓ. We omit the size of the value space θ the communication, computation and storage complexities.

The corollary follows from the fact that the probability to have a dummy being processed in $\mathsf{M}^{\mathcal{R}}/\overline{\mathsf{M}}/1$ is equal to $p_0 = (1-\rho)/(1+\lambda \cdot \mu^{-1})$. This implies that the server always process a *real* token and never wastes resources processing a dummy token. Note however that the as a result of ρ being close to 1, the query latency will significantly increase and tends to ∞.

6.5 Asymptotic Comparison

In Table 1, we compare the asymptotic behavior of EXH to low-leakage EMMs. We assume that EXH is instantiated using LDX which itself instantiated using a logarithmic-bandwidth ORAM such as OptORAMa [6]. We also include the mid-leakage construction π_{bas} as a baseline. To simplify our presentation, we provide the asymptotic efficiency of all the schemes as a function of the initial size of the multi-map, N, the initial maximum response length, m, as well as the initial number of labels, s. In particular, the underlying assumption we are making here is that the maximum size of the multi-map is $\mathsf{ub} = O(N)$, the maximum response length after ub updates is $O(m)$ and the maximum number of labels is $O(s)$. This simplifies our asymptotic analysis but also helps us compare the efficiency of EXH with static schemes like [45].

Low-leakage mode. In low-leakage mode, $\lambda \cdot \mathrm{E}[R] < \mu$ and $\lambda \cdot \mathrm{E}[U] < \mu$ so that the stability conditions of the queues are satisfied. One can observe in Table 1 that the communication and computational complexity of EXH is independent of the maximum response length m and is $O(\mathrm{E}[R] \cdot \log N)$. We would like to highlight that this is a significant improvement over existing constructions since $\mathrm{E}[R]$ is usually asymptotically dominated by m (see the `Enron` example below). The round complexity is $O(\mathrm{E}[R])$ and the client state is $O(s + \mathrm{E}[R^2])$. Similarly, note that both the round complexity and the client state are independent of m which is highly desirable as we discuss in Section 7. The server storage is $O(N)$ which is similar to low- and mid-leakage schemes. Finally, EXH's query and update latency is equal to

$$O\left(\frac{\mathrm{E}[R^2]}{\lambda \cdot \mathrm{E}[R]}\right) \quad \text{and} \quad O\left(\frac{\mathrm{E}[U^2]}{\lambda \cdot \mathrm{E}[U]}\right),$$

where $\mathrm{E}[R]$ and $\mathrm{E}[R^2]$ and $\mathrm{E}[U]$ and $\mathrm{E}[U^2]$ denote the expected value and the second moment of the response length distribution $\mathcal{R}$ and the update length distribution $\mathcal{U}$ of the multi-map, respectively.

Subliminal mode. In this mode, the servicing rate $\mu \in \mathbb{N}_{>0}$ is a public parameter that is independent of any other paramter including the arrival rate λ and the expected response length $\mathrm{E}[R]$. One can see in Table 1 that EXH has a communication and computational complexity that is independent of the maximum response length m. The behavior of EXH in this mode varies as a function of the relationship between μ, λ and $\mathrm{E}[R]$. In particular, given a fixed service rate $\mu > \lambda \cdot \mathrm{E}[R]$, EXH's stability condition holds and the scheme will always output correct results. The communication and computational complexities however will vary substantially as a function of how close μ and the product $\lambda \cdot \mathrm{E}[R]$ are. If $\lambda \cdot \mathrm{E}[R] = O(\mu)$, then EXH's efficiency behaves exactly the same as in the low-leakage mode. On the other hand, if $\lambda \cdot \mathrm{E}[R] = o(\mu)$, the communication and computation complexity get worse and are $\omega(\mathrm{E}[R] \cdot \log N)$. On the other hand, if the stability condition is violated, then EXH always outperforms the low-leakage mode but at the cost of correctness. In particular, the latency will be unbounded which implies that some queries are never going to be executed. Finally, we would like to emphasize that in subliminal mode, EXH, achieves the best leakage profile we are aware of at the expense of correctness and an increase in latency. More precisely, EXH only leaks the size of the data structure, dsize, and has query and update latency

$$O\Big(\frac{\mathrm{E}[R^2]}{\mathrm{E}[R] \cdot (\mu - \lambda \cdot \mathrm{E}[R])}\Big) \quad \text{and} \quad O\Big(\frac{\mathrm{E}[U^2]}{\mathrm{E}[U] \cdot (\mu - \lambda \cdot \mathrm{E}[U])}\Big),$$

when $\mu > \lambda \cdot \mathrm{E}[R]$ and ∞ otherwise.

The `Enron` dataset case. While the expected response length $\mathrm{E}[R]$ is by definition smaller than the maximum response length m, we were interested in better understanding their asymptotic behavior as a function of the multi-map size, N. For this, we sampled different subsets of the `Enron` dataset [3], and noticed that the maximum response length, the expected response length $\mathrm{E}[R]$ and the second moment of the response length $\mathrm{E}[R^2]$ are $O(N^{0.58})$, $O(N^{0.18})$ and $O(N^{0.64})$, respectively. One can observe that $\mathrm{E}[R] = o(m)$. In this case, the communication and computational complexity of EXH in low-leakage mode is asymptotically dominated by those of twoEMM, AZL, FZL and zeroSSE.

7 Evaluation

In this section, we empirically evaluate EXH and show that it *practically* scales.

Implementation. We implemented EXH in `C++` with 1,373 lines of code. The network layer was implemented using the `Asio` library [1]. We used the `Boost` library to instantiate a lock-free queue for the client queue [2], and used `OpenSSL` [51] as the cryptographic library with `AES-CTR-128` as the symmetric encryption scheme. Finally, we make use of the non-recursive variant of Path ORAM [61] to instantiate the underlying oblivious RAM primitive in LDX. Note that even though it is not the ideal choice when it comes to client storage, it is concretely efficient.

Datasets. We use the `Enron` email dataset [3] which is a publicly available collection of emails of around 150 `Enron` employees. The total uncompressed size of the dataset is equal to 1.32GB. An important characteristic of the `Enron` dataset is that it follows a Zipf-like distribution where a large

number of keywords appears in a small number of documents. The Zipf distribution, in particular, has been widely shown to capture keyword frequencies in text [50] and is used to analyze various structured encryption schemes [37, 43].

Evaluation setup. For our empirical evaluation, we use a machine with an `AMD Ryzen 7 5700x` processor and 32GB of RAM in a local network. As a first step, we parse the `Enron` dataset using the `Clusion` parser [53] and randomly generate 5 multi-maps of sizes 2^{12} to 2^{20} and with maximum response lengths between 25 and $7,844$. The multi-maps map keywords to document ids. Query arrivals follow a Poisson process for which we vary the arrival rate λ to capture different loads. In particular, we chose $\lambda = 32$ queries per second for a small query load, $\lambda = 64$ queries per second for a medium query load, and $\lambda = 128$ queries per second for a high query load. With respect to the service rate μ, we simply set it to be the inverse of the time needed by the server to process a sub-token; and as such, μ varies as a function of the size of the multi-map and the efficiency of the Get protocol of the underlying scheme. While the queries arrive following a Poisson process, the keywords that are queried are chosen uniformly at random from the set of keywords in the multi-map.[13] Throughout our evaluation, we generate $10,000$ queries and terminate the process independently of whether the queue is empty or not and gather various statistics.[14] In particular, we are interested in the following metrics:

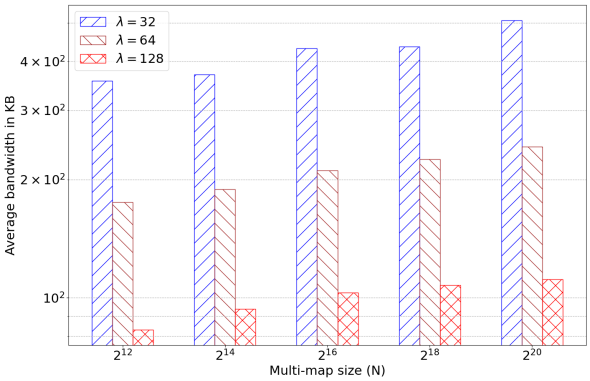
- *Storage*: the number of bits required to store the encrypted structure.
- *Communication complexity*: the number of bits exchanged to execute a single query.
- *Latency*: the time it takes in milliseconds to receive the query response. We study various statistics including the median, the 75th percentile p75, and the maximum latency.
- *Queue size*: the length of the queue. We study various statistics including the median, the 75th percentile p75 and the maximum size.

Server-side storage. Our evaluation shows that EXH storage overhead scales with N. Specifically, for $N = 2^{12}$, EXH structure requires 1.81MB whereas it requires 512MB for $N = 2^{20}$.

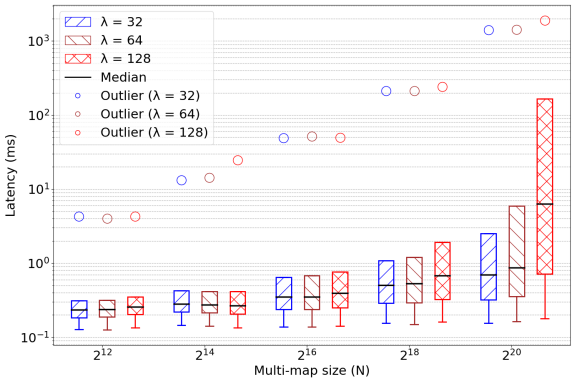
Communication complexity. Figure 4a shows the communication complexity per query in KBytes as a function of the size of the multi-map for three different arrival rates $\lambda \in \{32, 64, 128\}$. Our empirical results are aligned with our theoretical analysis which showed that the communication of EXH, for fixed μ and λ, grows as a function of $\log N$ rather than the maximum response length m. This is crucial from a practical standpoint as it shows that EXH can scale to much larger datasets. As an instance, EXH requires 958KB to execute a single query when $N = 2^{20}$ and $\lambda = 32$. EXH's communication complexity drops significantly when $\lambda = 128$, requiring only 242.89KB to execute a single query. This drop in communication complexity per query when increasing λ hints to a situation where the queue was *extremely* stable (or always empty). In particular, at $\lambda = 128$, the queue receives more elements to process than at $\lambda = 32$, which implies that the server, at a higher arrival rate, processes less dummy queries – effectively using the bandwidth.

[13] In most practical scenarios, queries are usually made for keywords that have a small response lengths. Changing the uniform distribution to such a skewed distribution will only make our results better compared to the two baselines. We chose not to do that to provide a more fair comparison.

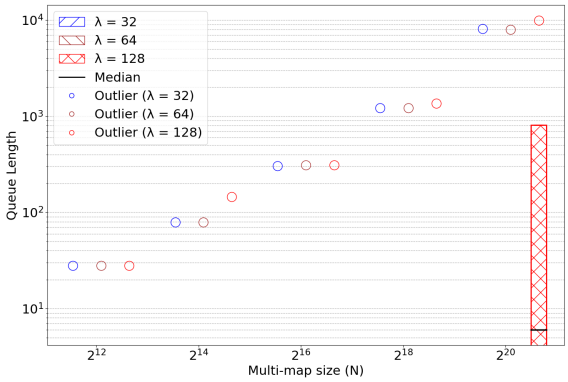
[14] We noticed that increasing the number of queries beyond $10,000$ queries has no effect on any of our efficiency metrics since the system has already reached a steady state.



(a) Communication complexity.



(b) Query latency.



(c) Queue size.

Figure 4: Communication complexity per query in KBytes, latency in milliseconds, and queue size. Both latency and queue size include the following statistics: p25, the median or p50, the p75, and the maximum or p100. The maximum is represented either as a circle or a bar depending on its proximity to the other statistical metrics.

Latency. Figure 4b shows various latency statistics as a function of the size of the multi-map when $\lambda \in \{32, 64, 128\}$. For all arrival rates and all sizes of the multi-map, EXH maintained a median latency below 10 milliseconds, a p75 latency below 166 milliseconds and a maximum latency below 1.82 seconds. In particular, the latency slightly increases as we increase the size of the multi-map. More precisely, the latency is function of both $\mathrm{E}[R^2]$ and $\mathrm{E}[R]$ which themselves depend of the size of the multi-map N. As an example, the expected response length at $N = 2^{12}$ is equal to 1.92 whereas at $N = 2^{20}$, the expected response length is equal to 23.32.

Client-side storage. Figure 4c shows various queue size statistics as a function of the size of the multi-map when $\lambda \in \{32, 64, 128\}$. For all arrival rates and all sizes of the multi-map, EXH has a median queue size equal to 6, a p75 queue size equal to 811, and a maximum queue size equal to $9,890$. In particular, we observe that when $\lambda < 128$, the queue was (almost) always empty, whereas

at $\lambda = 128$, the queue started queueing more elements. This is an indication that the system rate ρ is getting closer to 1. As discussed above, this is also an indication that the server is processing less dummy elements at $\lambda = 128$ when compared to $\lambda = 32$.

Completion ratio. EXH completed all $10,000$ queries for all arrival rates and all multi-map sizes.

References

[1] Asio C++ Library. https://think-async.com/Asio/.

[2] Boost C++. https://www.boost.org.

[3] Enrom Email Dataset. https://www.cs.cmu.edu/~enron/.

[4] Ghous Amjad, Seny Kamara, and Tarik Moataz. Injection-secure structured and searchable symmetric encryption. *Cryptology ePrint Archive*, 2023.

[5] 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.

[6] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. Optorama: optimal oblivious ram. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 403–432. Springer, 2020.

[7] Léonard Assouline and Brice Minaud. Weighted oblivious ram, with applications to searchable symmetric encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 426–455. Springer, 2023.

[8] Dimitri Bertsekas and Robert Gallager. *Data networks*. Athena Scientific, 2021.

[9] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, psu and volume-hiding multi-maps. *Cryptology ePrint Archive*, 2023.

[10] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *Network and Distributed System Security Symposium (NDSS '20)*, 2020.

[11] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. Sse and ssd: page-efficient searchable symmetric encryption. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*, pages 157–184. Springer, 2021.

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

[13] 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.

[14] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE 42^{nd} Annual Symposium on the Foundations of Computer Science (FOCS 2001)*, pages 111–126. IEEE, 2001.

[15] Jin Cao, William S Cleveland, Dong Lin, and Don X Sun. On the nonstationarity of internet traffic. In *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 102–112, 2001.

[16] 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.

[17] 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.

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

[19] 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.

[20] David Cash, Ruth Ng, and Adam Rivkin. Improved structured encryption for sql databases via hybrid indexing. In *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part II*, pages 480–510. Springer, 2021.

[21] Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 238–248. IEEE, 2007.

[22] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security (ACNS '05)*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.

[23] 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.

[24] 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.

[25] 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.

[26] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *Advances in Cryptology - CRYPTO '18*, pages 371–406. Springer, 2018.

[27] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security (ESORICS '15). Lecture Notes in Computer Science*, volume 9327, pages 123–145, 2015.

[28] 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.

[29] Marilyn George, Seny Kamra, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In *Advances in Cryptology - EUROCRYPT 2021*, 2021.

[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] 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 David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 315–331. ACM, 2018.

[33] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1067–1083. IEEE, 2019.

[34] 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.

[35] Charanjit Jutla and Sikhar Patranabis. Efficient searchable symmetric encryption for join queries. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part III*, pages 304–333. Springer, 2023.

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

[37] S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In *Advances in Cryptology - Eurocrypt' 19*, 2019.

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

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

[40] Seny Kamara, Abdelkarim Kati, Jamie De Maria, Tarik Moataz, Andrew Park, and Amos Treiber. MAPLE: MArkov Process Leakage attacks on Encrypted Search. Technical Report 2023/810, IACR ePrint Cryptography Archive, 2023. https://eprint.iacr.org/2023/810.

[41] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. Sok: Cryptanalysis of encrypted search with leaker–a framework for leakage attack evaluation on real-world data. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 90–108. IEEE, 2022.

[42] 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 24*, pages 149–180. Springer, 2018.

[43] Seny Kamara and Tarik Moataz. Bayesian leakage analysis: A framework for analyzing leakage in encrypted search. Technical Report 2023/813, IACR ePrint Cryptography Archive, 2023. https://eprint.iacr.org/2023/813.

[44] Seny Kamara and Tarik Moataz. Design and analysis of a stateless encrypted document database, 2023. https://www.mongodb.com/collateral/stateless-document-database-encryption-scheme.

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

[46] Seny Kamara, Tarik Moataz, Stan Zdonik, and Zheguang Zhao. Opx: An optimal relational database encryption scheme. Technical report, IACR ePrint Cryptography Archive, 2020.

[47] Thomas Karagiannis, Mart Molle, Michalis Faloutsos, and Andre Broido. A nonstationary poisson view of internet traffic. In *IEEE INFOCOM 2004*, volume 3, pages 1558–1569. IEEE, 2004.

[48] Evgenios M. Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. Leakage inversion. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, nov 2022.

[49] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. *IACR Cryptology ePrint Archive*, 2017:701, 2017.

[50] Wentian Li. Random texts exhibit zipf's-law-like word frequency distribution. *IEEE Transactions on information theory*, 38(6):1842–1845, 1992.

[51] The OpenSSL Library. See http://www.openssl.org.

[52] C. Liu, L. Zhu, M. Wang, and Y.-A. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf. Sci.*, 265:176–188, May 2014.

[53] T. Moataz and S. Kamara. Clusion. https://github.com/encryptedsystems/Clusion.

[54] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *USENIX Security Symposium*, pages 127–142, 2021.

[55] 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.

[56] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 79–93. ACM, 2019.

[57] George Rasch. The poisson process as a model for a diversity of behavioral phenomena. 1963.

[58] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. *Fundamentals of queueing theory*, volume 399. John Wiley & Sons, 2018.

[59] 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.

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

[61] 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.

[62] Luis Von Ahn, Nicholas Hopper, and John Langford. Covert two-party computation. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 513–522, 2005.

[63] Charles V. Wright and David Pouliot. Early detection and analysis of leakage abuse vulnerabilities. *IACR Cryptol. ePrint Arch.*, page 1052, 2017.

[64] 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.

[65] Zeguang Zhao, Seny Kamara, Tarik Moataz, and Stan Zdonik. Kafedb: a structurally-encrypted relational database management system. Technical report, 2020.

[66] Zheguang Zhao, Seny Kamara, Tarik Moataz, and Stan Zdonik. Encrypted databases: From theory to systems. In *Conference on Innovative Data Systems Research (CIDR '21)*, 2021.

A Leakage-Free Encrypted Dictionary

LDX is a leakage-free dictionary scheme that makes a black-box use of a leakage-free array encryption scheme $\Sigma_{\mathsf{RAM}} = (\mathsf{Setup}, \mathsf{Access})$. The scheme is detailed in Figure 5 and works as follows.

Setup. The setup protocol is an interactive protocol between the client and the server. The client inputs a security parameter 1^k, an upper bound ub, and a dictionary DX whereas the server inputs a security parameter 1^k. It first parses the dictionary DX as a set of label/value pairs $\{\ell, v_\ell\}_{\ell \in \mathbb{L}_{\mathsf{DX}}}$. It also initializes an empty dictionary $\mathsf{DX}_{\mathsf{st}}$ as well as an array RAM of size ub. The dictionary $\mathsf{DX}_{\mathsf{st}}$ will map every label ℓ to the index in which the value v_ℓ will be stored. For each $\ell \in \mathbb{L}_{\mathsf{DX}}$, we place v_ℓ in the next empty cell of the array and update $\mathsf{DX}_{\mathsf{st}}$ accordingly. For all $j \in \{\#\mathbb{L}_{\mathsf{DX}} + 1, \cdots, \mathsf{ub}\}$, set $\mathsf{RAM}[j] := \mathbf{0}^k$. Finally, the client and the server compute

$$(K, \mathsf{st}_{\mathsf{RAM}}, \mathsf{ERAM}) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Setup}_{\mathbf{C},\mathbf{S}}\Big((1^k, \mathsf{RAM}); 1^k\Big)$$

and the client outputs the key K, the state $\mathsf{st} := (\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{RAM}})$ whereas the server outputs the encrypted dictionary $\mathsf{EDX} := \mathsf{ERAM}$.

Get. The get protocol is an interactive protocol between the client and the server. The client inputs a key K, a state st, and a label ℓ, whereas the server inputs the encrypted dictionary EDX. It first parses the state st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{RAM}})$, computes the address $\mathsf{addr} := \mathsf{DX}_{\mathsf{st}}[\ell]$ and then the client and the server compute

$$((\mathsf{st}_{\mathsf{RAM}}, v), \mathsf{ERAM}) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Access}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st}_{\mathsf{RAM}}, (\mathsf{read}, \mathsf{addr}); \mathsf{ERAM}\Big);$$

the client outputs the value v along with the updated state $\mathsf{st} := (\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{RAM}})$ whereas the server outputs the updated encrypted dictionary $\mathsf{EDX} := \mathsf{ERAM}$.

Put. The put protocol is an interactive protocol between the client and the server. The client inputs a key K, a state st, and a label/value pair (ℓ, v), whereas the server inputs the encrypted dictionary EDX. It first parses the state st as $(\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{RAM}})$, computes the address $\mathsf{addr} := \mathsf{DX}_{\mathsf{st}}[\ell]$ and then the client and the server compute

$$(\mathsf{st}_{\mathsf{RAM}}, \mathsf{ERAM}) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Access}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st}_{\mathsf{RAM}}, (\mathsf{write}, \mathsf{addr}, v); \mathsf{ERAM}\Big);$$

the client outputs the updated state $\mathsf{st} := (\mathsf{DX}_{\mathsf{st}}, \mathsf{st}_{\mathsf{RAM}})$ whereas the server outputs the updated encrypted dictionary $\mathsf{EDX} := \mathsf{ERAM}$.

Efficiency. Assuming that we instantiate the leakage-free array encryption scheme using the optimal oblivious RAM construction optORAMa [6], LDX has a communication and time complexity equal to $O(\log \mathsf{ub})$, a storage complexity equal to $O(\mathsf{ub} \cdot \theta)$ and a client state that is $O(1)$. Note however that when using LDX as a building block of EXH, we do not need an ORAM with a constant client state as EXH locally keeps track of various state information including $\mathsf{DX}_{\mathsf{st}}$ and the queues $\mathbf{Q}_{\mathsf{Q}}$ and $\mathbf{Q}_{\mathsf{U}}$. Given this observation, one can use the non-recursive Path ORAM construction [61]

as it is simpler, concretely more efficient (very small hidden constants in the asymptotics), has a logarithmic communication and time complexity, and has a single round-trip. Readers may notice, however, that the client needs to store a position map that has a size equal to $O(\mathsf{ub} \cdot \log \mathsf{ub})$ which can be comparable to the storage complexity of the server when $\theta = O(\log \mathsf{ub})$. In other words, using the non-recursive version of Path ORAM when instantiating LDX only makes sense in settings where $\theta = \omega(\log \mathsf{ub})$.

Leakage. LDX is a leakage-free dictionary where the setup leakage is composed of the upper bound on the size of the dictionary ub whereas the operation leakage for both queries and puts is null. We state more formally the security guarantees of LDX in the following theorem. The proof is omitted as it follows trivially from the security guarantees of ORAM.

Theorem A.1. *If* Σ_{RAM} *is* $(\mathsf{dsize}, \bot)$*-secure, then* LDX *is* $(\mathsf{dsize}, \bot)$*-secure.*

Let $\gamma, \theta \in \mathbb{N}_{\geq 1}$ and $\Sigma_{\mathsf{RAM}} = (\mathsf{Setup}, \mathsf{Access})$ be an array encryption scheme. Consider the scheme $\mathsf{LDX} = (\mathsf{Setup}, \mathsf{Get})$ with label space $\mathbb{L}_{\mathsf{DX}} = \{0,1\}^{\gamma}$ and value space $\mathbb{V}_{\mathsf{DX}} = \{0,1\}^{\theta}$ defined as follows:

- $\mathsf{Setup}_{\mathbf{C},\mathbf{S}}(1^k, \mathsf{params}, \mathsf{DX}; 1^k)$:
  1. parse DX as $\{\ell, v_\ell\}_{\ell \in \mathbb{L}_{\mathsf{DX}}}$ and params as ub;
  2. initialize an empty dictionary $\mathsf{DX_{st}}$ and an array RAM of size ub;
  3. initialize a counter $\mathsf{addr} := 0$;
  4. for all $\ell \in \mathbb{L}_{\mathsf{DX}}$,
     (a) set $\mathsf{RAM}[\mathsf{addr}] := v_\ell$;
     (b) set $\mathsf{DX_{st}}[\ell] := \mathsf{addr}$;
     (c) increment the counter addr;
  5. for all $j \in \{\#\mathbb{L}_{\mathsf{DX}} + 1, \ldots, \mathsf{ub}\}$, set $\mathsf{RAM}[j] := \mathbf{0}^k$;
  6. set $\mathsf{cnt} := \mathsf{addr}$;
  7. the client $\mathbf{C}$ and the server $\mathbf{S}$ compute
  $$(K, \mathsf{st_{RAM}}, \mathsf{ERAM}) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Setup}_{\mathbf{C},\mathbf{S}}\Big((1^k, \mathsf{RAM}); 1^k\Big)$$
  8. the client $\mathbf{C}$ outputs the key K, the state $\mathsf{st} := (\mathsf{st_{RAM}}, \mathsf{DX_{st}}, \mathsf{cnt}, \mathsf{ub})$ whereas the server $\mathbf{S}$ outputs $\mathsf{EDX} := \mathsf{ERAM}$.
- $\mathsf{Get}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}, \ell; \mathsf{EDX})$:
  1. st as $(\mathsf{st_{RAM}}, \mathsf{DX_{st}}, \mathsf{cnt}, \mathsf{ub})$, $\mathsf{DX_{st}}[\ell]$ as addr, and EDX as ERAM;
  2. compute
  $$\big((\mathsf{st_{RAM}}, v), \mathsf{ERAM}\big) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Access}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st_{RAM}}, \big(\mathsf{read}, \mathsf{addr}\big); \mathsf{ERAM}\Big);$$
  3. the client $\mathbf{C}$ outputs the updated state st and the value v whereas the server outputs the updated encrypted dictionary $\mathsf{EDX} := \mathsf{ERAM}$.
- $\mathsf{Put}_{\mathbf{C},\mathbf{S}}(K, \mathsf{st}, \ell, v; \mathsf{EDX})$:
  1. parse st as $(\mathsf{st_{RAM}}, \mathsf{DX_{st}}, \mathsf{cnt}, \mathsf{ub})$ and EDX as ERAM;
  2. if $\mathsf{cnt} > \mathsf{ub}$, abort;
  3. if $\mathsf{DX_{st}}[\ell] \neq \perp$, set $\mathsf{addr} := \mathsf{DX_{st}}[\ell]$; otherwise,
     (a) set $\mathsf{addr} := \mathsf{cnt}$;
     (b) increment the counter cnt and set $\mathsf{DX_{st}}[\ell] := \mathsf{cnt}$;
  4. compute
  $$\big(\mathsf{st_{RAM}}, \mathsf{ERAM}\big) \leftarrow \Sigma_{\mathsf{RAM}}.\mathsf{Access}_{\mathbf{C},\mathbf{S}}\Big(K, \mathsf{st_{RAM}}, \big(\mathsf{write}, \mathsf{addr}, v\big); \mathsf{ERAM}\Big);$$
  5. the client $\mathbf{C}$ outputs the updated state st whereas the server outputs the updated encrypted dictionary $\mathsf{EDX} := \mathsf{ERAM}$.

Figure 5: LDX: a leakage-free dictionary.