Optimal Average-Complexity Ideal-Security Order-Preserving Encryption

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

Axel Schröpfer
SAP
Karlsruhe, Germany
axel.schroepfer@sap.com

ABSTRACT

Order-preserving encryption enables performing many classes of queries – including range queries – on encrypted databases. Popa et al. recently presented an ideal-secure order-preserving encryption (or encoding) scheme, but their cost of insertions (encryption) is very high. In this paper we present an also ideal-secure, but significantly more efficient order-preserving encryption scheme. Our scheme is inspired by Reed's referenced work on the average height of random binary search trees. We show that our scheme improves the average communication complexity from $O(n \log n)$ to $O(n)$ under uniform distribution. Our scheme also integrates efficiently with adjustable encryption as used in CryptDB. In our experiments for database inserts we achieve a performance increase of up to 81% in LANs and 95% in WANs.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed databases*

Keywords

Order-Preserving Encryption; Indistinguishability; Ideal Security; Efficiency; Adjustable Encryption; In-Memory Column Database

1. INTRODUCTION

Order-preserving encryption [6, 8, 9, 32] enables performing many classes of queries – including range queries – on encrypted data without modification of the database engine. Recent results in database implementation suggest that these queries are quite practical in terms of performance [17, 33]. Hence, researchers have suggested a number of order-preserving encryption schemes. Nevertheless, the security of these schemes is still under discussion.

Boldyreva et al. provide the first formal treatment of security in order-preserving encryption [8]. Intuitively an ideal-security order-preserving encryption (indistinguishability under ordered chosen plaintext attack – IND-OCPA) scheme leaks nothing but the order: The ideal encryption of plaintexts $\{7, 12, 19\}$ is $\{1, 2, 3\}$, i.e. exactly their order.

This type of encryption is quite hard to achieve. Simply imagine encrypting the value of 13 in the ideal-secure ciphertexts above. Clearly, the ciphertexts of 13 and 19 conflict, because both are supposed to be 3. Boldyreva et al. prove that it is impossible to design such an encryption scheme with linear-length ciphertexts, if the encryption scheme is static and stateless. They therefore settle for a weaker security notion (random order-preserving function). It has later been shown that this security definition leaks at least half of the bits [9, 37].

Popa et al. modify the construction of the encryption scheme (calling it now an encoding scheme) [32]. They first prove that it is still not possible to construct a linear-length encryption scheme, even if the encryption function can be stateful. They then settle for an interactive protocol which updates the encryption on inserts. This achieves the goal of ideal-security.

The main idea of Popa et al. is to update the ciphertexts when inserting new values. These updates are also clearly necessary, yet the cost of updates in Popa et al.'s scheme is quite high. Let there be n elements in the database. Even in the best case their scheme incurs a communication cost of $\Omega(n \log n)$.

As Popa et al. point out any immutable[1] encryption scheme must have exponential ciphertext size, yet this is not a problem in most cases. On average, such updates can be kept to a minimum. Reed proves that the height of a random binary search is tightly centered around $O(\log n)$ [34]. We use this result to construct an order-preserving encryption scheme that has $\Omega(n)$ lower bound communication cost and even $O(n)$ in the average case under uniform distribution which is also the theoretical lower bound. Table 1 shows the comparison between ours and the other formally analyzed order-preserving encryption schemes by Popa et al. and Boldyreva et al.

We apply our order-preserving encryption scheme to encryption of an outsourced database. The client retains the key and queries are performed on encrypted data as in [6, 15, 16, 33]. Our subject of investigation is a column-store, in-memory database [12, 36, 43]. Column-store databases store data in columns for faster sequential access, e.g. for aggregations. In order to fit all data in main memory they compress it using a dictionary [44, 4, 7]. We first argue that

[1]In a *mutable* encryption scheme ciphertexts can change.

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

ACM 978-1-4503-2957-6/14/11 ...$15.00.
http://dx.doi.org/10.1145/2660267.2660277.

	Ideal-Secure	Compat. with AE[2]	Best Case	Avg. Case	Worst Case
Boldyreva et al. [8, 9]	no	yes	$O(n)$	$O(n)$	$O(n)$
Popa et al. [32]	yes	no	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
This paper	yes	yes	$O(n)$	$O(n)$	$O(n^2/\log n)$

Table 1: Comparison between OPE schemes

our order-preserving encryption scheme is better suited for this kind of database architecture (Section 4). We also argue that our scheme is efficiently compatible with adjustable encryption as introduced by CryptDB [33] which further increases the security of an outsourced database. Then, we show that our scheme results in improved performance for a number of synthetic and real-world benchmarks (Section 5). In our experiments for database inserts we achieve a performance increase of up to 81% (factor 5) in LANs and 95% (factor 18) in WANs.

In summary, our contribution is a novel order-preserving encryption scheme with updates that

- is ideally *secure* under the IND-OCPA definition by Boldyreva et al. Yet, we point out a new weakness in this definition (Section 3.4.1) that fortunately has no relevance in the database setting.
- is more *efficient* in the average case and in our experimental benchmarks.

The remainder of the paper is structured as follows: In the next Section we review related work – other order-preserving encryption schemes, their applications and related cryptographic schemes. In Section 3 we present our order-preserving encryption scheme, including its algorithms, security proof and complexity analysis. Then, in Section 4 we show how we apply it to an outsourced, encrypted database. We summarize our findings from the experiments in Section 5. This includes a validation of our assumptions and database benchmarks. Finally, we present our conclusions in Section 6.

2. RELATED WORK

2.1 Order-Preserving Encryption

A number of order-preserving encryption schemes have been proposed in the literature [6, 8, 9, 32, 5, 17, 19, 20, 23, 25, 26, 28, 38, 39, 42]. Yet, most of them use ad-hoc or speculative security models. The work by Popa et al. [32] is the first to achieve ideal-security. We improve on their result by lowering cost in the best and average case and enabling compatibility with adjustable encryption.

The notion of ideal-security (IND-OCPA) of order-preserving encryption has been put forward by Boldyreva et al. [8]. On the one hand they settle for a weaker notion of random order-preserving functions in their construction. Yum et al. [42] further improved their construction, but remained in the same security model of random order-preserving functions. On the other hand they achieve immutable ciphertexts as in regular encryption. As Popa et al. already point out in [32] immutable ciphertexts are not a necessity for encrypted databases. We improve on Popa et al.'s result by reducing the probability of updates to be negligible in the size of the plaintext. This results in optimal average cost, equal to that of immutable ciphertexts.

[2]Adjustable Encryption

The idea of a random order-preserving function is to uniformly select among all order-preserving functions for a domain. Clearly, the ciphertext image needs to be larger than the domain, such that there are several to choose from. The security of random order-preserving functions has been challenged. Boldyreva et al. have shown that it inherently leaks at least half of the plaintext bits [9, 37]. This significantly eases an inference attack on an encrypted database. Our scheme – similar to Popa et al. – provides ideal-security. Nevertheless, we also use a large ciphertext image, but employ an encoding technique similar to Popa et al. [32]. Still, Popa et al. usually encode in a smaller ciphertext image at higher update costs than ours.

There is a large number of other order-preserving encryption schemes [5, 17, 19, 20, 23, 25, 26, 28, 39] which provide no formal, but rather ad-hoc security analysis, including the original proposal by Agrawal et al. [6]. Xiao et al. [38] define a notion based on nearby values, but it remains unclear how to enforce this in an encrypted database or similar setting.

2.2 Applications

Order-preserving encryption has a number of applications. Most notably database-as-a-service (DAS) [6, 15, 16, 33]. 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 engine. We also work on databases, but specifically on an in-memory, column-store database. Furthermore, we efficiently support the notion of adjustable encryption as put forth in CryptDB [33]. We emphasize that the proposal for ideal-secure order-preserving encryption by Popa et al. [32] is inefficient in combination with adjustable encryption.

Besides databases order-preserving encryption has many applications in general software-as-a-service, e.g., business software and e-mail [1, 2]. We do not specifically address them in this paper, but expect a high degree of compatibility with our scheme.

2.3 Other Cryptographic Schemes

Searchable encryption achieves a stronger notion of security than order-preserving encryption. Searchable encryption for range queries has been presented in [10, 27, 35]. 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. Applying these schemes also requires a change of the database engine. These two drawbacks make most schemes quite impractical in many cases. Lu [27] presents a searchable encryption scheme for ranges with logarithmic time-complexity, but its indexing information makes it (almost) as vulnerable as order-preserving encryption, since the proposed sorted

tree reveals the order of all elements except of those between the leafs of the same bottom node.

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. In searchable encryption the function is the order relation. Recently, functional encryption has been designed for general functions [14]. Specific functions, such as the inner product, have been proposed before [21].

One can also construct schemes that maintain the result of the function as a ciphertext. Fully homomorphic encryption [13] enables this for arbitrary functions. Previously, interactive techniques such as secure computation [40, 41] have been used. Since in many cases searchable encryption is already too inefficient, we did not consider any of its generalizations as an alternative.

3. OUR SCHEME

3.1 Example

Before we describe our order-preserving encryption scheme in detail we would like to introduce a motivating example. Consider a salary table in a database consisting of first name, last name and salary amount. Table 2 is an example.

First Name	Last Name	Salary
John	Smith	$ 10,000
Jack	Smith	$ 15,000
Jack	Walker	$ 12,000
John	Daniels	$ 18,000

Table 2: Example Salary Table

Our goal is to encrypt this database table, but still be able to search, e.g. for all last names starting with N to Z and salaries larger than $13,000$. Hence, we encrypt *each column* using its own key (state) in our order-preserving encryption scheme (and potentially use adjustable encryption on top as [33]). For first names we need to encrypt 2 values (a compression of 2), for last names we need to encrypt 3 values and for salary amount we need to encrypt 4 values. Correspondingly the dictionaries of those columns would only contain those values.

3.2 Algorithm

Let $x_1, \ldots, x_i, \ldots, x_n$ be the sequence of plaintexts inserted, such that $0 \leq x_i < N$. Let $y_1, \ldots, y_i, \ldots, y_n$ be the corresponding ciphertexts, such that $0 \leq y_i < M$. We describe how to choose M in Section 3.5. Note that the ciphertexts y_i may be modified during the process of encryption. Let $x_{j_1}, \ldots, x_{j_m}$ and $y_{j_1}, \ldots, y_{j_m}$ be the ordered sequence of distinct plaintexts and ciphertexts, respectively.

Consider the following example: $N = 16$ and $M = 256$. Let $n = 5$, $x_1 = 13$, $x_2 = 5$, $x_3 = 7$, $x_4 = 5$ and $x_5 = 12$. Note that $x_2 = x_4$ is a duplicate. Then $m = 4$, $y_1 = 128$, $y_2 = 64$, $y_3 = 96$ and $y_5 = 112$ (without necessity for any ciphertext modification). For our ordered sequence we have $j_1 = 2$, $j_2 = 3$, $j_3 = 5$, and $j_4 = 1$, i.e., $x_{j_1} = 5$, $y_{j_1} = 64$ and so on.

In our example above the encryption algorithm proceeds as follows: Assume the entry $John$, $Smith$, $10,000$ is added first. Then we create three local states – one for each column. In the first we add $John$ and assign it a ciphertext of, e.g., 128. We keep the pair $\langle John, 128\rangle$ in the local state and send the value 128 to the database server. The same for $Smith$ and 10.000, each with a ciphertext of 128. Now, we add $Jack$, $Smith$, 15.000. The value $Jack$ is lexicographically less than $John$, so it gets ciphertext 64. The value $Smith$ has already been added and the ciphertext 128 is repeated. The value $15,000$ is larger than $10,000$, so it gets ciphertext 192. We add the pairs to the local state on the client and send the ciphertexts to the database server. Note that the state for last names does not need to be updated. We repeat this for the other two columns.

Algorithm 1 Encryption

Input: x_i

Output: y_i

State: $\langle x_{j_1}, y_{j_1}\rangle, \ldots$
Initialization: $\langle -1, -1\rangle$, $\langle N, M\rangle$

1. Find $\langle x_{j_k}, y_{j_k}\rangle$, $\langle x_{j_{k+1}}, y_{j_{k+1}}\rangle$, such that $x_{j_k} \leq x_i < x_{j_{k+1}}$.
2. If $x_{j_k} = x_i$, then return y_{j_k}.
3. If $y_{j_{k+1}} - y_{j_k} = 1$, then
   3.1. Initialize to $\langle -1, -1\rangle, \langle N, M\rangle$.
   3.2. Update with $Update(x_{j_1}, \ldots)$[‡] (Algorithm 2).
   3.3. Find $\langle x_i, y_i\rangle$[§].
   3.4. Return y_i.
4. Compute
$$y_i = y_{j_k} + \left\lceil \frac{y_{j_{k+1}} - y_{j_k}}{2} \right\rceil$$
5. Insert $\langle x_i, y_i\rangle$[§].
6. Return y_i.

The input to the encryption algorithm (Algorithm 1) is a plaintext x_i. Encryption is stateful and stores an ordered list of plaintext-ciphertext pairs $\langle x_i, y_i\rangle$. This list is initialized to $\langle -1, -1\rangle$, $\langle N, M\rangle$. The output of the encryption, i.e. the ciphertext y_i, is sent to the database server. We give an explanatory analysis of how the algorithm works in Section 3.3.

We emphasize that our encryption algorithm is keyless. Our transformation is information-theoretic relying only on the state of the algorithm. The state of the algorithm plays the role of the key, i.e. it is secret information. Differently from a key it is not pre-generated, but grows with the number of encryption operations. The size of the state of the encryption algorithm is the size of the dictionary of the database (see Section 4). We therefore do not keep a copy of the data, but only of the dictionary. We can hence reap the same size benefits as dictionary compression (roughly over a factor of 20 [30] which is already achieved for the dictionary and the data identifier column).

[‡]Maintain a copy of plaintexts before initializing state.
[§]The pair is at position $k + 1$.

The update algorithm (Algorithm 2) potentially updates all ciphertexts produced so far. It re-encrypts all (distinct) plaintexts in order, i.e. the median element first and so on. Thus, it produces a (temporarily) balanced tree.

The state of the encryption algorithm is updated on the database client. This updated state needs to be sent to the database server and its persistent data needs to be updated – potentially all database rows. This affects not only the column store, but also the entire dictionary.

Algorithm 2 Update

Input: Sorted and distinct $x_{j_1}, \ldots, x_{j_i}$

1. $Encrypt(x_{j_{\lfloor \frac{i}{2} \rfloor + 1}})$.
2. If $i > 1$, then $Update(x_{j_1}, \ldots, x_{j_{\lfloor \frac{i}{2} \rfloor}})$.
3. If $i > 2$, then $Update(x_{j_{\lfloor \frac{i}{2} \rfloor + 2}}, \ldots, x_{j_i})$.

The decryption algorithm (Algorithm 3) is a simple lookup in the state.

Algorithm 3 Decryption

Input: y_i

Output: x_i

State: $\langle x_{j_1}, y_{j_1} \rangle, \ldots$

1. Find $\langle x_i, y_i \rangle$.
2. Return x_i.

3.3 Analysis

Our order-preserving encryption algorithm builds a binary search tree as does Popa et al.'s. Different from theirs, ours is not necessarily balanced and relies on the uniformity assumption about the input distribution. We only balance the tree when necessary, i.e., then we perform an update operation. This enables us to maintain the dictionary on the client and therefore achieve a significant performance gain and compatibility with adjustable onion encryption.

Recall the example from above: $N = 16$ and $M = 256$. Let $n = 5$, $x_1 = 13$, $x_2 = 5$, $x_3 = 7$, $x_4 = 5$ and $x_5 = 12$. Then $m = 4$, $y_1 = 128$, $y_2 = 64$, $y_3 = 96$ and $y_5 = 112$ (without necessity for any ciphertext modification). We can see the binary tree growing from the root in Figure 2 as we encrypt new plaintexts. Since our choice of M is a power of 2, the path from the root to leaf forms the prefix of the binary representation of the ciphertext. An edge to a higher-order plaintext results in a 1; an edge to a lower-order plaintext results in a 0. The postfix of the ciphertext is $10^\star$.

Our order-preserving encryption algorithm also forms a monotonically increasing curve as does Boldyreva et al.'s (or any order-preserving scheme). In difference to theirs, ours is only monotonically increasing whereas theirs is *strictly* monotonically increasing, since they consider every possible input value. Also, our scheme encrypts the plaintexts on the x-axis in random order, namely as they are encrypted, whereas Boldyreva choose the mean remaining plaintext value

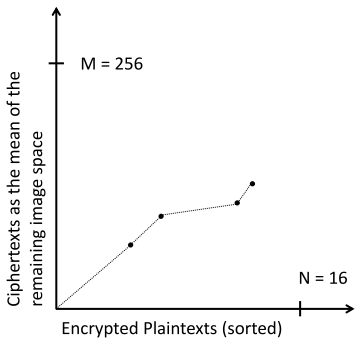

Figure 1: Monotonically Increasing Curve Plot for Encryption Scheme

to encrypt. We choose the mean for the ciphertext, whereas Boldyreva et al. choose the ciphertext uniform randomly in the remaining image space. Figure 1 shows the curve from our running example. On the one hand, since we are not strictly monotonically increasing, we must modify the ciphertexts (rebalance the tree) when we run out of ciphertext space. On the other hand, this enables to achieve IND-OCPA ideal security. Since we reduce the number of updates to be negligible in the average case, we can still achieve superior performance compared to maintaining the dictionary on the server as Popa et al. do.

3.4 Ideal Security

We first give a proof that our encryption scheme is IND-OCPA secure despite its update algorithm. This implies that the update operation does not impact security.

We define the following security game between an encryptor E and an adversary Adv based on the IND-OCPA definition from [9] allowing for update operations.

1. The encryptor E chooses a random bit b.
2. The encryptor E and the adversary Adv engage in a polynomial number of rounds in which the adversary may be adaptive. At round i:
   (a) The adversary Adv sends values x_i^0, x_i^1 ($0 \leq x_i^{\{0,1\}} < N$ to the encryptor E.
   (b) The encryptor E returns $Encrypt(x_i^b)$. If the encryptor E invokes $Update(x_{j_1}, \ldots, x_{j_i})$, then the adversary may observe the updated list $y_{j_1}, \ldots, y_{j_i}$.
3. The adversary Adv outputs b', its guess for b.

We say that the adversary Adv wins the game if its guess is correct ($b = b'$) and the sequences $x_1^0, \ldots$ and $x_1^1, \ldots$ have the same order relation, i.e., for all $i, j : x_i^0 < x_j^0 \Leftrightarrow x_i^1 < x_j^1$. Let win_{Adv} be the random variable indicating the success of the adversary in the above game.

Definition 1. *An OPE scheme is (perfectly) IND-OCPA secure if for all p.p.t. adversaries Adv* $Pr[win_{Adv}] = 1/2$.

Theorem 2. *Our OPE scheme is (perfectly) IND-OCPA secure.*

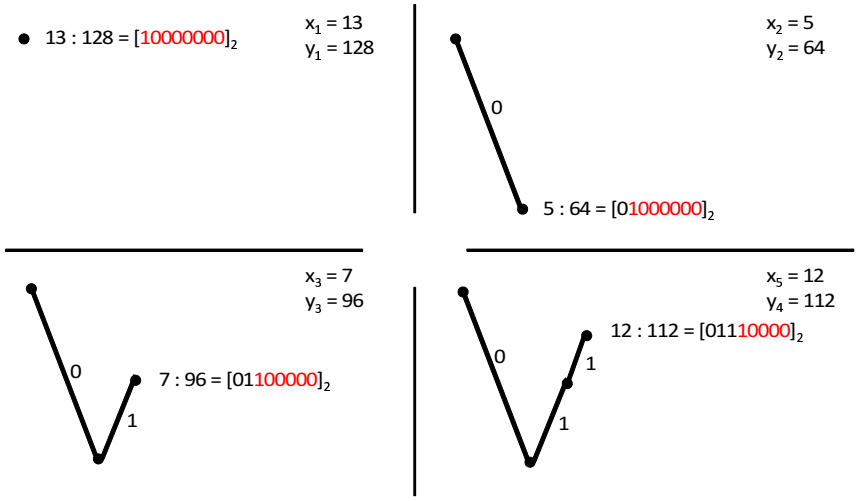

Figure 2: Search Trees for Insertion of 13, 5, 7, 12

PROOF. Observe that our encryption algorithm (Algorithm 1) will start with the same initial state independent of b. We now state the following lemma.

LEMMA 3. *If the sequences of $x_1^b, \ldots$ have the same order relation, the state of the encryption will contain the same $y_{j_1}, \ldots$ independent of b.*

PROOF. We prove this by induction. Assume it holds for round i. Then since the sequences have the same order relation, the algorithm will find pairs with the same $y_{j_k}, y_{j_{k+1}}$ in step 1 due to the induction assumption.

Step 2 is a check within one of the two sequences $x_1^b, \ldots$ and since they have the same order relation, the condition will evaluate the same in both cases. Then, due to the induction assumption the return value y_{j_k} will be the same independent of b.

Updates are triggered in step 3 of Algorithm 1. Clearly, the choice is only made by the values of $y_{j_k}, y_{j_{k+1}}$. Due to the induction assumption the choice is therefore independent of b.

Then the computation also leads to same y_i in step 4. The state is therefore updated with an x_i of the same order relation and the same y_i. Hence, the induction holds for $i+1$.

Clearly, Lemma 3 holds for $i = 0$, since we start with the same initial state. □

In summary, our encryption algorithm outputs the same values y_i and performs the same update operations in both cases of b. Therefore, any adversary Adv can at best guess the value b'.

3.4.1 Insertion Order

As shown our OPE scheme is ideal-secure, but it does leak additional information to the order. Namely, when observing the encryption at the database, i.e. the values y_i, one can determine a partial order of insertion. Our encryption scheme forms a binary search tree. The lowest bit set in the ciphertext marks the height of its position within tree (if M is a power of 2). The lower the height, the later the element has been inserted. Of course, the adversary cannot determine the insertion order between elements of the same height. Therefore it remains a partial order.

First, we do not consider this leakage to be problematic in the use with encrypted databases. Determining the time of compromise in an encrypted database is excruciatingly difficult and therefore the worst-case that the database is always compromised is assumed. Under this worst-case assumption, the adversary obtains the insertion order anyway – even in case of the *same-time* indistinguishability definition of Popa et al. [32].

Second, the IND-OCPA definition cannot account for the insertion order, since – as in any other chosen plaintext attack – the adversary controls this insertion order. It is therefore known to the adversary. Hence, it is not surprising that our scheme still can fulfill this strict security definition.

3.4.2 Domain Coverage

The security of order-preserving encryption relies on the assumption that the plaintext values only sparsely populate their domain. If all values in a domain are encrypted, order-preserving encryption is completely insecure – even if ideal-secure. The ideal-secure order-preserving encryption of the values from 1 to n is 1 to n, i.e. plaintexts and ciphertexts are identical. While not yet quantified, it is always important to keep this observation in mind when using order-preserving encryption.

Clearly, this assumption is violated when encrypting auto-increment counters. The order-preserving encryption of an auto-increment counter – often used as identifiers and foreign keys in databases – is the counter itself. It therefore should not be order-preserving encrypted at all.

This also alleviates the problem that auto-increment counters incur the maximum encryption cost in our scheme. They result in the maximum number of update operations possible, since they follow the worst-case schedule of encryptions. Yet, since they are not to be encrypted at all (for security reasons), they do not represent a problem (for performance reasons).

3.5 Theoretical Performance Analysis

We need to consider the best case, the average case and the worst case complexity of our algorithm. For the average case we assume a uniform distribution of the input.

First, we define a cost model for our algorithms. Local operations on the client can be implemented efficiently – even for large plaintext sets –, since there are no complex (cryptographic) computations, such as modular exponentiations or bilinear maps. Instead all computations are simple arithmetic and simple data structure lookups, but update operations on the database are costly. We therefore mainly consider the cost of inserting one element into the database. Since communication is the main cost, we count the byte size of interaction between the database server and the client as the cost of one insertion. Also the number of rounds is important and our scheme always requires 1 round per insert whereas Popa et al.'s scheme requires $O(\log n)$, but since the size of communication in our case is significantly larger in case of an update we use this as the cost for fair comparison. In our experiments we also vary the delay of the network in order to investigate the importance of rounds.

Second, we determine the complexity of the basic algorithms. If encryption proceeds without update, then we only need to send the new ciphertext to the database: cost $O(1)$, i.e. Algorithm 1 has cost $O(1)$, if steps 3.1 to 3.4 are not executed. A single update operation has cost $O(n)$, since we need to update all elements so far, i.e. Algorithm 2 has cost $O(n)$. We now need to determine the probability of an update in the best, average and worst case.

THEOREM 4. *In the best case our algorithms incur cost $O(n)$ in communication with the database server. This is also the theoretical lower bound.*

PROOF. The best case is when all elements of a perfectly balanced binary search tree are inserted in pre-order traversal order. In this best case we never need to perform an update, since the result is also a perfectly balanced binary search tree. Hence, for n elements we have cost $nO(1) = O(n)$. This also the lower bound, because we need to send each of the n elements at least once. □

The worst case is also easy to analyze.

THEOREM 5. *In the worst case our algorithms incur cost $O(n^2/\log n)$ in communication with the database server.*

PROOF. As already pointed out in Section 3.4.2, the worst case adversarial schedule of ordered plaintext inserts results in an update operation roughly all $O(\log M)$ elements. As we will later show, we choose $M = O(n)$ and such that $M > 2N$, i.e. there is always at least $\log N$ ciphertext space to be filled before an update operation. Therefore the worst case cost is $n/O(\log n) \cdot O(n) = O(n^2/\log n)$. □

Next, we analyze the average case performance under uniform input distribution.

THEOREM 6. *If the ciphertext domain $M > 2^\lambda N$, then in the average case under uniform input distribution our algorithms incur cost $O(n)$ in communication with the database server.*

PROOF. For analyzing the average case complexity we resort to the result of Reed [34]. As already noted, we observe that our ciphertexts form a binary search tree. The first plaintext element inserted is the root (the center ciphertext). Subsequent plaintexts are placed to the left or right depending on their order relation. Reed investigated the distribution of heights of binary search trees. We restate his main theorem (Theorem 1 in [34]).

THEOREM 7. *Let H_n be the height of a random binary search tree of n nodes. Then, $E[H_n] = 4.31107 \cdots \ln n - 1.95302 \cdots \ln \ln n + O(1)$ and $Var[H_n] = O(1)$.*

Note that the maximum length of a ciphertext directly corresponds to the height of the tree. This implies for our encryption scheme that – on average – a ciphertext space $O(\log n)$ will be sufficient. Furthermore, since the variance is constant, it will be sufficient with high probability.

We therefore propose to use a value of $M = O(n)$. Furthermore, we need to reduce the update probability $Pr[Upd]$. The average complexity for all insertions is $n(1+Pr[Upd]O(n))$. Only for $Pr[Upd] \leq O(1/n)$ we achieve $O(n)$ overall average complexity. We can use Lemma 7 of [34].

LEMMA 8. *Let $X_{n,h}$ be the (random) set of nodes at depth h. Then, there is a (universal) constant $C_2 > 2$ such that, for $i > 0$, we have $Pr[X_{n,E[H_n]+i} \neq \emptyset] < C_2 2^{-i/2}$.*

This means that the probability of encountering a ciphertext with length longer than the expected value decreases exponentially with the length of the ciphertext. Hence, if we add a buffer of at least $2 \log n$ bits to the ciphertext length, then the probability of exceeding that buffer is at most $O(1/n)$. This accomplishes the probability of an update $Pr[Upd] \leq O(1/n)$.

In summary, for a plaintext space of $N = 2^l$ we recommend a ciphertext space of λl bits, i.e. $M = 2^{\lambda l}$. The expected average case complexity of inserting n elements is then $O(n)$. Clearly, $\lambda \geq 4.31107 + 2$ is safe, but we evaluate the choice of λ in our experiments. □

4. ENCRYPTED DATABASES

We investigate our encryption algorithms as part of an encrypted, in-memory, column-store database. This has a couple of design implications we highlight in this section.

Column-store databases, such as [12, 36, 43] show excellent performance for analytical workloads. For this they store the data column-wise instead of row-wise. All data for a single column can such be accessed and processed very quickly. The speed of processing can be enhanced further if the data is stored in main memory.

A common compression technique is order-preserving dictionary compression [4, 7]. In dictionary compression data values are replaced by data identifiers and in a dictionary their relation is stored. A dictionary is order-preserving, if the order relation of the data identifiers is the same as the order relation of the data values. Dictionary compression usually achieves compression rates of a factor around 20 [30].

Order-preserving dictionaries have the advantage that select operations – even for range queries – can be performed without accessing the dictionary. The database operator is fed with the data identifier (or data identifiers for range queries) to select and can then process the column. Any select operation that needs to lookup the dictionary can be very costly.

Also update or insert operations can be very costly. They often need to recompute the entire column of data. This may also involve some further compression operations [4, 44].

The crucial insight is that the order-preserving dictionary is an ideal-secure order-preserving encryption. The database performs this operation automatically for us, although not as an encryption operation. It therefore becomes a crucial design decision for an encrypted database how to integrate with this dictionary.

One approach is to strip the dictionary of the data values and keep those at the client. This has been proposed by Hildenbrand et al. in [17]. On the one hand this achieves ideal-security for the order-preserving encryption, since the database only learns the data identifiers. On the other hand this prevents all operations that require access to the data values, such as aggregation which is a very common operation in analytical work loads.

Another approach is to encrypt the data values in the dictionary. This has been proposed by Popa et al. in [32]. It also achieves ideal-security on the database, but requires $O(n \log n)$ cost for inserting n elements, since each element needs to be sorted into the dictionary. When using homomorphic encryption [29] this can also achieve aggregation.

A disadvantage of both approaches is that the database always needs to be encrypted in only order-preserving encryption. Order-preserving encryption may leak more information than is necessary for the queries performed. In [33] Popa et al. introduce the concept of adjustable encryption. Encryption is layered from order-preserving on the innermost layer over deterministic encryption to randomized encryption on the outermost layer. Depending on the operation performed one or more layers of encryption are removed before executing the operator. This results in significantly better security, since only a subset of columns needs to be encrypted order-preserving.

Another main objective of our work is to also efficiently integrate with adjustable encryption. This means that the order-preserving encryption should be the inner-most layer of an onion of encryption. This implies that encryption needs to be performed (mostly) at the client, since other layers of encryption need to be applied. We describe our scheme in the next section.

4.1 Database Integration

We perform encryption at the SQL layer and do not interfere with the dictionary of the in-memory, column-store database. Instead, we keep a local copy of the dictionary as the state of the order-preserving encryption function similar to [17] and perform updates using the SQL update command. Before inserting (or updating) a database row, we encrypt each value. We encrypt the plaintext value using our encryption algorithm (Algorithm 1). Then, we encrypt the ciphertext further using a proxy-reencrytpable deterministic encryption scheme [31]. Finally, we encrypt this ciphertext using a standard randomized encryption algorithm, e.g. AES in counter mode. Figure **??** shows the layers of our adjustable encryption.

We sent the final ciphertext as the data value in the insert or update commands to the database. We keep a local copy of the dictionary as the state of the order-preserving encryption function with the corresponding plaintext values. Furthermore, we sent a separate copy in homomorphic encryption [29]. When we perform a select operation that requires either deterministic or order-preserving encryption we sent the corresponding key to the database which decrypts using a stored procedure. Note that decryption is permanent and never restored.

Since our order-preserving encryption scheme is mutable, it may be necessary to update all ciphertexts. In case our encryption algorithm triggers this update (Algorithm 2), we re-encrypt the entire local dictionary. We again perform encryption to the top-most layer currently stored in the database. Then, we issue update commands replacing all current dictionary values with their new ciphertexts. Of course, this operation is very costly and its occurrence must be kept to a minimum. We show theoretically in Section 3.5 and experimentally in Section 5 that we achieve this.

This design allows us to operate the encrypted database on the SQL layer, i.e. we do not interfere with the dictionary on the server. Instead, we modify the data values on the client and use standard SQL commands. Furthermore, we can easily add layers of encryption on the client and the ciphertexts in database are often encrypted at higher layers than order-preserving encryption.

In order to see the problems with adjustable encryption and Popa et al.'s scheme of [32], recall the encryption procedure. As long as deterministic or randomized encryption is the top most layer, the ciphertexts may not be sorted. This means that the protocol of [32] may not be applied. Once it is adjusted to order-preserving encryption, then all ciphertexts need to be sorted. This requires a complete update of the database, similar to our Algorithm 2. Hence, not only is insertion in order-preserving encryption in their scheme more expensive in the average case than in ours, also in adjustably encrypted database they require one update in many cases whereas the probability of update is negligible in our scheme.

5. EXPERIMENTS

We perform three sets of experiments: the number of updates depending on the parameter λ, correlation between plaintext and ciphertext for the different order-preserving encryption schemes and performance of database inserts between our and Popa et al.'s scheme.

For these experiments we use two data sets for plaintext inputs. We use a uniform contribution of inputs in accordance with our theoretical assumption. We sample draw n independently, identically distributed uniform random variables from $\mathbb{Z}_N$ using a pseudo-random number generator available from the operating system. Then, we also use a sample drawn from the real-world. It results from a large scale communication transcript, the Enron mail set [22], that has been used in the evaluation of various, related research efforts [11, 18, 24].

From the Enron data set we derive inputs in the following way. We downloaded and extracted the 2009 version of the data set [3]. The data set consists of folders and files containing plaintext mail correspondence of Enron employees. We retrieve samples from the data set by reading words from the mail bodies of the correspondence. Each word is treated as a number. In order to read an l bit number, we read $\lceil l/8 \rceil$ character bytes and interpret those as an unsigned integer (mod l). We continuously read the mail stream, i.e., after processing n numbers, we read the next, subsequent n numbers for the next experiment.

Throughout all experiments we use as hardware HP Z820 workstations with 8 quad core CPUs and 128 GB RAM, operating SUSE Linux Enterprise Server 11 SP2. We run experiments using 64 bit versions of Java JDK 1.7.0_45 and GCC 4.3 (used for cryptographic routines implemented in C++, executed with JNI).

5.1 Parameter λ

The expansion factor λ determines the ciphertext expansion, but also the expected number of updates. It should be chosen as small as possible, but still large enough in order to prevent frequent updates. We derive a safe theoretical bound of $\lambda \geq 6.31107$ in Section 3.5. However, this theoretical bound rests on the assumption of uniform input distribution. We therefore experimentally test our order-preserving encryption scheme also on real-world inputs from the Enron data set. We aim to test how real-world distributions may affect our scheme. We already know that there are worse distributions, such as ordered insertions, but most real-world data is not ordered. We hypothesize that in most real-world cases an expansion factor of $\lambda \geq 6.31107$ is safe.

5.1.1 Setup

In our experiments we measure the average number of updates that occur. We vary three parameters: the expansion factor λ, the number l of bits in the plaintext domain, and the number n of plaintexts. For the expansion factor we choose $\lambda = 2, 3, \ldots$. For the number of plaintext bits we choose $l = 4, 8, 16$, i.e. $N = 2^l$. Then M is determined by λ, i.e. $M = 2^{\lambda l}$. These bit sizes may seem small and, of course, our algorithm and implementation can encrypt arbitrarily large inputs, but in order to trigger an update one needs to encrypt close to the entire input domain and this can only be achieved in a realistic time frame for such small bit sizes. We choose the number of plaintexts depending on the size of the plaintext domain, i.e. we choose a fraction α. We choose $\alpha = 0.25, 0.5, 0.75, 1$, i.e. $n = \alpha 2^l$. We run 25 experiments for any combination of parameters.

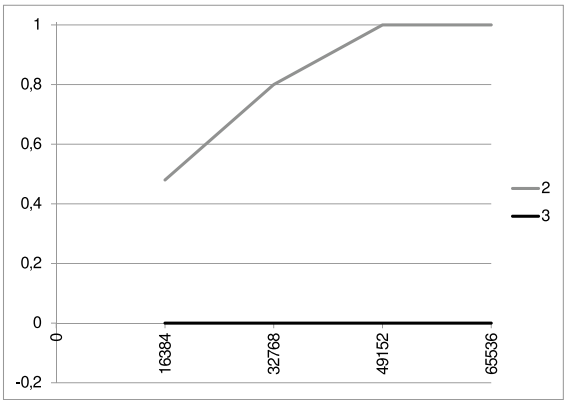

Figure 3: Average Number of Updates per Number of Inputs ($l = 16$, $\lambda = 2, 3$)

Figure 3 depicts the average number of updates for the uniform data set, $\lambda = 2, 3$ and $l = 16$ per number of elements $\alpha = 0.25, 0.5, 0.75, 1$ on a linear x-axis. Figure 4 depicts the average number of updates for the uniform data set, $\lambda = 2, 3$ and $\alpha = 1$ per size of the plaintext $l = 4, 8, 16$ on a logarithmic x-axis. We observed that updates only occurred

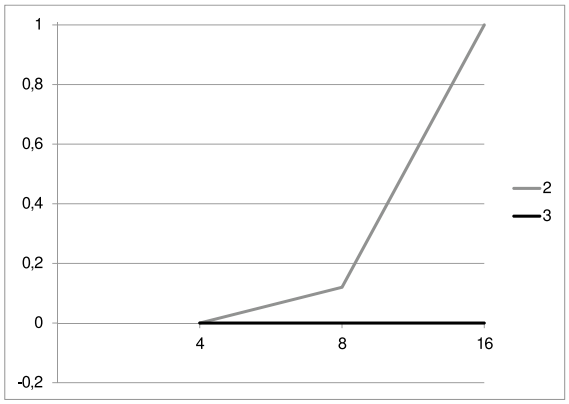

Figure 4: Average Number of Updates per Plaintext Size ($\alpha = 1$, $\lambda = 2, 3$)

for $\lambda = 2$ in the uniformly distributed data set. Already for $\lambda = 3$ no updates occurred. We never encountered more than one update in a single experiment. Furthermore, we do not show any results for the Enron data sets, since for the same set of parameters no update ever occurred.

5.1.2 Discussion

Despite that we feel that more experiments are needed to reliably recommend a choice of λ in all real-world cases, we make the following observations: Our hypothesis has not been falsified. Our encryption scheme behaves better for the real-world Enron data set than the uniform distribution, since no update occurred even for $\lambda = 2$. This shows that there are real-world data sets, particularly text-based as the Enron data set, that are well amenable to our order-preserving encryption scheme.

For an increasing number of elements, the necessary expansion factor λ increases. We can see this in Figure 3 for $\lambda = 2$. This is not surprising, since the size of the tree is growing and hence its height.

For an increasing size of the plaintext domain, the necessary expansion factor λ increases, but is significantly below the theoretical threshold. We can see this in Figure 4 for $\lambda = 2$ on the logarithmic x-axis. We attribute this effect to the asymptotic nature of the theoretic analysis. We know that for the uniform distribution there is a bound, but this bound is only approached slowly. For many practically relevant values we are still far from reaching that bound. Hence, the expansion factor λ may be (carefully) chosen smaller.

For performance reasons it is advisable to choose λ large enough to prevent updates. One may still argue that for security reasons λ should be smaller. Recall from Section 3.4.2 that as the number of encrypted inputs increases, the security of order-preserving encryption decreases. Hence, one may choose λ small enough to encounter a performance hit when there is a security problem. We argue that this is a too ambiguous signal, since the performance hit may come from several sources, such as encryption adjustment. Instead, the client should issue a security warning, which he can simply base on the size of the encryption state.

5.2 Correlation

The first formally verified proposal for order-preserving encryption by Boldyreva et al. [8] settled for a weaker notion

than ideal-security. They showed that ideal-security is not achievable using stateless, immutable encryption functions. Later Popa et al. presented a mutable order-preserving encryption scheme that achieves ideal-security [32]. Our scheme is ideal-secure (and mutable) as well. We hypothesize that Popa et al.'s scheme is at least as secure as Boldyreva et al.'s and that our scheme is at least as secure as Popa et al.'s.

For this experiment we need implementations of not only our scheme, but also Boldyreva et al.'s and Popa et al.'s. For Boldyreva et al.'s we use the implementation of Popa available from CryptDB [33]. Popa et al.'s scheme we reimplemented using the description in [32] as a blue-print and divide the implementation into an encryption client and an encryption server running on the database. We implemented their (mOPE) protocol for deterministic encryption, because it is faster, but only used the order-preserving ciphertexts for cryptanalysis, i.e. there is no security advantage in using randomized encryption (stOPE). As deterministic encryption scheme we use AES-128 in ECB mode.

5.2.1 Setup

We measure the correlation coefficient r between the plaintexts and the ciphertexts. As we point out in Section 3.3 any order-preserving encryption scheme forms a monotonically increasing curve. Hence, there is a stronger correlation between plaintexts and ciphertexts than in – for example – deterministic encryption. Yet, the smaller the coefficient, the less the correlation can be used for cryptanalysis, i.e., a smaller correlation coefficient – closer to 0 – is better. A correlation coefficient of 1 (or -1) implies a deterministic derivation function of plaintexts from ciphertexts.

For Boldyreva et al.'s scheme we use the parameters proposed in [8] choosing a random key of l bits. For Popa et al.'s scheme there are no parameters to set. For our scheme we set $\lambda = 3$, such that there are no updates. We set $l = 16$, i.e. $N = 65536$ and vary the number n of inputs: $n = 1024, 2048, 4096, 8192, 16384, 32768$. Recall that $M = 2^{\lambda l} = 2^{48}$. We run 25 experiments for each value of n and each data set – uniform and Enron.

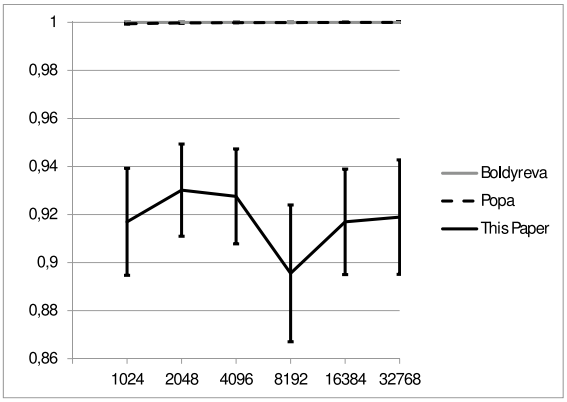

Figure 5: Correlation Coefficient with Uniform Inputs per Number of Inputs

Figure 5 depicts the correlation coefficient for uniformly random inputs per number of inputs on the logarithmic x-axis. Figure 6 depicts the correlation coefficient for inputs from the Enron data set per number of inputs also on a logarithmic x-axis. We show the 90% confidence intervals as error bars. For Boldyreva et al.'s scheme no error bars are visible, since they are too small; similarly, for Popa et al.'s scheme under uniform distribution. In Figure 5 Boldyreva et al.'s and Popa et al.'s curves almost overlap. The precise data reveals a small advantage for Popa et al.'s scheme.

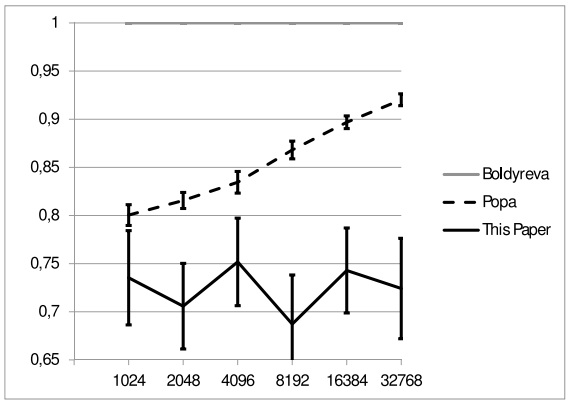

Figure 6: Correlation Coefficient with Enron Inputs per Number of Inputs

5.2.2 Discussion

Although our experiments are certainly not conclusive for judging the security of order-preserving encryption, we make the following observations: Our hypotheses have not been falsified. Under uniform distribution and particularly in the Enron data set our encryption scheme performs better than the not ideal-secure by Boldyreva et al. In the Enron data set our correlation coefficient can be as small as 0.68 whereas Boldyreva et al.'s is only marginally smaller than 1. In the Enron data set also Popa et al.'s scheme performs significantly better than Boldyreva et al.'s. The ideal-secure, order-preserving encryption schemes perform better and hence it is advisable to use them.

In both – uniformly distributed and Enron – data sets our scheme performs better than Popa et al.'s. A validated explanation of this observation remains an open research question. We conjecture that frequent rebalancing of the search tree negatively impacts security. Note that we never rebalance (no updates, $\lambda = 3$) whereas Popa et al. use a balanced tree in order to limit the worst-case complexity. We emphasize that both schemes are ideal-secure. Still, our results indicate that our scheme has a higher diffusion between plaintexts and ciphertexts than both other schemes.

In Popa et al.'s scheme in the Enron data set, we observe an increase in correlation as more inputs are encrypted. This supports our security caveat from Section 3.4.2 that as we approach the full input domain security decreases. As mentioned in the previous experiment, a security alert on the client seems necessary.

5.3 Database Inserts

We have shown in Section 3.5 that our scheme has lower average communication complexity than Popa et al.'s under uniform distribution. Furthermore, we have shown in Section 5.1 that our scheme also behaves well for real-world data sets. We also argued in Section 4.1 that our scheme efficiently integrates with outsourced, encrypted databases.

So, we hypothesize that our scheme has better performance for insertion in an encrypted database than Popa et al.'s.

5.3.1 Setup

We measure the wall-clock time it takes to encrypt an input value and insert it into the database. We particularly also include the database time, although it is the same for both schemes, but we believe that the entire operation is time critical. The encryption time includes local operations and (potential) updates to the database. In case of Popa et al.'s scheme their (mOPE) protocol is run.

We do not report decryption or database selection times, since they are negligible (less than 1 ms) compared to encryption and insertion and also do not significantly differ between the two schemes.

We use the following machine setup. There is one client and one server machine connected via a network. Both machines are HP Z820 workstations. For Popa et al.'s scheme we execute the encryption client on the client machine and the encryption server as well as the database on the server machine. The encryption client calls the encryption server using TCP-based RMI[3], and the encryption server uses TCP sockets to update the database. We follow the recommendation in [32] to host the encryption server close to the database, i.e., we actually run them on the same machine.

For our scheme, we run the encryption algorithms on the client and the database on the server. The client uses TCP sockets to update the database.

Clearly, the performance of the network connecting client and server has a significant impact on the speed of database inserts. We run our experiments using two different, physical network conditions: LAN and WAN. In the LAN setting, both – client and server – are hosted on the same Ethernet segment. In the WAN setting, client and server are hosted at distant locations (roughly 30 miles or 50 km distance). The communication link has roughly 30 Mbits/s bandwidth and 10 ms latency. As column-store database we use SAP HANA 1 SP5.

Based on the experiments in Section 5.1, we use the expansion factors $\lambda = 2, 3$. We set the number of bits in the plaintext size as $l = 4, 8, 16$. The number of plaintexts we choose depends on the size of the plaintext domain, i.e. we choose a fraction α. We choose $\alpha = 0.25, 0.5, 0.75, 1$, i.e. $n = \alpha 2^l$. We run 25 experiments for each value of n and each data set – uniform and Enron.

Figure 7 shows the performance results of Popa et al.'s scheme and ours for an expansion factor $\lambda = 3$. In the LAN setting, our scheme has a performance improvement of 63% on average and 77% in the best case for the Enron data set. The performance improvement is higher under uniform distribution with 72% on average and 81% in the best case.

There is an even higher performance improvement in the WAN setting ($\lambda = 3$). As we show in Figure 8, we have an average improvement of 83% and 95% in the best case for the Enron data set. For the uniform data set we have 83% improvement on average and 88% in the best case.

Furthermore, in order to evaluate the performance impact of updates in our algorithm, we compare the performance of database inserts with expansion factors of $\lambda = 2$ and $\lambda = 3$. Figures 9 and 10 depict the database insert performance

[3]RMI was significantly faster than a reference implementation based on plain TCP sockets.

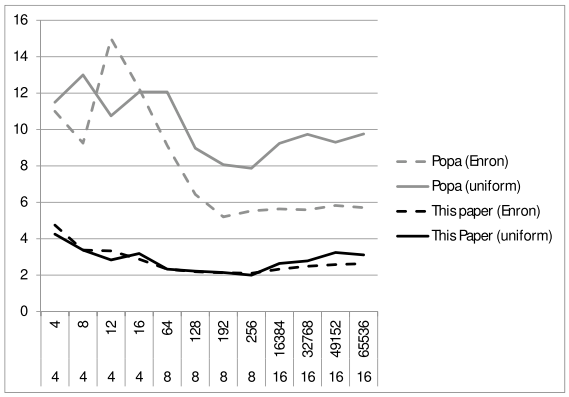

Figure 7: Time per Database Insert in ms on LAN ($\lambda = 3$) for input sizes of l and $\alpha 2^l$ elements

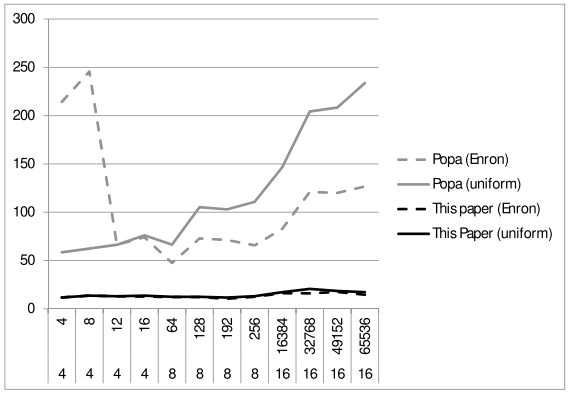

Figure 8: Time per Database Insert in ms on WAN ($\lambda = 3$) for input sizes of l and $\alpha 2^l$ elements

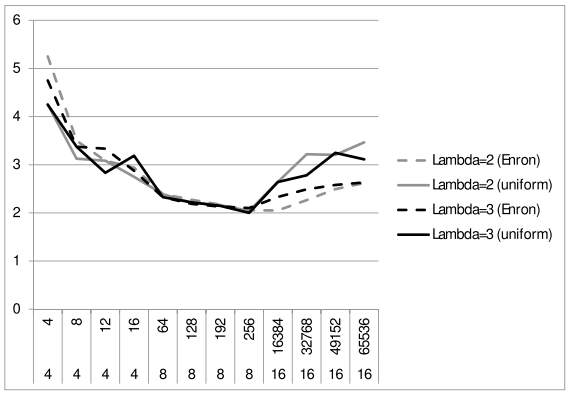

Figure 9: Time per Database Insert in ms on LAN (our scheme) for input sizes of l and $\alpha 2^l$ elements

results of our scheme for $\lambda = 2, 3$ for the Enron and the uniform data set in LAN and WAN setting, respectively.

5.3.2 Discussion

We make the following observations: Our hypothesis has again not been falsified. Our proposed scheme clearly and unambiguously outperforms Popa et al.'s for both network settings and data sets. Our scheme requires almost no in-

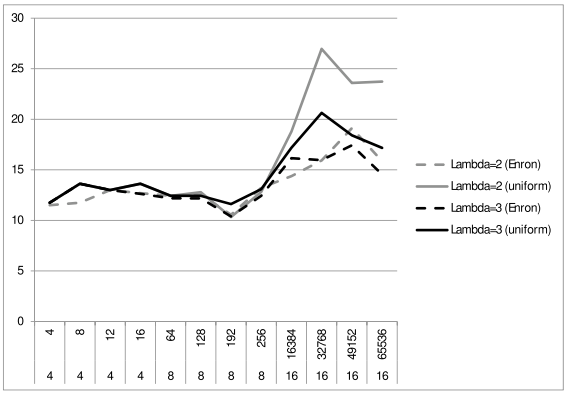

Figure 10: Time per Database Insert in ms on WAN (our scheme) for input sizes of l and $\alpha 2^l$ elements

teraction with the database except for the database insert operation itself. In contrast, the scheme of Popa et al. usually transmits multiple messages for each encryption, in order to traverse the balanced tree of the encryption server. As the messages are exchanged in rounds, the communication suffers from the network latency which we can clearly see in the WAN setting. In addition, Popa et al.'s update the database regularly whereas our scheme performs updates rarely or even never. These effects are clearly visible in our measurements.

We also see that the difference of the measured times between the two data sets is larger for Popa et al.'s scheme. Except for very low values of input size l and number of elements $\alpha 2^l$, the Enron set has lower database insert times. We attribute this to the fact that there are less distinct values and hence less rounds are required to traverse the balanced tree, i.e., the tree depth is lower for the Enron data set compared to a uniformly distributed data set. In this case also the number and size of database updates are smaller as compared to a uniform distribution of inputs.

When comparing different values of $\lambda = 2, 3$ in our scheme, we observe similar results for most of experiments performed for different pairs of input size l and number of elements $\alpha 2^l$. For larger values of input size and number of elements, e.g. $l = 16$ and 32768 elements, we observe a larger distance between the measured times, in particular between the two data sets. We attribute this effect to the updates (cf. Figure 3) of our scheme. Still, the impact of updates on a single database insert is less than 3% (of Popa et al.'s scheme's performance) in our WAN setting and less than 5% in our LAN setting compared to a performance gain of up to 95% and 81%, respectively. This indicates that even when including update operations our encryption scheme is still significantly faster.

6. CONCLUSIONS

We present a novel order-preserving encryption scheme. It has optimal average communication complexity of $O(n)$ and is provably ideal-secure. We have shown in our database benchmark that it significantly – with a performance gain of up to 95% – outperforms previous work. Further experimental results indicate that our scheme works well with real-world data sets and has a higher diffusion between plaintexts and ciphertexts than previous work. Our scheme also efficiently integrates with adjustable encryption. Hence, it is currently the best suited scheme for outsourced, encrypted databases based on order-preserving encryption.

7. REFERENCES

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

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

[3] `https://www.cs.cmu.edu/~enron/enron_mail_20110402.tgz`.

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

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

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

[7] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2009.

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

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

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

[11] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Proceedings of the 33rd International Conference on Advances in Cryptology*, CRYPTO, 2013.

[12] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA database – an architecture overview. *IEEE Data Engineering Bulletin*, 35(1):28–33, 2012.

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

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

[15] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the*

ACM International Conference on Management of Data, SIGMOD, 2002.

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

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

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

[19] H. Kadhem, T. Amagasa, and H. Kitagawa. 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:2520–2533, 2010.

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

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

[22] B. Klimt and Y. Yang. The enron corpus: a new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning*, ECML, 2004.

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

[24] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan. Search pattern leakage in searchable encryption: attacks and new constructions. Technical Report 163, IACR Cryptology ePrint Archive, 2013.

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

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

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

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

[29] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 18th International Conference on Advances in Cryptology*, EUROCRYPT, 1999.

[30] H. Plattner. A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2009.

[31] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

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

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

[34] B. Reed. The height of a random binary search tree. *Journal of the ACM*, 50(3):306–332, 2003.

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

[36] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented dbms. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB, 2005.

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

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

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

[40] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23th Symposium on Foundations of Computer Science*, FOCS, 1982.

[41] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *Proceedings of the 27th Symposium on Foundations of Computer Science*, FOCS, 1986.

[42] D. H. Yum, D. S. Kim, J. S. Kim, P. J. Lee, and S. J. Hong. Order-preserving encryption for non-uniformly distributed plaintexts. In *Proceedings of the 12th International Workshop on Information Security Applications*, WISA, 2011.

[43] M. Zukowski, P. A. Boncz, N. Nes, and S. Héman. Monetdb/x100 - a dbms in the cpu cache. *IEEE Data Engineering Bulletin*, 28(2):17–22, 2005.

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