MAPLE: MArkov Process Leakage attacks on Encrypted Search

Seny Kamara
MongoDB & Brown University
seny.kamara@mongodb.com

Abdelkarim Kati
Mohammed-VI Polytechnic University
abdelkarim.kati@um6p.ma

Tarik Moataz[*]
MongoDB
tarik.moataz@mongodb.com

Jamie DeMaria[†]
Elementl
DeMaria@alumni.brown.edu

Andrew Park[‡]
Carnegie Mellon University
andrewpark@cmu.edu

Amos Treiber
Technical University of Darmstadt
treiber@encrypto.cs.tu-darmstadt.de

Abstract

Encrypted search algorithms (ESAs) enable private search on encrypted data and can be constructed from a variety of cryptographic primitives. All known sub-linear ESA algorithms leak information and, therefore, the design of leakage attacks is an important way to ascertain whether a given leakage profile is exploitable in practice. Recently, Oya and Kerschbaum (*Usenix '22*) presented an attack called IHOP that targets the query equality pattern—which reveals if and when two queries are for the same keyword—of a sequence of *dependent* queries.

In this work, we continue the study of query equality leakage on dependent queries and present two new attacks in this setting which can work either as known-distribution or known-sample attacks. They model query distributions as Markov processes and leverage insights and techniques from stochastic processes and machine learning. We implement our attacks and evaluate them on real-world query logs. Our experiments show that they outperform the state-of-the-art in most settings but also have limitations in practical settings.

1 Introduction

Encrypted search algorithms (ESAs) enable private search on encrypted data. They can be constructed from a range of primitives such as searchable symmetric encryption (SSE) [13, 16, 23, 51] / structured encryption (STE) [14], fully homomorphic encryption (FHE) [21], and oblivious RAM (ORAM) [25].

ESA constructions achieve various trade-offs between expressiveness, efficiency, and security and the latter is mainly characterized by well-defined *leakage patterns* like, for example, the query equality pattern, which leaks if and when queries are repeated.

ESA cryptanalysis. Well-defined leakage patterns are useful to describe leakage but do not tell us whether the leakage can be exploited or not in practice. This is typically addressed by designing *leakage attacks* which use the observed leakage usually with some auxiliary information to try and recover information about queries and/or data (we refer the reader to [32] for a survey of ESA cryptanalysis). Leakage attacks and their evaluations cover a lot of different settings, scenarios, and leakage patterns and make a variety of assumptions. Known-data attacks assume the attacker has access to some of the client data while sampled-data attacks assume the attacker has access to a sample taken from a distribution that is close to the distribution of the client's data. Most attacks are passive (i.e., they do not interact with the system) and persistent (i.e., they can observe the interaction between client and server). In most empirical evaluations of leakage attacks, client queries are sampled from various artificial distributions but [32] recently showed that using real-world query data in the form of *query logs* can often lead to very different accuracy results.

Dependent queries. Most leakage attacks assume that queries are independent but, in practice, this may not be the case. For example, after querying for a certain disease, a user may query for a corresponding medication or when querying for the city of "New York" a client may be more likely to also query for the state of "New York". Leakage attacks in the dependent setting was recently considered for the first time by Oya and Kerschbaum [45]. Here, it is assumed that client queries are sampled from a Markov process, meaning that a query depends only on the previous query. At a very high level, their attack, called IHOP, solves an optimization problem whose costs are set using the number of transitions between queries observed via the query equality pattern and the number of expected transitions between keywords given by some auxiliary information. The evaluation of the attack does not use query logs but, instead, relies on Wikipedia data. More precisely, it uses the transition probabilities between different Wikipedia pages as a stand-in for the transition probabilities between keywords/queries. The result is that the evaluation of [45] essentially studies the IHOP attack in a setting where client queries are Wikipedia pages (rather than their content) and the query distribution corresponds to the Wikipedia graph.

[*] Work done in part at Brown University and Aroki Systems.
[†] Work done while at Brown University.
[‡] Work done in part at Brown University.

Proceedings on Privacy Enhancing Technologies 2024(1), 430–446

https://doi.org/10.56553/popets-2024-0025

1.1 Our Contributions

In this work, we continue the study of query equality leakage in the dependent setting as initiated by [45] and we focus on STE/SSE-based ESAs. This is an important leakage pattern to study because it is very common; i.e., most practical constructions reveal it [5, 8, 11–14, 16, 23, 35, 36, 51]. We make the following contributions:

(1) We introduce a new framework based on hidden Markov models (HMM) to model client queries and query equality leakage.
(2) We use our framework to design two new passive and persistent query-recovery attacks, called Stationary and Decoder, that work in either the known- or sampled-data setting against dependent queries (i.e., the same setting as IHOP). While Stationary serves as a warmup attack, its underlying stochastic techniques are fundamental to the design of the Decoder attack.
(3) We implement our attacks as well as the IHOP attack in the open-source framework LEAKER [32] and conduct a broad evaluation of all three attacks on both real-world and synthetic data. More precisely, we use the AOL [46] and TAIR [19] query logs as our dependent queries. Furthermore, we use a variety of synthetic distributions to determine the cases for which the attacks work and do not work. We summarize our results in Table 1 where one can see that our attacks significantly outperform IHOP on the TAIR dataset and most artificial distributions. But, in general, we found that the attacks only work well when: (1) the auxiliary sequence includes the client's exact query sequence; and (2) the client's query distribution is sparse in the sense that, for every keyword, the set of keywords to transition to is small.

Discussion of results and limitations. As discussed above, our attacks do well in a particular setting but leakage attacks should not only be evaluated in settings where they work well. It is important that they be evaluated in a variety of settings including ones where they might perform poorly. This is crucial in order to understand whether an attack can be considered practical or not and points us to settings where cryptanalysis can be improved.

With this in mind, we note that all three attacks required *a lot of auxiliary knowledge.* More precisely, in our evaluations, the attacks did not perform well given auxiliary sequences that did not include the client's *exact* query sequence. Furthermore, the attacks are also very computationally expensive, running in $O(m^2(m+t))$ time, where m is the number of unique keywords and t is the length of the query sequence. For this reason, both our evaluation and the one of [45] were only done on small values of m; 500 in [45] and up to 1 500 in ours. In many realistic settings, however, a much higher value of m would be expected and the attacks' computational cost may become prohibitive. For example, for $m = 2^{19}$, which is approximately the number of keywords in the English Wiktionary [54], the attacks would take over 2^{57} steps. Because of this, none of the experiments conducted in this work or in [45] tell us how the attacks would perform on query distributions over large keyword spaces.

Given these limitations, we do not believe that, at this stage, query-recovery attacks in the dependent-query setting are practical and can be characterized as "devastating" or as "severe threats". Nevertheless, even theoretical leakage attacks are important as they point us towards potential weaknesses in designs and, often, lay the groundwork for future more practical attacks.

2 Related Work

In the following, we review related work on encrypted search and leakage attacks.

Encrypted search algorithms (ESAs). The first explicit ESA construction for exact keyword search was proposed by Song, Wagner and Perrig [51]. Searchable symmetric encryption (SSE) definitions were given by Goh [23] and Chang and Mitzenmacher [13] but the notion of adaptive semantic security for SSE was proposed by Curtmola, Garay, Kamara and Ostrovsky [16]. [16] also first formalized leakage and presented the first sub-linear and optimal-time constructions. Index-based SSE constructions were later generalized as structured encryption (STE) by Chase and Kamara [15]. STE can be used to design sub-linear SSE schemes (i.e., schemes that support private keyword search over encrypted document collections) but has additional applications.

ESAs can also be built via property-preserving encryption (PPE) [2, 5], oblivious RAMs (ORAM) [25], secure multi-party computation (MPC) [24, 56], fully-homomorphic encryption (FHE) [21], and functional encryption (FE) [7]. Fuller et al. [20] provide a survey on ESAs.

Leakage attacks. Sublinear ESA constructions (e.g., based on ORAM and SSE/STE) leak well-defined information. To ascertain how exploitable this information is, *leakage attacks* try to recover information about queries and/or data using the leakage and, sometimes, auxiliary information. Leakage attacks can be classified along several dimensions. The *target* includes either queries, in which case it is a query-recovery attack; or data, in which case it is a data-recovery attack. The *adversarial model* includes: the snapshot model, where the attacker obtains snapshots of the encrypted data; or the persistent model, where the attacker obtains the encrypted data and any interaction between the client and server. The attack's *auxiliary information* can include: *known-data*, where the adversary receives the client's plaintext data; *sampled-data*, where the adversary receives a sample from a distribution that is close to client's query and/or data distribution. Finally, attacks can be passive or active in which case the adversary can inject data and/or queries.

The first leakage attack was given by Islam et al. [28] and was a passive query-recovery attack in the persistent model against co-occurrence leakage and required sampled-data as auxiliary information; though later evaluations found that, to achieve reasonable recovery rates it needs known-data as auxiliary information [10]. Additional sampled-data attacks [18, 27, 40, 44, 45] and known-data attacks [6, 10, 42, 49] were later proposed against a variety of leakage patterns and under a variety of assumptions. There are also a large number of leakage attacks that target range search specifically [26, 37, 39]

Data Source	Scenario	Max. Efficacy per Attack				Median Efficacy per Attack			
		IHOP [45]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl	IHOP [45]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl
TAIR [19]	Exact	23.8%	10.5%	**99.9**%	99.4%	10.1%	7.6%	**99.1**%	98.6%
	Other	4.8%	**5.2**%	2.9%	3.1%	2.9%	**3.4**%	1.9%	1.4%
AOL [46]	Exact	**90.1**%	58.2%	88.3%	87.2%	62.5%	21.3%	53.7%	**66.7**%
	Other	14.3%	**13.3**%	12.6%	0.4%	**0.8**%	0.1%	0.2%	0.1%
Artificial	Sparse	2.4%	8.1%	79.6%	**87.2**%	0.3%	4.9%	69.2%	**71.1**%
	Non-sparse	2.0%	5.7%	**20.7**%	18.4%	0.3%	3.1%	12.8%	**13.1**%

Table 1: Summary of our results, highlighting the maximum and median efficacy (in % of correctly recovered queries) of our new attacks and the IHOP attack [45] in the same setting (dependent sampled queries given query equality leakage) on 50 000 queries. Attacks are evaluated on the AOL [46] or TAIR [19] query logs or on artificial query distributions. For AOL/TAIR, in the *Exact* adversarial scenario, the attacker has an auxiliary sequence that includes the client's exact query sequence, while it does not in the *Other* scenario. For the artificial distributions, the attacker has access to the query distribution in the form of a Markov chain transition matrix. In the respective scenario, the distribution is parameterized such that the matrix is *sparse* (each keyword has a minimum connection to 1% of keywords that can follow it) or *non-sparse* (minimum connection to 10% of keywords).

under various assumptions. A few recent works have proposed theoretical frameworks to quantify leakage in ESA constructions [26, 29, 30, 38, 55]. Kamara et al. [32] provide a survey of leakage attacks and an open-source Python framework called LEAKER to make attack evaluation easier and comparable across works. The framework is then used to re-evaluate a number of attacks on real-world query logs, showing that the choice of data and queries can significantly affect the efficacy of many attacks.

The IHOP attack.

As far as we know, the IHOP attack of Oya and Kerschbaum [45] was the first leakage attack to exploit possible query dependencies [1]. At a high level, this is done by modeling the client's query distribution as a Markov process and formulating a quadratic optimization problem that is a function of the transitions between observed queries given by the query equality pattern and of transition probabilities given as auxiliary information. The optimization problem is then solved with a linear assignment solver. In this work, we also propose attacks that exploit the query equality of dependent query sequences but our techniques are different. In contrast to [45], we model the observed query equality leakage on a dependent query sequence as the output of the observable process of a hidden Markov model (HMM). Our model allows us to make use of sophisticated HMM inference techniques to recover the query sequence. As we demonstrate in our evaluations, our proposed attacks can significantly outperform the IHOP attack.

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 $\{1,\ldots,n\}$. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{q}$ of n elements, we refer to its ith element as q_i or $\mathbf{q}[i]$. If S is a set then $\#S$ refers to its cardinality. k will denote the security parameter.

Searchable symmetric encryption (SSE). SSE schemes are cryptographic schemes that allow a client to outsource an encrypted document collection to a server while supporting for private keyword search on it. Sub-linear and optimal-time SSE schemes can be constructed using standard symmetric encryption and a multi-map encryption scheme. The latter is a type of STE scheme that encrypts multi-map data structures in such a way that they can be privately queried. More precisely, a static and structured or index-based SSE scheme $\mathsf{SSE} = (\mathsf{Setup}, \mathsf{Search})$ consists of two efficient algorithms. Setup takes as input a security parameter 1^k and a document collection $\mathbf{D} = (\mathrm{D}_1,\ldots,\mathrm{D}_n)$ of documents over a space $\mathbb{D}$ and outputs a secret key K and an encrypted document collection $(\mathsf{EMM}, \mathrm{ct}_1,\ldots,\mathrm{ct}_n)$, where EMM is an encrypted multi-map produced by the underlying multi-map encryption scheme and ct_i, for $i \in [n]$, are standard encryptions of the documents. Search is a two-party protocol between a client and a server. The client inputs its secret key K and a keyword w of the keyword (or query) space $\mathbb{W}$ and the server inputs an encrypted collection $(\mathsf{EMM}, \mathrm{ct}_1,\ldots,\mathrm{ct}_n)$. The client receives a set of encrypted documents $\{\mathrm{ct}_i\}_{i \in \mathrm{ids}(w)}$ and the server receives $\perp$.

Leakage. Every operation of an SSE construction is associated with leakage which can be itself composed of many *leakage patterns*. We call the composition of all of these leakage patterns a *leakage profile*. In particular, for static structured ESAs, we differentiate between the setup leakage, $\mathcal{L}_s$, which is the information revealed to the server at setup time, and the query leakage, $\mathcal{L}_q$, which is the information revealed to the server at query time. Leakage patterns are families of functions with different spaces associated to the underlying data collection. For a more detailed discussion on leakage patterns we refer the reader to [34]. In this paper we focus specifically on the *query equality pattern* which is defined as follows:

- the *query equality pattern* is the function family $\mathrm{qeq} = \{\mathrm{qeq}_{k,t}\}_{k,t \in \mathbb{N}}$ with $\mathrm{qeq}_{k,t} : \mathbb{D}_k \times \mathbb{W}_k^t \rightarrow \{0,1\}^{t \times t}$ such that $\mathrm{qeq}_{k,t}(\mathbf{D}, w_1,\ldots,w_t) = B$, where B is a binary $t \times t$ matrix such that $B[i,j] = 1$ if $w_i = w_j$ and $B[i,j] = 0$ if $w_i \neq w_j$. Since qeq is not a function of the document collection we usually just write $\mathrm{qeq}_{k,t} : \mathbb{W}_k^t \rightarrow \{0,1\}^{t \times t}$ where $\mathrm{qeq}_{k,t}(w_1,\ldots,w_t) = B$.

[1] While this work focuses on the setting where queries are dependent, the IHOP attack as well as our attacks can also be evaluated when queries are sampled independently.

The query equality pattern is sometimes referred to as the search pattern in the SSE literature.

In this paper, we focus on constructions that have the following leakage profile:

$$\Lambda = \left(\mathcal{L}_s, \mathcal{L}_q\right) = \left(\star, \left(\mathrm{qeq}, \star\right)\right).$$

where $\star$ refers to any arbitrary collection of leakage patterns. Note that most structured SSE schemes in literature have Λ as their leakage profile [8, 9, 11, 12, 15, 17, 22, 33, 47]. This is true for all property-preserving encrypted search algorithms as well [1, 5]. This illustrates the importance of studying the query equality pattern which is the main focus of this work.

Adversarial models and security. There are different adversarial models that we usually consider in the encrypted search area. The most common adversaries are *persistent* and *snapshot* adversaries. The former receives the encrypted data as well as the transcripts of all query executions. The latter is weaker and only receives the encrypted data after the execution of every query. In this paper we consider that the adversary is persistent. Security definitions are parametrized by a leakage profile. In particular, leakage-parametrized security definitions were introduced by Curtmola et al. [17] and capture the following: given a leakage profile Λ, we say that an SSE scheme is Λ−secure, if a persistent (or a snapshot) adversary cannot learn more information than what is captured by the leakage profile, Λ. For formal definitions, we refer the reader to [3, 15, 17].

Types of attacks. In this paper, we only consider passive attacks where the adversary does not get to choose the data or the queries. Moreover, we solely focus on query recovery attacks where the adversary has either access to: (1) the exact query distribution of the client or, (2) a sample of the queries. We will refer to the former as *known distribution* attacks and to the latter as *known sample* attacks. Note that while knowing the exact distribution is a very strong assumption, it helps us nonetheless to understand what can be achieved in such a setting.

3.1 Stochastic Processes

The majority of leakage attacks work in a setting in which queries are drawn independently from the client's query distribution, **Q**. As discussed in the introduction, the independence assumption is very strong and likely does not hold in practice. Furthermore, it is not clear how dependency would impact the recovery rate of existing attacks.[2] In this work, as in [45] we assume that the client's queries are dependent. More precisely, we assume that the client's query sequence is sampled from discrete stochastic process with a well-defined dependency structure. A discrete stochastic process is an ordered set of random variables that are indexed using some countable set S (e.g., the integer set $\mathbb{N}$). One can define various forms of dependencies between the random variables but in this work, we focus on *Markov chains.* Markov chains are a natural choice for modeling dependency in query distributions as illustrated by the fact that they are extensively used in the context of information retrieval and natural language processing [48].

Markov chain. A Markov chain is a stochastic process composed of an ordered set of random variables which verifies two main properties: the *Markov* property and the *time-homogenous* property. The former means that the output of the nth random variable, X_n, in the stochastic process only depends on the output of the previous random variable, X_{n-1}.[3] The latter means that the likelihood of any two consecutive random variables outputting the same pair (i, j) is fixed. We provide a formal definition of Markov chains below.

Definition 3.1 (Markov chain). *A stochastic process $\Theta = \{X_n \in \{1, \cdots, \#S\} \; : \; n \geq 0\}$ on a countable set of states S is a Markov chain if for any $n \geq 0$, the following two properties hold,*

- *(Markov property): for all $i_0, \cdots, i_n, j \in \{1, \cdots, \#S\}$,*

$$\Pr\left[X_{n+1} = j \mid X_0 = i_0, \cdots, X_n = i_n\right] = \Pr\left[X_{n+1} = j \mid X_n = i_n\right],$$

- *(time-homogenous property): for all $i, j \in \{1, \cdots, \#S\}$,*

$$\Pr\left[X_{n+2} = j \mid X_{n+1} = i\right] = \Pr\left[X_{n+1} = j \mid X_n = i\right].$$

The output of a random variable in a Markov chain is usually referred to as a *state.* And an instance of a Markov chain can be viewed as a series of transitions over a finite number of states. We sometimes refer to a Markov chain instantiation as a realization. Markov chains can be defined using only two parameters: (1) a *transition matrix*; and (2) an *initial distribution.* The transition matrix captures the probability of transitioning from a state to another whereas the initial distribution captures the probability of landing in a given state at the beginning of the process. In the following, we define these two parameters.

Definition 3.2 (Markov Chain Parameters). *A Markov chain Θ on a countable set of states S is characterized by two parameters:*

- *(transition matrix): is a square matrix $\mathsf{T} = (\mathsf{T}_{i,j})_{i,j \in [\#S]}$ that verifies for all $n \geq 0$*

$$\mathsf{T}_{i,j} = \Pr\left[X_{n+1} = j \mid X_n = i\right] \quad \textit{and} \quad \sum_{j \in [\#S]} \mathsf{T}_{i,j} = 1.$$

- *(initial distribution): is a vector μ of size $\#S$ such that for all $i \in \{1, \cdots, \#S\}$,*

$$\mu_i = \Pr[X_0 = i].$$

In the subsequent parts of this paper, we write $\Theta = (\mathsf{T}, \mu)$ to denote a Markov chain Θ parameterized by the transition matrix T and the initial distribution μ. Given a transition matrix of size $n \times n$, we often refer to the set of indices $\{1, \cdots, n\}$ as the states of the transition matrix.

Query distribution. In this work, we consider the setting where the query distribution is a Markov chain. In particular, given a Markov chain $\Theta = (\mathsf{T}, \mu)$, the states of the transition matrix correspond to the query space $\mathbb{W}$. More formally, this correspondence can be captured by a bijection

[2] An evaluation of this is a non-trivial and interesting open problem.

[3] There is a natural generalization of the Markov process where the output of the nth random variable depends on the previous $k \in \mathbb{N}$ outputs. This is why the above process is often called a first order Markov process.

$g : \mathbb{W} \rightarrow \{1, \cdots, \#\mathbb{W}\}$ that maps every keyword to a state. For simplicity, we assume that g maps the ith keyword in $\mathbb{W}$ to i. The values of the transition matrix $\mathsf{T}_{i,j}$ correspond to the probability of querying keyword w_j knowing that the current query is for the keyword w_i, for some $i, j \in \{1, \cdots, \#\mathbb{W}\}$. The initial distribution μ is a vector that captures the probability of querying any keyword $w \in \mathbb{W}$ at the beginning of the process.

4 The Stationary Attack

In this section, we describe our first attack, Stationary. We would like to highlight that the main goal of this attack is to serve as a warmup for our main attack, Decoder. In particular, the Stationary attack shows how one could recover the query sequence solely based on the knowledge of the stationary distribution of the auxiliary Markov chain. The attack can be thought of as a frequency attack as it does not leverage the dependencies between queries beyond what is implicitly included in the stationary distribution. We describe the Stationary attack as a known-distribution attack in the sense that it requires the adversary to know the *exact* query distribution in form of a query transition matrix, but we will show in Section 6 how we can use this attack as a building block to design a known-sample attack. Before describing our attack, we describe two fundamental notions of Markov chains which are: (1) the notion of *stationary distributions*; and (2) the *average number of visits.*

Stationary distribution. A stationary distribution is a probability distribution that captures the probabilities to be at any given state independently of the initial distribution of the Markov chain. In other words, if the stochastic process runs for a long period of time, then the stationary distribution captures the fraction of time spent in a given state. From a mathematical standpoint, the stationary distribution π is a row matrix that remains the same even when multiplied by the transition matrix of the Markov chain.

Definition 4.1 (Stationary Distribution). *Given a Markov chain $\Theta = (\mathsf{T}, \mu)$, we say that a distribution π is stationary over a countable set S if for all $i \in [\#S]$*

$$\pi_i = \sum_{j \in [\#S]} \pi_j \cdot \mathsf{T}_{j,i}.$$

The existence and the uniqueness of the stationary distribution depend on the structure of the transition matrix. As an example, *ergodic* Markov chains have a unique stationary distribution but many others do not. We note, however, that in all the evaluations of our attacks (see Section 7), we were always able to generate the stationary distribution. We refer the reader to [43] for more details.

Average number of visits. A close concept to the one above is the average number of visits made to a particular state. Given a sequence of t states that a Markov chain visited, one can compute the frequency of visits made to every state. In contrast to stationary distributions, the average number of visits can always be computed.

```
• Stationary($\boldsymbol{\ell}, \Theta$):
  (1) parse $\Theta$ as $(\mathsf{T}, \mu)$ and $\boldsymbol{\ell}$ as $(B_{i,j})_{i,j \in [t]}$;
  (2) initialize a map $\alpha : [t] \rightarrow \mathbb{W}$;
  (3) compute the stationary distribution
          $\pi = \pi \cdot \mathsf{T}$
      if it exists, otherwise abort;
  (4) compute the query frequency $\mathbf{v} = (v_1, \cdots, v_t)$ of
      unique queries in qeq such that
          $v_i = \sum_{j \in [t]} B_{i,j}/t;$
  (5) instantiate an empty set $U$;
  (6) for all $i \in [t]$,
      (a) find $\rho$ that verifies
              $\rho = \underset{j \in [\#\mathbb{W}] \backslash U}{\text{argmin}} \ |v_i - \pi_j|;$
      (b) set $\alpha(i) := w_\rho$ and add $\rho$ to $U$;
  (7) output $\alpha$.
```

Figure 1: Our attack: Stationary**.**

Definition 4.2 (Average number of visits). *For a given Markov chain $\Theta = \{X_n \ : \ n \geq 0\}$, the number of visits to the ith state for all $n \geq 0$ equals*

$$v_{i,n} = \frac{1}{n} \cdot \sum_{j=1}^{n} \mathbf{1}_{(X_j = i)},$$

where $\mathbf{1}_{(X_j=i)}$ is an indicator function defined as

$$\mathbf{1}_{(X_j=i)} = \begin{cases} 1 & \text{if } X_j = i \\ 0 & \text{otherwise.} \end{cases}$$

There is a relationship between the number of visits and the stationary distribution. In particular, after a large number of steps $n \in \mathbb{N}$, the stationary distribution is approximatively equal to the average number of visits. We formalize this relationship in the lemma below.

Lemma 4.3 (Convergence of the average number of visits). *Given a Markov chain $\Theta = (\mathsf{T}, \mu)$ and its stationary distribution π, the average number of visits to the ith state for a sufficiently large $n \in \mathbb{N}$ verifies*

$$v_{i,n} \approx \pi_i,$$

where π_i is ith value of the stationary distribution π.

The above lemma is an important component of our our Stationary attack which we describe in Figure 1.

Attack overview. Stationary takes as input the query equality $\boldsymbol{\ell} := \text{qeq}(\mathbf{q})$ of the client's query sequence and a Markov chain Θ representing the client's query distribution. First, it parses the query equality as a square matrix $(B_{i,j})_{i,j \in [t]}$ where t denotes the length of the sequence of queries. It also parses the Markov chain Θ to obtain the transition matrix T.[4] The goal of the attack is to map every query, or every index in$\{1, \cdots, t\}$, to the corresponding keyword in the keyword space $\mathbb{W}$. For this, the attack initializes a mapping $\alpha : [t] \rightarrow \mathbb{W}$. It then

[4] Note that the Stationary attack does not require the knowledge of the initial distribution μ of Θ. This implies that the Stationary attack requires slightly less information than the exact knowledge of the query distribution.

computes the stationary distribution π of the Markov chain's transition matrix, T, such that

$$\pi = \pi \cdot \mathsf{T}.$$

Note that the attack aborts if the computation of the stationary distribution is not possible—refer to our discussion above. Next, given the query equality, the attack computes the query frequency $\mathbf{v}$ of all unique queries. This can be calculated by simply summing the number of times every unique query is made, and then dividing it by the total number of queries t. From a stochastic lens, the query frequency $\mathbf{v}$ is exactly equal to the average number of visits over the states of the Markov chain. Given the result of Lemma 4.3, we know that the average number of visits to the ith state is approximately equal to the ith component of the stationary distribution, π. As a result, given the stationary distribution $\pi = (\pi_1, \cdots, \pi_m)$ and the number of visits $\mathbf{v} = (v_1, \cdots, v_t)$, the attack simply maps the closest value in π to v_i, for all $i \in [t]$; effectively mapping every query to a state (and therefore to a keyword). Finally, the attack outputs the mapping α.

Efficiency. Given a query sequence of length t and a keyword space $\mathbb{W}$ of size m, the total running time of the Stationary attack is

$$O\big(m \cdot (m^2 + t)\big).$$

This running time can be broken down into three main parts.

- (*part 1*): computing the stationary distribution π requires $O(m^3)$ steps,[5]
- (*part 2*): calculating the query frequency $\mathbf{v}$ takes $O(t)$ steps,
- (*part 3*): calculating the arg min takes $O(m \cdot t)$ steps.

We observe that the computation of the stationary distribution is the most expensive part of the attack.

Note. The Stationary attack is similar to other attacks like Frequency-An [41] or Att-Gen [40]. Though these attacks all rely on computing the argmin between the observed and known frequencies, they do not exploit the dependencies of queries. The Stationary attack, on the other hand, does and highlights an important relationship between the stationary distribution and the average number of visits which is a crucial observation that our main attack, Decoder, leverages.

5 The Decoder Attack

In this section, we describe our second attack Decoder. Similar to Stationary, we will first describe Decoder as a known-distribution attack but later in Section 6, we show how to use it to build a known-sample attack. As any known-distribution attack, the Decoder attack tries to solve the following problem:

Given observed leakage and a known query distribution, what is the query sequence that most likely explains this leakage?

Our attack considers this question in the context of query distributions that are Markov chains and solves it modeling the problem as an inference problem on hidden Markov models (HMM). Before we describe the attack we first recall what hidden Markov models are and how they are connected to our problem.

Hidden Markov Model (HMM). An HMM is a pair of dependent stochastic processes where the first process is a hidden Markov chain while the second process is observable, i.e., its output states can be observed. In particular, the output of the second process depends on the output of the first process. We provide a formal definition below.

Definition 5.1 (Hidden Markov Model (HMM)). *A hidden Markov model (HMM)* HMM *is composed of:*

- *a Markov chain process* $\Theta = \{X_n \in [\#S] \; : \; n \geq 0\}$ *over a countable set* S *whose outputs are hidden,*
- *a stochastic process* $\Gamma = \{Y_n \in [\#T] \; : \; n \geq 0\}$ *over a countable set* T *that verifies,*

$$\Pr\left[Y_n = i \mid X_n = j\right] = \mathsf{O}_{i,j}$$

where $\mathsf{O} = (\mathsf{O}_{i,j})_{i \in [\#S], j \in [\#T]}$ *is the observation matrix.*

The observation matrix captures the probability of observing the jth value in T given that the hidden process is at the ith state in S. We characterize a hidden Markov model HMM with three parameters: (1) the transition matrix T of the hidden Markov chain process Θ; (2) the initial distribution μ of Θ; and (3) the observation probability O. We write $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$.

HMM inference. For the Decoder attack, we model the user's query distribution as the *hidden* Markov chain of an HMM and the corresponding leakage as the observable process. Given such an HMM, we are interested in solving the question above. In particular, given the observation (the leakage) and the Markov chain parameters (the query distribution), we can leverage existing results in stochastic processes to output the sequence of queries that best explains the observation. This can be efficiently solved using the Viterbi algorithm [52] which was first described in 1967 as a decoder for convolutional codes. At a high level, the Viterbi algorithm finds the query sequence $\mathbf{q}^\star$ that maximizes

$$\Pr\left[\mathbf{q} \mid \mathbf{o}, \mathsf{HMM}\right]$$

where $\mathbf{o}$ is the observed leakage and HMM is the hidden Markov model. We describe the Viterbi algorithm in Figure 8 of Appendix A, but at a high level, given an $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$ and a sequence of observation $\mathbf{o} = (o_1, \cdots, o_t)$, the Viterbi algorithm outputs a sequence $\mathbf{r} = (r_1, \cdots, r_t)$ where r_i is a state in the hidden Markov chain, for all $i \in [t]$.

We describe the Decoder attack in Figure 2 and provide more details below. Note that Decoder can be instantiated in different ways depending on how the observation matrix is constructed. In this work, we consider two ways to do this which results in two instantiations: Decoder-N and Decoder-B. These two variants achieve different recovery rates depending on the shape of the auxiliary distribution as we will see in Section 7.4. We take a top-down approach where we first describe the generic Decoder attack and then describe the two variants.

Attack overview. At a high level, Decoder is composed of two phases. The first phase, and by far the most challenging of the two, consists of computing the observation matrix O

[5] Note that there are more efficient ways to calculate π. We are assuming that the underlying solver makes use of the LU decomposition [50].

```
• Decoder($\boldsymbol{\ell}, \Theta$):
  (1) parse $\Theta$ as $(\mathsf{T}, \mu)$ and $\boldsymbol{\ell}$ as $(B_{i,j})_{i,j \in [t]}$;
  (2) initialize a map $\alpha : [t] \rightarrow \mathbb{W}$;
  (3) compute the stationary distribution
          $\pi = \pi \cdot \mathsf{T}$,
      if it exists, otherwise abort;
  (4) compute the set $\mathcal{I}$ of all unique queries from the leakage
          $\mathcal{I} := \left\{ i \ : \ B_{i,j} = 0, \text{for all } j < i \text{ and } i \in [t] \right\}$;
  (5) compute the query frequency $\mathbf{v} = (v_i)_{i \in \mathcal{I}}$ of queries
      such that for all $i \in \mathcal{I}$
          $v_i = \sum_{j \in [t]} B_{i,j}/t$;
  (6) $\boxed{\text{compute } \mathsf{O} \leftarrow \mathsf{Obv}(\pi, \mathbf{v});}$
  (7) set $\mathsf{HMM}$ to $(\mathsf{T}, \mathsf{O}, \mu)$;
  (8) set $\mathbf{o} := (o_1, \cdots, o_t)$ where for all $i \in [t]$,
      $o_i := j^\star$   where   $B_{i,j^\star} = 1$ and $B_{i,k} = 0 \ \forall k < j^\star$;
  (9) compute $\mathbf{r} \leftarrow \mathsf{Viterbi}(\mathsf{HMM}, \mathbf{o})$;
  (10) set $\alpha(i) := w_{r_i}$, for all $i \in [t]$;
  (11) output $\alpha$.
```

Figure 2: Our attack: Decoder.

of the (observable) Markov chain. Recall that we assume that the adversary only knows the query distribution but does not know the observation probabilities O. Now, given the observation matrix O, the attack has a complete description of an HMM which it then uses as input to the Viterbi algorithm. Ultimately, the accuracy of Decoder relies on two criteria: first, how well the observation matrix O captures the relationship between the hidden state and the observation state; and second, how accurate the inference algorithm is given the HMM and the concrete observation. The latter is handled by the Viterbi algorithm.[6] The former is harder to deal with because the observation matrix could be instantiated differently depending on the auxiliary distribution. We provide two possible instantiations, but first we give details on how the Decoder attack works.

Phase 1: Decoder takes as inputs the query equality pattern $\boldsymbol{\ell} := \mathsf{qeq}(\mathbf{q})$ of the client's query sequence and the Markov chain Θ. First, it parses the query equality as a square matrix $(B_{i,j})_{i,j\in[t]}$ where t denotes the length of the query sequence. It also parses the Markov chain Θ to get the transition matrix T and the initial distribution μ. In addition, it initializes a mapping $\alpha : [t] \rightarrow \mathbb{W}$. Given T, the attack computes the stationary distribution $\pi = (\pi_1, \cdots, \pi_m)$ where $m = \#\mathbb{W}$ is the number of keywords (or states) in $\mathbb{W}$. Similar to the Stationary attack, the attack aborts if the stationary distribution does not exist.[7] Next, the frequency, v_i, of each unique query $i \in \mathcal{I}$ is first calculated using the query equality

$$v_i = \sum_{j\in[t]} B_{i,j}/t.$$

Note that the set $\mathcal{I}$ corresponds to the set of unique queries in $\boldsymbol{\ell}$. In particular, given $(B_{i,j})_{i,j\in[t]}$, we can identify the position of the first time a query for a keyword $w \in \mathbb{W}$ is made. Then, the attack computes the observation matrix in line 6 of Figure 2. Note that the Obv function can be instantiated in various ways, but we specifically focus on two approaches: Obv-N, which is based on the ℓ_1-norm; and Obv-B, which is based on the binomial distribution.

Phase 2: the only remaining element to prepare before running the Viterbi algorithm is the sequence of observation, $\mathbf{o}$. Given the query equality pattern $(B_{i,j})_{i,j\in[t]}$, the attack builds the sequence of observation of length t as follows: first, it assigns every new query an index which is equal to the time the query is made. So if a query is made more than once, it will be mapped to the same position it was assigned to the first time it was made. More formally, for every $i \in [t]$, we have

$$o_i := j^\star \quad \text{where} \quad B_{i,j^\star} = 1 \text{ and } B_{i,k} = 0 \ \forall k < j^\star,$$

where $j^\star \in [i]$ represents the position the first time the ith query was made. Given the observation matrix O built in phase 1, the attack now has a complete set of parameters for a hidden Markov model $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$. Given HMM and the sequence of observation $\mathbf{o}$, the attack runs the Viterbi algorithm which outputs $\mathbf{r}$. The output represents the most likely sequence of visited states in the hidden process. Finally, it populates and outputs the mapping α such that the ith query maps to the r_i'th keyword, w_{r_i}, for all $i \in [t]$.

The ℓ_1-norm variant of Decoder**.** The Obv-N function builds on the observation that the number of times an adversary sees a given state in the observable stochastic process of the HMM is likely to equal the (known) stationary distribution of a specific keyword, and specifically, the one with the closest value. This same observation is leveraged by the Stationary attack and is formally captured by Lemma 4.3 which states that the average number of visits made to the ith state tends to the ith item of the stationary distribution for a given Markov chain.[8] We detail Obv-N in Figure 3 and it works as follows. For all $i \in [\#\mathbb{W}]$, for the jth unique query where $j \in [\#\mathbf{v}]$, if the distance between the frequency v_j and the stationary distribution of the ith state is less than ϵ, set $\mathsf{O}_{i,j}$ to $1 - |v_j - \pi_i|$. Otherwise, set $\mathsf{O}_{i,j}$ to 0. The matrix components are then normalized such that the sum of each row is equal to 1. Note that this is a requirement for a well-formed observation matrix. We made the choice of using the ℓ_1-norm, but other distances can be used for this phase as well. Note also that this variant makes use of an error parameter that is fixed throughout our implementation. We refer to the Decoder attack that makes use of the Obv-N function as Decoder-N.

The binomial variant of Decoder**.** The Obv-B function builds on the observation that the leakage, $\boldsymbol{\ell}$, can be viewed as a series of binomial experiments with different success values. In particular, the attack views the leakage as a sequence of $\#\mathcal{I}$ binomial experiments such that in every experiment, we fix the

[6] Note that one could replace the Viterbi algorithm inside our Decoder attack with any new algorithm that provides better efficiency and/or accuracy.

[7] Throughout all of our experiments, we were always able to calculate the stationary distribution.

[8] Note that this observation applies to our case since the number of states in the observable stochastic process is smaller or equal to the number of states in the hidden one, but also because we know that every hidden state can only map to a single observable state. This holds true because the query equality pattern is a permutation function.

```
• Obv-N($\pi, \mathbf{v}$):
  (1) parse $\mathbf{v}$ as $(v_i)_{i \in \mathcal{I}}$ and $\pi$ as $(\pi_i)_{i \in [\#\mathbb{W}]}$,
  (2) for all $i \in [\#\mathbb{W}]$,
      (a) initialize a counter count := 0;
      (b) for all $j \in \mathcal{I}$,
          (i) if $|v_j - \pi_i| \leq \epsilon$, then set $\mathsf{O}_{i,\mathsf{count}} := 1 - |v_j - \pi_i|$
              and 0 otherwise;
          (ii) increment count;
      (c) set $\lambda_i := \sum_{j \in [\#\mathcal{I}]} \mathsf{O}_{i,j}$;
      (d) for all $j \in [\#\mathcal{I}]$, set $\mathsf{O}_{i,j} := \mathsf{O}_{i,j} / \lambda_i$;
```

Figure 3: The Obv-N **variant.**

```
• Obv-B($\pi, \mathbf{v}$):
  (1) parse $\mathbf{v}$ as $(v_i)_{i \in \mathcal{I}}$ and $\pi$ as $(\pi_i)_{i \in [\#\mathbb{W}]}$,
  (2) for all $i \in [\#\mathbb{W}]$,
      (a) initialize a counter count := 0;
      (b) for all $j \in \mathcal{I}$,
          (i) set $\mathsf{O}_{i,j} = \binom{t}{t \cdot v_j} \cdot \pi_i^{t \cdot v_j} \cdot (1 - \pi_i)^{t - t \cdot v_j}$;
          (ii) increment count;
      (c) set $\lambda_i := \sum_{j \in [\#\mathcal{I}]} \mathsf{O}_{i,j}$;
      (d) for all $j \in [\#\mathcal{I}]$, set $\mathsf{O}_{i,j} := \mathsf{O}_{i,j} / \lambda_i$;
```

Figure 4: The Obv-B **variant.**

number of successes to the number of times a specific query has been queried for. The attack then assigns the observation matrix component to the corresponding binomial probability mass function. More formally, given a fixed keyword w_i, we consider its corresponding stationary distribution π_i as the parameter of the binomial distribution, for all $i \in [\#\mathbb{W}]$. And we consider the sequence length t as the number of trials in each experiment. Then for every unique query j, we know from the leakage that it has been queried $t \cdot v_j$ times. The attack then sets

$$\mathsf{O}_{i,j} = \binom{t}{t \cdot v_j} \cdot \pi_i^{t \cdot v_j} \cdot (1 - \pi_i)^{t - t \cdot v_j}.$$

The matrix components are then normalized such that the sum of each row is equal to 1, refer to Figure 4 for a detailed description. The rationale behind this variant can also be explained through the lens of the Stationary attack. In particular, one needs to first observe that the binomial probability mass function reaches its maximum value when the number of successes is equal to the expected value, which is equal to $t \cdot \pi_i$. This means that $\mathsf{O}_{i,j}$ attains its maximum value if and only if $t \cdot \pi_i \approx t \cdot v_j$ which again shows the relationship to Lemma 4.3. We refer to the Decoder attack that makes use of the Obv-B function as Decoder-B.

Remark. Both variants share the property that the components of the observation matrix attain their maximum values, for a given row, at exactly the same indices. However, contrary to the ℓ_1-norm variant, the binomial variant never assigns a zero to any component in the matrix which leaves more possible sequences for the attack to output. This is especially helpful in cases where the sequence length t is small or when the leakage resulted from an unlikely observation.

Efficiency. Given a query sequence of length t and a keyword space $\mathbb{W}$ of size m, the running time of the Decoder attack is

$$O\big(m^2 \cdot (m + t)\big).$$

The asymptotic calculation can be broken down into four main parts:

- *(part 1)* computing the stationary distribution π takes $O(m^3)$ steps,
- *(part 2)* calculating the query frequency $\mathbf{v}$ takes $O(t)$ steps,
- *(part 3)* populating the observation matrix O takes $O(m^2)$ steps for both variants,
- *(part 4)* computing the Viterbi algorithm takes $O(m^2 \cdot t)$ steps.

Note that computing the Viterbi algorithm and the stationary distribution are the most expensive parts of the attack.

6 From Distributions to Samples

In this section, we transform our known-distribution attacks into known-sample versions, which we call Stationary-Smpl and Decoder-Smpl. [9] In the following, we first describe the different forms of auxiliary data our attacks can take as input.

Adversarial knowledge. Assume that the adversary is given as auxiliary data aux, a sequence of queries $\mathbf{q} := (q_1, \cdots, q_n)$ where the queries are for keywords in a set $\mathbb{W}'$ which can be different from the client's keyword space $\mathbb{W}$. In this case, we write aux $= (q_1, \cdots, q_n)$. We also consider a more general setting where the auxiliary data is composed of multiple query sequences aux $= \big(\mathbf{q_1}, \cdots, \mathbf{q_p}\big)$ where $\mathbf{q}_i = (q_{i,1}, \cdots, q_{i,t_i})$, for all $i \in [p]$. Moreover, we assume that the auxiliary query sequences are sampled from of a Markov chain process.

Learning the Markov chain. Our known-sample attacks build on the Stationary and the Decoder attacks. As described in Sections 4 and 5, these attacks take as input the query equality pattern and a Markov chain. However, along with the query equality pattern, the adversary only has a sequence of queries as input. A natural question then arises:

Can we learn a Markov chain knowing only its realization?

Similar to the decoding attack, learning the parameters of a Markov chain given some observation is one of the most fundamental inference problems in the area of hidden Markov models. The well-known Baum-Welch (BW) algorithm [53] can efficiently find the HMM parameters that maximizes the probability of making a given observation. At a high level, BW outputs the parameters $\mathsf{HMM}^{\star}$ that maximizes the following quantity

$$\Pr\big[\mathbf{o} \mid \mathsf{HMM}\big]$$

where $\mathbf{o}$ is a sequence of observations. In particular, we use BW to generate the transition matrix T which we then feed to either Stationary or Decoder. We provide the details of the Baum-Welch algorithm in Figure 9 of Appendix A. We describe the Stationary-Smpl and Decoder-Smpl attacks in Figure 5.

[9] We similarly denote by Decoder-N-Smpl and Decoder-B-Smpl the known-sample versions of the Decoder attack when the observation function is Obv-N and Obv-B, respectively. This notation will become helpful when describing our experimental results.

```
• $\star$-Smpl($\boldsymbol{\ell}$, aux):
  (1) initialize an empty map $\alpha^\star$ and set $s^\star := \infty$;
  (2) for all $\mathbf{q} \in$ aux,
      (a) compute HMM $\longleftarrow$ Baum-Welch($\mathbf{q}$);
      (b) parse HMM as $(\mathsf{T}, \mathsf{O}, \mu)$ and set $\Theta := (\mathsf{T}, \mu)$;
      (c) compute $(\alpha, s) \longleftarrow$ Stationary($\boldsymbol{\ell}, \Theta$);
      (d) if $s < s^\star$, set $\alpha^\star := \alpha$ and $s^\star := s$;
  (3) output $\alpha^\star$.
```

Figure 5: Our inference attack: $\star$-Smpl, where $\star$ is a placeholder for Stationary **and** Decoder**.**

Attack overview. Both Stationary-Smpl and Decoder-Smpl take as inputs the query equality qeq and the auxiliary data aux. . For every query sequence $\mathbf{q}$ in aux, the attacks run Baum-Welsh algorithm to learn the Markov chain $\Theta_{\mathbf{q}}$. Then the attacks simply run their corresponding known data attacks as a subroutine. The main difference is that instead of only outputting a mapping α, the subroutine attacks also output an error score s. This score can be viewed as a metric that quantifies the quality of the mapping. The smaller the score, the better the mapping. In the following, we describe how the score is calculated for both the Stationary and Decoder attacks:

- for the Stationary attack, the score s is calculated by summing the 1-norm distance between the query frequency v_i and the chosen stationary probability π_ρ such that, for all $i \in [t]$,

  $$s := s + |v_i - \pi_\rho|.$$

  The score is then equal to the sum of all the distances between v_i and π_ρ, where ρ corresponds to the position in π that minimizes the distance $|v_i - \pi_j|$.[10]
- for the Decoder attack, we also introduce a score that tries to capture the accuracy of the Viterbi algorithm. The idea is similar to the above but tries to measure the accuracy of the chosen Viterbi path. We refer to the reader to Figure 8 in Appendix A for more details.

For every iteration, every attack outputs a mapping α and a score s. The attacks compare the new score s to the previous smallest score $s^\star$, if $s < s^\star$, the attack changes its preferred mapping to α and sets $\alpha^\star := \alpha$.

Efficiency. The efficiency of Stationary-Smpl and Decoder-Smpl is similar to the one of Stationary and Decoder, respectively, except for the additional cost of the Baum-Welsh (BW) algorithm. BW has a running time equal to $O(m_i^2 \cdot t_i)$ where m_i is the size of the keyword space of the ith query sequence $\mathbf{q}$ in aux, whereas t_i is the length of $\mathbf{q}_i$. To sum up, the time complexity of Stationary-Smpl is equal to $O\big(m \cdot (m^2 + t) + \sum_{i=1}^{p} m_i^2 \cdot t_i\big)$, whereas the time complexity of Decoder-Smpl is equal to $O\big(m^2 \cdot (m+t) + \sum_{i=1}^{p} m_i^2 \cdot t_i\big)$, where p is the number of query sequences in aux. If we assume that $m_i = m$ and $t_i = t = O(m)$, for all $i \in [p]$, then Stationary-Smpl and Decoder-Smpl attacks have a running time equal to $O\big(p \cdot m^3\big)$. Even though the running time of both attacks is polynomial in m, they can be prohibitive in practice. We noticed this during our evaluations as we struggled to scale our attacks to large keyword spaces. As an example, for keyword spaces of size 1, 000 and auxiliary sequences aux composed of 10 query sequences, both attacks require around 2^{48} steps. There are ways to reduce the overhead by using more efficient variants of BW, Viterbi and of the computation of the stationary distribution, but this would lead to a loss in accuracy.

7 Empirical Evaluation

In this section, we evaluate our known-distribution and known-sample attacks across a wide variety of scenarios and use both real-world query logs and synthetic query distributions. We implement and evaluate our attacks using the LEAKER framework [32] and we compare the recovery rates of our attacks with the ones of the IHOP attack [45]—the only currently-known attack that exploits the query equality pattern under dependent queries. We start by briefly describing our implementation, our query logs, query distributions, and our evaluation setting. We then describe our results before finally providing our takeaways on the various risks our attacks pose.

7.1 Implementation

We implemented our attacks in Python 3.9 and added it as an extension to the open-source LEAKER [32] framework which already implements 15 leakage attacks found in more than 12 different papers. LEAKER's modular design allowed us to integrate and comparatively evaluate the effectiveness of our attacks under different assumptions using queries from multiple query logs and distributions.

While LEAKER has several modules for exact and range search attacks, it only supports the independent query generation. As a result, we extended LEAKER to support dependent queries and attack evaluations in this setting. We implemented Stationary, Decoder with its two variants Decoder-N and Decoder-B, Stationary-Smpl, Decoder-Smpl with its two variants Decoder-N-Smpl and Decoder-B-Smpl, and finally IHOP which totaled 2, 595 lines. All the implementations can be found in the following private repository [4]. Upon acceptance of our paper, we will open a pull request in order to merge our results with the LEAKER framework.

7.2 Query Logs and Query Distributions

In this section, we describe the query logs as well as the synthetic query distributions we used in our evaluation.

Query logs. No prior work evaluated their attacks on real-world query logs. In particular, the evaluation of the IHOP attack [45] used the url links in Wikipedia pages to model client query distributions.

For our evaluation, we use the following two publicly available query logs from [32]:

- AOL is a publicly available search engine query log [46] that contains web searches. AOL contains 52M queries that were issued by 656 thousand users between March

[10] While there is a correlation between a small score s and the accuracy of the mapping α, it is not however an implication. It is possible to have a score $s = 0$ and the accuracy of the mapping being completely off. So our decision to pick the mapping with the smallest score is a heuristic decision.

1^{st} and May 31^{st}, 2006. The total number of unique keywords is 2.9M.
- *The Arabidopsis Information Resource* [19], or TAIR, is a publicly available query log for plant genetic annotations containing 650 thousand unique keywords issued by 1.3 thousand users between January 1^{st}, 2012 and April 30^{th}, 2013. The total number of unique keywords is 14K.

Each query log can be viewed as a sequence $\mathbf{q}$ itself composed of several client query sequences, $\mathbf{q}_u$, for $u \in \mathbb{U}$, where $\mathbb{U}$ denotes the set of users in the log. Given $\mathbf{q}$, LEAKER's *pre-processor* module parses, tokenizes, extracts, stems and removes stop words. In the case of the AOL query log, we also discarded the queries issued by the $1,000$ most active users because their query behavior suggested that they were bots.

Query distributions. We consider four synthetic query distributions all of which are Markov chains with various transition matrices. The first distribution, Uniform, captures settings where all the keywords can be queried with uniformly distributed transition probabilities. In particular, in this case, all the query sequences are possible. The second distribution, Zipf, is similar to Uniform in the sense that all query sequences are possible, but some transitions are more likely than others. The next two distributions capture a different setting where, given a current keyword, the next query can only be made from a subset of all possible keywords. In other words, some keywords are *unreachable*. This constraint results in creating sparsity in the transition matrix and we control its degree in two different ways. For Binomial-Zipf, we assume that the number of possible transitions (non-zero values) per row follows a Binomial distribution, whereas for Zipf-Zipf, we assume that the number of possible transitions is Zipf distributed. We describe all the distributions below, where we show how to generate their transition matrices.[11]

- Uniform: for every $i, j \in [n]$, compute $\mathsf{T}_{i,j} \overset{\$}{\leftarrow} \{0,1\}^k$.
- Zipf: for every $i \in [n]$, we pick a permutation $\beta_i : [n] \longrightarrow [n]$ at random. We then set $\mathsf{T}_{i,j} := f_{s,n}(\beta_i(j))$ for all $j \in [n]$, where $f_{s,n}$ is the probability mass function of the Zipf distribution with parameter $s \geq 0$ and support size n,
$$f_{s,n}(k) = \frac{k^{-s}}{H_{n,s}},$$
where $H_{n,s} = \sum_{i=1}^{n} i^{-s}$ is the general harmonic number. In our experiments, we set $s = 2$.
- Binomial-Zipf: for every $i \in [n]$, sample a permutation $\beta_i : [n] \longrightarrow [n]$ at random. And for all $j \in [n]$, sample a value $\theta_{i,j} \overset{\$}{\leftarrow} \mathsf{Bernoulli}(p)$ where $0 \leq p \leq 1$. If $\theta_{i,j} = 1$, then set $\mathsf{T}_{i,j} := f_{s,n}(\beta_i(j))$, otherwise set $\mathsf{T}_{i,j} := 0$. In our evaluation, we set $p = 0.5$ and $s = 2$. Observe that the number of non-zero transition probabilities follows a Binomial distribution.
- Zipf-Zipf$_\nu$: first, for every $i \in [n]$, sample a Zipf value $\theta_i \overset{\$}{\leftarrow} \mathsf{Zipf}(s, n)$, then compute $\theta_i := \theta_i + \nu$ which represents the number of non-zero transitions the ith state can have. We then sample a permutation $\beta_i : [n] \longrightarrow [n]$ at random, select a set $S_i = \{j_1, \cdots, j_{\theta_i}\}$ of size θ_i at random from $\{1, \cdots, n\}$, and compute $\mathsf{T}_{i,j_p} := f_{s,n}(\beta_i(j_p))$ for all $p \in [\theta_i]$, and $\mathsf{T}_{i,j} := 0$ otherwise.

Scenarios	Leakage qeq($\mathbf{q}$)		Auxiliary $\mathbf{s}$
	Known $\mathbf{q}$	Sampled $\mathbf{q}$	
Exact (E)	–	$\mathbf{q} \leftarrow \mathsf{BW}(\mathbf{q}_i)$	$\mathbf{q}_i$
All (A)	–	$\mathbf{q} \leftarrow \mathsf{BW}(\mathbf{q}_i)$	$(\mathbf{q}_1, \cdots, \mathbf{q}_n)$
Split (S)	$\mathbf{q}_{i\|2}$	$\mathbf{q} \leftarrow \mathsf{BW}(\mathbf{q}_{i\|2})$	$\mathbf{q}_{i\|1}$
Other (O)	$\mathbf{q}_i$	$\mathbf{q} \leftarrow \mathsf{BW}(\mathbf{q}_i)$	$(\mathbf{q}_1, \cdots, \mathbf{q}_{i-1}, \mathbf{q}_{i+1}, \cdots, \mathbf{q}_n)$

Table 2: Summary of the evaluation setup for known-sample attacks to recover the queries of the *i*th user.

The initial distribution μ is the same for all distributions and is simply $\mu = (1/n, \cdots, 1/n)$, i.e., all keywords are equally likely to be queried at the beginning.

Sparsity. To measure the sparsity of a given distribution, we compute the minimum Hamming weight (HW) of a Markov chain as follows. First, we compute the HW of the ith state which is the number of non-zero transition probabilities in T_i, for $i \in [n]$. The minimum HW of a Markov chain is then simply the minimum HW across all states. We later show in Section 7.4 that varying the sparsity of the transition matrix significantly impacts the accuracy of our known-distribution attacks.

Note. Using synthetic query distributions is by no means an ideal setup, but it is unfortunately our only option due to the scarcity of publicly-available query logs. The query logs we run our attacks against are a great resource but they are limited and do not necessarily capture the most common query distributions. We try to fill this gap with synthetic query distributions so that we can better understand how the attacks behave in various scenarios. Note that the four distributions described above are not exhaustive, but they were carefully crafted to capture different properties of the transition matrix which, we believe, can impact the recovery rate of the attacks. Such properties include, at a high level, the degree of connectivity between the states, the level of sparsity and different shapes of the stationary distributions.

7.3 Evaluation Setup

We evaluate our known-distribution attacks on synthetic query distributions, and we evaluate our known-sample attacks on the publicly available query logs described in Section 7.2. In the following, we describe the evaluation setup for each setting.

Known-distribution setting. For every query distribution, we sample a query sequence $\mathbf{s}$ and compute the query equality pattern on it. Sampling from a Markov chain works as follows. We pick an initial state uniformly at random from all possible states and, for every transition, we pick the next state based on the probabilities of the transition matrix. The adversary is then given the qeq on the sampled query sequence $\mathbf{s}$ and the query distribution.

Known-sample setting. We consider four different cases that capture different scenarios. For all scenarios, we need to specify the target client/user (say ith user). Moreover, in

[11] Note that the details of row normalization are straightforward and therefore skipped from the description below.

every case except for the first and the second, we consider the leakage to be either: *fixed* or *sampled*. For the former, we generate the leakage by computing the qeq on (a subset of) the ith user's query log sequence. For the latter, we: (1) learn the ith user's query distribution by applying the Baum-Welsh algorithm to (a subset of) its query log sequence; (2) sample a new query sequence $\mathbf{s}$ from the learned Markov chain; and (3) compute the qeq on sampled sequence $\mathbf{s}$. Note that all the following four cases apply to both the AOL and TAIR query logs. We summarize these four cases in Table 2 and describe them below.

- *Exact* (E): the adversary receives as auxiliary sequence the query log sequence $\mathbf{q}_i$ of the ith user. Moreover, in this case, we only consider the *sampled leakage* setting where the adversary receives as leakage the query equality pattern of a query sequence $\mathbf{s}$ sampled from the distribution learned from $\mathbf{q}_i$ using BW.
- *All-Users* (A): the adversary receives as auxiliary sequence the query log sequences of all users including of the the ith user (the target user).
  In this case, we only consider *sampled leakage* where the adversary receives the query equality pattern of a query sequence $\mathbf{s}$ sampled from a distribution learned from the ith user's query log sequence $\mathbf{q}_i$.
- *Split* (S): the adversary receives as auxiliary sequence the first half of the ith user's query log sequence, $\mathbf{q}_{i|1}$. In the fixed leakage setting, the adversary also receives the query equality pattern of the second half of the ith user's query log sequence, $\mathbf{q}_{i|2}$. In the sampled leakage setting, the adversary receives the query equality pattern on a query sequence $\mathbf{s}$ sampled from a distribution learned from the second half of the ith user's query log sequence, $\mathbf{q}_{i|2}$.
- *Other-Users* (O): the adversary receives as auxiliary sequence the query logs sequences of all users *except* for the ith user (the target user). The query equality pattern that the adversary receives in both the *fixed* and *sampled* leakage settings is similar to the *All-Users* case.

These settings are listed in decreasing strength with respect to adversarial knowledge and may correspond to certain real-world events which we describe below:

- Exact (E): This scenario captures a setting in which the adversary (possibly the server) is able to compromise a user's machine for a long period of time and obtain its query log. After the period of compromise, the adversary can only observe the query equality pattern on the new queries issued by the user.
- All-Users (A): In this scenario, the adversary not only obtains the query log of the target user, but a set of users in the system. This can occur by compromising all the users' machines for a period of time. Similar to the above, after the period of compromise, the adversary can only observe the query equality pattern on new queries issued by the *target* user.
- Split (S): This scenario is similar to the *Exact* scenario. The difference however is in how the user generates its queries after the period of compromise, please refer to the paragraph above for more details.
- Other-Users (O): In this scenario, the adversary is not able to compromise the target user's machine but is able to compromise the other users' machines.

Experimental setup. Our experiments were run on an Ubuntu 20.04 machine with 390GB memory and 1TB disk space. When evaluating our known-data attacks, we varied the following parameters. First, for all query distributions, we varied the size of the keyword space, m, from 250 up to $1,500$. Second, we varied the sparsity of the transition matrix for the Zipf-Zipf$_\nu$ query distribution by varying the minimum Hamming weight from 5 to 450; here, the size of the keyword space is fixed to 500. Finally, we also varied the size of the sampled query sequence, t, from $1,000$ up to $5 \cdot 10^5$ queries[12] . We run each attack 30 times and report the median, maximum and minimum recovery rates. The recovery rate is simply the fraction of correctly recovered queries over the length of query sequence.

When evaluating our known-sample attacks, we proceeded as follows. First, for each query log, we selected 10 users. These users were fixed for the entire set of experiments. These users were selected carefully so that their respective query log sequences had between 500 and $1,000$ unique keywords.[13] While there were more than 10 users that verified this condition, we just picked 10 arbitrarily for feasibility. For the AOL query log, the number of unique keywords per user varies between 313 and 782, whereas for TAIR it varies between 587 and 848. We then selected 5 users from the 10 as target users. All our results are the average recovery rate of running the attacks against each individual user of the selected 5 users. We run these attacks 10 times per user and report the median, maximum and minimum recovery rates over all 5 attacked users. Note that these 5 users are again selected and fixed throughout the entire experiment and that our experiments target users individually but present aggregated results.

7.4 Experimental Results

The recovery rate of our known-sample attacks with sampled and fixed leakage are given in Figure 6 and Table 3, respectively. The recovery rates of our known-distribution attacks are in Figure 7 and Figure 11 in Appendix B. Due to space limitations, we only report the results for the Zipf-Zipf distribution in the main body of the paper. We now summarize our findings with a focus on the median metric.

Known-sample attacks with sampled leakage. Our results with sampled leakage (cf. Figure 6) show that our known-sample attacks achieve their best recovery rates in the *All-Users* and *Exact* settings. In particular, in the latter, Decoder-N-Smpl achieves 99.1% recovery rate (cf. Table 1) with TAIR and 53.7% with AOL for query sequence of size $50,000$. The Decoder-B-Smpl attack behaves similarly but with a slight increase in the case of the AOL dataset where it achieves 98.6% recovery rate

[12] While users may not issue a large number of queries, we use a wide range to identify where the attacks do and do not work. A similar range was also used to evaluate the IHOP attack [45].

[13] We were limited to such a small number of keywords because of the non-trivial computational overhead of our attacks as well as the IHOP attack.

Data Source	Scenario	Max. Efficacy per Attack				Median Efficacy per Attack			
		IHOP [45]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl	IHOP [45]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl
TAIR [19]	Other (0)	**5.4%**	4.7%	2.1%	2.3%	3.0%	**3.8%**	1.4%	1.7%
	Split (S)	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
AOL [46]	Other (0)	**0.4%**	0.0%	0.1%	0.1%	0.0%	0.0%	0.0%	0.0%
	Split (S)	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table 3: Results summary of our attacks and the IHOP attack [45] in the *known-sample setting* with *fixed* leakage.

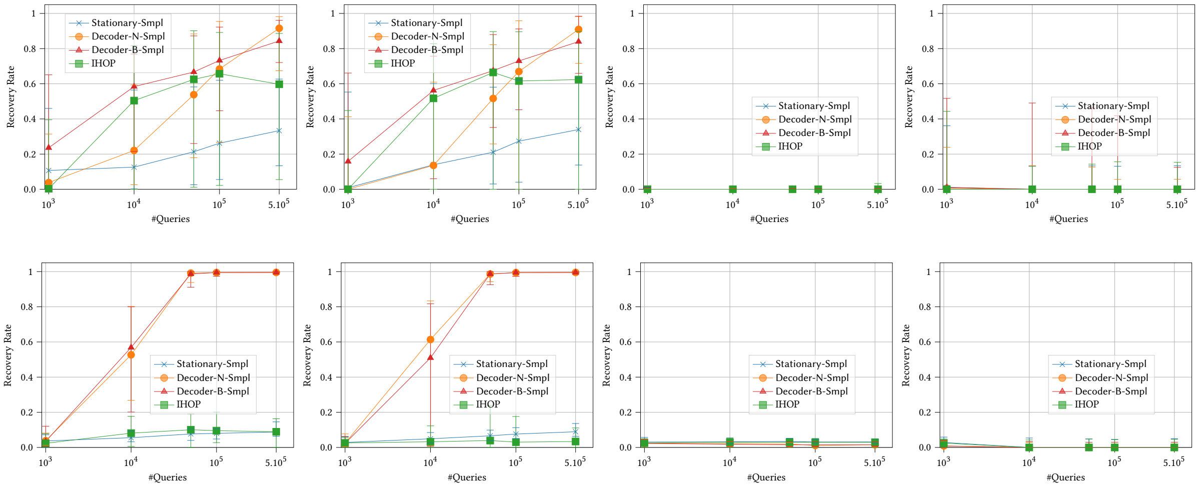
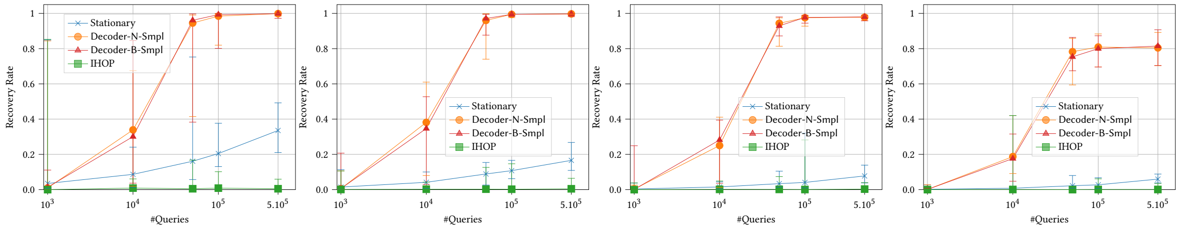


Figure 6: Our new attacks and the IHOP attack [45] evaluated against AOL (top) and TAIR (bottom) in the *known-sample* setting for the *Exact*, *All*, *Other* and *Split* scenarios (from left to right).



250 States

500 States

1 000 States

1 500 States

Figure 7: Our new attacks and the IHOP attack [45] evaluated on the Zipf-Zipf query distribution. All evaluations are done using a *fixed* minimum Hamming weight = 2.

with TAIR and 66.7% with AOL. This is compared to 10.1% and 62.5% for IHOP with the same datasets. When increasing the size of the query sequence to 500, 000, Decoder-N-Smpl and Decoder-B-Smpl have a median recovery rate of approximately 99% with TAIR and approximately 90% and 85%, respectively, with AOL compared to about 10% and 60% for IHOP. That is, the larger the query sequence the more accurate the two variants of the Decoder-Smpl become on both query logs. We also note that two variants of the Decoder-Smpl attack together significantly outperform IHOP with TAIR and AOL for query sequences of any size. Stationary performs worse than the other attacks in the *Exact* setting, achieving a median recovery rate of 7.6% and 21.3% with TAIR and AOL, respectively.

All the attacks achieve poor results in the *Split* and *Other* settings—the most realistic settings of our work. In particular, Stationary-Smpl has a median recovery rate of 3.4% which is the highest among all the attacks on TAIR. For AOL, all the attacks achieve a recovery rate smaller than 1% with IHOP achieving the highest recovery rate of 0.8%.

Overall, the two variants of the Decoder-Smpl attack outperform together the IHOP attack and the Stationary-Smpl attack in almost all instances.

Known-sample attacks with fixed leakage. Recall that in this case, we only consider the *Split* and *Other* settings (cf. Table 3). We observe that none of the attacks worked in the *Split* and *Other* settings. In the *Split* setting, the attacks had a 0% recovery rate. In the *Other* setting, the Stationary-Smpl had the best median recovery rate of 3.8% which was achieved with TAIR.

Known-distribution attacks. We present the results for our known-distribution attacks on the Zipf-Zipf distribution in Figure 7 where the minimum Hamming weight is 2. We observe that the two variants of the Decoder attack significantly outperform all the other attacks for any number of states. Starting from query sequences of length 50, 000, both Decoder-N and Decoder-B achieve a 99% recovery rate.

In Figure 10 in Appendix B, we varied the Hamming weight and fixed the number of states to 500. We observed that an increased Hamming weight decreased the recovery rate of the attacks. Starting with Hamming weight of 100, the maximum recovery rate is already below 20%. This suggests the "denser" transition matrices may be harder to attack even when the adversary knows the client's query distribution. Finally, refer to Figure 11 of Appendix B for more results on the three other query distributions Uniform, Zipf and Binomial-Zipf. Overall, we obtained lower recovery rates for these distributions because, we believe, their transition matrices are significantly denser.

A note on the Decoder **variants.** Our evaluation shows that the recovery rates of Decoder-N and Decoder-B, and similarly, of Decoder-N-Smpl and Decoder-B-Smpl, are similar throughout the scenarios. The only exception was for AOL in the sampled leakage setting, where we observed that Decoder-B-Smpl does better when the length of the query sequences is smaller than 100, 000.

Summary. Our evaluation shows that Decoder-N-Smpl and Decoder-B-Smpl in the *Exact* and *All-Users* settings with sampled leakage,[14] and Decoder-N and Decoder-B with a known distribution achieve the best recovery rates. However, despite our attacks outperforming the state-of-the-art in most cases, it is also fair to say that, from our results, none of the attacks work in the more realistic settings such as the *Split* and *Other* settings. It thus remains open to find attacks that work in more realistic settings where the attacker does not know the exact query sequence of the user.

Additionally, attacks with significant recovery rates required query sequences with at least 10, 000 queries, refer to Figure 6. None of our real-world query logs had sequences of such length so we could only simulate longer query sequences (when either sampling the queries in the known-sample setting or when working in the known distribution setting). In particular, in AOL [46], the most active user issued 815 queries and the average sequence length is 79. In TAIR [19], the most active user issued 2, 059 queries and the average sequence length is 500. Assessing the efficacy of our attacks on real query logs with longer query sequences is an important future work. However, we would like to emphasize that studying the efficacy of our attacks on query sequences of smaller lengths is still very important and that there is no setting which is more realistic than another. This is for several reasons. First, ESA deployments can be used in variety of settings where users could generate query sequences with different lengths over a given period of time. Another reason is that, as proposed in [31], the query complexity of an attack can be used to set the query capacity of an ESA; where the query capacity is the number of queries that can be executed before the underlying structure is rebuilt. In such a setting, knowing how an attack performs even on relatively small query sequences is important.

Moreover, it is important to highlight that the computational overhead of all the attacks—ours included—could make them prohibitive in practice. In fact, in order to run our experiments, we had to choose specific users in the query logs and use a small number of states in the known-distribution setting otherwise the evaluation would take many years to complete. For example, evaluating our attacks in the *Others* scenario with sampled leakage took almost 38 hours to complete.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful feedback and specifically for suggesting we consider using the binomial distribution to build the observation matrix. This led to the design of the binomial variant of the Decoder attack. This project received funding from the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230. This project was carried out using the African Super-Computer Center HPC service, supported by Mohammed-VI Polytechnic University.

References

[1] R. Ada Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles (SOSP)*. 85–100.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *International Conference on Management of Data (SIGMOD)*.

[3] Ghous Amjad, Seny Kamara, and Tarik Moataz. 2019. Breach-Resistant Structured Encryption. In *Proceedings on Privacy Enhancing Technologies (Po/PETS '19)*.

[4] Anonymized. 2023. Private repository. https://github.com/anonymous-repo-submission/artifact.

[5] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. 2007. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference (CRYPTO)*.

[6] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2020. Revisiting Leakage Abuse Attacks. In *Network and Distributed System Security Symposium (NDSS '20)*.

[7] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference (TCC)*.

[8] R. Bost. 20016. Sophos - Forward Secure Searchable Encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*.

[9] R. Bost, B. Minaud, and O. Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*.

[10] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. 2015. Leakage-Abuse Attacks against Searchable Encryption. In *ACM Conference on Communications and Computer Security (CCS '15)*. ACM, 668–679.

[11] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *Network and Distributed System Security Symposium (NDSS '14)*.

[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology - CRYPTO '13*. Springer.

[13] Yan-Cheng Chang and Michael Mitzenmacher. 2005. Privacy preserving keyword searches on remote encrypted data. In *International Conference on Applied Cryptography and Network security (ACNS)*.

[14] M. Chase and S. Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT '10 (Lecture Notes in*

[14] We would like to note that the Decoder-B-Smpl attack achieves a better recovery rate when the query sequence was less than 100, 000 queries.

Computer Science, Vol. 6477). Springer, 577–594.

[15] Melissa Chase and Seny Kamara. 2010. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*.

[16] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*. ACM, 79–88.

[17] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[18] Marc Damie, Florian Hahn, and Andreas Peter. 2021. A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents. In *USENIX Security Symposium (USENIX Security)*.

[19] Maria Esch, Jinbo Chen, Stephan Weise, Keywan Hassani-Pak, Uwe Scholz, and Matthias Lange. 2014. A query suggestion workflow for life science IR-systems. *Journal of Integrative Bioinformatics* 11, 2 (2014).

[20] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. 2017. SoK: Cryptographically protected database search. In *IEEE Symposium on Security and Privacy (S&P)*.

[21] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC)*.

[22] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1038–1055.

[23] E-J. Goh. 2003. *Secure Indexes*. Technical Report 2003/216. IACR ePrint Cryptography Archive. See http://eprint.iacr.org/2003/216.

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*.

[25] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996).

[26] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2019. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE Symposium on Security and Privacy (S&P)*.

[27] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. 2021. Rethinking Searchable Symmetric Encryption. *IACR ePrint* 879 (2021).

[28] M. Saiful Islam, M. Kuzu, and M. Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *Network and Distributed System Security Symposium (NDSS '12)*.

[29] Mireya Jurado, Catuscia Palamidessi, and Geoffrey Smith. 2021. A Formal Information-Theoretic Leakage Analysis of Order-Revealing Encryption. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. 1–16. https://doi.org/10.1109/CSF51468.2021.00046

[30] Mireya Jurado and Geoffrey Smith. 2019. Quantifying Information Leakage of Deterministic Encryption. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop* (London, United Kingdom) *(CCSW'19)*. Association for Computing Machinery, New York, NY, USA, 129–139. https://doi.org/10.1145/3338466.3358915

[31] S. Kamara. 2015. Encrypted Search. *XRDS* 21, 3 (March 2015), 30–34. https://doi.org/10.1145/2730908

[32] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. 2022. SoK: Cryptanalysis of Encrypted Search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[33] S. Kamara and T. Moataz. 2019. Computationally Volume-Hiding Structured Encryption. In *Advances in Cryptology - Eurocrypt' 19*.

[34] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. 2018. Structured Encryption and Leakae Suppression. In *Advances in Cryptology - CRYPTO '18*.

[35] S. Kamara and C. Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Financial Cryptography and Data Security (FC '13)*.

[36] S. Kamara, C. Papamanthou, and T. Roeder. 2012. Dynamic Searchable Symmetric Encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press.

[37] G. Kellaris, G. Kollios, K. Nissim, and A. O' Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *ACM Conference on Computer and Communications Security (CCS '16)*.

[38] Evgenios M Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. 2022. Leakage Inversion: Towards Quantifying Privacy in Searchable Encryption. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1829–1842.

[39] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2021. Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1502–1519.

[40] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. 2014. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences* 265 (2014).

[41] M. Naveed, S. Kamara, and C. V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *ACM Conference on Computer and Communications Security (CCS)* (Denver, Colorado, USA) *(CCS '15)*. ACM, 644–655. https://doi.org/10.1145/2810103.2813651

[42] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H Deng. 2021. LEAP: Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[43] James R Norris. 1998. *Markov chains*. Number 2. Cambridge university press.

[44] Simon Oya and Florian Kerschbaum. 2021. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *USENIX Security Symposium (USENIX Security)*.

[45] Simon Oya and Florian Kerschbaum. 2022. IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization. In *USENIX Security Symposium (USENIX Security)*.

[46] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *International Conference on Scalable Information Systems (INFOSCALE)*.

[47] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 79–93. https://doi.org/10.1145/3319535.3354213

[48] Karl Pearson. 1905. The problem of the random walk. *Nature* 72, 1865 (1905), 294–294.

[49] Ruben Groot Roessink, Andreas Peter, and Florian Hahn. 2021. Experimental review of the IKK query recovery attack: Assumptions, recovery rate and improvements. In *International Conference on Applied Cryptography and Network Security (ACNS)*.

[50] Alex Schwarzenberg-Czerny. 1995. On matrix factorization and efficient least squares solution. *Astronomy and Astrophysics Supplement, v. 110, p. 405* 110 (1995), 405.

[51] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy (S&P)*.

[52] Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13, 2 (1967), 260–269.

[53] Lloyd R Welch. 2003. Hidden Markov models and the Baum-Welch algorithm. *IEEE Information Theory Society Newsletter* 53, 4 (2003), 10–13.

[54] Wikimedia Foundation. 2022. Wiktionary Statistics. https://en.wiktionary.org/wiki/Special:Statistics. Accessed 2022-10-31.

[55] Charles V. Wright and David Pouliot. 2017. Early Detection and Analysis of Leakage Abuse Vulnerabilities. *IACR Cryptol. ePrint Arch.* (2017), 1052. http://eprint.iacr.org/2017/1052

[56] Andrew C Yao. 1982. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science (FOCS)*.

A Markov Algorithms

We present the Viterbi algorithm in Figure 8 and the Baum-Welch algorithm in Figure 9. The Viterbi algorithm [52] takes as input a sequence of observed states $\mathbf{o} = (o_1, \cdots, o_t)$ and a hidden Markov model $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$. It outputs the most likely sequence of states of the hidden Markov chain that produced $\mathbf{o}$ as well as an error score s. Note that the value s is only calculated when the Viterbi algorithm is run as a subroutine of the Decoder-Smpl attack, but not for Decoder. The Baum-Welch algorithm [53] takes as input an observed sequence $\mathbf{o} = (o_1, \cdots, o_t)$ and outputs a local maximum of the hidden Markov model parameters $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$. Note that the algorithm has the convergence level as well as the number of states hardcoded. The convergence level parameter can be tuned to obtain better accuracy.

```
• Viterbi(HMM, o):
  (1) parse $\mathbf{o} = (o_1, \cdots, o_t)$ and $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$;
  (2) instantiate two zero-matrices $\delta$ and $\phi$ of size $t \times n$ where $n$ is the number of states in T;
  (3) for all $i \in [n]$, compute
        $\delta_{1,i} = \mu_i \cdot \mathsf{O}_{i,o_1}$   and   $\phi_{1,i} = 0$;
  (4) for all $j \in \{2, \cdots, t\}$ and $i \in [n]$, compute
        $\delta_{j,i} = \max_{k \in [n]} \left(\mathsf{T}_{k,i} \cdot \delta_{j-1,k}\right) \cdot \mathsf{O}_{i,o_j}$   and   $\phi_{1,i} = \text{argmax}_{k \in [n]} \left(\mathsf{T}_{k,i} \cdot \delta_{j-1,k}\right)$;
  (5) set
        $r_t := \text{argmax}_{k \in [n]} (\delta_{t,k})$;
  (6) compute the error score
        $s := \max_{k \in [n]} (\delta_{t,k})$;
  (7) for $j \in \{t-1, \cdots, 1\}$, set
        $r_j := \phi_{j+1, r_{j+1}}$;
  (8) output $\mathbf{r} = (r_1, \cdots, r_t)$ and (optionally) s.
```

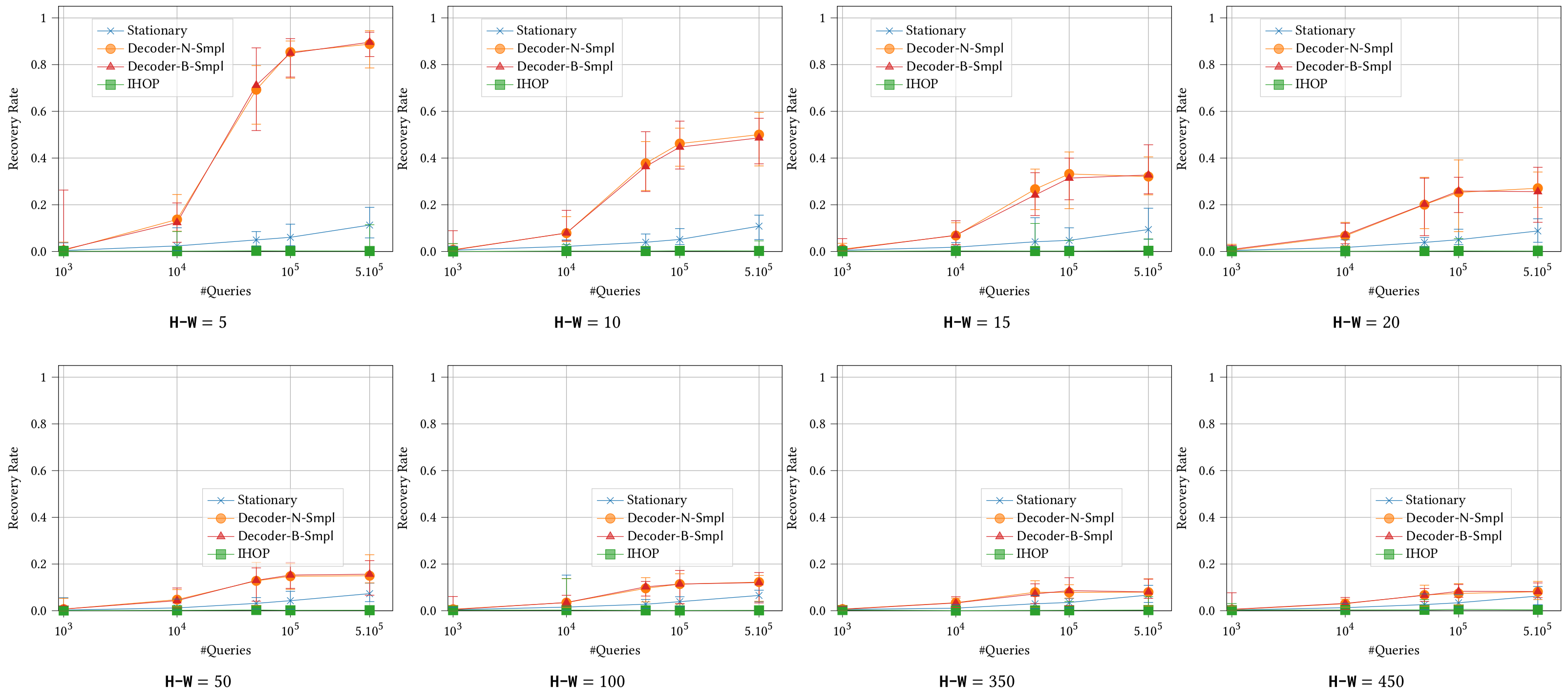
Figure 8: The Viterbi algorithm [52].

```
• Baum-Welch(o):
  (1) parse $\mathbf{o} = (o_1, \cdots, o_t)$ and initialize count := 0;
  (2) instantiate a hidden Markov model $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$ such that
      (a) populate $\mathsf{T} = (\mathsf{T}_{i,j})_{i,j \in [n]}$ such that,
          (i) compute $\mathsf{T}_{i,j} \xleftarrow{\$} \{0,1\}^k$;
          (ii) set $\mathsf{T}_{i,j} := \mathsf{T}_{i,j} / \lambda_i$ where $\lambda_i = \sum_{j=1}^{n} \mathsf{T}_{i,j}$;
      (b) populate $\mathsf{O} = (\mathsf{O}_{i,j})_{i,j \in [n]}$ such that,
          (i) compute $\mathsf{O}_{i,j} \xleftarrow{\$} \{0,1\}^k$;
          (ii) set $\mathsf{O}_{i,j} := \mathsf{O}_{i,j} / \lambda_i$ where $\lambda_i = \sum_{j=1}^{n} \mathsf{O}_{i,j}$;
      (c) set $\mu := (1/n, \cdots, 1/n)$;
  (3) while count $\leq$ level,
      (a) for all $i \in [n]$ and $t' \leq t$, compute
            $\alpha_{i,1} = \mu_i \cdot \mathsf{O}_{1,i}$   and   $\alpha_{i,t'+1} = \mathsf{O}_{o_{t'+1},i} \cdot \sum_{j=1}^{n} \left(\alpha_{j,t'} \cdot \mathsf{T}_{j,i}\right)$
      (b) for all $i \in [n]$ and $t' \leq t$, compute
            $\beta_{i,t} = 1$   and   $\beta_{i,t'} = \sum_{j=1}^{n} \beta_{j,t'+1} \cdot \mathsf{T}_{i,j} \cdot \mathsf{O}_{(t'+1),j}$
      (c) calculate the following for all $i, j \in [n]$ and $t' \leq t$,
            $\gamma_{i,t'} = \left(\alpha_{i,t'} \cdot \beta_{i,t'}\right) \cdot \left(\sum_{j=1}^{n} \alpha_{j,t'} \cdot \beta_{j,t'}\right)^{-1}$
          and,
            $\xi_{i,j,t'} = \left(\alpha_{i,t'} \cdot \mathsf{T}_{i,j} \cdot \beta_{j,t'+1} \cdot \mathsf{O}_{o_{t'+1},j}\right) \cdot \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{i,t'} \cdot \mathsf{T}_{i,j} \cdot \beta_{j,t'+1} \cdot \mathsf{O}_{o_{t'+1},j}\right)^{-1}$
      (d) update the parameters for HMM for all $i, j \in [n]$ ,
        $\mu_i = \gamma_{i,1}$   and   $\mathsf{T}_{i,j} = \left(\sum_{t'=1}^{t-1} \xi_{i,j,t'}\right) \cdot \left(\sum_{t'=1}^{t-1} \gamma_{i,t'}\right)^{-1}$   and   $\mathsf{O}_{i,j} = \left(\sum_{t'=1}^{t} \mathbf{1}_{(o_{t'}=j)} \cdot \gamma_{i,t'}\right) \cdot \left(\sum_{t'=1}^{t} \gamma_{i,t'}\right)^{-1}$
      (e) increment count;
  (4) output $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$.
```

Figure 9: The Baum-Welch algorithm [53].

B Evaluation Results

We present in Figure 10 the evaluation result when we vary the Hamming weight and fix the number of states to 500. We present in Figure 11 the evaluation results of our known-distribution attacks when the user queries are sampled from the Binomial-Zipf, Uniform and Zipf distributions.



H-W = 5

H-W = 10

H-W = 15

H-W = 20

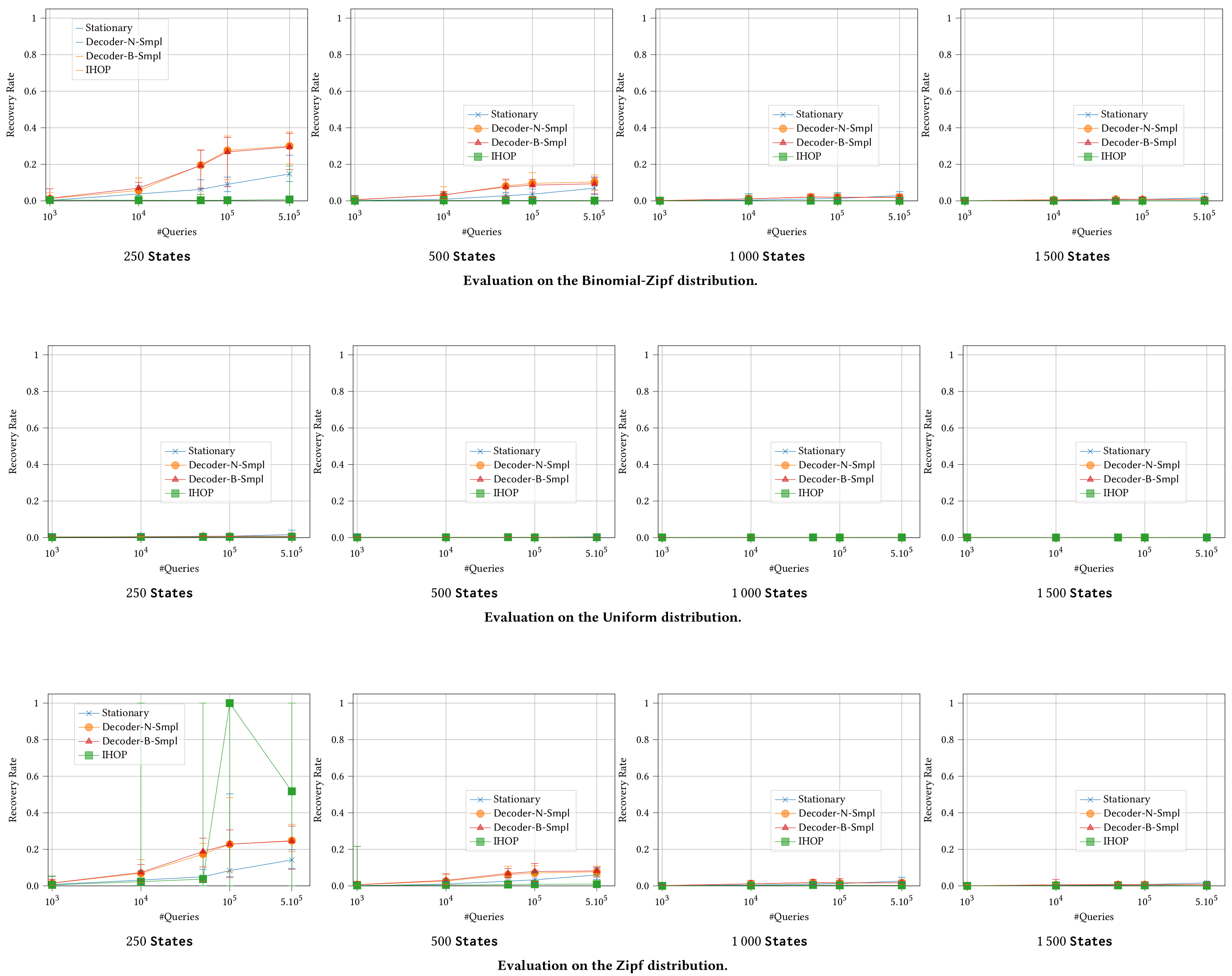
H-W = 50

H-W = 100

H-W = 350

H-W = 450

Figure 10: Our new attacks and the IHOP attack [45] evaluated on the Zipf-Zipf distribution setting. Evaluations are done using a *variable* Hamming weight and a *fixed* number of states = 500**.**



Evaluation on the Binomial-Zipf distribution.

Evaluation on the Uniform distribution.

Evaluation on the Zipf distribution.

Figure 11: Our new attacks and the IHOP attack [45] evaluated in the known-distribution setting. The results represent the average recovery rate over 30 runs.