Generic Attacks on Secure Outsourced Databases

Georgios Kellaris*
Boston University and
Harvard University
gkellaris@g.harvard.edu

George Kollios†
Boston University
gkollios@cs.bu.edu

Kobbi Nissim‡
Ben-Gurion University
and Harvard University
kobbi@seas.harvard.edu

Adam O'Neill
Georgetown University
adam@cs.georgetown.edu

ABSTRACT

Recently, various protocols have been proposed for securely outsourcing database storage to a third party server, ranging from systems with "full-fledged" security based on strong cryptographic primitives such as fully homomorphic encryption or oblivious RAM, to more practical implementations based on searchable symmetric encryption or even on deterministic and order-preserving encryption. On the flip side, various attacks have emerged that show that for some of these protocols confidentiality of the data can be compromised, usually given certain auxiliary information.

We take a step back and identify a need for a formal understanding of the inherent efficiency/privacy trade-off in outsourced database systems, *independent of the details of the system*. We propose abstract models that capture secure outsourced storage systems in sufficient generality, and identify two basic sources of leakage, namely *access pattern* and *communication volume*. We use our models to distinguish certain classes of outsourced database systems that have been proposed, and deduce that all of them exhibit at least one of these leakage sources.

We then develop generic *reconstruction attacks* on any system supporting range queries where either access pattern or communication volume is leaked. These attacks are in a rather weak passive adversarial model, where the untrusted server knows only the underlying *query* distribution. In particular, to perform our attack the server need not have any prior knowledge about the data, and need not know any of the issued queries nor their results. Yet, the server can reconstruct the secret attribute of every record in the database after about N^4 queries, where N is the domain size. We provide a matching lower bound showing that our attacks are essentially optimal. Our reconstruction attacks using communication volume apply even to systems based on homomorphic encryption or oblivious RAM in the natural way.

Finally, we provide experimental results demonstrating the efficacy of our attacks on real datasets with a variety of different features. On all these datasets, after the required number of queries our attacks successfully recovered the secret attributes of every record in at most a few seconds.

Keywords: Secure outsourced databases, generic attacks

1. INTRODUCTION

As organizations struggle with the accumulation of large amounts of data, a popular practice is to *outsource* them to third party servers. Because their data may be sensitive (e.g., medical or financial), a natural idea is to employ cryptographic techniques to ensure confidentiality while still allowing efficient query processing.

Many cryptographic techniques exist that can be applied to this problem, ranging from solutions based on tools such as fully homomorphic encryption (FHE) [20, 46] or oblivious RAM (ORAM) [23, 25] to more practical solutions based on weaker primitives such as structural encryption [33] or deterministic and order-preserving encryption [42, 2]. However, the privacy provided by the resulting secure outsourced database systems is poorly understood. For example, recent works focusing on some practical solutions [28, 3, 31, 11, 38], have shown that confidentiality can be compromised given auxiliary information on the data.

Here, we take a step back and identify a need for a formal understanding of the inherent efficiency/privacy trade-off in outsourced database systems, *independent of the details of the system*. In particular, we develop the first attack against systems leaking only *communication volume*. This attack applies even to systems based on FHE or ORAM.

1.1 Our Results

Abstract Models. We present abstract models that capture secure outsourced database systems in sufficient generality to reason about their security in a generic and implementation independent way. In particular, we consider two basic sources of leakage for such systems: *access pattern* and *communication volume*. Access pattern refers to the server learning which "encrypted" records are returned as the result of a query. On the other hand, communication volume refers to the server learning *how many* encrypted records are returned as a result of a query.

*Work supported by NSF Grants no. CNS-1414119 and CNS-1565387.

†Work partially supported by the NSF CNS-1414119 grant.

‡Work supported by NSF Grant no. NSF CNS-1565387 and grants from the Sloan Foundation.

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978386

A Taxonomy. In order to reason about which protocols have which leakage channels (and how to design new protocols without them), it is useful to distinguish some classes of outsourced database systems. In particular, we define *atomic* outsourced database systems, where the server stores a collection of ciphertexts such that each record in the dataset (or rather the search key for each record) is encrypted as one of these ciphertexts, but there may be additional ciphertexts encrypting a dummy record; further, communication from the server on a query includes some subset of these ciphertexts. It is useful to also define *static* atomic systems, where the encrypted records are not changed as a result of a query, and *non-storage-inflating* atomic systems, where the number of encrypted records is equal to the number of records in the database. We observe that static atomic systems that are non-storage-inflating must leak the access pattern, i.e., which of the encrypted records are returned on a query. This covers most systems based on searchable symmetric encryption or on deterministic and order-preserving encryption [42, 2, 4]. In the more general (not necessarily atomic) setting, we also define *fixed communication overhead* protocols, where the length of communication sent from the server as a result of a query is proportional to the number of records matching the query. Systems simply based on FHE or ORAM have fixed communication overhead. We observe that such systems leak the communication volume.

Reconstruction Attacks. We develop reconstruction attacks on outsourced database systems where either access pattern *or* communication volume is leaked, and thus our attacks apply to most solutions proposed in prior work. In a reconstruction attack an adversary exploits leakage to recover the search keys. A successful reconstruction attack should recover a significant fraction of the search keys with good probability, preferably in polynomial-time and with a polynomial number of queries in the database size. Here we consider a rather weak adversarial model (hence making our attacks stronger) in which the untrusted server only knows the underlying query distribution (i.e., does not have prior knowledge about the stored data and does not get to directly learn the issued queries or their results). Thus, unlike prior work [28, 3, 31, 11, 38] our adversary is *passive* (does not choose the queries) and does not get any auxiliary information about the data. Our attacks specifically apply to outsourced database systems for range queries. Furthermore, we will assume queries are *uniform*. Clearly, these are limiting assumptions but they are an important natural basic case to start with; see discussion below.

Attacks using Access Pattern. Our reconstruction attack using the access pattern proceeds in two phases: first, the adversary identifies a record with the minimal (or maximal) value. Then, it uses the frequency in which this record is returned on random queries to determine its index. Finally it uses statistics on the occurrence of other records with the minimal record to determine their index. A simple analysis, based on Chernoff bounds, shows that the adversary can exactly reconstruct the entire index set after observing $O(N^4)$ queries, where N is the domain size, assuming queries are uniform. For dense databases $\tilde{O}(N^2)$ queries suffice.

Attacks Using Communication Volume. Our reconstruction attacks using communication volume proceed as follows. Let n be the number of records in the dataset. The attacker first observes $O(N^4)$ queries and determines for every value of $0 \leq i \leq n$ the number u_i of range queries (out of all possible queries) which return exactly i records. Next, the adversary derives a polynomial $F(x)$ whose positive integer coefficients are computable from $u_1, \ldots, u_n$. It turns out that if $F(x)$ uniquely factors into two polynomials $d(x), d^R(x)$ then each of these polynomials can be used to recover the entire index set. That was true for all datasets in our experiments.

A Matching Lowerbound. We complement our attacks with a lowerbound showing the existence of datasets that cannot be distinguished by attackers that observe significantly less than N^4 uniformly chosen range queries. This lowerbound holds both for attacks using access pattern and for attacks using communication volume.

Experiments. In order to demonstrate our attacks, we have built the following prototype. The server is instantiated as a mySQL server, and the users upload their data and query them through a CryptDB proxy. A packet sniffer residing on the server side monitors the query answers, and executes our attacks. We attacked several real datasets with different indexed domains, number of records, and record distributions. The average size of each dataset was a few thousands of records. After collecting the required number of queries, our attacks ran in a few seconds for the worst case. Both attacks (i.e., using the access pattern and using the communication volume) managed to reconstruct *all* datasets.

1.2 Discussion

In reality queries are not uniform. However, we believe that our attacks represent a significant weakness that needs to be addressed, because (1) good systems should provide protection regardless of query distribution (2) uniform or almost uniform queries on a small subset of the domain are realistic, and when that happens our attacks apply, and (3) other than the assumption on the query distribution, our attack model is very weak. Our attacks show that secure outsourced databases should avoid being static non-storage-inflating, as well as with fixed communication overhead.

Open Questions. We leave open the question on which datasets our reconstruction attack using communication volume succeeds. Additionally, it's open to extend our attacks to other query distributions and also to use only "short" range queries, as they are typically observed in practice. Finally, our attacks require N^4 queries in general and an open question is which privacy guarantees can be made when N is large. We note that in the case of order-preserving encryption, security on very large domains was studied by [8], who showed that OPE hides half of the bits of each plaintext for a dataset of uniformly chosen points (where the adversary gets the OPE encryptions of these points).

1.3 Related Work

Work in Cryptographic Community. For an overview of cryptographic techniques for search on encrypted data, we recommend the talk of Kamara [32]. Broadly, the techniques include multi-party computation [47, 24], oblivious RAM [25, 25], searchable symmetric and structural encryp-

tion [45, 22, 13, 15, 14, 12, 40, 19, 33], functional encryption [9, 43], property-preserving encryption [5, 1, 7, 39] and homomorphic encryption [10, 21]. These techniques provide different levels of security based on their leakage. In particular, [32] distinguishes between $\mathcal{L}_1$ and $\mathcal{L}_2$ leakage where access pattern corresponds to $\mathcal{L}_2$ leakage, but neither one considers communication volume. [37] identifies the basic leakages of schemes that combine searchable encryption and ORAM. It refers to access pattern leakage as $LC3$, and communication volume as $LC2$.

Work in Database Community. In the database community, the problem of querying an encrypted database was introduced by [26]. Depending on the query type, different methods have been proposed. In this work, we focus on range queries over arithmetic valued attributes. Existing solutions for range queries can be divided into three categories; (i) bucketization techniques that partition the domain space and group data records before indexing (e.g. [27, 29, 28]), (ii) order-preserving encryption schemes that use deterministic encryption which ensures that the order of the cyphertexts is the same as the order of the plaintexts (e.g. [1, 7, 41]), and (iii) solutions that use specialized data structures (e.g. [35, 44, 17]). Finally, notable examples of outsourced database systems that support range queries are CryptDB [42], Cipherbase [2], and TrustedDB [4].

Attacks. Several works (e.g., [28, 3, 31]) have shown that all the current methods allowing range queries on encrypted data can reveal information about the distribution of the plaintext values on the search domain. Islam, Kuzu, and Kantarcioglu [30] studied attacks exploiting access pattern based on auxiliary information. Liu, Zhu, Wang and Tan [36] developed such attacks based on the query pattern (i.e., information about which queries repeat). Recently, attacks on existing systems have been introduced ([11, 38]), but these attacks are application dependent as they make assumptions about the data distribution and exploit weaknesses of specific encryption protocols (e.g., OPE). Finally, [16] is the closest to our work. It is similarly general, assuming only access pattern is leaked. However, the presented attack only recovers (partial) order of the records depending on the retrieved range queries. Applied to our setting where the queries are uniformly drawn, it can reconstruct the full order after observing enough answers. Specifically, the algorithm first considers all possible orders of records, and then prunes some of them by observing answers of queries.

2. THE MODEL

We describe our abstract models of secure outsourced database systems. While the focus of our work is on attacks, our models are general enough to prove positive results as well.

2.1 Outsourced database systems

We abstract a database as a collection of records associated with search keys

$$\mathcal{D} = \{(r_1, \text{SK}_1), \ldots, (r_n, \text{SK}_n)\}.$$

We will assume that all records have fixed length κ, and that search keys are elements of domain $\mathcal{X}$ (essentially, the search keys can be viewed as the database indexing information). A query is a predicate

$$q : \mathcal{X} \to \{0, 1\}.$$

Applying a query q to a database $\mathcal{D}$ results in all records whose search keys satisfy q, i.e.,

$$q(\mathcal{D}) = \{r_i : q(\text{SK}_i) = 1\}.$$

EXAMPLE 2.1. *Throughout this article $\mathcal{X}$ will be an ordered domain of $N \in \mathbb{N}$ elements $\{1, \ldots, N\}$. We will consider the family of interval or range queries*

$$\mathcal{Q} = \left\{q_{[a,b]}\right\}_{1 \le a \le b \le N}; \quad q_{[a,b]}(c) = \begin{cases} 1 & a \le c \le b \\ 0 & otherwise \end{cases}$$

Overall, there are $\binom{N}{2} + N$ queries in $\mathcal{Q}$. Applying $q_{[a,b]}$ on a database $\mathcal{D}$ results in all records with search keys in the range $[a, b]$, i.e.,

$$q_{[a,b]}(\mathcal{D}) = \{r_i : a \le \text{SK}_i \le b\}.$$

Let $\mathcal{Q}$ be a collection of queries. An outsourced database system for queries in $\mathcal{Q}$ consists of two protocols between a user $\mathcal{U}$ and a server $\mathcal{S}$:

Setup protocol Π_{SETUP}: $\mathcal{U}$ has as input a database $\mathcal{D} = \{(r_1, \text{SK}_1), \ldots, (r_n, \text{SK}_n)\}$; $\mathcal{S}$ has no input. The output for $\mathcal{U}$ is a query key K and the output for $\mathcal{S}$ is a data structure $\mathcal{DS}$.[1]

Query protocol Π_{QUERY}: $\mathcal{U}$ has as input a query $q \in \mathcal{Q}$ and the key K produced in the setup protocol; $\mathcal{S}$ has as input $\mathcal{DS}$ produced in the setup protocol. The output for $\mathcal{U}$ is $q(\mathcal{D})$; $\mathcal{S}$ has no formal output.

We note that our model is somewhat similar to the notion of structured encryption due to Chase and Kamara [14], but is more general and applies more easily to general protocols for outsourced database systems.

Atomic Systems. We also define the special case that the outsourced database system is *atomic* in the following sense:

1. $\mathcal{DS} = (\mathcal{DS}_1, \mathcal{DS}_2)$ where $\mathcal{DS}_1 = (c_1, \ldots, c_{n'}\}$ contains encrypted records and $\mathcal{DS}_2$ depends solely on $(\text{SK}_1, \ldots, \text{SK}_n)$ (but not on the content of $r_1, \ldots, r_n$). For correctness, $\mathcal{DS}_1$ should contain at least one encrypted copy of each of the records $r_1, \ldots, r_n$. It may also contain additional encryptions of records in the database or encryptions of dummy records (hence, generally $n' \ge n$).

2. The communication sent from $\mathcal{S}$ to $\mathcal{U}$ consists of elements of $\mathcal{DS}_1$ plus information that depends solely on $\mathcal{DS}_2$ (and hence does not depend on $r_1, \ldots, r_n$).

Static and Non-Static: For simplicity of exposition, the above refers more specifically to the *static* atomic case: there are no updates to $\mathcal{D}$ beyond initial setup, and, furthermore, no updates to $\mathcal{DS}$ while queries are made. More generally, we can allow $\mathcal{U}$ and $\mathcal{S}$ in Π_{QUERY} to also take as inputs their current states and output new states. This in particular allows them to modify the query key K and $\mathcal{DS}$, respectively.

[1]More formally, both $\mathcal{U}$, $\mathcal{S}$ also have as input a security parameter.

Non-Storage-Inflating: In the case that $n = n'$ above (and hence the server-side storage contains no additional encryptions of records in the database or dummy records) we say the protocol is *non-storage inflating*. We observe that most of the existing practical systems such as CryptDB and Cipherbase are non-storage-inflating.

Fixed Communication Overhead. In the general (not necessarily atomic) setting we define another class of outsourced database systems we call *fixed communication overhead*. We say that an outsourced database system for $\mathcal{Q}$ has fixed communication overhead if for any database $\mathcal{D}$ of size n there are constants α and β (depending only on the security parameter and n) such that for any sequence of queries $q_1, \ldots, q_k \in \mathcal{Q}$, after Π_{SETUP} is run on $\mathcal{D}$ and Π_{QUERY} is run on q_i for each $i = 1$ to k, the length of communication from $\mathcal{S}$ to $\mathcal{U}$ on the i-th execution of Π_{QUERY} is $\alpha \cdot |q_i(\mathcal{D})| + \beta$.[2]

2.2 Adversarial models

We present generic attacks on the privacy of secure outsourced databases by an honest-but-curious $\mathcal{S}$. Intuitively, we want to guarantee that all $\mathcal{S}$ can learn is some well defined "leakage." As discussed above, this typically includes the pattern of accesses to encrypted records in $\mathcal{DS}_1$, or the number of records retrieved by $\mathcal{U}$ in every execution of Π_{QUERY}. To be more general, we follow a formalization of Chase and Kamara [14] in the context of "structured encryption."

For an outsourced database system Π, assume a fixed database sampling algorithm $\mathsf{databaseGen}$, query sampling algorithm $\mathsf{QueryGen}$, leakage functions $\mathcal{L}_{\text{SETUP}}, \mathcal{L}_{\text{QUERY}}$, and simulator Sim. Consider the following experiments[3]:

Real Experiment: Sample $\mathcal{D} \leftarrow \mathsf{databaseGen}$ where $\mathcal{D} = \{(r_1, \text{SK}_1), \ldots, (r_n, \text{SK}_n)\}$ and run $\Pi_{\text{SETUP}}(\mathcal{D}, \perp)$. Then, repeat the following until $\mathcal{S}$ halts: Sample $q \leftarrow \mathsf{QueryGen}$ and run $\Pi_{\text{QUERY}}(q, \mathcal{DS})$. The output of the experiment is the output of $\mathcal{S}$.

Ideal Experiment: Sample $\mathcal{D} \leftarrow \mathsf{databaseGen}$ where $\mathcal{D} = \{(r_1, \text{SK}_1), \ldots, (r_n, \text{SK}_n)\}$ and give $\mathcal{L}_{\text{SETUP}}(\mathcal{D})$ to Sim. Then, repeat the following until Sim halts: Sample $q \leftarrow \mathsf{QueryGen}$, run $\mathcal{L}_{\text{QUERY}}(q, \text{SK}_1, \ldots, \text{SK}_n)$, and give the result to Sim. The output of the experiment is the output of Sim.

DEFINITION 2.2. *We say that outsourced database system Π is $(\mathcal{L}_{\text{SETUP}}, \mathcal{L}_{\text{QUERY}})$-secure if there is a simulator Sim such that for any $\mathsf{databaseGen}, \mathsf{QueryGen}$ the output distributions of the above experiments are computationally indistinguishable.*

Above $\mathcal{L}_{\text{SETUP}}$ is called the "setup leakage" and $\mathcal{L}_{\text{QUERY}}$ is called the "query leakage." We identify the fundamental leakage channels of outsourced database systems as special cases of the query leakage.

Access Pattern Leakage: In the case of $\mathcal{L}_{\text{QUERY}}$ for an *atomic* outsourced database system, we define the special case $\mathcal{L}_{\text{ACCESS}}$ (called "access pattern leakage") that outputs a subset of $S \subseteq [n']$ corresponding to indices in $\mathcal{DS}_1 = (c_1, \ldots, c_{n'})$ encrypting a record matching the query, i.e., S contains exactly those indices $i \in [n']$ for which c_i is an encryption of some r_j such that $r_j \in q(\mathcal{D})$.

Communication Volume Leakage: In the general (not necessarily atomic) setting, we also define the special case $\mathcal{L}_{\text{COMM}}$ (called "communication volume leakage") that outputs $|q(\mathcal{D})|$. Note that in the case of an atomic outsourced database system, this corresponds to $|S|$ above.

Leakage of atomic systems and fixed communication overhead. We observe that static, atomic, and non-storage-inflating outsourced database systems leak the *access pattern*, i.e., their query leakage includes $\mathcal{L}_{\text{ACCESS}}$. This includes practical systems based on searchable symmetric encryption or on deterministic and order-preserving encryption. Similarly, outsourced database protocols with fixed communication overhead leak the *communication volume*, i.e., their query leakage includes $\mathcal{L}_{\text{COMM}}$. This includes "full-fledged" protocols based on FHE or ORAM in the natural way.

REMARK 2.3. *While some specific implementations of outsourced database systems may also leak to $\mathcal{S}$ information about the query q and $\text{SK}_1, \ldots, \text{SK}_n$, aiming for generality we ignore this additional leakage in our attacks.*

2.3 Reconstruction attacks

We will be mostly interested in *reconstruction attacks* [18] (a.k.a. *blatant non-privacy*) on outsourced database systems. In a reconstruction attack, an adversary exploits leakage to recover the search keys. As it is possible to encrypt the database records with a semantically secure encryption scheme, a reconstruction attack results in the maximum information an attacker could learn about the database. The existence of a reconstruction attack hence demonstrates a complete failure of the outsourced database system to keep the dataset private beyond what is achieved by storing the encrypted records with cleartext indexing information.[4] Our goal will be to demonstrate the existence of reconstruction attacks with a weak adversary: passive, with no prior knowledge about the dataset, and with no ability to decipher the queries issued by $\mathcal{U}$.

Namely, consider the ideal experiment where Sim outputs a guess $\{\hat{\text{SK}}_1, \ldots, \hat{\text{SK}}_n\}$.

DEFINITION 2.4. *Outsourced database system Π is said to be (α, β)-reconstructible w.r.t. $\mathsf{databaseGen}$, $\mathsf{QueryGen}$, $\mathcal{L}_{\text{SETUP}}$, $\mathcal{L}_{\text{QUERY}}$ if Sim's output $\{\hat{\text{SK}}_1, \ldots, \hat{\text{SK}}_n\}$ satisfies*

$$\Pr_{\mathsf{Sim}}[|\{\text{SK}_1, \ldots, \text{SK}_n\} \,\Delta\, \{\hat{\text{SK}}_1, \ldots, \hat{\text{SK}}_n\}| \leq \alpha n] \geq 1 - \beta,$$

where Δ denotes symmetric set difference. If $\alpha = 0$ and β is inverse polynomial, we say that Π is fully reconstructible *(w.r.t. $\mathsf{databaseGen}$, $\mathsf{QueryGen}$, $\mathcal{L}_{\text{SETUP}}$, $\mathcal{L}_{\text{QUERY}}$) .*

REMARK 2.5. *When our attack algorithms succeed they produce two candidate search key sets one of which is exactly $\{\text{SK}_1, \ldots, \text{SK}_n\}$ and the other is its reflection over the domain $[1, \ldots, N]$, i.e., $\{N - \text{SK}_1, \ldots, N - \text{SK}_n\}$.*

[2]The experiment is somewhat informal here with the inputs of the parties implied. The actual inputs follow the same format as the security definitions presented in Section 2.2.

[3]We leave the search key domain $\mathcal{D}$ and collection of queries $\mathcal{Q}$ supported by Π implicit for readability, and assume that subsequent sampling algorithms output elements from the right sets.

[4]We note, however, that the existence of a reconstruction attack does not always preclude hiding the queries.

Note that the above models the scenario that the adversary possibly knows the *distributions* of either the data or the queries (or both), but does not directly learn the issued queries or their results. Thus, we consider a relatively weak adversarial model for reconstruction attacks. In fact, our main reconstruction attacks only use adversarial knowledge of the distribution of queries and the data can be arbitrary.

In contrast to previous attacks on specific systems [30, 36, 11, 38], our goal is to capture the fundamental leakage rather than the weaknesses of a specific implementation, database, or cryptographic tool.

To summarize, we consider a passive attacker that does not bring her previous domain knowledge (e.g., no assumptions on the data distribution), cannot affect the dataset (e.g., by injecting records), and does not directly query the database. We assume that the adversary knows the domain size, the total number of records, and that the output of QueryGen is uniform. Finally, the adversary can only observe the encrypted answers of queries.

In the sequel, we devise reconstruction attacks on outsourced database systems for range queries. We first consider the leaked access pattern model (Section 3), and then the leaked communication volume model (Section 4).

3. ATTACK USING THE ACCESS PATTERN

We present an attack using access pattern leakage, i.e., assuming the query leakage $\mathcal{L}_{\text{QUERY}}$ includes $\mathcal{L}_{\text{ACCESS}}$. Such leakage is typical for current systems based on deterministic primitives such as order-preserving encryption, or on symmetric searchable encryption.

Attack Overview. Assume that the database consists of n records and $\text{SK}_1, \ldots, \text{SK}_n$ are their (unknown) search keys (i.e., positions in the domain $\mathcal{X} = \{1, \ldots, N\}$). Let $i_1, \ldots, i_n$ be the actual order of $\text{SK}_1, \ldots, \text{SK}_n$. We assume for simplicity that there is at most one record per position, but the attack trivially extends to the general case. The algorithm initially determines its guess for the order of the sets of records, $\hat{i}_1, \ldots, \hat{i}_n$. This could be done using the method of [16], but we give a simpler and more efficient method using our assumption on the query distribution. Namely, the algorithm first samples enough queries so that all subsets of indices that can match a query are returned with high probability. It determines $\hat{i}_1$ by searching the query results for the largest proper subset of the set of all indices, taking $\hat{i}_1$ to be the symmetric difference of the two. Given $\hat{i}_1$, it determines $\hat{i}_2$ by searching the query results for the smallest proper superset of $\hat{i}_1$. In general, given $\hat{i}_1, \ldots, \hat{i}_{j-1}$, the algorithm determines $\hat{i}_j$ by searching the query results for the smallest proper superset of $\hat{i}_1, \ldots, \hat{i}_{j-1}$.

If we have at least one record per a domain position (i.e., no empty domain positions), then, recovering the order suffices for the reconstruction attack. Otherwise, we have to determine the position of each record in the domain, in order to identify which positions are empty. Towards this, after determining the order, the attack determines its guess for the exact positions of the records, $\hat{\text{SK}}_1, \ldots, \hat{\text{SK}}_n$ by exploiting the uniformity of the QueryGen. Specifically, it utilizes the number of queries that include only the records at positions $\text{SK}_{i_1}\text{SK}_{i_2}, \ldots, \text{SK}_{i_j}$, which is unique for each domain position. Interestingly, this phase of the attack does not use the order of the records beyond the first record, but we include order recovery because it suffices in the case that the records are dense in the domain. The pseudocode and detailed description of the attack are in Appendix B.

Next, we show the correctness of the attack and bounds on the number of required queries p_1 for recovering the order, and p_2 for recovering the positions.

Main Result. To showcase the attack, we prove the following theorem.

THEOREM 3.1. *Let* Π *be an outsourced database system for range queries with access pattern leakage. Then,* Π *is fully reconstructible wrt.* databaseGen, QueryGen, $\mathcal{L}_{\text{SETUP}}$, *and* $\mathcal{L}_{\text{ACCESS}}$, *where the output of* QueryGen *is* uniform, *and* databaseGen *and* $\mathcal{L}_{\text{SETUP}}$ *are arbitrary. The reconstruction algorithm requires* $O(N^4 \log N)$ *queries.*

We note that if the records are *dense* in the domain, recovering only the order suffices for a reconstruction attack, in which case we require only $O(N^2 \log N)$ queries. This is why we include full order recovery in the attack.

PROOF. The theorem follows via Claims 3.2 and 3.3. □

CLAIM 3.2. *Let* $p_1 = O(N^2 \log N)$. *Then, an execution of* $\mathsf{GetOrder}^{\mathcal{O}}$ *(see Appendix B) returns the correct output (up to reflection) with inverse polynomial probability.*

PROOF. The claim follows by the coupon collector's problem. □

CLAIM 3.3. *Let* $p_2 = O(N^4 \log N)$. *Then, assuming* i_1 *is correctly recovered (up to reflection), an execution of* $\mathsf{GetDist}^{\mathcal{O}}$ *(see Appendix B) returns the correct output (up to reflection) with inverse polynomial probability.*

PROOF. See Appendix C. □

A Matching Lowerbound. To conclude this section, we show the optimality of our attack in terms of the number of queries that need to be observed.

THEOREM 3.4. *Assume uniform output of* QueryGen. *There exists a distribution* databaseGen *such that no outsourced database system for range queries that leaks the access pattern is fully reconstructable w.r.t.* databaseGen, QueryGen, $\mathcal{L}_{\text{SETUP}}$, $\mathcal{L}_{\text{ACCESS}}$, *with* $O(N^4)$ *queries, where* $\mathcal{L}_{\text{SETUP}}$ *is arbitrary.*

PROOF. The proof derives directly from Claim D.1 in the Appendix D □

4. ATTACK USING THE COMMUNICATION VOLUME

Access pattern attacks can be avoided using cryptographic tools such as Oblivious RAM and Fully Homomorphic Encryption, and it is tempting conclude that such measures suffice to prevent reconstruction attacks against outsourced database systems. However, here we show that such an attack assuming that the query leakage $\mathcal{L}_{\text{QUERY}}$ includes $\mathcal{L}_{\text{COMM}}$.

Attack Overview. Let n be the total number of records. We label them according to their order in the domain as

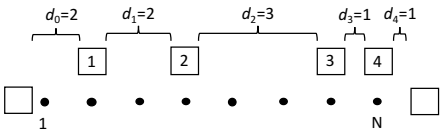

Figure 1: Dataset example

record $1, 2, \ldots, n$, and their positions as $1 \leq \text{SK}_1 \leq \text{SK}_2 \leq \cdots \leq \text{SK}_n \leq N$. Define

$$d_i = \begin{cases} \text{SK}_1 & i = 0 \\ \text{SK}_{i+1} - \text{SK}_i & 1 \leq i \leq n-1 \\ N - \text{SK}_n + 1 & i = n \end{cases}$$

(Equivalently, extend the range to also include the points 0 and $N+1$ and assume two additional "fake" records on these locations, i.e., $\text{SK}_0 = 0$ and $\text{SK}_{n+1} = N+1$ and define $d_i = \text{SK}_{i+1} - \text{SK}_i$). Figure 1 shows an example for $N = 8$ and $n = 4$ where dots represent the domain positions, and each box represents a record. The positions of the records are $\text{SK}_1 = 2$, $\text{SK}_2 = 4$, $\text{SK}_3 = 7$, and $\text{SK}_4 = 8$, and we compute $d_0 = \text{SK}_1 = 2$, $d_1 = \text{SK}_2 - \text{SK}_1 = 2$, $d_2 = \text{SK}_3 - \text{SK}_2 = 3$, $d_3 = \text{SK}_4 - \text{SK}_3 = 1$, and $d_4 = 8 - \text{SK}_4 + 1 = 1$.

There are exactly $\binom{N}{2} + N = N(N+1)/2$ distinct interval queries $q_{[a,b]}$ where $1 \leq a \leq b \leq N$. We first determine the number of distinct queries that return $0 < i \leq n$ records:

$$u_i = \begin{array}{l}\text{number of distinct queries (out of } \binom{N}{2} + N) \\ \text{that return } i \text{ records.}\end{array}$$

Similarly, let

$$\hat{u}_0 = \begin{array}{l}\text{number of distinct queries (out of } \binom{N}{2} + N) \\ \text{that return no records.}\end{array}$$

Then, we can express $u_1, \ldots, u_n$ and $\hat{u}_0$ in terms of the values d_i. Specifically,

$$d_0 \cdot d_n = u_n.$$

For example, if $d_0 = 2$ and $d_n = 1$ as in Figure 1, then two queries ($q_{[1,N]}$ and $q_{[2,N]}$) out of the $\binom{N}{2} + N$ would include these two (and hence all) records. Similarly, we get

$$\begin{aligned} d_0 \cdot d_{n-1} + d_1 \cdot d_n &= u_{n-1} \\ d_0 \cdot d_{n-2} + d_1 \cdot d_{n-1} + d_2 \cdot d_n &= u_{n-2} \\ &\vdots \\ d_0 \cdot d_1 + d_1 \cdot d_2 + \cdots + d_{n-1} \cdot d_n &= u_1. \end{aligned}$$

The number of queries $\hat{u}_0$ that return no records is a special case, and we add the equation

$$d_0^2 + d_1^2 + \ldots + d_n^2 = 2 \cdot \hat{u}_0 + N + 1.$$

By setting $u_0 = 2 \cdot \hat{u}_0 + N + 1$, we get the following system of $n+1$ quadratic equations over the non-negative integers:

$$\sum_{i=0}^{m} d_i \cdot d_{n-(m-i)} = u_{n-m} \text{ for } m \in [0, n]. \quad (1)$$

We now show how to solve this system of quadratic equations. Consider the polynomial

$$d(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_n x^n$$

and its "mirror" polynomial

$$d^R(x) = d_n + d_{n-1} x + d_{n-2} x^2 + \cdots + d_0 x^n,$$

and define

$$F(x) = d(x) \cdot d^R(x).$$

A crucial observation is that the coefficients of $F(x)$ are $u_0, \ldots, u_n$:

$$F(x) = u_n x^{2n} + u_{n-1} x^{2n-1} + \cdots + u_0 x^n + \cdots + u_{n-1} x + u_n.$$

I.e., we can construct $F(x)$ by determining the values u_i based on the query answers volume, and, furthermore, factoring $F(x)$ into two polynomials with non-negative integer coefficients $\hat{d}(x)$ and its "mirror" $\hat{d}^R(x)$ provides a solution to the system 1.

If $F(x)$ uniquely factors into two polynomials with non-negative integer coefficients then these would be $d(x)$ and $d^R(x)$ and hence $u_0, \ldots, u_n$ suffice for exact reconstruction. In practice, one can use algorithms for factoring polynomials with integer coefficients, e.g., the LLL algorithm [34].

Figure 2 depicts the resulting reconstruction algorithm. Initially, $\mathcal{R}^{\mathcal{O}}$ constructs the vector $\mathbf{u}$ ($\mathcal{R}^{\mathcal{O}}$:line 1) by executing $\mathsf{ConstructU}^{\mathcal{O}}$. The latter retrieves p (the value of p is discussed later) uniformly drawn queries from the oracle $\mathcal{O}$ ($\mathsf{ConstructU}^{\mathcal{O}}$:lines 1-2). $\mathcal{O}$ essentially draws a random query q, and returns the size of the answer (i.e., the communication volume leakage). Next, $\mathsf{ConstructU}^{\mathcal{O}}$ determines the number of distinct queries that returned a specific number of records, and returns the result ($\mathsf{ConstructU}^{\mathcal{O}}$: lines 3-4). Then, $\mathcal{R}^{\mathcal{O}}$ recovers the d_i's ($\mathcal{R}^{\mathcal{O}}$:line 2) by executing $\mathsf{FactorF}^{\mathcal{O}}$ using $\mathbf{u}$. $\mathsf{FactorF}^{\mathcal{O}}$ initially sets the u_0 value ($\mathsf{FactorF}^{\mathcal{O}}$:line 1), and defines the polynomial $F(x)$ by using as coefficients the values of $\mathbf{u}$ ($\mathsf{FactorF}^{\mathcal{O}}$:line 2). Then, it runs any state-of-the-art polynomial factorization algorithm and stores the factors of $F(x)$ in a list ($\mathsf{FactorF}^{\mathcal{O}}$:line 3). It picks an arbitrary factor from the list as a factor of $d(x)$, and removes it from the list along with its mirror ($\mathsf{FactorF}^{\mathcal{O}}$:lines 5-7). Finally, it checks that the resulting polynomial $d(x)$ has non-negative coefficients and if so returns these coefficients ($\mathsf{FactorF}^{\mathcal{O}}$:lines 8-10).[5] Finally, $\mathcal{R}^{\mathcal{O}}$ uses the d_i's to determine the positions of the records in the domain ($\mathcal{R}^{\mathcal{O}}$:lines 3-6) (recall that each d_i represents the distance of record i from record $i+1$).

Main Result. The attack described above provides full reconstruction when $F(x)$ uniquely factors into irreducible $d(x), d^R(x)$ over the integers. In case the factorization of $F(x)$ results in more than 2 irreducible factors, there may be more than two candidate solutions for the database, and the algorithm picks an arbitrary solution.

REMARK 4.1. *We give an example where a dataset is not uniquely reconstructed. Let $N = 11$ and consider a dataset with $\text{SK}_1 = 1, \text{SK}_2 = 6$. This corresponds to $d(x) = 1 + 5x + 6x^2$ (and $d^R(x) = 6 + 6x + x^2$). We get that $F(x) = d(x) \cdot d^R(x) = 6 + 35x + 62x^2 + 35x^3 + 6x^4$. Over the integers, $F(x)$ factors as $(1+2x)(2+x)(1+3x)(3+x)$. Our attack algorithm may hence choose to recover $d(x) = (1+2x)(3+x) = 3 + 7x + 2x^2$ (and $d^R(x) = (2+x)(1+3x)$) which results in $\text{SK}_1 = 3, \text{SK}_2 = 10$. Note that this is not a reflection of the true dataset.*

We note here that our experiments indicate that for real life databases the factorization is likely to result in two irreducible factors $d(x)$ and $d_R(x)$. Indeed, this was the case for all 6,786 datasets used in our evaluation (Section 5.2 below).

[5] As described, $\mathsf{FactorF}^{\mathcal{O}}$ finds one solution to $F(x) = d(x) \cdot d^R(x)$, it can be modified to find all such solutions.

Algorithm $\mathcal{R}^{\mathcal{O}}$:
/* Construct $\mathbf{c}$ */
1. $\mathbf{u} \leftarrow \mathsf{ConstructU}^{\mathcal{O}}$
/* Recover distances and actual values */
2. $\hat{\mathbf{d}} \leftarrow \mathsf{FactorF}^{\mathcal{O}}(\mathbf{u})$
3. $\hat{\text{sK}}_1 \leftarrow \text{r}[0]$
4. For $j = 2$ to n do:
5. $\quad \hat{\text{sK}}_j \leftarrow \hat{\text{sK}}_{j-1} + \hat{\text{d}}[j-1]$
6. Return $(\hat{\text{sK}}_1, \ldots, \hat{\text{sK}}_n)$

Oracle $\mathcal{O}$:
1. $q \leftarrow\!\!\$\ \mathsf{QueryGen}$
2. $C \leftarrow \mathcal{L}_{\text{COMM}}(q, \text{sK}_1, \ldots, \text{sK}_n)$
3. Return C

Algorithm $\mathsf{ConstructU}^{\mathcal{O}}()$:
1. For $k = 1$ to p do:
2. $\quad U_k \leftarrow\!\!\$\ \mathcal{O}$
3. $\quad \text{u}[U_k] \leftarrow \text{u}[U_k] + 1$
4. Return $\mathbf{u} \cdot \frac{N(N+1)}{2 \cdot p_1}$

Algorithm $\mathsf{FactorF}^{\mathcal{O}}(\mathbf{u})$:
1. Set $\text{u}[0] = 2\text{u}[0] + N + 1$
2. Set $F(x) = \text{u}[n]x^{2n} + \text{u}[n-1]x^{2n-1} + \ldots +$
$\quad +\text{u}[1]x^{n+1} + \text{u}[0]x^n + \text{u}[1]x^{n-1} + \ldots + \text{u}[n]$
3. $factors \leftarrow \text{Factorize}(F(x))$
4. Set $d(x) = 1$ (and $d^R(x) = 1$)
5. For each pair of factors e and reciprocal(e) in $factors$ do
6. $\quad$ Set $d(x) = d(x) \cdot e$ (and $d^R(x) = d^R(x) \cdot$ reciprocal(e))
7. $\quad$ remove e and reciprocal(e) from $factors$
8. If coefficients of $d(x)$ are non-negative
9. $\quad$ Set $\hat{\mathbf{d}}$ as the coefficients of $d(x)$ and return $\hat{\mathbf{d}}$
10. Otherwise fail

Figure 2: Reconstruction algorithm using the communication volume leakage

REMARK 4.2. *Factorization might be slow for large number of records. Thus, we design a simple algorithm (see Appendix E) that checks the possible combinations of d_i's in order to determine the correct values, and is faster than factorization in practice.*

To complete the attack description, we now give a bound on the number of queries needed for estimating the coefficients u[i].

CLAIM 4.3. *Let $p = O(N^4 \log N)$. Then, an execution of* $\mathsf{ConstructU}^{\mathcal{O}}$ *returns the correct u_i's with inverse polynomial probability.*

PROOF. Let $\mathbf{E}[\text{u}[i]]$ be the expected value of u[i]. We draw p queries to ensure that every u[i] lies in the range

$$\text{u}[i] \in [\mathbf{E}[\text{u}[i]] - \epsilon, \ \ \mathbf{E}[\text{u}[i]] + \epsilon]$$

with probability at least $1 - \delta$, for $\epsilon = O(1/N^2)$ and inverse polynomial δ. Using the same steps as in the proof of Theorem 3.1, we get that p is $O(N^4 \log N)$. □

CLAIM 4.4. *An execution of* $\mathsf{FactorF}^{\mathcal{O}}$ *on the correct inputs $u_n, u_{n-1}, \ldots, u_0$ returns the correct outputs $d_0, d_1, \ldots, d_n$ when $d(x)$ is irreducible.*

PROOF. Our algorithm factorizes $F(x)$ (line 3). If $d(x)$ is irreducible over the integers, then the factoring of $F(x)$ results in $d(x)$ and $d^R(x)$. □

Next, we show the optimality of our attack in the number of required queries.

LEMMA 4.5. *Let the output of* QueryGen *be uniform. Then there is a distribution* databaseGen *such that no outsourced database system for range queries that leaks the communication volume is fully reconstructable wrt.* databaseGen, QueryGen, $\mathcal{L}_{\text{SETUP}}$, *and* $\mathcal{L}_{\text{COMM}}$ *with $O(N^4)$ queries, where* $\mathcal{L}_{\text{SETUP}}$ *is arbitrary.*

PROOF. The proof derives directly from Claim D.1 in the Appendix D. □

5. EXPERIMENTS

We implemented and ran our attacks on an Intel Core i7 2.5GHz machine with 16GB of RAM, running MacOS 10.11. Using Parallels, we created two virtual machines running Ubuntu Linux 14.04, each with 2 CPU cores and 4GB of memory. Specifically, our implementation is depicted in Figure 3. We installed mySQL server on the first virtual machine (hereafter called server), and CryptDB on the second (called proxy). We implemented our client in Java and ran it on the proxy. The client stores a database to the server through the CryptDB proxy, and chooses an attribute to be indexed. The CryptDB proxy encrypts each record before storing it to the mySQL server. Then, the user asks range queries on the indexed attribute, the CryptDB proxy retrieves the required encrypted records from the server, decrypts them, and sends them back to the user. Additionally, we implemented a packet sniffer in Java, residing on the server side, which can only observe the network packets from the server to the proxy. In our implementation, the sniffer ignores the communication between the user and the proxy. Finally, the packet sniffer performs our attacks.

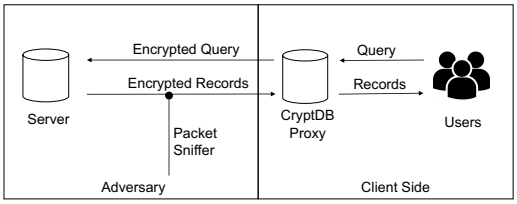

Figure 3: System implementation

In order to evaluate the performance of each attack, we encrypted and uploaded to the server the 518 datasets from the Texas Hospital Inpatient Discharge Public Use Data File of 2008 (PUDF)[6], and the 1049 datasets from the 2009 HCUP Nationwide Inpatient Sample (NIS)[7]. Each dataset represents a specific hospital, and consists of records of hospitalized patients. We indexed different attributes deriving in

[6]http://archive.ahrq.gov/
[7]http://www.hcup-us.ahrq.gov/

total 6,786 databases and ran the attacks on all of them. Both the attack using the access pattern and the attack using the communication volume successfully reconstructed all the search keys in every case. Table 1 summarizes the characteristics of each data source. The datasets and the domains are similar to those in [38].

Table 1: Dataset Characteristics

Source	Datasets	Index	N	n(max)	n(avg)
PUDF	518	Mortality Risk	4	55,605	5,612
		Age (<18)	6	20,454	1,170
		Age ($\geq$ 18)	16	34,162	4,130
		Age (All)	22	50,626	5,300
		Length of Stay	365	55,605	5,612
NIS	1049	Age (<18)	18	16,954	1,195
		Age ($\geq$ 18)	107	106,252	6,240
		Age (All)	125	121,663	7,435
		Length of Stay	365	121,663	7,435

The average number of patients per hospital from the PUDF source is 5,612 (with minimum 1 and maximum 55,605). Each patient record has size 1,486 bytes (1,547 bytes after the encryption). We used 3 attributes of the PUDF datasets as the ordered domains for the range queries, namely Mortality Risk, Age, and Length of Stay. The Mortality Risk has 4 possible values; minor, moderate, major, and extreme. The Age domain consists of 22 different values, each representing an age interval, instead of a specific age in years. We further divided it into two domains; one for patients under the age of 18, and one for adult patients, assuming that some users are only interested in these attribute values. The Length of Stay represents the number of days a patient was hospitalized. Each domain position is a specific number of days, ranging from 0 to 364. Finally, the combinations of search keys and datasets produce essentially 2,590 different databases because for each domain, the records are distributed differently.

The NIS datasets incorporate similar information, with average number of patients equal to 7,435 (minimum 1 and maximum 121,663) and record size of 621 bytes (684 bytes after the encryption). However, there is no Mortality Risk information. As such, we indexed only the Age and Length of Stay attributes. Again, we divided the Age domain into two additional domains for minor and adult patients respectively. NIS has more detailed age information, i.e., each domain position represents a specific age in years, ranging from 0 to 124. The effective databases are in this case 4,196.

We first gathered enough queries in order to be able to run each attack. The user issues uniformly drawn range queries to the proxy. For each query, the proxy retrieves the encrypted records, decrypts them, and sends them back to the user. Then, the client asks the next query. The running time depends on the number of required queries (or in the domain size), the number of records to decrypt, the network speed, and the number of users. For all the datasets, this time varied from some seconds to one month (for retrieving N^4 queries on the large domain with size $N = 365$). For datasets where the required time for executing enough queries exceeded an hour, we simulated the query retrieval. However, we expect that in real life scenarios the system is running for more than several months, and more importantly, that several users are issuing queries. In case of 100 users issuing queries, the packet sniffer would gather enough information 100 times faster (e.g., only a few hours for the largest domain size of $N = 365$), because it can capture the packets of all the users simultaneously. Finally, we note that our attack always succeeds to fully reconstruct the records on the domain.

5.1 Access Pattern Attack

Table 2: Access pattern attack

Source	Index	Ordering	Positions	Dense
PUDF	Mortality Risk	1 ms	1 ms	85%
	Age (<18)	1 ms	1 ms	34.1%
	Age ($\geq$ 18)	1 ms	1 ms	67.3%
	Age (All)	1 ms	1 ms	32.2%
	Length of Stay	43 ms	4.2 sec	0%
NIS	Age (<18)	1 ms	1 ms	31.5%
	Age ($\geq$ 18)	1 ms	202 ms	0%
	Age (All)	1 ms	356 ms	0%
	Length of Stay	5 ms	3.4 sec	0%

In this section we evaluate the running time of the attack using the access pattern leakage. Table 2 summarizes our results. Column Ordering represents the average required time to recover the order of the records using $N^2 \log N$ queries, while Positions depicts the average time needed to recover the positions, requiring N^4 queries. If a dataset is dense, then the attack terminates when it determines the order. The last column represents the percentage of datasets where the targeted domain is dense.

The attack on the PUDF datasets for the Mortality Risk attribute ran in milliseconds. Moreover, 85% of the datasets have dense domain, rendering the ordering sufficient to reconstruct the data, and thus, it only requires to observe $N^2 \log N$ queries. For domains extracted from the age attribute, the running time is similar to that for the Mortality Risk. Recall that the age attribute has a small domain, since the original data report the age range for each patient. 67.3% of datasets that include only the records of adults have dense domains, where for records with $age < 18$ there are 34.1% dense domains, and for all the ages, we have 32.2% dense domains. For the Length of Stay attribute, recovering the order took 43 milliseconds, and the positions 4.2 seconds. In this case, we only encountered sparse domains.

Regarding the NIS datasets, the attack recovered the order in a few milliseconds for all the attributes. However, it was enough for reconstruction only for 31.2% datasets with domain representing ages younger than 18. All the other domains were sparse, and as such, the attack completed in 0.2 to 3.4 seconds.

5.2 Communication Volume Attack

Next, we evaluate our attack utilizing only the communication volume leakage. In order to determine the values of $\mathbf{u}$ array, i.e., the coefficients of the $F(x)$ polynomial, we have retrieved N^4 queries.

After building vector $\mathbf{u}$, we performed the attack using the factorization algorithm and the brute-force version (Appendix E). Due to the large amount of records, the factorization algorithm needed several hours to terminate. As such, we only ran it on datasets with fewer than 150 records. However, our brute-force attack terminated in seconds for the complete datasets in all cases. More importantly, the attack successfully reconstructed all the data, i.e., $d(x)$ was irreducible in all 6,786 databases.

Table 3 summarizes the attack performance. Column Factor shows the average running time of the attack using the factorization algorithm, after constructing the $\mathbf{u}$ vector with

the coefficients of $F(x)$. Column BruteForce depicts the average running time of our brute-force algorithm, showing that in practice it terminates within a few milliseconds.

Table 3: Communication volume attack

Source	Index	Factor ($n \leq 150$)	BruteForce
PUDF	Mortality Risk	11 min	22 ms
	Age (<18)	39 sec	1.7 ms
	Age ($\geq$ 18)	4.3 min	15 ms
	Age (All)	4.1 min	390 ms
	Length of Stay	3 min	22 ms
NIS	Age (<18)	3.3 min	2 ms
	Age ($\geq$ 18)	6 min	34 ms
	Age (All)	5.1 min	189 ms
	Length of Stay	4 min	44 ms

The reconstruction attack using the factorization method runs in a few minutes for almost all the settings, since it was executed only for datasets with fewer than 150 records. For larger datasets we observed that it required hours and hence, we omit the results. On the other hand, the running time of the brute force attack is much shorter (runs in milliseconds), and depends mainly on the number of records. Interestingly, the attack is slower when reconstructing the age attribute than the Length of Stay, although the number of records are almost the same. This is due to the fact that the distribution of the Length of Stay values is highly skewed, with the majority of records concentrated in the first few domain positions. As such, the brute force algorithm can prune values of d_i's that correspond to more uniformly distributed data early on, resulting in fewer candidate datasets to check.

6. REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *SIGMOD*, 2004.

[2] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. In *CIDR*, 2013.

[3] A. Arasu, K. Eguro, R. Kaushik, and R. Ramamurthy. Querying encrypted data (tutorial). In *ICDE*, 2013.

[4] S. Bajaj and R. Sion. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *TKDE*, 26(3):752–765, 2014.

[5] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, 2007.

[6] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46(8):1853–1859, 1967.

[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.

[8] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, 2004.

[10] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, 2005.

[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS*, 2015.

[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*. 2013.

[13] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.

[14] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, 2010.

[15] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, 2006.

[16] J. L. Dautrich Jr and C. V. Ravishankar. Compromising privacy in precise query protocols. In *EDBT*, 2013.

[17] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In *SIGMOD*, 2016.

[18] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[19] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: a scalable private dbms. In *S&P*, pages 395–410, 2015.

[20] C. Gentry. Computing arbitrary functions of encrypted data. *CACM*, 53(3):97–105, 2010.

[21] C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[22] E.-J. Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[23] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC*, 1987.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[25] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *JACM*, 43(3):431–473, 1996.

[26] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[27] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[28] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *VLDBJ*, 21(3):333–358, 2012.

[29] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, 2004.

[30] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[31] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In *CODASPY*, 2014.

[32] S. Kamara. How to search on encrypted data, 2015. https://cs.brown.edu/~seny/slides/encryptedsearch-full.pdf.

[33] S. Kamara and T. Moataz. Sql on

structurally-encrypted databases. Cryptology ePrint Archive, Report 2016/453, 2016. http://eprint.iacr.org/.

[34] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[35] J. Li and E. R. Omiecinski. Efficiency and security trade-off in supporting range queries on encrypted databases. pages 69–83, 2005.

[36] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014.

[37] M. Naveed. The fallacy of composition of oblivious ram and searchable encryption. Cryptology ePrint Archive, Report 2015/668, 2015.

[38] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *CCS*, 2015.

[39] O. Pandey and Y. Rouselakis. Property preserving symmetric encryption. In *EUROCRYPT*, 2012.

[40] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *S&P*, 2014.

[41] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *SP*, pages 463–477, 2013.

[42] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *SOSP*, 2011.

[43] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, 2009.

[44] E. Shi, J. Bethencourt, T.-H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *SP*, 2007.

[45] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *SP*, 2000.

[46] V. Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *FOCS*, 2011.

[47] A. C. Yao. Protocols for secure computations. In *SFCS*, 1982.

APPENDIX

A. TAIL INEQUALITIES

We use the following tail inequalities:

THEOREM A.1 (CHERNOFF-HOEFFDING INEQUALITIES). *Let $X_1, \ldots, X_n$ be independent random variables such that $X_i \in [0,1]$ and $\mathbf{E}[X_i] = \mu$ and denote $S = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then, for all $\epsilon > 0$:*

$$\textit{Hoeffding:} \quad \Pr[S > \mu + \epsilon] \leq e^{-2n\epsilon^2} \text{ and } \Pr[S < \mu - \epsilon] \leq e^{-2n\epsilon^2}. \tag{2}$$

$$\textit{Chernoff:} \quad \Pr[S > (1+\epsilon)\mu] \leq e^{-n\mu\epsilon^2/3} \text{ and } \Pr[S < (1-\epsilon)\mu] \leq e^{-n\mu\epsilon^2/2}. \tag{3}$$

THEOREM A.2 (AZUMA'S INEQUALITY). *Let X_i be a martingale satisfying $|X_i - X_{i-1}| < \Delta$. Then, for all $\epsilon > 0$*

$$\Pr[|X_i - X_0| \geq \epsilon] \leq 2e^{-\epsilon^2/2i\Delta^2}.$$

We get the following corollary:

$$\Pr[|X_i - X_0| \geq \sqrt{i} \cdot t \cdot \Delta] \leq 2e^{-t^2/2}. \tag{4}$$

B. PSEUDOCODE OF THE ATTACK USING THE ACCESS PATTERN

Figure 4 shows the pseudocode of the attack. It initially retrieves the order of the records by calling $\mathsf{GetOrder}^{\mathcal{O}}$ ($\mathcal{R}^{\mathcal{O}}$:line 1). $\mathsf{GetOrder}^{\mathcal{O}}$ retrieves p_1 uniformly drawn queries ($\mathsf{GetOrder}^{\mathcal{O}}$:line 1) from the oracle $\mathcal{O}$. The latter essentially chooses a random range query q, determines the access pattern leakage, and returns it. Then, $\mathsf{GetOrder}^{\mathcal{O}}$ searches the query results for the largest proper subset L' of the set of all indices $[n]$ ($\mathsf{GetOrder}^{\mathcal{O}}$:line 2), and sets $\hat{i}_1$ as the symmetric difference of the two ($\mathsf{GetOrder}^{\mathcal{O}}$:line 3). Next, for each subsequent record j ($\mathsf{GetOrder}^{\mathcal{O}}$:line 4), it determines $\hat{i}_j$ by searching the query results for the smallest proper superset of $\hat{i}_1, \ldots, \hat{i}_{j-1}$ $\mathsf{GetOrder}^{\mathcal{O}}$:lines 5-6. Finally, it returns the ordering ($\mathsf{GetOrder}^{\mathcal{O}}$:line 7).

After recovering the order, $\mathcal{R}^{\mathcal{O}}$ checks if there are no empty positions in the domain ($\mathcal{R}^{\mathcal{O}}$:line 2), and if this is the case, the attack terminates by returning the ordering ($\mathcal{R}^{\mathcal{O}}$:line 3). Otherwise, it continues to recover the positions ($\mathcal{R}^{\mathcal{O}}$:line 5). $\mathsf{GetDist}^{\mathcal{O}}$ initially retrieves p_2 uniformly drawn queries ($\mathsf{GetDist}^{\mathcal{O}}$:line 1) from the oracle $\mathcal{O}$. Then, it determines the first value of A, i.e., number of queries that include the record that is first in the ordering ($\mathsf{GetDist}^{\mathcal{O}}$:line 2), and value B, which represents the number of queries that include the record that is last in the ordering ($\mathsf{GetDist}^{\mathcal{O}}$:line 3). Then, it sets as ordering either the original one or its reverse so that the record which is closest to a domain endpoint comes first[8], and it finds the position $\widehat{\mathrm{SK}}_{\hat{i}_1}$ of the first record up to reflection ($\mathsf{GetDist}^{\mathcal{O}}$:lines 4-8). Next, it resamples the queries and for every other record in the ordering ($\mathsf{GetDist}^{\mathcal{O}}$:line 9), it discovers the value A, i.e., the number of queries that include the previously determined records and the next one in the ordering ($\mathsf{GetDist}^{\mathcal{O}}$:line 11). Finally, after computing all the positions by utilizing A ($\mathsf{GetDist}^{\mathcal{O}}$:line 12), the algorithm returns the positions of the records ($\mathsf{GetDist}^{\mathcal{O}}$:line 13).

C. PROOF OF CLAIM 3.3

PROOF. For $i \leq j \leq m$, let E_j be the event that SK_{i_j} is correctly recovered; here we allow the position to be recovered up to reflection, as long as the reflection is consistent for all the positions. Our goal is to set p_2 to ensure that all of the E_j's occur together with inverse polynomial probability. Using a union bound

$$\Pr\left[\bigwedge_{j=1}^{n} E_j\right] \geq \Pr[E_1] \cdot \left(1 - \left(\sum_{j=2}^{n} \Pr\left[\overline{E_j} \mid E_1\right]\right)\right) \tag{5}$$

For $1 \leq j \leq n$, let A_j be the random variable counting the number of queries that match the records with positions $\mathrm{SK}_{i_1}, \mathrm{SK}_{i_2}, \ldots, \mathrm{SK}_{i_j}$. We first show how to set p_2 to

[8]This procedure ensures that we receive the correct positions, up to reflection, when all the records lie on either the first or second half of the domain.

Algorithm $\mathcal{R}^{\mathcal{O}}$:
1. $(\hat{i}_1, \ldots, \hat{i}_n) \leftarrow\$ \mathsf{GetOrder}^{\mathcal{O}}$
2. If $n = N$
3. Return $(\hat{i}_1, \ldots, \hat{i}_N)$
4. Else
5. $(\hat{\text{SK}}_{\hat{i}_1}, \ldots, \hat{\text{SK}}_{\hat{i}_n}) \leftarrow\$ \mathsf{GetDist}^{\mathcal{O}}(\hat{i}_1, \ldots, \hat{i}_n)$
6. Return $(\hat{\text{SK}}_{\hat{i}_1}, \ldots, \hat{\text{SK}}_{\hat{i}_n})$

Oracle $\mathcal{O}$:
1. $q \leftarrow\$ \mathsf{QueryGen}$
2. $L \leftarrow \mathcal{L}_{\text{ACCESS}}(q, \text{SK}_1, \ldots, \text{SK}_n)$
3. Return L

Algorithm $\mathsf{GetOrder}^{\mathcal{O}}$:
1. For $k = 1$ to p_1 do: $L_k \leftarrow\$ \mathcal{O}$
2. Let L' be a query result s.t. $|L'| = n - 1$
3. $\hat{i}_1 \leftarrow [n] \setminus L'$
4. For $j = 2$ to n do:
5. Let L' be a query result of the form $L' = \{\hat{i}_1, \ldots, \hat{i}_{j-1}\} \cup \{k\}$
6. $\hat{i}_j \leftarrow k$
7. Return $(\hat{i}_1, \ldots, \hat{i}_n)$

Algorithm $\mathsf{GetDist}^{\mathcal{O}}(\hat{i}_1, \ldots, \hat{i}_n)$
1. For $k = 1$ to p_2 do: $L_k \leftarrow\$ \mathcal{O}$
   /* Determine first point */
2. Let $A = \{L_k \mid 1 \le k \le p_2, \hat{i}_1 \in L_k\}$
3. Let $B = \{L_k \mid 1 \le k \le p_2, \hat{i}_n \in L_k\}$
4. If $A < B$
5. $\hat{\text{SK}}_{\hat{i}_1} \leftarrow \min_z, z \in \text{argmin}_z \, |A/p_2 - 2z(N - z + 1)/(N(N + 1))|$
6. Else
7. $Reverse(\{\hat{i}_1, \ldots, \hat{i}_n\})$
8. $\hat{\text{SK}}_{\hat{i}_1} \leftarrow \min_z, z \in \text{argmin}_z \, |B/p_2 - 2z(N - z + 1)/(N(N + 1))|$
   /* Two possible z values above, take the smallest */
9. For $k = 1$ to p_2 do: $L_k \leftarrow\$ \mathcal{O}$
10. For $j = 2$ to n do:
    /* Determine j-th point using first point */
11. Let $A = \{L_k \mid 1 \le k \le p_2, \{\hat{i}_1 \cup \hat{i}_2 \cup \ldots \cup \hat{i}_j\} \subseteq L_k\}$
12. $\hat{\text{SK}}_{\hat{i}_j} \leftarrow \text{argmin}_z \, |A/p_2 - 2\hat{\text{SK}}_{\hat{i}_1}(N - z + 1)/(N(N + 1))|$
13. Return $(\hat{\text{SK}}_{\hat{i}_1}, \ldots, \hat{\text{SK}}_{\hat{i}_n})$

Figure 4: Reconstruction algorithm using the access pattern leakage.

ensure that E_1 happens with inverse polynomial probability. According to $\mathsf{GetDist}^{\mathcal{O}}$:line 3, to ensure E_1 happens with inverse polynomial probability, we want that

$$A_1/p_2 \in \left[\frac{2\text{SK}_{i_1}(N - \text{SK}_{i_1} + 1)}{N(N+1)} \pm \epsilon\right]$$

with probability at least $1 - \delta$, for $\epsilon = O(1/N^2)$ and inverse polynomial δ. To see that $\epsilon = O(1/N^2)$ suffices, note that we want that for choices of z different from SK_{i_1} and its reflection

$$\left|\frac{2\text{SK}_{i_1}(N - \text{SK}_{i_1} + 1)}{N(N+1)} - \frac{2z(N - z + 1)}{N(N+1)}\right|$$

is $\Omega(1/N^2)$. But this follows from the fact that the absolute value of the numerator is at least 1 in this case. Now using Hoeffding's inequality (Equation 2 in Section A) we have that

$$\Pr\left[A_1/p_2 > \frac{2\text{SK}_{i_1}(N - \text{SK}_{i_1} + 1)}{N(N+1)} + \epsilon\right] \le e^{-2p_2\epsilon^2}$$

and similarly for the lower bound. Above, to calculate the expectation of A_1, we use the fact that the probability that a uniformly random query $q_{[a,b]}$ satisfies $\text{SK}_{i_1} \in [a, b]$ is

$$\frac{2\text{SK}_{i_1}(N - \text{SK}_{i_1} + 1)}{N(N+1)}$$

since for any $a \le \text{SK}_{i_1}$ there are $N - \text{SK}_{i_1} + 1$ values for b satisfying $\text{SK}_{i_1} \in [a, b]$. Thus, we set $e^{-2p_2\epsilon^2} = \delta$. Solving for p_2 we get $p_2 = O(N^4 \log N)$ as desired.

A similar argument shows that when $p_2 = O(N^4 \log N)$ the probability of $\overline{E_j}$ given E_1 is inverse polynomial probability for each $j \ge 2$, using the fact that the subsequent p_2 queries after the first p_2 queries are independently sampled ($\mathsf{GetDist}^{\mathcal{O}}$:line 4). In the argument we can take say $\delta = O(1/N^2)$ (instead of simply any inverse polynomial as above) so that the overall expression in Equation 5 is inverse polynomial as desired. □

D. OPTIMALITY OF OUR ATTACKS

Here we show that our reconstruction attacks are nearly optimal. Namely, we show that for an outsourced database system for range queries, there are two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ such that an adversary needs to observe $\Omega(N^4)$ uniformly chosen queries to distinguish whether $\mathcal{D}_1$ or $\mathcal{D}_2$ is outsourced. This bound holds for either $\mathcal{L}_{\text{ACCESS}}$ or $\mathcal{L}_{\text{COMM}}$ leakage functions.

THEOREM D.1. *Let the output of* QueryGen *be uniform. Then there is a distribution* databaseGen *such that no outsourced database system for range queries is fully reconstructable w.r.t.* databaseGen, QueryGen, $\mathcal{L}_{\text{SETUP}}$, $\mathcal{L}_{\text{QUERY}}$ *with $O(N^4)$ queries, where $\mathcal{L}_{\text{SETUP}}$ is arbitrary and $\mathcal{L}_{\text{QUERY}}$ includes $\mathcal{L}_{\text{COMM}}$.*

PROOF. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be equal-sized databases where in $\mathcal{D}_1$ all records lie at position $\text{SK}_1 = (N+1)/2$, and in $\mathcal{D}_2$ all records lie at $\text{SK}_2 = \text{SK}_1 + 1$ (assume N is odd). With both databases, an adversary can observe see exactly two types of queries: those that return all records record and those that return no records.

The total number of different queries is $T = N(N+1)/2$. In case of $\mathcal{D}_1$ the number of non-empty queries is $T_1 = \frac{(N+1)^2}{4}$ and hence the probability of observing a non-empty query is $p = T_1/T$. In case of $\mathcal{D}_2$ the probability to receive a non-empty query $p + \delta$ where $\delta = 1/T$.

Consider a setting where the database is chosen to be $\mathcal{D}_1$ or $\mathcal{D}_2$ with equal probability before the adversary begin observing queries. The adversary's a priory belief is that the database is $\mathcal{D}_1$ or $\mathcal{D}_2$ with equal probability. After observing each query answer, the adversary updates her belief. Let $\Pr[\mathcal{D}_1|a_1, a_2, \ldots, a_i]$ be the posterior probability the database is $\mathcal{D}_1$, and $\Pr[\mathcal{D}_2|a_1, a_2, \ldots, a_i]$ be the posterior probability the database is $\mathcal{D}_2$, after observing i queries.

Writing the log-ratio of the posterior probabilities and using Bayes rule we can describe how the adversary's confi-

Algorithm $\mathcal{R}^{\mathcal{O}}$:
/* Construct $\mathbf{u}$ */
1. $\mathbf{u} \leftarrow \mathsf{ConstructU}^{\mathcal{O}}$
/* Recover distances and actual values */
2. $\mathbf{d} \leftarrow \mathsf{ConstructD}^{\mathcal{O}}(\mathbf{u}, NULL, 0)$
3. $\hat{\text{SK}}_1 \leftarrow \text{d}[0]$
4. For $j = 2$ to n do:
5. $\quad \hat{\text{SK}}_j \leftarrow \hat{\text{SK}}_{j-1} + \text{d}[j-1]$
6. Return $(\hat{\text{SK}}_1, \ldots, \hat{\text{SK}}_n)$

Oracle $\mathcal{O}$:
1. $q \leftarrow\!\!\$\ \mathsf{QueryGen}$
2. $C \leftarrow \mathcal{L}_{\text{COMM}}(q, \text{SK}_1, \ldots, \text{SK}_n)$
3. Return C

Algorithm $\mathsf{ConstructU}^{\mathcal{O}}()$:
1. For $k = 1$ to p do:
2. $\quad U_k \leftarrow\!\!\$\ \mathcal{O}$
3. $\quad \text{u}[U_k] \leftarrow \text{u}[U_k] + 1$
4. Return $\mathbf{u} \cdot \frac{N(N+1)}{2 \cdot p_1}$

Algorithm $\mathsf{ConstructD}^{\mathcal{O}}(\mathbf{u}, \mathbf{d}, m)$:
1. If $m = n$ and $\sum_{i=0}^{m} \text{d}[i] = N + 1$
2. $\quad$ Return $\mathbf{d}$
3. If $m \leq n/2$
4. $\quad$ For $k = 1$ to $N/2$ do
5. $\quad\quad$ For $l = 1$ to $N - k + 1$ do
6. $\quad\quad\quad \text{d}[m] \leftarrow k$
7. $\quad\quad\quad \text{d}[n-m] \leftarrow l$
8. $\quad\quad\quad$ If $\sum_{i=0}^{m} (\text{d}[i] \cdot \text{d}[n-(m-i)]) = \text{u}[n-m]$
9. $\quad\quad\quad\quad \mathsf{ConstructD}^{\mathcal{O}}(\mathbf{u}, \mathbf{d}, m+1)$
10. Else
11. $\quad$ If $\sum_{i=0}^{m} (\text{d}[i] \cdot \text{d}[n-(m-i)]) = \text{u}[n-m]$
12. $\quad\quad \mathsf{ConstructD}^{\mathcal{O}}(\mathbf{u}, \mathbf{d}, m+1)$

Figure 5: Brute-force reconstruction algorithm

dence whether the database is $\mathcal{D}_1$ or $\mathcal{D}_2$ evolves:

$$\begin{aligned} C_i &= \log \frac{\Pr[\mathcal{D}_1 | a_1, a_2, \ldots, a_i]}{\Pr[\mathcal{D}_2 | a_1, a_2, \ldots, a_i]} \\ &= \log \left(\frac{\Pr[\mathcal{D}_1 | a_1, a_2, \ldots, a_{i-1}]}{\Pr[\mathcal{D}_2 | a_1, a_2, \ldots, a_{i-1}]} \cdot \frac{\Pr[a_i | \mathcal{D}_1]}{\Pr[a_i | \mathcal{D}_2]} \right) \\ &= C_{i-1} + \log \left(\frac{\Pr[a_i | \mathcal{D}_1]}{\Pr[a_i | \mathcal{D}_2]} \right) = C_{i-1} + step_i. \end{aligned}$$

Thus, $C_0, C_1, \ldots$ describes a random walk on the real line where $C_0 = 0$ and an empty observed query corresponds to a (positive) $step_i = -\log\left(1 - \frac{\delta}{1-p}\right)$ and a non-empty observed query corresponds to a (negative) $stepI = -\log\left(1 + \frac{\delta}{p}\right)$. We analyze this random walk in the case where the database is $\mathcal{D}_1$ to show a lowerbound on the number of steps needed to reach $C_i > C$ for constant C. (A similar analysis holds when the database is $\mathcal{D}_2$.) We get

$$step_i = \begin{cases} -\log\left(1 + \frac{\delta}{p}\right) & \text{w.p. } p \\ -\log\left(1 - \frac{\delta}{1-p}\right) & \text{w.p. } 1-p \end{cases}$$

Using

$$\log\left(1 + \frac{\delta}{p}\right) \in \left[\frac{\delta}{p} - \frac{\delta^2}{p^2}, \frac{\delta}{p}\right] \quad (6)$$

$$\log\left(1 - \frac{\delta}{1-p}\right) \in \left[\frac{-\delta}{1-p} - \frac{\delta^2}{(1-p)^2}, \frac{-\delta}{1-p}\right] \quad (7)$$

we get that $0 \leq \text{E}[step_i] \leq \frac{\delta^2}{p(1-p)} \leq \frac{16}{N^4}$. Hence, $0 \leq \text{E}[C_i] \leq \frac{16i}{N^4}$. Note that $C_i - \text{E}[C_i]$ is a martingale with $C_0 = \text{E}[C_i] = 0$ and $\Delta = \frac{8}{N^2}$. Using Equation (4) we get that $\Pr[|C_i - \text{E}[C_i]| > 64\sqrt{i} \log N / N^2] \leq e^{-\log^2 N/2}$. We conclude that, except with negligible probability, $C_i < C$ unless $i = \tilde{\Omega}(N^4)$. □

E. BRUTE-FORCE ALGORITHM UTILIZING THE COMMUNICATION VOLUME LEAKAGE

The factorization procedure for our attack using the communication volume leakage may be slow for large datasets. This is due to the fact that factorizing a polynomial takes $O(n^3)$ time (e.g., Berlekamp algorithm [6]), where n is the degree of the polynomial, i.e, the number of records in our setting. As such, we design a brute force algorithm that checks all the possible combinations of d_i's in order to determine the correct values, while pruning values that cannot be the answer.

Specifically, algorithm $\mathcal{R}^{\mathcal{O}}$ in Figure 5 determines all the d_i's, which are used to determine all the SK_i's as follows. Initially, it constructs vector $\mathbf{u}$ ($\mathcal{R}^{\mathcal{O}}$:line 1), which holds the u_i's, by executing $\mathsf{ConstructU}^{\mathcal{O}}$. After computing $\mathbf{u}$, $\mathcal{R}^{\mathcal{O}}$ passes it as argument to $\mathsf{ConstructD}^{\mathcal{O}}$ ($\mathcal{R}^{\mathcal{O}}$:line 2), which determines vector $\mathbf{d}$ that holds the d_i's. Each time, $\mathsf{ConstructD}^{\mathcal{O}}$ computes pairs of feasible values $\text{d}[m]$ and $\text{d}[n-m]$, and for each pair, it finds all the feasible values for $\text{d}[m+1]$ and $\text{d}[n-m-1]$, and so on ($\mathsf{ConstructD}^{\mathcal{O}}$:lines 3-9). Essentially, the algorithm builds a tree of height $n/2$, where each path represents a possible vector $\mathbf{d}$. Then, it checks which of these paths can be an actual solution ($\mathsf{ConstructD}^{\mathcal{O}}$:lines 11-12), and returns one of them ($\mathsf{ConstructD}^{\mathcal{O}}$:lines 1-2). Finally, $\mathcal{R}^{\mathcal{O}}$ computes the $\hat{\text{SK}}_i$'s ($\mathcal{R}^{\mathcal{O}}$:lines 3-5), and returns them ($\mathcal{R}^{\mathcal{O}}$:line 6).

Although we do not have any formal guarantees about the running time of the algorithm, it prunes many candidates at every step, because, for some accepted values of $\text{d}[m]$ and $\text{d}[n-m]$, there are no integer values of $\text{d}[m+1]$ and $\text{d}[n-m-1]$ that satisfy $\sum_{i=0}^{m+1} (\text{d}[i] \cdot \text{d}[n-(m-i)]) = \text{u}[n-m]$. Our experiments showed that our attack runs in milliseconds even for domains of size $N = 365$ and thousands of records.