A Comparative Evaluation of Order-Preserving and Order-Revealing Schemes and Protocols

Dmytro Bogatov, George Kollios and Leo Reyzin

Abstract

Database query evaluation over encrypted data has received a lot of attention recently. Order Preserving Encryption (OPE) and Order Revealing Encryption (ORE) are two important encryption schemes that have been proposed in this area. These schemes can provide very efficient query execution but at the same time may leak some information to adversaries. In this paper, we present the first comprehensive comparison among a number of important OPE and ORE schemes using a framework that we developed. We evaluate protocols that are based on these schemes as well. We analyze and compare them both theoretically and experimentally and measure their performance over database indexing and query evaluation techniques using not only execution time but also I/O performance and usage of cryptographic primitive operations. Our comparison reveals some interesting insights concerning the relative security and performance of these approaches in database settings. Furthermore, we propose a number of improvements for some of these scheme and protocols. Finally, we provide a number of suggestions and recommendations that can be valuable to database researchers and users.

1 Introduction

Database security has received increased attention recently due to the enormous amount of sensitive data that are stored in databases everyday. Furthermore, the increasing popularity and availability of cloud infrastructures makes moving database management to the cloud even more appealing, thus increasing the security concerns. One approach to protect the confidentiality of the data is to use encryption and performing database query evaluation over encrypted records. To that end, the idea of specialized encryption schemes for database operations has been proposed. In particular, Agrawal et al. [1] proposed the idea of Order Preserving Encryption (OPE). Namely, an encryption scheme where ciphertexts have the same order as their plaintexts. This is a very useful primitive since it allows a database system to make comparisons between chiphertexts and get the same results as if it had operated on plaintexts. At the same time, the order of the plaintexts is leaked (and in most cases much more.)

Boldyreva et al. [7] were the first to treat OPE schemes from a cryptographic point of view and provided security models and analysis for these schemes. Moreover, they presented a scheme that satisfies formal security guarantees and is relatively easy to implement. Since then, a number of OPE schemes have been proposed that provide different performance and security guarantees. Furthermore, in order to improve the security of these schemes, Boneh et al. [9] proposed the idea of Order Revealing Encryption (ORE). In this approach, ciphertexts have no particular order and look more like typical semantically secure encryptions. The database system has a special comparison function that can be used to compare two ciphertexts. These schemes are more secure than OPE schemes, although still leak some information, and in general are more expensive to compute. In addition, a number of protocols that can be used in an outsourced database setting have been proposed as well, that are closely related to these schemes.

Currently, it is a very challenging task for users to choose an appropriate OPE or ORE scheme or protocol for their applications. Many of the papers presented these schemes are mostly theoretical and concentrate more on the security definitions and analysis and less on the performance. Actually, some of these schemes have not been even implemented properly. Furthermore, even-though the main target of these schemes are database applications, most of them have not been evaluated in database settings.

To address this problem, in this paper we design a new framework that allows for systematic and extensive comparison of OPE and ORE schemes and protocols for database applications. We employ these schemes in database indexing techniques (i.e. B+ trees) and query protocols and we report various costs including I/O complexity.

Our contributions can be summarized as follows:

- We discuss the security definitions for a number of important schemes and protocols and we contrast their leakage profiles.
- We present the main ideas behind these schemes and protocols and we provide our analysis and implementation challenges for each one. In addition, we present improvements for some of the schemes that make them more efficient and/or more secure.
- We present an experimental evaluation using both real and synthetic datasets using our new framework that tracks not only time but also primitive usage, I/O complexity, and communication cost.
- Finally, we give an overall picture of the different methods and we make recommendations to practitioners.

1.1 Related work

A number of OPE schemes have been proposed recently including [1, 42, 7, 8, 47, 30, 51, 26, 29, 27, 53, 52, 35, 17, 36]. Popa, Li, and Zeldovich [43] present a nice analysis of these schemes and they are the first to show that using a stateful scheme you can achieve the ideal security guarantees for OPE. We pick two of these schemes (BCLO [7] and FH-OPE [29]) that are the most representative and outperform other schemes.

In addition, there are a number of ORE schemes [9, 14, 34, 13, 12, 10, 20, 18] that have been proposed. We choose the most practical and most secure of them [14, 34, 13], to include in the comparison. Also, there are some approaches that assume an outsourced setting where the client may have to communicate with the server during query processing [45, 31, 5, 15]. We choose two of these protocols [45, 31] because they are based on OPE and ORE approaches and therefore have similar security models with these schemes. We would like to point out that there are some other methods that can be used to run range queries on encrypted data that use different types of schemes and techniques. See [5] and [37] for an overview of other methods. We do not consider them in this paper in order to make the comparison fair and manageable. Finally, we would like to stress that the schemes and protocols discussed here should be used with care. The schemes provide specific primitives, security guarantees and leakage profiles, and it is up to the practitioner how to use them. Recent work has shown that systems that leak additional information may compromise the confidentiality and privacy of the whole data, e.g [22, 23, 40, 19, 28, 11, 16, 33, 4, 50].

2 Background

We start by describing the concepts of Order Preserving and Order Revealing Encryption schemes followed by the security definitions of these schemes.

2.1 Order Preserving Encryption

An OPE scheme is a tuple of polynomial-time algorithms KGen, Enc, Dec defined over a well-ordered domain $\mathcal{D}$ with the following properties:

- $\text{KGen}(1^\lambda) \rightarrow \text{sk}$. On input a security parameter λ (formally, in unary, to ensure polynomial running time), the randomized setup algorithm KGen outputs a secret key sk.
- $\text{Enc}(\text{sk}, m) \rightarrow \text{ct}$. On input the secret key sk and a message $m \in \mathcal{D}$, the possibly randomized encryption algorithm Enc outputs a *numerical* ciphertext ct.
- $\text{Dec}(\text{sk}, \text{ct}) \rightarrow m$. On input the secret key sk and a numerical ciphertext ct, the deterministic decryption algorithm Dec outputs the original message m.

Definition of correctness for an OPE scheme consists of two parts. The first part ensures the correctness of encryption and decryption algorithms.

$$\Pr[\text{Dec}(\text{sk}, \text{Enc}(\text{sk}, m)) = m] = 1 - \text{negl}(\lambda)$$

The second part ensures that the order of ciphertexts is preserved.

$$\Pr[\boldsymbol{P}(m_1, m_2) = \boldsymbol{P}(\text{ct}_1, \text{ct}_2)] = 1 - \text{negl}(\lambda)$$

for $\boldsymbol{P}$ being a comparison operator ($<$, $\leq$, $=$, $\geq$, $>$).

It is possible to decrypt the ciphertext by doing binary search on the domain, encrypting values, until the given ciphertext is found. However, such operation would be expensive — $\mathcal{O}(\log |\mathcal{D}|)$, and therefore the scheme includes a decryption component.

2.2 Order Revealing encryption

An Order Revealing Encryption (ORE) scheme is a successor of an OPE scheme with the main difference that the ciphertexts are no longer numerical; however, it provides another public (keyless) algorithm that compares the ciphertexts. An ORE definition (used in [14]) is the following:

- $\text{KGen}(1^\lambda) \to \text{sk}$. On input a security parameter λ, the randomized setup algorithm KGen outputs a secret key sk.
- $\text{Enc}(\text{sk}, m) \to \text{ct}$. On input the secret key sk and a message $m \in \mathcal{D}$, the possibly randomized encryption algorithm Enc outputs a ciphertext ct.
- $\text{Cmp}(\text{ct}_1, \text{ct}_1) \to b$. On input two ciphertexts ct_1, ct_2, the comparison algorithm Cmp outputs a bit $b \in \{0, 1\}$.

The correctness definition is

$$\Pr[\text{Cmp}(\text{ct}_1, \text{ct}_2) = \mathbf{1}(m_1 < m_2)] = 1 - \text{negl}(\lambda)$$

Note that an ORE scheme does not include a decryption algorithm. Additionally to the binary search approach, one can simply append a symmetric encryption of the plaintext to the produced ciphertext and use it for decryption. The tradeoff is between the speed of computation and the ciphertext size. Binary search will cost $\mathcal{O}(\log |\mathcal{D}|)$ encryptions, while appending symmetric encryption will increase (for realistic parameters, double or triple) the ciphertext size.

Finally, note that OPE is a special case of ORE, where the comparison algorithm is trivial.

2.3 Randomized schemes and left / right framework

Some schemes that are analyzed in this paper deviate slightly from the above definitions. In particular, stateful schemes consume and return state that serves as a secret key. Also, randomized schemes have more complex comparison algorithms as it is not possible to simply check numerical ciphertexts for equality in the OPE setting. Finally, some schemes use "left / right framework" defined in [34] which generates two ciphertexts per encryption — left and right. When we compare two encrypted values, we use the left ciphertext of one value with the right ciphertext of the other. In some schemes, one side is semantically secure.

2.4 Protocols

For the purposes of this paper, a secure search protocol is defined as a client-server communication involving three stages — handshake, construction and search. Communication occurs between a client, who owns some sensitive data, and an honest server. The handshake algorithm Setup is optional and is used for generating keys and exchanging some meta-data. The construction algorithm Insert accepts a data point which is a tuple of index and value both of which are sensitive, and involves the client sending to the server encrypted data for storage. The search algorithm Search accepts two search endpoints which are sensitive, and involves the client sending to the server range queries and the server correctly responding to them.

Definition 1 (Generic ORE Protocol). For an ORE scheme $\text{ORE} = (\text{KGen}, \text{Enc}, \text{Cmp})$, symmetric encryption scheme $\text{S} = (\text{KGen}, \text{Enc}, \text{Dec})$ and data structure $\text{DS} = (\text{Insert}, \text{Search})$, the protocol Π is defined as follows.

- Π.Setup:
  1. Generate $K = \text{ORE.KGen}(\cdot)$ and $k = \text{S.KGen}(\cdot)$
  2. Initialize DS
- $\Pi.\text{Insert}(i, v)$:
  1. Client encrypts a data point index $\text{ORE.Enc}(K, \cdot)$
  2. Client encrypts a data point value $\text{S.Enc}(k, \cdot)$
  3. Client sends this tuple to the server
  4. Server stores the encrypted data point $\text{DS.Insert}(\cdot)$
- $\Pi.\text{Search}(l, r)$:
  1. Client encrypts the endpoints $\text{ORE.Enc}(K, \cdot)$
  2. Client sends this tuple to the server

3. Server answers the query DS.$\textsc{Search}(\cdot, \cdot)$
4. Server sends the result (symmetrically encrypted values) back to the client
5. Client decrypts all elements S.$\textsc{Dec}(k, \cdot)$

Note that the protocol requires a data structure that works on encrypted values. Most existing efficient data structures (e.g. B-tree [3]) only need to compare values to each other to function correctly (see Section 3.6).

Also note that the server may interact with the client during all stages (e.g. ask the client to sort a small list of ciphertexts).

In the case of left / right framework ORE schemes, the protocol is slightly different. It is always an option to generate both ciphertexts for each encryption, but that would diminish the value of left / right framework. Instead, the search stage is modified so that the client sends only one side of the ciphertexts, while the server stores in the data structure only the other side of ciphertexts. In this case, the data structure must be uploaded to the server in advance, since insertion still requires both ciphertexts to be sent by the client.

2.5 Security definitions

Each scheme and protocol we analyze has its own security definition. We attempt to unify these definitions and analyze them under a common framework. We also attempt to assess relative security of these definitions and analyze their leakages.

In this work we consider the snapshot model, where the attacker can observe all the database contents at different time instants. All security definitions of the schemes that we discuss here are based on this model. Also, the snapshot attacker is the most common attacker that we face today [5]. The idea is that a hacker or an insider can steal the database and all its contents at any point in time.

2.5.1 IND-OCPA

The first security notions for OPE were proposed by Boldyreva et al. [7]. Their first definition says that the scheme is IND-OCPA secure if an adversary cannot distinguish which of the two sequences of plaintexts was encrypted; the adversary is allowed to supply the two sequences with arbitrary plaintexts as long as the plaintexts are of the same length and both sequences have the same order pattern. Boldyreva et al. [7] proved that this definition is unachievable for a *stateless* OPE unless the ciphertext space size is super-exponential in the plaintext space size.

2.5.2 POPF-CCA

Because IND-OCPA is unachievable by stateless schemes, Boldyreva et al. [7] introduced another definition that is similar to the security definition of pseudorandom functions (PRFs) and is achievable. The definition says that a scheme is POPF-CCA secure if an adversary cannot distinguish the scheme from a randomly sampled order-preserving function with the same domain and range. POPF-CCA is the only CCA definition as opposed to CPA definitions. We have found that CCA and CPA definitions are interchangeable in this context, since the adversary can always perform a decryption via encryption for OPE and ORE schemes.

2.5.3 IND-FAOCPA

Kerschbaum [29] introduced a new security definition that captures not only IND-OCPA, but also frequency-hiding. The definition follows IND-OCPA, but gives the adversary more power by relaxing the "same order pattern" requirement as follows: the adversary is allowed to have equal elements in one sequence even if the corresponding elements in the other are unequal (but unequal elements must still be ordered the same way in both sequences). This definition has been criticized in [39].

2.5.4 Security of ORE with Leakage

Chenette et al. [14] presented a simulation-based security of ORE with a clearly defined leakage function. Their definition says that the scheme is secure with a leakage $\mathcal{L}(\cdot)$ if there exists an algorithm (simulator) that has access to the leakage function and can generate output indistinguishable from the one generated by the real scheme. Chenette et al. [14] also mention that this definition captures IND-OCPA if $\mathcal{L}(\cdot)$ is set to be the order for every pair of plaintexts.

2.5.5 IND-CPA-DS

Instead of focusing on security of encryption alone, Kerschbaum and Tueno [31] considers the security of the data structure that results after a sequence of plaintexts is inserted. Their definition is limited to the case when the resulting data structure is a simple array. They say that an efficiently searchable encrypted data structure (ESEDS) is secure (IND-CPA-DS) if the adversary, who can choose two equal-length sequences of messages, cannot tell, by looking at a snapshot of the resulting encrypted data structure, which of the two sequences was inserted; moreover, the adversary cannot tell which plaintext corresponds to which array location.

2.5.6 Frequency-hiding order preserving protocol

Roche et al. [45] introduced a simulation-based definition of security for an entire query protocol, which includes information in the data structure and the queries made. Their definition is simulation-based: they call a protocol *frequency-hiding order-preserving* (FH-OP), if there exists an algorithm (simulator) that has access to the sequence of insertions and searches (with input-elements replaced by indices), and the oracle that can compare input elements (given their indices), and can generate output indistinguishable from the one generated by real protocol. They consider a stronger, parameterized definition: if the simulator does not query the oracle for all pairs, and, in fact, after querying its oracle, still cannot infer the order of u pairs, they call such a protocol *frequency-hiding partial order-preserving* (FH-POP) with u incomparable elements pairs; the higher the value of u, the better the security.

2.6 Leakage summary

IND-OCPA secure schemes leak only the order of the original plaintexts and their equality. This definition is ideal, but is unachievable for stateless schemes.

POPF-CCA secure schemes are as secure as a randomly sampled order-preserving function, and thus do not have a clear leakage profile. For example, if the domain size is equal to the range size, there is only one order-preserving function to sample, namely the identity function, and such construction leaks all plaintext values.[1] Thus, the leakage profile depends on the domain and range sizes. Boldyreva, Chenette, and O'Neill [8] show that when the range is bigger than the domain, a POPF-CCA-secure scheme leaks approximately (most significant) half of the bits, with the amount of leakage increasing as more ciphertexts become available.

IND-FAOCPA is a stricter definition than IND-OCPA since it does not leak the equality of the plaintexts (frequency). It still reveals the order of the plaintexts.

ORE security definitions define clear leakage functions, although the implications of such leakage for actual security are usually not obvious. Examples are the location of the most-significant differing bit in [14], the location of the most-significant differing block in [34] and the equality pattern of the most-significant differing bit in [12].

IND-CPA-DS is an adaptation of IND-CPA for data structures. It reveals the order of plaintexts, but does not reveal the equality of ciphertexts and ciphertexts with the minimum and maximum value. It was designed to have less leakage than POPF-CPA, but the exact leakage profile is unclear, like with most definitions in this space.

FH-OP reveals only the order of plaintexts, while FH-POP reveals the order of subsets of plaintexts.

Both IND-CPA-DS and FH-OP are protocol security definitions, in contrast to the other definitions, which describe OPE or ORE schemes. Both of the protocol security definitions do not hide the size of the stored data.

3 Schemes and protocols

We proceed by describing and analyzing the schemes and protocols we have chosen. All plaintexts are assumed to be 32-bit signed integers, or n-bit inputs in complexity analysis. OPE ciphertexts are assumed to be 64-bit signed integers.

Each scheme and protocol has its own subsection where the first part is the construction and security overview, and the second part is our theoretical and experimental analysis.

[1] However, such construction would still be, technically, POPF-CCA secure.

Scheme	Primitive usage		Cipher size, or state size	Security	
	Encryption	Comparison		Definition	Leakage
BCLO [7]	$\boldsymbol{n}$ **HG**	none	$2n$	POPF-CCA	**Half of the bits**
CLWW [14]	n PRF	none	$2n$	ORE with leakage	**MSDB**
Lewi-Wu [34]	$\boldsymbol{2n/d}$ **PRP** $2\frac{n}{d}\left(2^d+1\right)$ PRF $\frac{n}{d}2^d$ Hash	$\frac{n}{2d}$ Hash	$\frac{n}{d}\left(\lambda+n+2^{d+1}\right)+\lambda$	ORE with leakage	The first differing block
FH-OPE [29]	1 Traversal	3 Traversals	$\boldsymbol{3\cdot n\cdot N}$	IND-FAOCPA	Insertion order
CLOZ [12]	n PRF n PPH 1 PRP	$\boldsymbol{n^2}$ **PPH**	$n\cdot h$	ORE with leakage	MSDB equality pattern

Table 1: ORE schemes primitive usage

3.1 BCLO OPE

The OPE scheme by Boldyreva et al. [7] was the first OPE scheme that provided formal security guarantees and was used in one of the first database systems that executes queries over encrypted data (CryptDB [44]).

The core principle of their construction is the natural connection between a random order-preserving function and the hypergeometric probability distribution. Authors formalize this principle proving a bijection between the set of all order-preserving functions from a domain of size M to a range of size $N \geq M$ and the set of all possible combinations of M out of N ordered items.

The encryption algorithm works by splitting the domain into two parts according to a value sampled from the hypergeometric distribution (HG) routine and splitting the range in half recursively. When the domain size contains a single element, the corresponding ciphertext is sampled uniformly from the current range.

To ensure deterministic pseudorandom computations, the authors use a primitive that they defined as *length-flexible pseudorandom function* (LF-PRF) which produces a "tape" — a sequence of pseudorandom bits. Essentially, this primitive is a pseudorandom generator (PRG) with the large-size seed value (generated by PRF). This PRG is seeded with the value derived from the key, current domain, range and plaintext or ciphertext, and they called it TapeGen.

This way not only they ensure that the algorithm is deterministic, but also allow for decryption. The decryption procedure takes the same "path" of splitting domain and range, and when the domain size reaches one, the only left value is the original plaintext.

This scheme is POPF-CCA secure, meaning that it is as secure as the underlying ideal object — randomly sampled order-preserving function from a certain domain to a certain range. For practical values of the parameters, Boldyreva, Chenette, and O'Neill [8] showed that the distance between the plaintexts can be approximated to an accuracy of about the square root of the domain size. In other words, approximately, half of the bits (the most significant) are leaked.

3.1.1 Analysis and implementation challenges

Efficient sampling from hypergeometric distribution is a challenge by itself. Authors suggest using a randomized yet exact (not approximate) Fortran algorithm by Kachitvichyanukul and Schmeiser [25]. It should be noted that the algorithm relies on infinite precision floating-point numbers, which most regular frameworks do not have. The security consequences of finite precision computations is actually an open question. The complexity of this randomized algorithm is hard to analyze; however, we empirically verified that its running time is no worse than linear in the input bit length. The authors also suggest a different algorithm for smaller inputs [49].

On average, encryption and decryption algorithms make n calls to HG, which in turn consumes entropy generated by TapeGen. The entropy, and thus the number of calls to PRG, needed for one HG run is hard to analyze theoretically. However, we derived this number experimentally (see Section 4).

BCLO has been implemented in numerous languages and has been deployed in a number of secure systems. We add C# implementation to the list, and provide some recommendations for practitioners. We recommend using a library that supports infinite precision floating-point numbers when building the hypergeometric sampler.

3.2 CLWW ORE

The ORE scheme by Chenette et al. [14], which authors call "Practical ORE", is the first efficient ORE implementation based on PRFs.

On encryption, the plaintext is split into n values in the following way. For each bit, a value is this bit concatenated with all less significant bits. This value is given to a keyed PRF and the result is added to the next more significant bit. This resulting list of n elements is the ciphertext.

The comparison is trivial. The algorithm compares two lists traversing them in-order looking for the case when one value is greater than the other by exactly one. This would mean that the first differing bit is found. If no such index exists, the plaintexts are equal.

This scheme satisfies ORE security definition with the leakage of the first differing bit (MSDB). Note that the most significant differing bit also leaks the approximate distance between two values. For example, if the most significant differing bit is the last bit, then the plaintexts' difference is one.

3.2.1 Analysis and implementation challenges

On encryption the algorithm makes n calls to PRF and the comparison procedure does not use any cryptographic primitive. Ciphertext is a list of length n, where each element is an output of a PRF modulo 3. The authors claim that the ciphertext's size is $n \log_2 3$, just 1.6 times larger than the plaintext's size. While this may be true for large enough n if ternary encoding is used, we found that in practice the ciphertext size is still $2n$. $1.6n$ for 32-bit words is 51.2 bits, which will have to occupy one 64-bit word, or two 32-bit words, therefore resulting in $2n$ anyway.

3.3 Lewi-Wu ORE

Lewi and Wu [34] introduced an improved version of CLWW [14] which leaks strictly less.

The novel idea was to use the "left / right framework" in which two ciphertexts get generated — left and right. The right ciphertexts are semantically secure, so an adversary will learn nothing from them. Comparison is only defined between the left ciphertext of one plaintext and the right ciphertext of another plaintext.

The idea is to split the plaintext into the blocks of certain number of bits. Next, is to apply CLWW [14] approach to blocks. Within one block, the algorithm computes all possible permutations of values, hashes them and adds the result of the comparison between the value and the non-permuted block value. This way the location of the differing bit inside the block is hidden, but the location of the first differing block is revealed.

Comparison recomputes the hashes for each block and verifies that any of them have the property that one is greater than the other by one.

This scheme satisfies ORE security definition with the leakage of the first differing *block*.

3.3.1 Analysis and implementation challenges

Let n be the size of input in bits (e.g. 32) and d be the number of bits per block (e.g. 2).

Left encryption loops $\frac{n}{d}$ times making one PRP call and two PRF calls each iteration. Right encryption loops $\frac{n}{d}2^d$ times making one PRP call, one hash call and two PRF calls each iteration. Comparison makes $\frac{n}{d}$ calls to hash at worst and half of that number on average. Please note that the complexity of right encryption is exponential in the block size. In the Table 1 the PRP usage is linear due to our improvement.

Assume PRF output size is λ. Left ciphertext size is therefore $\frac{n}{d}(\lambda + n)$. Right ciphertext size is $\lambda + \frac{n}{d}2^{d+1}$. The total ciphertext size is then $\frac{n}{d}\left(\lambda + n + 2^{d+1}\right) + \lambda$.

The implementation details of this approach has an interesting security question. Although the authors suggest using 3-rounds Feistel networks [46] for PRP and use it in their implementation, it may not be secure for small input sizes. Feistel networks security depends on the input size [21] — exponential in the inputs size. However, the typical input for PRP in their scheme is 2–8 bits, thus even exponential number is small.

We have considered multiple PRP implementations to use instead of the Feistel networks. We have found that Knuth shuffle algorithm [32] fits particularly well. It is secure for any input size, and its performance, although degrades with input size, is acceptable for small inputs. Another important aspect of the implementation is that for each block we need to compute the permutation of all the values inside the block. This operation applied many times can be expensive. To address this, we propose to

Protocol	I/O requests (result excluded)		Security	Communication per operation (result excluded)		
	Construction	Query		Construction	Query volume	Query size
B+ tree with ORE	$\mathcal{O}\left(\log_B \frac{N}{B}\right)$	$\mathcal{O}\left(\log_B \frac{N}{B}\right)$	**Same as ORE**	$\mathcal{O}(1)$	$\mathcal{O}(1)$	
Kerschbaum [31]	$\boldsymbol{\mathcal{O}\left(\frac{N}{B}\right)}$	$\mathcal{O}\left(\log_2 \frac{N}{B}\right)$	IND-CPA-DS	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(\log_2 N)$	
POPE [45] cold	$\mathcal{O}(1)$	$\boldsymbol{\mathcal{O}\left(\frac{N}{B}\right)}$	FH-OP	$\mathcal{O}(1)$	$\boldsymbol{\mathcal{O}(N)}$	
POPE [45] warm		$\mathcal{O}\left(\log_L \frac{N}{B}\right)$	**FH-OP Pratial**		$\mathcal{O}(\log_L N)$	$\mathcal{O}(L \log_L N)$

Table 2: Protocols' performance values

generate a PRP table once for the whole block and then use this table when you need to compute the location of an element of permutation. This can reduce the PRP usage (indeed, we observe a reduction from 80 to 32 calls in our case.) We evaluate this improved approach in our experimental section.

3.4 FH-OPE

Frequency-hiding OPE by Kerschbaum [29] is a stateful scheme that hides the frequency of the plaintexts — adversary would not know if any two plaintexts were equal.

This scheme is stateful, which means that the client needs to keep a data structure and update it with every encryption and decryption. The data structure is a binary search tree where the node's value is the plaintext and node's position in a tree is the ciphertext. For example, if the domain is from 1 to 128, and the node is the left child of the root, the ciphertext would be 32.

To encrypt a value, the algorithm traverses the tree until it finds a spot for the new plaintext, or finds the same plaintext. If the same plaintext is found, in order to hide the frequency, the algorithm tosses a coin and goes left or right depending on the outcome up to the leaf. This way, the invariant of the tree — intervals of the same plaintexts do not overlap — is maintained. The ciphertext generated from the new node's position is returned.

The property that every duplicate plaintext will have a new pseudo-random ciphertext makes the scheme randomized. Therefore, the comparison algorithm is more complicated than in the regular deterministic OPE.

To properly compare ciphertexts, the algorithm needs to know the boundaries — the minimum and maximum ciphertexts for a particular plaintext. The client is responsible for traversing the tree to find the plaintext for the ciphertext and then minimum and maximum ciphertext values. Having these values, the comparison is trivial — equality is a check that the value is within the boundaries, and other comparison operators are similar.

The security of the scheme relies on the large range size to domain size ratio. Authors recommend at least 6 times longer ciphertexts than the plaintexts in bit-length, which is 192-bit numbers that are not commonly supported. This scheme satisfies IND-FAOCPA definition, meaning that it does not leak the equality pattern or relative distance between the plaintexts. However, it leaks the insertion order which affects the tree structure.

Authors pay special attention to the state size. In particular, they have a number of heuristics to minimize it, however, these are mostly about compacting the tree and the result highly depends on the tree content. They also introduce a variant of the scheme they call "imperfect frequency hiding". The idea is to generate a new ciphertext for duplicate plaintexts with some probability. The fewer ciphertexts the scheme generates, the better the storage is, the less secure the scheme is. In our analysis, we consider the worst case performance without the use of heuristics. In our experimental evaluation, however, we did implement compaction and we report the results using the heuristics.

3.4.1 Analysis and implementation challenges

If the binary tree grows in only one direction, at some point it will be impossible to generate another ciphertext. In this case, the tree has to be re-balanced. This procedure will invalidate all ciphertexts already generated. However, the client can still generate new ones by manipulating its state. This property makes the scheme difficult to use in some protocols since they usually rely on the ciphertexts on the server being always valid. The authors explicitly mention that the scheme works under the assumption of uniform input. However, the re-balancing will be caused by insertion of just $\log_2 r + 1$ sequential data points for r being the range size, which is 65 for 64-bit integer range.

The scheme makes one tree traversal on encryption and decryption. Comparison is trickier as it requires one traversal to get the plaintext, and two traversals for minimum and maximum ciphertexts. We understand that it is possible to get these values in fewer than three traversals, but we did not optimize the scheme for the analysis and evaluation.

For practitioners we note that the stateful nature of the scheme implies that the client storage is no longer negligible as the state grows proportionally to the number of encryptions. We also note that implementing compaction extensions will affect code complexity and performance. Finally, we stress again that some inputs — namely all non-uniform inputs — can break the scheme by causing all ciphertexts to be invalid. It is up to the users of the scheme to ensure uniformity of the input, which poses serious restrictions on the scheme usage.

3.5 CLOZ ORE

Cash et al. [12] introduced a new ORE scheme that provably leaks less than any previous scheme. Their construction uses bilinear maps for a new primitive they have defined, which allows to hide the first differing bit.

The idea is to use Chenette et al. [14] construction, but permuting the list of PRF outputs. It is not necessary to know the original order of those outputs, as one can simply find a pair where one element is greater than the other by one. This is not enough to reduce leakage, however, since an adversary can count how many elements two ciphertexts have in common.

To address this problem, the authors define a new primitive they call a *property-preserving hash* (PPH).

A PPH as defined in [12] should be instantiated with a predicate (property) on two elements of the domain. PPH Test will then output 1 if the predicate is true for the original elements, and false otherwise. For the purposes of this scheme, the predicate we are interested in is $y = x + 1$, or testing if the first element is greater the the second by 1.

Informally, PPH is correct if PPH Test routine is correct, and PPH is secure if an adversary cannot distinguish a real PPH from a fake one without the secret key. Please refer to the original paper [12] for formal correctness and security definitions.

Equipped with the PPH primitive, the authors "hash" the elements of the ciphertexts before outputting them. Due to security of PPH, the adversary would not be able to count how many elements two ciphertexts have in common, thus, would not be able to tell the location of differing bit.

The strong side of the scheme is its security. The scheme leaks an *equality pattern* of the most-significant differing bits. As defined in [12], the intuition behind equality pattern is that for any triple of plaintexts m_1, m_2, m_3, it leaks whether m_2 differs from m_1 before m_3 does.

3.5.1 Analysis and implementation challenges

On encryption, the scheme makes n calls to PRF, n calls to PPH Hash and one call to PRP. Comparison is more expensive, however, as the scheme makes n^2 calls to PPH Test. Ciphertext size depends on PPH implementation and is equal to n times the PPH hash size.

The scheme has two limitations that make it impractical. The first one is the square number of calls to PPH, which is around 1024 for a single comparison.

The second problem is the PPH itself. Authors suggest a construction based on bilinear maps. Hash of an argument is an element of a group, and the test algorithm is computing a pairing. This operation is very expensive — order of magnitude more expensive than any other primitive we have implemented for other schemes. Pairings are also hard to implement efficiently. The best open source implementation of pairings is C++ library PBC, and the fastest pairing (without pre-processing) is computed in 25 ms according to PBC benchmarks [38]. Combined with 1024 calls to the primitive, the expected performance of Cmp algorithm is 25.6 seconds on average for unequal plaintexts, and 51.2 seconds for equal ones.

Due to complexity and impracticality of the PPH primitive, we implemented a "fake" version of it — correct, but insecure, which does not use pairings. Therefore, in our analysis we did not benchmark the speed of the scheme. We measured all other data nevertheless.

3.6 Search protocol from ORE

So far we have analyzed OPE and ORE schemes without much context. One of the best uses of an ORE is within a secure protocol. In this section we provide a construction of a search protocol built with a B+ tree working on top of an ORE scheme and analyze its security and performance.

The general idea is to consider some data structure that is optimized for range queries, and to modify it to change all comparison operators to ORE Cmp scheme calls. This way the data structure can operate only on ciphertexts. Performance overhead would be that of using the ORE scheme's Cmp routine instead of a plain comparison. Space overhead would be that of storing ciphertexts instead of plaintexts. Security of such construction is not as trivial since the data structure itself leaks something in addition to the scheme's leakage.

In this paper, we have implemented B+ tree [3] (with a proper deletion algorithm [24]) as a data structure.

B+ tree is a good choice for underlying data structure because all its operations — insertions, updates, searches and deletions — require nothing beyond ordering of its elements. Moreover, these operations require comparisons of arguments to already inserted elements, thus enabling the use of "left / right framework".

The B+ tree leaks nothing more than the underlying ORE scheme, except for the smallest and the largest element. In particular, if the underlying (ideal) scheme reveals only the order, B+ tree constructed out of scheme's ciphertexts will only reveal, which ciphertext is smallest and largest, in addition to the order of the ciphertexts.

For protocols, we also analyze the I/O performance and the communication cost. In particular, we are interested in the expected number of I/O requests the server would have made to the secondary storage, and the number and size of messages parties would have exchanged.

The relative performance of the B+ tree depends only on the page capacity (ciphertexts are larger than the plaintexts and therefore the B+ tree will have smaller branching factor). Therefore, the query complexity is:

$$\mathcal{O}\left(\log_B\left({}^{N}/_{B}\right) + {}^{r}/_{B}\right)$$

where B is the number of records (ciphertexts) in a block, N is the number of records (ciphertexts) in the tree and r is the number of records (ciphertexts) in the result (none for insertions).

Communication volume of the protocol is relatively small. For insertions, the client transfers one ciphertext in one message. For queries, the client transfers two ciphertexts in one message, and gets one message of result size back.

3.7 Kerschbaum-Tueno

Kerschbaum and Tueno [31] proposed a new data structure, which satisfies IND-CPA-DS security definition and is efficiently searchable according to their own definition (search operation has poly-logarithmic running time and linear space complexity).

In short, the idea is to maintain a (circular) array of symmetrically encrypted ciphertexts in order. On insertion, the array is rotated around a uniformly sampled offset to hide the location of the smallest element. Client interactively performs a binary search requesting an element, decrypting it and deciding which way to go.

Authors prove that this construction is IND-CPA-DS secure. It provably hides the frequency due to semantic encryption and hides the location of the first element due to random rotations.

3.7.1 Analysis and implementation challenges

Insertions are I/O-heavy because they involve rotation of the whole data structure. All records will be read and written, thus the complexity is $\mathcal{O}\left({}^{N}/_{B}\right)$. Searches are faster since they involve logarithmic number of blocks. The first few blocks can be cached and the last substantial number of requests during the binary search will target a small number of blocks. The complexity is then $\mathcal{O}\left(\log_2 {}^{N}/_{B}\right)$.

Communication volume is small as well. Insertion requires $\log_2 N$ messages from each side. Searches require double that number because separate protocol is run for both endpoints. Inherently, the response is sent in a single message.

The data structure is linear in size, and the client storage is always small. Sizes of messages are also small as only a single ciphertext is usually transferred.

For practitioners we have a few points. The construction in the original paper [31] contains a typo as m and m' must be swapped in the insertion algorithm. Also, we have found some rare edge cases; when duplicate elements span over the modulo, the algorithm may not return the correct answer. Both inconsistencies can be fixed however. This protocol is not optimized for I/O operations for insertions, and thus would be better suited for main memory datasets.

Protocol	I/O requests (result included)		Communication per operation (result excluded) Volume (messages)		Size (bytes)	
	Construction	Query	Construction	Query	Construction	Query
B+ tree w. ORE	3	87	2	2	177	342
Kerschbaum [31]	**490**	11	40	86	671	1454
POPE [45] cold	1	**1482**	2	**494996**	32	**8974948**
POPE [45] warm		324		904		43690

Table 3: Simulation result for protocols' performance values

3.8 POPE

Roche et al. [45] presented a protocol, direct improvement over mOPE [43], which is particularly suitable for large number of insertions and small number of queries. The construction is heavily based on buffer trees [2] to support fast insertion and lazy sorting.

The idea is to maintain a POPE tree on the server and have the client manipulate that tree. POPE tree is similar to B-tree, in that the nodes have multiple children and nodes are sorted on each level. Each node has an ordered list of *labels* of size L and an unbounded unsorted set of encrypted data called buffer. Parameter L controls the list size, the leaf's buffer size, and the size of client's working set. The insertion procedure simply adds an encrypted piece of data to the root's buffer.

The query procedure is more complex. To answer a query, the server interacts with the client to split the tree according to the query endpoints. First, buffers of all internal nodes from root to the leaf containing the endpoint are cleared. Clearing a buffer means sending elements to the children according to the client's instructions. The second stage is reducing leaf's size to L by splitting the leaf and updating parents, like in regular B-tree. It may happen that after a number of leaves' splits intermediate node's list size exceeds L. That is why in the final stage, rebalancing, splits intermediate nodes to ensure size constraint. If the root has an overflow, the root is split and a new root is created, increasing the height of the tree. After the tree is split according to both endpoints, answering a query means sending all ciphertexts in all buffers between the two endpoint leaves.

This construction satisfies the security definition of frequency-hiding partial order preserving (FH-POP) protocol. According to Theorem 3 from [45], after n insertions and m query operations with local storage of size L, where $mL \in o(n)$, the POPE scheme is a frequency-hiding partial-order-preserving with $\Omega\left(\frac{n^2}{mL\log_L n} - n\right)$ incomparable pairs of elements. Aside from leaking pairwise order of a bounded number of elements, the construction provably hides the frequency (i.e. equality) of the elements.

The authors provide cost analysis of their construction. Insertions are always cheap — one round is needed to send an element to the server. Search operations are expected to require $\mathcal{O}(\log_L n)$ rounds. It must be noted that the first queries will require many more rounds, since large buffers must be sorted.

3.8.1 Analysis and implementation challenges

In our analysis we count each request-response communication as a round. This is different from [45] where they use *streaming* a number of elements as a single round. The rationale for our approach is that if we allow persistent channels additionally to messages, then any protocol can open a channel for each operation. Thus, we do not allow channels for all protocols in our analysis.

Also, as noted by authors, if $L = n^\epsilon$ for $0 < \epsilon < 1$, then the amortized costs become $\mathcal{O}(1)$. While this is true, in our analysis the choice of L depends on the storage volume block size for I/O optimizations, instead of the client's volatile storage capacity. Thus, the costs remain logarithmic.

Insertion bandwidth is constant and small — one ciphertext is transferred. Search bandwidth depends heavily on the current state of the tree. When the tree is completely unsorted (the first query), all elements of the tree will be transferred to split the large root, then possibly internal node will have to be split requiring sending of $\frac{N}{L}$ elements, and so on, thus $\mathcal{O}(N + r)$. When the tree is completely sorted (after a large number of uniform queries), the bandwidth will be similar to that of a standard B+ tree — $\mathcal{O}(L\log_L N + r)$. The average case is hard to compute; however, authors prove an upper bound on bandwidth after n insertions and m queries — $\mathcal{O}(mL\log_L n + n\log Lm + n\log L(\ln n))$.

POPE tree is not optimized for I/O the way B-tree is. Insertion requires a single I/O request for the block where the server appends the element. Search complexity is more complex to analyze as

is bandwidth complexity. In the worst-case (first query), all blocks need to be accessed $\mathcal{O}\left(\frac{N}{B}+\frac{r}{B}\right)$. In the best-case all nodes occupy exactly one block and I/O complexity is the same as with B+ tree $\mathcal{O}\left(\log_L \frac{N}{B}+\frac{r}{B}\right)$. The average case is in between and matters get worse as the node is not guaranteed to occupy a single block due to arbitrary sized buffers.

Client's persistent storage is negligibly small — it stores the encryption key. Volatile storage is bounded by L.

For practitioners we present a number of things to consider. Buffer within one node is unsorted, so in the worst-case, L-sized chunks remain unordered. Due to this property, the query result may contain up to $2(L-1)$ extra entries, which the client will have to discard from the response.

The first query after a large number of insertions will result in client sorting the whole N elements, and thus, POPE has different performance for cold and warm start. Also, even to navigate an already structured tree, the server has to send to the client the whole L elements and ask where to go on all levels.

Furthermore, [45] does not stress the fact that after alternating insertions and queries, it may happen that some intermediate buffers are not empty, thus returning buffers between endpoints must include intermediate buffers as well. The consequence is that the whole subtree is traversed between paths to endpoints, unlike the B+ tree case when only leaves are involved.

Finally, POPE tree is not optimized for I/O operations. Even if L is chosen so that the node fits in the block, only leaves and only after some number of searches will optimally fit in blocks. Arbitrary sized buffers of intermediate nodes and the lack of underflow requirement do not allow for I/O optimization.

4 Evaluation

All experiments were conducted on a single machine. We use macOS 10.13.6 with 8-Core 3.2GHz Intel Xeon W processor, 32 GB DDR4 ECC main memory and 1 TB SSD disk. The main code is written in C# and runs on .NET Core 2.1.3.

4.1 Implementation

We have implemented most of the primitives, data structures, and constructions ourselves. For some primitives and all schemes we provided the first open-sourced cross-platform C# implementation. We emphasize that neither primitives, nor schemes are production-ready; however, we believe they can be used in research projects and prototypes.

This software project is documented and tested (over 97% coverage). All code including primitives, data structures, schemes, protocols, simulation logic, benchmarks, build scripts and tests is published on GitHub [6] under CC BY-NC 4.0 license. Additionally, we have published parts of the projects as stand-alone .NET Core (nuGet) packages, and we will host a web-server where users can run simulations for small inputs.

4.1.1 Primitives

All schemes and protocols use the same primitives most of which we implemented ourselves. All primitives rely on the default .NET Core AES implementation. .NET Core uses platform-specific implementation of AES, thus leverages AES-NI instruction. In our project all keys' sizes are 128 bits, as is AES block size.

The most used primitive is a pseudo-random generator (PRG). We implemented AES-based PRG which uses AES in CTR mode and caches unused entropy. This way we can supply seed as large as 128 bits (AES key). PRG generates entropy in 128 bits chunks, and if only a fraction of the entropy is requested, the residue is carried over to the next call. When converting entropy to integers, we use techniques to reduce bias (e.g. discarding the entropy in some cases).

The second most used primitive is a PRF. We implemented it using AES without initialization vector in ECB mode. Since we used a single block, this approach is still secure. For symmetric encryption we use AES with initialization vector in CBC mode. For hash we use default .NET Core SHA2 implementation. If keyed hash is used, we push the input through a PRF before supplying it to SHA2.

For pseudo-random permutation (PRP) we have two implementations. We implemented unbalanced Feistel networks [46] using our PRF and .NET Core `BitArray` class. We have a regular (3 round) and strong (4 rounds) version. The second implementation of PRP is for small inputs. We generate a permutation table using Knuth shuffle [32] and cache it for the next call. This implementation is more

Scheme	Encryption	Comparison	Size (bits)
BCLO [7]	**41 HG**	none	64
CLWW [14]	32 PRF	none	64
Lewi-Wu [34]	**32 PRP** 160 PRF 64 Hash	9 Hash	2816
FH-OPE [29]	1 Traversal	1 Traversal	**86842**
CLOZ [12]	32 PRF 32 PPH 1 PRP	**1046 PPH**	4096

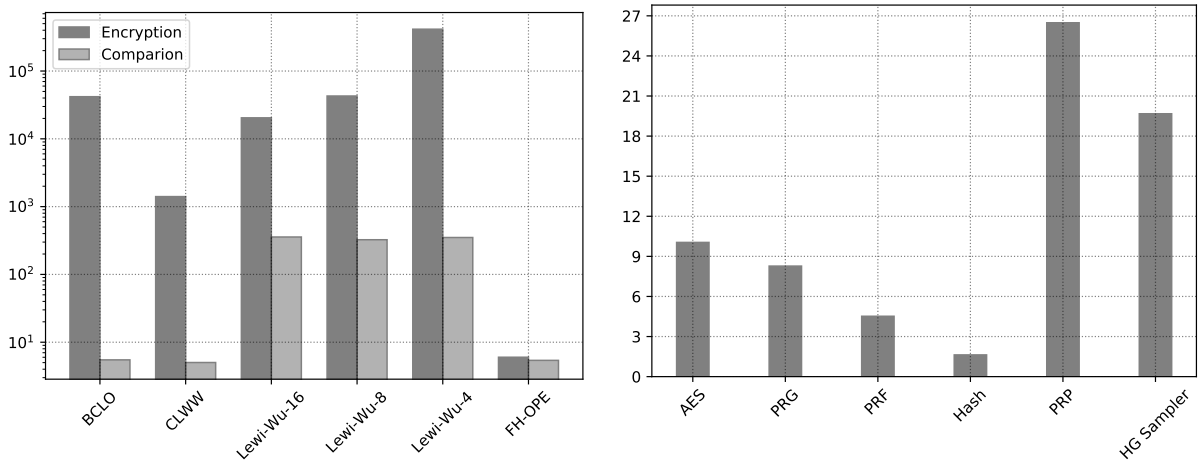
Table 4: Simulation result for ORE schemes primitive usage

secure for small inputs and is optimized if the whole permutation is needed. Both schemes (Lewi-Wu and CLOZ) that use PRP, use the Knuth shuffle implementation because of the small inputs.

BCLO [7] relies on special primitives. We implemented LF-PRF (TapeGen in [7]) using our implementations of PRF and PRG as suggested in [7]. For implementation of hypergeometric sampler we used algorithm [25] and code from GitHub [44].

4.1.2 Schemes and protocols

We implemented schemes and protocols precisely as in the original papers. When we found problems or improvements, we put those in implementation challenges notes, but did not alter the original designs in our code, unless explicitly stated. Each ORE scheme implements a C# interface; thus our own implementation of B+ tree operates on a generic ORE. For baseline, we have a stub implementation of the interface, which has identity functions for encryption and decryption. It is important to note that all schemes and protocols use exclusively our implementations of primitives. Thus we rule out the possible bias of one primitive implementation being faster than the other.



(a) Schemes benchmark (time in microseconds, log scale). Lewi-Wu parameter is the number of blocks.

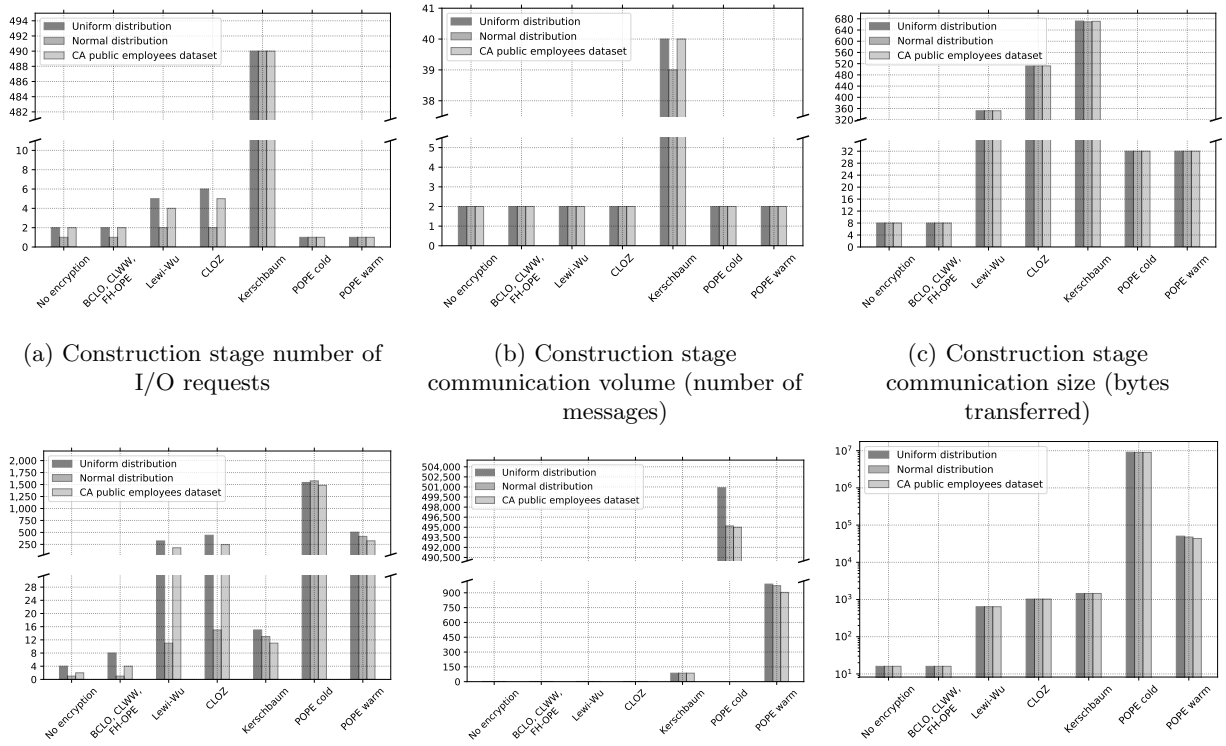
(b) Primitives benchmark (time in microseconds)

Figure 1: Schemes and primitives benchmarks

4.1.3 Simulations

We have four types of simulations.

Protocol simulation runs the three protocol stages — handshake, construction and search — on supplied data for all protocols including all schemes coupled with B+ tree. In this simulation we measure the primitive usage, number of ORE scheme operations (when applies), communication volume and



(a) Construction stage number of I/O requests

(b) Construction stage communication volume (number of messages)

(c) Construction stage communication size (bytes transferred)

(d) Queries stage number of I/O requests

(e) Queries stage communication volume (number of messages)

(f) Queries stage communication size (bytes transferred, log scale)

Figure 2: Performance values for different data distributions

size, and the number of I/O requests. We intentionally do not measure elapsed time, since it would be extremely inaccurate in this setting — simulation and measurement routines take substantial fraction of time.

Scheme simulation runs all five ORE schemes and tracks only the primitive usage.

The scheme benchmark, however, is designed to track time. We use Benchmark.NET [41] to ensure that reported time is accurate. This tool handles things like cold / warm start, elevating process' priority, and performing enough runs to draw statistically sound conclusions. This benchmark reports elapsed time up to nanoseconds for all five schemes and their variants.

Finally, primitive benchmark uses the same tool, but benchmarks the primitives. We use it to compare different implementations of primitives (e.g. Feistel PRP vs pre-generated permutation) and to have basis to approximate scheme and protocol time consumption based on primitive usage.

Simulation and benchmark routines are documented, tested and published as the rest of the software.

4.2 Setup

For our simulations, we have used three datasets. Two synthetic distributions, that are uniform (range is all 32 bit signed integers) and normal (with mean 0 and standard deviation 1000). The real datasets is California public employees salaries [48] ("total pay and benefits" column). Synthetic datasets and subsets of the real dataset are generated pseudo-randomly.

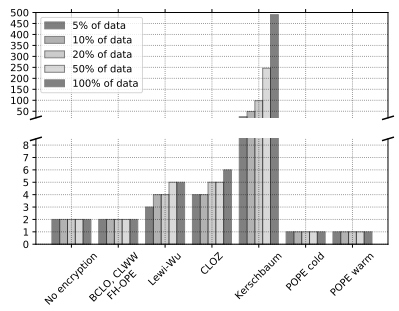
4.3 Results

4.3.1 Schemes primitive usage

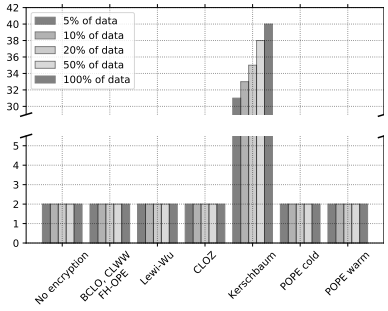
In Table 4 we show the simulation-derived values of each OPE and ORE scheme's primitive usage. Each scheme is given 1000 data points of each dataset. First, scheme encrypts each data point, then decrypts each ciphertext and then performs five comparisons (all possible types) pairwise. This micro-simulation is repeated 100 times. Resulting values for primitive usage are averaged for each scheme. State and ciphertext sizes are calculated after each operation and the values are averaged.

The FH-OPE number of traversals for comparison is one since endpoints are found once and then five comparisons are made for the same two ciphertexts. The FH-OPE state size is smaller than expected due to compactions described in [29]. Ciphertext size of CLOZ assumes that the output size of PPH hash is 128 bits. Please note that the simulated values are consistent with the theoretical calculations.

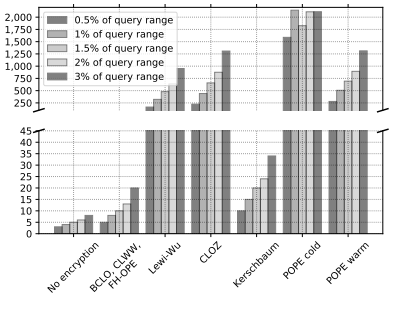
4.3.2 Schemes and primitive benchmarks



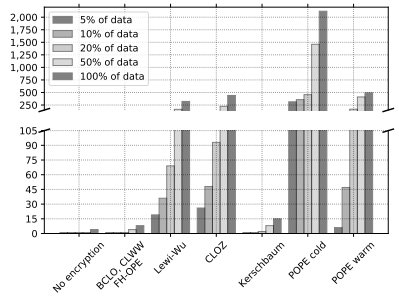
(a) Construction stage number of I/O requests



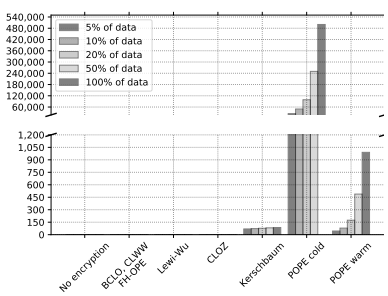
(b) Construction stage communication volume



(a) Queries stage number of I/O requests for different query sizes

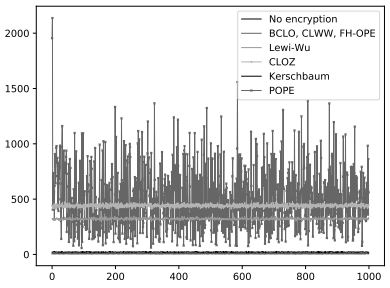


(c) Queries stage number of I/O requests



(d) Queries stage communication volume

Figure 3: Protocol scalability



(b) Number of I/O requests over time (number of executed queries).

Figure 4: Number of I/O requests for different queries

Using the Benchmark.NET tool [41], we have accurately tracked the performance of the schemes and primitives running of different parameters (see Figure 1). Benchmark.NET decides how many times to run the routine to get statistically sound results. The routines are simply invocations of schemes and primitives of randomly sampled data.

Please note the logarithmic scale of schemes performances. FH-OPE is fast since it does not perform CPU-heavy operations and works in main memory. CLWW is the fastest stateless scheme. Its comparison is so simple that it works almost as fast as regular integer comparison. Lewi-Wu performance degrades exponentially with the increase of block size mainly due to exponential number of PRF executions and the performance of PRP degrading exponentially. Note also that Lewi-Wu comparison takes noticeable time due to Hash primitive usage.

In primitives benchmark it is clear that most primitives use AES under the hood. PRG and PRF take less than AES because they do not include initialization vector generation needed for symmetric encryption. PRP is implemented as a Knuth shuffle [32] and its complexity is exponential in input bit length. Input size of 2 bits is shown on Figure 1. Both PRG and PRP make use of internal cache in our implementations. PRG does not discard the entropy generated by AES cycle, so one AES cycle can supply four 32-bits integers. PRP generates the permutation table once and does not regenerate it if the same key and number of bits are supplied. Hypergeometric sampler uses PRG internally and therefore its performance depends heavily on the PRG implementation. From this plot it is clear why a linear number of sampler usage in BCLO may be better than exponential number of PRP usage in Lewi-Wu.

4.3.3 Protocols

In this experiment we have run each protocol (plus B+ tree with no encryption for the baseline) with each of the three datasets. Datasets sizes are 247000 and the number of queries is 1000. Queries are

generated uniformly at random with a fixed range — 1% of data range. The cache size is fixed to 128 blocks, and the B+ tree branching factor as well as block sizes for other protocols are set such that the page size is 4 kilobytes. It effectively means that BCLO, CLWW and FH-OPE branching factors are 512, Lewi-Wu gets 11, CLOZ gets 8, and Kerschbaum with POPE get 256 elements per page. The values we are measuring are the number of I/O operations, communication volume, and size for both construction and query stages. Being "cold" in our simulations means executing the first query. See Table 3 for the snapshot for particular distribution (CA employees). Values for ORE based protocols are averaged.

Figure 2 shows all values we tracked for all protocols and distributions.

Note that all ORE based protocols behave the same except when ciphertext size matters. Thus, since BCLO, CLWW and FH-OPE have the same ciphertext size, they create B+ trees with the same page capacity and have the same number of I/Os for different operations. Lewi-Wu and CLOZ schemes have relatively large ciphertexts and thus induce larger traffic (see Subfigure 2c) and smaller B+ tree branching factor resulting in greater number of I/O requests (see Subfigure 2d).

Kerschbaum protocol requires high number of I/O requests during construction since it needs to insert an element into the arbitrary place in an array and rotate the data structure on a disk. It also induces large communication volume since the insertion is interactive unlike in other protocols.

POPE suffers huge penalty on the first query (see Subfigures 2d, 2e and 2f) since it reads and sends all blocks to the client for sorting. POPE performance improves as more queries are executed.

Note that the values do not vary a lot among different data distributions except for I/O requests. I/O performance depends on the result size for queries, and is therefore more sensitive to data distribution.

Also note that using an ORE scheme in B+ tree does not add any substantial I/O overhead (see "No encryption").

On Figure 4a it is clear that query performance does not depend substantially on the query size. ORE schemes with large ciphertext sizes are little more sensitive to a query size since the number of blocks needed to answer the query increases.

Figure 3 shows Table 2 asymptotic values. Kerschbaum construction I/Os and cold POPE query values grow linearly with inputs, while the other protocols grow logarithmically.

On Figure 4b protocols' performance over time is shown. Evidently, POPE is the only protocol where cold vs warm makes a difference. It incurs massive amount of work whenever it hits a large unordered buffer.

4.4 Remarks

Having done theoretical and practical evaluations of the protocols, we have found that primitive usage is a much better performance measure than the plain time measurements. We have also found that I/O optimizations is a vital characteristic of a protocol and cannot be neglected. When it comes to a practical use, the observed time of a query execution is a mix of a number of factors and I/O requests can slow the system down dramatically.

Following are our recommendations for practitioners. First of all, a practitioner has to decide on the acceptable level of leakage and the minimum performance requirements. All protocols trade security for performance.

BCLO [7] is an efficient stateless construction whose main advantage is the format of its ciphertexts. This property allows BCLO to be used directly in the legacy systems where it is infeasible to alter the comparison routine.

Frequency-hiding OPE [29] is fast, producing numerical ciphertexts, but is stateful and has the uniformity requirement. We recommend using FH-OPE if the amount data is small and the client's main memory would fit the state. One also has to ensure uniformity of the input.

Lewi-Wu [34] is easily customizable in terms of tuning performance to security ratio, and it offers the benefits of left / right framework. This makes it a preferred choice for B+ tree based protocols, especially if one can construct a tree on the client and then upload it, utilizing the semantic security of the plaintexts.

CLWW [14] has similar to Lewi-Wu construction, but provides weaker security guarantees for the sake of performance. CLWW is the fastest of the protocols we analyzed, so it is suggested for systems with high performance requirements.

Kerschbaum protocol [31] offers semantically secure ciphertexts, hiding the location of the smallest and largest of them, and has a simple implementation. The protocol is well-suited for bulk insertions and scales well. We recommend it for systems where one does bulk insertions or uploads the data in advance.

POPE [45] offers a "deferred" B+ tree implementation. By deferring the sorting of its ciphertexts, POPE remains more secure for the small number of queries. POPE has the fastest insertion routine and does not reveal the order of most of its ciphertexts. We recommend POPE for the systems where there are a lot more insertions than queries. An example of such system could be an archive where users usually upload, but rarely download. We would also recommend to "warm up" the structure to avoid a substantial delay upon the first query.

We found our framework to be a powerful tool for analyzing the protocols. We encourage protocol developers to contribute their implementations and running the corresponding simulations.

5 Conclusion

In this paper we present the first comprehensive comparison of the most important OPE and ORE schemes and protocols and discuss their advantages and disadvantages. We discuss security guarantees and explain the leakage profile for each one; furthermore we discuss some limitations and improvements to certain schemes that make them more practical and/or secure. We also present an analytical and experimental comparison of the performance of these methods. In addition, we provide suggestions to a practitioner, who may want to use some of these schemes.

Last but not least, we provide source code for our framework [6] that includes very efficient implementations of these schemes and protocols.

An important future work is to understand better the meaning of the different leakage profiles and their implications. Furthermore, another direction is to try to improve the performance of the most secure schemes (e.g. [12]).

6 Acknowledgments

We would like to thank Adam O'Neill and George Kellaris for helpful discussions. George Kollios and Dmytro Bogatov were supported by an NSF SaTC Frontier Award CNS-1414119. Leonid Reyzin was supported in part by NSF grant 1422965.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. "Order Preserving Encryption for Numeric Data". In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD '04. ACM, 2004, pp. 563–574.

[2] L. Arge. "The Buffer Tree: A Technique for Designing Batched External Data Structures". In: *Algorithmica* 37.1 (Sept. 2003), pp. 1–24.

[3] R. Bayer and E. McCreight. "Organization and Maintenance of Large Ordered Indices". In: *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '70. ACM, 1970, pp. 107–141.

[4] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. "The Tao of Inference in Privacy-protected Databases". In: *Proc. VLDB Endow.* 11 (2018), pp. 1715–1728.

[5] T. Boelter, R. Poddar, and R. A. Popa. "A Secure One-Roundtrip Index for Range Queries". In: *IACR Cryptology ePrint Archive* (2016), p. 568.

[6] D. Bogatov. *ORE Benchmark*. `https://github.com/dbogatov/ore-benchmark`. 2018.

[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. "Order-Preserving Symmetric Encryption". In: *Advances in Cryptology - EUROCRYPT 2009*. Springer Berlin Heidelberg, 2009, pp. 224–241.

[8] A. Boldyreva, N. Chenette, and A. O'Neill. "Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions". In: *Advances in Cryptology – CRYPTO 2011*. Springer Berlin Heidelberg, 2011, pp. 578–595.

[9] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. "Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation". In: *Advances in Cryptology - EUROCRYPT 2015*. Springer Berlin Heidelberg, 2015, pp. 563–594.

[10] M. Bun and M. Zhandry. "Order-Revealing Encryption and the Hardness of Private Learning". In: *Theory of Cryptography*. Springer Berlin Heidelberg, 2016, pp. 176–206.

[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. "Leakage-Abuse Attacks Against Searchable Encryption". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 668–679.

[12] D. Cash, F.-H. Liu, A. O'Neill, M. Zhandry, and C. Zhang. "Parameter-Hiding Order Revealing Encryption". In: *Advances in Cryptology – ASIACRYPT 2018*. 2018. Forthcoming.

[13] D. Cash, F.-H. Liu, A. O'Neill, and C. Zhang. *Reducing the Leakage in Practical Order-Revealing Encryption*. Cryptology ePrint Archive, Report 2016/661. 2016.

[14] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. "Practical Order-Revealing Encryption with Limited Leakage". In: *Fast Software Encryption*. Springer Berlin Heidelberg, 2016, pp. 474–493.

[15] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. N. Garofalakis. "Practical Private Range Search Revisited". In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pp. 185–198.

[16] F. B. Durak, T. M. DuBuisson, and D. Cash. "What Else is Revealed by Order-Revealing Encryption?" In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1155–1166.

[17] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes. "A Structure Preserving Database Encryption Scheme". In: *Secure Data Management*. Springer Berlin Heidelberg, 2004, pp. 28–40.

[18] J. Eom, D. H. Lee, and K. Lee. "Multi-Client Order-Revealing Encryption". In: *IEEE Access* (2018), pp. 45458–45472.

[19] P. Grubbs, T. Ristenpart, and V. Shmatikov. "Why Your Encrypted Database Is Not Secure". In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM, 2017, pp. 162–168.

[20] H. Haagh, Y. Ji, C. Li, C. Orlandi, and Y. Song. "Revealing Encryption for Partial Ordering". In: *Cryptography and Coding*. Springer International Publishing, 2017, pp. 3–22.

[21] V. T. Hoang and P. Rogaway. "On Generalized Feistel Networks". In: *Proceedings of the 30th Annual Conference on Advances in Cryptology*. Springer-Verlag, 2010, pp. 613–630.

[22] M. S. Islam, M. Kuzu, and M. Kantarcioglu. "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation". In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. 2012.

[23] M. S. Islam, M. Kuzu, and M. Kantarcioglu. "Inference attack against encrypted range queries on outsourced databases". In: *Fourth ACM Conference on Data and Application Security and Privacy, CODASPY'14, San Antonio, TX, USA - March 03 - 05, 2014*. 2014, pp. 235–246.

[24] J. Jannink. "Implementing Deletion in B+-trees". In: *SIGMOD Rec.* 24.1 (Mar. 1995), pp. 33–38.

[25] V. Kachitvichyanukul and B. Schmeiser. "ALGORITHM 668: H2PEC: sampling from the hypergeometric distribution". In: 14 (Dec. 1988), pp. 397–398.

[26] 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". In: (2010), pp. 2520–2533.

[27] H. Kadhem, T. Amagasa, and H. Kitagawa. "Optimization Techniques for Range Queries in the Multivalued-partial Order Preserving Encryption Scheme". In: *Knowledge Discovery, Knowledge Engineering and Knowledge Management*. Springer Berlin Heidelberg, 2013, pp. 338–353.

[28] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. "Generic Attacks on Secure Outsourced Databases". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1329–1340.

[29] F. Kerschbaum. "Frequency-Hiding Order-Preserving Encryption". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 656–667.

[30] F. Kerschbaum and A. Schroepfer. "Optimal Average-Complexity Ideal-Security Order-Preserving Encryption". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 275–286.

[31] F. Kerschbaum and A. Tueno. "An Efficiently Searchable Encrypted Data Structure for Range Queries". In: *arXiv preprint arXiv:1709.09314* (2017).

[32] D. E. Knuth. *Seminumerical algorithms*. 3rd ed. Vol. 2. Addison-Wesley, 2016, pp. 145–146.

[33] M. Lacharite, B. Minaud, and K. G. Paterson. "Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 297–314.

[34] K. Lewi and D. J. Wu. "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds". In: ACM, 2016, pp. 1167–1178.

[35] D. Liu and S. Wang. "Programmable Order-Preserving Secure Index for Encrypted Database Query". In: *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*. 2012, pp. 502–509.

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

[37] Z. Liu, K.-K. R. Choo, and M. Zhao. "Practical-oriented protocols for privacy-preserving outsourced big data analysis: Challenges and future research directions". In: *Computers & Security* 69 (2017), pp. 97–113.

[38] B. Lynn. *PBC Benchmarks*. 2018. URL: `https://crypto.stanford.edu/pbc/times.html` (visited on 08/15/2018).

[39] M. Maffei, M. Reinert, and D. Schröder. "On the Security of Frequency-Hiding Order-Preserving Encryption". In: *Proceedings of the International Conference on Cryptology and Network Security*. Springer, 2017.

[40] M. Naveed, S. Kamara, and C. V. Wright. "Inference Attacks on Property-Preserving Encrypted Databases". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 644–655.

[41] .NET Foundation. *Benchmark.NET*. `https://github.com/dotnet/BenchmarkDotNet`. 2018.

[42] G. Özsoyoglu, D. A. Singer, and S. S. Chung. "Anti-Tamper Databases: Querying Encrypted Databases". In: *Data and Applications Security XVII: Status and Prospects, IFIP TC-11 WG 11.3 Seventeenth Annual Working Conference on Data and Application Security, August4-6, 2003, Estes Park, Colorado, USA*. 2003, pp. 133–146.

[43] R. Popa, F. Li, and N. Zeldovich. "An Ideal-Security Protocol for Order-Preserving Encoding". In: *IEEE Symposium on Security and Privacy*. 2013, pp. 463–477.

[44] R. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. "CryptDB: Protecting Confidentiality with Encrypted Query Processing". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. ACM, 2011, pp. 85–100.

[45] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich. "POPE: Partial Order Preserving Encoding". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1131–1142.

[46] B. Schneier and J. Kelsey. "Unbalanced Feistel networks and block cipher design". In: *Fast Software Encryption*. Springer Berlin Heidelberg, 1996, pp. 121–144.

[47] I. Teranishi, M. Yung, and T. Malkin. "Order-Preserving Encryption Secure Beyond One-Wayness". In: *Advances in Cryptology – ASIACRYPT 2014*. Springer Berlin Heidelberg, 2014, pp. 42–61.

[48] Transparent California. *2017 salaries for State of California*. `https://transparentcalifornia.com/salaries/2017/state-of-california/`. 2017.

[49] A. J. Walker. "An Efficient Method for Generating Discrete Random Variables with General Distributions". In: *ACM Trans. Math. Softw.* 3.3 (Sept. 1977), pp. 253–256.

[50] X. Wang and Y. Zhao. "Order-Revealing Encryption: File-Injection Attack and Forward Security". In: *Computer Security*. Springer International Publishing, 2018, pp. 101–121.

[51] S. Wozniak, M. Rossberg, S. Grau, A. Alshawish, and G. Schaefer. "Beyond the Ideal Object: Towards Disclosure-resilient Order-preserving Encryption Schemes". In: *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop*. ACM, 2013, pp. 89–100.

[52] L. Xiao and I.-l. Yen. *A Note for the Ideal Order-Preserving Encryption Object and Generalized Order-Preserving Encryption*.

[53] L. Xiao, I.-L. Yen, and D. T. Huynh. "Extending Order Preserving Encryption for Multi-User Systems". In: *IACR Cryptology ePrint Archive* (2012), p. 192.