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

Dmytro Bogatov George Kollios Leonid Reyzin

Computer Science, Boston University {dmytro,gkollios,reyzin}@bu.edu

BOSTON UNIVERSITY

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. More protocols have been introduced that are based on Searchable Symmetric Encryption (SSE), Oblivious RAM (ORAM) or custom encrypted data structures. We present the first comprehensive comparison among a number of important secure range query protocols using a framework that we developed. 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.

Order-Preserving Encryption

An Order-*Preserving* Encryption (OPE) scheme is a tuple of polynomial-time algorithms $\textsc{Setup}, \textsc{Encrypt}, \textsc{Decrypt}$ defined over a well-ordered domain $\mathcal{D}$ with the following properties:

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

OPE scheme is correct when $\textsc{Decrypt}$ is correct

$$\Pr[\textsc{Decrypt}(\textsc{sk}, \textsc{Encrypt}(\textsc{sk}, m)) = m] = 1 - \text{negl}(\lambda)$$

and when the order of ciphertexts is preserved

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

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

Order-Revealing Encryption

An Order-*Revealing* Encryption (ORE) scheme is different from OPE in the format of ciphertexts and public keyless $\textsc{Compare}$ routine

- $\textsc{Encrypt}(\textsc{sk}, m) \rightarrow \textsc{ct}$. On input the secret key $\textsc{sk}$ and a message $m \in \mathcal{D}$, the possibly randomized encryption algorithm $\textsc{Encrypt}$ outputs a *non-numeric* ciphertext $\textsc{ct}$.
- $\textsc{Compare}(\textsc{ct}_1, \textsc{ct}_2) \rightarrow b$. On input two ciphertexts $\textsc{ct}_1, \textsc{ct}_2$, the comparison algorithm $\textsc{Compare}$ outputs a bit $b \in \{0, 1\}$.

ORE is correct when $\textsc{Compare}$ is correct

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

for $\mathbf{1}(\cdot)$ being 1 when the condition is true.
To decrypt, having a secret key one can either do a binary search over domain, or attach a symmetric encryption to the ciphertext and use it for decryption.

ORE-based Secure Range Query protocol

For an ORE scheme $\text{ORE} = (\textsc{Setup}, \textsc{Encrypt}, \textsc{Compare})$, symmetric encryption scheme $\text{S} = (\textsc{Setup}, \textsc{Encrypt}, \textsc{Decrypt})$ and data structure $\text{DS} = (\textsc{Insert}, \textsc{Search})$, the protocol Π is defined as follows.

- $\Pi.\textsc{Setup}$:
  - Generate $K = \text{ORE}.\textsc{Setup}(\cdot)$ and $k = \text{S}.\textsc{Setup}(\cdot)$
  - Initialize DS
- $\Pi.\textsc{Insert}(i, v)$:
  - Client encrypts a data point index $\boldsymbol{i} = \text{ORE}.\textsc{Encrypt}(K, i)$
  - Client encrypts a data point value $\boldsymbol{v} = \text{S}.\textsc{Encrypt}(k, v)$
  - Client sends $\{\boldsymbol{i}, \boldsymbol{v}\}$ to the server
  - Server stores the encrypted data point $\text{DS}.\textsc{Insert}(\{\boldsymbol{i}, \boldsymbol{v}\})$
- $\Pi.\textsc{Search}(l, r)$:
  - Client encrypts the endpoints $\{\boldsymbol{l}, \boldsymbol{r}\} = \text{ORE}.\textsc{Encrypt}(K, \{l, r\})$
  - Client sends $\{\boldsymbol{l}, \boldsymbol{r}\}$ to the server
  - Server answers the query $\boldsymbol{r} = \text{DS}.\textsc{Search}(\{\boldsymbol{l}, \boldsymbol{r}\})$
  - Server sends the result $\boldsymbol{r}$ back to the client
  - Client decrypts all elements $r = \text{S}.\textsc{Decrypt}(k, \boldsymbol{r})$

ORE schemes primitive usage

Scheme	Primitive usage		Greater of cipher	Security	
	Encryption	Comparison	or state size	Definition	Leakage
BCLO [1]	$\boldsymbol{n}$ **HG**	none	$2n$	POPF-CCA	**Half of the bits**
CLWW [2]	n PRF	none	$2n$	ORE with leakage	**Most significant differing bit**
Lewi-Wu [3]	${}^{2n}/_{d}$ **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	Most significant differing block
FH-OPE [4]	1 Traversal	3 Traversals	$\boldsymbol{3 \cdot n \cdot N}$	IND-FAOCPA	Insertion order
CLOZ [5]	n PRF n PPH 1 PRP	$\boldsymbol{n^2}$ **PPH**	$n \cdot h$	ORE with leakage	Most significant differing bit equality pattern

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 hyper-geometric distribution sampler, **PPH** is property-preserving hash with h-bit outputs built with bilinear maps and **bolded** are weak points of the schemes

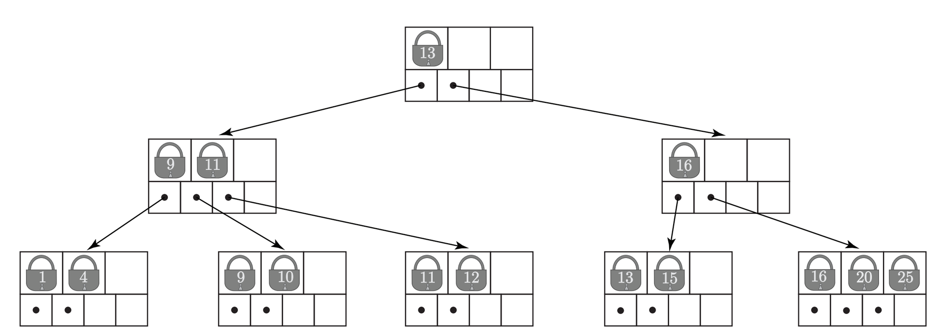
Protocols performance values

Protocol	I/O requests		Security	Communication (result excluded)		
	Construction	Query		Construction	Query volume	Query size
B+ tree with ORE	$\log_B \frac{N}{B}$	$\log_B \frac{N}{B} + \frac{r}{B}$	**Same as ORE**	1	1	
Kerschbaum-Tueno [6]	$\boldsymbol{\frac{N}{B}}$	$\log_2 \frac{N}{B} + \frac{r}{B}$	IND-CPA-DS	$\log_2 N$	$\log_2 N$	
POPE [7] cold	1	$\boldsymbol{{}^{N}/_{B}}$	FH-OP	1	$\boldsymbol{N}$	
POPE [7] warm		$\log_L \frac{N}{B} + \frac{r}{B}$	**FH-OP Pratial**		$\log_L N$	$L \log_L N$
Logarithmic-BRC [8]	—	$\boldsymbol{r}$	Same as SSE	—	1	
ORAM with B+ tree	$\boldsymbol{\log_2 N \log_B N}$	$\boldsymbol{\log_2 N \left(\log_B N + \frac{r}{B}\right)}$	Fully hiding	$\log_2 \frac{N}{B}$	$\log_2 \frac{N}{B}$	

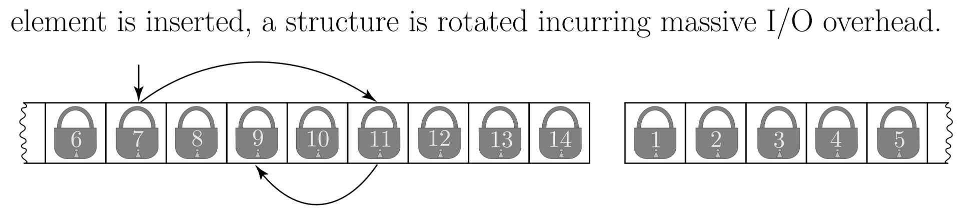
N is a total data size, B is an I/O page size, L is a POPE tree branching factor, r is the result size in records and **bolded** are weak points of the protocols. All values are in $\mathcal{O}$ notation.

Protocols descriptions

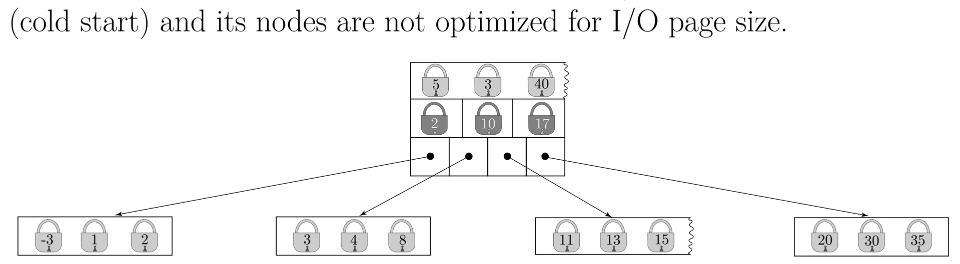
ORE with B+ tree. In essence, this construction is a regular B+ tree with ORE ciphertexts as indices and $\textsc{Compare}$ routine built in. This construction's strength is its I/O optimization — $\mathcal{O}\left(\log_B \left({}^{N}/_{B}\right) + {}^{r}/_{B}\right)$.



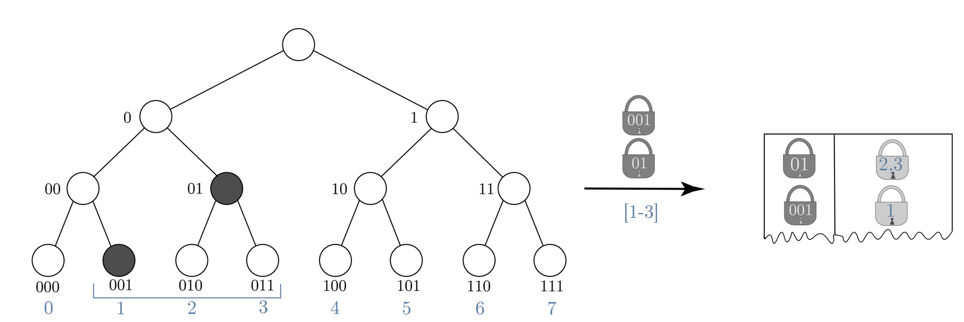
Kerschbaum-Tueno [6]. This construction maintains an array of symmetrically encrypted indices on the server, in-order, but with applied modular rotation. When inserting or searching, server interactively traverses the structure like a binary tree asking client for a direction to go. Each time a new element is inserted, a structure is rotated incurring massive I/O overhead.



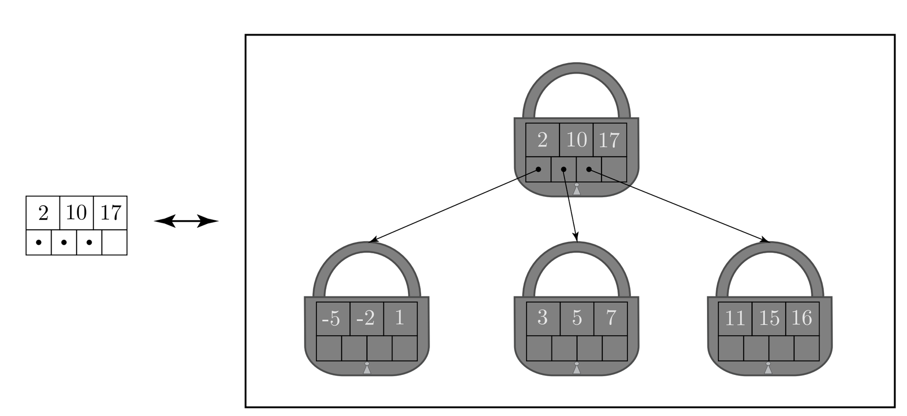
POPE [7]. This construction is based on buffer trees to support fast insertion and lazy sorting. All indices are symmetrically encrypted and server asks the client to sort a list of ciphertexts thus structuring the tree. New elements are always inserted in the root's buffer, and during queries are pushed down to the leaves. This construction incurs massive I/O overhead on first query (cold start) and its nodes are not optimized for I/O page size.



Logarithmic BRC usnig SSE [8]. This construction builds a virtual binary tree over the input domain and assigns each input element keywords from the path from this element to the root. This keyword-index mapping is encrypted with SSE. On query, a client finds the minimal number of nodes that cover the range and queries the SSE server for these keywords. This construction's I/O performance is that of SSE — linear in result size.

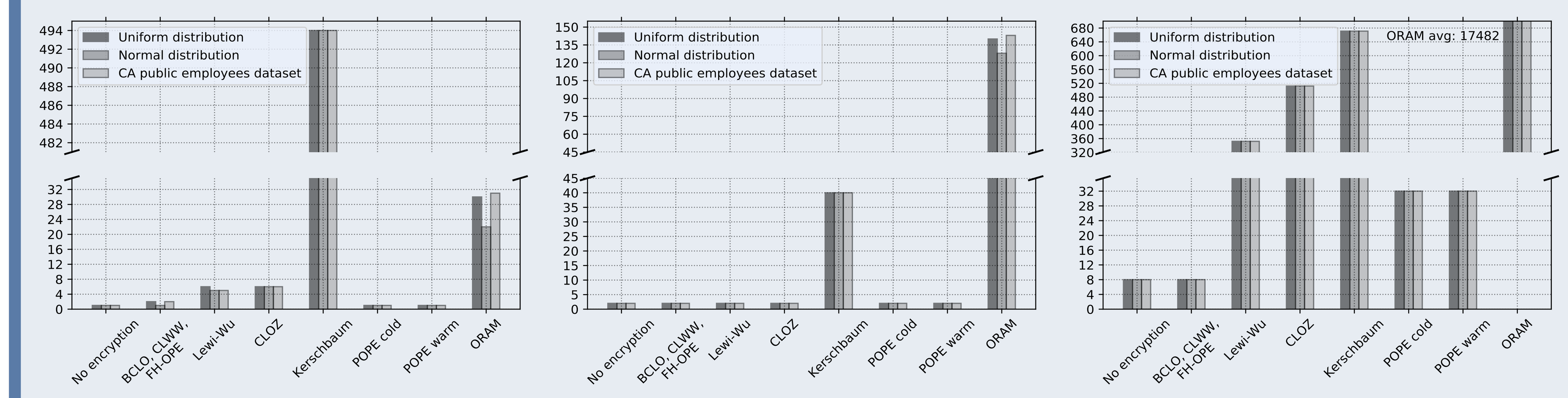


B+ tree in ORAM. A client operates on a regular B+ tree, but each time a node is accessed it is read or written to the ORAM server. This effectively squares the I/O usage since for each node in the logarithmic path, there is a logarithmic overhead in ORAM. This construction, however, is the most secure — additionally to data it hides the access pattern.



Our results

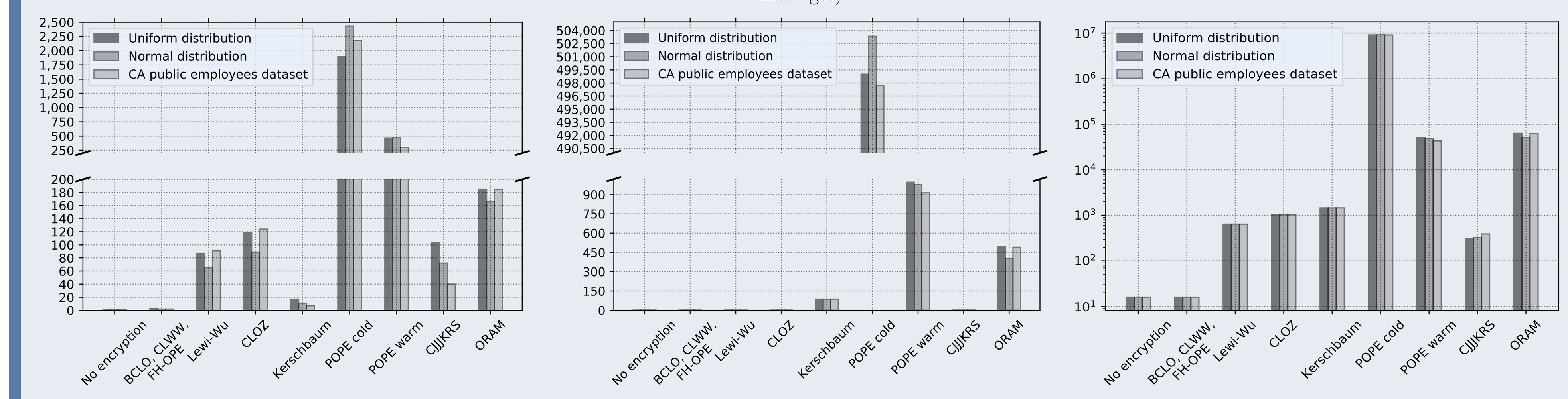
Performance values for different data distributions



Construction stage number of I/O requests

Construction stage communication volume (number of messages)

Construction stage communication size (bytes transferred)



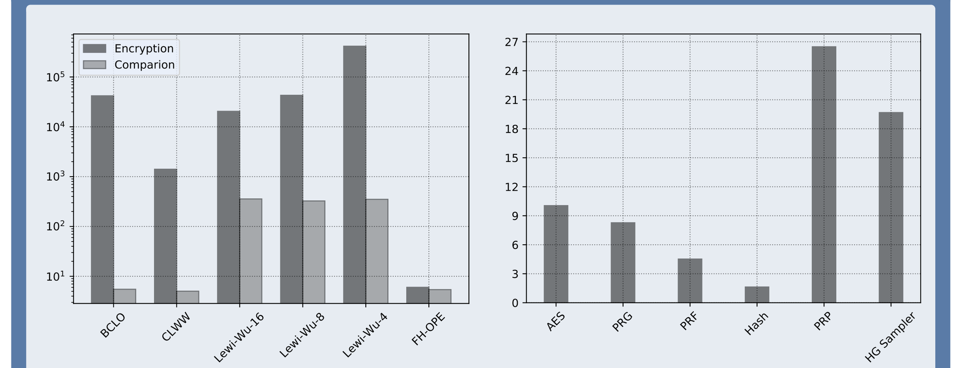
Query stage number of I/O requests

Query stage communication volume (number of messages)

Query stage communication size (bytes transferred, logarithmic scale)

Schemes and primitives

Schemes and primitives benchmarks



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

Primitives benchmark (time in microseconds)

Benchmark methodology

- We have implemented almost all primitives, schemes, data structures and protocols ourselves. The code is written in C# and runs on .NET Core 2.2. The code is tested with over a thousand unit tests and the coverage is above 97%.
- Almost all primitives are based on AES.
- Faithful modeling of I/O using different caching policies — LRU, LFU and FIFO.
- Time for primitive and schemes benchmark is measured with Benchmark.NET. I/O requests, primitive usage and communication are measured by firing events from within execution and carefully catching them.
- Synthetic data sets are generated pseudo-randomly and real one is California public employees salaries.
- All computations except rare symmetric encryptions are deterministic given global seed. All experiments can be reproduced exactly.
- Our tool is capable of generating massive detailed fine-grained reports for protocol executions. We present only the most interesting tiny fraction of the experimental results.

Conclusions

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.

ORE with B+ tree is tuneable in security / performance tradeoff. Index data structure and underlying ORE scheme can be replaced independently.

Kerschbaum protocol [6] offers semantically secure ciphertexts, hides the location of the smallest and largest of them, has a simple implementation, but requires batch insertions.

POPE [7] offers a "deferred" B+ tree implementation and remains more secure for the small number of queries. Incurs massive I/O hit on first queries and is not optimized for I/O like B+ tree.

Logarithmic BRC usnig SSE [8] relies on underlying SSE scheme's security and offers a different tradeoff — performance as a function of the result size. It is also not optimized for non-batch insertions.

ORAM offers the strongest security. Performance hit, although heavy, is comparable with other protocols. ORAM server acts as a generic secure key-value store.

Acknowledgements

We would like to thank Adam O'Neill, George Kellaris and Ioannis Demertzis for helpful discussions and Daria Bogatova for help with protocol diagrams. We also thank ESSA2 workshop organizers and attendees for in-depth discussions of their constructions. 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] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241. Springer Berlin Heidelberg, 2009.

[2] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption*, pages 474–493. Springer Berlin Heidelberg, 2016.

[3] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. pages 1167–1178. ACM, 2016.

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

[5] David Cash, Feng-Hao Liu, Adam O'Neill, Mark Zhandry, and Cong Zhang. Parameter-hiding order revealing encryption. In *Advances in Cryptology – ASIACRYPT 2018*, 2018.

[6] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. *arXiv preprint arXiv:1709.09314*, 2017.

[7] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. Pope: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1142. ACM, 2016.

[8] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. pages 185–198. ACM, 2016.

Published at PVLDB, 12(8), 2019
dbogatov.org/assets/docs/ore-benchmark.pdf