University Mohammed VI Polytechnic

University Mohammed VI Polytechnic
Center for Doctoral Studies

Thesis submitted for the Degree of
Doctor of Philosophy
Science, Engineering and Technology

Specialty

Applied Crytography

Presented by

Abdelkarim KATI

Supervised by

Dr-Ing Tarik MOATAZ

In the Department

College Of Computing

Cryptanalysis of Encrypted Search Algorithms:

From Theory To Practice

Defended on XX/02/2024, with the presence of the following assessment committee:

-	Pr. Mustapha Hedabou,	UM6P	President
-	Pr. David Cash,	University of Chicago	Referee
-	Pr. Melek Önen,	Eurecom	Referee
-	Dr. Erik-Oliver Blass,	Airbus	Referee
-	Pr. Guevara Noubir,	Northeastern University	Examinator
-	Dr-Ing. Brice Minaud,	École Normale Supérieure	Guest Examinator
-	Dr-Ing. Tarik Moataz,	MongoDB	Thesis Scientific- Director
-	Pr. Youssef Iraqi,	UM6P	Thesis Administrative- Director

2024 – 2025

College of Computing

Cryptanalysis of Encrypted Search Algorithms: From Theory To Practice

College Of Computing
at Mohammed-VI Polytechnic University

Doctoral Thesis

submitted in fulfillment of the requirements for the degree of
Doctor of Engineering (Dr.-Ing.)

by

Abdelkarim Kati, M.Sc.

Advisor: Dr.-Ing. Tarik Moataz

Date of submission: 01.12.2023
Date of defense: 12.02.2024

Affidavit

I, the undersigned, hereby declare that the submitted Doctoral Thesis is my own work. I have only used the sources indicated and have not made unauthorised use of services of a third party. Where the work of others has been quoted or reproduced, the source is always given.

I further declare that the dissertation presented here has not been submitted in the same or similar form to any other institution for the purpose of obtaining an academic degree.

Ben-Guerir, 01.12.2023

Abdelkarim Kati, M.Sc.

Abstract

In recent years, there has been a consistent increase in the frequency of privacy violations associated with the emergence of new technological solutions that involves the collection, complex processing and storage of data. We believe that this privacy violation trend is due to a multitude of factors such as data leaks and the lack of access control mechanisms. Within this context, the European Union Agency for Cybersecurity (ENISA) and UK Information Commissioner's Office (ICO) have recently released a report on Data Protection Engineering and a guidance on privacy-enhancing technologies ("PETs") respectively, where regulators recommend key methods in designing software and hardware solutions, processing operations and achieving privacy or data protection functionalities that respect the rights and freedoms of individuals or groups in relation to their personal data under legal requirements of Articles 25, 5, and 32 of the GDPR. In recent years, a lot of works has been done to design cryptographic protocols that allow the processing of encrypted data. In particular, the servers managing the data should not have access to the data in the clear. In this thesis, we focus specifically on the problem of searching over encrypted data that was explicitly introduced by the works of Song, Wagner and Perrig in 2000 [SWP00].

We believe that a broad deployment of Encrypted Search Algorithms (ESAs) is a highly needed step towards practical data security, this insures the privacy of individuals and reduces the risks posed by existing deployable solutions. *ESAs* are constructions that allow the users to securely search over encrypted outsourced data where the server never decrypts the data, this capability comes with complex tradeoffs between efficiency, expressiveness, and security. After more than two decades of research advances, *ESAs* constructions are becoming a practical reality, with existing products such as MongoDBs queryable encryption (QE) [Mon23a; Mon23b], a commercial product based on Structured Encryption (STE) that is able to run encrypted queries on encrypted data. Amazon has also recently released AWS Database Encryption SDK[Ama23], which provides record-level encryption on top of DynamoDB using some form of property preserving encryption.

Concerning the security aspect, *ESAs* constructions are characterized by leakage profiles modeling a set of leakage patterns , e.g., the query equality (or search) pattern, which reveals whether and when an issued query is repeated. As a means to assess how exploitable each profile can be under which assumptions, *leakage attacks* have been proposed, where the attacks leverage a leakage pattern coupled with some auxiliary information to recover the content of private data and or queries. Consequently, putting into question the security guarantees of many encrypted search systems. However, the practical evaluation of leakage attacks has, so far, been highly restricted and heavily reliant on assumptions about the users' behavior and data. Therefore, the leakage attacks are either seen as demonstrating *the* insecurity of an encrypted search (with a specific leakage profile) because there exists *a* successful attack instance, or they are regarded as only of theoretical interest because the few successful instances require assumptions that can be seen as unrealistic. It is not clear in which cases leakage attacks do not work, as well as which specific properties of the system

influence their success, ergo, their actual impact on the security of encrypted search systems remains open.

Concretely, based on existing limited evaluations, one can observe that the attacks' performance is highly depend on users' behavior, e.g., query distribution or query selectivity. Additionally, existing evaluations over artificial behavior have been confined to a handful of datasets, while none of the prior works considered real-world user behavior. It remains an open question how well attacks fare on completely different instances of real-world datasets. Our goal in this thesis was to improve the community general understanding of leakage by leveraging real-world query data in order to draw definite conclusions about the attacks practical efficacy, since we could not simply ascertain whether a given leakage profile is exploitable in practice based solely on existing works. This is due to several reasons, including but not limited to the lack of open-source implementations (which are needed to reproduce results), conducting empirical evaluations on restricted datasets, and in some cases relying on relatively strong assumptions which can significantly affect the accuracy. Furthermore, prior works often leave out information about the parameters chosen during pre-processing phase and only present a restricted number of results. This actively hides the effects of variables on the recovery rates, and omits instances where the attacks do not work well. Therefore, a comprehensive benchmarking study of the effects of leakage attacks is gravely needed in order to bridge the existing gap of identifying the cases where the attacks are successful, since their performance depends on the users' behavior. We believe that by broaching this subject we can lessen this gap, through an empirical evaluation conducted on a range of representative real-world data under the same conditions.

In this thesis, we move towards addressing these limitations. First, we design and implement LEAKER, an open-source framework that evaluates the major leakage attacks against any dataset and that we hope will serve the community as a common way to evaluate leakage attacks. We identify new real-world datasets that capture different use cases for ESAs and, for the first time, include real-world user queries. Finally, we use LEAKER to systematically evaluate known attacks on our datasets, uncovering sometimes unexpected properties that increase or diminish accuracy. Our evaluation shows that some attacks work better on real-world data than previously thought and that others perform worse. This part of the thesis has lead to the following publication:

[KKM+22] S. KAMARA, A. KATI, T. MOATAZ, T. SCHNEIDER, A. TREIBER, M. YONLI. **"SoK: Cryptanalysis of Encrypted Search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data"**. In: *7th IEEE European Symposium on Security and Privacy (EuroS&P'22)*. Full version: https://ia.cr/2021/1035. Code: https://encrypto.de/code/LEAKER. IEEE, 2022, pp. 90–108.

Additionally, 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. This part of the thesis has lead to the following publication:

[KKM+24] S. KAMARA, A. KATI, T. MOATAZ, J. DEMARIA, A. PARK, A. TREIBER. **"MAPLE: MArkov Process Leakage attacks on Encrypted Search"**. In: *The 24th annual Privacy Enhancing Technologies Symposium (PETS)*. Full version: https://ia.cr/2023/810. Code: https://github.com/anonymous-repo-submission/artifact. PETS, 2024, TBD.

Granted, many of the questions we have identified remain open as future challenges to be tackled, but overall the contributions we have made throughout this thesis have nonetheless improved our understanding of the impact and exploitability of leakage in a practical real-world settings.

Résumé

Ces dernières années, on observe une augmentation constante de la fréquence des violations de la vie privée liées à l'émergence de nouvelles solutions technologiques impliquant la collecte, le traitement complexe et le stockage des données. Nous croyons que cette tendance aux violations de la confidentialité est due à une multitude de facteurs tels que les divulgation de données et le manque de mécanismes de contrôle d'accès. Dans ce contexte, l'Agence de l'Union européenne pour la cybersécurité (ENISA) et l'Information Commissioner's Office (ICO) du Royaume-Uni ont récemment publié un rapport sur l'ingénierie de la protection des données et un guide sur les technologies qui améliorant la confidentialité ("PETs"), respectivement. Les régulateurs recommandent des méthodes clés pour concevoir des solutions logicielles et matérielles, pour traiter des opérations et atteindre des fonctionnalités de protection des données privilegier qui respectant les droits et libertés des individus et des groupes conformément aux exigences légales des articles 25, 5 et 32 du RGPD. Au cours des dernières années, de nombreux travaux ont été réalisés pour concevoir des protocoles cryptographiques qui permettant le traitement de données chiffrées. En particulier, les serveurs gérant les données ne devraient pas avoir accès aux données en clair. Dans cette thèse, nous nous concentrons spécifiquement sur le problème de la recherche sur des données chiffrées introduit explicitement par les travaux de Song, Wagner et Perrig en 2000 [SWP00].

Nous pensons qu'un déploiement étendu des algorithmes de recherche chiffrée (ESAs) est une étape fortement nécessaire vers la sécurité pratique des données, assurant la vie privée des individus et réduisant les risques posés par les solutions déployables existantes. Les ESAs sont des constructions permettant aux utilisateurs de rechercher de manière sécurisée des données chiffrées externalisées, où le serveur ne déchiffre jamais les données. Cette capacité s'accompagne de compromis complexes entre l'efficacité, l'expressivité et la sécurité. Après plus de deux décennies d'avancées dans la recherche, les constructions des ESAs deviennent une réalité pratique, avec des produits existants tels que le chiffrement interrogeable (QE) de MongoDB [Mon23a; Mon23b], un produit commercial basé sur le chiffrement structuré (STE) capable d'exécuter des requêtes chiffrées sur des données chiffrées. Amazon a également récemment publié AWS Database Encryption SDK [Ama23], qui offre un chiffrement au niveau de l'enregistrement sur DynamoDB en utilisant une forme de chiffrement préservant les propriétés.

En ce qui concerne l'aspect sécurité, les constructions des ESAs sont caractérisées par des profils de fuite modélisant un ensemble de motifs de fuite, par exemple, le motif d'égalité de requête (ou de recherche), qui révèle si et quand une requête émise est répétée. Pour évaluer dans quelles conditions chaque profil peut être exploité, des *attaques de fuite* ont été proposées, où les attaques utilisent un motif de fuite couplé à des informations auxiliaires pour récupérer le contenu de données privées et/ou de requêtes, remettant ainsi en question les garanties de sécurité de nombreux systèmes de recherche chiffrée. Cependant, l'évaluation pratique des attaques de fuite a, jusqu'à présent, été très limitée et fortement dépendante d'hypothèses sur le comportement et les données des utilisateurs. Par conséquent, les attaques de fuite sont soit considérées comme démontrant l'insécurité d'une recherche chiffrée (avec

un profil de fuite spécifique) parce qu'il existe une instance d'attaque réussie, soit elles sont considérées comme relevant seulement d'un intérêt théorique car les rares instances réussies nécessitent des hypothèses qui peuvent être perçues comme irréalistes. Il n'est pas clair dans quels cas les attaques de fuite ne fonctionnent pas, ni quelles propriétés spécifiques du système influent sur leur succès, par conséquent, leur impact réel sur la sécurité des systèmes de recherche chiffrée reste ouvert.

Concrètement, à partir des évaluations limitées existantes, on peut observer que la performance des attaques dépend fortement du comportement des utilisateurs, par exemple, la distribution des requêtes ou la sélectivité des requêtes. De plus, les évaluations existantes sur un comportement artificiel se limitent à quelques ensembles de données, tandis que aucun des travaux antérieurs n'a pris en compte le comportement réel des utilisateurs. Il reste une question ouverte de savoir comment les attaques fonctionnent sur des instances complètement différentes de jeux de données du monde réel. Notre objectif dans cette thèse était d'améliorer la compréhension générale de la communauté sur la fuite en tirant parti des données de requêtes du monde réel afin de tirer des conclusions définitives sur l'efficacité pratique des attaques, étant donné que nous ne pouvions pas simplement déterminer si un profil de fuite donné est exploitable en pratique uniquement sur la base des travaux existants. Cela est dû à plusieurs raisons, notamment le manque d'implémentations open source (nécessaires pour reproduire les résultats), la réalisation d'évaluations empiriques sur des ensembles de données restreints, et dans certains cas, une dépendance à des hypothèses relativement fortes qui peuvent affecter significativement la précision. En outre, les travaux antérieurs laissent souvent de côté des informations sur les paramètres choisis lors de la phase de prétraitement et ne présentent qu'un nombre restreint de résultats. Cela cache activement les effets des variables sur les taux de récupération et omet les cas où les attaques ne fonctionnent pas bien. Par conséquent, une étude complète de référence sur les effets des attaques de fuite est gravement nécessaire pour combler le fossé existant afin d'identifier les cas où les attaques réussissent, étant donné que leur performance dépend du comportement des utilisateurs. Nous croyons qu'en abordant ce sujet, nous pouvons réduire ce fossé, grâce à une évaluation empirique sur une gamme de données du monde réel représentatives dans les mêmes conditions.

Dans cette thèse, nous nous efforçons de résoudre ces limitations. Tout d'abord, nous concevons et implémentons LEAKER, un framework open source qui évalue les principales attaques de fuite contre n'importe quel ensemble de données et que nous espérons servira à la communauté comme une manière commune d'évaluer les attaques de fuite. Nous identifions de nouveaux ensembles de données du monde réel qui capturent différents cas d'utilisation pour les ESAs et, pour la première fois, incluent des requêtes d'utilisateurs du monde réel. Enfin, nous utilisons LEAKER pour évaluer systématiquement les attaques connues sur nos ensembles de données, découvrant parfois des propriétés inattendues qui augmentent ou diminuent la précision. Notre évaluation montre que certaines attaques fonctionnent mieux sur des données du monde réel que précédemment pensé et que d'autres sont moins performantes. Cette partie de la thèse a conduit à la publication suivante :

[KKM+22] S. KAMARA, A. KATI, T. MOATAZ, T. SCHNEIDER, A. TREIBER, M. YONLI. **"SoK: Cryptanalysis of Encrypted Search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data"**. In: *7th IEEE European Symposium on Security and Privacy (EuroS&P'22)*. Full version: https://ia.cr/2021/1035. Code: https://encrypto.de/code/LEAKER. IEEE, 2022, pp. 90–108.

De plus, nous poursuivons l'étude de la fuite d'égalité de requête sur des requêtes dépendantes et présentons deux nouvelles attaques dans ce contexte qui peuvent fonctionner soit comme des attaques de distribution connue, soit comme des attaques d'échantillonnage connu. Elles modélisent les distributions de requêtes comme des processus de Markov et tirent parti des idées et des techniques des processus stochastiques et de l'apprentissage machine. Nous implémentons nos attaques et les évaluons sur des journaux de requêtes du monde réel. Nos expériences montrent qu'elles surpassent l'état de l'art dans la plupart des configurations mais ont aussi des limitations dans des situations pratiques. Cette partie de la thèse a conduit à la publication suivante :

[KKM+24] S. KAMARA, A. KATI, T. MOATAZ, J. DEMARIA, A. PARK, A. TREIBER. **"MAPLE: MArkov Process Leakage attacks on Encrypted Search"**. In: *The 24th annual Privacy Enhancing Technologies Symposium (PETS)*. Full version: https://ia.cr/2023/810. Code: https://github.com/anonymous-repo-submission/artifact. PETS, 2024, TBD.

Certes, bon nombre des questions que nous avons identifiées restent ouvertes en tant que défis futurs à relever, mais dans l'ensemble, les contributions que nous avons apportées tout au long de cette thèse ont néanmoins amélioré notre compréhension de l'impact et de l'exploitabilité des fuites dans un contexte pratique du monde réel.

Acknowledgments

Contents

Abstract II

Résumé V

Acknowledgments VIII

Contents IX

1 Introduction 1
- 1.1 Encrypted Search Algorithms (ESAs) . . . 3
- 1.2 Encrypted Search Algorithms (ESAs) Cryptanalysis . . . 3
- 1.3 Limitations . . . 5
- 1.4 Thesis Outline & Contributions . . . 6

2 Related Work 8

3 Preliminaries 11

4 LEAKER 14
- 4.1 Existing Leakage Attacks . . . 14
- 4.2 The LEAKER Framework . . . 21
- 4.3 Data Collections & Query Logs . . . 24
- 4.4 Empirical Evaluation . . . 28

5 MAPLE 46
- 5.1 Stochastic Processes . . . 47
- 5.2 Statistical Inference Attacks . . . 49
- 5.3 From Distributions to Samples . . . 58
- 5.4 Empirical Evaluation . . . 60

6 Conclusion and Future Work 71
- 6.1 Summary . . . 71
- 6.2 Discussion & Guidlines . . . 73
- 6.3 Future directions . . . 75

7 Appendices 76
- 7.1 LEAKER Additional Empirical Evaluations . . . 76

7.2 LEAKER Additional Data 79
7.3 LEAKER Code Snippet 87
7.4 Markov Models 88
7.5 Markov Algorithms 89
7.6 Markov Additional Empirical Evaluations 90

Bibliography **94**

Lists **104**

1 Introduction

As we look back on the past two decades, the data landscape has been expanding at an unprecedented rate, making the digitization of information nearly ubiquitous. Thus, individuals and enterprises have been projected to create, capture, replicate, and consume an estimated 180 zettabytes of data annually with a compound annual growth rate (CAGR) of 21.2% by 2026 [IDC23], Where almost all this data will be stored in the cloud.

Public cloud service providers play a pivotal role by providing an infrastructure that grants businesses and individuals access to computational power and storage space under a flexible pay-as-you-go model, thereby circumventing the substantial costs associated with establishing and maintaining proprietary data centers, which includes hardware, construction, air conditioning, and security expenditures. This renders cloud computing an economically viable solution for both extensive bulk data processing and storage needs, thus offering faster innovation, flexible resources and capitalizes on economies of scale.

As organizations started moving their workloads off-premise, a massive shift towards remote work adoption changed the nature of how organizations work. This mass cloud adoption has been notably expedited by the global pandemic, which compelled companies to provide employees with access to business systems from anywhere. Yet, it left them grapple with the challenges of managing an ever-expanding pool of security vulnerabilities and the escalating complexity inherent in multi-cloud environments. The scarcity of proficient cybersecurity personnel exacerbates this predicament.

Research by the UK cyber resilience and data breaches survey relating to digital data in 2023 [Nat23] has shown that cyber security breaches and attacks remain a common threat, where 32% of businesses and 24% of charities overall identified any breaches or attacks from the last 12 months (a decrease from 39% of businesses and 30% of charities the previous year. The drop is mainly driven by smaller organisations). The number is much higher for medium and large businesses, 59% and 69% respectively, also and high-income charities with £500,000 or more in annual income, 56%, suffered at least one data breach. It is essential to note that the risk landscape in the cloud differs from that in an on-premises environment. As the reliance on cloud services intensifies, it becomes imperative to address the unique security considerations posed by cloud storage, acknowledging the distinct challenges that arise in this dynamic and interconnected digital ecosystem.

Meanwhile, the constant occurrences of data breaches have demonstrated the clear need for data encryption which serves as a crucial measure for safeguarding the confidentiality of data being collected and managed, especially customer sensitive information like electronic health

record or financial records. This protective strategy addresses both data in transit and data at rest across various devices and user interactions. Hence, complying with data privacy and protection regulations, including standards like FIPS (Federal Information Processing Standards) and HIPPA (Health Insurance Portability and Accountability Act of 1996) mandating organizations to encrypt sensitive customer data to comply with legal requirements.

The adoption of cloud encryption not only fortifies security measures but also ensures compliance with regulations, maintains data integrity, and mitigates the risks associated with data breaches. While encryption using standard cryptographic primitives provides data security in transit and or at rest, which would prevent information leakage in the event of a compromise, this trivial solution neither scale nor ensure the security of data at its most valuable state: namely, as it is being used. The reason is that standard cryptographic primitives hinders the users' computational and search capabilities over encrypted data. Compiling the user to either downloading all the content to its locally (trusted) environment, which defies the purpose of outsourcing the data, or alternatively entrusting the third party service provider with the key to decrypt all data. However, this approach, while used by a plethora of cloud providers such as Amazon S3 [Ama23], discloses the content of both data and queries. Therefore, it is important to ensure the confidentiality of both of the data and queries when outsourcing sensitive data to the cloud.

With the advent of privacy-sensitive applications, the ability to cryptographically separates the roles of providing, administering, and accessing data has become of the utmost importance, in order to ensure the confidentiality of both of outsourced data and queries. Therefore, technical means allowing digital services while preventing the privacy erosion are often referred to as Privacy-Enhancing Technologies (PETs). PETs surveyed in [HZNF15], are a very wide class of techniques to computing over encrypted data, including but not limited to anonymous communication, statistical methods such as differential privacy [DR+14], obfuscation, anonymization, private federated learning, homomorphic encryption and multi-party computation.

The concept of Encrypted Search (ES) realizes various privacy and security goals and provides a promising solution to this problem by using different cryptographic primitives allowing the outsourcing of encrypted data whilst maintaining search capabilities , i.e., the authorised client process of encrypted data without revealing the data in the clear to the server. However, most existing constructions achieve practicality at the cost of leaking some information to the server. The information varies depending on the expressiveness, the scheme, the threat model; but tends to be statistical in nature. As an instance a scheme can leak if and when the same *encrypted* query has been issued by the client. In order to find out what can be learned from leaked information we provide, in this thesis, a *detailed empirical cryptanalysis of Encrypted Search Algorithms (ESAs)* in the semi-honest security model, also known as honest-but-curious adversarial model, where the untrusted party (server) may attempt to learn additional information from the exchanged data to gain insights into the confidential information.

1.1 Encrypted Search Algorithms (ESAs)

Encrypted search algorithms (ESAs) enable the users to privately search over their outsourced encrypted data. ESAs have received a lot of attention due to applications to cloud storage and database security (see the survey by Fuller et al. [FVY+17] for an overview). At a high level, ESAs consist of two algorithms: a $setup$ algorithm that encrypts a data collection (or a database) and encrypts it with a returned secret key sk. This encrypted collection will be then outsourced to a remote server. And a $search/query$ algorithm that uses the secret key sk and a users query q to retrieve all the matching entries from the server $q(\mathcal{C}) = \{e \in \mathcal{C} : q(e)\}$. Dynamic ESAs also have an $update$ algorithm to add or remove data.

Provably secure ESAs can be constructed from a various, sometimes intersecting cryptographic primitives: property-preserving encryption (PPE) [AKSX04; BBO07], fully-homomorphic encryption (FHE) [Gen09], searchable/structured symmetric encryption (SSE/STE) [SWP00; Goh03; CM05; CGKO06; CK10], functional encryption [BSW11], private set intersection (PSI) [HEK12; PSTY19], private information retrieval (PIR) [CGKS95], or oblivious RAM ORAM [GO96] (see below). A standard structured encryption (STE) can also be viewed as an instance of structured encryption (STE) for keyword search, though STE can encompass other query types as well as other structures than search structures. In this work we focus on ESAs realized via SSE/STE techniques that we refer to as structured ESAs.

Efficiency. ESA constructions achieve varying 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. When evaluating the efficiency of an ESA, we usually focus on the query or search time; that is the time needed to search over the encrypted data. ESAs based on (FHE) or secure multi-party computation (SMPC) are leakage free, but require at least linear time search complexity in the size of the data so they are usually considered impractical. Search based on functional encryption [BSW11] also has linear complexity. For practical purposes, one needs sub-linear ESAs which can be achieved with (ORAM) in $\mathsf{opt} \cdot \log^{O(1)} n$ time. On the other hand, structured and property-preserving ESAs, based Searchable Symmetric Encryption (SSE) or Structured Encryption (STE) respectively, which can achieve opt search time, where opt is the optimal time to search and n is the number of items in the data collection.

1.2 ESAs 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 [**KKM+22**] for a survey of ESAs 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 [**KKM+22**] recently showed that using real-world query data in the form of *query logs* can often lead to very different accuracy results.

Adversarial models and leakage. ESAs can be analyzed in different adversarial models. The most common are the *snapshot* and *persistent* models. A snapshot adversary receives a copy of the encrypted data at various times and tries to recover information about the data collection. A persistent adversary receives the encrypted data and a transcript of the query operations and tries to recover information about the data collection and the queries. The information an adversary can recover about the data or queries is referred to as *leakage* and, ideally, one would prefer a zero-leakage solution, which can be achieved in several ways. In the snapshot model, it is possible to design very efficient zero-leakage ESAs using structured encryption [AKM19], whereas in the persistent model zero-leakage ESAs can be designed using FHE at the cost of linear-time queries. In the persistent model oblivious, structured, and property-preserving ESAs all leak some information; though recent work has shown how to suppress some of this leakage for certain encrypted data structures [KMO18; KM19; PPYY19; APP+21; GKM21]. For instance, prominent leakage profiles include the *response identity* (or access) pattern, which reveals the response to a query, or the *query equality* (or search) pattern, which reveals whether and when a query repeats.

Leakage attacks. Because sub-linear solutions leak information, cryptanalysis plays an important role in the area of encrypted search. By designing *leakage attacks* one can try to ascertain whether a leakage profile is exploitable. Starting with the work of Islam et al. [IKK12], leakage attacks were first designed against structured ESAs in the persistent model. Later, Naveed et al. [NKW15] designed attacks against PPE in the snapshot model and Kellaris et al. [KKNO16] showed attacks against oblivious ESAs in the persistent model.

These works were improved by a series of papers [CGPR15; GLMP18; LMP18; GLMP19; GJW19; BKM20; KPT20; KPT21; OK21; RPH21]. While the attacks improve our understanding of leakage, it can sometimes be difficult to draw definite conclusions about their performance in practical settings. This is due to several limitations: (1) none of the implementations are open-source (with the exception of [RPH21]) which makes it burdensome to reproduce results, especially in new empirical settings; (2) most of the prior evaluations are based on a few and often small datasets without real query logs; (3) some attacks and/or their evaluations are based on assumptions which may or may not be realistic, depending on the application scenario. We stress that some of these limitations are due to the fact that obtaining real-world query logs is very challenging. In fact, this has been recognized as an important

impediment to the evaluation of attacks for some time [GJW19; RPH21]. A consequence of this is that prior evaluations vary greatly in what assumptions they make about queries, e.g., some evaluations of keyword attacks use the most frequent keywords [IKK12; CGPR15] while others use the least frequent keywords [BKM20; RPH21], a choice that can significantly affect results.

1.3 Limitations

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 [OK22]. 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 [OK22] 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.

Standardization. While the implications of leakage attacks remain ambiguous, efforts to deploy ESAs are continuing to move forward. For example, NIST's recent standardization intentions of privacy-preserving technologies include, among others, ESAs and call for reference material and security analyses [NIS21]. Thus, a common attack evaluation framework and more realistic evaluations are paramount to enable a widespread, secure use of ESAs and their standardization.

Vulnerability vs. risk. The above problem stems from a lack of data and knowledge about how clients use search algorithms. As a result, attacks are regarded as demonstrating a potential ESA *vulnerability*, but it is unsurprising that research is in disagreement over the *risk*, i.e., the practical impact of cryptanalysis via leakage attacks. This can be observed by the way new works portray risk: ESA constructions typically argue low risk, stating that attacks require assumptions deemed unrealistic, whereas attack or leakage mitigation papers present *an* attack *instance* as a considerable violation of privacy guarantees.

1.4 Thesis Outline & Contributions

Contributions. This thesis has two main contributions.

The first goal is to broaden the scope of ESA cryptanalysis [**KKM+22**]. Since how well leakage attacks fare on completely different instances of real-world datasets remained an open question, an empirical evaluation of the effects of leakage on a range of representative real-world data under the same conditions was necessary to identify the use cases in which the attacks are successful.

The second goal is to continue the study of query equality leakage in the dependent setting as initiated by [OK22] and we focus on STE/SSE-based ESAs [**KKM+24**]. This is an important leakage pattern to study because it is very common; i.e., most practical constructions reveal it [SWP00; Goh03; CM05; CGKO06; BBO07; CK10; KPR12; CJJ+13; KP13; CJJ+14; Bos16].

We accomplish these goals with the following contributions:

1. In Chapter 4, we designed and implemented an open-source Python framework called LEAKER that evaluates the major leakage attacks against keyword and range (oblivious or structured) ESAs on arbitrary datasets. It is intended as an easy-to-use reference tool for research on leakage attacks and mitigations. With LEAKER we enable and invite the community to contribute additional attacks and evaluations to continually advance our understanding of leakage. We also identify a wide set of real-world datasets that capture different realistic use cases for ESAs and, *for the first time, include real user queries* (query logs), which has long been a well-known challenge in the field. For keyword search, this includes search engine and genetic data. For range search, this includes scientific, medical, human resources, sales, and insurance data. The datasets we consider cover significantly more settings than those used in previous work and can serve the community for future benchmarks. Finally, we use LEAKER to systematically evaluate attacks on oblivious and structured ESAs on real-world data. Our analysis is an important step towards understanding the practicality of many well-known leakage attacks and provides some new insights which were not obtained by previous evaluations. For example, we find that the BKM attacks[1] of Blackstone et al. [BKM20] can achieve higher recovery rates than previously reported when evaluated with real query logs. On the other hand, the recovery rates of the IKK attack of Islam et al. [IKK12] and of the COUNT attacks of Cash et al. [CGPR15] were lower on our datasets. Similarly, the recovery rates of many well-known range attacks were lower on our real-world query logs and data collections. However, when using synthetic query distributions based on statistics from our query logs, the recovery rates improved enough to be considered practical. Particularly, we deem keyword ESAs leaking response identifiers or volume patterns at risk, as relevant attacks can uncover significant information with knowledge as little as 5% of the original database even for small-scale, private instances. We find that attacks exploiting total volume information are ineffective for a partial knowledge $< 80\%$. In the case where

[1] Throughout, we denote attacks by first letter of author names.

wide ranges are covered by range queries, we consider attacks harmful since they can uncover significant information in such settings using response identifier leakage. If query equality and the order of the database entries are also leaked, we observe significant real-world risk. However,we would like to note that the order is not a common leakage pattern for structured and oblivious ESAs. Range attacks exploiting response lengths are not successful at all. We discuss the implications of this in Chapter 6.

2. In Chapter 5, we introduce a new framework based on hidden Markov models (HMM) to model client queries and query equality leakage. 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. Moreover, we can instantiate Decoder with multiple variants, resulting in two attacks we call Decoder-N and Decoder-B. We implement our attacks as well as the IHOP attack in the open-source framework LEAKER [**KKM+22**] and conduct a broad evaluation of all three attacks on both real-world and synthetic data. More precisely, we use the AOL [PCT06] and TAIR [ECW+14] 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 4.1 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.

Organization. The reminder of this thesis is organised as follows. In Chapter 2, we review prior and subsequent works on encrypted search and leakage-based attacks in order to provide context to our results. In Chapter 3, we formally define searchable symmetric encryption (SSE) schemes, adversarial models and scenarios. Then we summarize the leakage profiles and auxiliary data collections used throughout our empirical evaluations.In Chapter 4, we present the design, implementation details and the evaluation results for our LEAKER framework given real-world datasets. In Chapter 5, we introduce our statistical inference attacks, extend LEAKER and evaluate them against the state-of-the-art query recovery attacks in the dependent setting. Finally, in Chapter 6, we conclude the thesis, provide some guidance on how to interpret our results and discuss security, limitations and give outlooks for remaining open questions and future research directions.

2 Related Work

In the following, we review prior and subsequent works on encrypted search and leakage attacks in order to provide context to our results.

Encrypted search algorithms (ESAs). Encrypted Search Algorithms (ESAs) refer to any cryptographic primitive/protocol that allows one to execute search algorithms on encrypted data.

The first explicit ESAs construction for exact keyword search was proposed by Song, Wagner and Perrig [SWP00], though previous work by Goldreich and Ostrovsky on ORAM [GO96] could also be used. Boneh et al. [BDOP04] then considered the problem of public-key searchable encryption. Searchable symmetric encryption (SSE) definitions were given by Goh [Goh03] and Chang and Mitzenmacher [CM05] but the notion of adaptive semantic security for SSE was proposed by Curtmola, Garay, Kamara and Ostrovsky [CGKO06]. [CGKO06] 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 [CK10]. STE (referred to as structure-only STE in [CK10]) 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 beyond SSE including various kinds of encrypted databases [KMO18; AAKM20; KMZZ20; KMPQ21; ZKMZ21].

ESAs can be built using a variety of techniques including property-preserving encryption (PPE) [AKSX04; BBO07], oblivious RAMs (ORAM) [GO96], secure multi-party computation (MPC) [Yao82; GMW87], fully-homomorphic encryption (FHE) [Gen09], and functional encryption (FE) [BSW11]. Fuller et al. [FVY+17] provide a survey on ESAs.

In this work [**KKM+22**] we refer to ESAs built from ORAM as oblivious ESAs and to ESAs built from SSE/STE as structured ESAs. Note, however, that the line between these notions is blurry as one can also view ORAM as a (low-leakage) structured encryption scheme for arrays [KMO18].

Oblivious RAM (Oblivious RAM (ORAM)). Oblivious RAM is a cryptographic primitive introduced by Goldreich and Ostrovsky in [GO96], in which the authors designed a shuffling and data re-encrypting scheme to conceal the access patterns of a user interacting with a RAM-based storage system. ORAM ensures that an observer cannot discern which specific memory locations are being accessed. Many schemes were developed, which could roughly be

classified into either: hierarchical/square root ORAM[GO96] or tree-based ORAM[SDS+13], with subsequent works such as multiple-server ORAM [LO13; SS13], or use of some server computational power [WS12; DDF+16; GMP16].

Leakage attacks. To start, we'll give a brief overview and categorize the existing leakage attack. Following that, we will present prior SOKs on leakage cryptanalysis with an emphasis on query recovery attacks. To conclude, we'll delve into more recent works conducted within the field based on statistical inference. Sublinear ESAs 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 a subset of 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. The attack *target* can be query reconstruction, which is usually the case for attacks in the keyword setting, or data reconstruction, which is usually the case for attacks in the range setting. 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. [IKK12] and was a passive query-recovery attack in the persistent model against co-occurrence leakage pattern (see paragraph 3 of Chapter 3) and required sampled-data as auxiliary information; though later evaluations found that, to achieve reasonable recovery rates it needs known-data as auxiliary information [CGPR15]. Additional sampled-data attacks [LZWT14; DHP21; GPP21; OK21; OK22] and known-data attacks [CGPR15; BKM20; NHP+21; RPH21] 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 [KKNO16; GLMP19; KPT21] under various assumptions. A few recent works have proposed theoretical frameworks to quantify leakage in ESAs constructions [WP17; GLMP19; JS19; JPS21; KMPP22]. Very recently, Kornaropoulos et al. [KMPP22] and Kamara and Moataz [KM23] provided more theoretical methods for quantifying leakage in encrypted search. With the exception of [OK22], all the attacks above assumed queries are independent.

File-injection attacks are active attacks that were proposed by Zhang, Papamanthou and Katz [ZKP16] as well as in [BKM20; PWLP20], attacks on k-NN queries by [KPT19; KPT20], and snapshot attacks on PPE-based ESAs were given by Naveed et al. [NKW15]. Leakage attacks in the persistent range setting were first proposed by [KKNO16] using the response identity or response length patterns (see paragraph 3 of Chapter 3) various atatcks improved on the existing results [GLMP18; LMP18; GLMP19; GJW19; MT19; FMA+20; KPT20; KPT21; MFST21]. As countermeasures, techniques to completely suppress different leakage patterns

are considered by [KMO18; KM19; PPYY19; APP+21; GKM21] and techniques to heuristically lower attack efficacy are given by [CLRZ18; PPYY19; GKL+20; SOPK21].

Other SoKs. Fuller et al. [FVY+17] provide a survey of ESAs which also contains an overview of some leakage attacks and Yao et al. [YZGW20] provide a survey of other leakage attacks,largely focusing on property-preserving encryption. While both works largely focus on PPE leakage attacks, our work[**KKM+22**] mainly focuses on leakage attacks against searchable symmetric encryption (SSE), structured encryption (STE), and oblivious RAM (ORAM) in addition to providing a new software framework and datasets to evaluate these attacks.

Query log analysis. Researchers across different fields have tried to better understand users' querying behavior. This is known as *query log analysis*, and is surveyed in [Jan06; JS06; JPL13]. Many works analyze query logs gathered by (proprietary) systems that only the authors had access to, such as libraries [JCMB00], web searches [JS06], blog searches [MD06], people searches [WBK+11], and data portal searches [KKI+17]. Unfortunately, almost none of these works had data we felt was appropriate to evaluate leakage attacks. We also reached out to multiple services for relevant statistics, but none were willing to provide us with even basic information. As no information relevant to leakage attacks was available, we thus identified novel, real-world query data sources (cf. Section 4.3) and performed our own analyses of leakage attacks on them (cf. Section 4.4).

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 $\{1,\ldots,n\}$, with $2^{[n]}$ its power set, and with $|s|_b$ the bit length of a string s. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{q}$ 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 $\mathcal{D}$ 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 data collection $\mathcal{D} = (\mathrm{D}_1,\ldots,\mathrm{D}_n)$ 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 $\{\mathrm{ct}_i\}_{i\in\mathsf{ids}(w)}$ and the server receives $\perp$.

Data collections $\mathcal{C}$.

- *Document collection* $\mathcal{D}$: A data collection over a keyword space $\mathbb{W}$ is a sequence of documents $(\mathrm{D}_1,\ldots,\mathrm{D}_n)$, each of which is a set $D_i \subseteq \mathbb{W}$. Given a keyword $w \in \mathbf{Q}$ from a query space $\mathbf{Q} \subseteq \mathbb{W}$, a keyword search returns the documents that contain w which we denote as $\mathcal{D}(w) = \{D \in \mathcal{D} : w \in D\}$. The function $\mathsf{ids} : \mathbb{W} \to 2^{[n]}$ takes a keyword as input and returns the identifiers of the documents in $\mathcal{D}(w)$. The frequency of a keyword w is the number of entries that contain w.
- *Numerical collection* $\mathbf{N}$: A data collection over the universe $[N]$ is a (multi) sequence of positive integers $(e_1,\ldots,e_n)$, each of which is an integer $e_i \in [N]$. The *density* of a

numerical collection $\mathbf{N}$ is defined as $\delta(\mathbf{N}) = \#\{e \in \mathbf{N}\}/N$; i.e., the number of unique values in $\mathbf{N}$ over the universe size. We will sometimes also consider the histogram of a numerical collection $v(\mathbf{N}) = \big(\#\{e \in \mathbf{N} : e = v\}\big)_{v \in [N]}$. Given a range $r = (a, b)$, where $a, b \in [N]$ and $a \leq b$, a range query returns $\mathbf{N}(r) = \{e \in \mathbf{N} : a \leq e \leq b\}$. We overload the function $\mathsf{ids} : [N] \times [N] \to 2^{[n]}$ which takes a range as input and returns the identifiers of the elements that are within the range $r = (a, b)$. The *width* of a range query $r = (a, b)$ is the value $b - a + 1$.

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 Encrypted Search Algorithms (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. We denote a (*document or numerical*) collection of data entries as $\mathcal{C}$, which is a sequence of length n of data. A query sequence $q_1, \ldots, q_t$, where each q_i is either a keyword or a range query issued by a user. We recall the most commonly exploited leakage patterns of Encrypted Search Algorithms (ESAs) as defined in [BKM20]:

- The *response identity* pattern rid (or access pattern) reveals the identifiers: $\mathsf{rid}(\mathcal{C}, q_1, \ldots, q_t) = \big(\mathsf{ids}(q_1), \ldots, \mathsf{ids}(q_t)\big)$.
- The *query equality* pattern qeq (or search pattern) reveals if and when queries are equal: $\mathsf{qeq}(\mathcal{C}, q_1, \ldots, q_t) = M \in \{0,1\}^{t \times t}$, where $M[i, j] = 1$ iff $q_i = q_j$.
- The *response length* pattern rlen leaks the number of matching entries: $\mathsf{rlen}\big(\mathcal{C}, q_1, \ldots, q_t\big) = \big(|\mathsf{ids}(q_1)|, \ldots, |\mathsf{ids}(q_t)|\big)$.
- The *co-occurrence* pattern co leaks how often keywords co-occur within the same document: $\mathsf{co}\big(\mathcal{D}, w_1, \ldots, w_t\big) = M \in [n]^{t \times t}$, where $M[i, j] = |\mathsf{ids}(w_i) \cap \mathsf{ids}(w_j)|$. This information is implied by rid and implies rlen.
- The *volume* pattern vol leaks the bit length of matching entries: $\mathsf{vol}\big(\mathcal{C}, q_1, \ldots, q_t\big) = \Big(\big(|e|_b\big)_{e \in \mathcal{C}(q_1)}, \ldots, \big(|e|_b\big)_{e \in \mathcal{C}(q_t)}\Big)$.
- The *total volume* pattern tvol leaks the total bit length of matching entries: $\mathsf{tvol}\big(\mathcal{C}, q_1, \ldots, q_t\big) = \Big(\sum_{e \in \mathcal{C}(q_1)} |e|_b, \ldots, \sum_{e \in \mathcal{C}(q_t)} |e|_b\Big)$.
- The *order* pattern order leaks the order of elements $\big(\mathsf{id}(e)\big)_{e \in \mathsf{sorted}(\mathcal{C})}$
- The *rank* pattern rank leaks the number of entries smaller than value v. $\big(\#\{e \in \mathcal{C} : e \leq v\}\big)$

The order and rank patterns require an order relation on the plaintext space and are mostly related to numerical data collections. For a more detailed discussion on leakage patterns we refer the reader to [KMO18].

In Chapter 5 we mainly focus on constructions that have the following leakage profile:

$$\Lambda = \Big(\mathcal{L}_s, \mathcal{L}_q\Big) = \Big(\star, \big(\mathsf{qeq}, \star\big)\Big).$$

where $\star$ refers to any arbitrary collection of leakage patterns. Note that most structured SSE schemes in literature have Λ as their leakage profile [CGKO06; CK10; CJJ+13; CJJ+14; BMO17; GPPJ18; KM19; PPYY19; Bos16]. This is true for all property-preserving encrypted search algorithms as well [BBO07; ARZB11]. This illustrates the importance of studying the query equality pattern. Hence, in (Sections 5.2.1, 5.2.2) we present our qeq-attacks that are applicable to all these constructions.

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. [CGKO06] 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 [CGKO06; CK10; AKM19].

Security definitions. The security of ESAs can be formalized using "leakage-parameterized" definitions following [CGKO06; CK10]. In this framework, a construction is proven secure with respect to a security definition that is parameterized with a specific leakage profile. Leakage-parameterized definitions for persistent adversaries were given in [CGKO06; CK10] and for snapshot adversaries in [AKM19]. [1] We recall these definitions here informally and refer the reader to [CGKO06; CK10; AKM19] for the formal definitions.

Types of attacks. In our works, 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.

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

4 LEAKER

With respect to the cryptanalysis of Encrypted Search Algorithms (ESAs), it is hard to evaluate the effectiveness of leakage attacks, since the majority of attack implementations are close-sourced. Research and practice in encrypted search was hindered as it is harder to verify claims (reproduce results) or evaluate attacks in environments other than the ones chosen by the original authors. This is reflected by the fact that, until now, barely any comparative evaluations were performed (the sole direct comparison exists only between CountV2 and IKK in [CGPR15]) and [LZWT14; PW16; OK21] in [OK21]. Coupled with the fact that leakage attacks are evaluated mainly on datasets without query logs, i.e., no real-world queries, leaving open the question of efficacy under increased real-world conditions.

Thus, for continuously developing a better understanding of the risk of ESAs leakage and contributing to standardization efforts [NIS21], we designed and implemented our framework called LEAKER with the goal of accessible and effortless leakage attack evaluation. With LEAKER, researchers can easily integrate new attacks and compare them to the state of the art or investigate the effect of countermeasures. Further possible users are researchers with access to proprietary query data, services interested in an assessment of their encrypted systems' vulnerabilities, and individuals keen on learning what a service might infer from their past queries.

4.1 Existing Leakage Attacks

Leakage attacks were first considered by Islam et al. [IKK12], who proposed the IKK attack which exploits co and knowledge of some fraction of user queries to recover the remaining queries. Naveed et al. [NKW15] then proposed inference attacks against PPE and Cash et al. [CGPR15] improved over IKK with their Count attack. What followed was a breadth of new attacks considering a greatly expanded set of settings, assumptions, and adversaries. In the following, we will categorize leakage attacks and present these existing works. We identify two main attack *families*, which we have already discussed in Chapter 3: *keyword* attacks for keyword ESAs, and *range* attacks for numerical ESAs. The objectives are to uncover information about the queries and/or the database. Note that basic comparisons exist in [FVY+17; YZGW20], but these have been largely focused on PPE. Here, we provide an overview of leakage attacks against oblivious and structured keyword (*point*) and range ESAs and classify them according to the following properties.

Types of Adversaries. The two main adversarial models considered against ESAs are *snapshot* and *persistent* adversaries. A snapshot adversary has only access to the encrypted structures and any associated ciphertexts. This captures attackers that, e.g., corrupt a server and read its memory. A persistent adversary has access to the encrypted structures, ciphertexts, and to the transcripts of query and update operations. This captures an attacker that corrupts a server and observes all interactions.

Target. Different information can be targeted. In a *data* reconstruction attack, the adversary tries to recover information about the data, whereas in a *query* reconstruction attack, it tries to recover information about the queries.

Auxiliary data. Many leakage attacks require some auxiliary data or knowledge. A *sampled-data* attack requires a sample from a distribution that is close to the data's distribution and a *sampled-query* attack requires one from a distribution close to the queries. On the other hand, a *known-data* attack requires explicit knowledge of a subset of the data (partial knowledge). A *known-query* attack requires knowledge of a subset of the queries. We leave a systematic evaluation of sampled-data attacks as future work.

Passive or active Attacks. A leakage attack can be either passive or active. In a passive attack, the adversary does not choose any data or queries. In an active attack, it is able to interact with the user, e.g., by injecting data.

This work [KKM+22]. We focus on the evaluation of passive persistent query- and data-recovery attacks where the adversary requires either no auxiliary data, known data or known queries. The reason is that snapshot adversaries are already seen as successful in real-world cases [NKW15]. More precisely, in the case of keyword search we evaluate query-recovery attacks with known-data and in the case of ranges we evaluate data-recovery attacks. We leave the evaluation of sampled-data and sampled-query attacks using real-world data as future work.

Risk factors. We summarize our findings in Table 4.1, where we consider query-recovery attacks that recover more than 15% of the queries and data-recovery attacks with reconstruction error less than 15% on our datasets as successful. This threshold is, of course, subjective so the reader can use our detailed empirical results from Section 4.4 to formulate their own interpretation and conclusions. Note that the recovery rate of an attack depends on a wide range of factors which we identify in Section 4.4. For example, in the case of ranges, these factors include the adversary's knowledge of the data, the query width, and characteristics of the data such as density and whether specific values appear. In addition, we also observed that a skew in the data towards endpoints (1 or N) can result in very different recovery rates (see details in Section 7.2.2.1).

Table 4.1: Major leakage attacks and our perceived risk. The target is either Keyword query (K) or Range data (Value/Count, RV/RC) reconstruction. B is the maximum width and k the amount of missing queries per width. $\mathcal{A}$ denotes that all possible response lengths occur (only within all widths $\leq B$ for $\mathcal{A}_B$, or k missing therein for $\mathcal{A}_{B,k}$). ○ shows no success on our real-world datasets, ◑ denotes some success (K for high partial Knowledge $\geq 75\%$, D for Dense data, W for large Widths close to N, S for Specific data values, E for Evenly and ¬E for unevenly distributed data collections), ● is severe risk across all of our evaluated instances.

Attack	Target	Leakage Profile (Chapter 3)	Auxiliary Data		Assumptions		Risk (Section 4.4)
			Queries	Data	Queries	Data	
IKK [IKK12]	K	co	Partial	Partial	Non-rep.	–	○
DetIKK [RPH21]	K	co	–	Partial	Non-rep.	–	◑ K
Count v.2 [CGPR15]	K	co	–	Partial	Non-rep.	–	◑ K
Subgraph-ID [BKM20]	K	rid	–	Partial	Non-rep.	–	●
Subgraph-VL [BKM20]	K	vol	–	Partial	Non-rep.	–	●
VolAn [BKM20]	K	tvol	–	Partial	–	–	◑ K
SelVolAn [BKM20]	K	tvol, rlen	–	Partial	–	–	◑ K
LMP-RK [LMP18]	RV	rid, rank	–	–	–	Dense	◑ D
LMP-ID [LMP18]	RV	rid	–	–	–	Dense	◑ D
LMP-App [LMP18]	RV	rid	–	–	–	Dense	◑ D
GenKKNO [GLMP19]	RV	rid	–	–	Uniform	–	◑ W∨¬E
ApproxValue [GLMP19]	RV	rid	–	–	Uniform	Specific	◑ S∧(W∨¬E)
ARR [KPT20]	RV	rid, qeq	–	–	–	–	◑ W
ARR-OR	RV	rid, qeq, order	–	–	–	–	●
GLMP [GLMP18]	RC	rlen	–	–	$\mathcal{A}$	–	○
GJW-Basic [GJW19]	RC	rlen	B	–	$\mathcal{A}_B$	–	○
GJW-Missing [GJW19]	RC	rlen	B, k	–	$\mathcal{A}_{B,k}$	–	○
APA [KPT21]	RC	rlen, qeq	–	–	–	–	◑ E

4.1.1 Attacks Against Keyword ESAs

For keyword search, most attacks are *known-data attacks* which require some partial knowledge of the data collection. And these are evaluated by calculation the ratio between the number of correct guesses over the total number of guesses.

The IKK attacks. The first leakage attack was described by Islam et al. [IKK12], who proposed a query reconstruction attack in the persistent model using the co-occurrence pattern co and a known fraction of the queries. Known as IKK it, roughly speaking, solves an optimization problem minimizing a distance between candidate and observed co-occurrence. Since finding the optimal solution is NP-complete, IKK uses *simulated annealing* [KGV83] to probabilistically find an approximation. Originally introduced as a sampled-data attack, we evaluate it as a known-data attack as in [CGPR15]. Recently, Roessink et al. [RPH21] showed how IKK can be improved with deterministic techniques from the Count attack [CGPR15] (described next). This modified attack, which we refer to as DetIKK, reduces the search space for the annealing process and, compared to IKK, requires no query knowledge.

The Count attacks. A simpler attack called the COUNT attack was proposed by Cash et al. [CGPR15]. Like IKK, COUNT is a query reconstruction attack that exploits the co-occurrence pattern co. There are two versions of it. The first, which we refer to as COUNT V.1, requires knowledge of some fraction of the data and queries. Its original description contained a bug which was addressed in an updated version of the paper and lead to an improved variant of the attack, COUNT V.2, which only requires knowledge of some fraction of the data. COUNT V.2 constructs a co-occurrence matrix and compares it to the observed co-occurrences from co. Candidate matches are identified via confidence intervals, and are iteratively eliminated if they are inconsistent with previously confirmed matches.

All three of these attacks, IKK, COUNT V.1, and COUNT V.2, were only evaluated on a subset of the Enron database [Coh15] that was restricted to the highest-selectivity keywords. When evaluated in the same setting, COUNT V.2 outperforms IKK but still requires $> 60\%$ partial knowledge.

The BKM attacks. Recently, Blackstone et al. [BKM20] introduced three new passive query reconstruction attacks. The BKM passive attacks outperform the IKK, COUNT V.1 and COUNT V.2 while exploiting a much smaller leakage profile. The first, VOLAN, exploits the total volume pattern tvol and matches a query to the keyword with the closest expected total volume from the adversary's known data. The second attack, SELVOLAN, extends this by further identifying candidates with a total volume in the known data falling within a window of the expected volume based on tvol. It then selects the best candidate using the response length pattern rlen. The third attack, SUBGRAPH, is a framework used to design several attacks using any *atomic* leakage pattern, i.e., any pattern leaking information about individual documents. It constructs two bipartite graphs: one from the observed leakage and the other from the known data. It then filters candidates via inconsistencies between the graphs, confidence intervals (the leakage roughly has to match the expected value), and an optional cross-filtering that tries to invert the leakage and checks if the candidate appears in all resulting entries. Two concrete attacks that result from the framework are SUBGRAPH-ID and SUBGRAPH-VL, which exploit the response identity pattern rid and the volume pattern vol, respectively.

Blackstone et al. [BKM20] further highlighted the fact that selectivity is a major factor for attack performance which has previously been neglected: For highest-selectivity queries taken from the Enron database, especially the SUBGRAPH attacks work well for a partial knowledge as low as 5%, but if the query space is populated with queries from the database with lowest selectivity, the attacks mostly fail, only uncovering 20% of the queries even at full database knowledge. It remains unclear how selective queries in real-world query logs are. Note that many attacks assume that queries do not repeat (though this assumption can be circumvented if qeq is known).

Prior evaluations. The above attacks were evaluated on the Enron e-mail dataset [Coh15], with [BKM20; RPH21] also using public e-mail sets [CL07]. Since only the code of [RPH21] is open-source, replicating results or comparing the attacks requires additional effort. It is

therefore more cumbersome to evaluate new data sources to, e.g., show an overfitting of the attacks to e-mail data. Furthermore, no evaluation has considered real-world queries but rather just sampled keywords from the data collection in an inconsistent manner (e.g., highest-selectivity [IKK12; CGPR15] vs. lowest-selectivity [BKM20; RPH21]). Given that this enables or breaks the attacks [BKM20; RPH21], evaluation on real-world queries remains an important open aspect that we tackle.

Other attacks not covered. An attack that exploits qeq with auxiliary data was given by Liu et al. [LZWT14], and was recently improved by Oya and Kerschbaum [OK21] also using rid. Recently, Damie et al. [DHP21] and Gui et al. [GPP21] proposed sampled-data attacks. Active file-injection attacks are considered by Zhang et al. [ZKP16], Blackstone et al. [BKM20], as well as Poddar et al. [PWLP20]. Additional specialized attacks against *specific* ESA instantiations were considered in [PW16; VMÖ17; AAGG18]. We focus on general passive attacks that do not rely on auxiliary data.

4.1.2 Attacks Against Range ESAs

We now turn to attacks against oblivious or structured range ESAs. In this setting, there are three variants of attacks: *reconstruction* attacks, *approximate reconstruction* attacks and *count reconstruction* attacks. More precisely, a reconstruction attack recovers the exact values in a numerical collection whereas an approximate reconstruction attack only recovers an approximation of the values. A count reconstruction attack recovers the (approximate) number of times the values occur. Range attacks tend to work “up to reflection", meaning that the attack recovers either the original numerical collection $(e_1, \ldots, e_n)$ or its reflection $\big(N - e_1 + 1, \ldots, N - e_n + 1\big)$. This can be viewed as a loss of 1 bit of information. In our experiments in Section 4.4 we always report the minimum error rate over either the original collection or its reflection.

The KKNO attacks. The first attacks against encrypted range schemes were proposed by Kellaris et al. [KKNO16]. Two attacks were described, both of which are data reconstruction attacks in the persistent model. The first, KKNO-1, exploits the response identity pattern rid and assumes that queries are chosen uniformly at random. At a high level, it determines an entry as having minimal (or maximal) value if it is not contained in the largest proper subset of all identifiers, and determines other entries based on co-occurrence with that entry. The second attack, KKNO-2, only needs the response length pattern rlen but still assumes uniform queries. Intuitively, the attack solves a system of quadratic equations of distances between values and the amount of observed queries with the corresponding response length to uncover the distances between entries. Because both attacks were directly improved upon (described next), we did not evaluate them in our work.

The (G)LMP attacks. Lacharité et al. [LMP18] improved on the query complexity of KKNO-1, by proposing three attacks which we refer to as LMP-RK, LMP-ID, and LMP-APP. These are data recovery attacks in the persistent model exploiting the response identity pattern rid, with LMP-RK also using the rank pattern. Specifically, the third attack only recovers an approximation of the values based on reconstructed intervals. In general, the attacks identify the left endpoints of the queries (e.g., via rank) and assign these values to entries by excluding differing entries seen in the response.

Grubbs et al. [GLMP18] also improved on the KKNO-2 attack with a new attack we refer to as GLMP that only requires the response length rlen. GLMP, however, only recovers the frequency (or *value counts*) of values as opposed to the values themselves. Note that KKNO-2 can reconstruct values using rlen alone but only if the data is dense. The attack relies on the assumption that the queries are made in such a way that all response lengths are observed. At a high level, it reduces the observed response lengths to "elementary" queries (in the sense that they have the form $(1, y)$) and uses graph theory to reconstruct counts based on basic properties of elementary queries.

The GJW attacks. Gui et al. [GJW19] proposed attacks in the persistent model that only exploit the response length rlen for count reconstruction; as opposed to the KPT attacks which also require the response identity and/or the query equality qeq. The main attack, called GJW-BASIC, works when the query width is bounded. It builds an initial solution similar to GLMP and uses a breadth-first search that incrementally extends solutions consistent with the leakage. Modifications were introduced for missing queries (GJW-MISSING) and other countermeasures, which we consider outside the scope of this work.

Approximate reconstruction attacks. Several works attempted to weaken the assumptions needed by the KKNO attacks and their extensions. Grubbs et al. [GLMP19] describe three attacks that do not require density but still assume uniform queries. To do this, these new attacks focus on sacrificial ϵ-approximate data reconstruction instead of exact reconstruction where $\epsilon = 1/N$ yields full data reconstruction. The first is GENKKNO, which extends KKNO-1 to an approximate data reconstruction attack by assigning values to entries based on a comparison of the observed and expected amount of occurrences in the leakage. Due to these estimations being individually symmetric with regard to the endpoints (1 and N), queries close to one endpoint are used to establish a global reflection. The second attack is APPROXVAL, which assumes the existence of at least one entry with values occurring in a specific data range, and uses a single favorably-located entry as an anchor that can be identified more easily than other entries. The values around the anchor are then estimated similarly to the GENKKNO attack. The third attack is called APPROXORDER and uses the PQ-tree data structure to approximate the order of the data collection based on the response identity pattern rid, assuming that the data is not heavily concentrated over a few values.

The KPT attacks. Kornaropoulos et al. [KPT20] describe an approximate value reconstruction attack in the persistent model that is agnostic to the query distribution, denoted as ARR (agnostic reconstruction range). It reduces to the problem of support size estimation, in which the number of outcomes not observed is estimated from the frequency of observed outcomes. By estimating support sizes of identifier subsets using the response identity and query equality patterns (rid and qeq), the corresponding distances and, therefore, the values can be uncovered and are assigned according to an order, which can be uncovered via APPROXORDER. While ARR does not require density or uniform queries, the instantiation of [KPT20] assumes that the values are all unique. They suggest alternatives for dealing with the more general case of repeating values, which we use in our work. We provide more details on this in Section 7.1.4. In our evaluation, ARR uses APPROXORDER to uncover the order, but we also investigate the case where it is directly leaked, which we denote by ARR-OR[1].

Very recently, Kornaropoulos et al. [KPT21] also applied support size estimation to the setting where only the response length and query equality patterns (rlen and qeq) are available, resulting in an approximate count reconstruction attack[2]. It generalizes previous rlen attacks to estimating the number of queries for specific response lengths and solving their relation to value distances, including observations about state-of-the-art encrypted range search schemes [FJK+15; DPP+16]. As a result, the attack is *parameterized* by the ESA, and denoted by APA (agnostic parameterized attack). Crucially, APA deals with multiple possible reconstructions and requires no assumptions about the query distribution and also deals with non-dense data, while the GLMP and GJW attacks require auxiliary data information.

Prior evaluations. No implementations of the previously mentioned attacks are open-source. For their evaluations, [KKNO16; GLMP18; GJW19; KPT21] all use subsets of the HCUP [Age88] medical dataset and the remainder of the above attacks were never evaluated on real-world data. The only practical comparison between attacks (GENKKNO and ARR) is given on artificial data in [KPT20]. To the best of our knowledge, no prior evaluation considered real-world queries, which was identified as an open question in the cryptanalysis of range ESAs [GJW19].

Other attacks not covered. The above works also introduced sampled-data variants: A version of LMP [LMP18] uses auxiliary data information to require fewer queries, while Grubbs et al. [GLMP18] also incorporate frequency information. APPROXDATA [GLMP19] demonstrates how to use APPROXORDER to uncover exact values if density and an auxiliary data distribution are given, and [GJW19] show how GJW-BASIC can be improved with auxiliary information about the data. [GLMP18] also recover values of update operations in case the frequencies have already been determined. k-NN queries are considered in [KPT19; KPT20] and Falzon et al. [FMA+20] and Markatou et al. [MFST21] recently proposed attacks on two-dimensional

[1] Note that order leakage is only common in PPE-based ESAs.
[2] It uncovers ordered values.

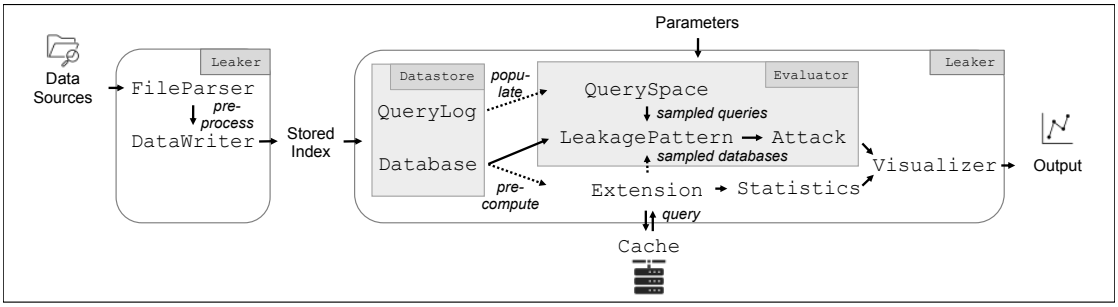


Figure 4.1: High-level overview of LEAKER's major components and attacks and/or statistics evaluation flow. Dashed arrows indicate optional usage. With the exception of `Extension`, all parts can be used for both keyword and range queries.

ranges. Also we did not consider the attack of Markatou et al. [MT19] which targets one-dimensional range queries but assumes that all possible queries are issued.

Note that all of the above attacks (point and range) concern structured or oblivious ESA. Attacks on PPE-based ESAs already indicate practical insecurity [BCO11; KS12; IKK14; CGPR15; NKW15; DDC16; GSB+17; BGC+18].

4.2 The LEAKER Framework

We designed and implemented LEAKER with the following goals in mind:

- *integration:* LEAKER makes the integration and evaluation of new attacks effortless. Researchers can focus on the design and leave the evaluation process to LEAKER. This ranges from interfacing with various data sources to plotting and visualizing the results.
- *comparisons:* LEAKER includes implementations of the main leakage attacks for both point/keyword and range queries. By having attacks implemented in the same framework, they are easier to compare. It also makes it easier to evaluate countermeasures on existing attacks.
- *data sources:* practitioners with proprietary data can use LEAKER to evaluate attacks on their specific data, leading to a tailored evaluation.
- *usability:* LEAKER runs on many platforms and does not require domain-specific knowledge—one only needs to specify the data sources and evaluation criteria.
- *open-source:* LEAKER (including its evaluation scripts) are freely available as open source software at `https://encrypto.de/code/LEAKER`.

4.2.1 Architecture

For interoperability, we implemented LEAKER in Python 3.8. It has 8 149 lines of code (1 148 are tests). It also includes implementations of the major keyword and range attacks, which we discussed in Section 4.1. LEAKER evaluates a leakage attack on a *data collection* using queries from a *query log* and outputs a visualization of the results. A `Database` is the dataset from which LEAKER will generate the data collection and a `QueryLog` is the dataset from which it will choose the queries. LEAKER has a highly modular design, is heavily interfaced, and contains component tests. This achieves our goals of effortless integration of future attacks or combinations thereof, pre-processing mechanisms, leakage profiles, mitigation techniques, query types, and evaluation strategies. Its classes support Python's built-in methods, letting, e.g., a database be used as a context manager or query spaces as lists. The result is an intuitive handling of data that can be processed in any desired way. Figure 4.1 illustrates an overview of LEAKER's modules, which we detail below: a pre-processor, a datastore, an attack and pattern library, an evaluator, a visualizer, and a statistical analyzer.

Pre-processor. The *pre-processor* parses and prepares data collections and query logs for use by the other LEAKER modules. It includes a set of parsers that can be arbitrarily combined to build a data collection or query log from a directory of files. Currently, LEAKER includes file parsers for `.csv`, `.json`, `.xml`, `.txt`, `.mbox`, `.pdf`, `.docx` and `.pptx` files. In the query logs that we identified, range queries were often part of a larger SQL query so we implemented a SQL parser that identifies and extracts range queries contained in complex SQL queries.

Once parsed, LEAKER uses standard information retrieval techniques to tokenize strings into keywords, extract stems, remove stop words, and identify numerical values. Due to its modular design, the pre-processor can be easily extended for new file types by simply implementing a new `FileParser`. We expect that LEAKER's generic pre-processing might independently prove useful for the encrypted search community.

Datastore and cache. After a file has been processed, it is passed to an indexer which stores the data in a set of internal data structures for later use. Numerical data is additionally discretized before being stored. Our choice of data structures to store and manage data collections and query logs is important because one of our main goals is to evaluate attacks on large datasets. For example, compared to previous evaluations of co-occurrence attacks, which used datasets with 500 [CGPR15], 1 500 [RPH21], or up to 2 500 keywords [IKK12], LEAKER evaluates the same attacks on datasets with more than 250 000 keywords (cf. Section 4.3). To achieve this, we use NumPy [HMW+20] arrays to store and process numerical data and Whoosh [Cha12] to store and process keyword data. We chose Whoosh because of its Python compatibilitiy and ability to store additional metadata such as document volume in the index.

To get significant results, a single LEAKER analysis can require a large number of repeated evaluations. We speed up NumPy operations by integrating the optimized just-in-time compiler Numba [Ana18]. To address the costly repeated querying of Whoosh structures on large datasets, we use memoization and store the results of Whoosh queries in a cache so we can reuse them across evaluations. We call the corresponding interface a data `Extension`. In particular, the cache also computes leakage patterns resulting from the query space and the entire database. Note, e.g., that the co-occurrence pattern has a storage complexity requirement of $O(\#\mathbb{W} \cdot (\#\mathbb{W} + n))$. This cache is interfaced as a database `Extension` and LEAKER already provides extensions relevant to all leakage patterns described in Chapter 3.

Attack & pattern library. LEAKER comes with a library of attacks and leakage patterns that can be called on any data collection and query log. We implemented the main attacks from Table 4.1 and—with the exception of GLMP and GJW[3]—verified their correctness by replicating the results from the original papers. The attack library is easily extendable to new attacks. One just has to realize an interface performing recovery given the observed required leakage.

More precisely, the user has to implement a `recover` method to instantiate a new `Attack`, along with the `LeakagePattern` instantiation of the required leakage[4]. For a keyword dataset, the output of `recover` consists of the list of uncovered keywords, or a list of values or counts for a range dataset. All attacks in LEAKER's library are purely in Python except for APPROXORDER [GLMP19], where we use a C++ implementation of PQ-trees [Gro08]. The files are automatically compiled and loaded without requiring any interaction. Since we could not find any public Python-compatible implementations of the Jackknife or Valiant-Valiant estimators used in the ARR and APA attacks [KPT20; KPT21] we implemented them ourselves in Python. To incorporate a new type of leakage pattern, one needs to implement a `leak` method which specifies the information learned given a sequence of queries and a database. Particularly for the keyword case, this method should also provide an optional sampling method for partial knowledge via a database `Extension`. The possibility to add new leakage patterns is extremely useful as there are several works that either alter the form of the pattern or suppress it such as in [CGPR15; BF17; CLRZ18; AHKM19; KM19; PPYY19; DPPS20].

Evaluator. This module evaluates attacks. Given an attack and a leakage pattern from the library and a data collection and query log from the datastore, LEAKER proceeds as follows. It creates the *observed leakage* of the leakage patterns on the data collection and query log. Since this can be expensive (e.g., the co-occurrence pattern requires $O(\#\mathbb{W} \cdot (\#\mathbb{W} + n))$ storage and time) it is also cached. Then, it executes the attack on the observed leakage a number of times (in parallel) and stores the results. To evaluate attacks that require known data

[3] We used synthetic data to verify the correctness of these two attacks because we did not have the original data that was used.

[4] LEAKER comes with instantiations of most common leakage patterns.

and/or queries, it first executes an attack-specific sampling algorithm for the known data. All different attacks on different samples can be run in parallel.

Visualizer. Once all results have been collected, they are used to compute the desired accuracy results (e.g., depending on a choice of available errors) and passed to a visualizer. The visualizer then translates them into graphical `.png` and TikZ plots, which can easily be extended to different error or visualization types. All plots in this work were generated with LEAKER.

Statistical analyzer. LEAKER also includes a statistical analyzer module which computes metrics to evaluate query and data distributions that are then also handled by a specialized `DataSink` which plots statistics over its data sources. This is useful to get a better understanding of a data collection's and a query log's characteristics. All statistic metrics and sinks used in this work are already provided by LEAKER.

4.3 Data Collections & Query Logs

In many instances, attack evaluations were confined to small and/or few data collections without any query logs, which are hard to find. To address this, we describe new datasets including query logs that we believe capture a broad range of realistic scenarios. The datasets include both publicly-available and private data collections and query logs. We provide in Section 7.2.1 a list of other possible data sources that could be used with LEAKER. In Section 7.2.1 we also describe how we pre-processed our data and in Section 7.2.2 we provide some statistics of our datasets. Though some aspects of our datasets are similar to previously-used datasets, we selected data with various properties to see the influence they might have.

Data availability. All data is publicly available, except for the private datasets we collected from our volunteers. All our data pre-processing is integrated into LEAKER. Data not usable for attacks but for statistical purposes is shown in Section 7.2.3. Particularly, a problem is that public data inherently does not model a setting where the data is private, in which a client's behavior might be very different. For these cases, we show evaluations on private data that clients performed on their own machines with LEAKER. Based on these different data, we are in a position to ascertain the severity of leakage attacks in different usage scenarios.

4.3.1 Keyword Data

We present an overview of all of our keyword data in Table 4.2 and give more background in the following.

Table 4.2: Overview of our keyword search use cases and dataset properties. $\#\mathbf{Q}_D$ is the size of the entire log and $\#\mathbf{Q}$ the amount of unique queries. n is the amount of documents and $\#\mathbb{W}$ the amount of unique keywords.

Case	Data	Query log			Data collection	
		#users	$\#\mathbf{Q}_D$	$\#\mathbf{Q}$	n	$\#\mathbb{W}$
Web	AOL [PCT06]	656k	52M	2.9M	151k	268k
Genetic	TAIR [ECW+14]	1.3k	650k	54k	115k	690k
Email	GMail (ours)	6	–	16-100	6k-47k	60k-895k
Cloud	Drive (ours)	1	–	45	200	19k

Search engines. A major proposed application for ESAs are encrypted search engines, e.g., for desktop search applications or to add search capabilities to email clients or file managers. Due to privacy concerns, we were not able to find public datasets matching these settings so we proceeded as follows.

First, we evaluated attacks on the private data of 7 volunteers by providing scripts to locally extract their GMail and Google Drive query logs and data collections. Out of these participants, 6 evaluated the attacks on their GMail accounts and 1 on their Google Drive account. They returned to us basic statistics of their data, the accuracy of the attacks, and their consent to use and publish the results. We will note the average results of all evaluations as well as the worst and best cases. No personal information is included in this work or in LEAKER. We highlight the Drive instance and three GMail instances of different activity levels: GMail-S, GMail-M, and GMail-L. Their basic dimensions are shown in Table 4.2. This data directly shows a *private cloud storage* and *private cloud email* search case.

Because the number of private users we had access to was small and because we cannot release their data, we also used *public* search engine data. Specifically, we used Wikipedia [Wik14] as a data collection and the AOL query log [PCT06]. We recognize, of course, that Wikipedia is *public* so it is not completely representative of the scenarios mentioned above in which one often queries *private* data. Furthermore, it is clear that using AOL queries on a Wikipedia data collection is not ideal but, given the scarcity of real-world data with matching query logs, this is the best one can do at the current time.

Genetic. We were also interested in domain-specific instances that might be queried in a totally different manner. As a prominent case, consider a lab querying large-scale human genetic data for health research, e.g., looking for expressions of specific proteins in gene annotations. Because this query data is very sensitive and not publicly available, we used the following approach.

We performed evaluations on publicly available data that can be seen as related to the above case. Concretely, we use *The Arabidopsis Information Resource* (TAIR) database [LDS+10] as a data collection as well as its publicly released query log [ECW+14]. The data contains genetic annotations and expression information of the *Arabidopsis Thaliana* plant, which itself is not

Table 4.3: Summary of our *scientific data* range query logs on the PhotoObjAll.dec collection [SGT+07] ($n = 5\,242\,134$ entries with domain $N = 10\,456$, density 95.82%, and an even data distribution). $\#\mathbf{Q}_D$ is the size of the entire log and $\#\mathbf{Q}$ the amount of unique queries.

Data	#users	$\#\mathbf{Q}_D$	$\#\mathbf{Q}$
SDSS-S	1	1.4k	215
SDSS-M	1	13.4k	5 562
SDSS-L	1	38.2k	8 220

sensitive. However, we believe it provides a close model for querying in genomic research, as similar information might be queried in the sensitive case of human genomic data.

4.3.2 Numerical Data

We found five datasets that capture scientific, medical, human resources, sales, and insurance scenarios. The dataset characteristics are summarized in Table 4.3 and Table 4.4. The data is discretized by scaling[5], rounding, and mapping to integers which allows us to evaluate attacks without losing too much precision. We display selected data distributions in Figure 1 in Section 7.2.2.1, from which we derive the general risk factors of different distributions.

Scientific data. Data from scientific research can often be sensitive. This is the case, e.g., with data generated from satellites, drug and medical studies, or nuclear experiments. For this, we use the Sloan Digital Sky Survey (SDSS), which contains a variety of astronomical data [SGT+07]. In addition to astronomy, the SDSS has also been used to investigate user behavior [Zha11; NBB+15]. We used the SDSS to create one data collection and 3 query logs (cf. Table 4.3) using a scale factor of 100. We do not notice that the data is significally distributed towards specific values and call it a rather even data distribution in our risk factors (cf. Section 4.1).

The data collection is built from the SDSS's PhotoObjAll declination database which contains $n = 5\,242\,134$ entries with a maximum value of $N = 10\,456$. The query logs we created have different sizes: SDSS-S includes 1 433 queries (215 unique), SDSS-M 13 412 queries (5 562 unique), and SDSS-L 38 273 queries (8 220 unique).

Medical. Due to high sensitivity, medical data has long been proposed as an ESA application and many works used health data like HCUP [Age88] to evaluate attacks [NKW15; KKNO16; GLMP18; LMP18; GJW19; KPT21]. While HCUP is a real-world dataset with millions of patients, its attributes usually have small domain, e.g., a patient's age. While evaluation on

[5] To scale a value we multiply it by a scale factor 10^a, where $a \in \mathbb{Z}$.

Table 4.4: Summary of our range use cases and data with n entries, domain size N, and density δ. E and ¬E denote even and uneven data distributions, respectively (cf. Section 4.1).

Case	Data collection	Scale	n	N	δ (%)	Distr.
Medical	MIMIC-T4 [JPS+16]	×10	8 058.00	73.00	80.8	¬E
	MIMIC-PC [JPS+16]	×10	7 709.00	2 684.00	8.6	¬E
	MIMIC-CEA [JPS+16]	×1	2 844.00	9 978.00	3.3	¬E
HR	Salaries [Gov18]	×0.01	536.00	395.00	2.3	E
Sales	Sales [Wal14]	×1	143.00	6 288.00	2.3	E
Insurance	Insurance [Cit18]	×1	886.00	25 425.00	1.2	¬E

small domains is important, we also wanted to know how various attacks performed on varying and large domains so we considered the *Medical Information Mart for Intensive Care* (MIMIC) dataset [JPS+16] which includes records of medical blood and urine tests performed on ICU patients of the Beth Israel Deaconess Medical Center between 2001 and 2012. We used MIMIC to create three data collections: (1) MIMIC-T4 for patients' free thyroxine, used to evaluate thryroid function; (2) MIMIC-PC for patients' protein/creatine ratio to detect kidney damage or pregnancy; and (3) MIMIC-CEA for patients' carcinoembyronic antigen to detect cancer. We note that MIMIC values are skewed towards low values with just a few high-value outliers, which we call a rather uneven data distribution in our risk factors (cf. Section 4.1).

Human resources. Human resource databases contain personally identifiable information like salaries and demographic information. To capture this, we used a March 2018 snapshot of minimum salaries of the UK Attorney General's Office junior civil servants [Gov18]. Value frequencies are rather uniform, which we call a rather even data distribution in our risk factors (cf. Section 4.1). We scale values by $\times 10^{-2}$ without loss of precision. Our database Salaries, which contains $n = 536$ records, a *small*-sized maximum value of $N = 395$, and has a density of 2.28%.

Sales. Sales data can also be sensitive as it contains trade secrets. We use data released by Walmart for a prediction competition [Wal14], containing weekly sales of department 1 of store 36 from 2010 to 2012. Value frequencies are (except for one outlier) uniform, which we call a rather even data distribution in our risk factors (cf. Section 4.1), and has a number of records $n = 143$, a *domain size* $N = 6\,288$, and a density of 2.26%.

Insurance. Insurance data may be sensitive since it can reveal financial or health challenges. We use a dataset of property damage insurance claims [Cit18] released by the New York Department of Transportation. They were filed with Allstate in September 2018. Values are skewed towards the low endpoint, i.e., a rather uneven data distribution in our risk factors (cf. Section 4.1).

4.3.3 Privacy Considerations

The experiments described in this work were exempt of IRB approval from our institutions. None of the datasets used were de-anonymized and we stress that LEAKER cannot be used to de-anonymize data; its only use is to evaluate the efficacy of leakage attacks. All the datasets we use are public with the exception of the private data for the search engine scenario. We obtained consent from all involved parties to publish the attack evaluations and some statistics. Access to PhysioNet's MIMIC [JPS+16] data is constrained and was only handled by approved authors who completed the required training courses and strictly adhered to PhysioNet's data use agreement.

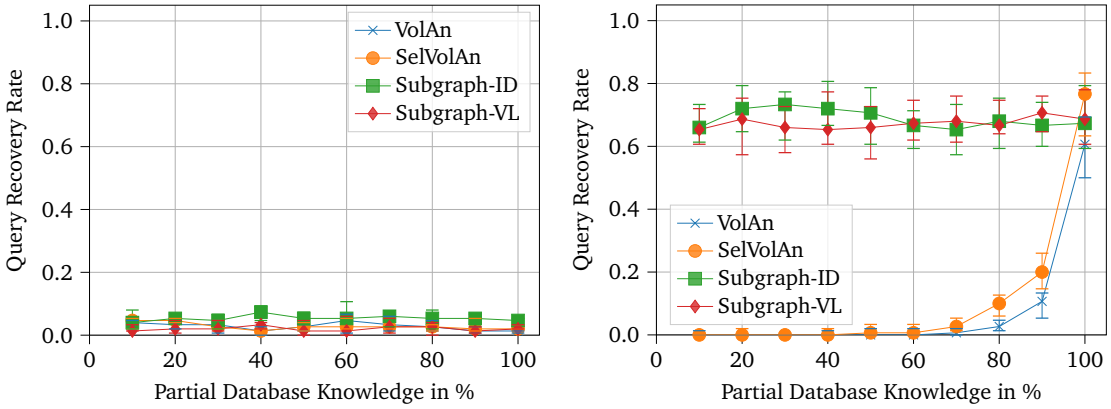
4.4 Empirical Evaluation

In this section, we use LEAKER to evaluate all the attacks described in Section 4.1 on the datasets from Section 4.3 and identify the main characteristics that impact each attack's recovery rate. For both keyword and range attacks, we lay out our evaluation strategy, interpret results, and infer the risk attacks pose in the respective cases. Keyword attacks are evaluated on real-world query logs, while we use both real-world and artificial query spaces for range attacks.

As prior evaluations (cf. Section 4.1, Section 7.2.2) were confined to one respective data setting and sampled queries, we believe our independent re-evaluation on a range of novel datasets *using queries from real-world sources* provides an additional understanding of leakage (see the discussion on ESAs security in Chapter 1).

4.4.1 Keyword Attacks

So far, keyword attacks were analyzed on real data by selecting queries from the keyword space, either just using the ones with the highest frequencies [IKK12; CGPR15] or using highest and lowest selectivities and a range of low frequencies [BKM20]. In fact, [IKK12; CGPR15] even restrict the keyword space to just the highest-frequency keywords. The authors of [BKM20] argue that in cases where users search for specific information, the frequency will be very low and show that attacks do not work in this setting. This is intuitive, as leakage will be confined to very little documents, making it less unique and identifiable in regards to other keywords of low frequency. Having understood query behavior in very different data sources, we challenge these assumptions by evaluating attacks on real-world keyword query logs. An intuition is presented in Figure 4.2: When evaluating the state-of-the-art attacks of [BKM20] in the lowest-frequency case (Figure 4.2 a) *by sampling from the database*, the attacks fail to recover any significant information and confirm the results of [BKM20]. However, the lowest-frequency queries issued by users have a much greater frequency than the lowest-frequency queries in the database. Therefore, the attacks work much better in reality than expected,



(a) Database **(b)** Query log

Figure 4.2: Fraction of correctly uncovered queries of the attacks of [BKM20] on the TAIR keyword database (cf. paragraph 4.3.1). Queries are (a) either taken from the database [LDS+10] (*prior artificial style*) or (b) from a client's query log (*our real-world evaluation*). Here, we use the queries of the respective source least frequent *in the database* (lowest-selectivity). Parameters are the same as in Section 4.4.1.

even when attacking users' queries with the lowest frequency (Figure 4.2 b). In this section, we show that in most cases, even for small-scale private email search with little adversarial knowledge, leakage attacks are highly successful in recovering the queries.

We present results for the private GMail and Drive logs in Figure 4.3 and results for the public AOL and TAIR query logs in Figure 4.4 and Figure 4.5. Further plots are given in Apendix Section 7.1.1. Prior to describing our results, we introduce relevant parameters and our experimental setting.

4.4.1.1 Evaluation Setup and Strategy

Our goal is to evaluate keyword attacks under realistic settings without the need for modeling databases nor query logs. We therefore perform multiple evaluations for multiple clients then plot recovery rates for different fractions of partial knowledge, similarly to prior work. However, for the public data sources, we had to sample queries from the query log instead of attacking the whole log due to its scale. This allowed us to investigate the worst case in terms for attack performance, i.e., lowest-frequency queries. We present the attack results on the *public* AOL and TAIR logs in Figure 4.5 and Figure 4.4, and the results on *private* cloud data in Drive and GMail in Figure 4.3. We shall first describe various relevant parameters to our experiments then summarize our findings:

Frequency. Each query in the query log matches a number of entries in the data collection which is its *frequency*. We investigated two settings: *high* frequency and *low* frequency. The former includes an average frequency ranging from 1 806 to 5 707, whereas the latter ranges from 1 to 859.

In particular, we noticed that queries with a high frequency have responses with distinct lengths (volumes), yet have a non-trivial overlap when it comes to the response identity, i.e., every two queries share several records in their responses. For a very low frequency, the opposite holds: fewer records are shared between every two queries, whereas the volume of the query responses tends to be the same.

In order to assess the impact of varying frequencies we populate query spaces with *different selectivities* from the query log (not the data collection, though the frequency is computed on the data collection). We look at the most and least active clients, separately. The resulting mean frequencies of the respective query space can be seen in Figure 4.5 as well. We note an anomaly of a low amount of unique queries found in the database for the most frequent AOL users, which results in the query spaces for highest and lowest frequency to be equal. As indicated by the high selectivities of the queries (cf. Section 7.2.4.1), attacks are mostly rather successful. At a higher frequency, volume information of multiple documents becomes very unique, whereas the keywords are so frequent in the database that identities become less unique for them. For a very low frequency, the opposite holds: fewer documents cannot be distinguished by volumes, but the individual identifiers the clients query for still amount to rather unique information. Yet for very low frequencies ($\mathsf{freq}(\mathcal{Q}) = 1$), however, we uncover a surprising result: SUBGRAPHVL does not work at all, while SUBGRAPHID can still correctly recover queries at a low but significant rate, we believe this is due to the underlying leakage types.

Apart from that, the volume analysis attacks of [BKM20] only perform well for a high frequency and a high partial knowledge. COUNTV2 [CGPR15] outperforms them only for queries of very high frequency ($\mathsf{freq}(\mathcal{Q}) = 5\,707$), which is just reached in the aggregated query space, where it achieves significant recovery rates with 30% or more partial knowledge. We also evaluated IKK [IKK12] using partial knowledge. However, due to the involved database sizes, we stopped it after a runtime of over 48 hours per run, after which it did not recover more than the 15% of queries leaked to it even with full adverserial knowledge and highest frequency queries in the aggregated space. We therefore conclude that IKK is not feasible for large-scale databases.

Number of users. When it comes to the number of users, there are two main settings in which structured and oblivious ESAs can be deployed: *single-user* or *multi-user*. The former characterizes a setting in which a single user queries its dataset, while in the latter multiple users query the same dataset. In the single-user case, we evaluated the attacks by taking the average recovery rate over the queries of 5 users each of which made at least 2 000 queries. In the multi-user setting, we evaluate attacks on a sample of the query logs which, in the case of AOL consists of 656 038 users and in the case of TAIR consists of 1 263 users. For the

private Google data, we only evaluated the attacks in the single-user setting since each user queried their own data collection. While the number of users in our private Google dataset is low, the experiments on AOL and TAIR can give some indications as to how the attacks would perform on datasets with a large number of users.

User activity. We noticed that users have different query activities, with some issuing a lot of queries while others are less active. Over all of our datasets, with the exception of the Google data, the most active user issued 6 389 queries while the least active user issued 2 000 queries. Since we did not find any noticeable difference in recovery rates (for all attacks) against the most active and the least active users, we only report the least active case in Figure 4.5. The most active case is provided in Figure 3 in Appendix.

Query sampling. Real-world query logs can be extremely large, and given the computational complexity of some attacks, we had to limit the number of queries we used. To do this we sampled the query logs using two approaches which we refer to as *full* sampling and *partial* sampling. Full sampling outputs a new, smaller query log composed of keywords sampled independently of whether they exist in the adversary's known-data set or not; in other words, it is possible that a keyword appears in the query log but is unknown to the adversary. In contrast, partial sampling generates a new and smaller query log that is only composed of keywords in the adversary's known-data set. This captures a worst-case scenario where the user only queries for keywords in documents known to the adversary.

We consider partial sampling because, contrary to having real-world queries, we have no data to indicate which parts of the data collection an adversary might obtain for its knowledge. This captures results for a worst case in which a user only queries for keywords that appear at least in one of the records in the adversary's knowledge. For each of the public query logs, we sampled logs of size 500, 2500 and 5000. In Figure 4.3, the size was equal to the entire length, i.e., the attacks targeted the entire query logs. Since we did not observe any major deviations in the recovery rates of the attacks based on size we only show the results for (sampled) query logs of size 500. Since the private query logs were already small, we did not sample them.

Query repetition. We observed that some of the queries in the log are repeated whereas some previous works assumed distinct queries. This assumption makes sense if the adversary can distinguish between queries based on the query equality (which is disclosed by many structured ESAs) and ignore repeated queries but the assumption no longer holds for oblivious ESAs or structured ESAs that do not leak the query equality [KMO18; GKM21]. Given that some attacks apply to both structured and oblivious ESAs and may be affected by this, it was important to evaluate both settings: *with* and *without* repetition.

Clients. We investigate two parameters concerning clients to establish if a client's activity has an influence on the attacks performance, we look at the different activity levels and take the most and least active clients, respectively in Figure 4.5. We limit individual clients to 5 to allow for a practical evaluation, and require a minimum of 2 000 queries to represent meaningful query spaces. Furthermore, in Figure 4.5, we evaluate the case where queries are aggregated from *all* clients. As expected (cf. Section 7.2.4.1) and seen in Figure 4.5, a *client's activity* has no influence over the attack performance.

Query distribution. Increasing the number of the samples does not introduce any queries with a frequency so low that would be of significance to be accounted for the attacks accuracy. Our intuition holds true after including additional query space sizes of 2 500 and 5 000 to the experiments of Figure 4.5, where they recover similar fractions of queries for all sizes.
Nonetheless, significant information could still be uncovered as a fraction of the chosen query sampling approach, as an example if we assume that queries are not repeated, then this can be used to exclude candidates by the attacks. To see the effect of this assumption, we show in Figure 4.4 an evaluation where the queries are sampled with replacement, compared to when sampling without replacement; cf. Figure 4.5)

Experimental setting. For public datasets, all our evaluations were run on an `Ubuntu 20.04` machine with 390GB memory and 1TB disk space. Unless explicitly specified, every attack is evaluated as follows: given a query log, a data collection, and a specific combination of the parameters highlighted above, we sample a new query log and data collection. For a fixed rate of adversarial knowledge (i.e., known-data rate), we first sample a subset of the entire data collection ℓ times, and for each sample, we sample a subset of the query log λ times. We denote this as a $\ell \times \lambda$ evaluation and it accounts for $\ell \cdot \lambda$ evaluations. For most experiments we picked $\ell = \lambda = 5$ and display the median, maximum and minimum recovery rate.[6] We vary the adversarial knowledge from 10% to 100% with a 10% incremental step. In the public setting, we attack 150 queries drawn from the query log according to their frequencies. We denote by `X-Y` a setting in which the type of user and the frequency are set to `X` and `Y`, respectively.[7] Recall that `X` can be set to *all* users (`A`) or to the single-user setting (`S`). `Y` can be set as *low* (`L`) or *high* (`H`) frequency. For private data, each participant ran the evaluation on their own machine using their entire query log, which did not require any query sampling process.

[6] We limited evaluations to 25 *per user* due to the computational overhead incurred by some attacks. As an instance, the COUNT v.2 attack took one day to complete a single iteration for 5 users. In addition, we were also pushed to bring down the values of some parameters in order for some of the attacks to work. Similar to any cryptanalysis work, we believe that future attacks should not only provide their recovery rates, but also a precise analysis of the computational cost.

[7] For ease of exposition, we mainly focus on these two parameters, but we also varied the type of the query sampling, the query repetition, or the length of the query log.

Discussion. There are obvious limitations to some of our experiments. One example is in how we obtain the adversary's known-data. Since we do not know how an adversary would choose/obtain this data in practice, we have to make some assumptions. For our experiments, we chose to run experiments with multiple known-data samples chosen uniformly at random but other approaches could be considered and it would be interesting to know how they affect the recovery rates. Another aspect is in how we sample the public query logs to generate smaller more tractable logs. We considered different strategies and opted for uniform sampling but, again, one could consider other ways to sample. In the following, we discuss our results for each of the attacks we have evaluated.

4.4.1.2 The IKK Attacks [IKK12; RPH21]

Given the attacks' quadratic costs in the number of keywords, evaluating them on TAIR was infeasible so we only considered AOL.

IKK did not work in any of our settings. This stands in contrast to previous results which showed high recovery rates but assuming almost full knowledge of the data [CGPR15; RPH21] on small datasets. The best recovery rate we observed was less than 15%; even with full knowledge of the data and in the A-H setting (all users, high frequency), which is the least realistic setting. We stopped the attack's annealing process after it ran for 48 hours. We attribute this to a large search space, since the DETIKK [RPH21] attack—which has a reduced number of states—does not suffer from this in the high-frequency setting (cf. Figure 4.5). This evaluation suggests that DETIKK may only work in practical settings with high known-data rates.

4.4.1.3 The Count Attack [CGPR15]

Like IKK, COUNT V.2 is also quadratic in the number of keywords so we only ran it on the AOL query log.

In our evaluations, COUNT V.2 succeeded in all of our settings but using full knowledge of the data. Without full knowledge, it only succeeded in the A-H setting with a recovery rate of 63% based on a 30% known-data rate. Here, the query frequency had a mean of 5 707. This aligns with the results stated in [BKM20]. However, we observed that without full knowledge of the data, COUNT V.2 failed to achieve adequate recovery rates. For example, in the S-L setting, it achieved 8% recovery rate even with knowledge of 90% of the data (cf. Figure 4.5 for more data points). Similarly to IKK, this suggests that COUNT V.2 may require very high known-data rates in more practical settings.

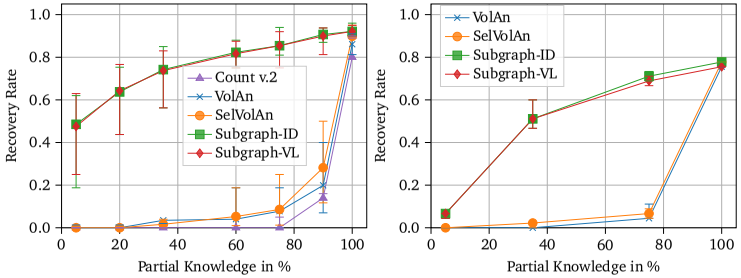


Figure 4.3: 3 × 3 private data evaluations (full query sampling, no repetitions, and single-user setting) on GMail (left) and Drive (right). Only one user evaluated COUNT V.2 for GMail.

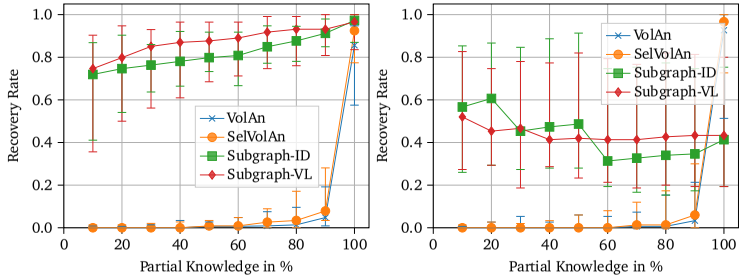


Figure 4.4: 5 × 5 evaluation of [BKM20] on lowest-frequency queries from the 5 least active AOL users for full sampling without restrictions (left) and sampling with replacement (right).

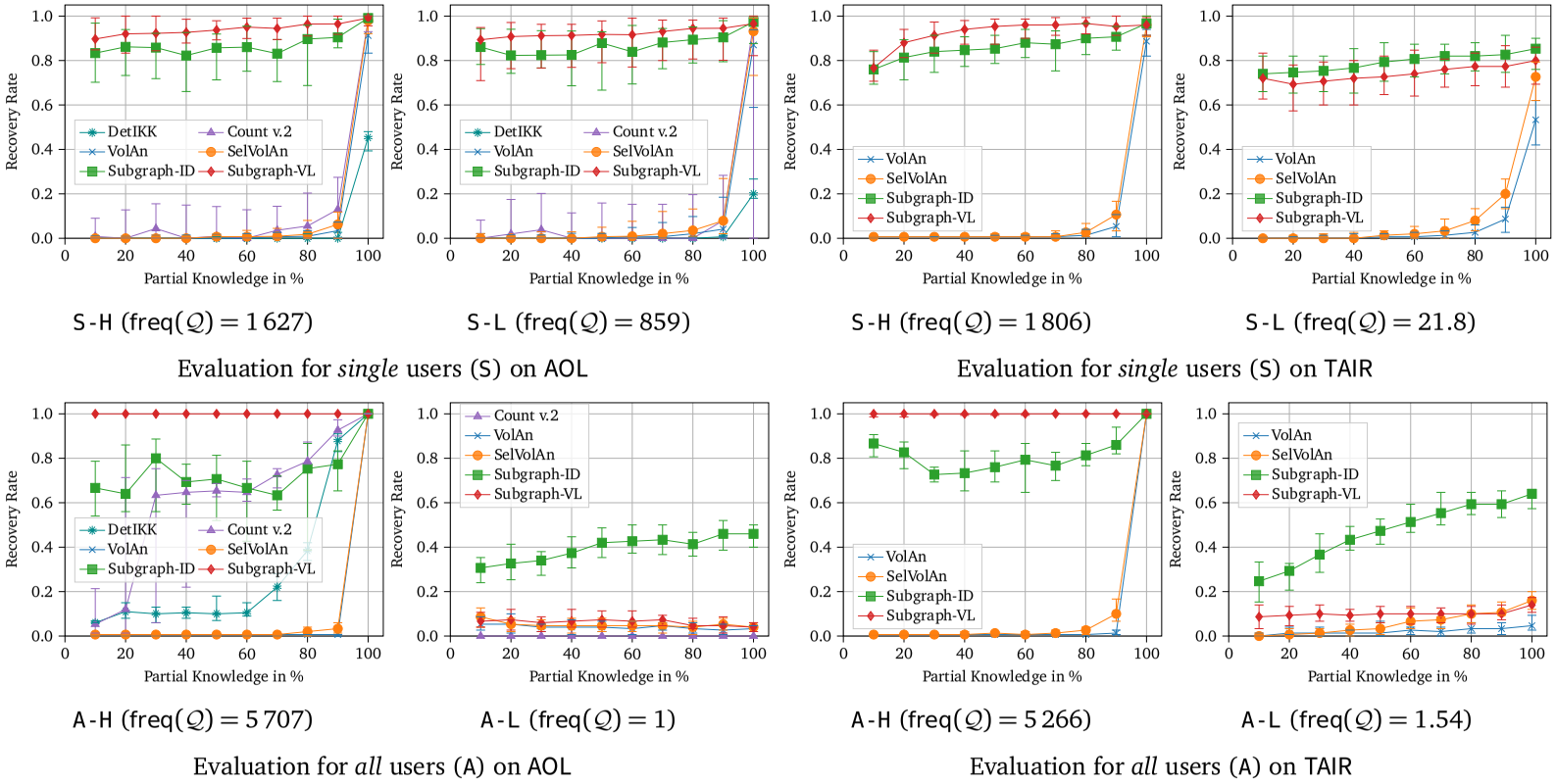


Figure 4.5: Attack evaluations against AOL and TAIR. All evaluations are 5 × 5, except for 3 × 3 evaluations done for COUNT V.2 and DETIKK. 150 queries are drawn for either each of the least active users (top; single user setting S) or all users (bottom; A) without replacement according to their query frequency from the 500 most (H) or least (L) selective queries in the query log that are also contained in the partial knowledge, respectively. The resulting mean frequency is given by freq($\mathcal{Q}$).

4.4.1.4 The BKM Attacks [BKM20]

The BKM attacks were efficient enough to evaluate on all of our query logs. We provide our results in Figure 4.3 and Figure 4.5 and detail our results for each of the attacks below.

The Subgraph framework. Against our private dataset (Figure 4.3), both attacks did surprisingly well. For the email case, more than 18.75% of the real-world queries are recovered with knowledge of only 5% of the data.[8] Though the private dataset had a very small number of users, we did observe that the recovery rates of the Subgraph attacks on the Gmail data were very consistent across users and always greater than 18.75%. On the Drive dataset, the attacks achieved a much lower recovery rate with low known-data rates; however when 35% of the data is known, both attacks recovered half the queries.

In the public data setting (Figure 4.5), both SUBGRAPH attacks also achieve non-trivial recovery rates. For high-frequency queries, SUBGRAPHVL achieves very high recovery rates almost independently of the known-data rate and of whether the queries come from a single user or aggregated users. SUBGRAPHID, however, has a slightly lower but still significant recovery rate: 86% in the S-H setting for AOL with known-data rate of 10%. For low-frequency queries (.-L), both attacks still perform well for an average frequency as low as $\mathsf{freq}(\mathcal{Q}) = 4.85$ (cf. Figure 3 in Section 7.1.1). For very low frequencies ($\mathsf{freq}(\mathcal{Q}) = 1$), however, we found that SUBGRAPHVL does not work at all, while SUBGRAPHID can still correctly recover queries at a low but significant rate (cf. Figure 4.5). This is in contrast to the original evaluation of the attack [BKM20], where queries with frequency 1 (sampled from the data collection) could not be recovered. This again illustrates the value of using real-world query logs for evaluations as it can uncover new settings in which existing attacks work.

In Figure 4.4, the queries are fully sampled from the query logs and we allow for repeated queries. We noticed that: (1) the variance of the recovery rate increases; and (2) the recovery rate of SUBGRAPH is reduced by about 40% if we allow for repeated queries. Both attacks achieve lower minimum recovery rates—around 40% for 10% known-data rate—whereas their median recovery rate drops to similar levels for all known-data rates if queries repeat. We can see that the recovery rate is affected by full sampling and repeated queries but remains significant in all cases. The attacks also work slightly better even with lower known-data rates on AOL compared to our private datasets (Figure 4.3). A possible intuition why these attacks may work better than others is that they rely on atomic leakage concerning individual documents. This difference could be because AOL has significantly more users but we believe more data is necessary to confirm this.

Total volume attacks. Compared to the SUBGRAPH attacks, the VOLAN and the SELVOLAN attacks achieved significantly lower recovery rates in the private data setting. More precisely,

[8] Given that email data often contains public information such as updates and spam, we consider a 5% known-data rate to be realistic.

Table 4.5: Normalized mean errors on the entire SDSS query logs. The collection is sampled 25× uniformly at random with size $n = 10^4$ ($n = 10^3$ for APA and ARR).

Instance	GKKNO	AVALUE	ARR	ARR-OR	APA-OR^{BT}	APA-OR^{ABT}
SDSS-S	0.413	0.432	0.473	0.249	0.242	0.239
SDSS-M	0.408	0.435	0.287	0.128	0.242	0.240
SDSS-L	0.417	0.456	0.286	0.141	0.241	0.242

they only achieved 20% recovery rate on the GMail dataset even with 75% known-data rate (see Figure 4.3).

We also noticed that the attacks did poorly in the public setting considered in Figure 4.5. For high-frequency queries, they needed almost perfect knowledge of the data to achieve an adequate recovery rate. Specifically, at least 70% of the data needs to be known in order to recover more than 7% of the queries in the S-H setting. For very low frequencies, none of the attacks worked with less than 100% known-data rate. Therefore, based on our datasets, we only consider them to pose a risk for very high known-data rates, though this may not necessarily generalize to all cases.

4.4.2 Range Attacks

Range Attack evaluations have primarily been performed on random databases or on multiple columns of a restricted set of real-world databases using artificial query distributions. Many attacks are concerned with full reconstruction, and hence the approach has been to evaluate a lot of instances of small-scale databases found in a data source. So far, the approach for full reconstruction range attacks has mainly been to evaluate the success probability (or error) over a lot of instances of small-scale databases from a data source, given the theoretical minimum of queries. We ran our evaluation against the numerical datasets described in Section 4.3.2.

Our results are summarized in Table 4.5 and Figure 6. Similar to our keyword attack evaluation (Section 4.4.1), we first describe the main parameters that impact accuracy as well as our experimental setting, and then proceed to a high-level description of our results. If an evaluation point is not shown in any of the tables or figures in this section and unless noted otherwise, we aborted it due to unacceptable runtimes.

Query distribution. This captures how a user queries its own dataset. For the SDSS dataset, we were fortunate to have access to both the query log and its corresponding data collection. For datasets with no available query log, we had to consider synthetic query distributions. Previously considered query distributions include the uniform distribution and instances of the beta distribution [KPT20; KPT21]. Given that none of these distributions are supported by real-world query logs, we introduce a new distribution we call the *truncated Zipf* distribution which has some of the basic properties of the distributions we observed in SDSS (cf. Section 7.2.2.3).

To summarize, this distribution is a variant of the standard Zipf(a, N) distribution where we fix a to be 5. It also has two additional parameters: the *maximum width* B, and the *fraction* f. The former removes any range that has width larger than B, and the latter ensures that only a fraction f of possible ranges occur. We denote this new variant of the Zipf distribution TZipf(B, f), where the distribution parameter a is always 5, the maximum window size of a query is B, and a maximum fraction of f of all unique queries can appear. We varied both B and f in our experiments and found that B can have a significant impact on recovery rates. Thus, we present results for a (relatively) small B and a (relatively) large B, and set small fractions of f that still give us a meaningful query space (20% for small, 2% for medium and large and 0.2% for very large collections).

Amount of queries. For the SDSS dataset(cf. Table 4.3), we did not sample the query logs and, instead, attacked the entire query logs, see Table 4.5. For the other numerical datasets, however, we sampled varying amounts of queries (10^2 to 10^5) with replacement.

Characteristics of the data. The datasets we use and that are described in Section 4.3, have different characteristics and properties which allows us to assess attacks in different scenarios. We recall some of their basic properties here (more details are provided in Section 7.2.2.1): the medical dataset, MIMIC, and the insurance dataset, Insurance, are skewed towards low values with just a few high-value outliers so we refer to them as *uneven* distributions, whereas entries in the human resources dataset, Salaries, and the sales dataset, Sales, are spread more evenly across the domain range so we refer to them as *even* distributions.

Data distribution. Through our various data collections, we also investigated different data distributions. Though we cannot show the exact distributions due to access restrictions, we note two major properties here: (1) MIMIC-T4 has a dense subset including all values in $\{1, \ldots, 42\}$, whereas such continuous dense subsets are not found in other collections. (2) MIMIC and Insurance are skewed towards low values with just a few high-value outliers, whereas entries in Salaries and Sales are spread more evenly across the domain range (the Sales distribution is almost uniform).

Data density and size. Other basic properties are the size of a dataset and the density of an attribute/column of dataset. The size refers to the number of records in the collection and the density of an attribute/column is the ratio of unique values in the attribute/column over its domain size. Note that the size of that n can be larger than N (cf. Table 4.3 and Table 4.4) We recall that the medical dataset MIMIC-T4 is much more dense than the other datasets.

Errors. These errors are a general indicator and enable comparisons across instances, though the severity still depends on the use case. For instance, a 10% error on salary data can already give a clear indication of income whereas a 10% deviation in a medical test result can alter a conclusion from clinically inconspicuous to noteworthy. Concerning database reconstruction, we use the *normalized mean absolute error* (MAE):

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|\mathsf{DB}[i]-\tilde{\mathsf{DB}}[i]|}{N}.$$

We noticed that due to the low density of our databases, the count reconstruction attacks fail at identifying values that do not occur in the database. Though this already shows that these attacks will not perform well in such real-world scenarios, we use a more meaningful error to highlight whether density is the issue at hand, or if the attacks also fail to uncover the counts of values that do appear. We call this the *normalized mean count error* (MCE) based on $S = \mathsf{sorted}\big(\{e : e \in \mathsf{DB}\}\big)$ with cardinality m:

$$\frac{1}{m}\sum_{i=1}^{m}1-\frac{min(|\mathsf{DB}(S[i])|,\widetilde{C}[i]|)}{max(|\mathsf{DB}(S[i])|,\widetilde{C}[i]|)},$$

which indicates how far off the count reconstruction is, e.g., on average 10% off the original value. We always report errors up to reflection (cf. Chapter 3).

Experimental setting. Contrary to the keyword attacks we evaluated which were all query recovery attacks, the range attacks we consider are all data recovery attacks. To measure success[9], we use a *normalized mean absolute* error. This error is a distance measure equal to the mean of the normalized differences between the true and recovered values. For count reconstruction attacks, we compute the error between the *sorted* values, i.e., independent of the order. To make this difference clear and comparable to all attacks (disregarding order information), we add the suffix -OR in the plots for these cases. As a reference, an error close to 0.5 means that the attack does not work, while one close to 0 is synonymous to a practical attack. Every attack is evaluated several times and we report the mean, maximum, and minimum error.

Our SDSS evaluations cover realistic settings by attacking the collection with leakage observed from real queries by real users. For the other datasets, we made assumptions about the query distributions, rooted in observations about the real SDSS queries. Of course, other distributions might be encountered in realistic conditions, but this allowed us to observe interesting effects of the different data properties under query distributions similar to ones that have been observed in a real system.

[9] We use the same machine as in the keyword case. Every range attack is evaluated against all datasets and all query distributions.

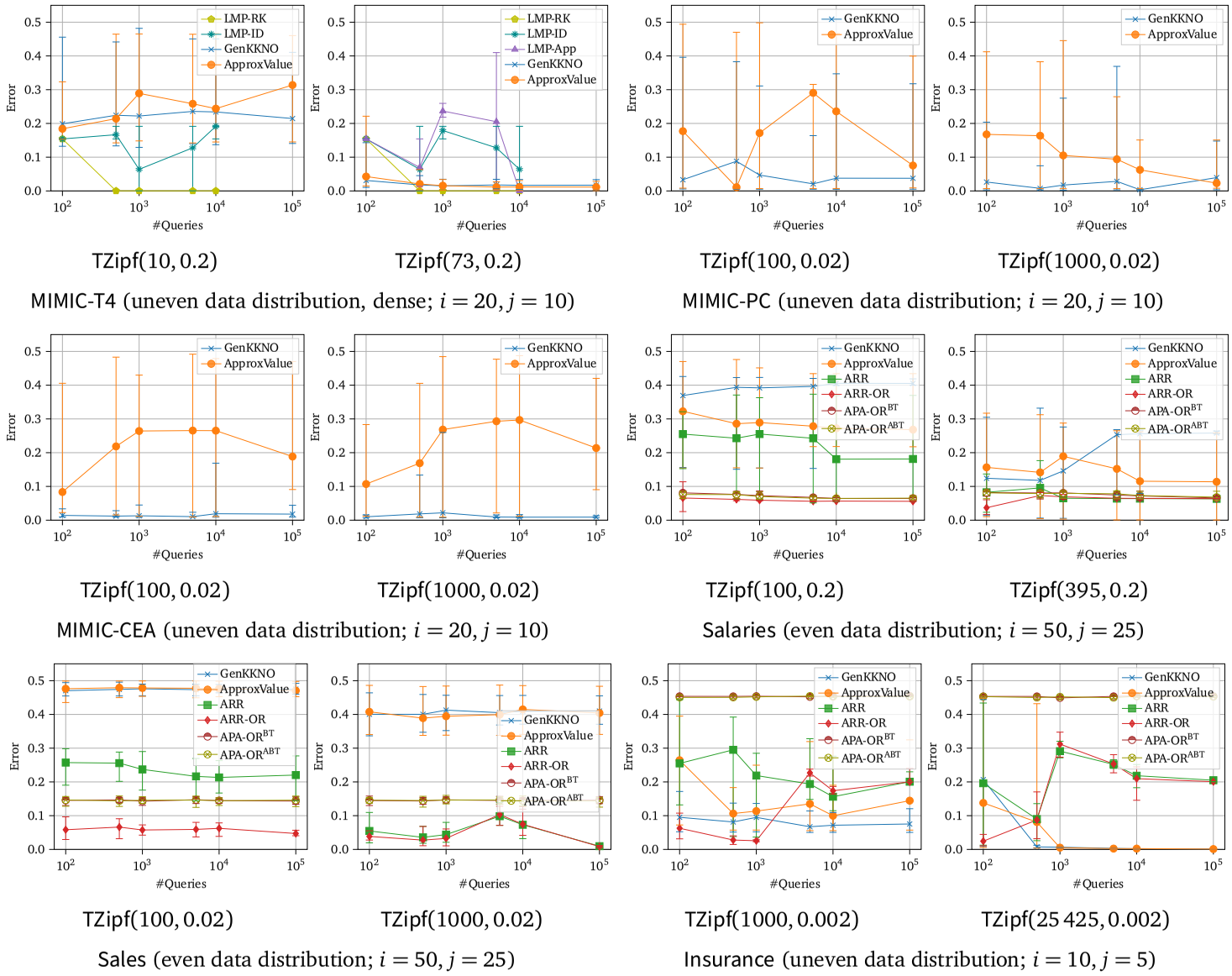


Figure 6: Normalized mean absolute error of value reconstruction attacks on different datasets using a truncated Zipf distribution. Captions include the respective dataset's general data distribution (cf. Section 4.1, Section 7.2.2.1) identified as risk factors. i resp. j iterations were performed for GENKKNO and APPROXVAL resp. LMP, ARR, and APA.

4.4.2.1 The (G)LMP Attacks [GLMP18; LMP18]

As expected, the LMP [LMP18] attacks were successful on MIMIC-T4, which is dense, under uniform queries. Yet, all LMP [LMP18] attacks failed completely across all other instances, including SDSS dataset. This is because all this data is non-dense, and the attacks even fail under the uniform distribution.

However, there are MIMIC-T4 instances that do not fail under the TZipf distribution. Instead, they exactly uncover all values within the dense subset $\{1, \ldots, 42\}$, but assign incorrect values to the outliers, they achieved very small error rates (e.g., LMP-RK achieves a 0.0003 error rate) because the outliers represent a tiny fraction of the data (around 0.37% of all values). We thus consider the attacks to only be applicable if the database is very dense. If the attacks output the reflection, the error is significantly larger. Based on our experiments, as predicted, we consider LMP as risky if the data is dense.

The GLMP [GLMP18] attack, which is a count-reconstruction attack, achieved perfect recovery on MIMIC-T4 if we disregard density.[10] This occurred with 10^5 queries from a truncated Zipf distribution with $\mathsf{B} = 73$ and $\mathsf{f} = 0.2$. The attack failed on the other datasets, While it finds solutions, they are assigned to the wrong values. As such, we believe that GLMP succeeds if the collection is completely dense and the query distribution is uniform, though similarly to LMP this may not necessarily generalize to all cases. Furthermore, if we disregard density, the correct counts are almost exclusively only found with a uniform query distribution (the only exception is for 10^5 queries under $\mathsf{TZipf}(73, 0.2)$ for MIMIC-T4). We do not consider it to work under real-world conditions.

4.4.2.2 The GJW Attacks [GJW19]

With the exception of the Salaries case, both GJW-BASIC and GJW-MISSING suffer aborts due to an infeasible search space larger than $|\{r \in \mathsf{rlen}\}|^3$. More precisely, for Salaries, the recovered counts were always completely incorrect when queries are sampled from TZipf, and while the correct counts were uncovered for a uniform distribution for 10^5 queries, they were assigned to the wrong values due to the low density. Similarly to GLMP, we do not consider them to work under our real-world conditions, though this may not generalize to all cases.

4.4.2.3 Approximate Reconstruction Attacks [GLMP19]

Though the GENKKNO and APPROXVAL attacks of [GLMP19] were explicitly designed to work with uniform queries, we still evaluated them using our SDSS data collections and query logs. As expected, they failed to recover any meaningful data, achieving an error of at

[10] Running the attack on a collection that has been made completely dense by removing values that do not occur from the collection's universe.

least 0.41 (cf. Tab. 4.5). However, and perhaps surprisingly, we did find that both attacks achieved significant recovery rates when queries were sampled from the truncated Zipf distribution (cf. Figure 6).

We identified two settings where GENKKNO succeeds with a truncated Zipf: (1) when it has relatively large B; and (2) when the data is skewed towards lower values as is the case in MIMIC and Insurance, where it even succeeds for a small B. Note that there was an exception to (1) which was when we evaluated it on the Sales dataset. Generally, we believe (1) holds because queries with large width are more likely to cover values close to one of the endpoints (1 or N), which is required to determine the global reflection (i.e., whether the value belongs to the first or second $N/2$-half of the domain).[11] For (2), we believe that the skewness of the values in a dataset helps to easily determine one of the endpoints, and therefore the global reflection. This is not the case for Sales, where the probability of hitting any value is almost uniform, as seen by an error larger than 0.4 for $\mathsf{B} = 100$.

The same holds for APPROXVAL, under the additional condition that specific values have to be present in the collection. Namely, the attack assumes that at least one value in the dataset is in the range $[0.2N, 0.3N]$ or its reflection to find its anchor. Though this is true for all data collections, the fraction of such records is much lower for MIMIC-PC (0.03%) and MIMIC-CEA (0.4%) than the others ($\geq 2.9\%$), which are exactly the cases where it has much worse and unpredictable performance compared to GENKKNO with a maximum error equal to 0.49 for 5 000 queries, see Figure 6.

4.4.2.4 The KPT Attacks [KPT20; KPT21]

Contrary to the previous attacks, these attacks achieved low error rates on the SDSS data and queries but were computationally demanding since they have to solve non-convex or nonlinear optimization problems with solution size $n + 1$. This computational overhead also meant we could not evaluate them on our MIMIC datasets. We also had to rely on SciPy [VGO+20] instead of the original MATLAB optimization to meet our open-source goals (cf. Section 4.2). In the following, we give more details on attack performance.

ARR and ARR-OR. Recall that ARR-OR requires the order as an input which is not the case for ARR. ARR-OR reconstructs with error rate 0.15 in almost all of our settings but standard encrypted ranges schemes do not leak the order [FJK+15; DPP+16]. In our experiments, ARR only came close to ARR-OR's performance when queries were sampled from truncated Zipf distributions with large B.

This includes SDSS, where ARR-OR is the only attack to succeed in attacking a real-world query log. While the error is still high (0.249) for SDSS-S containing only 215 unique queries, the error drops to 0.128 for SDSS-M containing 5.6k unique queries (cf. Tables 4.3

[11] This is not the case for the SDSS logs, also diminishing accuracy.

and Table 4.5). This stands in contrast to ARR, where no significant SDSS information is uncovered.

For the other datasets, ARR-OR achieves low errors even for just 100 queries and low maximum query widths B, see Figure 6. Thus, as order is usually not leaked, the ARR attack is sensitive to the order reconstruction of ApproxOrder [GLMP19], which diminishes performance for low B.

We also note a peculiarity of ARR against Insurance, where the error rate with 10^3 queries was significantly larger than with 100 queries. This stems from slightly overestimating distances between entries with the same, low value making up almost half of the collection (cf. Figure 1 in Section 7.2.2.1). We believe this to occur for more queries as they yield more possible results outside this specific, low value, thereby decreasing the weight for its result set, which consequently enables a low yet cumulatively significant error. the error btwn ARR/ARR-OR does not matter here as it is just caused by the order permutation

APA. Since APA [KPT21] is parameterized by the underlying range scheme, we show results for the state-of-the-art schemes [FJK+15] and [DPP+16]. Also, APA recovers ordered values, whereas no attack uncovering the order based on this leakage profile is known. In general, APA is the only range attack not utilizing rid that can uncover significant information outside the case of dense data and a uniform query distribution. This was not the case for GLMP and GJW that also do not rely on rid (but do not need qeq).

Although APA only achieves an error rate of about 0.24 on the SDSS data collection and queries, it performs well on other datasets and with various query amounts and query widths. It is sensitive, however, to the data distribution. In particular, while it achieves an error rate of 0.06 on Salaries and of 0.15 on Sales, it had an error rate of 0.45 on Insurance.

For the latter case, the found solutions greatly underestimate large value distances while they are more accurate in the former case on an even data distribution containing smaller distances, which we thus see as a risk factor for APA. The impact of using different range schemes seems to be insignificant.

4.4.2.5 Evaluation on Real-World Query Logs

We present results on attacking SDSS in Table 4.5. Additionally, we ran the LMP attacks but had to abort all of them due to significant runtimes before any results were uncovered, with the exception of those on SDSS-S, where all attacks failed to find a solution. The count reconstruction attacks of [GJW19] similarly all abort due to a prohibitive search space, and, while not aborting, GLMP18 is never able to find a solution.

The remaining results show that GeneralizedKKNO and ApproxValue of [GLMP19] completely fail. GeneralARR struggles as well, but improves over [GLMP19]. Significant reconstruction is only achieved by GeneralARR-OR, giving us an answer to question **1)**

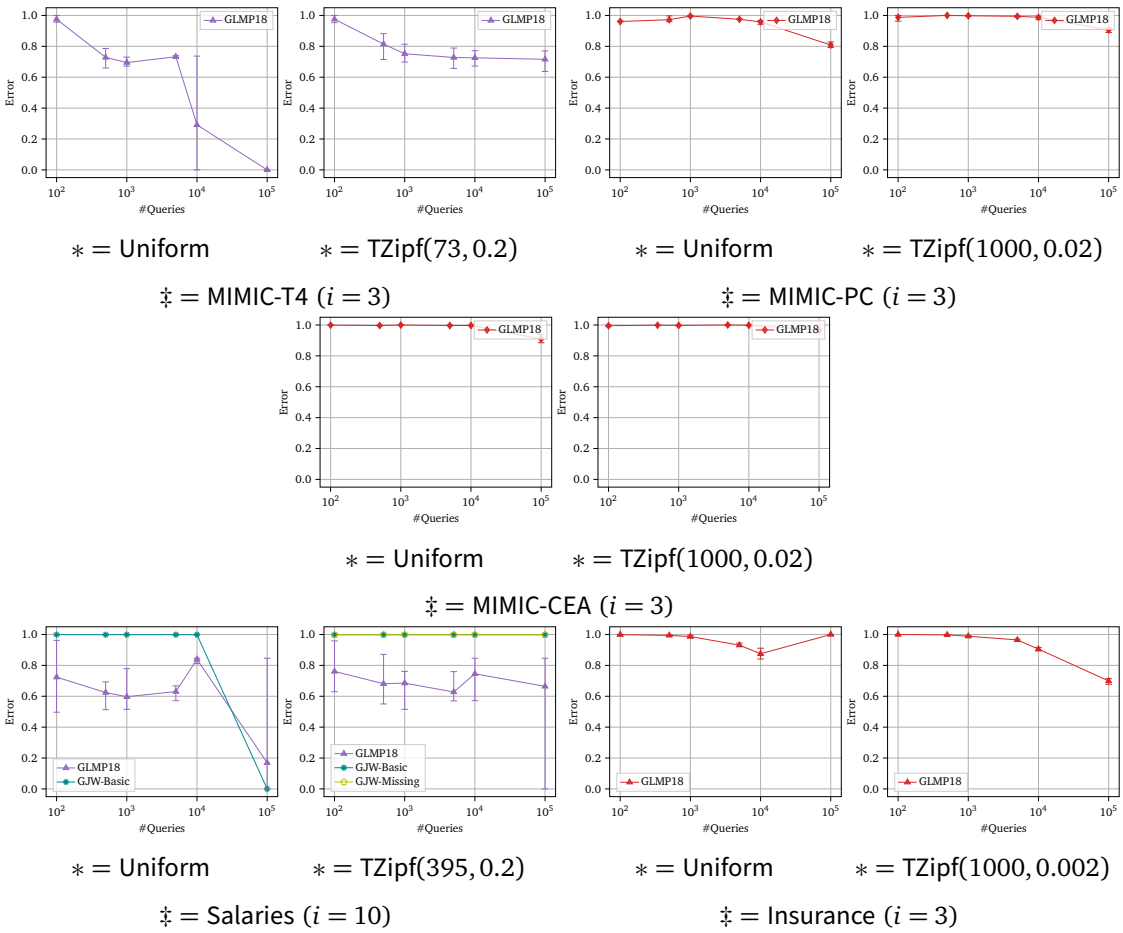


Figure 7: MCE of count reconstruction attacks on ‡ using a ∗ distribution. i displays how many iterations were performed.

of Section 4.4.2: so far, our real-world query logs can only be attacked if in addition to rid and qeq, the order is known to the adversary, e.g., if PPE is used. Otherwise, GENERALARR improves over [GLMP19] but still fails at reconstructing information we deem significant. One underlying reason is the identification of repetitions (cf. Section 7.1.3): with the approximated order, only a maximum mean of 8.27% of repetitions is identified for SDSS-L, whereas GENERALARR-OR identifies a maximum of 67% for SDSS-L. Note here that the size of the log increases this percentage, as it is only 0.08% and 2.48% for SDSS-S with the approximate or exact order, respectively. This is not visible in Table 4.5, because errors easily propagate due to recovery being based on uncovering distances between entries. Additionally, some repetitions of GENERALARR are false positives, but none are observed if the order is leaked. While ARR accomplished reconstruction agnostic to the query distribution, their reliance on APPROXORDER hinders performance, giving us an answer to question *2)*. We thus identify a *query distribution agnostic order reconstruction attack* to be important future work. in order to provide a first attack that performs well on real-world data.

4.4.2.6 Evaluation on Real-World Databases

Our value reconstruction evaluation on varying distributions and real-world databases is found in Figure 6. Though we also evaluated these attacks on the quite unrealistic uniform query distribution, we notice that depending on the parameters of TZipf, attacks uncover significant information. Therefore, we vary these parameters: For small databases, we assume that a considerable fraction of possible queries (20%) can occur, while this fraction is significantly lower for larger databases (2% for medium and large, 0.2% for very large databases). For each database, a rather small and a rather large window size limit is considered. We do not show any LMP results because the attacks consistently fail even for a uniform query distribution.

On the contrary, count reconstruction attacks fail to uncover significant information for TZipf, except if the window size limit is large. We thus display performance for the uniform query distribution and TZipf with a very large window size limit in Figure 7. For the uniform case, we evaluate GLMP18 and GJW19-BASIC[12], while for TZipf we also investigate GJW19-MISSING, given that it was designed for the case that queries are missing. We do not show the plots for MIMIC-PC, MIMIC-CEA, Sales, and Insurance, as no significant information is uncovered even for the uniform query space (MCE > 0.8 for GLMP18; aborts for GJW).

We conclude for the remaining dimensions of Section 4.4.2:

Query Distribution. As expected, attack performance can heavily depend on the query distribution. Value reconstruction attacks (cf. Figure 6) often uncover significant results for the more realistic restricted TZipf distribution. There, they work very well for a small fraction of possible queries and a large window size limit, though there are cases where even a small limit suffices (see 4) below). In particular, the attacks of [GLMP19] frequently decrease errors for larger window sizes, while GENERALARR-OR remains unaffected. The performance of GENERALARR-OR is only matched by GENERALARR for large windows. Noteworthy are GENERALIZEDKKNO and APPROXVALUE of [GLMP19], which are performant even though they were specifically designed for a uniform query distribution. Note, however, that for MIMIC, these attacks generally have a small error as they tend to reconstruct small values, but do not recover outliers (cf. *5)* below). Nonetheless, even for a restricted Zipfian distribution, these attacks can perform well if the window limit size is large. It remains open how realistic this setting is because the purpose of outsourcing a database is negated if a large fraction of it is queried. Therefore, risk needs to be determined on a case-by-case basis, possibly by using LEAKER.

For the count reconstruction attacks (cf. Figure 7), we notice a more extreme pattern: Significant information can only be uncovered under the unrealistic uniform distribution with small databases, with an exception for GLMP18 on Salaries and TZipf with a full window size limit. Recall that the MCE does not consider counts for values that do not appear in the

[12]Other attacks of [GJW19] are designed for specific cases of query behavior not suited for the uniform query distribution.

database, and that all these attacks failed at uncovering the counts for the actual values in these non-dense databases.

Amount of Queries. We observe that the amount of issued queries can matter. For value reconstruction attacks, the effect is mostly noticeable for GeneralARR: With more queries, our identification of repeating values becomes more accurate (mostly perfect for more than 10^4 queries) and significantly reduces the error. The [GLMP19] attacks display no such deviance. In count reconstructions, a small error is only possible for a large amount of queries.

Data Distribution and Size. The LMP attacks fail even under a uniform distribution due to the low density. Other value reconstruction attacks fare much better with this. However, ApproxValue assumes that specific entries exist in DB (at least one in $[0.2N, 0.3N]$ or its reflection) to find its anchor values. Though this is true for all databases, the fraction of such records is much lower for MIMIC-PC (0.03%) and MIMIC-CEA (0.4%) than the others. This is reflected by much worse and unpredictable performance of ApproxValue compared to GeneralizedKKNO in these databases.

Furthermore, we note major differences depending on the DB value distribution: MIMIC and Insurance are skewed towards low values occurring frequently with just a few high-value outliers, whereas entries in Salaries and Sales are spread more evenly across the domain range. The [GLMP19] attacks work well in the former cases even for small windows, whereas GeneralARR requires many queries to work in the former but not in the latter cases. This is especially noticeable for Sales, which has only one repeated value, i.e., a rather uniform frequency distribution. There, [GLMP19] fails completely while GeneralARR is accurate.

For value reconstruction, the database size does not have much influence, but GeneralARR has computational constraints for large n (MIMIC-T4) or N (Insurance).

The count reconstruction attacks fail to assign the counts to the correct values if the data is not dense. Otherwise, they can uncover the correct counts for some small databases but are not feasible for attacking larger database instances.

Leakage Profiles. Different to the keyword attacks, it becomes clear that significant leakage is needed to attack range databases in realistic settings. Attacks that have some successful recoveries in the more realistic TZipf setting require rid, and to achieve the more consistent accuracy of GeneralARR-OR, the leakage profile must also include qeq, order. The count reconstruction attacks relying solely on rlen do not work in a realistic setting.

5 MAPLE

As far as we know, the IHOP attack of Oya and Kerschbaum [OK22] 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 comparison, our attacks leverage stochastic techniques that can be applied in the same setting.* In this work [**KKM+24**], we also propose attacks that exploit the query equality of dependent query sequences but our techniques are different. In contrast to [OK22], 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. In contrast to [OK22], we model the observed query equality leakage on a dependent query sequence as the output of an HMM. Hence, we capture not only the query distribution but how it interacts with the qeq pattern as well, which allows us to use more sophisticated HMM inference techniques to recover information about the query sequence. As we demonstrate in our evaluations, *this can significantly outperform IHOP*.

The evaluation of [OK22] in the query-dependent setting uses both the sampled-data (the first half of the data source is test query data and the second half training auxiliary data) and the known-data (test and train data are equivalent) auxiliary knowledge. It does not use a query log but relies on the Wikipedia Clickstream dataset, which provides statistics of users' transitions between Wikipedia pages. These pages (and not the keywords filling them) correspond to keywords in their setting. In contrast, our evaluation uses real-world query logs that encompass queries from real keyword query systems. Furthermore, in their evaluation the keyword universes of the auxiliary attacker knowledge and the user's queries are equivalent [OK22]: First, a keyword universe of the 500 most well-connected keywords is created. Then, transition probabilities between these keywords are filled using the respective dataset (adversarial training or user test data). This ensures that each keyword the user queries for is in the attacker's knowledge. *We however use a different approach where the keyword universe stems directly from the auxiliary knowledge* given to the attacker, thereby not

[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.

ignoring keywords with a low connectivity and not ensuring equality of the keyword universe known to the attacker and that of the user.

5.1 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 [OK22] 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*, and leave extending our results to different types of processes as an interesting open question. 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 [Pea05].

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 5.1.1 (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 $i, j \in \{1, \cdots, \#S\}$,*

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

[2] Note that the impact of query dependency on the recovery rate of existing attacks is an unknown. In particular, revisiting existing attack with the assumption that queries are dependent (instead of independent) is non-trivial and would require a massive effort. We would leave it as an 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.

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.

The Markov property characterizes a dynamic stochastic process in which the future state depends only on the present state and not on any previous state. This is also called a first order Markov chain, which we focus on in this work.[4] The second property captures the fact that the probability to move from a state to another is independent of time and will be fixed in the entire process. There are many examples of Markov chains, like *random walks* [Pea05] or the *gambler's ruin* [Coo09].

Definition 5.1.2 (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 $n \geq 0$*

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

- *(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 $g : \mathbb{W} \to \{1, \cdots, \#\mathbb{W}\}$ 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] We can in general define an m-order Markov chain such that $\Pr[X_{n+1} = j \mid X_{n-m+1}, \cdots, X_n] = Pr[X_{n+1} = j \mid X_n]$.

5.2 Statistical Inference Attacks

5.2.1 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 5.3 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 5.2.1 (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 5.4), we were always able to generate the stationary distribution. We refer the reader to [Nor98] for more details. Note that π is invariant by the transition matrix T, i.e., $\pi = \pi \cdot \mathsf{T}$. Also, the existence and uniqueness of a stationary distribution depend on some properties of the Markov chain. In this work, we will be interested in the *irreducible chains*, a class of Markov chains for which the stationary distribution is unique if all states are recurrent positive.

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. We define it more formally below.

Definition 5.2.2 (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 5.2.3 (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 $\pi_i = \sum_{j \in [\#S]} \pi_j \cdot p_{j,i}$, i.e., π_i is ith value of the stationary distribution π.

The above lemma is an important component of our Stationary attack, and one could think of our attack as an analogue of frequency analysis. Our attack is detailed in Figure 1 and works at a high level as follows.

Attack overview. Stationary takes as input the query equality $\ell := \mathsf{qeq}(\mathbf{q})$ 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.[5] 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 $\alpha : [t] \to \mathbb{W}$. It then computes the stationary distribution π of the Markov chain's transition matrix, T, such that

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

[5]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.

- Stationary(ℓ, Θ):
  1. parse Θ as (T, μ) and ℓ as $(B_{i,j})_{i,j\in[t]}$;
  2. initialize a map $\alpha : [t] \to \mathbb{W}$;
  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 ρ that verifies
     $$\rho = \underset{j\in[\#\mathbb{W}]\setminus U}{\operatorname{argmin}} |v_i - \pi_j|;$$
     b) set $\alpha(i) := w_\rho$ and add ρ to U;
  7. output α.

Figure 1: Our attack: Stationary.

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 $\mathbf{v}$ 'Histogram' of all unique queries. That is, given the query equality $\ell := \mathsf{qeq}(\mathbf{q})$, the attack determines for every query the number of times the same query appears in the sequence. 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 5.2.3 introduced in Section 5.1, 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 and the error score comprising of the total distance between the average number of visits and the selected component of the stationary distribution.

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,[6]
- (*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, which requires $O(m^3)$ iterations where m is also the number of states composing the Markov chain.

Note. The Stationary attack is similar to other attacks like `Frequency-An` [NKW15] or `Att-Gen` [LZWT14]. 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.

5.2.2 The Decoder Attack

In this section, we describe our second attack, Decoder, which is a generic attack that can be instantiated in various ways depending on the underlying instantiation of the observation matrix. In this work, we consider two ways to instantiate the observation matrix and we refer to the resulting attacks as Decoder-N and Decoder-B. These two variants achieve different recovery rates depending on the shape of the auxiliary distribution as we will detail in Section 5.4.4.

Similar to Stationary, we will first start by describing the generic Decoder as a known-distribution attack and then follow with the description of the two variants but later in Section 5.3, 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). And as such, we were able to inherit many important results that have been made in the HMM area. For our attack specifically, we leverage the Viterbi algorithm [Vit67] – an algorithm that was first described in 1967 as a decoder for convolutional codes. Before we describe the attack we first recall what hidden Markov models are and how they are connected to our problem.

[6] Note that there are more efficient ways to calculate π. We are assuming that the underlying solver makes use of the LU decomposition [Sch95].

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 5.2.4 (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,*

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

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 [Vit67] 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

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

where $\mathbf{o}$ is the observed leakage and HMM is the hidden Markov model. We describe the Viterbi algorithm in Figure 7 of Section 7.5, 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 5.4.4. We take a top-down approach where we first describe the generic Decoder attack and then describe the two variants.

- $\mathsf{Decoder}(\boldsymbol{\ell}, \Theta)$:
  1. parse Θ as (T, μ) and $\boldsymbol{\ell}$ as $(B_{i,j})_{i,j\in[t]}$;
  2. initialize a map $\alpha : [t] \to \mathbb{W}$;
  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 HMM to $(\mathsf{T}, \mathsf{O}, \mu)$;
  8. set $\mathbf{o} := (o_1, \cdots, o_t)$ where for all $i \in [t]$,

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

  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 α.

Figure 2: Our attack: Decoder.

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 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.[7] 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 $\alpha : [t] \to \mathbb{W}$. 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.[8] 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

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

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

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 5.2.3 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.[9] 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 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 $\mathsf{O}_{i,j}$

[9] 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 π 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 π 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.

attains its maximum value if and only if $t \cdot \pi_i \approx t \cdot v_j$ which again shows the relationship to Lemma 5.2.3. 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.

5.3 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. [10] 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 $\mathsf{aux} = (q_1, \cdots, q_n)$. We also consider a more general setting where the auxiliary data is composed of multiple query sequences $\mathsf{aux} = (\mathbf{q_1}, \cdots, \mathbf{q_p})$ 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 Section 5.2.1 and 5.2.2, 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 [Wel03] 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

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

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

[10] 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(ℓ, aux):
  1. initialize an empty map $\alpha^\star$ and set $s^\star := \infty$;
  2. for all $\mathbf{q} \in \mathsf{aux}$,
     a) compute $\mathsf{HMM} \leftarrow \mathsf{Baum\text{-}Welch}(\mathbf{q})$;
     b) parse HMM as $(\mathsf{T}, \mathsf{O}, \mu)$ and set $\Theta := (\mathsf{T}, \mu)$;
     c) compute $(\alpha, s) \leftarrow \mathsf{Stationary}(\ell, \Theta)$;
     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.

the Baum-Welch algorithm in Figure 8. We describe the Stationary-Smpl and Decoder-Smpl attacks in Figure 5.

Attack overview. Both Stationary-Smpl and Decoder-Smpl take as inputs the query equality qeq and the auxiliary data aux, while the Decoder-Smpl takes an additional error parameter ϵ. 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 $|v_i - \pi_j|$.[11]
- 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 7 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 $\alpha^\star := \alpha$.

[11] 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.

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.

5.4 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 [**KKM+22**] and we compare the recovery rates of our attacks with the ones of the IHOP attack [OK22]—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.

5.4.1 Implementation

We implemented our attacks in Python 3.9 and added it as an extension to the open-source LEAKER [**KKM+22**] framework due its interoperability and modular design. LEAKER 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 integrating both point/keyword and range queries attacks, it only supports the independent query generation. As a result, we extended LEAKER to support dependent queries and attack evaluations in this setting by adding the respective backend implementations for a *dependent-query space* and *dependent-query evaluator*. The *dependent-query space* creates and populates the query space $\mathbb{W}$ using information from either a query log or an artificial distribution. The *dependent-query evaluator* evaluates a dependent-query attack on query sequence $\mathbf{q}$ drawn from a dependent-query space given

query equality leakage. 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 public repository [KKM+23] and as a branch in the open-source framework LEAKER [**KKM+22**].

5.4.2 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 [OK22] used the url links in Wikipedia pages to model client query distributions. We do believe that when it comes to attacks based on query equality pattern, it is crucial to assess their accuracy against realistic settings. Thus, the only data necessary to evaluate our attacks are query logs or artificial query distributions. Essentially a query log is a log file that records each client's query behavior, and it varies depending on the used system.

For our evaluation, we use the following two publicly available query logs from [**KKM+22**]:

- AOL is a publicly available search engine query log [PCT06] that contains web searches. AOL contains 52M queries that were issued by 656 thousand users between March 1^{st} and May 31^{st}, 2006. The total number of unique keywords is 2.9M.
- *The Arabidopsis Information Resource* [ECW+14], 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.[12]

- Uniform: for every $i, j \in [n]$, compute $\mathsf{T}_{i,j} \xleftarrow{\$} \{0,1\}^k$.
- Zipf: for every $i \in [n]$, we pick a permutation $\beta_i : [n] \to [n]$ at random. We then set $\mathsf{T}_{i,j} := f_{s,n}\big(\beta_i(j)\big)$ 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 $\beta_i : [n] \to [n]$ at random. And for all $j \in [n]$, sample a value $\theta_{i,j} \xleftarrow{\$} \mathsf{Bernoulli}(p)$ where $0 \leq p \leq 1$. If $\theta_{i,j} = 1$, then set $\mathsf{T}_{i,j} := f_{s,n}\big(\beta_i(j)\big)$, 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 $\theta_i \xleftarrow{\$} \mathsf{Zipf}(s,n)$, 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 $\beta_i : [n] \to [n]$ 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 $\mathsf{T}_{i,j_p} := f_{s,n}\big(\beta_i(j_p)\big)$ for all $p \in [\theta_i]$, and $\mathsf{T}_{i,j} := 0$ otherwise.

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 5.4.4 that varying the sparsity of the transition matrix significantly impacts the accuracy of our known-distribution attacks.

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

Scenarios	Leakage $\mathsf{qeq}(\mathbf{q})$		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 5.1: Summary of the evaluation setup for known-sample attacks to recover the queries of the ith user.

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.

5.4.3 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 5.4.2. 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 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 5.1 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 the fixed leakage setting, the adversary receives the query equality pattern of the ith user's query log sequence, $\mathbf{q}_i$. 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.

Note that because we do not consider *fixed leakage* for the *Exact* and *All-Users* scenarios, we never give the attacker the exact sequence subject to the query equality leakage. 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. In particular, in this setting we sample client's queries from a similar distribution as the one of the observed queries during the period of the compromise.
- 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. These queries are generated similar to above.
- 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. In particular, in this scenario, we split the query log into two parts, a first part given as auxiliary information to the adversary while the second part is used to generate the queries for which the adversary observes its query

equality pattern. This is a more realistic setting when compared to the Exact setting in the sense that the query log itself has been used instead of making assumptions on the query distribution.

- Other-Users (0): In this scenario, the adversary is not able to compromise the target user's machine but is able to compromise the other users' machines.

Our goal is to evaluate query equality attacks under assumptions of varying degrees of realisticness for both real world data collections and synthetic distributions. Thus, similarly to prior work, we have performed multiple evaluations on individual users based on different query scenarios and evaluation settings. The *adversarial scenarios* cover the different ways the adversarial knowledge is given. The *evaluation setting* covers the way the user's queries are modeled. We proceed by presenting our scenarios and settings before displaying the way our evaluations are executed. In Table 5.1, we present an overview of all possibilities of our evaluation setup.

Experimental setting. We cover several settings for generating the user's queries in our evaluations. It was noticed in prior work [OK22] that attacks may require a large amount of queries. Since our query logs do not contain that many queries per user, we opted for query sampling, i.e., executing a random walk over the transition matrix P from a random start state. The transition matrix P modeling the user's queries is learned via the Baum-Welch algorithm (cf. paragraph 7.4), where we ensure to remove from the input query sequence the keywords that result in end states, i.e., states in the tail of the query sequence that only appear once and therefore have no outgoing edges. This approach is what we call a *sampled* evaluation. For any evaluation on query logs, we will consider both the sampled and the *not-sampled* evaluation. The latter just means that we do not use the Baum-Welch algorithm and only attack the exact sequence of queries given by the query log. Note that an evaluation being sampled or not-sampled is independent of its given scenario. As artificial distributions are given via a transition matrix, we only consider sampled evaluations in these cases.

Experiment 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 $\mathsf{Zipf\text{-}Zipf}_{\nu}$ 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[13] . 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.

[13]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 [OK22].

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.[14] 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, i.e., *we do not specifically attack multi-user schemes*.

Data Source	Scenario	Max. Efficacy per Attack				Median Efficacy per Attack			
		IHOP [OK22]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl	IHOP [OK22]	Stationary-Smpl	Decoder-N-Smpl	Decoder-B-Smpl
TAIR [ECW+14]	All (A)	16.7%	10.5%	99.6%	**99.7%**	3.9%	7.5%	**99.2%**	98.6%
	Exact (E)	15.2%	10.0%	**99.6%**	3.9%	7.6%	**99.1%**		
	Other (O)	**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 [PCT06]	All (A)	90.8%	62.9%	**97.8%**	88.1%	62.1	38.3%	**96.3%**	75.0%
	Exact (E)	92.6%	62.8%	**97.0%**	62.4%	38.9%	**96.2%**		
	Other (O)	**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%

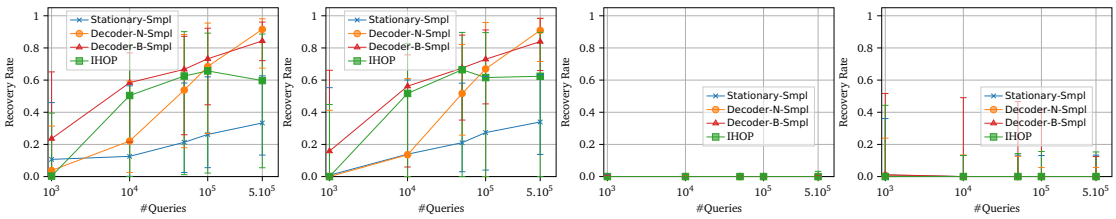
Table 5.2: Results summary of our attacks and the IHOP attack [OK22] in the *known-sample setting* with *fixed* leakage, highlighting the maximum and median recovery rates (in % of correctly recovered queries). The attacks are evaluated on the TAIR [ECW+14] and the AOL [PCT06] query logs. The adversarial scenarios for the attacks are described in paragraph 5.4.3 and paragraph 5.4.3.

5.4.4 Experimental Results

The recovery rate of our known-sample attacks with sampled and fixed leakage are given in Figure 6 and Table 5.2, respectively. The recovery rates of our known-distribution attacks are in Figures 7 and 10 in Section 7.6. 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.

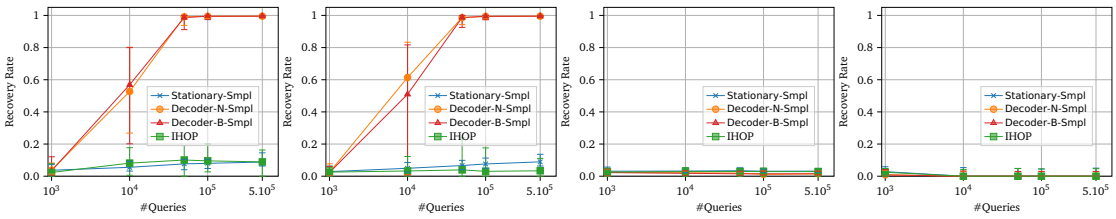
Known-sample attacks with sampled leakage. Our results with sampled leakage (cf. Figure 6) show that our known-sample attacks achieve their best recovery rates in the *All-Users* and *Exact* settings. In particular, in the latter, Decoder-N-Smpl achieves 99.1% recovery rate (cf. Table 5.2) with TAIR 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 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

[14] 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.



(E) (A) (0) (S)

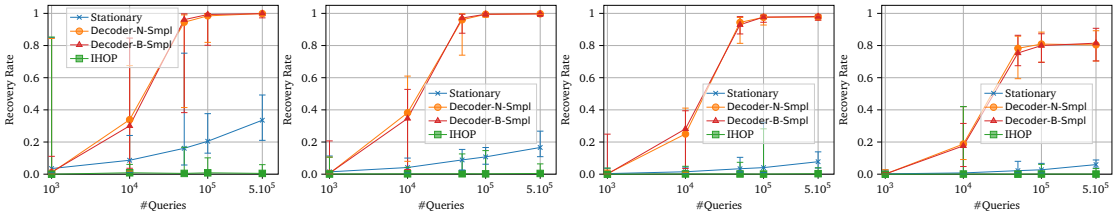
Evaluation for each of 5 *users* on AOL.



(E) (A) (0) (S)

Evaluation for each of 5 *users* on TAIR.

Figure 6: Our new attacks and the IHOP attack [OK22] 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

Evaluation for *Zipf-Zipf* Artificial distribution with a *fixed* hamming weight.

Figure 7: Our new attacks and the IHOP attack [OK22] evaluated on the Zipf-Zipf query distribution. All evaluations are done using a *fixed* minimum Hamming weight $= 2$.

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; and with AOL for query sequences of size larger than 10^5. 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; except on AOL and when the query sequence is medium sized, e.g. 10,000 queries. In this case, IHOP achieves better recovery rates.

Known-sample attacks with fixed leakage. Our results show that our attacks achieve their best recovery rates in the *All-Users* setting. In particular, Decoder-Smpl has a very high median recovery rate, 99.2% (cf. Table 5.2), with TAIR, and 96.3% with AOL. This significantly outperforms the IHOP attack [OK22], which had a median recovery rate of 3.9% with TAIR and of 62.1% with AOL. Our Stationary attack performs worse than the others in the *All-Users* setting, with a median recovery rate of 7.5% and 38.3% with TAIR and AOL, respectively. Recall that in this case, we only consider the *Split* and *Other* settings (cf. Table 5.2). 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. Overall, our Decoder-Smpl attack outperforms IHOP and Stationary in almost all settings and with all query logs and does particularly well with TAIR.[15]

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 9, 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 10 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.

[15] Note that in [OK22], IHOP was reported to achieve much higher recovery rates in the *Split* scenario. This might be due to the fact that [OK22] ensures that the query and auxiliary sequences have keywords from the same space. This is not enforced in our experiments in order to capture a more realistic setting.

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, [16] 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 longer 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 [PCT06], the most active user issued 815 queries and the average sequence length is 79. In TAIR [ECW+14], the most active user issued 2,059 queries and the average sequence length is 500. Our data hence indicates that, overwhelmingly, users in keyword search deployments such as AOL and TAIR may not produce a workload with sequences sufficiently long in order for the attacks to become successful. For the more realistic workloads of the AOL and TAIR data, the attacks are largely unsuccessful. 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, which has been highlighted by the successfull results of Decoder-B-Smpl attack. Thus, we conclude 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 [Kam15], 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, evaluations in this area have also used data from one source to model both the client and the attacker, leaving it open to find and evaluate data that can model the arguably more realistic case of auxiliary adversary information obtained from a distinct source.

Finally, 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,

[16] 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.

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.

6 Conclusion and Future Work

In this chapter, we conclude this thesis with a summary of our contributed works for Encrypted Search Algorithms (ESAs) cryptanalysis, offer a guideline on how to interpret our results, discuss the security implications, highlight the underlying limitations of inference attacks and finally wrap up with an outlook for potential future work.

6.1 Summary

Because much of *ESAs* research is theoretical, figuring out the impact of attacking a specific leakage profile in a practical real-world setting, i.e. large-scale data platform, presented a new open-challenge. Throughout our evaluations we depicted the different assumptions over data and query distribution usually made by prior works. Based on these assumptions, the attacks usually overlook the prohibitive computational cost while presenting their effectiveness. Thus, having a clear understanding of when attacks work and under which setting was important aspect of our work in order to evaluate their feasibility for large scale deployable solutions.

But, due to closed-source nature of previous implementations and constrained evaluations without real-world query data, a systematic re-evaluation effort of scale was severely hindered. With our open-source LEAKER framework, we enable the community to easily implement, independently evaluate, and compare current and future attacks. By systematically using it to re-evaluate the major leakage attacks for keyword and range queries with novel real-world queries and datasets.We uncovered sometimes unexpected settings where our evaluations could indicate that leakage attacks pose noteworthy risks (cf. Table 4.1 & Table 5.2). We summarize the implications of our work in the following paragraphs.

Keyword search attacks. The IKK [IKK12] and COUNT [CGPR15] attacks did not work as well on our datasets whereas the SUBGRAPH attacks of [BKM20] performed surprisingly well even in settings with low known-data rates, and on low-frequency queries on small private datasets. This contradicts previous intuitions that SUBGRAPH attacks might not work as well on real-world data [BKM20]. While some SUBGRAPH assumptions lower accuracy, it still poses a significant risk, especially if query equality can also be used to identify repeating queries. As a consequence, we view SUBGRAPH attacks as practical even for low-frequency keywords and therefore recommend schemes that hide the response identity and response length patterns in such settings (for example, by using some of the recent leakage suppression

techniques and constructions [KMO18; KM19; PPYY19; BKM20; GKL+20; GPPW20; APP+21; GKM21]). Compared to the other attacks, SUBGRAPH relies on *atomic* leakage of individual documents, and our evaluations strengthen the intuition that such leakage might be more risky. Our results also confirm the conclusions of [BKM20] that the VOLAN and SELVOLAN attacks (which exploit the total volume and response length patterns) could pose some risk for known-data rates $\geq 75\%$.

The risk of *sampled-query* attacks [LZWT14; OK21] remains open and still needs to be evaluated in real-world settings, e.g., by using and extending our LEAKER framework. Such high knowledge is realistic for public databases, indicating that private information retrieval is more appropriate in that case. Our evaluation tokenized queries and considered each word as a separate query. Investigating the case for *Boolean queries* [CJJ+13; KM17; LPS+18] and its additional leakage is a challenge we leave open.

Range search attacks. In contrast to keyword search, our evaluations uncovered many subtleties in the case of range search. As expected—since many range attacks are designed to work for specific query distributions—the only attack to succeed on all of our real-world datasets was ARR-OR which requires leakage of the response identity, the query equality and the order. Standard encrypted range schemes, however, do not leak the order [FJK+15; DPP+16].

If range queries have large width or if they are skewed towards the end points, we found that leaking the response identity is risky because the GENKKNO attack was successful. If, in addition, the query equality is also leaked then we found that the ARR attack was also successful. We found that leaking the response length and the query equality on evenly distributed data could be risky in light of the APA [KPT21] attack. However, our experiments showed that attacks that solely rely on the response length rarely worked on our datasets. This could also indicate an (intuitive) gap between different leakage types for range search, but more data and more evaluations are necessary to confirm this. For this, *leakage suppression* techniques [KMO18; KM19; PPYY19; DPPS20; GKL+20; GPPW20; APP+21; GKM21; SOPK21], which mitigate or eliminate various leakage patterns, could potentially be applied—at some computational and/or storage cost. These techniques already address different patterns so it remains open to integrate them into range schemes and also evaluate them under real-world queries.

Statistical Inference attacks. In order to highlight their effectiveness, attacks exploiting the query equality leakage pattern in the dependent-query setting, mainly opted for a set of evaluation settings that we deem highly unlikely to appear in real world setting in order to achieve a noteworthy recovery rate. Thus, for attacks to work given either query logs or artificial distributions, the attacker requires high levels of auxiliary information and a significant number of issued queries,i.e., the expected length of the query sequence for an attack to work is at least an order of magnitude greater than the highest amount of queries per user in both AOL and TAIR. Furthermore, requiring access to the clients exact query

distribution, making sure that the adversarial knowledge is a superset of the client's keyword space and picking only the most frequent keywords over a small keyword spaces is what we constitute to be unrealistic levels of auxiliary information in our evaluations. Conversely, when an attack is evaluated over these settings it would naturally skim over the underlying prohibitive computational cost, e.g., $O(m(m^2 + t))$ is the cost for IHOP attack [OK22] as well as our attacks. This huge computational burden inherently hinders all proposed query equality attacks in the literature to a variant degree.

Given the previously highlighted limitations, the goal of our work was not to only determine the scenarios where the attacks would work, but rather distinguish the nuance present in the more realistic evaluation settings, i.e., evaluations with query correlation scenarios where the users are more likely to issue dependent queries. Furthermore, we have been able to evaluate our attacks on relatively larger keyword spaces in comparison with IHOP [OK22] and achieve significantly better recovery rates throughout most of our evaluation (cf. Tables 5.1, Tables 5.2). Where our evaluations have shown that Decoder-Smpl in the *Exact* and *All-Users* settings and Decoder achieve the best recovery rates, while IHOP failed at recovering the threshold for both scenarios (i.e., the threshold is a subjective lower-bound that we set to 15% to denote the percentage of queries recovered for an attack to be successful). However, it is also fair to say that none of the attacks work in more realistic settings such as the *Split* and *Other* settings. Additionally, successful attacks need query sequences of a high length that is often not observed in the real-world query logs we used. 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. For a broader scope to model different attack instances, the evaluation would take many years to complete.

As previously highlighted, the efficacy of our new attacks and the IHOP attack[OK22] is significantly higher "only" when a considerable number of queries have been issued and the user's original query sequence is present among others in the auxiliary knowledge of the attacker.

6.2 Discussion & Guidlines

Guidance on how to interpret our results. We stress that the goal of our work is to better understand the state-of-the-art leakage attacks and not to provide a security analysis of various leakage profiles. In other words, the fact that an attack has low recovery rates on a given leakage profile under real-world queries and data says nothing about the security of that leakage profile. Understanding whether an attack improves on real-world data or does worse, however, is important for several reasons. First, since the recovery rates of leakage attacks often depend (strongly) on the query and data distributions, it is natural to ask how they perform on a variety of distributions and especially on distributions that capture real-world scenarios. Second, understanding how attacks perform on real-world data allows designers to

improve their intuition about their designs and whether there is enough of a *security margin* for practical deployment or whether they should switch to a scheme with a different leakage profile. What constitutes the right *security margin* here is, of course, subjective and different opinions are possible but making such evaluations available to the community is important. A third reason that evaluating state-of-the-art attacks on a variety of datasets is important is that it can help us to uncover new data and query characteristics that affect recovery rates.

Discussion of inference attacks 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.

Particularly, our results indicate that the success of the currently-known attacks exploiting the query equality pattern is not very realistic in the dependent-query case. For one, attacks require a lot of issued queries to work (often more than 10^4). However, especially for keyword search of end users, these amounts may not occur in many realistic cases[1].

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 [OK22] were only done on small values of m; 500 in [OK22] 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 [Wik22], the attacks would take over 2^{57} steps. Because of this, none of the experiments conducted in this work or in [OK22] tell us how the attacks would perform on query distributions over large keyword spaces.

Despite most evaluations being rather contrived, e.g., by ensuring the keyword universes of the user and of the attacker information are equal (like in [OK22]) or giving the attacker exact knowledge of private data, attacks should also not be ignored. For example, an ESAs instance with an extremely limited and publicly known keyword universe may be much more susceptible to our attacks than most other instances. An instance where an attacker may actually already know private data may also be much more susceptible than those where the attacker did not obtain such already significant information. Thus, our analyses underline the nuance of ESAs cryptanalysis and that when drawing conclusions about leakage attacks, many aspects require careful consideration for determining how concerning attacks are regarding a particular ESAs deployment.

[1] For instance, the highest amounts of queries per user are 8 334 and 5 410 for AOL [PCT06] and TAIR [ECW+14], respectively.

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.

6.3 Future directions

The risk posed by leakage attacks is hardly an abstract one, figuring out the impact of attacking a specific leakage profile in a practical real-world setting, presented multiple new open-challenges. Despite the considerable efforts made by prior works, there hasn't been significant comprehensive work investigating these attacks beyond their underlying leakage profile, and we believe that the conclusions that were drawn lack the nuance which could only be depicted by evaluating data under realistic assumptions.

In order to continually advance our common understanding of leakage implications in real world settings, we have conducted an extensive cryptanalytic re-evaluation of ESAs leakage, where we explored whether a given leakage pattern, can be realistically exploited under different assumptions over data and query distribution by an inference or a leakage abuse attack in the passive, persistent, known-data setting.

Throughout our re-evaluation efforts, we have first provided a categorization of the different attacks, leakage profiles leveraged by each, adversarial models and attack mode they rely on, target information to be recovered, and the type/percentage of auxiliary information assumed to be present. And based on the nuanced results we have uncovered, we believe that we have made huge strides towards having a clear understanding of when leakage attacks work and under which setting in order to evaluate their feasibility for large scale deployable ESAs solutions.

However, a lot of challenges remain until we can achieve a more holistic view of the implication of leakage under real-world settings. For that purpose, more studies of the effects of leakage and of leakage suppression technics are still needed. Concretely, evaluating sampled-query and sampled-data attacks [LZWT14; GLMP18; LMP18; GLMP19; GJW19; DHP21; OK21; OK22], as well as improving the overheads of leakage *suppression/mitigation* techniques such as [CLRZ18; KMO18; KM19; PPYY19; GKL+20; APP+21; GKM21; SOPK21].

Throughout our evaluations, we where restricted to one query log and similar artificial distributions on many datasets. Luckily, researchers with access to proprietary data can now use LEAKER for even tailored conclusions. Still, we hope that LEAKER will prove useful not only to the encrypted search community, but indispensable for more interdisciplinary research considerations, and will be extended and improved with new datasets and implementations of new and state-of-the-art attacks, so that cryptographic mechanisms can be used to their full potential.

7 Appendices

7.1 LEAKER Additional Empirical Evaluations

7.1.1 Evaluations for Most Active Users

We show further evaluations of the AOL and TAIR datasets in Figure 3 for the single user setting using five users with the highest activity. Note that the results are not significantly different to the least active users case in Figure 4.5 in Section 4.4.1. This confirms our statistical analysis that activity does not influence the frequency of queries (cf. Section 7.2.2). Further, we note an anomaly of a low amount of unique queries for AOL's most active users, resulting in equal query spaces for highest and lowest frequency.

7.1.2 Sampled-Data and Sampled-Query Attacks

We leave the evaluation of sampled-data and sampled-query attacks under real-world data as future work. Properly modeling real-world auxiliary data is challenging since the auxiliary data needs to come from a real-world distribution that is "close" to real-world query or data distribution. Collections of datasets with these properties are hard to find.

come additional challenges. For example, of modeling the sample comprising the adversary's knowledge, which is more straightforward for the known-data and known-query cases, where the adversary has direct access to parts of the original data. For instance, the original sampled-data evaluation of IKK in [IKK12] applied Gaussian noise to the original co-occurrence information to simulate attacker knowledge. Recent sampled-data attacks [DHP21; GPP21] divide the dataset into two disjoint parts in their evaluations, one of which is given to the adversary as knowledge while the other part is used as a target data collection. This provides important knowledge how well adversaries can attack schemes with disjoint knowledge. For an approach similar to ours incorporating data for real-world evaluations, we believe finding actual distinct datasets like, e.g., a public sample and a private data collection being targeted, would be interesting future work on this domain.

7.1.3 Agnostic Reconstruction Range

The agnostic reconstruction range attack (ARR) [KPT20] is incapable of reconstructing databases containing repeating values. To allow for evaluation on real-world data, we add pre- and post-treatments, which circumvent this issue and result in an attack that can recover repeating values. Our pre-treatment identifies repeating values and removes them from the ARR input. This has the side effect of reducing the input size, which in turn can result in runtime benefits. The post-treatment receives the output of ARR and then repeats the values which were previously identified to be repeating. We refer to this new attack as ARR if ApproxValue [GLMP19] is used to approximate order(**N**), and as ARR-OR if order(**N**) is leaked directly. They are shown in Alg. Algorithm 1. We note that this is not the only way ARR could be extended for repeating values and that future approaches might prove more fruitful. However, our approach already uncovers a significant amount of repetitions (cf. Section 4.4.2).

Our pre-treatment identifies repeating values based on the fact that the identifiers of repeating values always co-occur in all query responses. Concretely, we observe that the first (as determined by the ordering) identifier of these repeating values will never have the highest order in a query response. The converse holds true for the last identifier and all identifiers in between will be neither the highest nor the lowest. We use these observations to identify co-occurring tuples by determining the identifiers that never occur as minima or maxima in the query responses and then searching for candidate sequences that satisfy these constraints. Additionally, we require that the start of the candidate occurs as a minimum and that the end occurs as a maximum, which ensures that our candidates do not overlap.

An issue that arises from this approach are false positives, which could occur if so few queries are issued that unequal values always co-occur. To mitigate this, we require that these tuples appear in at least $minW$ distinct queries; this threshold can be set empirically.

7.1.4 ARR with Repeating Values

Agnostic Reconstruction Range (ARR) [KPT20] can be slightly modified in order to also cover the case of repeating values, i.e., a non-injective mapping from records to values [KPT20]. This can be achieved by allowing the distance between values to be 0 and requires a deviation from the original pseudocode of [KPT20] since the employed error function would be undefined if a distance of 0 was to be used. Concretely, for finding the distance $L_i = e_i - e_{i-1}$ between ordered data collection entries e_i and e_{i-1}, an error function E between pairs L_i, L_j, $j > i$ and the support size $\hat{L}_{i,j}$ estimating $L_{i,j} = L_i \cdot L_j$ is used for finding the minimum solutions L_i, L_j, thereby reconstructing the (ordered) data collection. The original error function $E_2(L_i, L_j) = \log(L_i)+\log(L_j)-\log(\hat{L}_{i,j})$ stems from a logarithmic transform of products into sums, allowing for an efficient representation of the optimization problem with a convex, linear function. However, this prevents a solution L_i to be 0, thus assuming no repeated values. the authors do not suggest that this function should be used for repeating values We therefore

Algorithm 1 ARR

Input: Query equality pattern $\mathsf{qeq}(\mathbf{N}, q_1, \ldots, q_t)$; Response identifier pattern $\mathsf{rid}(\mathbf{N}, q_1, \ldots, q_t)$; Ordering of the database entries $\mathsf{order}(\mathbf{N}) = (id_1, id_2 \ldots, id_n)$; Database universe $[N]$; Confidence threshold $minW$

Output: Approximate reconstruction $(\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_n)$

minima $\leftarrow set()$
maxima $\leftarrow set()$
for all unique $R \in \mathsf{rid}$ **do**
    minima.$add\left(min\left(\{\, i : id_i \in \mathsf{order} \wedge id_i \in R \,\}\right)\right)$
    maxima.$add\left(max\left(\{\, i : id_i \in \mathsf{order} \wedge id_i \in R \,\}\right)\right)$
end for
neverMin $\leftarrow [n] \setminus$ minima
neverMax $\leftarrow [n] \setminus$ maxima
cand $\leftarrow \{ \left(i, \ldots, i+j\right) : j \geq 1$
    $\wedge\, i \in$ neverMax $\setminus$ neverMin
    $\wedge\, i+1, \ldots, i+j-1 \in$ neverMin $\cap$ neverMax
    $\wedge\, i+j \in$ neverMin $\setminus$ neverMax $\}$
count$(C) \leftarrow |\{ R \in \mathsf{rid} : \forall i \in C : id_i \in R \}|$
dup $\leftarrow \{ C \in \mathsf{cand} :$ **count**$(C) > minW \}$
toRem $\leftarrow \{ (id_{i+1} \ldots, id_{i+j}) : (i, \ldots, i+j) \in \mathsf{dup} \}$
order$'$ $\leftarrow$ order without identifiers from $toRem$
rid$'$ $\leftarrow$ rid without elements from $toRem$
Ordered $\tilde{v}_1, \ldots, \tilde{v}_p \leftarrow \mathrm{ARR}(\mathsf{qeq}, \mathsf{rid}', \mathsf{order}', [N])$
$i \leftarrow 1$
for all $k \in [p]$ **do**
    if i starts a sequence C of dup **then**
        $j \leftarrow |C|$
        $\tilde{e}_i, \ldots, \tilde{e}_{i+j-1} \leftarrow \tilde{v}_k$
        $i \leftarrow i+j$
    else
        $\tilde{e}_i \leftarrow \tilde{v}_k$
        $i \leftarrow i+1$
    end if
end for
Return $\tilde{e}_1 \ldots, \tilde{e}_n$ permuted according to order

use $E_1(L_i, L_j) = (L_i \cdot L_j - \hat{L}_{i,j})$, which was introduced in [KPT20], as the error function to cover the more general and realistic case of repeated values occurring in the data collection. Additionally, the default value for lengths needs to be set to 0 rather than 1. Changing the error function also results in a new optimization problem, which is not convex in general.

As a result of the more complex optimization problem, we noticed increased runtimes and attacking our MIMIC instances became infeasible (cf. Section 4.4.2.4).

7.2 LEAKER Additional Data

7.2.1 Alternate Sources and Pre-Processing

Search engines. There are other query logs we could have used, including from the Excite [Jan06], Yandex [Yan14], and Sogou [Sog08; LMZ+11] search engines or from Yahoo! Answers [Yah16] or the TREC session track [NIS14]. We preferred the AOL dataset, however, due to its large size and the fact that it comes divided by user.

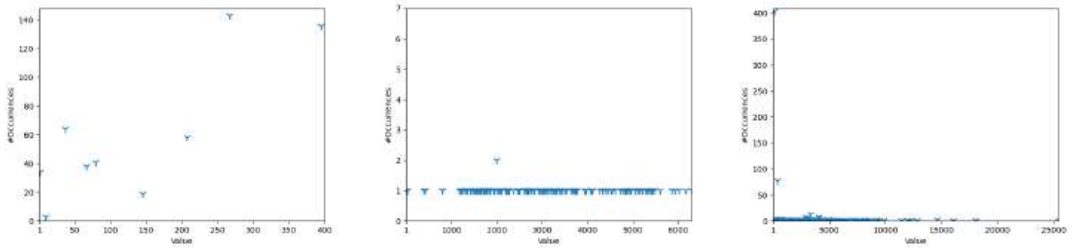
The AOL query log contains queries issued between March 1st and May 31st, 2006. The query text and query time are stored in plain; the client id is a pseudonym. At the latest when journalists prominently identified a client contained in the log [BZ06], it became clear that the release of pseudonymized query logs is a severe violation of privacy and, as a result, few logs have subsequently been released, with the AOL log remaining heavily used as a basis for query analysis. We discarded the 1 000 most active users from the data because they appeared to be bots. Our processed log AOL yields 52 287 049 queries (2 862 476 unique keywords) by 656 038 users. Also, 67 383 of the 2.9M keywords of the AOL query log can be found in the data collection which provides us with a large dataset.

Genetic. The TAIR query log contains all queries issued between January 1st, 2012 and April 30th, 2013 and is associated with user *sessions*. To obtain the TAIR data collection state at the time of the queries, we use the *Araport11* release [Pho11] together with the TAIR 2013 update data [Pho13]. Out of the 650k queries, 5 272 can be found in the collection, giving us a sufficiently large dataset[1].

Scientific data. We also identified SQLShare [JMH+16] as a potential dataset. It contains a range of scientific measurements by physicists, biologists and social scientists. However, after integrating it and analyzing the query logs we found very few range queries; a total of 12. We therefore discarded this dataset, though it could prove useful in the future if more relevant queries are added.

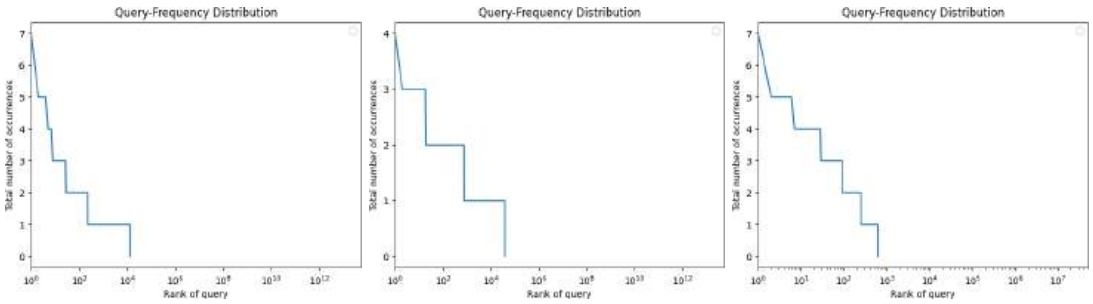
Census data. The SPARTA project [MIT15] includes data and SQL query generation based on census data. We did not use this in our work, as the queries are generated randomly to fit desired response lengths, but we believe this could potentially be useful in other evaluation scenarios that require the generation of relational queries.

[1] We attribute the low size of the intersection between query and data to the missing publications and people datasets, which have not been released for download.



Salaries Sales Insurance

Figure 1: Frequencies of numerical dataset values.



SDSS-M SDSS-L Zipf distribution ($a = 5$) restricted to 0.002% of all queries

Figure 2: Query frequency distribution of SDSS query logs on the PhotoObjAll collection (here scaled by $\times 10^5$; $N = 10\,455\,488$) as well as an artificial Zipf distribution on a random collection with $N = 10^4$.

7.2.2 Statistical Analysis

We used LEAKER to analyze the datasets of Section 4.3 and describe relevant statistical insights here.

7.2.2.1 Data Distributions

Fig. 1 shows the frequencies of values for the datasets that we can display without violating access restrictions. Notice that Insurance has a significant skew towards low values, with values greater than 10 000 out of a maximum of $N = 25\,425$ being very rare outliers (8 out of 886). We note that this is also the case in all MIMIC data, in that most entries have a low value: for MIMIC-T4, only 301 out of 8 058 entries have a value greater than 20 out of a maximum of $N = 73$; for MIMIC-PC only 9 out of 7 709 entries have value greater than 500 out of a maximum of $N = 2\,684$; and for MIMIC-CEA only 25 out of 2 844 entries have value greater than 2 500 out of a maximum of $N = 9\,978$.

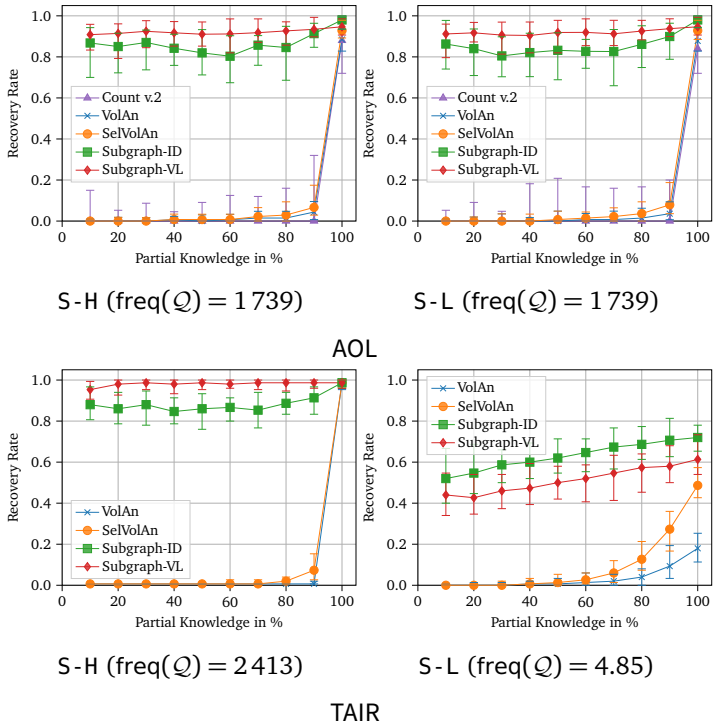


Figure 3: X-Y attack evaluations against AOL and TAIR. All evaluations are 5×5, except for 3×3 evaluations done for COUNT v.2. 150 queries are drawn for each of the most active users (single user setting; S) without replacement according to their query frequency from the 500 most (H) or least (L) frequent queries in the query log that are also contained in the partial knowledge, respectively. The resulting mean frequency is given by $\text{freq}(\mathcal{Q})$.

In contrast, this is clearly not the case for Salaries and Sales, where no such skew is noticeable in Figure 1. Since we notice that different attacks behave very differently according to whether this skew exists (cf. Section 4.4), we call the former case with the skew an *uneven* data distribution in our potential general risk factors (cf. Section 4.1), and consequently we denote the latter case as an *even* data distribution.

7.2.2.2 Keyword data – Frequency distribution

The frequency of queries has been identified as the main attack performance metric [BKM20; RPH21] and, therefore, we analyzed the frequency distribution of queries issued in real-world systems. In particular, [BKM20] already noted that keyword data usually follows the Pareto principle, i.e., the probability mass function is *heavy-tailed* with the bulk of the keywords appearing in a few documents. This was used by [BKM20] to argue that, because most keywords appear in the tail with a low occurrence, most queries might have a very low frequency as well. A similar argument for low-frequency keywords in real-world queries can be found in [RPH21]. Being able to see which keywords are queried in real systems for the first time, we noted that this is not the case for any of our data sources: The queries are usually not from the tail of the data distribution and have a high frequency (a mean of 1 804 for AOL, 2 023 for TAIR and 326 for GMail). [2] Only Drive has a mean query frequency of 11.2, which was considered as *pseudo-low* by [BKM20]. We conclude that, in our evaluations, *users are not interested in querying keywords of a low frequency* and conjecture that they may rely on the system's ranking to obtain the desired results.

Additionally, we investigated if the activity of a user has an effect on their queries' frequencies, but found no correlation between number of queries and mean frequency (Pearson correlation of about 0.1 for TAIR and 0.014 for AOL).

7.2.2.3 Range data – Query distribution

The core of range attack analysis has been the query distribution. The heavily-used uniform distribution is an unlikely case in the real world, and while specific parametrizations of the beta (family) distribution were considered [KPT20; KPT21] and already provide important insights, these do not have any empirical basis. Using real query logs (cf. paragraph 4.3.2), we investigated two major factors of query distributions: query frequencies and their widths.

We plot frequencies of *all possible* queries for SDSS-M and SDSS-L in Figure 2 as well as a comparable Zipf distribution. The case of SDSS-S does not need to be plotted, as queries only appear once or twice. The Zipf distribution has a probability mass function of

$$p(k) = \frac{k^{-a}}{\zeta(a)},$$

[2] When analyzing AOL here, we restrict ourselves to the 25 000 most active users for the sake of feasibility.

where ζ is the Riemann Zeta function and a is the shape parameter. This means that an element's frequency is inversely proportional to its rank among all elements according to decreasing frequency. From Figure 2, we deduced that queries are roughly sampled according to a Zipf distribution, with a high shape parameter (higher ranks are very unlikely) but this only holds for a *tiny fraction of queries*.

We also looked at the query widths and found that they are fixed: SDSS-S either has a width of 113 or 112, while sizes range between 161 and 173 for SDSS-M and 51 and 61 for SDSS-L.

Based on these empirical observations, we considered a new kind of query distribution in our experiments for data without query logs, where queries are distributed according to Zipf, but, in contrast to [KPT20; KPT21], they are restricted to specific widths and a fraction of possible queries before the probabilities are assigned. However, since this analysis was confined to one specific case, it might not be representative. While we consider a large fraction of possible queries missing as intuitive, the fixed-sized widths might be unique to SDSS and we expect more variable widths in other cases. We thus varied the *upper bound* of widths in our experiments. For some attacks, a large upper bound (close to N) has been identified as a risk factor (cf. Section 4.4).

7.2.3 Additional Real-World Data

We also consider the following data where the queries are not available, but the logs contain relevant information.

Public Database Search. Our processed PubMed query log [Nat09] contains all 58 026 194 search queries issued during 1 034 098 client sessions in March 2008. Originally, it was used to investigate search trends [IMNL09]. Client session ids are pseudonyms and the query keywords are not contained in the log, which makes a direct attack evaluation impossible. However, the amount of query results, i.e., the selectivity, is stored for each query and we can therefore analyze the selectivities and, hence, susceptibility to leakage attacks. We use PubMed to model a *public medical search use case*, where the underlying data may be public, but the queries are of a very sensitive nature.

Private Database Search. Search behavior for specialized private databases is publicly available in the form of the Pocketdata query log [KACZ15], which contains about 45 Million SQLite statements issued to smartphones of eleven participants of the University of Buffalo gathered over multiple days up to one month of regular smartphone usage. It was gathered to produce a test query set for mobile database use cases. We use its 33 301 694 SELECT queries as basis for our Pocketdata query log and view each database of each client as a separate database. With this, we obtain a query log over 4 614 databases. Though the query values are hidden in this log, the result lengths, i.e., the selectivities, are recorded. We therefore cannot run attacks on this log, but we can analyze the selectivities and infer how successful

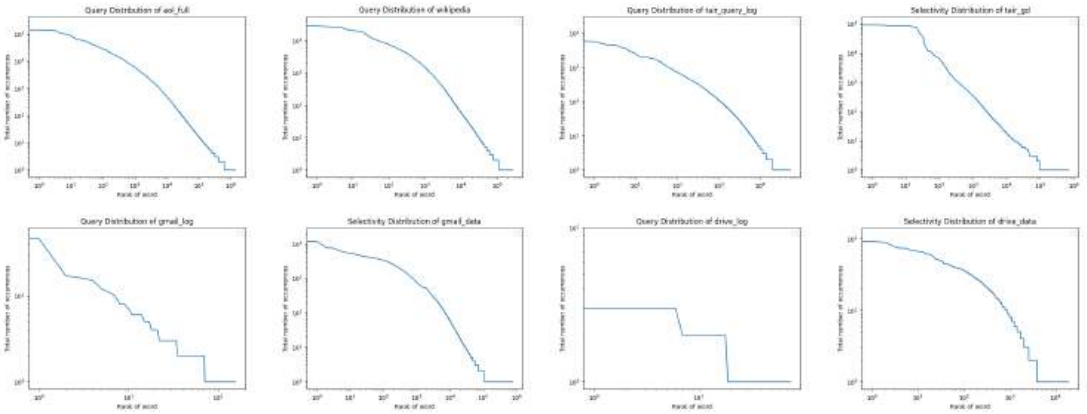


Figure 4: Query and keyword distributions (log-log scale) of query logs and corresponding databases for AOL, TAIR, GMail-L, and Drive. The queries are aggregated over all users.

attacks would be. We use Pocketdata to model SQLite *queries on private specialized databases*, being automatically generated based on client behavior.

7.2.4 Statistical Analysis of Real-World Data

In particular, we are mostly interested in the selectivity distribution of keyword queries and the query distribution of range queries. This type of analysis will show us how clients perform queries and indicate how realistic the assumptions of prior work were. Based on this, we will be able to estimate the general success of leakage attacks before showing the performance of specific attacks in concrete evaluations in Section 4.4. Due to space issues, we omit plots and summarize the results.

7.2.4.1 Keyword Data

For keyword data, the selectivity of the queries has been identified as the main performance metric [BKM20]. We therefore analyze the selectivity of queries issued in real-world systems. Additionally, we investigate if the activity of a client has an effect on their queries' selectivities. When analyzing AOL here, we restrict ourselves to the 25 000 most active clients for the sake of feasibility.

Looking at all our databases and query logs, we find that the individual keyword distributions roughly follow the Pareto principle, i.e., they are *heavy-tailed* with the bulk of the keywords appearing in a few documents, and some keywords appearing in many documents.

We observe that, contrary to what has been done in previous work, one cannot make a claim about the selectivity of queries by just observing the database distribution or the

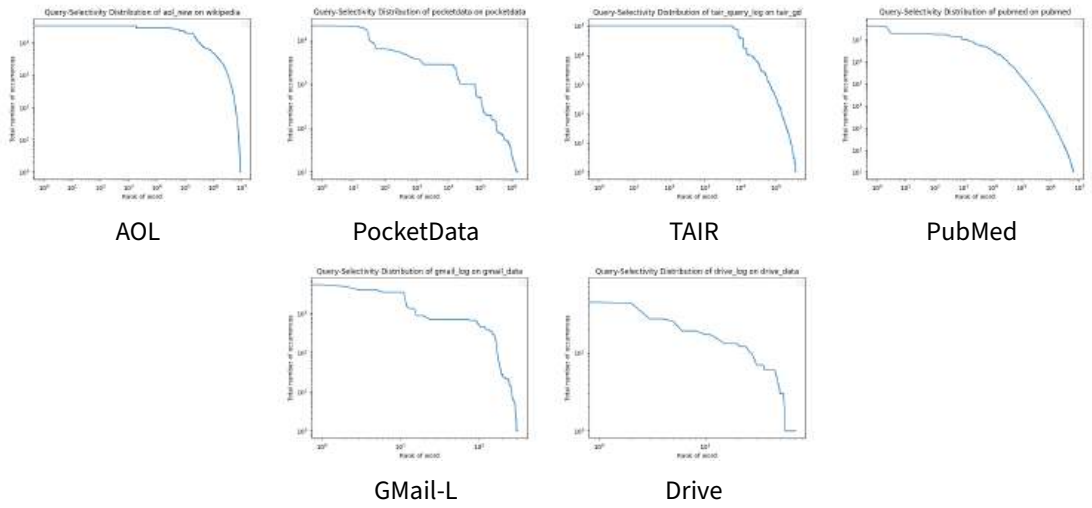


Figure 5: Query log query selectivity distributions (log-log scale).

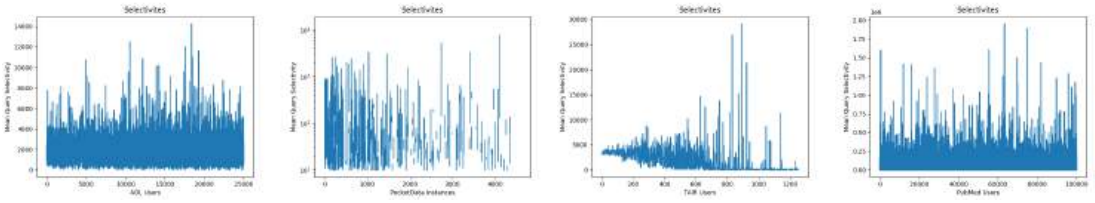
query distribution. This is because across all evaluated data, we find that there is *no linear correlation* between a query's frequency and its selectivity in the database. For instance, the mean Pearson correlation coefficient between query frequency and its selectivity of all TAIR clients is just about 0.1, while the one for the AOL clients is 0.014.

What matters instead in a real system is the selectivity of the issued queries, i.e., the combined query-selectivity distribution, which we can compute even for the Pocketdata and PubMed query logs, where no keywords are available but the corresponding selectivities are stored. Our results indicate that in the evaluated systems, clients' queries generally *have a very high selectivity* for both public and private data. The rounded mean selectivities over the multi-set of all queries issued by all clients/instances are 1 804 for AOL, 176 for Pocketdata, 2 023 for TAIR, and 24 198 for PubMed. This is also the case for the queries issued on private GMail data, with the mean of the mean selectivities being 326. Notably, GMail-S has a significantly lower mean selectivity of 61.7, which we still consider high because attacks perform relatively well on it (cf. Section 4.4.1). The mean selectivity is low (11.2) only for Drive.

Furthermore, we investigate if a client's activity influences their behavior with regards to leakage attacks. Frequent clients might produce a lot of high-selectivity queries, whereas occasional clients might be interested in more specific information. In contrast to that intuition, we discover that *a client's activity does not influence the selectivity of their queries*. We cannot come to a conclusion for private data due to insufficient client activity information.

7.2.4.2 Range Data

The central part in evaluating the effectiveness of leakage attacks on range databases has been the query distribution. Until now, range queries have only been modeled very artificially,



AOL PocketData TAIR PubMed

Figure 6: Mean query selectivity of all AOL, Pocketdata, and TAIR users, and the 100 000 least active PubMed users/instances (sorted by descending activity).

mostly via uniformly distributed queries [KKNO16; LMP18; GLMP19; KPT20] or distributions that result in all possible elementary volumes [GLMP18; GJW19]. Since a uniform distribution is considered to be an unlikely real-world distribution, specific Zipf-like distributions were considered in [KPT20]. It is important to note that all these distributions eventually provide all possible queries or all elementary queries. The only exception is [GJW19], which allows for a limited amount of missing queries within certain query windows. However, all distributions were created by intuitions about possible user behavior and do not have any empirical basis. Using the SDSS query logs (cf. paragraph 4.3.2), we are the first to take a look at two major factors of real range query distributions: query frequencies and their window sizes.

We plot the frequencies of *all possible* queries for SDSS-M and SDSS-L in Figure 2 as well as a comparable artificial distribution. Here, we scale by $\times 10^5$ to give more precision in the analysis. While this still incurs a precision loss in the database, it does not lose precision in the issued queries. The distribution for SDSS-S does not need to be plotted: only a fraction of 1.464×10^{-13} of possible queries occur twice, while a fraction of 2.593×10^{-11} of possible queries occur once. Figure 2 also displays an artificial Zipf distribution with the probability mass function

$$p(k) = \frac{k^{-a}}{\zeta(a)},$$

where ζ is the Riemann Zeta function and a is the shape parameter. From the distributions of SDSS-M and SDSS-L, we deduce that queries are roughly sampled according to a Zipf distribution with a high shape parameter (higher ranks are very unlikely), but this only holds for a *tiny fraction of queries*. In the smaller SDSS-S log, queries are mostly unique, but most possible queries do not occur at all as well.

We also look at the distribution of windows and find that, across our SDSS logs, the window sizes are relatively fixed: 10.6% of the multiset of SDSS-S queries have a window of 113 and the remaining queries have a window of 112. For SDSS-M, windows range between 161 and 173, with 54.9% being 173 and 27.1% being 161. In SDSS-L, the windows are between 51 and 61, with 29.5% being 51 and 23.4% being 61.

7.3 LEAKER Code Snippet

We provide example LEAKER implementations in Listings 7.1 and Listing 7.2.

Listing 7.1: Slightly simplified example of LEAKER code for implementing the basic count attack (Algorithm 1 of [CGPR15]) using co leakage.

```
class BasicCount(KeywordAttack):
    def __init__(self, known_data_collection):
    # Set up self._known_keywords the set of known keywords, self._known_coocc the known
         co-occurence matrix, and _known_unique_rlens mapping unique rlens to known
        keywords.

    @classmethod
    def required_leakage(cls):
        return [CoOccurrence()]

    def _known_response_length(self, keyword):
        #rlen is the diagonal of co matrix
        return self._known_coocc.co_occurrence(keyword, keyword)

    def __initialize_known_queries(self, queries, rlens):
        return {i: self._known_unique_rlens[rlens[i]] for i, _ in enumerate(queries) if
            rlens[i] in self._known_unique_rlens}

    def recover(self, data_collection, queries):
        coocc = self.required_leakage()[0](data_collection, queries)
        rlens = [coocc[i][i] for i, _ in enumerate(queries)]

        known_queries = self.__initialize_known_queries(queries, rlens)

        while True:
            unknown_queries = [i for i, _ in enumerate(queries) if i not in known_queries]
            old_size = len(known_queries)
            for i in unknown_queries:
                candidate_keywords = [k for k in self._known_keywords if k not in
                    known_queries.values() and rlens[i] == self._known_response_length(k)]
                for s in candidate_keywords[:]:
                    for j, k in known_queries.items():
                        if coocc[i][j] != self._known_coocc.co_occurrence(s, k):
                            candidate_keywords.remove(s)
                            break
                if len(candidate_keywords) == 1:
                    known_queries[i] = candidate_keywords[0]
                    if old_size >= len(known_queries):
                        break

        uncovered = []
        for i, _ in enumerate(queries):
            if i in known_queries:
                uncovered.append(known_queries[i])
            else:
                uncovered.append("")
```

```
43
44        return uncovered
```

Listing 7.2: Simple example of LEAKER code for implementing the co pattern (without pre-computation and caching).

```
1 class CoOccurrence(LeakagePattern):
2     def leak(self, data_collection, queries):
3         doc_ids = {q: map(lambda doc: doc.id(), data_collection(q)) for q in queries}
4         return [[len([i for i in doc_ids[qp] if i in doc_ids[q]]) for qp in queries] for
              q in queries]
```

7.4 Markov Models

In our attacks that we present in (Sections 5.2.1, 5.2.2), we are interested in a setting where the states of a Markov chain cannot be observed, i.e., are hidden, but the outcomes of another process can be observed that is influenced by the hidden process. This describes a *hidden Markov model* (HMM), which is a doubly embedded stochastic process where the first process is hidden while the second one is observable. We provide a formal definition below.

Definition 7.4.1 (Stochastic Process)**.** *A stochastic process $\mathcal{X}$ on a countable set $S = \{S_1, \cdots, S_N\}$ is a collection of random variables with values in S defined on a probability space $(\Omega, \mathcal{F}, \Pr)$ such that*

$$\mathcal{X} = \{X_n \ : \ n \geq 0\},$$

where Pr *is the probability measure, $\mathcal{F}$ a family of events and Ω an event space.*

The set S represents the *state space* of the stochastic process and X_n the random variable that denotes the state of the process at step n.

Definition 7.4.2 (Finite-Dimensional)**.** *Given a stochastic process $\mathcal{X} = \{X_n \ : \ n \geq 0\}$ on a countable set $S = \{S_1, \cdots, S_N\}$, the finite dimensional of $\mathcal{X}$ is*

$$\Pr[X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n],$$

for all $i_0, i_1, \cdots, i_n \in S$ and for all $n \geq 0$.

The finite-dimensional is similar to what a probability mass function is for a distribution; it characterizes a stochastic process. All different stochastic processes will include some conditioning between its random variable, as it is the case for the Markov chain that we are going to define below.

Definition 7.4.3 (Hidden Markov Model (HMM))**.** *A hidden Markov model (HMM)* $\mathcal{M}$ *is composed of two processes* $\mathcal{X} = \{X_n \in [\#S] \ : \ n \geq 0\}$ *over a countable set* S *and* $\mathcal{Y} = \{Y_n \in [\#T] \ : \ n \geq 0\}$ *over a countable set* T *such that:*

- $\mathcal{X}$ *is a Markov chain parametrized by* $\Xi = (\mathsf{P}, \mu)$*, and*
- $\Pr[Y_n = i \mid X_n = j] = o_{i,j}$*, for all* $i \in [\#S]$ *and* $j \in [\#T]$*.*

Let $\mathsf{O} = (o_{i,j})_{\substack{i \in [\#S] \\ j \in [\#T]}}$ denote the observation probability, i.e., the probability of observing the jth state of T from the ith hidden state of S. For simplicity, we write $\Theta = (\mathsf{T}, \mathsf{O}, \mu)$ to denote a HMM $\mathcal{M}$ parameterized by T, O, and μ.

Inference. In the setting of a HMM, given the observable information and the HMM parameters, information about the hidden process can be inferred. One instance is the Viterbi algorithm [Vit67]. We detail Viterbi in Section 7.5. In short, given an HMM parameterized by $\Theta = (\mathsf{T}, \mathsf{O}, \mu)$, this dynamic programming algorithm determines the most likely explanation of the sequence of hidden states of $\mathcal{X}$ up to a number of steps t that produced an observed sequence of states $\mathbf{q} = (q_1, \cdots, q_t)$ of the observable process $\mathcal{Y}$.

Learning. Conversely, another set of tasks associated with HMMs concern learning the parameters. We consider the well-known Baum-Welch algorithm [Wel03] for this that finds the HMM parameters that maximize the probability of making a given observation of a sequence of states $\mathbf{q} = (q_1, \cdots, q_t)$ of the observable process $\mathcal{Y}$. Particularly, we will use this to obtain the transition matrix T for a given sequence of states. We present the algorithm in Figure 8 of Section 7.5.

7.5 Markov Algorithms

We present the Viterbi algorithm in Figure 7 and the Baum-Welch algorithm in Figure 8. The Viterbi algorithm [Vit67] 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 [Wel03] 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($\mathsf{HMM}, \mathbf{o}$):
  1. parse $\mathbf{o} = (o_1, \cdots, o_t)$ and $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$;
  2. instantiate two zero-matrices δ and ϕ of size $t \times n$ where n is the number of states in T;
  3. for all $i \in [n]$, compute
  $$\delta_{1,i} = \mu_i \cdot \mathsf{O}_{i,o_1} \quad \text{and} \quad \phi_{1,i} = 0;$$
  4. for all $j \in \{2, \cdots, t\}$ and $i \in [n]$, compute
  $$\delta_{j,i} = \max_{k \in [n]} \left(\mathsf{T}_{k,i} \cdot \delta_{j-1,k} \right) \cdot \mathsf{O}_{i,o_j} \quad \text{and} \quad \phi_{1,i} = \operatorname{argmax}_{k \in [n]} \left(\mathsf{T}_{k,i} \cdot \delta_{j-1,k} \right);$$
  5. set
  $$r_t := \operatorname{argmax}_{k \in [n]} (\delta_{t,k});$$
  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 7: The Viterbi algorithm [Vit67].

- Baum-Welch($\mathbf{o}$):
  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

$$\alpha_{i,1} = \mu_i \cdot \mathsf{O}_{1,i} \quad \text{and} \quad \alpha_{i,t'+1} = \mathsf{O}_{o_{t'+1},i} \cdot \sum_{j=1}^{n} \left(\alpha_{j,t'} \cdot \mathsf{T}_{j,i}\right)$$

     b) for all $i \in [n]$ and $t' \leq t$, compute

$$\beta_{i,t} = 1 \quad \text{and} \quad \beta_{i,t'} = \sum_{j=1}^{n} \beta_{j,t'+1} \cdot \mathsf{T}_{i,j} \cdot \mathsf{O}_{(t'+1),j}$$

     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 $i, j \in [n]$,

$$\mu_i = \gamma_{i,1} \quad \text{and} \quad \mathsf{T}_{i,j} = \left(\sum_{t'=1}^{t-1} \xi_{i,j,t'}\right) \cdot \left(\sum_{t'=1}^{t-1} \gamma_{i,t'}\right)^{-1} \quad \text{and} \quad \mathsf{O}_{i,j} = \left(\sum_{t'=1}^{t} \mathbf{1}_{(o_{t'}=j)} \cdot \gamma_{i,t'}\right) \cdot \left(\sum_{t'=1}^{t} \gamma_{i,t'}\right)^{-1}$$

     e) increment count;
  4. output $\mathsf{HMM} = (\mathsf{T}, \mathsf{O}, \mu)$.

Figure 8: The Baum-Welch algorithm [Wel03].

7.6 Markov Additional Empirical Evaluations

We present in Figure 9 the evaluation result when we vary the Hamming weight and fix the number of states to 500. We present in Figure 10 the evaluation results of our known-distribution attacks when the user queries are sampled from the Binomial-Zipf, Uniform and Zipf distributions.

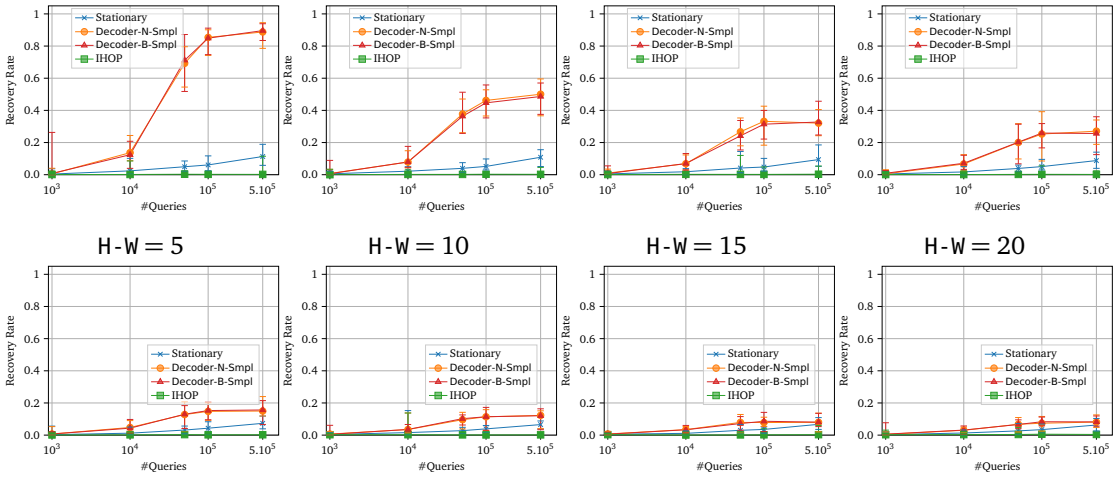
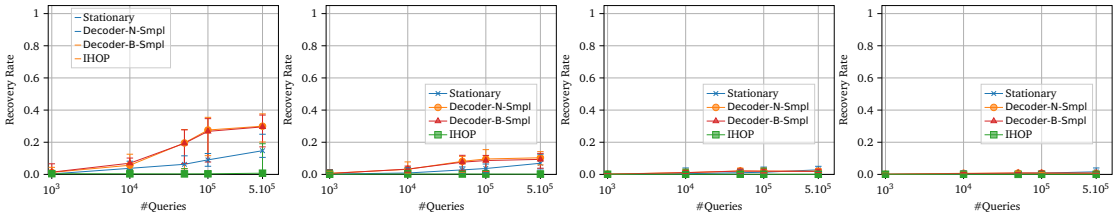
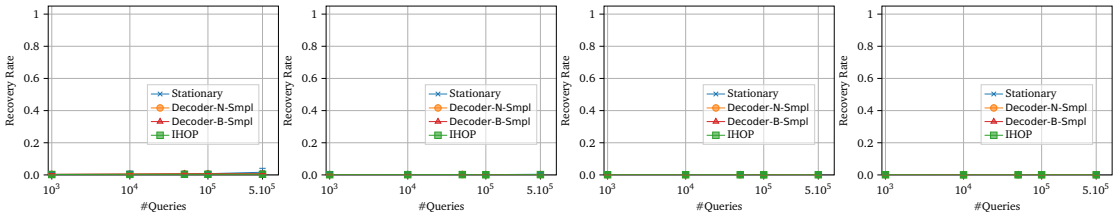


Figure 9: Our new attacks and the IHOP attack [OK22] evaluated on the Zipf-Zipf distribution setting. Evaluations are done using a *variable* Hamming weight and a *fixed* number of states = 500.



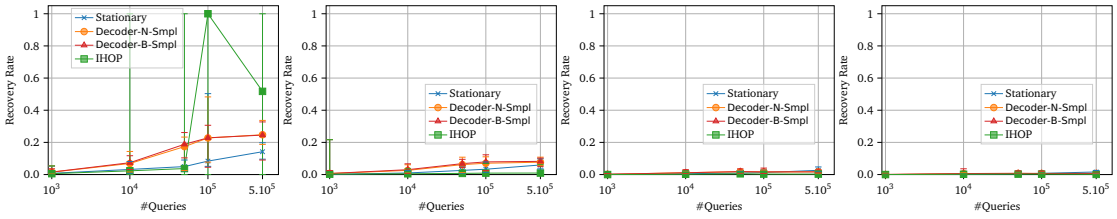
250 States 500 States 1 000 States 1 500 States

Evaluation on the Binomial-Zipf distribution.



250 States 500 States 1 000 States 1 500 States

Evaluation on the Uniform distribution.



250 States 500 States 1 000 States 1 500 States

Evaluation on the Zipf distribution.

Figure 10: Our new attacks and the IHOP attack [OK22] evaluated in the known-distribution setting. The results represent the average recovery rate over 30 runs.

Bibliography

[AAGG18] M. A. ABDELRAHEEM, T. ANDERSSON, C. GEHRMANN, C. GLACKIN. **"Practical attacks on relational databases protected via searchable encryption"**. In: *International Conference on Information Security (ISC)*. 2018.

[AAKM20] D. ADKINS, A. AGARWAL, S. KAMARA, T. MOATAZ. **"Encrypted blockchain databases"**. In: *ACM Conference on Advances in Financial Technologies (AFT)*. 2020.

[Age88] AGENCY FOR HEALTHCARE RESEARCH AND QUALITY. **"The National (Nationwide) Inpatient Sample (NIS) of the Healthcare Cost and Utilization Project (HCUP)"**. https://www.hcup-us.ahrq.gov/nisoverview.jsp. Accessed 2022-10-17. 1988.

[AHKM19] A. AGARWAL, M. HERLIHY, S. KAMARA, T. MOATAZ. **"Encrypted databases for differential privacy"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2019.3 (2019).

[AKM19] G. AMJAD, S. KAMARA, T. MOATAZ. **"Breach-Resistant Structured Encryption"**. In: *Proceedings on Privacy Enhancing Technologies (Po/PETS '19)*. 2019.

[AKSX04] R. AGRAWAL, J. KIERNAN, R. SRIKANT, Y. XU. **"Order preserving encryption for numeric data"**. In: *International Conference on Management of Data (SIGMOD)*. 2004.

[Ama23] AMAZON S3. **"Clients data and queries content discloses"**. hhttps://aws.amazon.com/s3/. Accessed 2023-11-17. 2023.

[Ana18] ANACONDA, INC. **"Numba – A just-in-time compiler for numerical functions in Python"**. http://numba.pydata.org. Accessed 2022-10-17. 2018.

[APP+21] G. AMJAD, S. PATEL, G. PERSIANO, K. YEO, M. YUNG. **"Dynamic Volume-Hiding Encrypted Multi-Maps with Applications to Searchable Encryption"**. In: *IACR ePrint* 765 (2021).

[ARZB11] R. ADA POPA, C. REDFIELD, N. ZELDOVICH, H. BALAKRISHNAN. **"CryptDB: Protecting confidentiality with encrypted query processing"**. In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2011, pp. 85–100.

[BBO07] M. BELLARE, A. BOLDYREVA, A. O'NEILL. **"Deterministic and efficiently searchable encryption"**. In: *Annual International Cryptology Conference (CRYPTO)*. 2007.

[BCO11] A. BOLDYREVA, N. CHENETTE, A. O'NEILL. **"Order-preserving encryption revisited: Improved security analysis and alternative solutions"**. In: *Annual Cryptology Conference (CRPYTO)*. 2011.

[BDOP04] D. BONEH, G. DI CRESCENZO, R. OSTROVSKY, G. PERSIANO. **"Public key encryption with keyword search"**. In: *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2004.

[BF17] R. BOST, P.-A. FOUQUE. **"Thwarting leakage abuse attacks against searchable encryption - A formal approach and applications to database padding"**. In: *IACR ePrint* 1060 (2017).

[BGC+18] V. BINDSCHAEDLER, P. GRUBBS, D. CASH, T. RISTENPART, V. SHMATIKOV. **"The Tao of inference in privacy-protected databases"**. In: *Proceedings of the VLDB Endowment* 11.11 (2018).

[BKM20] L. BLACKSTONE, S. KAMARA, T. MOATAZ. **"Revisiting leakage abuse attacks"**. In: *Network and Distributed System Security Symposium (NDSS)*. 2020.

[BMO17] R. BOST, B. MINAUD, O. OHRIMENKO. **"Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives"**. In: *ACM Conference on Computer and Communications Security (CCS '17)*. 2017.

[Bos16] R. BOST. **"Sophos - Forward Secure Searchable Encryption"**. In: *ACM Conference on Computer and Communications Security (CCS '16)*. 20016.

[BSW11] D. BONEH, A. SAHAI, B. WATERS. **"Functional encryption: Definitions and challenges"**. In: *Theory of Cryptography Conference (TCC)*. 2011.

[BZ06] M. BARBARO, T. J. ZELLER. **"A face is Exposed for AOL Searcher No. 4417749"**. New York Times. 2006.

[CGKO06] R. CURTMOLA, J. GARAY, S. KAMARA, R. OSTROVSKY. **"Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions"**. In: *ACM Conference on Computer and Communications Security (CCS '06)*. ACM, 2006, pp. 79–88.

[CGKS95] B. CHOR, O. GOLDREICH, E. KUSHILEVITZ, M. SUDAN. **"Private information retrieval"**. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. 1995.

[CGPR15] D. CASH, P. GRUBBS, J. PERRY, T. RISTENPART. **"Leakage-abuse attacks against searchable encryption"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015.

[Cha12] M. CHAPUT. **"Whoosh"**. https://whoosh.readthedocs.io/en/latest/. Accessed 2022-10-17. 2012.

[Cit18] CITY OF NEW YORK. **"Recoupment for damaged city-owned property"**. https://data.cityofnewyork.us/Transportation/Recoupment-for-Damaged-City-owned-Property/68k5-hdzw. Accessed 2022-10-17. 2018.

[CJJ+13] D. CASH, S. JARECKI, C. JUTLA, H. KRAWCZYK, M.-C. ROŞU, M. STEINER. **"Highly-scalable searchable symmetric encryption with support for boolean queries"**. In: *Annual Cryptology Conference (CRYPTO)*. 2013.

[CJJ+14] D. CASH, J. JAEGER, S. JARECKI, C. JUTLA, H. KRAWCZYK, M. ROSU, M. STEINER. **"Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation"**. In: *Network and Distributed System Security Symposium (NDSS '14)*. 2014.

[CK10] M. CHASE, S. KAMARA. **"Structured Encryption and Controlled Disclosure"**. In: *Advances in Cryptology - ASIACRYPT '10*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 577–594.

[CL07] G. V. CORMACK, T. R. LYNAM. **"TREC public spam corpus"**. https://plg.uwaterloo.ca/~gvcormac/treccorpus07/. Accessed 2022-10-17. 2007.

[CLRZ18] G. CHEN, T.-H. LAI, M. K. REITER, Y. ZHANG. **"Differentially private access patterns for searchable symmetric encryption"**. In: *IEEE Conference on Computer Communications (INFOCOM)*. 2018.

[CM05] Y.-C. CHANG, M. MITZENMACHER. **"Privacy preserving keyword searches on remote encrypted data"**. In: *International Conference on Applied Cryptography and Network security (ACNS)*. 2005.

[Coh15] W. W. COHEN. **"Enron dataset"**. https://www.cs.cmu.edu/~./enron/. Accessed 2022-10-17. 2015.

[Coo09] J. L. COOLIDGE. **"The gambler's ruin"**. In: *The Annals of Mathematics* 10.4 (1909), pp. 181–192.

[DDC16] F. B. DURAK, T. M. DUBUISSON, D. CASH. **"What else is revealed by order-revealing encryption?"** In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2016.

[DDF+16] S. DEVADAS, M. v. DIJK, C. W. FLETCHER, L. REN, E. SHI, D. WICHS. **"Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM"**. In: *TCC 2016*. 2016.

[DHP21] M. DAMIE, F. HAHN, A. PETER. **"A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents"**. In: *USENIX Security Symposium (USENIX Security)*. 2021.

[DPP+16] I. DEMERTZIS, S. PAPADOPOULOS, O. PAPAPETROU, A. DELIGIANNAKIS, M. GAROFALAKIS. **"Practical private range search revisited"**. In: *International Conference on Management of Data (SIGMOD)*. 2016.

[DPPS20] I. DEMERTZIS, D. PAPADOPOULOS, C. PAPAMANTHOU, S. SHINTRE. **"*SEAL*: Attack mitigation for encrypted databases via adjustable leakage"**. In: *USENIX Security Symposium (USENIX Security)*. 2020.

[DR+14] C. DWORK, A. ROTH. **"The algorithmic foundations of differential privacy"**. In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014).

[ECW+14] M. ESCH, J. CHEN, S. WEISE, K. HASSANI-PAK, U. SCHOLZ, M. LANGE. **"A query suggestion workflow for life science IR-systems"**. In: *Journal of Integrative Bioinformatics* 11.2 (2014).

[FJK+15] S. FABER, S. JARECKI, H. KRAWCZYK, Q. NGUYEN, M. ROSU, M. STEINER. **"Rich queries on encrypted data: Beyond exact matches"**. In: *European Symposium on Research in Computer Security (ESORICS)*. 2015.

[FMA+20] F. FALZON, E. A. MARKATOU, AKSHIMA, D. CASH, A. RIVKIN, J. STERN, R. TAMASSIA. **"Full database reconstruction in two dimensions"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020.

[FVY+17] B. FULLER, M. VARIA, A. YERUKHIMOVICH, E. SHEN, A. HAMLIN, V. GADEPALLY, R. SHAY, J. D. MITCHELL, R. K. CUNNINGHAM. **"SoK: Cryptographically protected database search"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2017.

[Gen09] C. GENTRY. **"Fully homomorphic encryption using ideal lattices"**. In: *ACM Symposium on Theory of Computing (STOC)*. 2009.

[GJW19] Z. GUI, O. JOHNSON, B. WARINSCHI. **"Encrypted databases: New volume attacks against range queries"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2019.

[GKL+20] P. GRUBBS, A. KHANDELWAL, M.-S. LACHARITÉ, L. BROWN, L. LI, R. AGARWAL, T. RISTENPART. **"Pancake: Frequency smoothing for encrypted data stores"**. In: *USENIX Security Symposium (USENIX Security)*. 2020.

[GKM21] M. George, S. Kamara, T. Moataz. **"Structured encryption and dynamic leakage suppression"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2021.

[GLMP18] P. Grubbs, M.-S. Lacharité, B. Minaud, K. G. Paterson. **"Pump up the volume: Practical database reconstruction from volume leakage on range queries"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2018.

[GLMP19] P. Grubbs, M.-S. Lacharité, B. Minaud, K. G. Paterson. **"Learning to reconstruct: Statistical learning theory and encrypted database attacks"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2019.

[GMP16] S. Garg, P. Mohassel, C. Papamanthou. **"TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption"**. In: *Advances in Cryptology - CRYPTO 2016*. 2016, pp. 563–592.

[GMW87] O. Goldreich, S. Micali, A. Wigderson. **"How to play any mental game"**. In: *ACM Symposium on Theory of Computing (STOC)*. 1987.

[GO96] O. Goldreich, R. Ostrovsky. **"Software protection and simulation on oblivious RAMs"**. In: *Journal of the ACM* 43.3 (1996), pp. 431–473.

[Goh03] E.-J. Goh. **"Secure Indexes"**. Tech. rep. 2003/216. See http://eprint.iacr.org/2003/216. IACR ePrint Cryptography Archive, 2003.

[Gov18] Government Digital Service. **"Organogram of Staff Roles & Salaries of Government Legal Department"**. https://data.gov.uk/dataset/34d08a53-6b96-4fb6-b043-627e2b25840d/organogram-of-staff-roles-salaries. Accessed 2022-10-17. 2018.

[GPP21] Z. Gui, K. G. Paterson, S. Patranabis. **"Rethinking Searchable Symmetric Encryption"**. In: *IACR ePrint* 879 (2021).

[GPPJ18] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, R. Jalili. **"New constructions for forward and backward private symmetric searchable encryption"**. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1038–1055.

[GPPW20] Z. Gui, K. G. Paterson, S. Patranabis, B. Warinschi. **"SWiSSSE: System-Wide Security for Searchable Symmetric Encryption"**. In: *IACR ePrint* 1328 (2020).

[Gro08] G. Grothaus. **"General implementation of the PQ-tree algorithm"**. https://github.com/Gregable/pq-trees. Accessed 2022-10-17. 2008.

[GSB+17] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, T. Ristenpart. **"Leakage-abuse attacks against order-revealing encryption"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2017.

[HEK12] Y. Huang, D. Evans, J. Katz. **"Private set intersection: Are garbled circuits better than custom protocols?"** In: *Network and Distributed Systems Security Symposium (NDSS)*. 2012.

[HMW+20] C. R. Harris, K. J. Millman, S. J. v. d. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. v. Kerkwijk, M. Brett, A. Haldane, J. F. d. Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant. **"Array programming with NumPy"**. In: *Nature* 585.7825 (2020).

[HZNF15] J. HEURIX, P. ZIMMERMANN, T. NEUBAUER, S. FENZ. **"A taxonomy for privacy enhancing technologies"**. In: *Computers & Security* 53 (2015).

[IDC23] IDC GLOBAL. **"The Worldwide IDC Global DataSphere five-year forecast 2022-2026"**. https://www.idc.com/getdoc.jsp?containerId=US49018922. Accessed 2023-11-17. 2023.

[IKK12] M. S. ISLAM, M. KUZU, M. KANTARCIOGLU. **"Access pattern disclosure on searchable encryption: Ramification, attack and mitigation"**. In: *Network and Distributed System Security Symposium (NDSS)*. 2012.

[IKK14] M. S. ISLAM, M. KUZU, M. KANTARCIOGLU. **"Inference attack against encrypted range queries on outsourced databases"**. In: *ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2014.

[IMNL09] R. ISLAMAJ DOGAN, G. C. MURRAY, A. NÉVÉOL, Z. LU. **"Understanding PubMed® user search behavior through log analysis"**. In: *Database* (2009).

[Jan06] B. J. JANSEN. **"Search log analysis: What it is, what's been done, how to do it"**. In: *Library & Information Science Research* 28.3 (2006).

[JCMB00] S. JONES, S. J. CUNNINGHAM, R. MCNAB, S. BODDIE. **"A transaction log analysis of a digital library"**. In: *International Journal on Digital Libraries* 3.2 (2000).

[JMH+16] S. JAIN, D. MORITZ, D. HALPERIN, B. HOWE, E. LAZOWSKA. **"SQLShare: Results from a multi-year SQL-as-a-service experiment"**. In: *International Conference on Management of Data (SIGMOD)*. 2016.

[JPL13] D. JIANG, J. PEI, H. LI. **"Mining search and browse logs for web search: A survey"**. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 4.4 (2013).

[JPS+16] A. E. JOHNSON, T. J. POLLARD, L. SHEN, H. L. LI-WEI, M. FENG, M. GHASSEMI, B. MOODY, P. SZOLOVITS, L. A. CELI, R. G. MARK. **"MIMIC-III, a freely accessible critical care database"**. In: *Scientific Data* 3.1 (2016).

[JPS21] M. JURADO, C. PALAMIDESSI, G. SMITH. **"A formal information-theoretic leakage analysis of order-revealing encryption"**. In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–16.

[JS06] B. J. JANSEN, A. SPINK. **"How are we searching the World Wide Web? A comparison of nine search engine transaction logs"**. In: *Information Processing & Management (IP&M)* 42.1 (2006).

[JS19] M. JURADO, G. SMITH. **"Quantifying information leakage of deterministic encryption"**. In: *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 2019, pp. 129–139.

[KACZ15] O. KENNEDY, J. AJAY, G. CHALLEN, L. ZIAREK. **"Pocket data: The need for TPC-MOBILE"**. In: *Technology Conference on Performance Evaluation and Benchmarking*. 2015.

[Kam15] S. KAMARA. **"Encrypted Search"**. In: *XRDS* 21.3 (2015), pp. 30–34.

[KGV83] S. KIRKPATRICK, C. D. GELATT, M. P. VECCHI. **"Optimization by simulated annealing"**. In: *Science* 220.4598 (1983).

[KKI+17] E. KACPRZAK, L. M. KOESTEN, L.-D. IBÁÑEZ, E. SIMPERL, J. TENNISON. **"A query log analysis of dataset search"**. In: *International Conference on Web Engineering (ICWE)*. 2017.

[KKM+22] S. KAMARA, A. KATI, T. MOATAZ, T. SCHNEIDER, A. TREIBER, M. YONLI. **"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)*. 2022.

[KKM+23] S. KAMARA, A. KATI, T. MOATAZ, J. DEMARIA, A. PARK, A. TREIBER. **"Repository for MAPLE: MArkov Process Leakage attacks on Encrypted Search"**. https://github.com/anonymous-repo-submission/artifact. 2023.

[KKM+24] S. KAMARA, A. KATI, T. MOATAZ, J. DEMARIA, A. PARK, A. TREIBER. **"MAPLE: MArkov Process Leakage attacks on Encrypted Search"**. In: *The annual Privacy Enhancing Technologies Symposium (PETS)*. 2024.

[KKNO16] G. KELLARIS, G. KOLLIOS, K. NISSIM, A. O'NEILL. **"Generic attacks on secure outsourced databases"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2016.

[KM17] S. KAMARA, T. MOATAZ. **"Boolean searchable symmetric encryption with worst-case sub-linear complexity"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2017.

[KM19] S. KAMARA, T. MOATAZ. **"Computationally Volume-Hiding Structured Encryption"**. In: *Advances in Cryptology - Eurocrypt' 19*. 2019.

[KM23] S. KAMARA, T. MOATAZ. **"Bayesian Leakage Analysis"**. Tech. rep. 2023.

[KMO18] S. KAMARA, T. MOATAZ, O. OHRIMENKO. **"Structured encryption and leakage suppression"**. In: *Annual International Cryptology Conference (CRYPTO)*. 2018.

[KMPP22] E. M. KORNAROPOULOS, N. MOYER, C. PAPAMANTHOU, A. PSOMAS. **"Leakage Inversion: Towards Quantifying Privacy in Searchable Encryption"**. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1829–1842.

[KMPQ21] S. KAMARA, T. MOATAZ, A. PARK, L. QIN. **"A decentralized and encrypted national gun registry"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

[KMZZ20] S. KAMARA, T. MOATAZ, S. ZDONIK, Z. ZHAO. **"An Optimal Relational Database Encryption Scheme"**. In: *IACR ePrint* 274 (2020).

[KP13] S. KAMARA, C. PAPAMANTHOU. **"Parallel and Dynamic Searchable Symmetric Encryption"**. In: *Financial Cryptography and Data Security (FC '13)*. 2013.

[KPR12] S. KAMARA, C. PAPAMANTHOU, T. ROEDER. **"Dynamic Searchable Symmetric Encryption"**. In: *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

[KPT19] E. M. KORNAROPOULOS, C. PAPAMANTHOU, R. TAMASSIA. **"Data recovery on encrypted databases with k-nearest neighbor query leakage"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2019.

[KPT20] E. M. KORNAROPOULOS, C. PAPAMANTHOU, R. TAMASSIA. **"The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2020.

[KPT21] E. M. KORNAROPOULOS, C. PAPAMANTHOU, R. TAMASSIA. **"Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

[KS12] V. KOLESNIKOV, A. SHIKFA. **"On the limits of privacy provided by order-preserving encryption"**. In: *Bell Labs Technical Journal* 17.3 (2012).

[LDS+10] P. LAMESCH, K. DREHER, D. SWARBRECK, R. SASIDHARAN, L. REISER, E. HUALA. **"Using the Arabidopsis information resource (TAIR) to find information about Arabidopsis genes"**. In: *Current Protocols in Bioinformatics* 30.1 (2010).

[LMP18] M.-S. LACHARITÉ, B. MINAUD, K. G. PATERSON. **"Improved reconstruction attacks on encrypted data using range query leakage"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2018.

[LMZ+11] Y. LIU, J. MIAO, M. ZHANG, S. MA, L. RU. **"How do users describe their information need: Query recommendation based on snippet click model"**. In: *Expert Systems with Applications* 38.11 (2011).

[LO13] S. LU, R. OSTROVSKY. **"Distributed Oblivious RAM for Secure Two-Party Computation"**. In: *TCC 2013*. 2013.

[LPS+18] S. LAI, S. PATRANABIS, A. SAKZAD, J. K. LIU, D. MUKHOPADHYAY, R. STEINFELD, S.-F. SUN, D. LIU, C. ZUO. **"Result pattern hiding searchable encryption for conjunctive queries"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2018.

[LZWT14] C. LIU, L. ZHU, M. WANG, Y.-A. TAN. **"Search pattern leakage in searchable encryption: Attacks and new construction"**. In: *Information Sciences* 265 (2014).

[MD06] G. MISHNE, M. DE RIJKE. **"A study of blog search"**. In: *European Conference on Information Retrieval (ECIR)*. 2006.

[MFST21] E. A. MARKATOU, F. FALZON, W. SCHOR, R. TAMASSIA. **"Reconstructing with Less: Leakage Abuse Attacks in Two-Dimensions"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021.

[MIT15] MIT LINCOLN LABORATORY. **"SPARTA Framework"**. https://github.com/mit-ll/SPARTA. Accessed 2022-10-17. 2015.

[MT19] E. A. MARKATOU, R. TAMASSIA. **"Full database reconstruction with access and search pattern leakage"**. In: *International Conference on Information Security (ISC)*. 2019.

[Nat09] NATIONAL INSTITUTES OF HEALTH. **"PubMed log data"**. https://www.ncbi.nlm.nih.gov/CBBresearch/Lu/LogStudy/. Accessed 2022-10-17. 2009.

[Nat23] NATIONAL CYBER STRATEGY. **"The UK Cyber Security Breaches Survey 2023"**. https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2023/cyber-security-breaches-survey-2023. Accessed 2023-11-17. 2023.

[NBB+15] H. V. NGUYEN, K. BÖHM, F. BECKER, B. GOLDMAN, G. HINKEL, E. MÜLLER. **"Identifying user interests within the data space-A case study with SkyServer"**. In: *International Conference on Extending Database Technology (EDBT)*. 2015.

[NHP+21] J. NING, X. HUANG, G. S. POH, J. YUAN, Y. LI, J. WENG, R. H. DENG. **"LEAP: Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021.

[NIS14] NIST. **"TREC 2014 session track"**. http://ir.cis.udel.edu/sessions/guidelines14.html. Accessed 2022-10-17. 2014.

[NIS21] NIST. **"Toward a PEC use-case suite (preliminary draft)"**. NIST White Paper (Draft). 2021.

[NKW15] M. NAVEED, S. KAMARA, C. V. WRIGHT. **"Inference attacks on property-preserving encrypted databases"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015.

[Nor98] J. R. NORRIS. **"Markov chains"**. 2. Cambridge university press, 1998.

[OK21] S. OYA, F. KERSCHBAUM. **"Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption"**. In: *USENIX Security Symposium (USENIX Security)*. 2021.

[OK22] S. OYA, F. KERSCHBAUM. **"IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization"**. In: *USENIX Security Symposium (USENIX Security)*. 2022.

[PCT06] G. PASS, A. CHOWDHURY, C. TORGESON. **"A picture of search"**. In: *International Conference on Scalable Information Systems (INFOSCALE)*. 2006.

[Pea05] K. PEARSON. **"The problem of the random walk"**. In: *Nature* 72.1865 (1905), pp. 294–294.

[Pho11] PHOENIX BIOINFORMATICS. **"Araport11 genome release"**. https://www.arabidopsis.org/download/index-auto.jsp?dir=%2Fdownload_files%2FGenes%2FAraport11_genome_release. Accessed 2022-10-17. 2011.

[Pho13] PHOENIX BIOINFORMATICS. **"TAIR data release"**. https://www.arabidopsis.org/download/index-auto.jsp?dir=%2Fdownload_files%2FPublic_Data_Releases%2FTAIR_Data_20131231. Accessed 2022-10-17. 2013.

[PPYY19] S. PATEL, G. PERSIANO, K. YEO, M. YUNG. **"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*. ACM, 2019, pp. 79–93.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient circuit-based PSI with linear communication"**. In: *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2019.

[PW16] D. POULIOT, C. V. WRIGHT. **"The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption"**. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2016.

[PWLP20] R. PODDAR, S. WANG, J. LU, R. A. POPA. **"Practical volume-based attacks on encrypted databases"**. In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. 2020.

[RPH21] R. G. ROESSINK, A. PETER, F. HAHN. **"Experimental review of the IKK query recovery attack: Assumptions, recovery rate and improvements"**. In: *International Conference on Applied Cryptography and Network Security (ACNS)*. 2021.

[Sch95] A. SCHWARZENBERG-CZERNY. **"On matrix factorization and efficient least squares solution."** In: *Astronomy and Astrophysics Supplement, v. 110, p. 405* 110 (1995), p. 405.

[SDS+13] E. STEFANOV, M. v. DIJK, E. SHI, C. FLETCHER, L. REN, X. YU, S. DEVADAS. **"Path ORAM: An Extremely Simple Oblivious RAM Protocol"**. In: *ACM Conference on Computer and Communications Security (CCS '13)*. 2013.

[SGT+07] V. Singh, J. Gray, A. Thakar, A. S. Szalay, J. Raddick, B. Boroski, S. Lebedeva, B. Yanny. **"SkyServer traffic report-The first five years"**. In: *arXiv preprint cs/0701173* (2007).

[Sog08] Sogou, Inc. **"SogouQ"**. http://www.sogou.com/labs/resource/q.php. Accessed 2022-10-17. 2008.

[SOPK21] Z. Shang, S. Oya, A. Peter, F. Kerschbaum. **"Obfuscated access and search patterns in searchable encryption"**. In: *Network and Distributed System Security Symposium (NDSS)*. 2021.

[SS13] E. Stefanov, E. Shi. **"Multi-cloud oblivious storage"**. In: *ACM CCS 13*. 2013.

[SWP00] D. X. Song, D. Wagner, A. Perrig. **"Practical techniques for searches on encrypted data"**. In: *IEEE Symposium on Security and Privacy (S&P)*. 2000.

[VGO+20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors. **"SciPy 1.0: Fundamental algorithms for scientific computing in Python"**. In: *Nature Methods* 17 (2020).

[Vit67] A. Viterbi. **"Error bounds for convolutional codes and an asymptotically optimum decoding algorithm"**. In: *IEEE transactions on Information Theory* 13.2 (1967), pp. 260–269.

[VMÖ17] C. Van Rompay, R. Molva, M. Önen. **"A leakage-abuse attack against multi-user searchable encryption"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017).

[Wal14] Walmart Inc. **"Walmart recruiting - Store sales forecasting"**. https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/overview. Accessed 2022-10-17. 2014.

[WBK+11] W. Weerkamp, R. Berendsen, B. Kovachev, E. Meij, K. Balog, M. De Rijke. **"People searching for people: Analysis of a people search engine log"**. In: *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 2011.

[Wel03] L. R. Welch. **"Hidden Markov models and the Baum-Welch algorithm"**. In: *IEEE Information Theory Society Newsletter* 53.4 (2003), pp. 10–13.

[Wik14] Wikimedia Foundation. **"Simple English Wikipedia"**. https://simple.wikipedia.org/. Accessed 2022-10-17. 2014.

[Wik22] Wikimedia Foundation. **"Wiktionary Statistics"**. https://en.wiktionary.org/wiki/Special:Statistics. Accessed 2022-10-31. 2022.

[WP17] C. V. Wright, D. Pouliot. **"Early detection and analysis of leakage abuse vulnerabilities"**. In: *Cryptology ePrint Archive* (2017).

[WS12] P. Williams, R. Sion. **"Single round access privacy on outsourced storage"**. In: *ACM Conference on Computer and Communications Security (CCS '12)*. 2012, pp. 293–304.

[Yah16] Yahoo Labs. **"Yahoo Labs Webscope language data"**. https://webscope.sandbox.yahoo.com/catalog.php?datatype=l&guccounter=1. Accessed 2022-10-17. 2016.

[Yan14] YANDEX N.V. **"Yandex personalized web search challenge 2014"**. https://www.kaggle.com/c/yandex-personalized-web-search-challenge/data. Accessed 2022-10-17. 2014.

[Yao82] A. C. YAO. **"Protocols for secure computations"**. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. 1982.

[YZGW20] J. YAO, Y. ZHENG, Y. GUO, C. WANG. **"SoK: A systematic study of attacks in efficient encrypted cloud data search"**. In: *International Workshop on Security in Blockchain and Cloud Computing (SBC)*. 2020.

[Zha11] J. ZHANG. **"Data use and access behavior in eScience: Exploring data practices in the new data-intensive science paradigm"**. Drexel University Philadelphia, PA, 2011.

[ZKMZ21] Z. ZHAO, S. KAMARA, T. MOATAZ, S. ZDONIK. **"Encrypted Databases: From Theory to Systems"**. In: *Conference on Innovative Data Systems Research (CIDR)*. 2021.

[ZKP16] Y. ZHANG, J. KATZ, C. PAPAMANTHOU. **"All your queries are belong to us: The power of file-injection attacks on searchable encryption"**. In: *USENIX Security Symposium (USENIX Security)*. 2016.

List of Figures

4.1 High-level overview of LEAKER's major components and attacks and/or statistics evaluation flow. Dashed arrows indicate optional usage. With the exception of `Extension`, all parts can be used for both keyword and range queries. 21

4.2 Fraction of correctly uncovered queries of the attacks of [BKM20] on the TAIR keyword database (cf. paragraph 4.3.1). Queries are (a) either taken from the database [LDS+10] (*prior artificial style*) or (b) from a client's query log (*our real-world evaluation*). Here, we use the queries of the respective source least frequent *in the database* (lowest-selectivity). Parameters are the same as in subsection 4.4.1. 29

6 Normalized mean absolute error of value reconstruction attacks on different datasets using a truncated Zipf distribution. Captions include the respective dataset's general data distribution (cf. section 4.1, subsubsection 7.2.2.1) identified as risk factors. i resp. j iterations were performed for GENKKNO and APPROXVAL resp. LMP, ARR, and APA. 39

7 MCE of count reconstruction attacks on ‡ using a $*$ distribution. i displays how many iterations were performed. 43

1 Our attack: Stationary. 51

2 Our attack: Decoder. 54

3 The Obv-N variant. 57

4 The Obv-B variant. 57

5 Our inference attack: $\star$-Smpl, where $\star$ is a placeholder for Stationary and Decoder. 59

6 Our new attacks and the IHOP attack [OK22] evaluated against AOL (top) and TAIR (bottom) in the *known-sample* setting for the *Exact*, *All*, *Other* and *Split* scenarios (from left to right). 67

7 Our new attacks and the IHOP attack [OK22] evaluated on the Zipf-Zipf query distribution. All evaluations are done using a *fixed* minimum Hamming weight $= 2$. 67

1 Frequencies of numerical dataset values. 80

2 Query frequency distribution of SDSS query logs on the **D**PhotoObjAll collection (here scaled by $\times 10^5$; $N = 10\,455\,488$) as well as an artificial Zipf distribution on a random collection with $N = 10^4$. 80

3 X-Y attack evaluations against **D**AOL and **D**TAIR. All evaluations are 5 × 5, except for 3 × 3 evaluations done for COUNT v.2. 150 queries are drawn for each of the most active users (single user setting; S) without replacement according to their query frequency from the 500 most (H) or least (L) frequent queries in the query log that are also contained in the partial knowledge, respectively. The resulting mean frequency is given by freq($\mathcal{Q}$). 81
4 Query and keyword distributions (log-log scale) of query logs and corresponding databases for AOL, TAIR, GMail-L, and Drive. The queries are aggregated over all users. 84
5 Query log query selectivity distributions (log-log scale). 85
6 Mean query selectivity of all AOL, Pocketdata, and TAIR users, and the 100 000 least active PubMed users/instances (sorted by descending activity). 86
7 The Viterbi algorithm [Vit67]. 90
8 The Baum-Welch algorithm [Wel03]. 91
9 Our new attacks and the IHOP attack [OK22] evaluated on the Zipf-Zipf distribution setting. Evaluations are done using a *variable* Hamming weight and a *fixed* number of states = 500. 92
10 Our new attacks and the IHOP attack [OK22] evaluated in the known-distribution setting. The results represent the average recovery rate over 30 runs. 93

List of Tables

4.1 Major leakage attacks and our perceived risk. The target is either Keyword query (K) or Range data (Value/Count, RV/RC) reconstruction. B is the maximum width and k the amount of missing queries per width. $\mathcal{A}$ denotes that all possible response lengths occur (only within all widths $\leq B$ for $\mathcal{A}_B$, or k missing therein for $\mathcal{A}_{B,k}$). ○ shows no success on our real-world datasets, ◑ denotes some success (K for high partial Knowledge ≥ 75%, D for Dense data, W for large Widths close to N, S for Specific data values, E for Evenly and ¬E for unevenly distributed data collections), ● is severe risk across all of our evaluated instances. 16
4.2 Overview of our keyword search use cases and dataset properties. $\#\mathbf{Q}_D$ is the size of the entire log and $\#\mathbf{Q}$ the amount of unique queries. n is the amount of documents and $\#\mathbb{W}$ the amount of unique keywords. 25

4.3 Summary of our *scientific data* range query logs on the PhotoObjAll.dec collection [SGT+07] ($n = 5\,242\,134$ entries with domain $N = 10\,456$, density 95.82%, and an even data distribution). $\#\mathbf{Q}_D$ is the size of the entire log and $\#\mathbf{Q}$ the amount of unique queries. 26
4.4 Summary of our range use cases and data with n entries, domain size N, and density δ. E and ¬E denote even and uneven data distributions, respectively (cf. section 4.1). 27
4.5 Normalized mean errors on the entire SDSS query logs. The collection is sampled 25× uniformly at random with size $n = 10^4$ ($n = 10^3$ for APA and ARR). 36

5.1 Summary of the evaluation setup for known-sample attacks to recover the queries of the ith user. 63
5.2 Results summary of our attacks and the IHOP attack [OK22] in the *known-sample setting* with *fixed* leakage, highlighting the maximum and median recovery rates (in % of correctly recovered queries). The attacks are evaluated on the TAIR [ECW+14] and the AOL [PCT06] query logs. The adversarial scenarios for the attacks are described in paragraph 5.4.3 and paragraph 5.4.3. 66

List of Abbreviations

PETs	Privacy-Enhancing Technologies
ESAs	Encrypted Search Algorithms
SSE	Searchable Symmetric Encryption
STE	Structured Encryption
ORAM	Oblivious RAM