Frequency-Hiding Order-Preserving Encryption

Florian Kerschbaum
SAP
Karlsruhe, Germany
florian.kerschbaum@sap.com

ABSTRACT

Order-preserving encryption allows encrypting data, while still enabling efficient range queries on the encrypted data. This makes its performance and functionality very suitable for data outsourcing in cloud computing scenarios, but the security of order-preserving is still debatable. We present a scheme that achieves a strictly stronger notion of security than any other scheme so far. The basic idea is to randomize the ciphertexts to hide the frequency of plaintexts. Still, the client storage size remains small, in our experiments up to 1/15 of the plaintext size. As a result, one can more securely outsource large data sets, since we can also show that our security increases with larger data sets.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; H.2.0 [**Database Management**]: General—*Security, integrity, and protection*

Keywords

Order-Preserving Encryption; Randomization; Indistinguishability

1. INTRODUCTION

Order-preserving encryption [5, 9, 10, 23, 30, 34] is a popular tool to encrypt data before outsourcing it. It allows to perform efficient range queries on the encrypted data. This makes it very suitable for achieving security and privacy in cloud computing.

The security of order-preserving encryption is still much debated. In their extended formal analysis [10] of their first scheme [9] Boldyreva et al. write that their work should not be interpreted as saying their scheme is "secure" or "insecure". Yet, we can make some observations about the security of order-preserving encryption. When encrypting only one plaintext order-preserving encryption can be perfectly secure (e.g. [23]) against a ciphertext-only attack, but when more plaintexts are encrypted (deterministic) order-preserving grows less secure. In any (deterministic) order-preserving encryption scheme the availability of ciphertexts for all (distinct) plaintexts lead to a simple attack. The sorted ciphertexts are mapped one-to-one to the sorted plaintexts.

This type of attack seems hard to prevent, since it is enabled by the ordering information implicit in the type of encryption. Furthermore, the ordering information is necessary for efficient search. This presents a problem for practitioners, since it is often hard to predict the final size of a data set and the number of distinct plaintexts and therefore there is no lower bound on the security of the encryption. In other cases it is known that all plaintexts in the domain will be encrypted and hence (deterministic) order-preserving is pointless.

In this paper we discuss a new option for increasing the security of order-preserving encryption: randomizing the ciphertexts. Our scheme is a new trade-off. We clearly increase security while preserving the functionality for most queries relying on the ordering information. However, we also increase client storage size and introduce a small error in some queries.

We proceed as follows. First, we present a new definition of security of order-preserving encryption: indistinguishability under *frequency-analyzing* ordered chosen plaintext attack. As we show in Section 7.2 the security captured in this definition is likely to increase when the data set size increases instead of decreasing. Our security notion is strictly stronger than indistinguishability under ordered chosen plaintext attack [9].

Second, we present a scheme that implements our security notion in Section 5 and prove it secure in Section 7.1. The basic idea is to randomize ciphertexts, such that no frequency information from repeated ciphertexts leaks. Note that randomized order-preserving encryption should still preserve the order of the plaintexts and hence any definition (and scheme) must take this information into account.

Third, we significantly reduce the client storage size by only approximating the security notion. This final scheme – we call imperfect frequency-hiding – has stronger security than any scheme before this paper, but also very acceptable client storage requirements. The ciphertexts in this scheme approximate a uniform distribution and hence improve against frequency-analyzing attacks. Furthermore, it is still indistinguishable under ordered chosen plaintext attack and the client storage size in our experiments was less than 0.1% of the database size.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813629.

We call an encryption scheme mutable, if its ciphertexts can or must be changed after initial encryption. Any scheme secure in our notion must be stateful and mutable. Consider the simplest order-preserving encryption scheme, namely the order itself as the ciphertexts, i.e. the plaintexts 5, 12, 7 are encrypted as 1, 3, 2. Given these ciphertexts and a new plaintext 8, the ciphertext 3 of plaintext 12 needs to be updated. Popa et al. [30] have shown that such updates are unavoidable for any scheme with polynomial ciphertext size secure against ordered chosen plaintext attacks.

With these kind of novel trade-offs – security under an increasing data set size, limited client storage cost and preservation of most order-based queries – we present a new option to the cloud security practitioner. Order-preserving encryption can be more trusted and yet remain practical. We summarize our contributions as follows.

- A definition of a new, stronger security notion for order-preserving encryption than indistinguishability under chosen plaintext attack.
- A scheme implementing this notion including compression mechanisms.
- An evaluation of security and client storage cost of this new scheme.

The remainder of the paper is structured as follows. In Section 2 we review related work, before we describe the problem in detail in Section 3. We define the security in Section 4 and explain our algorithms in Section 5. We present the final, practical scheme in Section 6. In Section 7 we summarize the results of evaluation – including the security proof – and present our conclusions in Section 8.

2. RELATED WORK

2.1 Order-Preserving Encryption Schemes

OPE scheme	IND-OCPA secure	frequency-hiding
[5]	No	No
[9]	No	No
[10]	No	No
[34]	No	No
[30]	Yes	No
[23]	Yes	No
[17]	No	Maybe
[35]	No	Maybe
Section 5	Yes	Yes
Section 6	Yes	Imperfect

Table 1: Overview of order-preserving encryption (OPE) scheme

Order-preserving encryption has been invented in the database community [5]. Agrawal et al. developed the first order-preserving encryption scheme. They assume that the distribution of the plaintexts is known. They then modify this distribution to match a fixed or randomly chosen distribution. While this also appears to be (somewhat) frequency-hiding it provides no formal security guarantees. We show that we can provably hide the information about the plaintext distribution.

The first formal security guarantee of order-preserving encryption has been provided by the cryptography community [9]. Boldyreva et al. introduce the notion of indistinguishability under ordered chosen plaintext attack. They also prove that no stateless scheme can achieve this notion and settle for an encryption scheme with the weaker security of a random order-preserving function. This scheme requires only storing a key on the client. Later they show that a random order-preserving function also achieves the security property of window one-wayness [10]. Furthermore, they give a scheme that achieves IND-OCPA security, but requires all plaintexts to be known in advance. Of course, if all plaintexts are known in advances, their order can be determined.

Teranishi et al. [34] present another order-preserving encryption scheme that is stateless, i.e. only requires storing a key on the client. Their idea is to introduce random large gaps in the ciphertexts and can show that their scheme using this technique can achieve a stronger notion than random order-preserving functions of partial-plaintext indistinguishability. Still, since they are stateless, they cannot achieve IND-OCPA security.

The first IND-OCPA secure order-preserving encryption scheme has been presented by Popa et al. [30]. Their scheme is stateful and requires storing information on an OPE server that we assume is ideally placed at the client site. They run a multi-round protocol which makes their scheme very inefficient due to the network delay. The size of the stored information is linear in the number of distinct plaintexts. Furthermore, their scheme is mutable, i.e. they need to update the order (and hence half of the ciphertexts), on almost any encryption operation. This has performance implications on the database server (operated in the cloud).

Kerschbaum and Schröpfer [23] provide the first efficient IND-OCPA secure order-preserving encryption scheme. They remove the need for a separate server, but also store information linear in the number of distinct plaintexts. Furthermore, they are able to reduce the probability of mutation to be negligible in n and reduce the number of rounds in the protocol between client and server to be constant instead of logarithmic in the number of distinct plaintexts. Their scheme has constant encryption cost in the average case.

So far, all mentioned order-preserving encryption schemes have been deterministic. Hildenbrand et al. [17] provide the first order-preserving encryption scheme which introduces some randomization. They divide the plaintext domain into disjoint sets and encrypt each set order-preservingly, but under a different key. They provide no formal security analysis, yet, since their scheme is stateless, it cannot even be IND-OCPA secure. We also note that their scheme has significant implications on the necessary rewriting of range queries on encrypted data. Wozniak et al. also add more randomization to a stateless order-preserving encryption scheme, but provide no security guarantee beyond random order-preserving functions [35].

Table 1 presents an overview of the discussed order-preserving encryption schemes. There is also a large number of other order-preserving encryption schemes [4, 19, 20, 24, 25, 26, 28, 38] which provide no formal, but rather ad-hoc security analysis, including the original proposal by Agrawal et al. [5]. Xiao et al. [37] define a notion based on nearby values, but it remains unclear how to enforce this in a practical setting.

2.2 Applications

Order-preserving encryption has many applications. Most notably database-as-a-service (DAS) [5, 15, 16, 31]. In DAS the database is outsourced to the cloud and values stored are encrypted before sent to the cloud. The database then performs its queries over encrypted data. Order-preserving encryption enables to perform range queries over an encrypted database without any changes to the database management system. CryptDB [31] has put forth the notion of adjustable encryption for databases which wraps around order-preserving encryption. Besides databases order-preserving encryption has many applications in general cloud software-as-a-service and web applications, e.g., business software and e-mail [1, 2].

2.3 Related Cryptographic Schemes

Order-preserving encryption is a special case of property-preserving encryption [6, 29]. Due to its applications to sorting and searching the order is a particularly useful property to preserve.

Searches on encrypted data can also be performed using other cryptographic schemes using modified, usually encryption-scheme specific search algorithms. Such cryptographic schemes are searchable, functional and homomorphic encryption.

Searchable encryption [33] achieves a stronger notion of security than order-preserving encryption. Searchable encryption for range queries has been presented in [12, 27, 32]. It uses a token of range boundaries generated by the secret key to match ciphertexts which are within the range of this token. Without the token ciphertexts are indistinguishable under chosen plaintext attack. Yet, searchable encryption schemes require a linear scan of the data, unless additional indexing information is provided. Lu [27] presents a searchable encryption scheme for ranges with logarithmic time-complexity, but its indexing information makes it as vulnerable as order-preserving encryption, since the proposed sorted tree reveals the order of all elements.

Searchable encryption is a special case of functional encryption. Functional encryption allows the evaluation of any function on a set of ciphertexts, such that the result of the function is revealed. Recently, functional encryption has been designed for general functions [14]. Specific functions, such as the inner product, have been proposed before [21]. Functional encryption can also reveal only the order while else remaining semantically secure [11].

Searching can also be implemented using homomorphic encryption where the search result remains unknown to the service provider. This implies if the result size is unbounded, the entire database needs to be transferred for any query. Fully homomorphic encryption [13] enables arbitrary search functions.

3. PROBLEM

Our scheme targets an outsourced, property-preserving encrypted database as an application. Consider the following example of this scenario: A owner of population data stored in a database encrypts his data using order-preserving encryption [5, 9, 10, 23, 30, 34] before outsourcing it to the cloud. The database can have the fields “first name”, “last name”, “birthday” and “gender” (and many more). When he wants to perform a range query, he sends the encrypted boundaries for the search term to the cloud service provider. The service provider performs a regular range query as he would on plaintext data using the encrypted boundaries, since the order of the plaintexts is preserved in the ciphertexts. The result is the set of encrypted data values in the queried range.

The performance and implementation flexibility advantages of order-preserving encryption are clear. In order for the server to perform a range query on a set of ciphertexts the client sends the ciphertexts for upper and lower bound of the range to the client. Range query on the server is then very efficient and can be performed in the same way as on plaintexts. All ciphertexts included in the range of the encrypted bounds correspond to plaintexts that are in the range. Furthermore, the server can optimize the query using all data structures for plaintext range queries. Hence such queries can, for example, be effortlessly integrated into existing database management systems [5, 31].

The original order-preserving encryption proposal [5] was designed for this scenario, but its security – and its security definition – leaves a lot to be desired. Obviously order-preserving encryption cannot be as strong as standard encryption, since by its definition information about the order is revealed. Some security researchers [10, 34] have shown that this ordering information also necessarily implies partial information about the distance of two plaintexts. Other security researchers [36] have argued that at least half of the bits leak. A complete characterization of the cryptanalytic implications of ordering information is still lacking.

If we consider the described outsourced database, i.e. structured data, another security problem of order-preserving encryption becomes apparent. In this case, each column of a database table is encrypted separately. For example, in a typical database table about people we would find fields such as first and last name, birthdate, gender, etc. – each encrypted with its own key. This allows queries on each individual field, e.g. all men born on the 4th of July.

Yet, in most of those fields inputs are not distributed uniformly. The last name Smith is certainly more common in the United States than Huxtable. This information about frequencies of inputs can be used in cryptanalysis of any database table encrypted with order-preserving encryption. In fact, Islam et al. [18] have shown that even partial frequency information – as it is the case in searchable encryption – can be used to decrypt entries with high probability. For many fields in a typical database information about the frequency distribution is readily available, e.g. many web sites list the most common first and last names per region.

The problem can be even more severe. In many fields we have many more database rows than values in the plaintext domain, i.e. it is likely that all plaintexts are encrypted at least once. In this case, the security of order-preserving encryption (up to this paper) breaks down. The ordered ciphertexts can simply be matched to the ordered plaintexts. Consider, for example, the field “gender”. It is unlikely that many database tables about people contain only men or women. Any order-preserving encryption would hence reveal the gender, since lexicographically “female” < “male”.[1]

In this paper we address this problem. We present an order-preserving encryption scheme that is randomized. Repeated plaintexts will (or can) become different ciphertexts. We define a much stronger notion of security for our scheme

[1]One could, of course, also sort in the reverse lexicographical order and have “male” < “female”.

than deterministic (order-preserving) encryption that implies hiding the frequency of any plaintext and no previous scheme can achieve it. We give a proof of security in this model and also show that our scheme has practical implications on the security of order-preserving encryption. Our scheme could be the first to thwart attacks as outlined above and still enable efficient range query as on plaintext data.

Consider the following example. When encrypting a database consisting of two women and two men, the gender attribute will be encrypted as 1, 2, 3, 4. The first two (1, 2) are ciphertexts for female and the second two (3, 4) are ciphertexts for male. The search can still be performed by the server on the encrypted data. When we search for all women, we search for the range $[1, 2]$, i.e. from the minimum ciphertext of the search term to the maximum ciphertext of the search term. When we search for the range $[female, male]$, we search for the range $[1, 4]$, i.e. from the minimum ciphertext of the lower bound to the maximum ciphertext of the upper bound.

As any order-preserving encryption scheme secure against ordered chosen plaintext attack we are stateful. Since the entropy of randomized ciphertexts is much higher than those of deterministic ciphertexts it is not surprising that in our randomized order-preserving encryption scheme we need to store much more state, i.e. much more information on the client. In fact, it is a major challenge to design a scheme with our security guarantees that does not imply storing all plaintexts on the client. We employ a number of data compression techniques described in Section 5.2 that reduce the amount of information stored on the client. We achieve a data compression of roughly a ratio of 15 for realistic data sets. These type of compression techniques make our scheme somewhat practical while still improving the security of ciphertexts. Furthermore, we present a fully practical tradeoff that is less secure (imperfect), but can easily achieve compression ratios of several thousands. We consider this the final scheme to be applied.

4. DEFINITIONS

A (stateful) order-preserving encryption scheme Π_{FHOPE} consists of the following three algorithms:

- $S \leftarrow \text{KeyGen}(\lambda)$: Generates a secret state S according to the security parameter λ.
- $S', y \leftarrow \text{Encrypt}(S, x)$: Computes a ciphertext y for plaintext x and updates the state from S to S'.
- $x \leftarrow \text{Decrypt}(S, y)$: Computes the plaintext x for ciphertext y based on state S.

We call the encryption scheme correct if $\text{Decrypt}(\text{Encrypt}(S, x)) = x$ for any valid state S and x. We call it order-preserving if the order is preserved, i.e. $y_i \geq y_j \Longrightarrow x_i \geq x_j$ for any i and j.

We now need to define the security guarantees of our frequency-hiding order-preserving encryption scheme, i.e. why it is frequency-hiding. We start with the definition of indistinguishability under ordered chosen plaintext attack (IND-OCPA) from [9]. IND-OCPA is – so far – the strongest definition of security of order-preserving encryption. It is achieved by the encryption schemes of [23, 30].

IND-OCPA security is defined by a game between a challenger and an adversary. The adversary prepares two sequences of plaintexts where both sequences have the same order and each plaintext is distinct within its sequence. He sends those two sequences to the challenger. The challenger encrypts one sequence and sends the ciphertexts to the adversary who guesses which of the two sequences it is.

A simple idea to extend the definition of IND-OCPA to randomized ciphertexts would be to lift the restriction of distinction of plaintexts. Yet, this is insufficient as the following example shows. In case of two distinct plaintexts (male and female), the only sequences with different frequencies that need to be indistinguishable are those that consists of only one plaintext. All other sequences of the same order have the same frequencies of plaintexts and hence this frequency information may leak from the ciphertexts, since it is present in both plaintext sequences.

Still, the random choices of the encryption algorithm cannot be completely independent of the sequence of the plaintexts (as it is the case for indistinguishability under chosen plaintext attack for non-order-preserving encryption), since the order of the plaintext sequences needs to be preserved. We therefore first define a *randomized order*. Loosely speaking, a randomized order is a permutation of the numbers $1, 2, \ldots, n$, which is ordered according to X and ties are broken randomly.

Definition 1. *Let n be the number of not necessarily distinct plaintexts in sequence $X = x_1, x_2, \ldots, x_n$ $(\forall i.x_i \in \mathbb{N})$. A* randomized order $\Gamma = \gamma_1, \gamma_2, \ldots, \gamma_n$ $(\forall i.1 \leq \gamma_i \leq n, \forall i, j.i \neq j \Rightarrow \gamma_i \neq \gamma_j)$ *of sequence X it holds that*

$$\forall i, j.x_i > x_j \Longrightarrow \gamma_i > \gamma_j$$

and

$$\forall i, j.\gamma_i > \gamma_j \Longrightarrow x_i \geq x_j$$

Consider the following example. Let the plaintext sequence be $X = 1, 2, 2, 3$. Possible randomized orders are $\Gamma_1 = 1, 3, 2, 4$ and $\Gamma_2 = 1, 2, 3, 4$. There are many plaintext sequences with randomized order Γ_1 not all of which have the same plaintext frequencies. Examples with any frequency of 1 are $X'_1 = 2, 2, 2, 3$, $X'_2 = 1, 2, 1, 3$, $X'_3 = 1, 1, 1, 3$ or $X'_4 = 1, 1, 1, 1$. There are many more.

The goal of our security definition is that the ciphertexts only leak the randomized order of the plaintexts, i.e. two sequences with the same randomized order – but different plaintext frequencies – should be indistinguishable. Note that the randomized order does not contain any frequency information, since each value always occurs exactly once. We define the following security game $Game_{FAOCPA}(\lambda)$:

1. The adversary chooses two sequences X_0 and X_1 of n not necessarily distinct plaintexts, such that they have at least one common randomized order Γ.[2] He sends them to the challenger.
2. The challenger flips an unbiased coin $c \in \{0, 1\}$, executes the key generation $\text{KeyGen}(\lambda)$, and encrypts X_c as Y, i.e. $S_i, y_i \leftarrow \text{Encrypt}(S_{i-1}, x_i)$. He sends the ciphertexts Y to the adversary.
3. The adversary outputs a guess $c^\star$ of c, i.e. which of the two sequences it is.

[2]Multiple common randomized orders are possible and allowed.

We can now define security against frequency-analyzing ordered chosen-plaintext attack.

DEFINITION 2. *We say a (stateful) order-preserving encryption scheme Π_{FHOPE} is IND-FAOCPA secure against frequency-analyzing ordered chosen plaintext attack if the adversary $\mathbb{A}$'s advantage of outputting c in $Game_{FAOCPA}(\lambda)$ is negligible in λ, i.e.*

$$Pr[\mathbb{A}(Game_{FAOCPA}(\lambda)) = c] < \frac{1}{2} + \frac{1}{poly(\lambda)}$$

IND-FAOCPA security is strictly stronger than IND-OCPA security, since the randomized order of distinct plaintexts is equal to its order. Hence IND-FAOCPA implies IND-OCPA, but IND-OCPA does not imply IND-FAOCPA, since as argued before the frequency of repeated plaintexts may leak, which may be used for (simple) cryptanalysis. After describing the algorithms implementing our frequency-hiding order-preserving encryption scheme we give a proof of security in the IND-FAOCPA model in Section 7.1.

4.1 Performing Queries on Randomized Order

The rewriting strategy for queries on a randomized order is somewhat different than on a deterministic order. We present an informal discussion. When querying for equality, e.g. with a `WHERE name = 'Alice'` clause, the query is translated into a range query. Let $min_C(x)$ be the minimal ciphertext in numerical order for a plaintext x. Similarly, let $max_C(x)$ be its maximal ciphertext. The query clause then becomes `WHERE name >=` min_C`('Alice') AND name <=` max_C`('Alice')`.

For range queries a similar transformation can be applied. The query clause `WHERE wage >= 10 and wage < 20` becomes `WHERE wage >=` min_C`(10) and wage <` min_C`(20)`.

More difficult are queries with comparison of database values (as opposed to constants). The problem arises, since a randomized order mixes the results of greater-than (>) and greater-or-equal (>=) comparisons and we cannot apply the min_C and max_C operators on the database. Consider the following the query clause `WHERE wage > (SELECT wage WHERE name = 'Alice')` which selects all employees with a wage greater than Alice's. Naively executing this query on a randomized order may introduce some additional employees which have the same salary as Alice. In this case, it can be solved by executing the sub-select first, applying the max_C operator on the client and then executing the query. An automated rewriting algorithm for such queries is presented in [22]. For a few, rare queries this download to the client can imply a significant overhead.

5. ALGORITHMS

We initially proceed as the deterministic order-preserving encryption scheme of [23] and insert plaintexts into a sorted binary tree in the order they are encrypted. Differently from their scheme in our algorithm the data structure on the client side is very important. We begin to describe the algorithm using a binary search tree with dynamically allocated leaves and pointers. Later we describe how we compress the tree in order to save space compared to storing all plaintexts on the client.

It is important to note that although our tree is dynamic, there is a threshold depth beyond which we need to rebalance. This threshold depth is determined by the ciphertext length and may be exceeded in rare cases of ordering of plaintexts to encrypt. The scheme in [23] contains a proof that the probability of exceeding this ciphertext length is negligible in case of uniformly distributed input. Their Theorem 6 also holds for our encryption scheme.

We begin by encrypting each plaintext as the mean value of the ciphertexts for the next plaintext just smaller and greater. We insert this plaintext, ciphertext pair in our binary search tree. We then handle plaintexts that have already been encrypted differently. In this case, i.e. when inserted plaintext and to be encrypted plaintext are equal, we traverse the tree in a randomly chosen fashion and insert the new plaintext as a leaf. We first define the basic data structure of the tree in Algorithm 1.

We fix the following parameters for all of our algorithms: Let n be the number of not necessarily distinct plaintexts to be inserted. Let N be the number of distinct plaintexts. Hence the bitlength is $k = \lceil \log_2 N \rceil$ in Algorithm 1. Let λ be the security parameter and the expansion factor of the ciphertext. Hence $l = \lambda k$ in Algorithm 1. The security parameter can also be used to determine the length of the seed for pseudo-randomness, although this seed should be much (but polynomial) longer than the expansion factor.

Algorithm 1 Tree Structure $Tree$

```
struct Tree {
  Tree left;
  Tree right;
  bitstring<k> plain;
  bitstring<l> cipher;
}
```

We now proceed for encryption as in randomized Algorithm 2. We denote the binary search tree – the state – as a set T of nodes $\{t\}$ and do not list it as a separate input. Furthermore we add inputs t for the current node and min and max for the lower and upper limit in order to enable recursion.

Initially the function is called with the plaintext x to be encrypted, the root of the tree T, $min = -1$ and $max = 2^{\lambda log_2 n}$. We create a new root, if none exists. If n is unknown, it needs to be estimated. We denote this initial call as ENCRYPT(x) leaving out the state S from our definition of a frequency-hiding order-preserving encryption scheme Π_{FHOPE}. Recall that according to [23] the probability of rebalancing is negligible in n for uniform inputs, if $\lambda > 6.4$. Furthermore, Kerschbaum and Schröpfer have shown that for real-world, non-uniform inputs smaller λ are likely to suffice [23].

An example of encryption is the plaintext sequence $X = 1, 2, 1, 3$ with $min = -1$ and $max = 128$. Then the first two plaintexts are deterministically $y_1 = 64$ and $y_2 = 96$, respectively. The ciphertext for the third plaintext $x_3 = 1$ is randomized, since it is a repeated plaintext. It could be either $y_3 = 32$ or $y_3 = 80$ depending on the random coin. The fourth ciphertext is deterministic $y_3 = 112$.

The function RANDOMCOIN draws uniformly distributed random coins from a keyed pseudo-random function. We now describe the decryption algorithm in the simplified data structure in Algorithm 3. The idea is to simply traverse the tree and find the matching ciphertext. We will later

Algorithm 2 Encryption ENCRYPT

Input: x, t, min, max

Output: y

State: Sorted binary tree T of nodes $\{t\}$
Initialization: T is empty

1. If $x = t.plain$, then $coin$ =RANDOMCOIN() $\in \{0,1\}$, else $coin = \perp$
2. If $x > t.plain$ or $coin = 1$, then

3a. If $t.right \neq$NULL, then

4. Return ENCRYPT$(x, t.right, t.cipher, max)$

3b. Else

5. If $max - t.cipher < 2$, then return REBALANCE$(x, -1, n)$
6. Insert $t.right$ = NEW $Tree\left(x, t.cipher + \lceil \frac{max - t.cipher}{2} \rceil\right)$ and return $t.right.cipher$.

3c. EndIf

7. If $x < t.plain$ or $coin = 0$, then

8a. If $t.left \neq$NULL, then

9. Return ENCRYPT$(x, t.left, min, t.cipher)$

8b. Else

10. If $t.cipher - min < 2$, then return REBALANCE$(x, -1, 2^{\lambda log_2 n})$
11. Insert $t.left = Tree(x, min + \lceil \frac{t.cipher - min}{2} \rceil$ and return $t.left.cipher$.

8c. EndIf

describe how to compress the data structure, such that not all encrypted plaintexts have to be stored on the client. We add the recursion input t in Algorithm 3 compared to our formal definition. A decryption call DECRYPT(x) forwards to DECRYPT$(x, root(T))$.

Algorithm 3 Decryption DECRYPT

Input: y, t

Output: x

State: Sorted binary tree T of nodes $\{t\}$

1. If $y > t.cipher$, then return DECRYPT$(y, t.right)$
2. If $y < t.cipher$, then return DECRYPT$(y, t.left)$
3. Return $t.plain$

Algorithm 4 is an intuitive to understand version of the rebalancing algorithm and more efficient versions can be engineered if needed.

Algorithm 4 Rebalancing the search tree REBALANCE

Input: x, min, max

Output: y

State: Sorted binary tree T of nodes $\{t\}$

1. Let $X = \{t.plain\} \cup \{x\}$
2. Sort X in ascending order
3. Let $T = \{$ NEW $Tree(x_{\lceil \frac{\lfloor X \rfloor}{2} \rceil}, min + \lceil \frac{max - min}{2} \rceil)\}$
4. Let $y = T.plain$
5. Let $X' = \{x_j | x_j < y\}$
6. Let $X'' = \{x_j | x_j > y\}$
7. Let y' =ENCRYPT$(x_{\lceil \frac{\lfloor X' \rfloor}{2} \rceil}, T_0, min, max)$
8. Let y'' =ENCRYPT$(x_{\lceil \frac{\lfloor X'' \rfloor}{2} \rceil}, T_0, min, max)$
9. Recursively iterate on line 5 with both y' and y'' as y and X' and X'' as X, respectively.
10. Find x in T and return $T_x.cipher$

5.1 Examples

Before describing space-saving measures we give some examples of trees that can emerge when inserting random plaintexts. We consider the example of a binary plaintext domain again, e.g. male and female. We have 4 plaintexts $x_i \in \{0,1\}$. We insert the following sequence $X = 0, 1, 0, 1$. We set the random coins to the sequence $1, 0$. The resulting sequence of trees is depicted in Figure 1.

Figure 1 is divided into four subfigures numbered 1 to 4. Each depicts the search tree after inserting one more element of the sequence with the new node in red. In subfigure 3 we see for the first time a plaintext repeating, but inserted beneath a parent with a different plaintext. We can trace the algorithm as follows: When inserting 0 for the second time, the algorithm encounters a 0 at the root. Due to the random coins it traverses to the right, where it encounters a 1 and must make a deterministic choice leading to the new leaf. In subfigure 4 we see that the next 1 inserted and plaintext nodes interleaving. In larger plaintext domains even intermediate elements can be placed at lower nodes.

Of course, repeated plaintexts can also be placed under parents with the same plaintext. If we insert two more elements $0, 1$ with random coins $0, 1$, the search tree will look as in Figure 2.

5.2 Compression

When storing the plaintexts in this simple tree as described we will use more space than when storing only the plaintexts. We need to accommodate the ciphertext and

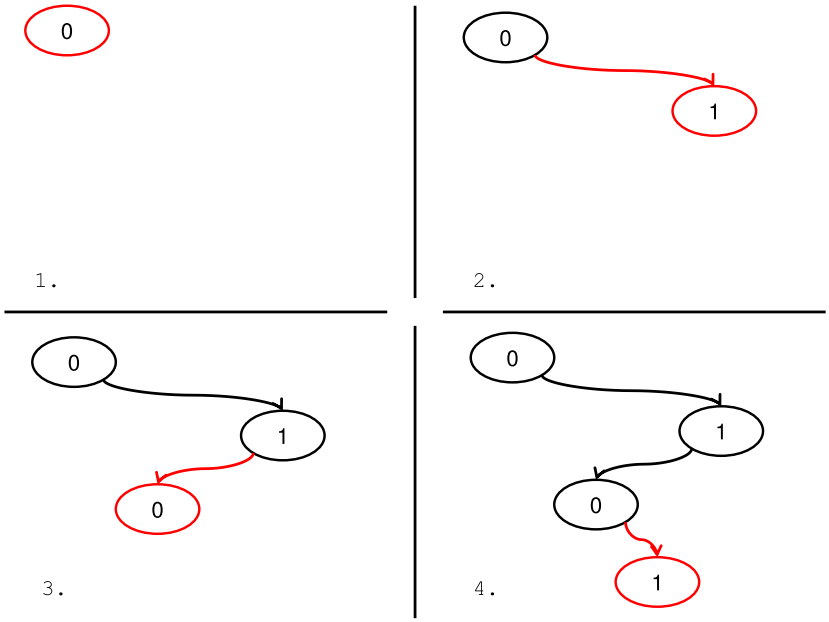

Figure 1: Growing Search Tree for Sequence $0, 1, 0, 1$

the pointers in the tree. In this section we describe how we compress the tree.

First, note that the ciphertext does not need to be stored. It can be computed dynamically as we traverse the tree. We can already save the space for this field, but we still need to compress the tree.

We can compress the plaintext using regular dictionary compression [3, 8, 39] and store repeated values as the index into the dictionary. This method is very effective in saving space for column-store databases. Moreover, we can further compress subtrees of repeated values. As we have seen in the examples before, repeated plaintexts can have parents of the same or a different plaintexts. We call subtrees of the same plaintext *clusters*.

In a cluster we do not need to store the plaintext for each node, instead we just store it once in the root of the cluster. While suppressing a field in a structure is supported in polymorphic object-oriented languages using inheritance we compress the tree even further. Instead of storing the tree structure we only store its traversal thereby compressing the size of the pointers. Let the size of the cluster be M. First we traverse the tree in post-order traversal order and assign the nodes the numbers from 1 to M in ascending order, i.e. the post-order traversal is now $1, 2, 3, \ldots$. We then traverse the tree in in-order traversal order and record the post-order-traversal numbers. As an example, a simple binary tree with one root and two leaves then becomes $1, 3, 2$.

We only store this traversal sequence. Every time we encounter a cluster we expand it to the full tree before proceeding in the traversal. This can be done in linear time and space using the algorithm of Andersson and Carlsson [7]. After proceeding to the next node with different plaintext we destroy the temporarily expanded tree. Hence, we save not only on the plaintext storage, but also on the pointer information in the tree. We only need to store (long) pointers between nodes of different plaintexts.

We evaluate the effectiveness of our compression in Section 7.4.

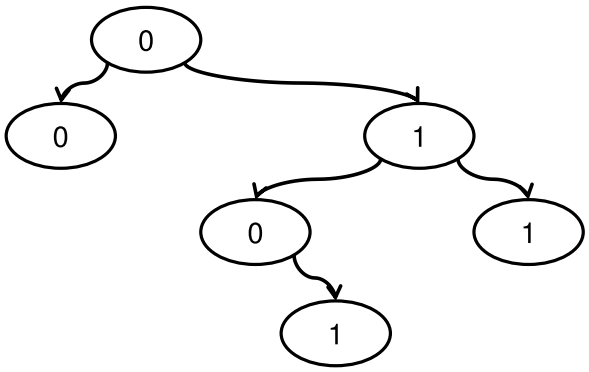

Figure 2: Possible Search Tree for Sequence $0, 1, 0, 1, 0, 1$

6. IMPERFECT FREQUENCY-HIDING

So far our scheme achieves IND-FAOCPA security (see proof in Section 7.1). This has security advantages, but also implies high storage space requirements at the client. In some application scenarios lower security and (significantly) lower space requirements can be desirable. For example, it is often enough if the distribution of ciphertexts is not perfectly hiding, but approaches a known distribution, most notably the uniform distribution. Islam et al. suggest and positively evaluate against guessing attacks approximating the expected frequency by a uniform distribution [18]. We can

approximate a uniform distribution – with some repeated ciphertexts – using our frequency-hiding encryption scheme and a heuristic pre-filter. This pre-filter hence trades some security for significant space savings.

The basic idea is as follows: For each distinct plaintext $x \in X$ we maintain the numbers c_x of times it has occurred and the numbers d_x of distinct ciphertexts (for this plaintext) already generated. We check whether a new plaintext will skew the distribution, such that the number of plaintexts per ciphertext *for this plaintext* is off by a factor more than r from the number of plaintexts per ciphertext *for all plaintexts*. If this is the case, we always generate a new ciphertext. Otherwise, we flip a biased coin (with probability p) and decide probabilistically whether to generate a new ciphertext. This randomizes the number of times a ciphertext can be observed at the cloud provider compared to the deterministic, smallest number of plaintexts. If we do not generate a new ciphertext, we uniformly chose among the already generated ones. Let $C = \{c_x | x \in X\}$ and $D = \{d_x | x \in X\}$ be the set of counters described before. Our pre-filter is parameterized by $r > 1$ and $0 \leq p \leq 1$. The entire filter is described in Algorithm 5.

Algorithm 5 Filtering for Imperfect Frequency-Hiding

Input: C, D, r, p, x

Output: y

State: $T = \{t_j\}$ in a sorted binary tree

1. Let c_x be the number of occurrences of plaintext x. Let d_x be the number of distinct ciphertexts for x.
2. If
$$\frac{c_x + 1}{d_x} > r \frac{\sum_{j \in X} c_j}{\sum_{j \in X} d_j}$$
then increment c_x, d_x and return $y = \text{ENCRYPT}(x)$.
3. Choose a random coin $c \in \{0, 1\}$ with $Pr[c = 1] = p$.
4. If $c = 1$, then increment c_x, d_x and return $y = \text{ENCRYPT}(x)$.
5. Build set $Y = \{t_j.cipher | t_j.plain = x\}$. Uniformly select y among Y, increment c_x and return y.

The parameter p also balances storage cost and security. For $p = 1$ the result of the filtered scheme is the same as the perfectly frequency-hiding order-preserving encryption scheme. For $p = 0$ and a uniform distribution of plaintexts the filtered scheme grows near the deterministic encryption of [23]. Nevertheless, for skewed plaintext distributions (and finite r) it approximates a uniform distribution and is hence more secure.

It is noteworthy that this filtered scheme is still IND-OCPA secure in the worst case and hence more secure than many other order-preserving encryption schemes. Still, it is not secure against frequency-analyzing ordered chosen plaintext attack, since some plaintext repetitions leak (and hence some frequency information leaks).

We evaluate the effectiveness of this security-reducing compression also in Section 7.4.

7. EVALUATION

We implemented our frequency-hiding order-preserving encryption scheme in Java 1.7. We run in the 64-Bit Hotspot Server VM on an Intel Core i5-4300 CPU with 1.9-2.5GHz and 16GB RAM.

7.1 Security Proof

We give a proof of security against frequency-analyzing ordered chosen plaintext attack of our encryption scheme of Section 5. We prove by constructing a simulator of the encryption that produces identical outputs for each of the two challenge sequences. Hence, the computational indistinguishability stems from implementing the random source as a pseudo-random function and not any other hardness assumption. Note that our scheme also trivially fulfills the notion of same-time security defined in [30] when ciphertexts are never deleted from the state, but only from the database.

THEOREM 3. *Our encryption scheme is secure against frequency-analyzing ordered chosen plaintext attack.*

PROOF. Our simulator proceeds as follows. The adversary sends the two plaintext sequences X_0 and X_1 of length n. The simulator needs to randomly select a randomized order common to X_0 and X_1 from the set of all common randomized orders. It does this using Algorithm 6.

Algorithm 6 Selecting A Randomized Order

Input: X_0, X_1, s.t. $|X_0| = |X_1| = n$

Output: Γ

1. Sort X_0 and X_1 in ascending order into Z_0 and Z_1, respectively.
2. Build a list W of triples $\langle z_{0,i}, z_{1,i}, i \rangle$.

3a. For each pair $x_{0,j}, x_{1,j}$ for $1 \leq j \leq n$ do

4. Build set $U = \{i | \langle z_{0,i}, z_{1,i}, i \rangle \in W \wedge z_{0,i} = x_{0,j} \wedge z_{1,i} = x_{1,j}\}$.
5. Uniformly select γ_j in U.
6. Remove $\langle x_{0,j}, x_{1,j}, \gamma_j \rangle$ from W.

3b. End for each

THEOREM 4. *Algorithm 6 runs in time $O(n^2)$.*

PROOF. The loop (from line 3*a* to 3*b*) iterates n times. Each operation (lines 4, 5, and 6) takes at most n operations on the set W or U, respectively. Hence the overall time is bound by $O(n^2)$ operations. □

Theorem 4 implies that our simulator is polynomial time and computational indistinguishability is feasible, even if we implement the random choices using a pseudo-random function. Once the randomized order Γ has been selected, the simulator needs to compute the ciphertexts. It simulates the RANDOMCOIN function in Algorithm 2 using a simulator of the random source, i.e. the random source could be replaced by hash functions (a random oracle).

First, it uniformly random selects plaintext sequence $X \in \{X_0, X_1\}$. During encryption, as it store plaintexts in the

state S – the binary search tree T – of the encryption function, it keeps track of the randomized order values γ_i. As it encrypts x_i and finally stores x_i in T, it also stores γ_i, i.e. for each value v in the tree T we also know γ_v. We denote γ_i the randomized order value of the current input plaintext x_i and γ_v the randomized order value of the current node in the tree traversal. If during encryption of x_i, the simulator encounters a call to RandomCoin, it calls Algorithm 7 with γ_i and γ_v instead.

Algorithm 7 Programmed Random Oracle for RandomCoin

Input: γ_i, γ_v

Output: 0 or 1

1. If $\gamma_i > \gamma_v$, then return 1.
2. Return 0.

Theorem 5. *The output of Algorithm 7 in the simulator is indistinguishable from uniform random coins.*

Proof. While the output of Algorithm 7 is deterministic, the choice of γ_i and γ_v is determined by Algorithm 6. Algorithm 6 uniformly selects a randomized order among all possible randomized orders. Since each randomized order results in a different binary search tree and each output of the RandomCoin function also produces a different binary search tree, the output of Algorithm 7 is indistinguishable from uniform random coins. □

This completes our simulator. The simulator produces the same output for both sequences, runs in polynomial time and its random output is indistinguishable from uniform random coins. Hence the probability that the adversary wins $Game_{FAOCPA}(\lambda)$ against our encryption of Section 5 is negligible in λ larger than $\frac{1}{2}$.

7.2 Indistinguishable Sequences

We can try to estimate the effectiveness of the two different security models – IND-OCPA and IND-FAOCPA – by estimating the size of the set of indistinguishable plaintext sequences. Two indistinguishable sequences can be given as a challenge in the security game and result in an indistinguishable ciphertext challenge. Clearly, the larger the set of sequences indistinguishable from a given one, the more secure the encryption. Under standard IND-CPA security any two sequences of plaintexts are indistinguishable.

An IND-OCPA secure scheme may leak the order of the plaintext sequence. Note that IND-OCPA security is also an upper bound for weaker order-preserving encryption schemes such as [9, 34, 35]. We give the number of indistinguishable sequences in Theorem 6.

Theorem 6. *Let D be the number of distinct plaintexts in the plaintext domain. For any sequence of N distinct plaintexts there are $\binom{D}{N}$ indistinguishable sequences under IND-OCPA.*

Proof. The order of a sequence of N distinct plaintexts is a subset of the domain. There are $\binom{D}{N}$ possible subsets and hence as many indistinguishable sequences. □

An IND-FAOCPA secure scheme leaks the *randomized* order of the plaintext sequence. Hence, an indistinguishable sequence must have at least one common randomized order. Different sequences may have different numbers of indistinguishable sequences. Consider encrypting two distinct plaintexts 0 and 1 ("female" and "male"). The sequence $X_0 = 0, 0$ has 4 indistinguishable sequences (including itself), but the sequence $X_1 = 0, 1$ has only 3. Consequently, we can only estimate the expected number of indistinguishable sequences. We give a lower bound of the expected number of indistinguishable sequences in Theorem 7.

Theorem 7. *Let D be the number of distinct plaintexts in the plaintext domain. For a uniformly chosen plaintext sequence of size n with N distinct plaintexts there are at least $N\,(D-1)^{\frac{n}{N}}$ indistinguishable sequences expected.*

Proof. For any set $\tilde{X}$ of unique plaintext x we can flatten the sub-sequence $x_i < x$ to 1 and the sub-sequence $x_j > x$ to D. Then there are $D-1$ additional choices for any element (without counting doubles) and an expected number of $\frac{n}{N}$ such elements. The order of choices within its repetitions is independent, since all are from the same plaintext and there N such sets $\tilde{X}$. Hence there are at least $N\,(D-1)^{\frac{n}{N}}$ expected indistinguishable sequences. □

We emphasize that the expected number of distinct plaintexts is $E[N] = D\left(1 - \left(\frac{D-1}{D}\right)^n\right)$. This number approaches D as n grows. Hence the number of indistinguishable sequences under IND-OCPA approaches 1 as n grows. To the contrary either $\frac{n}{N}$ or N $(N \leq n)$ approaches infinity as n grows and consequently the number of indistinguishable sequences under IND-FAOCPA also approaches infinity as n grows.

We conclude that the encryption secure against IND-FAOCPA remains secure (even increases security) for long sequences whereas the security of encryption secure against (only) IND-OCPA deteriorates. Hence, the encryption of long (and growing) sequences of plaintexts is clearly more secure under an IND-FAOCPA secure scheme than under a scheme that is only IND-OCPA secure. This applies to any deterministic order-preserving encryption scheme, since all grow to only one indistinguishable sequence – the plaintext domain.

7.3 Statistical Security

Each order-preserving encryption scheme – deterministic or randomized – can be described as a monotonically increasing function. Hence a simple statistical cryptanalysis technique is to model the encryption as a linear function. Given two plaintext, ciphertext pairs the parameters of this function can be estimated.

We can estimate the effectiveness of this attack by measuring the correlation between the ciphertext and plaintext. For a set of plaintext, ciphertext pairs we compute the Pearson correlation coefficient r. The Pearson correlation coefficient measures a linear relation between two sets of random variables.

We encrypt $n = 16384$ (possibly repeated) plaintexts independently uniformly chosen from a set of $N \in \{64, 256, 1024, 4096, 16384, 65536\}$ distinct plaintexts. We set the ciphertext space to 60 bits.[3] We make 100 experimental runs and

[3]As expected with 60 bits ciphertext space in none of our experimental runs we had to rebalance the search tree.

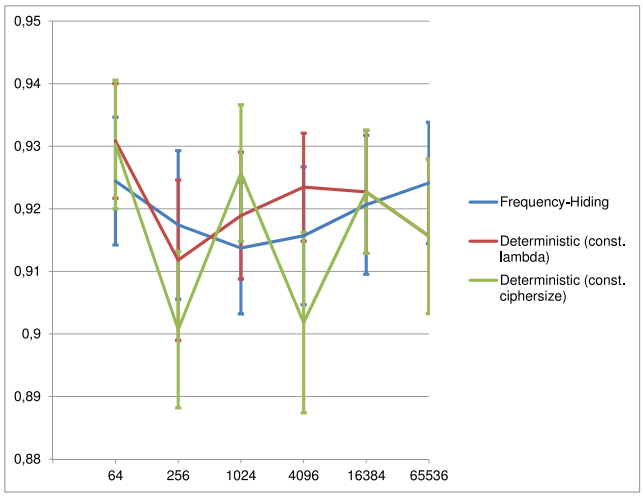

Figure 3: Pearson correlation coefficient over the number of different plaintexts in the domain

compute the correlation coefficient r.t' We also compute the 90% confidence interval.

We compare our frequency-hiding order-preserving encryption scheme to the deterministic order-preserving encryption scheme of Kerschbaum and Schröpfer [23]. Their scheme is IND-OCPA secure and they have shown that their correlation is often lower than the one of the schemes by [9, 30]. We use the same parameters for n and N, but consider two different options for the ciphertext length. In one case we keep the expansion factor $\lambda = 5$ fixed to the same value, i.e. the ciphertext length is $5 \log_2 N - 10$ bits. In the other case we keep the ciphertext length fixed at 60 bits.

The result is depicted in Figure 3 with the confidence intervals as error bars. We see that the confidence intervals for the different data series clearly overlap. Hence we cannot conclude that randomized order-preserving encryption improves defense against this cryptanalytic attack. Yet, we can conclude that randomized order-preserving encryption is no weaker under this attack.

7.4 Client Storage Space

As mentioned client storage space is a limiting resource for frequency-hiding order-preserving encryption. We encrypt experimental data sets and measure the storage consumption of our client data structures.

First, we encrypt a database table with all people in Germany. The German population is roughly $n = 8 \times 10^7$. We encrypt the fields "first name", "last name", "birthdate" and "gender". We estimate the distribution of those fields using public information[4]. We performed 20 test runs and report the average storage size. The extent of the 90% confidence is always below 0.05% of the measured value. In none of the experimental runs we had to rebalance the tree. Our results are summarized in Table 2.

Overall we achieve a compression ratio of almost 15. This may suffice for small to medium databases with most clients. We also see that compression performs better for larger field sizes like in first and last name. Dictionary compression performs better for these fields than for small fields. In fact, for small field sizes – as in the gender attribute – our compression techniques may even increase the storage space.

Second, we measure the effectiveness of search tree compression by itself. We consider already dictionary compressed values, i.e. field size is equal to $\lceil log_2 N \rceil$ bytes, and measure the size of our search tree as described in Section 5. We encrypt $n \in \{10^5, 10^6, 10^7\}$ plaintexts uniformly chosen among $N \in \{16, 256, 4096, 65536, 1048576\}$ distinct plaintexts. We performed 20 test runs and report the average storage size. The extent of the 90% confidence interval is always below 0.1% of the measured size. In none of our experimental runs we had to rebalance the tree. We show the computed compression ratio in Figure 4.

We see that search tree compression performs up to a compression ratio of roughly 2 around the center number of distinct plaintexts ($N \sim \sqrt{n}$). We conclude that for suitable plaintext domains our compression technique performs well, but for others different techniques may be needed. We therefore also evaluate imperfect frequency-hiding order-preserving encryption as in Section 6.

We use a plaintext domain of $N = 2$, since our overall compression performed worst with this parameter. We

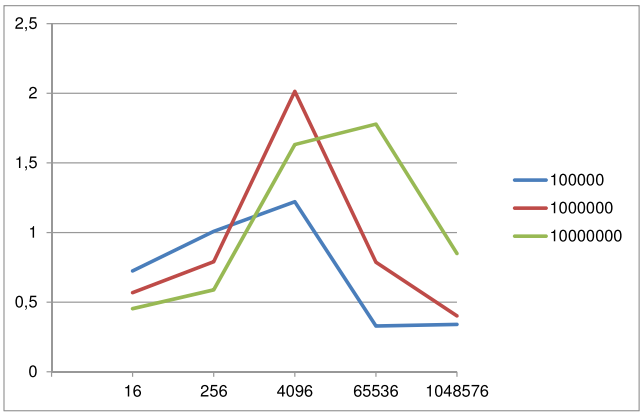

Figure 4: Compression ratio of dictionary compressed value over the number of distinct plaintexts N for different number of plaintexts n

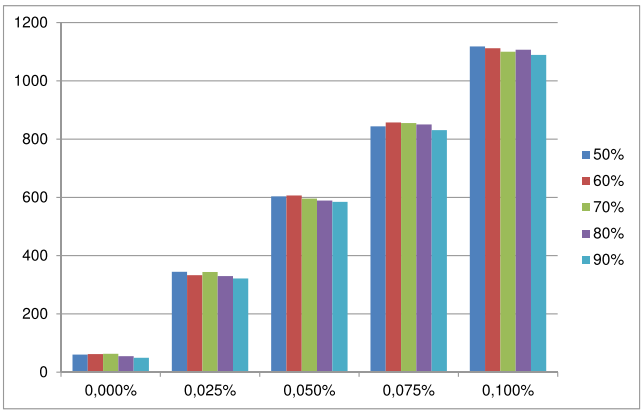

Figure 5: Number of distinct ciphertexts over the parameter p for different plaintext distributions ($r = 1.025$)

[4] `http://de.wikipedia.org/wiki/Familienname` `http://www.beliebte-vornamen.de/lexikon`

Field	N	Field Size (byte)	Plaintext Size (Mbyte)	Compressed Size (Mbyte)	Compression Ratio
First Name	10000	49	3738.4	106.1	35.2
Last Name	1000000	50	3814.7	104.8	36.4
Birthdate	40000	4	305.2	90.4	3.4
Gender	2	1	76.3	232.4	0.3
Total		104	7934.6	533.8	14.9

Table 2: Compression Effect on Table of People in Germany

encrypt $n = 10^6$ plaintexts with probability $Pr[x = 1] \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, i.e. we also consider skewed distributions. We fix the parameter $r = 1.025$, i.e. the deviation from the expected number of repetitions is bound to 2.5%, and let the parameter p range over $\{0, 0.025\%, 0.05\%, 0.075\%, 0.1\%\}$. We perform 20 test runs and report the average number e of distinct plaintexts. Note that without prefiltering in our previous experiments the average number of distinct plaintexts is n. Hence, we achieve an additional compression of n/e. The extent of the 90% confidence interval is always below 9.5% of the measured number. We show the number of distinct ciphertexts in Figure 6.

We see that the number of distinct ciphertexts approximately grows linear with the parameter p. We also see that the plaintext distribution, i.e. the probability of $x = 1$, has minor influence on the number of distinct ciphertexts. Furthermore, compression is very effective. We achieve a compression ratio of over 17000 for $p = 0$ and still over 900 for $p = 0.1\%$.

We also evaluate the impact of the parameter r. We keep $n = 10^6$ and $Pr[x = 1] \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. We fix the parameter $p = 0.05\%$, i.e. the probability of choosing a new ciphertext without equalizing the distribution is 0.05% and hence we expect at least 500 distinct ciphertexts. We also perform 20 test runs and report the average number e of distinct plaintexts. The extent of the 90% confidence interval is always below 16.6% of the measured number. We show the number of distinct ciphertexts in Figure 6.

We see that the number of distinct ciphertexts is slightly above 500 due to the variance in the input distribution. Yet, the number approaches 500 as the parameter r increases, since more variance is tolerated. Again, our compression is very effective. We achieve a compression ratio roughly between 1700 ($r = 1.025$) and 1900 ($r = 1.1$).

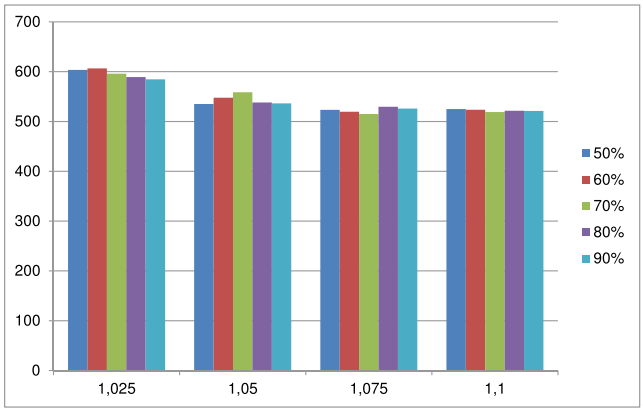

Figure 6: Number of distinct ciphertexts over the parameter r for different plaintext distributions ($p = 0.05$)

We can conclude that for plaintexts which are not amenable to our standard compression techniques or very large databases with resource-constrained clients we can use imperfect frequency-hiding order-preserving encryption. We remark that imperfect frequency-hiding encryption does not achieve our security notion of indistinguishability under frequency-analyzing ordered chosen plaintext attack.

8. CONCLUSIONS

We present a new, strictly more secure order-preserving encryption scheme. We show that the size of the set of indistinguishable sequences and hence security is likely to increase with increasing data sets. We compress the data on the client by a ratio of almost 15, but can achieve much higher ratios (of up to several thousands) with some repeated ciphertexts and leaking some frequency information. This allows the user a choice between security and storage cost.

In summary we provide a new option for encryption data in the cloud. This option can be favorable when higher security than deterministic order-preserving encryption is desired, but client storage cost (and search time) should still remain low.

9. REFERENCES

[1] `http://www.ciphercloud.com/`.

[2] `http://www.vaultive.com/`.

[3] Abadi, D., Madden, S., and Ferreira, M. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM International Conference on Management of Data* (2006), SIGMOD.

[4] Agrawal, D., El Abbadi, A., Emekçi, F., and Metwally, A. Database management as a service: challenges and opportunities. In *Proceedings of the 25th International Conference on Data Engineering* (2009), ICDE.

[5] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. Order preserving encryption for numeric data. In *Proceedings of the ACM International Conference on Management of Data* (2004), SIGMOD.

[6] Agrawal, S., Agrawal, S., Badrinarayanan, S., Kumarasubramanian, A., Prabhakaran, M., and Sahai, A. Function private functional encryption and property preserving encryption: new definitions and positive results. Tech. Rep. 744, IACR Cryptology ePrint Archive, 2013.

[7] Andersson, A., and Carlsson, S. Construction of a tree from its traversals in optimal time and space. *Information Processing Letters 34*, 1 (1990).

[8] Binnig, C., Hildenbrand, S., and Färber, F. Dictionary-based order-preserving string compression

for main memory column stores. In *Proceedings of the ACM International Conference on Management of Data* (2009), SIGMOD.

[9] Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. Order-preserving symmetric encryption. In *Proceedings of the 28th International Conference on Advances in Cryptology* (2009), EUROCRYPT.

[10] Boldyreva, A., Chenette, N., and O'Neill, A. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *Proceedings of the 31st International Conference on Advances in Cryptology* (2011), CRYPTO.

[11] Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., and Zimmerman, J. Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In *Proceedings of the 34th International Conference on Advances in Cryptology* (2015), EUROCRYPT.

[12] Boneh, D., and Waters, B. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th Theory of Cryptography Conference* (2007), TCC.

[13] Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Symposium on Theory of Computing* (2009), STOC.

[14] Goldwasser, S., Kalai, Y. T., Popa, R. A., Vaikuntanathan, V., and Zeldovich, N. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the Symposium on Theory of Computing* (2013), STOC.

[15] Hacigümüs, H., Iyer, B. R., Li, C., and Mehrotra, S. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the ACM International Conference on Management of Data* (2002), SIGMOD.

[16] Hacigümüs, H., Mehrotra, S., and Iyer, B. R. Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering* (2002), ICDE.

[17] Hildenbrand, S., Kossmann, D., Sanamrad, T., Binnig, C., Färber, F., and Wöhler, J. Query processing on encrypted data in the cloud. Tech. Rep. 735, Department of Computer Science, ETH Zurich, 2011.

[18] Islam, M., Kuzu, M., and Kantarcioglu, M. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Proceedings of the 19th Network and Distributed System Security Symposium* (2012), NDSS.

[19] Kadhem, H., Amagasa, T., and Kitagawa, H. Mv-opes: multivalued-order preserving encryption scheme: a novel scheme for encrypting integer value to many different values. *IEICE Transactions on Information and Systems E93.D* (2010), 2520–2533.

[20] Kadhem, H., Amagasa, T., and Kitagawa, H. A secure and efficient order preserving encryption scheme for relational databases. In *Proceedings of the International Conference on Knowledge Management and Information Sharing* (2010), KMIS.

[21] Katz, J., Sahai, A., and Waters, B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology* (2008), EUROCRYPT.

[22] Kerschbaum, F., Härterich, M., Kohler, M., Hang, I., Schaad, A., Schröpfer, A., and Tighzert, W. An encrypted in-memory column-store: the onion selection problem. In *Proceedings of the 9th International Conference on Information Systems Security* (2013), ICISS.

[23] Kerschbaum, F., and Schröpfer, A. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 21st ACM Conference on Computer and Communications Security* (2014), CCS.

[24] Lee, S., Park, T.-J., Lee, D., Nam, T., and Kim, S. Chaotic order preserving encryption for efficient and secure queries on databases. *IEICE Transactions on Information and Systems E92.D* (2009), 2207–2217.

[25] Liu, D., and Wang, S. Programmable order-preserving secure index for encrypted database query. In *Proceedings of the 5th International Conference on Cloud Computing* (2012), CLOUD.

[26] Liu, D., and Wang, S. Nonlinear order preserving index for encrypted database query in service cloud environments. *Concurrency and Computation: Practice and Experience 25*, 13 (2013), 1967–1984.

[27] Lu, Y. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *Proceedings of the 19th Network and Distributed System Security Symposium* (2012), NDSS.

[28] Özsoyoglu, G., Singer, D. A., and Chung, S. S. Anti-tamper databases: querying encrypted databases. In *Proceedings of the 17th Conference on Data and Application Security* (2003), DBSEC.

[29] Pandey, O., and Rouselakis, Y. Property preserving symmetric encryption. In *Proceedings of the 31th International Conference on Advances in Cryptology* (2012), EUROCRYPT.

[30] Popa, R. A., Li, F. H., and Zeldovich, N. An ideal-security protocol for order-preserving encoding. In *34th IEEE Symposium on Security and Privacy* (2013), S&P.

[31] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (2011), SOSP.

[32] Shi, E., Bethencourt, J., Chan, H. T.-H., Song, D. X., and Perrig, A. Multi-dimensional range query over encrypted data. In *Proceedings of the 2007 Symposium on Security and Privacy* (2007), S&P.

[33] Song, D. X., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In *Proceedings of the 21st IEEE Symposium on Security and Privacy* (2000), S&P.

[34] Teranishi, I., Yung, M., and Malkin, T. Order-preserving encryption secure beyond one-wayness. In *Proceedings of the 20th International Conference on Advances in Cryptology*.

[35] Wozniak, S., Rossberg, M., Grau, S., Alshawish, A., and Schaefer, G. Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. In *Proceedings of the ACM Workshop on Cloud Computing Security Workshop* (2013), CCSW.

[36] Xiao, L., Bastani, O., and Yen, I.-L. Security analysis for order preserving encryption schemes. Tech. Rep. UTDCS-01-12, Department of Computer Science, University of Texas Dallas, 2012.

[37] Xiao, L., and Yen, I.-L. A note for the ideal order-preserving encryption object and generalized order-preserving encryption. Tech. Rep. 350, IACR Cryptology ePrint Archive, 2012.

[38] Xiao, L., Yen, I.-L., and Huynh, D. T. Extending order preserving encryption for multi-user systems. Tech. Rep. 192, IACR Cryptology ePrint Archive, 2012.

[39] Zukowski, M., Heman, S., Nes, N., and Boncz, P. Super-scalar ram-cpu cache compression. In *Proceedings of the 22nd International Conference on Data Engineering* (2006), ICDE.