Inference Attacks on Property-Preserving Encrypted Databases

Muhammad Naveed
UIUC*
naveed2@illinois.edu

Seny Kamara
Microsoft Research
senyk@microsoft.com

Charles V. Wright
Portland State University

leakage which is revealed from the EDB and the query protocol.

To better understand the impact of this leakage, an important research direction in encrypted search, initiated by Islam, Kuzu and Kantarcioglu [25], is the design of *inference attacks* which try to recover information about the data or queries by combining leakage with publicly-available information (e.g., census data or language statistics). The most well-known example of an inference attack is frequency analysis which is used to break classical ciphers. Another example is the query-recovery attack of Islam *et al.* against searchable symmetric encryption (SSE) schemes [25].

PPE-based EDBs. In the context of structured data and, in particular, of relational databases, the state-of-the-art encrypted search solutions are based on PPE schemes like deterministic and order-preserving encryption. Roughly speaking, a PPE scheme is an encryption scheme that leaks a certain property of the plaintext. For example, an order-preserving encryption (OPE) scheme encrypts a set of messages in such a way that their ciphertexts reveal the order of the messages (i.e., the *order* property). A deterministic encryption (DTE) scheme encrypts a set of messages in such a way that their ciphertexts reveal whether they are equal or not (i.e., the *equality* property).

The CryptDB system [35] first showed how to use PPE to construct an encrypted database system that supports a subset of SQL. The CryptDB design was adopted in the more recent Cipherbase [10] and Encrypted BigQuery systems [2]. The former uses a combination of trusted hardware and cryptography to efficiently support *full* SQL. At a very high-level, each DB operation can be either done in a secure co-processor or over encrypted data using the same approach as CryptDB. In this work, when referring to Cipherbase, we implicitly mean the variant where the operations in question are *not* executed in the secure co-processor.

These CryptDB-like systems have several advantages. In particular, they are competitive with real-world relational database systems and they require a minimal number of changes to the standard/legacy database infrastructure. The key to their efficiency and "legacy-friendliness" is the use of PPE which, roughly speaking, allows them to operate on encrypted data in the same way as they would operate on plaintext data. This enables fast operations on encrypted data and the use of standard database algorithms and optimizations.

1.1 Our Contributions

The use of PPE has important consequences on the security of encrypted database systems. Specifically, since PPE schemes leak a non-trivial amount of information, it is well-known that PPE-based designs like CryptDB and its variants are vulnerable to inference attacks. The extent to which these systems are vulnerable, however, has never been investigated.

In this work, we study concrete inference attacks against EDBs based on the CryptDB design. At a very high-level, these systems encrypt each DB column with layers of different encryption schemes. When queried, the system decrypts the layers until it reaches a layer that supports the necessary operation. In particular, this means that columns that support either range or equality queries are left encrypted with OPE or DTE, respectively. With this in mind, we consider inference attacks that take as input an OPE- or DTE-encrypted column and an auxiliary and public dataset and return a mapping from ciphertexts to plaintexts.

We stress that EDB systems are not designed to provide privacy but the much stronger requirement of *confidentiality*. As such, for an attack to be successful against an EDB it is not required to de-identify the records of the DB as would be the case, say, against a differentially-private DB [21]. In the setting of EDBs, an attack is successful if it recovers even *partial* information about a *single* cell of the DB. As we will see later, our attacks recover a lot more.

Concrete attacks. We study the effectiveness of four different attacks. Two are well-known and two are new:

- *frequency analysis:* is a well-known attack that decrypts DTE-encrypted columns given an auxiliary dataset that is "well-correlated" with the plaintext column. The extent of the correlation needed, however, is not significant and many publicly-available datasets can be used to attack various kinds of encrypted columns with this attack.
- ℓ_p*-optimization:* is a new *family* of attacks we introduce that decrypts DTE-encrypted columns. The family is parameterized by the ℓ_p-norms and is based on combinatorial optimization techniques.
- *sorting attack:* is an attack that decrypts OPE-encrypted columns. This folklore attack is very simple but, as we show, very powerful in practice. It is applicable to columns that are "dense" in the sense that every element of the message space appears in the encrypted column. While this may seem like a relatively strong assumption, we show that it holds for many real-world datasets.
- *cumulative attack:* is a new attack we introduce that decrypts OPE-encrypted columns. This attack is applicable even to low-density columns and also makes use of combinatorial optimization techniques.

Evaluating inference attacks. As discussed above, most inference attacks need an auxiliary source of information and their success depends on how well-correlated the auxiliary data is with the plaintext column. The choice of auxiliary data is therefore an important consideration when evaluating an inference attack. A strongly correlated auxiliary dataset may yield better results but access to such a dataset may not be available to the adversary. On the other hand, misjudging which datasets are available to the adversary can lead to overestimating the security of the system. An additional difficulty is that the "quality" of an auxiliary dataset is application-dependent. For example, census data may be well-correlated with a demographic database but poorly correlated with a medical database.

So the question of how to empirically evaluate inference attacks is non-trivial. In this work, we use the following methodology: (1) we choose a real-world scenario where the use of EDBs is *well-motivated*; (2) we consider encrypted columns from real-world data for the scenario under consideration; and (3) we apply the attack on the encrypted column using any relevant *publicly*-available auxiliary dataset.

Empirical results. For our empirical analysis, we chose databases for electronic medical records (EMRs) as our motivating scenario. Such medical DBs store a large amount of

private and sensitive information about both patients and the hospitals that treat them. As such they are a primary candidate for the real-world use of EDBs and appear frequently as motivation in prior work.

To evaluate our attacks, we consider DTE- and OPE-encrypted columns for several attributes using real patient data from the U.S. hospitals provided by the National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP). [1]

Following are the highlights of our results:

- *ℓ_2-optimization* (vs. DTE-encrypted columns): the attack recovered the mortality risk and patient death attributes for 100% of the patients for at least 99% of the 200 largest hospitals. It recovered the disease severity for 100% of the patients for at least 51% of those same hospitals.
- *frequency analysis* (vs. DTE-encrypted columns): the attack had the same results as ℓ_2-optimization.
- *sorting attack* (vs. OPE-encrypted columns): the attack recovered the admission month and mortality risk of 100% of patients for at least 90% of the 200 largest hospitals.
- *cumulative attack* (vs. OPE-encrypted columns): the attack recovered disease severity, mortality risk, age, length of stay, admission month, and admission type of at least 80% of the patients for at least 95% of the largest 200 hospitals. For 200 small hospitals, the attack recovered admission month, disease severity, and mortality risk for 100% of the patients for at least 99.5% of the hospitals.

Discussion. Our experiments show that the attacks considered in this work can recover a large fraction of data from a large number of PPE-based medical EDBs. In light of these results it is clear that these systems *should not be used in the context of EMRs*. One may ask, however, how the attacks would perform against non-medical EDBs, e.g., against human resource DBs or accounting DBs. We leave this as important future work but conjecture that the attacks would be at least as successful considering that much of the data stored in such DBs is also stored in medical DBs (e.g., demographic information).

We also note that even though the attacks can already recover a considerable amount of information from the EDBs, the results presented in this work should be viewed as a *lower bound* on what can be extracted from PPE-based EDBs. The first reason is that the attacks only make use of leakage from the EDB and do not exploit the considerable amount of leakage that occurs from the queries to the EDB. The second reason is that our attacks do not even target the weakest encryption schemes used in these systems (e.g., the schemes used to support equi- and range-joins).

2. RELATED WORK

Deterministic encryption was first formally studied by Bellare, Boldyreva and O'Neill in [11], where a security definition as well as various constructions were provided. OPE was introduced by Agrawal, Kiernan, Srikant and Xu [7] and studied formally by Boldyreva, Chenette, Lee and O'Neill in [12] and Boldyreva, Chenette and O'Neill [13]. In particular, it was shown in [12] that all (non-interactive) OPE schemes must leak more than the order. The work of Popa, Li and Zeldovich [34] describes an *interactive* protocol for an OPE-like functionality that leaks at most the order. The efficiency of this protocol was later improved by Kerschbaum and Schroepfer [29]. In [14], Boneh, Lewi, Raykova, Sahai, Zhandry and Zimmerman construct a (non-interactive) order-revealing encryption (ORE) scheme that leaks at most the order. The difference between OPE and ORE is that in the latter ciphertexts can be compared using an arbitrary algorithm whereas in OPE it must be the standard comparison operation.

The CryptDB system [35] was the first to support a large fraction of SQL on encrypted data. The CryptDB-design was later adopted by the Cipherbase [10] and Encrypted BigQuery systems [2]. Akin and Sunar [8] describe attacks that enable a malicious database administrator to recover plaintexts from CryptDB through a combination of passive monitoring and active tampering with the EDB.

Frequency analysis was first described by the Arab philosopher and mathematician al-Kindi in the ninth century [9]. Techniques for recovering plaintext encrypted with substitution ciphers using language statistics are well-known (see for example pp.245–250 of [32]). Brekne, Årnes and Øslebø [16] describe frequency attacks for recovering IP addresses anonymized under a prefix-preserving encryption scheme [38].

Islam *et al.* [25] described the first inference attack against an encrypted search solution. This attack, referred to as the IKK attack, exploits the access pattern leakage of SSE constructions together with auxiliary information about the frequencies of keyword pairs and knowledge of a subset of client queries (in plaintext) to recover information about the remaining queries. In comparison, the attacks we consider here: (1) recover the *database*, as opposed to queries; and (2) require no knowledge of any queries (neither in plaintext nor encrypted). Furthermore, a recent study by Cash, Grubbs, Perry and Ristenpart [18] shows that the accuracy of the IKK attack is so low that it is not usable in practice (unless the adversary already knows most of the underlying data). In contrast, the attacks considered in this work are highly-accurate and very efficient. The fact that our attacks are more powerful than the IKK attack is natural since PPE schemes leak considerably more than SSE schemes.

Sanamrad, Braun, Kossman and Venkatesan [36] also consider the security of OPE schemes in the context of encrypted databases. They propose a set of security definitions and discuss previously-known attacks (e.g., frequency analysis and sorting). Unlike standard security definitions, however, the security notions proposed in [36] are attack-specific (e.g., they define security only against frequency analysis) and guarantee only one-wayness; as opposed to standard cryptographic notions which guarantee that *partial* information is protected. Finally, [36] also proposes deterministic and probabilistic OPE variants. The deterministic variant is still vulnerable to our attacks (albeit requiring larger encrypted columns).

There is an extensive literature on OPE variants including probabilistic OPE, modular OPE, etc. [19, 22, 26, 27, 31, 39]. As far as we know, none of these constructions are used in any EDB system. We leave it as future work to assess the efficacy of our attacks against these variants.

[1] We stress that we strictly adhered to the HCUP data use agreement. In particular, our study is not concerned with the problem of de-anonymization. The data was not de-anonymized nor any attempt was made to do so.

3. PRELIMINARIES

Relational databases. A relational database is a collection of tables where each row corresponds to an entity (e.g., a customer or an employee) and each column corresponds to an attribute (e.g., age, height, salary). For any given attribute, we refer to the set of all possible values that it can take as its *space*. The *attribute space* of a column is the space of that column's attribute. If a column supports equality or range queries, then we refer to it as an equality or range column. The structured query language (SQL) is a special-purpose language for querying relational databases.

Datasets. A dataset $\mathbf{d} = (d_1, \ldots, d_n)$ is a sequence of elements from a universe $\mathbb{D}$. We assume, without loss of generality, that every space $\mathbb{D}$ is totally ordered. We view the *histogram* of a dataset $\mathbf{d}$ as a $|\mathbb{D}|$-dimensional vector over $\mathbb{N}_{\geq 0}$ with, at position i, the number of times the ith element of $\mathbb{D}$ appears in $\mathbf{d}$. We denote by $\mathsf{Hist}(\mathbf{d})$ the operation that computes the histogram of a dataset $\mathbf{d}$. The *cumulative distribution function* (CDF) of a dataset $\mathbf{d}$ is a $|\mathbb{D}|$-dimensional vector over $\mathbb{N}_{\geq 0}$ with, at position i, the number of times the first through ith elements of $\mathbb{D}$ that appear in $\mathbf{d}$. We denote by $\mathsf{CDF}(\mathbf{d})$ the operation that computes the CDF of a dataset $\mathbf{d}$. The CDF of $\mathbf{d}$ is the vector $\mathbf{f}$ such that for all $i \in [n]$, $f_i = \sum_{j=1}^{i} h_j$, where $\mathbf{h} = (h_1, \ldots, h_n)$ is the histogram of $\mathbf{d}$.

We denote by $\mathsf{Unique}(\mathbf{d})$ the dataset that results from removing all duplicates in $\mathbf{d}$ (i.e., from keeping only the first occurrence of every element in $\mathbf{d}$). The rank of an element $d \in \mathbb{D}$ in a dataset $\mathbf{d}$ is the position of its first occurrence in $\mathbf{d}$ if $d \in \mathbf{d}$ and 0 if $d \notin \mathbf{d}$. We denote the rank of d in $\mathbf{d}$ by $\mathsf{Rank}_{\mathbf{d}}(d)$.

We will often need to sort datasets. The result of sorting $\mathbf{d}$ by value is the sequence $\mathbf{d}' = (d_{i_1}, \ldots, d_{i_n})$ such that $d_{i_1} < \cdots < d_{i_n}$. We denote this operation $\mathbf{d}' \leftarrow \mathsf{vSort}(\mathbf{d})$. When dealing with the histogram $\mathbf{h}$ of a dataset $\mathbf{d}$ over $\mathbb{D}$, we identify the coordinates of $\mathbf{h}$ with the elements of $\mathbb{D}$.

Encryption. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a tuple of three algorithms that work as follows. Gen takes as security paramater as input and returns a secret key K; Enc takes as input a key K and a message m and returns a ciphertext c; and Dec takes as input a key K and a ciphertext c and returns a message m. The standard notion of security for encryption is security against chosen-plaintext attacks (CPA). We refer the reader to [28] for a detailed description of this notion. Here, we only mention that it is well-known that for symmetric-key encryption, CPA-security can only be achieved if Enc is either stateful or randomized.

Deterministic encryption. A symmetric DTE scheme $\mathsf{DTE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme for which Enc is not randomized; that is, each message m is mapped by Enc to a *single* ciphertext under a key K.

Order-preserving encryption. A symmetric OPE scheme $\mathsf{OPE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme with the following property: if $m_1 > m_2$ then $\mathsf{Enc}_K(m_1) > \mathsf{Enc}_K(m_2)$; if $m_1 = m_2$ then $\mathsf{Enc}_K(m_1) = \mathsf{Enc}_K(m_2)$; and if $m_1 < m_2$ then $\mathsf{Enc}_K(m_1) < \mathsf{Enc}_K(m_2)$.

Additively homomorphic encryption. colorred A symmetric additively homomorphic encryption (AHE) scheme $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme with the added property that: $\mathsf{Dec}_K\ \mathsf{Enc}_K(m_1) \otimes \mathsf{Enc}_K(m_2) = m_1 + m_2$, where $\otimes$ is an operation over the ciphertext space of AHE and not necessarily addition.

Join encryption. The CryptDB system supports two kinds of Join operations: equi-joins and range-joins. Equi-joins are supported using a scheme $\mathsf{EJOIN} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which is a combination of DTE and hashing. Range-joins are supported using an encryption scheme $\mathsf{RJOIN} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ based on OPE. We note that after a join query (of either kind), two joined columns are left encrypted under the same key.

Searchable encryption. The systems also make use of searchable encryption scheme $\mathsf{SRCH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Dec})$ for keyword search operations. In CryptDB, this is instantiated with a variant of the scheme of the scheme of Song, Wagner and Perrig [37].

Onion encryption. Popa *et al.*use the term *onion* to refer to the composition of encryption schemes. For example, given two encryption schemes $\mathsf{SKE}^1 = (\mathsf{Gen}^1, \mathsf{Enc}^1, \mathsf{Dec}^1)$ and $\mathsf{SKE}^2 = (\mathsf{Gen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2)$ the $\mathsf{SKE}^1 \circ \mathsf{SKE}^2$ encryption of a message m is defined as

$$\mathrm{ct} = \mathsf{Enc}^1_{K_1}\ \mathsf{Enc}^2_{K_2}(m)\ .$$

3.1 PPE-based Encrypted Databases

We recall high-level architecture of EDB systems based on the CryptDB design. The system is composed of three entities: an application App, a proxy Prx, and a server Srv. The application and proxy are trusted while the server is untrusted. To create an encrypted database EDB from a database DB, the proxy generates a master secret key msk and uses it to encrypt each table as follows. First, an anonymized schema is created where the attributes for each column are replaced with random labels. The mapping between the attributes and their labels is stored at the proxy. Then, each cell is encrypted using four different *onions*. More specifically, the following four onions are used

- *Equality onion*: encrypts a string s as
$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{SKE}}_{K_S}\ \mathsf{Enc}^{\mathsf{DTE}}_{K_D}\ \mathsf{Enc}^{\mathsf{EJOIN}}_{K_J}(s) \quad ;$$
- *Order onion*: encrypts a string s as
$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{SKE}}_{K_S}\ \mathsf{Enc}^{\mathsf{OPE}}_{K_O}\ \mathsf{Enc}^{\mathsf{RJOIN}}_{K_{OJ}}(s) \quad ;$$
- *Search onion*: encrypts a keyword w as
$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{SRCH}}_{K_S}\ w\ ;$$
- *Add onion*: encrypts an integer i as
$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{AHE}}_{K_A}\ i\ ;$$

To support queries on encrypted data, the encrypted cells in the EDB are decrypted down to a certain layer. This process is referred to as *peeling* in [35] and every cell in a given column is peeled to the same level. The proxy keeps track of the layer at which each column is peeled.

To query an encrypted database the application issues a SQL query that is rewritten by the proxy before being sent to

the server. In the new query, each column name is replaced with its random label and each constant is replaced with a ciphertext determined as a function of the semantics of the query. More precisely, for each type of operation the proxy does the following:

- *equality*: v is replaced with ct = $\mathsf{DTE.Enc}_K(v)$;
- *range*: v is replaced with ct = $\mathsf{OPE.Enc}_K(v)$;
- *search*: v is replaced with tk = $\mathsf{SRCH.Token}_K(v)$;
- *addition*: v is replaced with ct = $\mathsf{AHE.Enc}_K(v)$;
- *join*: v is replaced with ct = $\mathsf{EJOIN.Enc}_K(v)$;

After re-writing the query, the proxy checks the onion levels of the relevant columns to determine if they need to be peeled further. If so, it sends the appropriate decryption keys to the server so that it peel the columns down to the appropriate layer.

4. THREAT MODEL

An EDB system should protect a database against a variety of threats. In this Section, we describe some of these threats and propose an adversarial model that captures them. In defining such a model, we make two things explicit: (1) the goal of the attack; and (2) the information the adversary holds when carrying out the attack.

4.1 Adversarial Goals

There are at least two kinds of attacks on EDBs which we refer to as *individual* attacks and *aggregate* attacks.

Individual attacks. In an individual attack, the adversary is concerned with recovering information about a row in the database. For example, if the EDB is a medical database where each row corresponds to a patient, then the goal of the attack would be to recover information about a specific patient, e.g., its age or name.

Aggregate attacks. In an aggregate attack, the adversary wants to recover statistical information about the entire database. Again, in the context of a medical database, this could be information such as the total number of patients with a particular disease or the number of patients above a certain age. We note that, depending on the context, aggregate attacks can be extremely harmful. For example, hospitals do not disclose the number of cancer patients they treat so as not to signal anything about the quality of their cancer treatments.

4.2 Adversarial Information

PPE-based EDBs like [10,35] are designed to protect against a semi-honest adversary that corrupts the server. Intuitively, this means that the adversary has access to everything the server sees but cannot influence it—in particular, it cannot make it deviate from the prescribed protocol. Since the adversary has complete access to what the server sees, it holds the encrypted database and can see the queries generated by the proxy.

Ciphertext-only. In this work, we focus on a considerably weaker adversary which has access to the encrypted database but not to the queries. We stress that this is a much weaker adversary than what is typically considered in the literature and captures all the threats that database customers are typically concerned with. This includes internal threats like malicious database administrators and employees, and external threats like hackers, nation states, and organized crime.

Steady state EDBs. We assume the adversary has access to the encrypted database in *steady state*, which means that the onions of each cell are peeled down to the lowest layer needed to support the queries generated by the application. Intuitively, one can think of the steady-state EDB as the state of the EDB after the application has been running for a while.

Auxiliary information. In addition to the encrypted database, we assume our adversary has access to auxiliary information about the system and/or the data. Access to auxiliary information is standard in any practical adversarial model since the adversary can always consult public information sources to carry out the attack. In particular, we consider the following sources of auxiliary information:

- *application details*: the application running on top of the encrypted database, possibly obtained from accessing the application (e.g., if it is a web service) or from documentation;
- *public statistics*: publicly available statistics, for example, census data or hospital statistics;
- *prior versions*: prior versions of the database, possibly obtained through a prior data breach.

We stress that our experiments will make use of a different subset of auxiliary sources and that none of the attacks need access to all of these sources.

4.3 Attack Accuracy

When an adversary executes an inference attack, it receives as output an assignment from the encrypted cells to the elements of the message space. Though our experiments in Section 9 show that there are many attributes for which the attacks are perfectly accurate, this is not always the case and for low-accuracy attributes it could be difficult for the attacker to distinguish correct assignments from incorrect ones. We note, however, that the attacks can still be damaging even for these attributes for the following reasons. First, the adversary can still learn statistics about the attribute which in some cases, like patient died during hospitalization or major diagnostic category, can be very sensitive for hospitals because it reveals information about the quality of their care. Second, the results can still be used for phishing-style attacks where the adversary only needs a small number of successes.

5. ATTACKING DTE COLUMNS

We describe two attacks against DTE-encrypted columns. The first is the well-known frequency analysis and the second is a family of attacks we refer to as ℓ_p-optimization attacks. The family is parameterized by the ℓ_p norms.

Here, $\mathbb{C}_k$ and $\mathbb{M}_k$ are the ciphertext and message spaces of the deterministic encryption scheme. We assume $|\mathbb{C}_k| = |\mathbb{M}_k|$ but if this is not the case we simply pad $\mathbb{M}_k$. For encryption schemes $|\mathbb{C}_k|$ is always at least $|\mathbb{M}_k|$.

5.1 Frequency Analysis

Frequency analysis is the most basic and well-known inference attack. It was developed in the 9th century and is used to break classical ciphers. As is well-known, frequency analysis can break deterministic encryption and, in particular, deterministically-encrypted columns. Given a DTE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and an auxiliary dataset $\mathbf{z}$ over $\mathbb{M}_k$, the attack works by assigning the ith most frequent element of $\mathbf{c}$ to ith most element of $\mathbf{z}$. For ease of exposition, we assume that $\mathbf{c}$ and $\mathbf{z}$ have histograms that can be strictly ordered; that is, for all $i \neq j$, $\psi_i \neq \psi_j$ and $\pi_i \neq \pi_j$, where $= \mathsf{Hist}(\mathbf{c})$ and $= \mathsf{Hist}(\mathbf{z})$. More precisely, the attack is defined as:

- **Frequency-An**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{c}))$;
  2. compute $\leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{z}))$;
  3. output $\alpha : \mathbb{C}_k \to \mathbb{M}_k$ such that

$$\alpha(c) = \begin{cases} \mathsf{Rank}_{\psi}(c) & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

If the histograms are not strictly ordered (i.e., there are $i \neq j$ such that $\psi_i = \psi_j$ or $\pi_i = \pi_j$) one can still run the attack by breaking ties in the sorting steps arbitrarily. In the worst-case, each tie will be broken erroneously and induce an error in the assignment so this will cause the attack to err on $a + b$ ciphertexts, where a and b are the number of ties in $\mathsf{Hist}(\mathbf{c})$ and $\mathsf{Hist}(\mathbf{z})$, respectively. The attack runs in $O\ |\mathbb{C}_k| \cdot \log |\mathbb{C}_k|\}$ time.

5.2 ℓ_p-Optimization

We now describe a family of attacks against DTE-encrypted columns we refer to as ℓ_p-optimization. The family is parameterized by the ℓ_p norms. The basic idea is find an assignment from ciphertexts to plaintexts that minimizes a given cost function, chosen here to be the ℓ_p distance between the histograms of the datasets. This has the effect of minimizing the *total* mismatch in frequencies across all plaintext/ciphertext pairs. The attack works as follows.

Given a DTE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and auxiliary information $\mathbf{z}$ over $\mathbb{M}_k$, the adversary first computes the histograms and of $\mathbf{c}$ and $\mathbf{z}$, respectively. It then finds the permutation matrix X that minimizes the ℓ_p distance between the ciphertext histogram and the permuted auxiliary histogram $X \cdot$. Intuitively, the attack finds the mapping of plaintexts to ciphertexts that achieves the closest overall match of their sample frequencies. Note that this is very different than frequency analysis which ignores the amplitude of the frequencies and only takes into account their rank. More precisely, the attack is defined as follows:

- ℓ_p**-Optimization**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\leftarrow \mathsf{Hist}(\mathbf{c})$;
  2. compute $\leftarrow \mathsf{Hist}(\mathbf{z})$;
  3. output $\arg\min_{X \in \mathbb{P}_n} \| \quad - X \cdot \quad \|_p$;

where $\mathbb{P}_n$ is the set of $n \times n$ permutation matrices. Note that in the ℓ_1-optimization attack, Step 3 can be formulated as a *linear sum assignment problem* (LSAP) [17]. The LSAP can be solved efficiently using the well-known Hungarian algorithm [30, 33] or any linear programming (LP) solver. In our experiments we use the former which runs in time $O(n^3)$. The precise LSAP formulation is:

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1} \sum_{j=1} C_{ij} X_{ij} \\ \textbf{subject to} \quad & \sum_{i=1} X_{ij} = 1, \quad 1 \leq j \leq |\mathbb{C}_k| \\ & \sum_{j=1} X_{ij} = 1, \quad 1 \leq i \leq |\mathbb{C}_k| \\ & X_{ij} \in \{0, 1\}, \quad 1 \leq i, j \leq |\mathbb{C}_k|. \end{aligned}$$

where the cost matrix $C = C_{ij}$ gives the cost of matching plaintext j to ciphertext i.

For $p = 1$, the costs are simply the absolute differences in frequency, so we set $C_{ij} = |\psi_i - \pi_j|$. For $2 \leq p \leq \infty$, however, Step 3 of the ℓ_p-optimization attack cannot be formulated directly as a LSAP because the ℓ_p norm is not a simple linear sum. Nevertheless, we show that it can still be efficiently solved using fast LSAP solvers. To see why, let $f_1 : \mathbb{R}^+ \to \mathbb{R}$ be the function $x \mapsto \sqrt[p]{x}$ and let $f_2 : \mathbb{N}^n_{\geq 0} \to \mathbb{N}_{\geq 0}$ be the function $\mathbf{v} \mapsto \sum_{i=1}^{n} v_i^p$. Then we note that the ℓ_p norm of a vector can be written as

$$\|\mathbf{v}\|_p = f_1 \ f_2(\mathbf{v}) \ .$$

Since f_1 is monotone increasing, the vector that minimizes $f_1 \circ f_2$ is the vector that minimizes f_2. It follows then that for any vector $\mathbf{v}$, the vector $\mathbf{w}$ with the minimum ℓ_p distance from $\mathbf{v}$ is the solution to

$$\arg\min_{\mathbf{w}} \sum_{i=1} |v_i - w_i|^p.$$

As long as $p < \infty$, this optimization problem can be formulated as a LSAP with cost matrix C such that $C_{ij} = |v_i - w_i|^p$. The attack takes $O(|\mathbb{C}_k|^3)$ time.

Remark on ℓ_p-optimization vs. frequency analysis. In our experiments, we found that frequency analysis and ℓ_p-optimization for $p = 2, 3$ performed equally well. In fact, for a fixed encrypted column and auxiliary dataset, they decrypted same exact ciphertexts. On the other hand, frequency analysis did consistently better than ℓ_1-optimization. This raises interesting theoretical and practical questions. From a theoretical perspective it would be interesting to understand the exact relationship between frequency analysis and ℓ_p-optimization. Our experiments tell us that ℓ_1-optimization is different from frequency analysis (since they generated different results) but they did not distinguish between frequency analysis and ℓ_2- and ℓ_3-optimization. As such, it would be interesting to either separate the attacks or prove that they are equivalent for some $p \geq 2$.

From a practical perspective, the main question is what is the motivation for ever using ℓ_p-optimization over frequency analysis? The main reason is that ℓ_p-optimization not only decrypts an encrypted column but, while doing so, also produces cost information about the different solutions it finds. Like the cumulative attack we describe in Section 6.2, this is due to its use of combinatorial optimization. As it turns out, this extra information can be leveraged to attack "hidden" columns (i.e., for which we do not know the attribute);

something we cannot always do with frequency analysis. We discuss this in more detail in Section 8.

6. ATTACKING OPE COLUMNS

In addition to the frequency information leaked by DTE, order-preserving encryption also reveals the relative ordering of the ciphertexts. Here we describe two attacks on OPE-encrypted columns that exploit this additional leakage to recover even more of the plaintext data. Note that the attacks only make use of order information so they work even against columns encrypted with ORE [14] and interactive order-preserving protocols [29, 34]. In particular, since all OPE instantiations necessarily leak more than just the order [12], stronger attacks are likely possible against OPE-encrypted columns.

Here, $\mathbb{C}_k$ and $\mathbb{M}_k$ are the ciphertext and message spaces of the OPE scheme. We assume, without loss of generality, that $|\mathbb{C}_k| = |\mathbb{M}_k|$. If this is not the case we pad $\mathbb{M}_k$ with additional symbols until it holds.

6.1 Sorting Attack for Dense Columns

The first attack on OPE-encrypted columns is trivial and applicable to all columns that satisfy a condition we call *density*. We call an OPE-encrypted column δ-dense, if it contains the encryptions of at least a δ fraction of its message space. If $\delta = 1$, we simply say that the column is dense.

The attack is described in detail below and works as follows. Note that it does not require any auxiliary information. Given an OPE-encrypted dense column $\mathbf{c}$ over $\mathbb{C}_k$ the adversary simply sorts $\mathbf{c}$ and $\mathbb{M}_k$ and outputs a function that maps each ciphertext $c \in \mathbf{c}$ to the element of the message space with the same rank. More precisely, the attack is defined as:

- **Sorting-Atk**$(\mathbf{c})$:
  1. compute $\leftarrow \mathsf{vSort\ Unique\ c}$;
  2. compute $\leftarrow \mathsf{vSort}\ \mathbb{M}_k$;
  3. output $\alpha : \mathbb{C}_k \to \mathbb{M}_k$ such that:

$$\alpha(c) = \begin{cases} \mathsf{Rank}_{\psi}(c) & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

The attack runs in $O\left(|\mathbb{C}_k| \cdot \log |\mathbb{C}_k|\right)$ time.

6.2 Cumulative Attack for Low-Density Columns

The main limitation of the sorting attack is that it is only applicable to dense columns. To address this, we describe a second attack for low-density OPE-encrypted columns we refer to as the *cumulative* attack. The attack requires access to auxiliary information and can recover a large fraction of column cells (see Section 9.2 for details).

Intuition. Given a DTE-encrypted column, the adversary learns the sample frequency of each ciphertext in the column. These sample frequencies make up the histogram for the encrypted column, and we showed in the previous section how the adversary can use them to match the DTE ciphertexts to their plaintexts by finding (c, m) pairs where c and m have similar frequencies.

Given an OPE-encrypted column, the adversary learns not only the frequencies but also the relative ordering of the encrypted values. Combining ordering with frequencies, the adversary can tell for each ciphertext c what fraction of the encrypted values are less than c. More formally, this is known as the *empirical cumulative distribution function* (ECDF, or simply CDF) of the data set.

In the cumulative attack, we leverage the CDF to improve our ability to match plaintexts to ciphertexts. Intuitively, if a given OPE ciphertext is greater than 90% of the ciphertexts in the encrypted column $\mathbf{c}$, then we should match it to a plaintext that also is greater than about 90% of the auxiliary data $\mathbf{z}$. Although our early experiments showed that CDFs alone enable very powerful attacks on OPE, we can achieve even better results using both the CDFs and the frequencies together. Here we use an LSAP solver to find the mapping of plaintexts to ciphertexts that minimizes the total sum of the mismatch in frequencies plus the mismatch in CDFs across all plaintext/ciphertext pairs.

Overview of attack. The attack is detailed below and works as follows. Given an OPE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and an auxiliary dataset $\mathbf{z}$ over $\mathbb{M}_k$, the adversary computes the histograms and and the CDFs ′ and of $\mathbf{c}$ and $\mathbf{z}$, respectively. It then finds the permutation that simultaneously matches both the sample frequencies and the CDFs as closely as possible. More precisely, the attack is defined as:

- **Cumulative-Atk**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\leftarrow \mathsf{Hist\ c}$ and $' \leftarrow \mathsf{CDF\ c}$;
  2. compute $\leftarrow \mathsf{Hist\ z}$ and $\leftarrow \mathsf{CDF\ z}$;
  3. output

$$\arg\min_{X \in \mathbb{PMP}} \mathbb{C}_k \quad \infty \quad \leftarrow \quad \leftarrow \quad \infty \quad \infty \quad \leftarrow$$

First, medical DBs hold highly personal and sensitive information and are often covered by privacy regulations such as the Health Portability and Accountability Act (HIPAA). EMRs are vulnerable to insider and outsider threats and are increasingly targeted by professional attackers including state sponsored adversaries and organized crime. This trend is illustrated by the recent attacks on Anthem—one of the largest U.S. health insurance providers—which compromised the health records of 80 million individuals. In fact, the Ponemon Institute's recent study on *Privacy and Security of Healthcare Data* [1] reports that criminal attacks are now the number one cause of healthcare data breaches with a 125% growth in attacks reported in the last 5 years. As such, the motivation to encrypt medical DBs is very strong. In fact, medical DBs often appear as the standard motivation in the encrypted database research literature (see, e.g., [35]).

Another reason we chose this scenario is that a subset of the data stored in EMRs (e.g., demographic data) is also held in other types of sensitive DBs including human resources DBs, accounting DBs, and student DBs. Information stored in these DBs may also be covered by privacy regulations such as the Family Educational Rights and Privacy Act (FERPA). Our results against medical DBs can therefore tell us something about these other kinds of DBs.

7.1 Target Data

Throughout, we refer to the data we use to populate the EDB as the *target* data. In our experiments we use data from the National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP) [3]. HCUP makes available the largest collection of longitudinal hospital care data in the U.S. The NIS database—which includes data on inpatients (i.e., patients that stay at a hospital for at least one night) from all the hospital in the U.S.—is available starting from 1988. The database is made available to researchers under controlled access: an online training is required and a data use limitation agreement must be signed before the data can be purchased and used. The NIS database includes attributes such as age, drugs, procedures, diagnosis, length of stay, etc. For our purposes, we only use a subset of the attributes (mostly due to space limitations) which we describe in Figure 1.

	Max	Min	Mean	SD
Large Hospitals	121,664	12,975	24,486	12,015
Small Hospitals	1,309	404	756	253

Table 1: Size of hospitals in number of patients

In our experiments we use the data from a subset of 1050 hospitals in the 2009 HCUP/NIS database as our target data. We note that any other year would have given similar results. For all but one of our experiments we use the 200 largest hospitals but for the evaluation of the cumulative attack against low-density columns we use data from 200 small hospitals. The 200 small hospitals are the ones ranked (in decreasing order) 701 through 900 in terms of patient-size. Smaller hospitals had too few patients to attack (less than 400 and some even less than 10). The number of patients in the 200 largest and the 200 small hospitals is shown in Table 1.

Target attributes. We chose a subset of columns/attributes from the 2009 HCUP/NIS dataset to attack. These attributes are listed in Figure 1. We believe these or similar attributes would be present in most real-world EMR systems. We confirmed that six of them, including sex, race, age, admission month, patient died, and primary payer, are used by OpenEMR [6], which is an open source fully-functional EMR application. We stress that the form in which these attributes are stored can vary (e.g., age can be stored as an integer or computed from a date of birth) but some variant of these attributes exist in OpenEMR.

- **Sex.** Sex can be either male or female. The most prominent feature of the sex attribute is that most hospitals have more female patients than male patients. This is possibly due to pregnancy, births, and the fact that women live longer. Sex is universally used in all databases that store information about people.
- **Race.** Race can have the following values: white, black, Hispanic, Asian or Pacific Islander, Native American, and other. Race is stored in most databases dealing with people for a variety of reasons.
- **Age.** Age can range from 0 to 124. Age 0 is for babies less than an year old. Some databases may store birth year instead of age, e.g., as part of full date of birth. Frequency counts for age and birth year are exactly the same.
- **Admission Month.** Admission month has values that range from January to December.
- **Patient died during hospitalization.** This attribute indicates whether a patient died during hospitalization.
- **Primary Payer.** Primary payer has six values: Medicare, Medicaid, private or health maintenance organization, self-pay, no charge, and other.
- **Length of Stay.** Length of stay ranges from 0 to 364 and represents the number of days a patient spends in a hospital. It is a very sensitive attribute and reveals information about other attributes such as the nature of the patient's disease.
- **Mortality Risk.** Mortality Risk has four values showing the likelihood of dying: minor, moderate, major, and extreme. It indicates the risk of a patient dying in the hospital.
- **Disease Severity.** Disease Severity has four values showing loss of function: minor (indicates cases with no comorbidity or complications), moderate, major, and extreme. It indicates the severity of the patient's disease.
- **Major Diagnostic Category.** Major Diagnostic Category has 25 values and gives the principal diagnosis such as "Diseases and Disorders of Kidney", "Burns", "Human Immunodeficiency Virus Infection (HIV)", etc.
- **Admission Type.** Admission type has six values: emergency, urgent, elective, newborn, trauma center, and other.
- **Admission Source.** Admission source has five values: emergency room, another hospital, another facility including long-term care, court/law enforcement, and routine/birth/other. It indicates from where the patient was admitted to the hospital.

Figure 1: Attributes/columns used in our evaluation.

To decide whether an attribute should be DTE or OPE-encrypted we did the following. For the attributes stored by OpenEMR (in some form), we simply checked the kinds of operations OpenEMR supported on it. If it supported either range queries or sorting operations, we considered it an OPE attribute. If OpenEMR supported equality queries on the attribute we considered it a DTE attribute. For the remaining attributes, we made assumptions which we be-

lieve to be reasonable. More specifically, we assumed an EMR system would support range queries on the length of stay attribute; sorting queries on the mortality risk, disease severity, and admission type attributes (e.g., for triage); and equality queries on major diagnostic category and admission source.

7.2 Auxiliary Data

All but one of our attacks (sorting) require an auxiliary dataset to decrypt a PPE-encrypted column. We used the following two auxiliary datasets:

Texas PUDF data. The first auxiliary dataset we use is the Texas Inpatient Public Use Data File (PUDF), which is provided by the Texas Department of State Health Services. This dataset—unlike the HCUP/NIS data—is publicly available online so there is no reason to believe an adversary would not use it to her advantage. Specifically, we use the 2008 Texas PUDF data. The Texas PUDF data until year 2008 can be downloaded from [4]. Usage of the data requires an acceptance of a data use agreement but we believe it is reasonable to assume that an adversary would not comply with such an agreement.

2004 **HCUP/NIS.** Unfortunately, the Texas PUDF data has a limited number of attributes which prevents us from studying the accuracy of our attacks on several attributes of interest. We therefore also run experiments using the 2004 HCUP/NIS database as auxiliary data (recall that our target data is the 2009 HCUP/NIS data). Note that each year of the HCUP/NIS data comes from a random sample of hospitals from a large number of U.S. hospitals and the entire data of each sampled hospital is included. This means that the 2004 HCUP/NIS data is not only different in time from the 2009 HCUP/NIS data but it is also comes from a different set of hospitals. There is a small number of common hospitals between 2004 and 2009 HCUP/NIS databases (less than 4%), but that does not have a noticeable impact on our experimental results.

Remark on additional datasets. Another example of a publicly-available auxiliary dataset is the Statewide Planning and Research Cooperative System (SPARCS) Inpatient data from the state of New York [5]. We do not report results using SPARCS as auxiliary data due to space limitations, but it gives similar results to those using the Texas PUDF data.

8. EXPERIMENTAL SETUP

All experiments were conducted on a high-end Mac laptop with Intel Core i7 processor and 16GB memory running OS X Yosemite (v10.10.2). We used Python version 2.7.6 and Matlab version 8.4.0 (R2014b). For our experiments we developed three tools: Parser, Column Finder, and Revealer which we now describe.

Parser. Parser is written in Python and parses the target and auxiliary data to create appropriate histograms. In the case of the target data, it creates one histogram per attribute/hospital pair. More precisely, for each pair it creates a histogram that reports the number of times some value v of the attribute appears in the hospital's data. In the case of the auxiliary data, it creates a single histogram for each attribute (i.e., over all hospitals).

Column Finder. Column Finder is also written in Python. Since CryptDB-like EDB systems encrypt column names, an adversary first needs to learn which encrypted columns correspond to the attribute of interest. We do this using the following approach. First, we determine if the attribute of interest is present in the EDB by checking the database schema of the application. Then we run Column Finder which works as follows:

1. it determines the number of distinct values for the column of interest in the auxiliary data. We'll refer to this column as the auxiliary column. As an example, Column Finder would use the auxiliary data to learn that age has 125 possible values or that sex has 2 possible values.

2. it then determines the number of distinct values stored in each DTE- and OPE-encrypted column of the EDB. This is trivial due to the properties of these encryption schemes. It then searches through these encrypted columns to find the ones that have approximately the same number of distinct values as the auxiliary column. We have to search for approximate matches since some values of an attribute may not be present in the target data. Since we know from the database schema of the application that the EDB contains an encrypted column for the attribute of interest, this step will find at least one column:

   (a) If it finds only one column, then that is the encrypted column for the attribute of interest.

   (b) If it finds more than one column with a close-enough number of distinct values such that it cannot determine which column belongs to the attribute of interest, then it outputs all of them.

Auxiliary Attribute	**Target Attributes**	**Accuracy**
Primary Payer	Admission Type, Primary Payer, Race	116
Race	Admission Type, Primary Payer, Race	152
Admission Type	Admission Type, Primary Payer, Race	128
Sex	Sex, Patient Died	200
Patient Died	Sex, Patient Died	200

Table 2: Column recovery: the accuracy column reports the number of hospitals for which the correct attribute (i.e., from the auxiliary attribute column) had the lowest ℓ_2-optimization cost among all target attributes.

Data Revealer. Revealer is written in Matlab and implements frequency analysis, ℓ_2-optimization, and the cumulative attack. The last two attacks use the Hungarian algorithm for the optimization step. We did not implement the sorting attack against dense columns since correctness and perfect accuracy is trivially true (we do run experiments to report the prevalence of dense columns in our target dataset and results are shown in Figure 4). Revealer takes as input the histogram of an auxiliary column from the output of Parser and the histograms for a set of target encrypted columns from the output of Column Finder. So, depending on the output of Column Finder, Revealer can receive either a single target histogram or multiple target histograms and in each case it works as follows:

- if it receives a single target histogram, Revealer simply runs the attack with its two inputs.

- if it receives multiple target histograms, Revealer runs one of the optimization-based attacks on the auxiliary histogram with *each* of the target histograms. It then outputs the result with the minimum cost.

Note that only the ℓ_p-optimization and cumulative attacks can be executed when there are multiple target histograms since frequency analysis does not have an inherent notion of cost that can be used. In our experiments, we found that when the target and auxiliary attributes are the same, the cost is significantly less than when they are different. This is reported in Table 2.

Time measurements. All the attacks take less than a fraction of a second per hospital. Table 3 reports the running times (averaged over 200 hospitals) for each attack over a different set of attributes; each with a different number of values.

Notice that attacking the length of stay column requires considerably more time than the rest. This is due to the fact that it has a large number of values (365) which especially affects the running time of ℓ_2-optimization and cumulative attacks which rely on optimization. Currently, our attacks are implemented in Matlab which is very slow compared to other languages like C so we believe that a C implementation would decrease the running time by several orders of magnitude.

Atts. (# of values)	Freq. An.	ℓ_2-opt.	Cumul.
Sex (2)	0.11ms	0.11ms	0.31ms
Mortality Risk (4)	0.12ms	0.12ms	0.49ms
Admission Source (5)	0.12ms	0.13ms	0.60ms
Major Diagnostic Category (25)	0.19ms	0.20ms	3.5ms
Age (125)	0.63ms	3.03ms	311.6ms
Length of stay (365)	1.73ms	68.7ms	35,910ms

Table 3: Time (in milliseconds) of attacks per hospital.

9. EXPERIMENTAL RESULTS

For each hospital and each column in the EDB, we compute the accuracy of our attack as the number of encrypted cells for which the recovered plaintext matches the ground truth, divided by the total number of column cells.

We present the results of these experiments in Figures 3, 2 and 5. Each plot shows the *empirical* Complementary Cumulative Distribution Function (CCDF) of our record-level accuracy across all the hospitals in our target data. For example, a point at location (x, y) indicates that we correctly recovered at least x fraction of the records for y fraction of the hospitals in the target data. The results show that our attacks recover a substantial fraction of the encrypted DBs and perform significantly better than random guessing.

9.1 Attacks on DTE-Encrypted Columns

Figure 3 shows the results of our ℓ_2-optimization attack against DTE-encrypted columns using the 2004 HCUP/NIS dataset as auxiliary. Figure 2 shows results of the same attack using Texas PUDF dataset as auxiliary.

Using the 2004 HCUP/NIS as auxiliary data (Figure 3), ℓ_2-optimization recovers cells for a significant number of patients, even for attributes with a large number of distinct values such as Age and Length of Stay. It recovered Mortality Risk and whether the patient died for 100% of the patients for 99% and 100% of the hospitals respectively. It also recovered the Disease Severity for 100% of the patients for 51% of the hospitals. The attack recovered Race for at least 60% of the patients for at least 69.5% of the hospitals; Major Diagnostic Category for at least 40% of the patients for 27.5% of the hospitals; Primary Payer for at least 90% of the patients for 37.5% of the hospitals; Admission Source for at least 90% of the patients for 38% of the hospitals; Admission Type for at least 60% of the patients for 65% of the hospitals.

Perhaps surprisingly, ℓ_2-optimization also recovered a relatively small but significant fraction of cells for the Age attribute. Recovering DTE-encrypted Ages is very difficult because Age takes on a large range of values, and multiple values have very similar frequencies. Nonetheless, it recovered Age for at least 10% of patients for 84.5% of the hospitals. The attack also works surprisingly well for Length of Stay despite its large range of 365 possible values: specifically, it recovers this attribute for at least 83% of the patients for 50% of the hospitals. The reason for this unexpected accuracy is that most patients stay in the hospital for only a few days. Therefore, by decrypting the plaintexts for a few very common lengths of stay (e.g., $1, 2, 3, \dots$), we recover a large fraction of the database.

Using the Texas PUDF data as auxiliary (Figure 2), the attack performs similarly well. There is a small decrease in accuracy for Race and Major Diagnostic Category. We believe this is due to regional differences in demographics across the U.S.

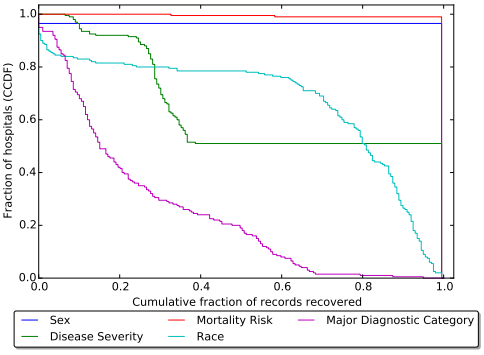
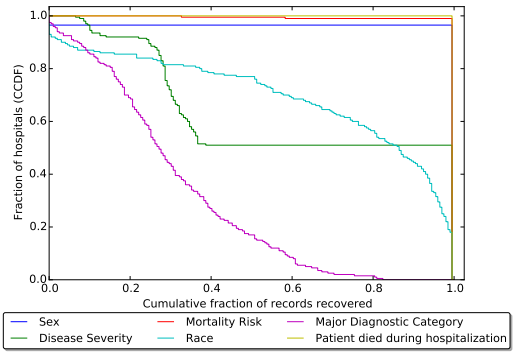


Figure 2: Results of ℓ_2-optimization on DTE-encrypted columns on 200 largest hospitals with 2009 HCUP/NIS as target data and Texas PUDF as auxiliary data

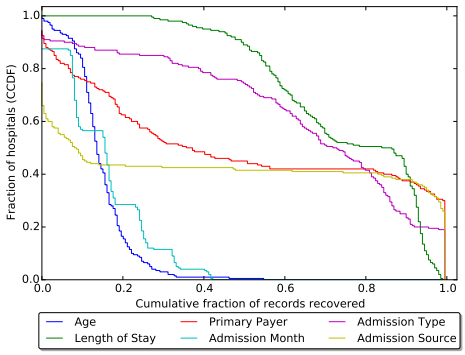
9.2 Attacks on OPE-Encrypted Columns

The sorting attack succeeds only if a column has density 1, meaning that all possible values of an attribute are present in both the target and the auxiliary data. If this condition holds, the sorting attack can recover *all* the OPE-encrypted cells in a column; otherwise it fails. Figure 4 shows the density for six selected attributes for the large and small 200 hospitals, respectively, of the 2009 HCUP/NIS dataset. For large hospitals, the density is 1 for 100% of the hospitals for Disease Severity and Mortality Risk and 90% of the hospitals for Admission Month. For small hospitals, the density is 1 for 95% of the hospitals for Disease Severity, Mortality Risk, and Admission Month. It follows that the sorting attack would recover 100% of the cells for these columns for these hospitals.

To evaluate the cumulative attack, we executed it over both large and small hospitals since the latter tend to have lower density on many attributes. Figure 5 shows the results. For large hospitals (Figure 5a) the attack performed

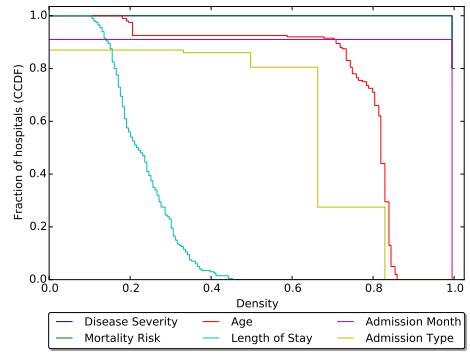


(a) First set of attributes

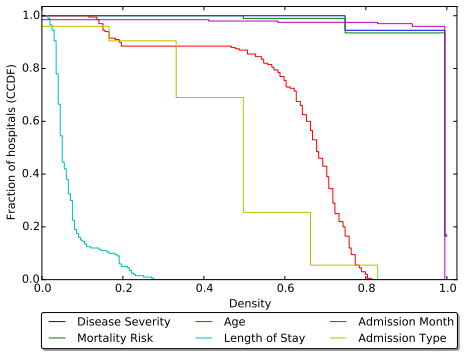


(b) Second set of attributes

Figure 3: Results of ℓ_2-optimization on DTE-encrypted columns on 200 largest hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data

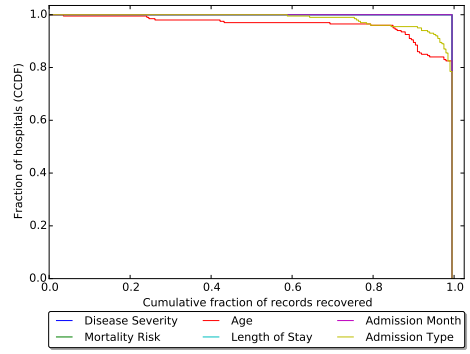


(a) Density for large hospitals

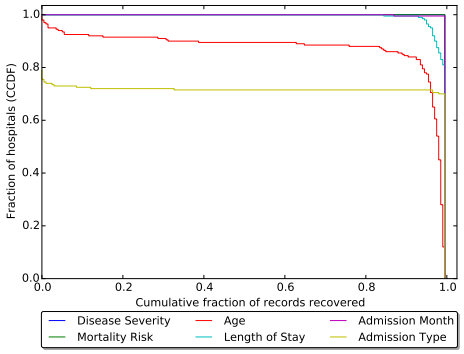


(b) Density for small hospitals

Figure 4: Density – Ratio of the number of values of an attribute present in a column to the total number of values of the attribute



(a) Results for the 200 largest hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data.



(b) Results for the 200 small hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data.

Figure 5: Results of Cumulative attack on OPE-encrypted columns

extremely well, even for low-density attributes. It recovered at least 80% of the patient records for 95% of the hospitals for all the attributes shown in Figure 5a. The attack recovered Admission Month, Disease Severity, and Mortality Risk for 100% of the patients for 100% of the hospitals; Length of Stay for at least 99.77% of the patients for 100% of the hospitals; Age for at least 99% of the patients for 82.5% of the hospitals ; and Admission Type for 100% of the patients for 78.5% of the hospitals.

For small hospitals (Figure 5b), despite the attributes' low densities, the attack still performed surprisingly well. It recovered Disease Severity and Mortality Risk for 100% of the patients for 100% of the hospitals; Admission Month for 100% of the patients for 99.5% of the hospitals; Length of Stay for at least 95% of the patients for 98% of the hospitals; Age for at least 95% of the patients for 78% of the hospitals; and Admission Type for 100% of the patients for 69.5% of the hospitals.

10. CONCLUSION

We study the concrete security of PPE-based encrypted database systems such as CryptDB and Cipherbase in a real-world scenario. We consider four different attacks and experimentally demonstrate that they can decrypt a large fraction of cells from DTE- and OPE-encrypted columns. We specifically show this for the case of EMR databases but we believe that the attacks would be as successful on a wide variety of databases as long as appropriate auxiliary information is available.

Acknowledgements

The authors thank Josh Benaloh, Melissa Chase, and Tadayoshi Kohno for helpful conversations and encouragement during the early stages of this work.

11. REFERENCES

[1] Fifth Annual Benchmark Study on Privacy and Security of Healthcare Data. http://www.ponemon.org/blog/criminal-attacks-the-new-leading-cause-of-data-breach-in-healthcare. Accessed: 2015-05-15.

[2] Google Encrytped Big Query. https://github.com/google/encrypted-bigquery-client.

[3] HCUP Databases. Healthcare Cost and Utilization Project (HCUP). 2008-2009. Agency for Healthcare Research and Quality, Rockville, MD. www.hcup-us.ahrq.gov/databases.jsp.

[4] Hospital Discharge Data Public Use Data File. http://www.dshs.state.tx.us/THCIC/Hospitals/Download.shtm.

[5] Hospital Inpatient Discharges (SPARCS De-Identified): 2012. https://health.data.ny.gov/Health/Hospital-Inpatient-Discharges-SPARCS-De-Identified/u4ud-w55t.

[6] OpenEMR. http://www.open-emr.org/. Accessed: 2015-05-15.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *SIGMOD*, pages 563–574, 2004.

[8] I. H. Akin and B. Sunar. On the difficulty of securing web applications using CryptDB. In *PriSec*, 2014.

[9] I. A. Al-Kadit. Origins of cryptology: The Arab contributions. *Cryptologia*, 16(2):97–126, 1992.

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

[11] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

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

[13] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *CRYPTO*, pages 578–595, 2011.

[14] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-inryp[(53g)1g(THCI)1(C-(in)1(g)-354(enc)1(ry)1(pwi)1(n.)ut)30(ea)1(ling)]TJ 0ob91(h)1(esca28(ersnl)1(ution)1(s.)-354(I)1(n)]T

[14] D. Bo9(a)1(,)-354(A)1(.)-354(Sa1(ette,)-354(Cry)1(pWZ)1(89(en)rol)1(utiFZ)1((ypU)1(t)1rt)1(hog)1()1(able)-353(encr)1:)1('N)1(eill.)]TJ 17.65De