A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols

Dmytro Bogatov
Boston University
Boston, MA 02215
dmytro@bu.edu

George Kollios
Boston University
Boston, MA 02215
gkollios@bu.edu

Leonid Reyzin
Boston University
Boston, MA 02215
reyzin@bu.edu

ABSTRACT

Database query evaluation over encrypted data can allow database users to maintain the privacy of their data while outsourcing data processing. Order-Preserving Encryption (OPE) and Order-Revealing Encryption (ORE) were designed to enable efficient query execution, but provide only partial privacy. More private protocols, based on Searchable Symmetric Encryption (SSE), Oblivious RAM (ORAM) or custom encrypted data structures, have also been designed. In this paper, we develop a framework to provide the first comprehensive comparison among a number of range query protocols that ensure varying levels of privacy of user data. We evaluate five ORE-based and five generic range query protocols. We analyze and compare them both theoretically and experimentally and measure their performance over database indexing and query evaluation. We report not only execution time but also I/O performance, communication amount, and usage of cryptographic primitive operations. Our comparison reveals some interesting insights concerning the relative security and performance of these approaches in database settings.

PVLDB Reference Format:
Dmytro Bogatov, George Kollios and Leonid Reyzin. A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols. *PVLDB*, 12(8): 933-947, 2019.
DOI: https://doi.org/10.14778/3324301.3324309

1. INTRODUCTION

Order-Preserving Encryption (OPE) was proposed by Agrawal et al. [1] in their seminal paper. The main idea is to "encrypt" numerical values into ciphertexts that have the same order as the original 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. A scheme was proposed in [1] but no security analysis was given.

Boldyreva et al. [8] were the first to treat OPE schemes from a cryptographic point of view, providing security models and rigorous analysis. The ideal functionality of such a scheme is to leak only the order of the plaintexts and nothing more. However, it was shown by Boldyreva et al. [8] that the ideal functionality is not achievable if the scheme is *stateless* and *immutable*. In order to achieve the ideal functionality, Popa, Li, and Zeldovich [56] proposed a mutable scheme that constructs a binary tree on plaintexts and uses paths as ciphertexts. This tree is the encrypted full state of the dataset, and once an insertion or a deletion rebalances the tree, multiple ciphertexts get mutated. Kerschbaum [39] proposed an improvement on this scheme that also hides the frequency of each plaintext (how many times a given value appears).

Furthermore, in order to improve the security of these schemes, Boneh et al. [10] proposed to generalize OPE to Order-Revealing Encryption (ORE). In ORE, 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 they still leak some information, and in general are more expensive to compute. Since these schemes leak some information, a number of recent works considered attacks on systems that may use these schemes [31, 32, 53, 26, 38, 12, 22, 44, 4, 65]. Most of these attacks assume the attacker possesses *auxiliary information* and no other protections are available.

OPE / ORE schemes can be used with almost no changes to the underlying database engine. To provide greater security, a number of more complex protocols for protecting data in outsourced databases have been proposed. These constructions are often interactive, rely on custom data structures and are optimized for certain tasks, such as range queries. Naturally, the more secure the protocol is, the larger performance overhead it incurs. The most secure of these — Oblivious RAM (ORAM) based protocol — provides strong, well-understood, cryptographic privacy guarantees with no information leakage.

Applications that can benefit from such schemes and protocols include cloud access security brokers (CASBs) and financial and banking applications. Indeed, a number of commercial CASBs including Skyhigh Networks [60] and CipherCloud [19] have been using some form of OPE or ORE schemes in their systems. In addition, financial institutions may be able to encrypt their data using the aforementioned schemes in order to provide another layer of security, assuming that the performance overhead is acceptable. For many of these applications the auxiliary information that is

Proceedings of the VLDB Endowment, Vol. 12, No. 8
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3324301.3324309

needed for the attacks mentioned above is either unavailable or difficult to get.

Currently, it is a very challenging task for users to choose an appropriate data privacy approach for their application, because the security and performance tradeoff is not well understood. Both security and performance of every approach need to be thoroughly evaluated. Characterizing security benefits of different approaches remains an open problem, unlikely to be solved in the immediate future. However, it is possible to evaluate the performance of each approach, so as to enable better-informed decisions about whether the improved performance of some schemes is worth the uncertainty about the security they achieve.

We emphasize that it is not trivial to evaluate the performance of these schemes. Many of the papers presenting the above approaches provide only a theoretical treatment and concentrate more on the security definitions and analysis and less on the performance. Some of these constructions have not been even implemented properly. Furthermore, even though the main target of these schemes and protocols 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 secure range query protocols in the context of 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.

The main contribution of this work is to 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. In the process, we present improvements for some of the schemes that make them more efficient and/or more secure. To make understanding of these schemes easier for the reader, we present the main ideas behind these constructions, discuss their security definitions and leakage profiles, and provide an analysis of implementation challenges for each one.

1.1 Related work

A number of OPE schemes have been proposed recently including [1, 55, 8, 9, 62, 40, 66, 35, 39, 36, 68, 67, 47, 23, 48]. Popa, Li, and Zeldovich [56] 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 [8] and FH-OPE [39]) that are the most representative and outperform other schemes.

In addition, there are a number of ORE schemes [10, 18, 46, 16, 15, 11, 28, 24] that have been proposed. We choose the most practical and most secure of them [18, 46, 16], 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 [58, 41, 5, 21]. We choose two of these protocols [58, 41] because they are based on order-preserving approaches and therefore have similar security models with ORE 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 [49] for an overview of other methods. In this paper we consider two of the protocols proposed in [21] that use Searchable Symmetric Encryption (SSE). We stress that the schemes and protocols discussed here should be used with care, and it is up to the practitioner how to use them given their security and performance profiles.

2. SECURITY PERSPECTIVE

Each scheme and protocol we analyze has its own security definition, which captures different leakage levels. 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 mostly consider the snapshot model, where the attacker can observe all the database contents at one time instant. Note that this excludes timing attacks such as measuring encryption time. All security definitions of the schemes and protocols 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 entire encrypted database and all its contents at some point in time.

Beyond the snapshot model, it is also possible to consider a stronger adversary who can track communication volume and data access patterns in real time. Approaches that help protect against such an attacker include ORAM for protection against access pattern leakage and differential privacy for protection against communication volume leakage. Although this model is not a primary target of this paper, our benchmark includes a protocol (Section 4.5.2) that is secure in this setting to show the cost of adding such protection.

We wanted to specifically comment on a work of Grubbs et al. [27], which demonstrates a series of attacks against OPE and ORE schemes. The attacks can be very successful, but they depend on certain prerequisites. First, all attacks assume the existence of a well-correlated auxiliary dataset. Second, the binomial attack, which works against a "perfectly secure frequency-hiding scheme", reliably recovers only high-frequency elements. Finally, the attacks are specifically devastating against encrypted strings (e.g. first and last names) as opposed to numerical data, and we also do not recommend using OPE / ORE for strings (see Section 2.1). One of the conclusions of our work is that security is negatively correlated with performance and it is up to a practitioner to trade off security and performance constraints.

2.1 A note on variable-length inputs

A generic OPE / ORE scheme accepts bit-strings of any length as inputs, and treats them as numbers or processes them bit-by-bit. We warn against supplying raw bytes of variable length (e.g. encoded strings) to OPE and ORE schemes, as such an approach will introduce both performance and security challenges.

From the performance standpoint, the complexity of OPE / ORE schemes usually depends on the input length at least linearly (see Table 1). 32-bit numbers already introduce a noticeable overhead for some (usually more secure) schemes, and supplying arbitrary-length inputs may worsen performance by at least an order of magnitude.

Security of such a construction will be minimal as most schemes leak some information about the magnitude of the difference, and longer inputs will naturally be treated as larger numbers. Thus, the difference between long and short

inputs will be apparent. We refer to the work of Grubbs et al. [27] as they have a practically supported discussion of security consequences of using OPE / ORE with arbitrary strings.

On the other hand, other protocols in our benchmark can usually handle variable-length inputs as long as they fit into a single block for the underlying block cipher.

3. OPE AND ORE SCHEMES

An Order-Revealing Encryption scheme is a triple of polynomial-time algorithms KGen, Enc and Cmp. KGen generates a key of parameterized length (the λ parameter). Enc takes a numerical input (as a bit string) and produces a ciphertext. Cmp takes two ciphertexts generated by the scheme and outputs whether the first plaintext was strictly less than the second. Note that being able to check this condition is enough to apply all other comparison operators ($<$, $\leq$, $=$, $\geq$, $>$). Also note that an ORE scheme does not include a decryption algorithm, because one can simply append a symmetric encryption of the plaintext to the produced ciphertext and use it for decryption.[1] An Order-Preserving Encryption (OPE) scheme is a particular case of an ORE scheme where ciphertexts are numerical and thus Cmp routine is trivial (the numerical order of ciphertexts is the same as underlying plaintexts). OPE may optionally include a decryption algorithm, since appending a symmetric ciphertext is no longer possible.

Both OPE and ORE schemes by definition allow to totally order the ciphertexts. This is their inherent leakage (by design) and all the OPE / ORE security definitions account for this and possibly additional leakage.

We proceed by describing and analyzing the OPE / ORE schemes we have benchmarked. 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.

From here, we will use the term ORE to refer to both OPE and ORE, unless explicitly stated otherwise. Each scheme has its own subsection where the first part is the construction overview followed by security discussion, and the second part is our theoretical and experimental analysis.

3.1 BCLO OPE

The OPE scheme by Boldyreva et al. [8] 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 [57]). The core principle of their construction is the natural connection between a random order-preserving function and the hypergeometric probability distribution.

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.

[1] *Given the secret key*, it is possible to decrypt a ciphertext by doing binary search on the plaintext domain: encrypting known values and comparing them against the target ciphertext, until the target plaintext is found. However, this would require $\mathcal{O}(\log|\mathcal{D}|)$ encryption and comparison operations.

All pseudo-random decisions are made by an internal PRG (TapeGen in [8]). This way not only the algorithm is deterministic, but also decryption is possible. The decryption procedure takes the same "path" of splitting domain and range, and when the domain size reaches one, the only value left is the original plaintext.

Security. This scheme is POPF-CCA secure [8], 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 [9] 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. Grubbs et al. [27] showed that this leakage allows to almost entirely decrypt the ciphertexts (given auxiliary data with a similar distribution) and encrypting strings (rather than numbers) with this scheme is especially dangerous (see Section 2.1).

Analysis and implementation challenges

Efficient sampling from the hypergeometric distribution is a challenge by itself. Authors suggest using a randomized yet exact (not approximate) Fortran algorithm by Kachitvichyanukul and Schmeiser [34]. 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 [64].

On average, encryption and decryption algorithms make n calls to HG, which in turn consumes entropy generated by the internal PRG. 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 5).

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 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. [18], 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 more significant bits. This value is given to a keyed PRF and the result is numerically added to the next less significant bit. This resulting list of n elements is the ciphertext.

The comparison routine traverses two lists in-order looking for the case when one value is greater than the other by exactly one, revealing location and value of the first differing bit. If no such index exists, the plaintexts are equal.

Security. A generic ORE security definition was introduced along with the scheme [18]. ORE leakage is more clearly quantified than in OPE. The definition says that the

Table 1: Primitive usage by OPE / ORE schemes. Ordered by security rank — most secure below. n is the input length in bits, d is a block size for Lewi-Wu scheme, λ is a PRF output size, N is a total data size, **HG** is a hyper-geometric distribution sampler, **PPH** is a property-preserving hash with h-bit outputs built with bilinear maps and **bolded** are weak points of the schemes. See Table 4 for practically derived values.

Scheme	Primitive usage		Ciphertext size,	Leakage
	Encryption	Comparison	or state size	(In addition to inherent total order)
BCLO [8]	$\boldsymbol{n}$ **HG**	none	$2n$	$\approx$ **Top half of the bits**
CLWW [18]	n PRF	none	$2n$	**Most-significant differing bit**
Lewi-Wu [46]	$\boldsymbol{2n/d}$ **PRP** $2\frac{n}{d}\left(2^d+1\right)$ PRF $\frac{n}{d}2^d$ Hash	$\frac{n}{2d}$ Hash	$\frac{n}{d}\left(\lambda+n+2^{d+1}\right)+\lambda$	Most-significant differing block
CLOZ [15]	n PRF n PPH 1 PRP	$\boldsymbol{n^2}$ **PPH**	$n \cdot h$	Equality pattern of most-significant differing bit
FH-OPE [39]	1 Traversal	3 Traversals	$\boldsymbol{3 \cdot n \cdot N}$	Insertion order

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. This scheme satisfies ORE security definition with the leakage $\mathcal{L}(\cdot)$ of the location and value of the first differing bit of every pair of plaintexts. Note that the most significant differing bit also leaks the approximate distance between two values.

Analysis and implementation challenges

On encryption the algorithm makes n calls to PRF and the comparison procedure does not use any cryptographic primitives. 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 [46] presented an improved version of the CLWW scheme [18] 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 approach is to split the plaintext into blocks of d bits. The ciphertext is computed block-wise. For the right side, the algorithm compares the given block value to all 2^d possible block values; each comparison result is added (modulo 3) to a PRF of the previous blocks. All 2^d comparison results go into the right ciphertext. The left side, which is shorter, is produced in such a way as to reveal the correct comparison result. This way the location of the differing bit inside the block is hidden, but the location of the first differing block is revealed.

Security. This scheme satisfies the ORE security definition introduced by Chenette et al. [18] with the leakage $\mathcal{L}(\cdot)$ of the location of the first differing *block*. This property allows a practitioner to set performance-security tradeoff by tuning the block size. Left / right framework is particularly useful in a B+ tree since it is possible to store only one (semantically secure) side of a ciphertext in the structure (see Section 4.1).

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. The ciphertext size is no longer negligible — $\frac{n}{d}\left(\lambda+n+2^{d+1}\right)+\lambda$, for λ being PRF output size.

The implementation details of this approach raise an interesting security question. Although the authors suggest using 3-rounds Feistel networks [59] 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 [29] — exponential in the input 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. Because the domain size is small (from 2^2 to 2^8 elements), we have decided to build a PRP by simply using the key as an index into the space of all possible permutations on the domain, where a permutation is obtained from the key via Knuth shuffle (this approach was mentioned in [52]). 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 generate a PRP table once for the whole block and then use this table when one needs 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.

Table 2: Performance of protocols. Ordered by security rank — most secure below. N is a total data size, B is an I/O page size, L is a POPE tree branching factor, r is the result size in records and **bolded** are weak points of the protocols. All values are in $\mathcal{O}$ notation. See Table 3 for practically derived values.

Protocol	I/O requests		Leakage	Communication (result excluded)	
	Construction	Query		Construction	Query
B+ tree with ORE	$\log_B \frac{N}{B}$	$\log_B \frac{N}{B} + \frac{r}{B}$	**Same as ORE**	1	1
Kerschbaum [41]	$\boldsymbol{\frac{N}{B}}$	$\log_2 \frac{N}{B} + \frac{r}{B}$	**Total order**	$\log_2 N$	$\log_2 N$
POPE [58] warm	1	$\log_L \frac{N}{B} + \frac{r}{B}$	**Partial order**	1	$\log_L N$
POPE [58] cold		$\boldsymbol{N/B}$	Fully hiding		$\boldsymbol{N}$
Logarithmic-BRC [20]	—	$\boldsymbol{r}$	Same as SSE	—	$\log_2 N$
ORAM	$\boldsymbol{\log^2 \frac{N}{B}}$	$\boldsymbol{\log_2 \frac{N}{B} \left(\log_B \frac{N}{B} + \frac{r}{B} \right)}$	Fully hiding (access pattern)	$\boldsymbol{\log^2 \frac{N}{B}}$	$\boldsymbol{\log^2 \frac{N}{B}}$

3.4 CLOZ ORE

Cash et al. [15] introduced a new ORE scheme that provably leaks less than any previous scheme. The idea is to use Chenette et al. [18] construction (see Section 3.2), but permute the list of PRF outputs. The original order of those outputs is not necessary, as one can simply find a pair that differs 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 and used in [15], allows one to expose a property (specifically $y \stackrel{?}{=} x + 1$) of two (numerical) elements such that nothing else is leaked. In particular, the outputs are randomized, so same element hashed twice will have different hashes. Please refer to the original paper [15] for formal correctness and security definitions.

Equipped with the PPH primitive, the algorithm "hashes" 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.

Security. The strong side of the scheme is its security. The scheme leaks $\mathcal{L}(\cdot)$ an *equality pattern* of the most-significant differing bits (satisfying Chenette et al. [18] definition). As defined in [15], 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. We do not know of any attacks against this construction (partially because no implementation exists yet, see next subsection), but it is inherently vulnerable to frequency attacks that apply to all frequency-revealing ORE schemes (see Section 2).

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, as the scheme makes n^2 calls to PPH Test.

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. The 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.

We have implemented this scheme in C++ using the PBC library [50] to empirically assess schemes's performance, and on our machine (see Section 5), a single comparison takes 1.9 seconds on average. Although we have produced the first (correct and secure) real implementation of this scheme in C++, it is infeasible to use it in the benchmark (it will take years to complete a single run). Therefore, for the purposes of our benchmark, we implemented a "fake" version of PPH — correct, but insecure, which does not use pairings. Consequently, in our analysis we did not benchmark the speed of the scheme, but measured all other data.

3.5 FH-OPE

Frequency-hiding OPE by Kerschbaum [39] is a stateful scheme that hides the frequency of the plaintexts, so the adversary is unable to construct a frequency histogram.

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, consider the range $[1, 128]$. Any plaintext that happens to arrive first (for example, 6), will be the root, and thus the ciphertext is 64. Then any plaintext smaller than the root, say 3, will become the left child of the root, and will produce the ciphertext 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, the traversal pseudo-randomly passes to the left or right child, 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.

Due to randomized ciphertexts, 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.

Authors have designed a number of heuristics to minimize the state size, however, these are mostly about compacting

the tree and the result depends highly on the tree content. In our analysis, we consider the worst case performance without the use of heuristics. In our experimental evaluation, however, we did implement compaction.

Security. 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 means ciphertexts should be 192-bit numbers that are not commonly supported. It is possible to operate over arbitrary-length numbers, but the performance overhead would be substantial. We did a quick micro-benchmark in C# and the overhead of using `BigInteger` is 15–20 times for basic arithmetic operations.

This scheme satisfies IND-FAOCPA definition (introduced along with the scheme [39]), meaning that it does not leak the equality or relative distance between the plaintexts. This definition has been criticized in [51], who claim that the definition is imprecise and propose an enhanced definition along with a small change to construction to satisfy this new definition. Both schemes leak the insertion order, because it affects the tree structure. We do not know of any attacks against this leakage, but it does not mean they cannot exist. Grubbs et al. [27] describe an attack against this scheme (binomial attack), but it applies to any perfectly secure (leaking only total order) frequency-hiding OPE.

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 rebalanced. This procedure will invalidate all ciphertexts already generated. 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 rebalancing will be caused by insertion of just 65 consecutive input elements 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 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 usage of the scheme.

4. SECURE RANGE QUERY PROTOCOLS

We proceed by describing and analyzing the range query protocols we have chosen. For the purpose of this paper, a secure range-query protocol is defined as a client-server communication involving construction and search stages. Communication occurs between a client, who owns some sensitive data, and an honest server, who securely stores it. In construction stage, a client sends the server the encrypted datapoints (index-value tuples) and the server stores them in some internal data structure. In search stage, a client asks the server for a range (usually specifying it with encrypted endpoints) and the server returns a set of encrypted records matching the query. Note that the server may interact with the client during both stages (e.g. ask the client to sort a small list of ciphertexts). Also note that we do not allow batch insertions as it would limit the use cases (e.g. client may require interactive one-by-one insertions).

The first protocol is a family of constructions where a data structure (B+ tree in this case) uses ORE schemes internally. Then, we present alternative solutions with varying performance and security profiles, not relying on ORE. Finally, we introduce two baseline solutions we will use in the benchmark — one that achieves the best performance and the other that achieves the maximal security.

4.1 Range query 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 scheme's Cmp 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.

In this paper, we have implemented a typical B+ tree [3] (with a proper deletion algorithm [33]) as a data structure.

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 (the longer the ciphertexts, the smaller the 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 amount of the protocol is relatively small as its insertions and queries require at most one round trip.

Security. The leakage of this protocol consists of leakage of the underlying ORE scheme plus whatever information about insertion order is available in the B+ tree. Please note that Lewi-Wu [46] ORE is particularly well-suited in this construction with its left / right framework, because only the semantically secure side of the ciphertext is stored in the structure. In this case, the ORE leakage becomes only the total order and the security of the protocol is comparable with other non-ORE constructions.

4.2 Kerschbaum-Tueno

Kerschbaum and Tueno [41] proposed a new data structure, which satisfies their own definitions of security (IND-CPA-DS) and efficiency (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.

Security. Authors prove that this construction is IND-CPA-DS secure (defined in the same paper [41]). The definition assumes an array data structure and therefore serves specifically this construction (as opposed to being generic). It provably hides the frequency due to semantic encryption and hides the location of the first element due to random rotations. Leakage-wise this construction is strictly better than B+ tree with ORE — they both leak total order, but [41] hides distance information and smallest / largest elements. Specifically, for all pairs of consecutive elements e_i and e_{i+1} it is revealed that $e_{i+1} \geq e_i$ except for one pair of smallest and largest elements in the set.

4.2.1 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.

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 [41] 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 batch uploads.

4.3 POPE

Roche et al. [58] presented a protocol, direct improvement over mOPE [56], which is especially 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, thus we do not concentrate on insertion analysis in this section.

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. On a high level, for each endpoint the buffers are cleared (content pushed down to leaves), and nodes in the paths are split. After that, answering a query means replying with all ciphertexts in all buffers between the two endpoint leaves.

The authors provide cost analysis of their construction. Search operations are expected to require $\mathcal{O}\left(\log_L n\right)$ rounds. It must be noted that the first queries will require many more rounds, since large buffers must be sorted.

Security. This construction satisfies the security definition of frequency-hiding partial order-preserving (FH-POP) protocol (introduced in the paper [58]). According to [58, Theorem 3], after n insertions and m queries with local storage of size L, where $mL \in o(n)$, the POPE scheme is frequency-hiding partial order-preserving with $\Omega\left(\frac{n^2}{mL\log_L n} - n\right)$ incomparable pairs of elements. Simply put, the construction leaks pairwise order of a *bounded* number of elements. Aside from this, the construction provably hides the frequency (i.e. equality) of the elements.

4.3.1 Analysis and implementation challenges

In our analysis we count each request-response communication as a round. This is different from [58] 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 the authors, if $L = n^\epsilon$ for $0 < \epsilon < 1$, then the amortized costs become $\mathcal{O}\left(1\right)$. 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.

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 $\mathcal{O}\left(N + r\right)$. 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 — $\mathcal{O}\left(L \log_L N + r\right)$. The average case is hard to compute; however, authors prove an upper bound on bandwidth after n insertions and m queries — $\mathcal{O}\left(mL \log_L n + n \log_L m + n \log_L(\lg n)\right)$.

POPE tree is not optimized for I/O the way B-tree is. Search complexity is hard 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 the buffers of arbitrary size.

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, [58] does not stress the fact that after alternating insertions and queries, it may happen that some intermediate buffers are not empty, thus returning buffers

Table 3: Simulation results for protocols' performance values

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	44	2	2	177	342
Kerschbaum [41]	**494**	7	40	86	671	1453
POPE [58] warm POPE [58] cold	1	300 **2175**	2	914 **497722**	32	43331 **9056644**
Logarithmic-BRC [20]	—	**40**	1	2	—	391
ORAM	**31**	**185**	**143**	**490**	**18254**	**62662**

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 where 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.

4.4 Logarithmic-BRC

Demertzis et al. [20] introduced a novel protocol called "Logarithmic-BRC" whose I/O complexity depends only on the result size, regardless of the database size. The core primitive for their construction is a Searchable Symmetric Encryption (SSE) scheme. An SSE scheme is a server-client protocol in which the server stores a specially encrypted keywords-to-documents map, and a client can query documents with keywords while the server learns neither keywords nor the documents. Note that the map stores short document identifiers instead of the actual documents, and we will use the term "documents" to mean "document identifiers" or "record IDs" in this section.

The construction treats record values as documents and index ranges as keywords so that records can be retrieved by the ranges that include them. Specifically, a client builds a virtual binary tree over the domain of indices and assigns each record a set of keywords, which is the path from that record to the root. This way, the root keyword is associated with all documents and the leaf keyword is associated with only one record.

Upon query, a client computes a cover — a set of nodes whose sub-trees cover the requested range. A client sends these keywords to the SSE server, which returns encrypted documents — result values. Of the several covering techniques suggested in the protocol [20] we have chosen the Best Range Cover (BRC), because it results in fewest nodes and does not return false-positives. Kiayias et al. [42] have proven that the worst-case number of nodes for domain of size N is $\mathcal{O}(\log N)$ and presented an efficient BRC algorithm.

Security. In a snapshot setting, this construction's security is that of the SSE. We have used [14] and [13] SSE schemes; their leakage in a snapshot setting is the database size and at most some initialization parameters. Thus, the security of these schemes is high enough to call them *fully hiding* in our setting. Additional access pattern leakage comes up during queries; exact implications of this leakage remain an open research problem but it is known that it can be harmful [38].

4.4.1 Analysis and implementation challenges

Communication involves a client sending at worst $\log_2 N$ keywords and server responding with the exact result.

For each keyword in the query set, server will query the SSE scheme, which will return r documents. Therefore, server's I/O complexity is that of SSE.

Demertzis et al. [20] have used [14] SSE scheme in their implementation, but we have found it slow it terms of I/O. Instead, we have implemented an improved scheme [13], which directly addresses I/O optimization.

Both SSE schemes' I/O complexity is linear with the result size r. [13] scheme makes at most one I/O per result document in the worst-case and there are extensions to significantly improve I/O complexity. We have implemented the `pack` extension, which packs documents in blocks to fit the I/O pages. We note that this extension can dramatically reduce the I/Os (see Section 5.3.3 and Figure 4a).

Logarithmic-BRC is very scalable as its performance does not depend on total data size and only degrades with the result size. Storage overhead, however, is significant. Each record is associated with the whole path in the binary tree — $\log_2 N$ nodes (keywords). The storage complexity is therefore $\mathcal{O}(N \log N)$, and the overhead is then a factor of $\log N$.

Updates, while addressed in the original protocol, are not very practical in this construction. Authors suggest using bulk-loading for updates, maintaining merge trees, and requiring the client to do a merge once in a while. The I/O complexity of such approach is unclear. In our implementation we perform the construction stage only in batch mode, and thus do not include it in the analysis. We also emphasize that the update routine was not implemented for evaluation in the original paper.

4.5 The two extremes

To put the aforementioned protocols in a context we introduce the baselines — an efficient and insecure construction we will refer to as *no encryption* and maximal security protocol we refer to as *ORAM*.

4.5.1 No encryption

This protocol is a regular B+ tree [3] without any ORE in it. It is the construction one can expect to see in almost any general-purpose database.

In terms of security it provides no guarantees — all data is in the clear. In terms of efficiency it is optimal. B+ tree

data structure is optimal in I/O usage, indices inside nodes are smallest possible (integers) and there is no overhead in comparing elements inside the nodes as opposed to working with ORE ciphertexts.

4.5.2 ORAM

Oblivious RAM (ORAM) is a construction that additionally to semantic security of a snapshot setting (see Section 2) provably hides the access pattern — a sequence of reads and writes to particular memory locations. With ORAM an adversary would not be able to recognize a series of accesses to the same location and will not differentiate reads versus writes. ORAM was introduced by Goldreich and Ostrovsky [25] who also proved its lower bound (strengthened in [45]) — logarithmic overhead per request. A number of efficient ORAM constructions were designed (see [17] for a good survey) and we use the state-of-the-art construction, PathORAM [61].

A generic ORAM server responds to read and write requests for a particular address. In our baseline protocol we store B+ tree nodes in ORAM. A client works with the tree as it normally would except each time it needs to access a node, it communicates with ORAM.

In terms of security this protocol is fully hiding in the snapshot model and provably hides the access pattern. We note that one can improve security even further by adding noise to the result obscuring communication volume. We also note that a practitioner can use a similar protocol with ORAM replaced with a trivial data store and have the tree nodes encrypted. It would be fully hiding in a snapshot setting, but we prefer the baseline that covers more than only the snapshot model.

In terms of performance this construction incurs some noticeable overhead. Regardless of specific ORAM being used, each access incurs at least logarithmic overhead according to lower bounds [25]. Combined with logarithmic complexity of the B+ tree itself, the complexity, both I/O and communication, is $\mathcal{O}\left(\log^2 N\right)$. We found that PathORAM has good I/O performance, as its internal tree structure translates into good cache affinity. Unlike in other protocols in our benchmark, ORAM client does most of the computational work. While the server only makes I/O requests, the client handles encryption, shuffling, and request logic.

We present this protocol as a baseline solution in terms of security over efficiency. We have not implemented standalone PathORAM, but rather a simulator which correctly reports I/O, communication and primitive usage. Surprisingly, we found that ORAM protocol's overhead, although higher than in ORE-based protocols, is in-line with the most secure protocols in our benchmark.

5. EVALUATION

All experiments were conducted on a single machine. We use macOS 10.14.2 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.

Interactive website

Additionally to making our source code, compiled binaries and Docker images available, we want to let researchers interactively run small-sized simulations. We host a website [7] where one can select a protocol (including baselines, CLOZ and both SSE schemes), cache size and policy and I/O page parameter; supply one's own data and query sets, and run the simulations. Simulations are run one at a time and usually complete within seconds. The user is then able to view the result — tables, plots, values and raw JSON, which we used to build plots for this paper. Input size on the website is limited for practical purposes and users are encouraged to run arbitrary-size simulations using our binaries or Docker images.

Table 4: Simulation results for ORE primitive usage

Scheme	Encryption	Comparison	Size (bits)
BCLO [8]	**41 HG**	none	64
CLWW [18]	32 PRF	none	64
Lewi-Wu [46]	**32 PRP** 160 PRF 64 Hash	9 Hash	2816
CLOZ [15]	32 PRF 32 PPH 1 PRP	**1046 PPH**	4096
FH-OPE [39]	1 Traversal	1 Traversal	**86842**

5.1 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 note that neither primitives, nor schemes are production-ready; however, we believe they can be used in research projects and prototypes. We also emphasize that the B+ tree implementation we are using, although our own with instrumentation in it, is not custom in any way, but rather standard as defined in the original paper [3] with deletion algorithm by [33].

This software project (22K lines of code, third of which are tests) 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 project as standalone .NET Core (nuGet) packages, and we host a webserver where users can run simulations for small inputs (see previous subsection).

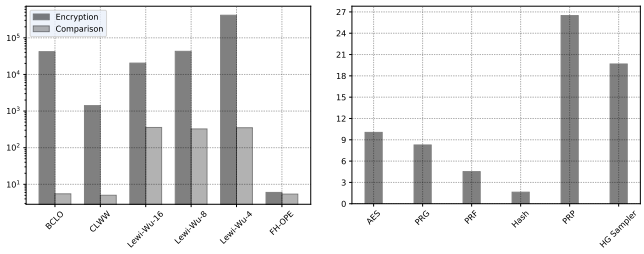
5.1.1 Primitives

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` CPU instruction. In our project all key sizes are 128 bits, as is AES block size.

We implemented AES-based PRG, which uses AES in CTR mode and caches unused entropy (as suggested in [30]). For PRF, since we need only 128-bit inputs and outputs, we used one application of AES [37, Proposition 3.27]. For symmetric encryption we use AES with a random initialization vector in CBC mode [37, Section 3.6.2]. For hash we use default .NET Core SHA2 implementation. For PRP, we implemented unbalanced Feistel networks [59] for large inputs and Knuth shuffle [43] for small inputs. Please see the README of project's repository [6] for low-level details.

5.1.2 Schemes and protocols

We implemented schemes and protocols precisely as in the original papers. When we found problems or improvements, we described them 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 the *no encryption* 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: Benchmarks of the schemes and primitives

5.1.3 Simulations

We have four types of simulations.

Protocol simulation runs both protocol stages — 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 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 [54] to ensure that the reported time is accurate. This tool handles issues 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 four schemes (excluding CLOZ) and their variants.

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

5.2 Setup

For our simulations, we have used three datasets. Two synthetic distributions, that are uniform (range is third of data size) and normal (standard deviation is 0.1 of data size). The real dataset is California public employees salaries ("total pay and benefits" column) [63]. Synthetic datasets and subsets of the real dataset are generated pseudo-randomly. Queries are generated uniformly at random with a range as a percentage of data size.

5.3 Results

5.3.1 Primitive usage by schemes

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, the 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. Please note that the simulated values are consistent with the theoretical calculations.

5.3.2 Benchmarks of schemes and primitives

Using the Benchmark.NET tool [54], we have accurately tracked the performance of the schemes and primitives running of different parameters (see Figure 1). ORE schemes benchmark setup is the same as in primitive usage simulation 5.3.1. Primitives were given randomly generated byte inputs and keys of different sizes (e.g. PRP of 2 to 32 bits). Benchmark.NET decides how many times to run the routine to get statistically sound results. For example, large variance results in more runs. To improve the accuracy, each run is compiled in release mode as a separate project and runs in a separate process with the highest priority.

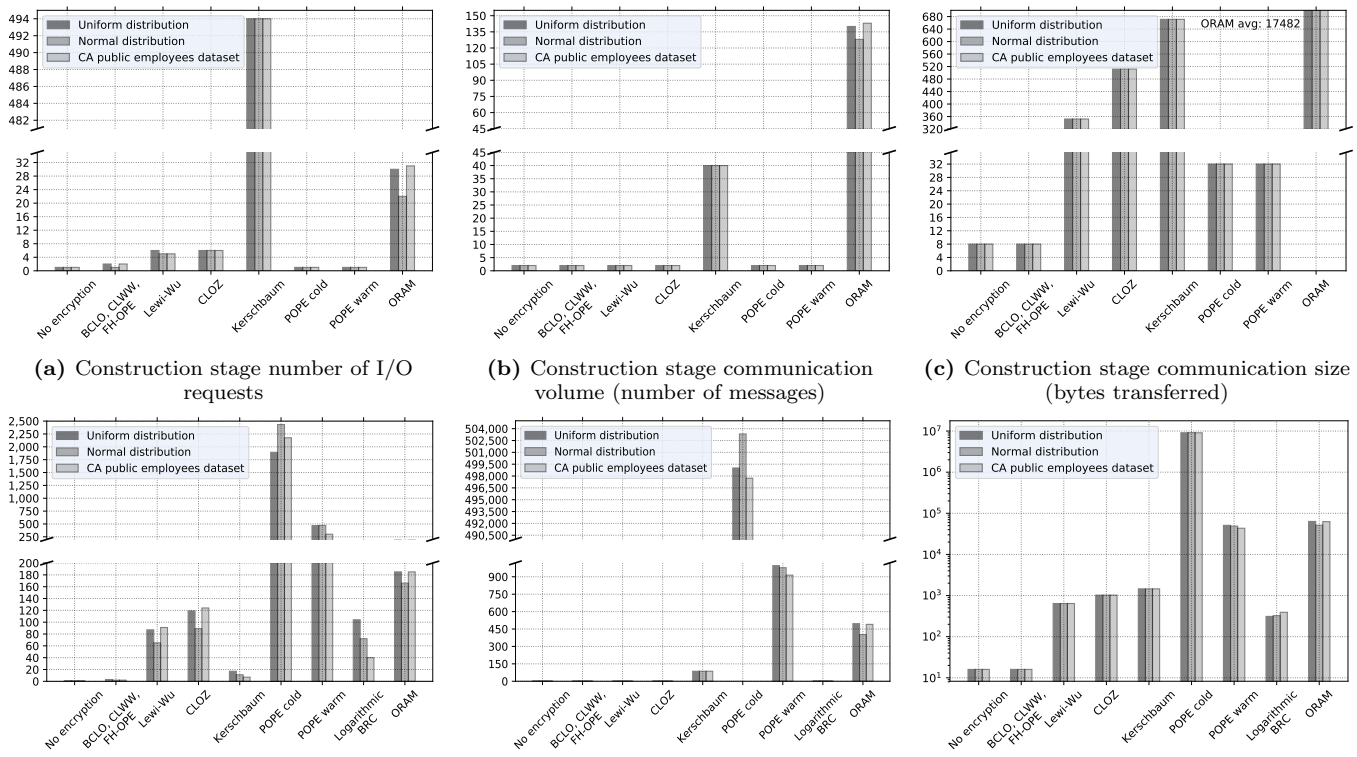
Please note the logarithmic scale of the schemes' performances. FH-OPE is fast since it does not perform CPU-heavy operations and works in main memory. Lewi-Wu performance degrades exponentially with the increase of block size mainly due to exponential number of PRF executions and the performance of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

In the 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 the initialization vector generation needed for symmetric encryption. PRP is implemented as a Knuth shuffle [43] and its complexity is exponential in the input bit length. Input size of 2 bits is shown on Figure 1. PRG does not discard the entropy generated by AES cycle, so one AES cycle can supply four 32-bit integers. PRP generates the permutation table once and does not regenerate it if the same key and number of bits are supplied.

5.3.3 Protocols

In this experiment we have run each protocol with each of the three datasets. Dataset sizes are 247000 (bounded by California Employees dataset size) and the number of queries is 1000. Queries are generated uniformly at random with a fixed range — 0.5% of data size. 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. The values we are measuring are the number of I/O operations, communication volume, and size for both construction and query stages.

See Table 3 for the snapshot for particular distribution (CA employees). Figure 2 shows all values we tracked for



(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

all protocols and distributions. Values for ORE based protocols are averaged. Being "cold" in our simulations means executing the first query and being "warm" means the first query has been previously executed. This difference makes sense only for POPE as its first query incurs disproportionately large overhead by design.

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.

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.

Logarithmic-BRC does not support interactive insertions and thus its construction stage is not benchmarked. Otherwise it is the most performant of all non-ORE protocols. Note, however, that its performance depends on the result size, not data size.

As expected, ORAM performs worse than the ORE-based protocols, but its performance is in-line with the non-ORE protocols. It may seem that ORAM does especially bad in construction communication (Subfigures 2e, 2f), but it is only because POPE has a shortcut in construction. This "debt" is being payed off during queries (Subfigure 2f).

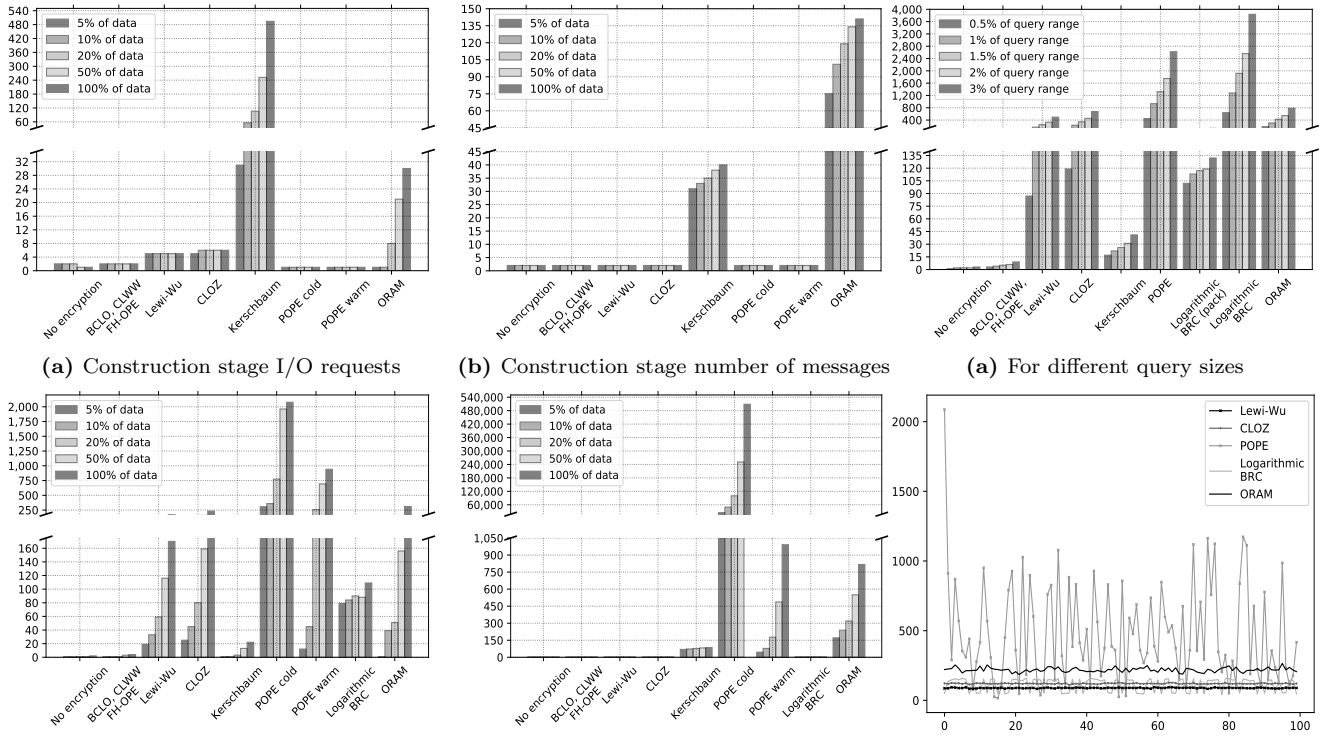
Note that the values do not vary a lot among different data distributions except for I/O requests. I/O performance depends on the result size for queries, and is therefore more sensitive to data distribution.

Also note that using an ORE scheme with relatively small ciphertext 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, except for Logarithmic-BRC, for which the relation is linear. Note that Logarithmic-BRC with optimally configured `pack` extension shows almost no growth. This is because for large ranges BRC will return the higher nodes (keywords matching many documents), which are optimally packed in I/O pages. As query range doubles, higher nodes are involved increasing the chance that requested keywords have their documents packed.

Figure 3 shows Table 2 asymptotic values. The simulation was run for uniform dataset of 247000 records (hundred percent), 1000 queries, 0.5% query range and 128 blocks cache size. Kerschbaum construction I/Os and cold POPE query values grow linearly with inputs, while the other protocols grow logarithmically, square-logarithmically, or do not grow.

Figure 4b shows how the performance of protocols fluctuates as queries are processed. Note that POPE and Logarithmic-BRC fluctuate the most (which is, in general, undesirable), and POPE is the only protocol where cold versus warm makes a difference.



(a) Construction stage I/O requests

(b) Construction stage number of messages

(a) For different query sizes

(c) Queries stage I/O requests

(d) Queries stage number of messages

Figure 3: Protocol scalability

(b) Over time (queries).

Figure 4: Number of I/O requests

6. REMARKS AND CONCLUSION

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. When it comes to 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.

ORE-based B+ tree protocol is provably I/O optimal and can potentially be extended by using another data structure with ORE. Its security/performance trade off is tunable by choosing and parametrizing the underlying ORE scheme. Each scheme we considered has its own unique advantages and drawbacks. BCLO [8] is the least secure scheme in the benchmark, but is stateless and produces numerical ciphertexts, so it may be used in the databases without any modifications. Frequency-hiding OPE [39] also has this property, hides the frequency of the ciphertexts, but is stateful and requires uniform input. Lewi-Wu [46] is easily customizable in terms of tuning performance to security ratio, and it offers the security benefits of left / right framework — particularly useful for B+ tree. CLWW [18] provides weaker security guarantees but is the fastest scheme in the benchmark.

Kerschbaum protocol [41] 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.

POPE [58] 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. It will be more performant for the systems where there are a lot more insertions than queries. We would also recommend to "warm up" the structure to avoid a substantial delay upon the first query.

Logarithmic-BRC is a perfect choice for huge datasets where query result size is limited. It is the only protocol with substantial space overhead, but it offers scalability and perfect (in a snapshot setting) security, and a carefully chosen and configured SSE scheme ensures that I/O grows slowly as a function of result size.

ORAM has shown the most interesting result. Its performance is not only adequate, but also in-line with the other even less secure protocols. With this empirical result, we expect more interest in ORAM research, possibly discovering tighter bounds, faster constructions and efficient ways to use the schemes. The performance of ORAM gives an upper bound on the acceptable performance level of less secure (access pattern revealing) protocols, as practitioners will choose ORAM over both less secure and less performant solutions.

We found our framework to be a powerful tool for analyzing the protocols, and we hope developers of new protocols will contribute implementations and evaluate them.

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. [15]).

7. ACKNOWLEDGMENTS

We thank Adam O'Neill, George Kellaris, Lorenzo Orecchia, Ioannis Demertzis, Oleksandr Narykov and Daria Bogatova for helpful discussions. We also thank the anonymous reviewers for useful suggestions. 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.

8. 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, pages 563–574. ACM, 2004.

[2] L. Arge. The buffer tree: a technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, Sept. 2003.

[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, pages 107–141. ACM, 1970.

[4] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. The tao of inference in privacy-protected databases. *PVLDB*, 11(11):1715–1728, 2018.

[5] T. Boelter, R. Poddar, and R. A. Popa. A secure one-roundtrip index for range queries. *IACR Cryptology ePrint Archive*, 2016.

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

[7] D. Bogatov. Interactive secure range queries simulations, 2019. `https://ore.dbogatov.org/`.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241. Springer Berlin Heidelberg, 2009.

[9] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *Advances in Cryptology – CRYPTO 2011*, pages 578–595. Springer Berlin Heidelberg, 2011.

[10] 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*, pages 563–594. Springer Berlin Heidelberg, 2015.

[11] M. Bun and M. Zhandry. Order-revealing encryption and the hardness of private learning. In *Theory of Cryptography*, pages 176–206. Springer Berlin Heidelberg, 2016.

[12] 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*, pages 668–679. ACM, 2015.

[13] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, and M. Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *In Network and Distributed System Security Symposium (NDSS '14*, 2014.

[14] 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 pages 353–373. Springer Berlin Heidelberg, 2013.

[15] 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.

[16] 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.

[17] Z. Chang, D. Xie, and F. Li. Oblivious RAM: a dissection and experimental evaluation. *PVLDB*, 9(12):1113–1124, 2016.

[18] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption*, pages 474–493. Springer Berlin Heidelberg, 2016.

[19] Ciphercloud. `https://www.ciphercloud.com/`.

[20] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In pages 185–198. ACM, 2016.

[21] 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*, pages 185–198, 2016.

[22] 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*, pages 1155–1166. ACM, 2016.

[23] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes. A structure preserving database encryption scheme. In *Secure Data Management*, pages 28–40. Springer Berlin Heidelberg, 2004.

[24] J. Eom, D. H. Lee, and K. Lee. Multi-client order-revealing encryption. *IEEE Access*:45458–45472, 2018.

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

[26] 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*, pages 162–168. ACM, 2017.

[27] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. Leakage-abuse attacks against order-revealing encryption. *2017 IEEE Symposium on Security and Privacy (SP)*:655–672, 2016.

[28] H. Haagh, Y. Ji, C. Li, C. Orlandi, and Y. Song. Revealing encryption for partial ordering. In *Cryptography and Coding*, pages 3–22. Springer International Publishing, 2017.

[29] V. T. Hoang and P. Rogaway. On generalized Feistel networks. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, pages 613–630. Springer-Verlag, 2010.

[30] R. Housley. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). RFC 3686, Jan. 2004. URL: `https://tools.ietf.org/html/rfc3686`.

[31] 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.

[32] 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*, pages 235–246, 2014.

[33] J. Jannink. Implementing deletion in B+-trees. *SIGMOD Rec.*, 24(1):33–38, Mar. 1995.

[34] V. Kachitvichyanukul and B. Schmeiser. ALGORITHM 668: H2PEC: sampling from the hypergeometric distribution. 14:397–398, Dec. 1988.

[35] 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:2520–2533, 2010.

[36] 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*, pages 338–353. Springer Berlin Heidelberg, 2013.

[37] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, second edition, 2014. ISBN: 9781466570269.

[38] 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*, pages 1329–1340. ACM, 2016.

[39] F. Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667. ACM, 2015.

[40] 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*, pages 275–286. ACM, 2014.

[41] F. Kerschbaum and A. Tueno. An efficiently searchable encrypted data structure for range queries. *arXiv preprint arXiv:1709.09314*, 2017.

[42] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 669–684. ACM, 2013.

[43] D. E. Knuth. *Seminumerical algorithms*, volume 2. Addison-Wesley, 3rd edition, 2016, pages 145–146.

[44] 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)*, pages 297–314, 2018.

[45] K. G. Larsen and J. B. Nielsen. Yes, there is an oblivious RAM lower bound! In *Advances in Cryptology - CRYPTO 2018*, pages 523–542, 2018.

[46] K. Lewi and D. J. Wu. Order-revealing encryption: new constructions, applications, and lower bounds. In pages 1167–1178. ACM, 2016.

[47] 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*, pages 502–509, 2012.

[48] D. Liu and S. Wang. Nonlinear order preserving index for encrypted database query in service cloud environments. *Concurrency and Computation: Practice and Experience*:1967–1984.

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

[50] B. Lynn. Pairings-based crypto (PBC). 2018. URL: `https://crypto.stanford.edu/pbc/` (visited on 08/15/2018).

[51] 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.

[52] B. Morris, P. Rogaway, and T. Stegers. How to encipher messages on a small domain. 2009. (Visited on 02/12/2019). `https://www.iacr.org/conferences/crypto2009/slides/p286-thorp.pdf`.

[53] 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*, pages 644–655. ACM, 2015.

[54] .NET Foundation. Benchmark.NET. `https://github.com/dotnet/BenchmarkDotNet`, 2018.

[55] 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*, pages 133–146, 2003.

[56] R. Popa, F. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy*, pages 463–477, 2013.

[57] 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, pages 85–100. ACM, 2011.

[58] 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*, pages 1131–1142. ACM, 2016.

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

[60] Skyhigh networks. https://www.skyhighnetworks.com/.

[61] E. Stefanov, M. v. Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, pages 299–310. ACM, 2013.

[62] I. Teranishi, M. Yung, and T. Malkin. Order-preserving encryption secure beyond one-wayness. In *Advances in Cryptology – ASIACRYPT 2014*, pages 42–61. Springer Berlin Heidelberg, 2014.

[63] Transparent California. 2017 salaries for State of California, 2017. https://transparentcalifornia.com/salaries/2017/state-of-california/.

[64] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, Sept. 1977.

[65] X. Wang and Y. Zhao. Order-revealing encryption: file-injection attack and forward security. In *Computer Security*, pages 101–121. Springer International Publishing, 2018.

[66] 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*, pages 89–100. ACM, 2013.

[67] L. Xiao and I.-l. Yen. A note for the ideal order-preserving encryption object and generalized order-preserving encryption.

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