Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds*

Kevin Lewi
Stanford University
klewi@cs.stanford.edu

David J. Wu
Stanford University
dwu4@cs.stanford.edu

ABSTRACT

In the last few years, there has been significant interest in developing methods to search over encrypted data. In the case of range queries, a simple solution is to encrypt the contents of the database using an order-preserving encryption (OPE) scheme (i.e., an encryption scheme that supports comparisons over encrypted values). However, Naveed et al. (CCS 2015) recently showed that OPE-encrypted databases are extremely vulnerable to "inference attacks."

In this work, we consider a related primitive called order-revealing encryption (ORE), which is a generalization of OPE that allows for stronger security. We begin by constructing a new ORE scheme for small message spaces which achieves the "best-possible" notion of security for ORE. Next, we introduce a "domain-extension" technique and apply it to our small-message-space ORE. While our domain-extension technique does incur a loss in security, the resulting ORE scheme we obtain is more secure than all existing (stateless and non-interactive) OPE and ORE schemes which are practical. All of our constructions rely only on symmetric primitives. As part of our analysis, we also give a tight lower bound for OPE and show that no efficient OPE scheme can satisfy best-possible security if the message space contains just three messages. Thus, achieving strong notions of security for even small message spaces requires moving beyond OPE.

Finally, we examine the properties of our new ORE scheme and show how to use it to construct an efficient range query protocol that is robust against the inference attacks of Naveed et al. We also give a full implementation of our new ORE scheme, and show that not only is our scheme more secure than existing OPE schemes, it is also faster: encrypting a 32-bit integer requires just 55 microseconds, which is more than 65 times faster than existing OPE schemes.

*The full version of this paper [43] with complete proofs is available at `http://eprint.iacr.org/2016/612`.

CCS'16, October 24-28, 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2976749.2978376

1. INTRODUCTION

Today, large corporations and governments collect and store more personal information about us than ever before. And as high-profile data breaches on companies and organizations (such as Anthem [1], eBay [39], and the U.S. Voter Database [26]) become startlingly common, it is imperative that we develop practical means for securing our personal data in the cloud.

One way to mitigate the damage caused by a database breach is to encrypt the data before storing it in the cloud. This, however, comes at the price of functionality: once data is encrypted, it is more difficult to execute searches over the data without first decrypting the data. As a result, security researchers have turned to developing methods that both protect the contents of the database, as well as support efficient operations, such as search, over the encrypted data.

Property-preserving encryption. One way to support searching over an encrypted database is through property-preserving encryption (PPE) [9, 48, 21]. A PPE scheme is an encryption scheme where the ciphertexts reveal a particular property on their underlying plaintexts. Examples include deterministic encryption, where the ciphertexts reveal equality between messages, and order-preserving encryption (OPE) [2, 9], where the ciphertexts reveal the ordering of messages. Deterministic and order-preserving encryption schemes have been used in CryptDB [51], and also commercially by SkyHigh Networks, CipherCloud, Google Encrypted BigQuery, and others. One of the main appeals of PPE for encrypting relational databases is that they are lightweight, and hence, can be deployed with minimal changes to existing databases. For instance, in an OPE scheme, the ciphertexts themselves are numeric and the order of the ciphertexts precisely coincides with the order of the plaintexts. Thus, searching over a column encrypted using OPE is *identical* to searching over an unencrypted column.

Limitations of PPE and OPE. While PPE, and in particular, OPE, provides a practical solution for searching on encrypted data, these schemes also leak significant amounts of information about their underlying plaintexts. For instance, Boldyreva et al. [10] showed that a *single* OPE ciphertext leaks half of the most significant bits of its underlying plaintext!

More recently, Naveed et al. [46] described a series of inference attacks on relational databases encrypted using deterministic and order-preserving encryption schemes. They show that, given just a data dump of an encrypted database along with auxiliary information from a public database, an

attacker can successfully recover nearly all of the underlying plaintext values from their respective ciphertexts.

Our goals. Motivated by the limited security of existing OPE schemes and the emerging threat of inference attacks on databases encrypted using PPE, our goal in this work is to construct a practical property-preserving encryption for comparisons that achieves stronger security guarantees compared to existing OPE schemes while at the same time providing robustness against offline inference attacks, such as those considered by Naveed et al.

Order-revealing encryption. To address the limitations of OPE, we rely on a closely-related, but more flexible, notion called order-revealing encryption (ORE) [12, 22]. In this work, we focus exclusively on non-interactive and stateless schemes—these are the only schemes we know of that are deployed on a large scale. We survey the work on alternative solutions in Section 8.

In an OPE scheme, both the plaintext and ciphertext spaces must be numeric and well-ordered. Moreover, the ciphertexts themselves preserve the order of the underlying plaintexts. While this property makes OPE suitable for performing range queries on encrypted data, it also limits the achievable security of OPE schemes. In their original work, Boldyreva et al. [9] introduced the notion of "best-possible" semantic security for OPE, which states that the ciphertexts do not leak any information beyond the ordering of the plaintexts. Unfortunately, in the same work and a follow-up work [10], they show that any OPE scheme with best-possible security must have ciphertexts whose length grows exponentially in the length of the plaintexts. Popa et al. [50] further extended this lower bound to apply to stateful, interactive OPE schemes. These lower bounds rule out any hope of constructing efficient OPE schemes for large message spaces. As a compromise, Boldyreva et al. [9] introduced a weaker notion of security (POPF-CCA) for OPE schemes, but it is difficult to quantify the leakage of schemes which are POPF-CCA secure.

Recently, Boneh et al. [12] introduced the notion of ORE, which does *not* place any restrictions on the structure of the ciphertext space. An ORE scheme simply requires that there exists a *publicly* computable function that compares two ciphertexts. By relaxing the constraint on the ciphertext space, the Boneh et al. scheme is the first (non-interactive and stateless) scheme to achieve best-possible semantic security. However, their construction relies on multilinear maps [14, 27, 23], and is extremely far from being practically viable. More recently, Chenette et al. [22] introduced a new security model for ORE that explicitly models the information leakage of an ORE scheme. They also give the first efficiently-implementable ORE scheme. However their scheme also reveals the index of the first bit that differs between two encrypted values.

1.1 The Left/Right Framework for ORE

Before describing our main contributions, we first highlight the "left/right" framework for order-revealing encryption that we use in this work. Our notions are adapted from similar definitions for multi-input functional encryption [33, 12], where the encryption function operates on different "input slots." In a multi-input functional encryption scheme (of which ORE is a special case), information about plaintexts is only revealed when one has a ciphertext for *every* slot.

We now describe how this notion of encrypting to different input slots applies to order-revealing encryption. In a vanilla ORE scheme, there is a single encryption algorithm that takes a message and outputs a ciphertext. The comparison algorithm then takes two ciphertexts and outputs the comparison relation on the two underlying messages. In the left/right framework, we modify this interface and decompose the encryption function into two separate functions: a "left" encryption function and a "right" encryption function. Each of these encryption functions takes a message and the secret key, and outputs either a "left" or a "right" ciphertext, respectively. Next, instead of taking two ciphertexts, the comparison function takes a left ciphertext and a right ciphertext, and outputs the comparison relation between the two underlying messages (encrypted by the left and right ciphertexts). We note that any ORE scheme in the left/right framework can be converted to an ORE scheme in the usual sense by simply having the ORE encryption function output both the left and right ciphertexts for a given message.

This left/right notion is a strict generalization of the usual notion of order-revealing encryption, and thus, can be used to strengthen the security guarantees provided by an ORE scheme. In particular, a key advantage of working in this framework is that we can now define additional security requirements on collections of left or right ciphertexts taken in isolation. For example, in both of the ORE constructions we introduce in this work (Sections 3 and 4), a collection of right ciphertexts taken individually is semantically secure—that is, no information about the underlying plaintexts (including their order relations) is revealed given only a collection of right ciphertexts. In Section 5, we describe precisely how semantic security of the right ciphertexts can be leveraged to obtain a range query protocol that is robust against offline inference attacks. We also note that the schemes presented in this work are the first practical ORE constructions in the left/right framework where one side (the right ciphertexts) achieves semantic security.

Finally, we note that the left/right framework extends naturally to property-preserving encryption schemes, and thus, opens up many new avenues of developing more secure cryptographic primitives for searching on encrypted data.

1.2 Our Contributions

In this work, we describe a new ORE scheme that achieves stronger security compared to existing practical OPE and ORE schemes, as well as a method to leverage our new ORE scheme to efficiently perform range queries while providing robustness against inference attacks. We now highlight our main contributions.

An efficient small-domain ORE. We begin by giving the first construction of a practical, *small-domain* ORE scheme with *best-possible* semantic security that only relies on pseudorandom functions (PRFs).[1] The restriction to "small" domains is due to the fact that the ciphertext length in our scheme grows linearly in the size of the plaintext space. All existing constructions of ORE that achieve best-possible security in the small-domain setting rely on pairings [42], general-purpose functional encryption [3, 16], or multilinear maps [12], and thus, are not yet practical. Our particular

[1] We prove security in the random oracle model, but it is possible to replace the random oracle with a PRF to show security under a slightly weaker indistinguishability-based notion of security.

construction is inspired by the "brute-force" construction of functional encryption by Boneh et al. [13, §4.1]. They show that functional encryption with respect to a "small" (i.e., polynomially-sized) class of functions can be constructed using only symmetric primitives. We adapt these methods to show how best-possible ORE (and more generally, functional encryption) can be efficiently constructed from symmetric primitives when the *message space* is small. Our construction is described in Section 3.

Domain extension for ORE. Of course, a small-domain ORE by itself is not very useful for range queries. Our second contribution is a recasting of the Chenette et al. [22] ORE construction as a general technique of constructing a large-domain ORE from a small-domain ORE. The transformation is not perfect and incurs some leakage. Applying this domain-extension technique to our new small-domain ORE, we obtain an ORE scheme whose leakage profile is significantly better than that of the Chenette et al. construction. In particular, our new ORE scheme operates on blocks (where a block is a sequence of bits) and the additional leakage in our scheme is the position of the first block in which two messages differ. For instance, if blocks are byte-sized (8 bits), then our ORE scheme only reveals the index of the first byte that differs between the two messages (and nothing more). In contrast, the Chenette et al. construction always reveals the index of the first *bit* that differs.[2] Thus, our new ORE construction provides significantly stronger security, at the cost of somewhat longer ciphertexts.

Encrypted range queries. While our new ORE scheme can almost[3] be used as a drop-in replacement for OPE to enable searching over an encrypted database, the scheme remains susceptible to an offline inference attack. To carry out their inference attacks, Naveed et al. [46] rely on the fact that OPE-encrypted ciphertexts enable equality tests and comparisons (by design). In our setting, we take advantage of the special structure of the ciphertexts in our ORE scheme to obtain a way of supporting range queries on encrypted data while protecting against offline inference attacks.

Our range query protocol critically relies on the fact that our ORE scheme is a left/right ORE scheme (Section 1.1). More precisely, a ciphertext ct in our ORE scheme naturally decomposes into a left component ct_L and a right component ct_R. To compare two ciphertexts, the comparison function only requires the left component of one ciphertext and the right component of the other. More importantly, the right components have the property that they are *semantically-secure* encryptions of their messages. To build an encrypted database system with robustness against range queries, the database server only stores the "right" ciphertexts (in sorted order). To perform a range query, the client provides the "left" ciphertexts corresponding to its range. The server can respond to the range query as usual since comparisons are possible between left and right ciphertexts. Robustness against offline inference attacks is ensured since the database dump only contains the right ciphertexts stored on the server, which are semantically-secure encryptions of their underlying messages. We describe our method in greater detail in Section 5.

New lower bounds for OPE. The core building block in our new ORE construction is a small-domain ORE with best-possible security. This raises the natural question of whether we could construct a small-domain OPE that also achieves best-possible security. Previously, Boldyreva et al. [9, 10] and Popa et al. [50] gave lower bounds that ruled out schemes where the ciphertext space is subexponential in the size of the plaintext space. But when the plaintext has size $\mathrm{poly}(\lambda)$ for a security parameter λ, there could conceivably exist an efficient OPE scheme with best-possible security. In this work, we show that this is in fact impossible. Using a very different set of techniques compared to [9, 10, 50], we show (Section 6) that *no efficient* (stateless and non-interactive) OPE scheme can satisfy best-possible security, even when the message space contains only 3 elements! Thus, to achieve strong security even in the small-domain setting, it is necessary to consider relaxations of OPE, such as ORE.

Experimental evaluation. Finally, we implement and compare our new ORE scheme to the ORE scheme by Chenette et al. [22] and the OPE scheme by Boldyreva et al. [9]. For typical parameters, our new ORE scheme is over 65 times faster than the Boldyreva et al. scheme, but has longer ciphertexts. For example, when working with byte-size blocks, encrypting a 32-bit integer requires just 55 μs and produces a ciphertext that is 224 bytes. Typically, range queries are not performed over extremely long fields, so the extra space overhead of our scheme is not unreasonable. Given the superior security conferred by our scheme (in both the online and offline settings), and faster throughputs, our ORE scheme is a very compelling replacement for existing OPE schemes.

Applying ORE. To conclude, we make a cautionary note that because of the leakage associated with any ORE scheme, the primitive is not always suitable for applications that demand a high level of security. Our hope, however, is that by giving precise, concrete characterization of the leakage profile of our construction (in both the online and offline settings when used to support encrypted database queries), practitioners are able to make better-informed decisions on the suitability of our construction for a specific application.

2. PRELIMINARIES

For $n \in \mathbb{N}$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. If $\mathcal{P}$ is a predicate on x, we write $\mathbf{1}(\mathcal{P}(x))$ to denote the indicator function for $\mathcal{P}$: that is, $\mathbf{1}(\mathcal{P}(x)) = 1$ if and only if $\mathcal{P}(x) = 1$, and 0 otherwise. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote a draw from $\mathcal{D}$. For a finite set S, we write $x \xleftarrow{\mathrm{R}} S$ to denote a uniformly random draw from S. In this work, we write λ to denote a security parameter. We say a function $f(\lambda)$ is negligible in λ if $f = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We write $\mathrm{negl}(\lambda)$ to denote a negligible function in λ and $\mathrm{poly}(\lambda)$ to denote a polynomial in λ. We say that an event occurs with negligible probability if the probability of the event occurring is $\mathrm{negl}(\lambda)$. For two bit strings x, y, we write $x \| y$ to denote the concatenation of x and y.

We also review the standard definition of pseudorandom functions (PRFs) [31]. A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a secure PRF if no efficient adversary can distinguish (except perhaps

[2] While Chenette et al. also describe a multi-bit generalization of their scheme, the generalized version leaks *more* information, namely the difference of the values in the first differing block. In our construction, only the index and nothing else is revealed.

[3] We say "almost," since using ORE in place of OPE would require writing a custom comparator for database elements.

with negligible probability) the outputs (on arbitrary points chosen adaptively by the adversary) of $F(k,\cdot)$ for a randomly chosen $k \xleftarrow{R} \mathcal{K}$ from that of a truly random function $f(\cdot)$ from $\mathcal{X}$ to $\mathcal{Y}$. Similarly, a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ is a secure pseudorandom permutation (PRP) if for all $k \in \mathcal{K}$, $F(k,\cdot)$ is a permutation on $\mathcal{X}$ and no efficient adversary can distinguish the outputs of $F(k,\cdot)$ where $k \xleftarrow{R} \mathcal{K}$ from the outputs of $\pi(\cdot)$ where π is a random permutation on $\mathcal{X}$.

2.1 Order-Revealing Encryption

An order-revealing encryption (ORE) scheme [12, 22] is a tuple of three algorithms $\Pi = (\mathsf{ORE.Setup}, \mathsf{ORE.Encrypt}, \mathsf{ORE.Compare})$ defined over a well-ordered domain $\mathcal{D}$ with the following properties:

- $\mathsf{ORE.Setup}(1^\lambda) \to \mathsf{sk}$: On input a security parameter λ, the setup algorithm outputs a secret key sk.
- $\mathsf{ORE.Encrypt}(\mathsf{sk}, m) \to \mathsf{ct}$: On input a secret key sk and a message $m \in \mathcal{D}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{ORE.Compare}(\mathsf{ct}_1, \mathsf{ct}_2) \to b$: On input two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, the compare algorithm outputs a bit $b \in \{0,1\}$.

Correctness. We say an ORE scheme over a well-ordered domain $\mathcal{D}$ is correct if for $\mathsf{sk} \leftarrow \mathsf{ORE.Setup}(1^\lambda)$ and all messages $m_1, m_2 \in \mathcal{D}$,

$$\Pr[\mathsf{ORE.Compare}(\mathsf{ct}_1, \mathsf{ct}_2) = \mathbf{1}(m_1 < m_2)] = 1 - \mathrm{negl}(\lambda).$$

Remark 2.1 (ORE Decryption). *Our above schema for ORE does not include a decryption function, but as noted by Chenette et al. [22, Remark 2.3], this is without loss of generality. In particular, we can construct a decryption algorithm* $\mathsf{ORE.Decrypt}$ *using the* $\mathsf{ORE.Encrypt}$ *and* $\mathsf{ORE.Compare}$ *algorithms (by performing a binary search).*

Security. The "best-possible" notion of security for order-revealing encryption is the notion of indistinguishability under an ordered chosen plaintext attack (IND-OCPA) introduced by Boldyreva et al. [9]. The IND-OCPA notion of security is a generalization of semantic security [34], and states that no efficient adversary can distinguish between the encryptions of any two sequences of messages, provided that the ordering of the messages in the two sequences is identical. We give the formal definition in the full version [43].

Due to the apparent difficulty in constructing efficient schemes that satisfy IND-OCPA security, Chenette et al. [22] introduced a weaker simulation-based notion of security for ORE schemes that allows for some leakage beyond just the ordering of the plaintexts. We recall their definition here.

Definition 2.2 (ORE with Leakage [22]). *Let* Π *be an ORE scheme—that is,* $\Pi = (\mathsf{ORE.Setup}, \mathsf{ORE.Encrypt}, \mathsf{ORE.Compare})$*, and let* $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_q)$ *be an adversary for some* $q = \mathrm{poly}(\lambda)$*. Let* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_q)$ *be a simulator, and let* $\mathcal{L}(\cdot)$ *be a leakage function. We define the experiments* $\mathsf{REAL}^{\mathrm{ORE}}_{\mathcal{A}}(\lambda)$ *and* $\mathsf{SIM}^{\mathrm{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$ *in Figure 1. We say that* Π *is a secure ORE scheme with leakage function* $\mathcal{L}(\cdot)$ *if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_q)$*, there exists a polynomial-size simulator* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_q)$ *such that the outputs of the two distributions* $\mathsf{REAL}^{\mathrm{ORE}}_{\mathcal{A}}(\lambda)$ *and* $\mathsf{SIM}^{\mathrm{ORE}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$ *are computationally indistinguishable.*

Remark 2.3 (Best-Possible Security). *The strongest notion of simulation-security is security with respect to the leakage function that only reveals the ordering of the plaintexts. This is the minimal leakage possible from an order-revealing encryption scheme. In particular, we define* $\mathcal{L}_{\mathrm{CMP}}$ *as follows:*

$$\mathcal{L}_{\mathrm{CMP}}(m_1, \ldots, m_t) = \{(i, j, \mathrm{CMP}(m_i, m_j) \mid 1 \le i < j \le t)\},$$

where $\mathrm{CMP}(m_i, m_j)$ *is the comparison function that outputs* -1 *if* $m_i < m_j$*,* 0 *if* $m_i = m_j$ *and* 1 *if* $m_i > m_j$*.*

3. ORE FOR SMALL DOMAINS

The order-revealing encryption in [22] reveals a significant amount of information, namely, the index of the first bit position that differs between two encrypted plaintexts. In this work, we show how to construct an ORE scheme that only leaks the first *block* that differs, where a block is a collection of one or more bits. For instance, we can construct an ORE scheme that only reveals the first *byte* that differs between two encrypted plaintexts, and nothing more.

The starting point for our construction is a "small-domain" ORE scheme with best-possible simulation security. The limitation is that the length of the ciphertexts in our ORE scheme grows linearly with the size of the message space, hence the restriction to small (polynomially-sized) domains. We show in Section 4 how to extend our small-domain ORE to obtain an order-revealing encryption scheme over large domains (i.e., exponentially-sized) that leaks strictly less information compared to the scheme by Chenette et al. [22].

As described in Section 1.1, we give our ORE construction in the left/right framework where we decompose the $\mathsf{ORE.Encrypt}$ function into two functions: $\mathsf{ORE.Encrypt_L}$ and $\mathsf{ORE.Encrypt_R}$. We refer to them as the "left encryption" and "right encryption" functions, respectively. Our particular construction has the property that only "left ciphertexts" can be compared with "right ciphertexts." Note that this is without loss of generality and we can recover the usual notion of ORE by simply defining the output of $\mathsf{ORE.Encrypt}(\mathsf{sk}, m)$ to be the tuple $(\mathsf{ORE.Encrypt_L}(\mathsf{sk}, m), \mathsf{ORE.Encrypt_R}(\mathsf{sk}, m))$.

3.1 Small-Domain ORE Construction

We begin with a high-level overview of our construction. Our scheme is defined with respect to a plaintext space $[N]$ where $N = \mathrm{poly}(\lambda)$. First, we associate each element $x \in [N]$ in the domain with an encryption key k_x. A (right) ciphertext for a value $y \in [N]$ consists of N encryptions of the comparison output $\mathrm{CMP}(x, y)$ between y and every element $x \in [N]$ in the domain, where the value $\mathrm{CMP}(x, y)$ is encrypted under k_x. The left encryption of a value x is simply the encryption key k_x. Given k_x and an encryption of $\mathrm{CMP}(x, y)$ under k_x, the evaluator can decrypt and learn the comparison bit $\mathrm{CMP}(x, y)$. The values of the other comparison bits are hidden by semantic security of the encryption scheme. Note, however, that we still need a way for the evaluator to determine *which* of the N ciphertexts is encrypted under k_x without learning the value of x. To ensure this, we sample a random permutation π on the domain $[N]$ during setup. The components in the right ciphertexts are then permuted according to π and the left encryption of x includes the permuted position $\pi(x)$. Given $\pi(x)$, the evaluator learns which component in the right ciphertext to decrypt, but learns nothing about x. Finally, to show sim-

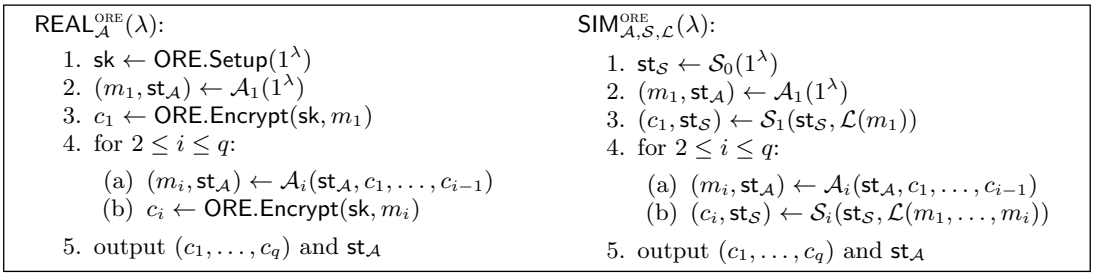

Figure 1: Real and ideal experiments for ORE with leakage (Definition 2.2).

ulation security, we require a "non-committing" encryption scheme, and for this, we rely on a random oracle [6].[4]

Construction. Let $[N]$ be the message space. Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a secure PRF and $H : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_3$ be a hash function (modeled as a random oracle in the security proof). Let CMP be the comparison function from Remark 2.3. Our ORE scheme $\Pi^{(\mathsf{s})}_{\mathsf{ore}}$ is defined as follows:

- $\mathsf{ORE.Setup}(1^\lambda)$. The setup algorithm samples a PRF key $k \xleftarrow{\text{R}} \{0,1\}^\lambda$ for F, and a uniformly random permutation $\pi : [N] \to [N]$. The secret key sk is the pair (k, π).
- $\mathsf{ORE.Encrypt_L}(\mathsf{sk}, x)$. Write sk as (k, π). The left encryption algorithm computes and returns the tuple $\mathsf{ct_L} = (F(k, \pi(x)), \pi(x))$.
- $\mathsf{ORE.Encrypt_R}(\mathsf{sk}, y)$. Write sk as (k, π). First, the right encryption algorithm samples a random nonce $r \xleftarrow{\text{R}} \{0,1\}^\lambda$. Then, for each $i \in [N]$, it computes the value
$$v_i = \text{CMP}(\pi^{-1}(i), y) + H(F(k, i), r) \pmod 3.$$
Finally, it outputs the ciphertext $\mathsf{ct_R} = (r, v_1, v_2 \ldots, v_N)$.
- $\mathsf{ORE.Compare}(\mathsf{ct_L}, \mathsf{ct_R})$. The compare algorithm first parses
$$\mathsf{ct_L} = (k', h) \quad \text{and} \quad \mathsf{ct_R} = (r, v_1, v_2, \ldots, v_N),$$
and then outputs the result $v_h - H(k', r) \pmod 3$.

Correctness. Let $\mathsf{sk} = (k, \pi) \leftarrow \mathsf{ORE.Setup}(1^\lambda)$, and take any $x, y \in [N]$. Let $\mathsf{ct}^{(x)}_{\mathsf{L}} = (k', h) \leftarrow \mathsf{ORE.Encrypt_L}(\mathsf{sk}, x)$ and and $\mathsf{ct}^{(y)}_{\mathsf{R}} = (r, v_1, \ldots, v_N) \leftarrow \mathsf{ORE.Encrypt_R}(\mathsf{sk}, y)$, Then, setting $z = \mathsf{ORE.Compare}(\mathsf{ct}^{(x)}_{\mathsf{L}}, \mathsf{ct}^{(y)}_{\mathsf{R}})$, we have

$$\begin{aligned} z &= v_h - H(k', r) \\ &= \text{CMP}(\pi^{-1}(h), y) + H(F(k, h), r) - H(k', r) \\ &= \text{CMP}(\pi^{-1}(\pi(x)), y) + H(F(k, \pi(x)), r) - H(F(k, \pi(x)), r) \\ &= \text{CMP}(x, y) \in \mathbb{Z}_3, \end{aligned}$$

Note that $\text{CMP}(x, y)$ provides the same amount of information as $\mathbf{1}(x < y)$ and $\mathbf{1}(y < x)$, so correctness follows.

Space usage. Before we give our formal security analysis, we first characterize the length of the ciphertexts in our ORE scheme for a message space of size N. The left ciphertexts $\mathsf{ct_L}$ in our scheme consists of a PRF key and an index, which are $\lambda + \lceil \log N \rceil$ bits long. The right ciphertexts $\mathsf{ct_R}$ consists of a nonce, together with N elements in $\mathbb{Z}_3$, which can be represented using $\lambda + \lceil N \log_2 3 \rceil$ bits. Thus, a complete ciphertext consists of $2\lambda + \lceil \log N \rceil + \lceil N \log_2 3 \rceil$ bits. However, as we note in the following remark, it is possible to obtain shorter ciphertexts if we allow the comparison algorithm to take the full ciphertext $(\mathsf{ct_L}, \mathsf{ct_R})$ as opposed to only the left half of the first ciphertext and the right half of the second ciphertext. Thus, when using the construction as a pure ORE scheme, we can obtain shorter ciphertexts. When leveraging our ORE scheme to build a range query system (Section 5), we will exploit the fact that comparisons can be performed given just the left component of one ciphertext and the right component of the other.

[4]We believe we can replace the random oracle with a PRF if we aim to prove an indistinguishability notion of security for our construction. For simplicity of presentation in this paper, we work with a simulation-based definition and prove security in the random oracle model.

REMARK 3.1 (SHORTER CIPHERTEXTS). *For a domain of size N, the right ciphertexts in our ORE construction contain N elements of $\mathbb{Z}_3$. Suppose instead we replaced the comparison function* CMP *with a function* CMP′ *where* $\text{CMP}'(x, y) = 1$ *if* $x \le y$ *and* 0 *otherwise. Then, a left encryption* $\mathsf{ct}^{(x)}_{\mathsf{L}}$ *of x and a right encryption* $\mathsf{ct}^{(y)}_{\mathsf{R}}$ *of y can be used to compute* $\text{CMP}'(x, y)$*, or equivalently, whether* $x \le y$*. If the comparison algorithm takes as input* $\mathsf{ct}^{(x)} = \left(\mathsf{ct}^{(x)}_{\mathsf{L}}, \mathsf{ct}^{(x)}_{\mathsf{R}}\right)$ *and* $\mathsf{ct}^{(y)} = \left(\mathsf{ct}^{(y)}_{\mathsf{L}}, \mathsf{ct}^{(y)}_{\mathsf{R}}\right)$*, then it can compute both* $\text{CMP}'(x, y)$ *and* $\text{CMP}'(y, x)$*. This means that given* $\mathsf{ct}^{(x)}$ *and* $\mathsf{ct}^{(y)}$*, the comparison algorithm can still determine if* $x < y$*,* $x = y$*, or* $x > y$*. With this modification, the right ciphertexts in our scheme have length N rather than* $\lceil N \log_2 3 \rceil$*.*

In the full version [43], we also show how our construction above can be extended to support more general functionalities (beyond just comparisons) over polynomial-sized domains. We now state our main security theorem, but defer the formal proof to the full version.

THEOREM 3.2. *The ORE scheme $\Pi^{(\mathsf{s})}_{\mathsf{ore}}$ is secure with the best-possible leakage function $\mathcal{L}_{\text{CMP}}$ from Remark 2.3 assuming that F is a secure PRF and H is modeled as a random oracle.*

4. DOMAIN EXTENSION FOR ORE

Although our small-domain ORE construction from Section 3 achieves the strongest possible notion of security for ORE, it is limited to polynomially-sized message spaces. In this section, we show how to construct an efficient ORE scheme for large domains which achieves provably stronger security guarantees than all existing efficient ORE constructions for large domains. Our construction can be viewed as a composition of our small-domain ORE construction together with the ORE scheme by Chenette et al. [22].

Intuitively, we can view the techniques used in the Chenette et al. construction as a domain-extension mechanism

for ORE. In particular, their construction can be viewed as a general transformation that takes as input a k-bit ORE scheme and outputs an kn-bit ORE scheme, with ciphertext expansion that grows linearly in n and a slight reduction in security (that degrades with n). Under this lens, the Chenette et al. construction can be viewed as taking a 1-bit ORE scheme (with best-possible security) and extending it to an n-bit ORE scheme. In this work, we apply this general domain-extension technique to our small-domain ORE from Section 3, and show how we can start with a d-bit ORE and extend it to a dn-bit ORE. By varying the parameters n and d, we obtain a performance-security tradeoff. At a high level, our composed construction implements encryption via several parallel (prefix-dependent) instances of the small-domain ORE scheme $\Pi_{\mathsf{ore}}^{(\mathsf{s})}$ from Section 3, one for each block of the plaintext. Using the techniques of Chenette et al. [22], a blinding factor is derived from the prefix of each block and used to mask the $\Pi_{\mathsf{ore}}^{(\mathsf{s})}$ ciphertexts for that block. We give the precise leakage of our construction in Theorem 4.1.

Construction. Fix a security parameter $\lambda \in \mathbb{N}$, a message space size $N > 0$, and integers $d, n > 0$ such that $d^n \geq N$. Let $F : \{0,1\}^\lambda \times [N] \to \{0,1\}^\lambda$ be a secure PRF on variable-length inputs,[5] $H : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_3$ be a hash function (modeled as a random oracle), and $\pi : \{0,1\}^\lambda \times [d] \to [d]$ be a secure PRP. For a d-ary string $x = x_1 x_2 \cdots x_n$, let $x_{|i} = x_1 x_2 \cdots x_i$ denote the d-ary string representing the first i digits of x (i.e., the length-i prefix of x), and let $x_{|0}$ be the empty prefix. We define our ORE scheme $\Pi_{\mathsf{ore}} = (\mathsf{ORE.Setup}, \mathsf{ORE.Encrypt_L}, \mathsf{ORE.Encrypt_R}, \mathsf{ORE.Compare})$ as follows.

- $\mathsf{ORE.Setup}(1^\lambda)$. The setup algorithm samples PRF keys $k_1, k_2 \xleftarrow{\text{R}} \{0,1\}^\lambda$. The master secret key is $\mathsf{sk} = (k_1, k_2)$.
- $\mathsf{ORE.Encrypt_L}(\mathsf{sk}, x)$. Let $\mathsf{sk} = (k_1, k_2)$. For each $i \in [n]$, the left encryption algorithm first computes the quantity $\tilde{x} = \pi(F(k_2, x_{|i-1}), x_i)$ and then sets $u_i = (F(k_1, x_{|i-1} \| \tilde{x}), \tilde{x})$. It returns the ciphertext $\mathsf{ct_L} = (u_1, \ldots, u_n)$.
- $\mathsf{ORE.Encrypt_R}(\mathsf{sk}, y)$. Let $\mathsf{sk} = (k_1, k_2)$. First, the right encryption algorithm uniformly samples a nonce $r \xleftarrow{\text{R}} \{0,1\}^\lambda$. Then, for each $i \in [n]$ and $j \in [d]$, it first sets $j^* = \pi^{-1}(F(k_2, y_{|i-1}), j)$, it computes

  $$z_{i,j} = \text{CMP}(j^*, y_i) + H(F(k_1, y_{|i-1} \| j), r) \pmod 3.$$

  It then defines the tuple $v_i = (z_{i,1}, \ldots, z_{i,d})$ and outputs the ciphertext $\mathsf{ct_R} = (r, v_1, v_2 \ldots, v_n)$.
- $\mathsf{ORE.Compare}(\mathsf{ct_L}, \mathsf{ct_R})$. The compare algorithm first parses

  $$\mathsf{ct_L} = (u_1, \ldots, u_n) \quad \text{and} \quad \mathsf{ct_R} = (r, v_1, v_2, \ldots, v_n),$$

  where for each $i \in [n]$, we write $u_i = (k'_i, h_i)$ and $v_i = (z_{i,1}, \ldots, z_{i,d})$. Then, let ℓ be the smallest index i for which $z_{i,h_i} - H(k'_i, r) \neq 0 \pmod 3$. If no such ℓ exists, output 0. Otherwise, output $z_{\ell,h_\ell} - H(k'_\ell, r) \pmod 3$.

Correctness. Correctness follows similarly to that for the small-domain ORE in Section 3. We give the full argument in the full version [43].

Security. Before stating our security theorem, we first specify our leakage function $\mathcal{L}_{\text{BLK}}^{(d)}$. Each ciphertext block in our ORE scheme is essentially a ciphertext for the underlying small-domain ORE, and the comparison operation proceeds block-by-block. Intuitively then, since our small-domain ORE scheme leaks nothing except the ordering (Theorem 3.2), the additional leakage of our new ORE scheme is the index of the first block that differs between two ciphertexts. In particular, for messages $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_n$ written in base d, we define the first differing block function $\mathsf{ind}_{\mathsf{diff}}^{(d)}(x, y)$ to be the first index $i \in [n]$ such that $x_j = y_j$ for all $j < i$ and $x_i \neq y_i$. If $x = y$, we define $\mathsf{ind}_{\mathsf{diff}}^{(d)}(x, y)$ to be $n + 1$. Then, our leakage function $\mathcal{L}_{\text{BLK}}^{(d)}$ for our extended ORE scheme is given by

$$\mathcal{L}_{\text{BLK}}^{(d)}(m_1, \ldots, m_t) = \{(i, j, \text{BLK}(m_i, m_j)) \mid 1 \leq i < j \leq t\},$$

where $\text{BLK}(m_i, m_j) = (\text{CMP}(m_i, m_j), \mathsf{ind}_{\mathsf{diff}}^{(d)}(m_i, m_j))$. In general, we refer to the parameter d as the arity (or base) of the plaintext space, which grows exponentially in the length (in bits) of the block. We now state our main security theorem.

THEOREM 4.1. *The ORE scheme Π_{ore} is secure with leakage function $\mathcal{L}_{\text{BLK}}$ assuming that F is a secure PRF and H is modeled as a random oracle.*

The proof of Theorem 4.1 can be viewed as a composition of the security proof for our underlying small-domain ORE (Theorem 3.2) and the security proof of the Chenette et al. scheme [22, Theorem 3.2]. We give the proof in the full version.

Space usage. Ciphertexts in our new ORE scheme consist essentially of n ciphertexts for our small-domain ORE scheme (with domain size d). More concretely, a left ciphertext in our new scheme consists of $n(\lambda + \lceil \log d \rceil)$ bits and a right ciphertext consists of $\lambda + n \lceil d \log_2 3 \rceil$ bits. Since the size of the plaintext space N satisfies $N \leq d^n$, ciphertext size in our new ORE scheme grow as $O((\lambda + d) \log_d N)$.

Non-uniform block sizes. In practice, some bits of the plaintext may be more sensitive than others. Leaking information about these bits is less desirable than leaking information about less sensitive bits. To accommodate the different sensitivities, we can use different input bases (e.g., use larger blocks for more sensitive bits) for the different blocks of the ciphertext. The leakage in the resulting scheme is still the index of the first (variable-sized) block that differs between two messages. Correctness is unchanged.

5. ENCRYPTED RANGE QUERIES

In this section, we formally define the properties of a client-server protocol for range queries over an encrypted database. In our model, a client stores an encrypted database on the server. The client can update the database (e.g., by adding or removing records) and issue range queries against the database. In a range query, the client specifies a numeric interval and the server responds by returning all ciphertexts whose underlying messages fall within that interval.

Although our definitions are stated in terms of numeric intervals, our methods are broadly applicable to more general settings—in particular, to any well-ordered domain such as English names. For example, when the database consists of encrypted alphanumeric strings, range queries can be used for both exact-keyword as well as prefixed-based search.

Our security definitions are adapted from existing definitions for searchable symmetric encryption (SSE) [24, 20].

[5] The Chenette et al. ORE construction also used a PRF on variable-length inputs. We refer to their construction [22, §3] for one possible way of constructing a PRF on variable-length inputs from a standard PRF.

We survey some of the work on SSE in Section 8. In our definitions we consider both the online and offline settings. In the online setting, the adversary sits on the server and sees both the encrypted database as well as the client's queries, while in the offline setting, the adversary just obtains a dump of the server's encrypted database. By showing that in the offline setting, the server's encrypted database provides semantic security, we can argue that our new range query scheme provides robustness against the kinds of offline inference attacks considered by Naveed et al. [46].

After formally defining the security requirements for a range query protocol, we give a construction based on our ORE scheme Π_{ore} from Section 4. Our protocol not only satisfies our security properties, but also has several additional appealing properties such as sublinear query time (in the size of the database) and optimal round complexity.

Our proposed protocol is easily extensible to the multi-client setting where many clients are interacting with the server. Each authorized client is simply given the secret key needed to query and update the database.

5.1 Range Query Schemes

We begin with a formal definition of a range query scheme, followed by our notions of online and offline security. We describe a range query scheme in terms of a set of algorithms, where each algorithm is a single-round protocol between the client and the server. In each protocol, the client is always stateless, but the server is stateful—in particular, the server's state represents the information stored on the server needed to efficiently respond to the client's queries, including the encrypted database itself.

Initially, the client runs a setup procedure that takes as input a plaintext database $\boldsymbol{D}$ of values and outputs a secret key sk and some token t representing the encrypted database. The token t is given to the server, and the server outputs some initial state st. Then, for each query (range query, insert query, delete query), the client uses the secret key sk to derive a token t representing its query, and sends t to the server. This token contains a masked version of the client's input for the query. On input a query token t, the server processes the query and updates its internal state. In a range query, the server also returns a response r, which the client uses to learn the answer to the range query.

More formally, let $\boldsymbol{D} \in [N]^M$ represent a (possibly empty) database consisting of $M \geq 0$ values, each in the range $[N]$. A range query scheme $\Pi_{\mathsf{rq}} = (\mathsf{RQ.Setup}, \mathsf{RQ.Range}, \mathsf{RQ.Insert}, \mathsf{RQ.Delete})$ consists of a tuple of algorithms defined as follows:

- $\mathsf{RQ.Setup}(1^\lambda, \boldsymbol{D}) \to (\mathsf{t}, \mathsf{st})$. The setup algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(1^\lambda, \boldsymbol{D}) \to (\mathsf{sk}, \mathsf{t})$. The client, on input the security parameter λ and database $\boldsymbol{D}$, produces a key sk which is kept secret, and a token t which is sent to the server.
  - $\mathsf{Server}(\mathsf{t}) \to \mathsf{st}$. The server takes as input the token t and outputs an initial state st.
- $\mathsf{RQ.Range}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The range query algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = (x, y)) \to \mathsf{t}$. The client, on input the secret key sk and a query q for the range $[x, y]$, produces a token t which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state st and the token t and produces an updated state st', along with a response r, which is sent to the client.
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{r}) \to \mathsf{S}$. The client, on input the secret key sk and the response r from the server, obtains a subset S of entries which represent the answer to the range query.
- $\mathsf{RQ.Insert}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The insert algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = x) \to \mathsf{t}$. The client, on input the secret key sk and a query q representing an insertion of the value x, produces a token t which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state st and the token t and produces an updated state st'.
- $\mathsf{RQ.Delete}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The delete algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = x) \to \mathsf{t}$. The client, on input the secret key sk and a query q representing a deletion of the value x, produces a token t which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state st and the token t and produces an updated state st'.

We now define the correctness and security properties of a range query scheme. At a high level, we say that a range query scheme is correct if for all range queries (x, y) the client makes, it obtains the set of entries in the database $\boldsymbol{D}$ (taking into account any insertion and deletion queries occurring before the range query) that lie in the interval $[x, y]$.

Correctness. Fix a security parameter λ, positive integers x, y, N, M where $x \leq y \in [N]$, a database $\boldsymbol{D} \in [N]^M$ and a sequence of ℓ insertion, deletion, and range queries $\mathsf{q}_1, \ldots, \mathsf{q}_{\ell-1}$. Let $\mathsf{q}_\ell = (x, y)$ be a range query. Let $(\mathsf{st}_\ell, \mathsf{r}) \leftarrow \mathsf{Server}(\mathsf{st}_{\ell-1}, \mathsf{Client}(\mathsf{sk}, \mathsf{q}_\ell))$ and $\mathsf{S} \leftarrow \mathsf{Client}(\mathsf{sk}, \mathsf{r})$, where $\mathsf{st}_{\ell-1}$ is the server's state after processing queries $\mathsf{q}_1, \ldots, \mathsf{q}_{\ell-1}$. Let $\boldsymbol{D}_0 = \boldsymbol{D}, \boldsymbol{D}_1, \ldots, \boldsymbol{D}_\ell$ be the effective database elements after each query—that is, for all $i \in [\ell]$, $\boldsymbol{D}_i = \boldsymbol{D}_{i-1}$ if q_i is a range query, $\boldsymbol{D}_i = \boldsymbol{D}_{i-1} \cup \{x\}$ if q_{i-1} is an insertion query for x, and $\boldsymbol{D}_i = \boldsymbol{D}_{i-1} \setminus \{x\}$ if q_{i-1} is a deletion query for x. We say a range query scheme $\Pi_{\mathsf{rq}} = (\mathsf{RQ.Setup}, \mathsf{RQ.Range}, \mathsf{RQ.Insert}, \mathsf{RQ.Delete})$ is correct if for all security parameters λ, integers N, M, x, y, databases $\boldsymbol{D} \in [N]^M$ and sequence of queries $\mathsf{q}_1, \ldots, \mathsf{q}_\ell$, we have that the client's response S satisfies $\mathsf{S} = \boldsymbol{D}_\ell \cap [x, y]$.

Security. Our first notion of security is online security, which models the information revealed to a malicious server in the range query protocol. Here, the adversary sees both the contents of the server's state (i.e., the encrypted database) as well as the client's queries. We give a simulation-based definition with respect to a concrete leakage function that operates over the plaintext values in the database and the queries. Our definition is adapted from the standard paradigm used to define security in searchable symmetric encryption schemes [24, 20].

Definition 5.1 (Online Security). *For all databases $\boldsymbol{D}$ and sequences of ℓ queries $\mathsf{q}_1, \ldots, \mathsf{q}_\ell$, define the sequence of states $\mathsf{st}_0, \ldots, \mathsf{st}_\ell$ and tokens $\mathsf{t}_0, \ldots, \mathsf{t}_\ell$ where $(\mathsf{t}_0, \mathsf{st}_0) \leftarrow \mathsf{RQ.Setup}(1^\lambda, \boldsymbol{D})$, and for each $i \in [\ell]$, $(\mathsf{t}_i, \mathsf{st}_i)$ is the output of the i^{th} query on input sk, q_i, and st_{i-1}. A range query scheme is online secure with respect to a leakage function $\mathcal{L}$*

if for every efficient adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ where

$$\big|\Pr[\mathcal{A}(1^\lambda, \mathsf{st}_0, \ldots, \mathsf{st}_\ell, \mathsf{t}_0, \ldots, \mathsf{t}_\ell) = 1] - \Pr[\mathcal{S}(1^\lambda, \mathcal{L}(\boldsymbol{D}, \mathsf{q}_1, \ldots, \mathsf{q}_\ell)) = 1]\big| = \mathrm{negl}(\lambda).$$

We also define an "offline" notion of security for a range query scheme. The offline setting models scenarios where the adversary obtains a dump of the contents of the server (i.e., the server's state), but does not observe any queries made by the client. Against offline adversaries, we require the much stronger property that the only thing leaked by the encrypted database is the size of the encrypted database. This is the best-possible leakage.

Definition 5.2 (Offline Security). *For all databases $\boldsymbol{D}$ and sequences of ℓ queries $\mathsf{q}_1, \ldots, \mathsf{q}_\ell$, define the sequence of states $\mathsf{st}_0, \ldots, \mathsf{st}_\ell$ and tokens $\mathsf{t}_0, \ldots, \mathsf{t}_\ell$ as in Definition 5.1 Let $|\mathsf{st}_\ell|$ be the bit-length of st_ℓ. A range query scheme is offline secure if for all efficient adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ where*

$$\Big|\Pr[\mathcal{A}(1^\lambda, \mathsf{st}_\ell) = 1] - \Pr[\mathcal{S}(1^\lambda, |\mathsf{st}_\ell|) = 1]\Big| = \mathrm{negl}(\lambda).$$

The importance of offline security. Although offline security is strictly weaker than online security, it captures the real-world scenario where an attacker breaks into a server and exfiltrates any data the server has stored on disk. While companies are often able to detect and protect against active online corruption of their servers, the question remains what happens after the fact when the attacker has also exfiltrated the database for offline analysis. Of course, the ideal solution to this problem is an encrypted database system that provides strong online security guarantees. However, existing systems with strong online security typically require redesigning the database management system and implementing elaborate cryptographic protocols for querying [17, 25], or leverage heavy, less practical tools such as fully homomorphic encryption [29] or oblivious RAMs [32]. On the flip side, an OPE-based solution yields a scheme that does not provide offline security in our model; this is one reason why OPE and other PPE-based encrypted database schemes are vulnerable to inference attacks. This is true even if we use an (interactive) OPE scheme with best-possible security; the ability to directly compare ciphertexts is sufficient to carry out the inference attacks. Thus, there is an interesting intermediate ground where we build systems that achieve decent online security, while still providing strong offline security guarantees to be robust against inference attacks.

5.2 An Efficient Range Query Scheme

We now describe how to build an efficient range query scheme using our ORE construction from Section 4. At a high level, the server's encrypted database consists of right ciphertexts for each value, stored in sorted order. The tokens t for each query consist of a left encryption of the query value. This allows the server to use the ORE comparison algorithm to perform binary search over the encrypted ciphertexts in the database. Thus, the server is able to answer queries efficiently and maintain the database in sorted order (during updates). To answer a range query, the server performs binary search to find the lower and upper boundaries in the encrypted database corresponding to its query and returns all ciphertexts lying within those bounds. The client then decrypts the ciphertexts to learn the response.

More formally, we define our range query scheme $\Pi_{\mathsf{rq}} = (\mathsf{RQ.Setup}, \mathsf{RQ.Range}, \mathsf{RQ.Insert}, \mathsf{RQ.Delete})$ as follows:

- $\mathsf{RQ.Setup}(1^\lambda, \boldsymbol{D}) \to (\mathsf{t}, \mathsf{st})$. The setup algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(1^\lambda, \boldsymbol{D}) \to (\mathsf{sk}, \mathsf{t})$. The client, on input the security parameter λ and database $\boldsymbol{D}$, generates a secret key $\mathsf{sk} \leftarrow \mathsf{ORE.Setup}(1^\lambda)$. Then, the client *sorts* the database $\boldsymbol{D}$, and for each sequential element $x_i \in \boldsymbol{D}$, the client computes $\mathsf{ct}_i \leftarrow \mathsf{ORE.Encrypt_R}(\mathsf{sk}, x_i)$, and sends the token $\mathsf{t} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_M)$ to the server.
  - $\mathsf{Server}(\mathsf{t}) \to \mathsf{st}$. The server simply sets $\mathsf{st} = \mathsf{t}$.
- $\mathsf{RQ.Range}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The range query algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = (x, y)) \to \mathsf{t}$. The client, on input the secret key sk and a query representing a range query for the range $[x, y]$, produces the token $\mathsf{t} = (\mathsf{ORE.Encrypt_L}(\mathsf{sk}, x), \mathsf{ORE.Encrypt_L}(\mathsf{sk}, y))$ which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state $\mathsf{st} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{M'})$ for some integer M', and the token $\mathsf{t} = (\mathsf{ct}_x, \mathsf{ct}_y)$. Using $\mathsf{ORE.Compare}$, it performs a binary search to find the ciphertexts in st that are "at least" ct_x and "at most" ct_y. Let r be the set of ciphertexts lying in this interval. The server outputs the response r and an updated state $\mathsf{st}' = \mathsf{st}$.
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{r}) \to \mathsf{S}$. The client, on input the secret key sk and the response $\mathsf{r} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$ for some integer m, outputs the tuple $\mathsf{S} = (\mathsf{ORE.Decrypt}(\mathsf{sk}, \mathsf{ct}_1), \ldots, \mathsf{ORE.Decrypt}(\mathsf{sk}, \mathsf{ct}_m))$. (Recall from Remark 2.1 that any ORE scheme can be augmented with a decryption algorithm.)
- $\mathsf{RQ.Insert}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The insert algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = x) \to \mathsf{t}$. The client, on input the secret key sk and a query representing an insertion of the value x, produces a token $\mathsf{t} = (\mathsf{ORE.Encrypt_L}(\mathsf{sk}, x), \mathsf{ORE.Encrypt_R}(\mathsf{sk}, x))$ which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state st and the token $\mathsf{t} = (\mathsf{ct}_1, \mathsf{ct}_2)$. Using $\mathsf{ORE.Compare}(\mathsf{ct}_1, \cdot)$, it performs a binary search over the contents of its database st to find the index at which to insert the new value. The server inserts ct_2 at that position and outputs the updated database st'.
- $\mathsf{RQ.Delete}(\mathsf{sk}, \mathsf{q}, \mathsf{st}) \to (\mathsf{t}, \mathsf{st}')$. The delete algorithm between the client and server proceeds as follows:
  - $\mathsf{Client}(\mathsf{sk}, \mathsf{q} = x) \to \mathsf{t}$. The client, on input the secret key sk and a query representing a deletion of the value x, produces a token $\mathsf{t} = (\mathsf{ORE.Encrypt_L}(\mathsf{sk}, x), \mathsf{ORE.Encrypt_R}(\mathsf{sk}, x))$ which is sent to the server.
  - $\mathsf{Server}(\mathsf{st}, \mathsf{t}) \to (\mathsf{st}', \mathsf{r})$. The server takes as input its current state st and the token $\mathsf{t} = (\mathsf{ct}_1, \mathsf{ct}_2)$. Using $\mathsf{ORE.Compare}(\mathsf{ct}_1, \cdot)$, it performs a binary search over the contents of its database st to find the indices of the elements in st equal to ct_1. It removes the entries at the matching indices and outputs the updated database st'.

Correctness. By correctness of the ORE scheme, the state st maintained by the server after each query is a (sorted) list of right encryptions (under sk) of the values in the database $\boldsymbol{D}$ after the corresponding insertions and deletions. Thus,

the response r returned by the server to the client in a range query for the range $[x, y]$ is precisely the subset of ciphertexts whose plaintext values fall in the range $[x, y]$. Correctness follows by correctness of ORE decryption (which in turn follows from correctness of the ORE scheme).

Additional properties. In addition to the core security and correctness properties that we want from a symmetric range query scheme, we also note several useful properties that our construction Π_{rq} achieves for handling efficient range queries in our client-server model.

- **Stateless client and single-round protocols.** The client does not need to maintain state between queries, and each query is a single round trip between the client and the server. Our protocol achieves optimal round complexity.
- **Short query tokens**. The size of each query token t is asymptotically optimal. They are approximately the same length as the inputs used to generate the query, and *independent* of the size of the database.
- **Fast responses**. The running time of the server's algorithms is *sublinear* (logarithmic) in the total number of elements in the database.

In the full version [43], we also describe how our techniques can be extended to databases with multiple columns.

Online security. Here, we give a informal characterization of the online leakage of our range query scheme. In the full version [43], we give a complete and formal specification of the scheme's leakage.

In our description below, we refer to the leakage function $\mathcal{L}^{(d)}_{\text{BLK}}(m_1, m_2)$ as the "ORE leakage" between two equal-length values m_1 and m_2. Informally, the "ORE leakage" in our setting is the ordering of m_1 and m_2 and the index of the first differing digit in the d-ary representation of m_1 and m_2. Our range query leakage function $\mathcal{L}_{\text{RQ}}$ then takes as input the database $\boldsymbol{D} = (d_1, \ldots, d_M)$, and a sequence of ℓ queries $\mathsf{q}_1, \ldots, \mathsf{q}_\ell$ and outputs:

- For each $i \in [M]$ and $j \in [\ell]$, the ORE leakage between each database value d_i and query q_j. For a range query of the form $\mathsf{q} = (x, y)$, this includes the ORE leakage between both pairs (d_i, x) and (d_i, y) for $i \in [M]$.
- For each query q_i, and each insertion or deletion query q'_j, the ORE leakage between q_i and q'_j. Similarly, for a range query of the form $\mathsf{q}_i = (x_i, y_i)$, this include the ORE leakage between both pairs (x_i, q'_j) and (y_i, q'_j).

Roughly speaking, our range query scheme reveals the ordering and the index of the first differing digit between every query and every message in the database. We also leak some information between range queries and insertion/deletion queries. We formalize these notions in the full version.

Offline security. Offline security (Definition 5.2) of our range query scheme Π_{rq} follows directly from the fact that the encrypted database stored on the server only contains a collection of right ciphertexts, which are simulatable given just the size of the collection (that is, the right ciphertexts are *semantically secure* encryptions of their values). We give the formal proof in the full version [43].

Robustness against offline inference attacks. Offline security for our protocol implies that the contents of the server's database are *always* semantically secure. Consequently, ciphertext-only inference attacks, such as those studied by Naveed et al. [46], do not directly apply.

In their model [46, §4.2], an attacker is able to obtain access to the "steady state" of an encrypted database, which describes the database in a state that includes all auxiliary information that is needed to perform encrypted searches efficiently. In our scheme, no such auxiliary information is needed on top of the ORE scheme, and yet we are still able to achieve offline security. In contrast, in other existing PPE-based schemes, comparisons are enabled by a underlying layer of OPE encryption, which is vulnerable to inference attacks. Thus, even though these schemes can be modified to satisfy our notion of offline security, their "steady-state" representation is in the form of OPE ciphertexts which are vulnerable to inference attacks. Our scheme achieves robustness against these ciphertext-only inference attacks because our steady-state representation is precisely our offline representation. Finally, we note that we can always add additional layers of encryption (e.g., onion encryption [51]) without compromising the security of our range query scheme, which can serve as a useful countermeasure against general adversaries.

6. IMPOSSIBILITY RESULT FOR OPE

In the full version of this paper [43], we give matching upper and lower bounds for stateless OPE schemes that satisfy the notion of best-possible semantic security. Our results strengthen the lower bound given by Boldyreva et al. [9, 10], and can be stated informally as follows:

THEOREM 6.1 (Informal). *There are no efficient order-preserving encryption schemes that satisfy best-possible security on a message space containing at least* 3 *elements.*

Our lower bound shows that even in the small-domain setting, strong security is only possible with ORE, and not OPE.

7. EXPERIMENTAL EVALUATION

To assess the practicality of our order-revealing encryption scheme from Section 4, we give a full implementation of our scheme and measure its performance on a wide range of parameter settings. We then compare the performance against the Boldyreva et al. [9] OPE scheme and the Chenette et al. [22] ORE scheme. In our implementation, we use the technique from Remark 3.1 to shrink the ciphertexts.

Instantiating primitives. Our implementation is entirely written in C. We operate at 128-bits of security ($\lambda = 128$). We instantiate the PRF with AES-128. To construct a PRP on $2d$-bit domains (for $d < 128$), we use a 3-round Feistel network using a PRF on d-bit inputs [44]. In our experiments, we only consider $d < 128$, and thus, can instantiate the PRF using AES (where the d-bit input is padded to 128-bits). For the random oracle, we consider two candidate constructions. In the first, we use SHA-256, a standard cryptographic hash function commonly modeled as a random oracle.

For our second instantiation of the random oracle, we use an AES-based construction. This allows us to leverage the AES-NI instruction set for hardware-accelerated evaluation of AES. Recall from Section 4 that our construction requires a random oracle mapping from a domain $\{0,1\}^{2\lambda} = \{0,1\}^{256}$

to $\mathbb{Z}_2$ (after applying the modification from Remark 3.1). On an input $(k, x) \in \{0,1\}^{128} \times \{0,1\}^{128}$, we take the output of the random oracle to be the least significant bit of $\mathrm{AES}(k, x)$. Certainly, if we model AES as an ideal cipher, then this construction implements a random oracle. We note that modeling AES as an idealized object such as a random permutation or an ideal cipher has been used in many other recent works such as constructing efficient garbling schemes [5] or the Simpira family of permutations [36].

In our implementation, we use the OpenSSL [55] implementations of AES and SHA-256 as well as the GMP [35] library for big integer arithmetic. Our full implementation contains approximately 750 lines of code. For our implementation of Boldyreva et al.'s OPE scheme, we use the C++ implementation from CryptDB [51],[6] and for our implementation of Chenette et al.'s ORE scheme, we use the C implementation FastORE.[7] In our benchmarks, we substitute AES for HMAC as the underlying PRF used in the FastORE library. We believe this provides a more balanced comparison of the performance tradeoffs between the Chenette et al. scheme and our new ORE scheme.

Benchmarks and evaluation. We run all of our experiments on a laptop running Ubuntu 14.04 with a 2.3 GHz Intel Core i7 CPU (Haswell microarchitecture) and 16 GB of RAM. Although our encryption algorithm is easily parallelizable, we do not leverage parallelism in our benchmarks. The processor supports the AES-NI instruction set, hence our decision to base as many primitives as possible on AES. Our micro-benchmarks for encrypting and comparing 32-bit integers are summarized in Table 1. In Figure 2, we compare the cost of encryption for the different schemes across different-sized message spaces.

From Table 1, the time needed to compare two ORE ciphertexts is similar to the time needed to compare two integers (in the OPE setting). Thus, while it is the case that deploying ORE in encrypted database systems would require implementing a custom comparator in the database management system, in practice, this incurs a very small computational overhead.

Compared to OPE, our new ORE scheme is significantly faster. For instance, when processing byte-size blocks, encrypting a single 32-bit value requires just over 50 μs of computation and is over 65 times faster compared to vanilla OPE. Even our SHA-256-based implementation is about 10x faster compared to OPE. Moreover, as shown in [22, Remark 2.6 and §4], an ORE scheme which leaks the first bit that differs between two encrypted messages is provably more secure than any OPE scheme which behaves like a truly random order-preserving function. Since our new ORE scheme leaks strictly less information than the Chenette et al. scheme, we conclude that our new ORE scheme is both more secure and faster compared to OPE schemes. Of course, when compared to the bit-by-bit construction of [22], our new ORE scheme is much slower. However, in exchange, our new ORE scheme confers stronger security as well as lends itself nicely towards a range query system that provides robustness against inference attacks.

One of the main limitations of our new ORE scheme is the increase in the ciphertext size. Both OPE and the Chenette et al. ORE schemes are able to achieve ciphertexts

[6] https://github.com/CryptDB/cryptdb

[7] https://github.com/kevinlewi/fastore

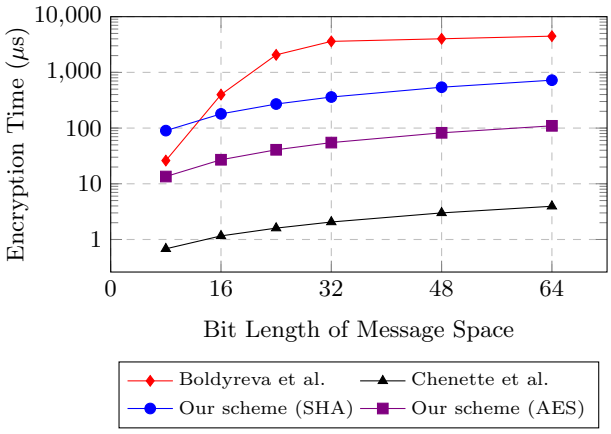

Figure 2: Performance comparison between our ORE scheme (Section 4) and existing OPE and ORE schemes. We use a fixed base representation $d = 8$ for our ORE scheme in these experiments. The two variants of our scheme, labeled SHA and AES, refer to how we instantiate the random oracle in our construction.

where the overhead is an additive or (small) multiplicative factor in the length of the messages. In our setting, because our main construction relies critically on a small-domain ORE scheme that offers best-possible security, and the existing small-domain ORE scheme have ciphertexts that grow linearly in the size of the message space, the size of the ciphertexts in our composed scheme grows quickly in the block size. Nonetheless, when encrypting byte-by-byte, encrypting a 32-bit integer requires just 224 bytes, which is quite modest for many practical applications. An interesting direction for future work is to construct a more compact small-domain ORE with best-possible security. Such a construction can be extended to a large-domain ORE with shorter ciphertexts by applying our techniques from Section 4.

8. RELATED WORK

In this section, we survey some of the literature on order-revealing and order-preserving encryption, as well as the existing work on searching over encrypted data.

OPE and ORE. The concept of order-preserving encryption was first introduced by Agrawal et al. [2], who explored the application of OPE for performing encrypted database queries. The first explicit OPE construction was formalized in the seminal work of Boldyreva et al. [9], and has subsequently been expanded on in a multitude of works [10, 48, 50, 54, 41, 40, 45, 52, 8]. Some of these works [10, 54] have focused on exploring the security properties of order-preserving encryption. Others [50, 41, 40, 52, 8] have considered stateful or interactive OPE solutions which avoid both the lower bounds in [9, 10, 50] as well as our strengthened lower bound from Section 6. However, synchronizing state and coordinating multi-round interactions in distributed, large-scale execution environments is often difficult, and consequently, nearly all existing OPE deployments (e.g., Sky-High Networks, CipherCloud) use stateless variants of OPE for sorting and filtering on encrypted data. Numerous ad hoc OPE schemes [7, 38] have also been proposed in recent years, but they often lack a formal security analysis.

Scheme	d	Encrypt	Compare	$\lvert\mathsf{ct}\rvert$	Leakage
Boldyreva et al. OPE [9]	–	3601.82 μs	0.36 μs	8 bytes	(Hard to quantify)
Chenette et al. ORE [22]	1	2.06 μs	0.48 μs	8 bytes	First bit that differs
Our ORE scheme (RO: SHA-256)	4	54.48 μs	0.38 μs	192 bytes	First block of d-bits that differs
	8	361.04 μs	0.98 μs	224 bytes	
	12	4370.64 μs	3.20 μs	1612 bytes	
Our ORE scheme (RO: AES)	4	16.50 μs	0.31 μs	192 bytes	First block of d-bits that differs
	8	54.87 μs	0.63 μs	224 bytes	
	12	721.37 μs	2.61 μs	1612 bytes	

Table 1: Performance comparison between our ORE scheme from Section 4 and existing OPE and ORE schemes. We consider two variants of our scheme: one where the random oracle is instantiated using an AES-based construction and one where the random oracle is instantiated with SHA-256. We describe these two instantiations in greater detail in Section 7. In these benchmarks, we use a 32-bit plaintext space, and measure the time needed to encrypt a (randomly chosen) message and the time needed to compare two ciphertexts. The parameter d is the block size (in bits) in our ORE scheme. Our micro-benchmarks are averaged over 50–10^7 iterations (the precise number is adjusted based on the approximate runtime of the algorithm).

The notion of order-revealing encryption was first introduced by Boneh et al. [12], who gave a construction from multilinear maps that satisfies best-possible security. More generally, ORE is a special case of multi-input functional encryption (MIFE) [33]. To date, the only constructions of general-purpose MIFE rely on heavy primitives such as indistinguishability obfuscation [4, 28] and are far too inefficient to deploy. Chenette et al. [22] recently proposed an efficient ORE scheme, which we improve upon and generalize in our work. In the small-domain setting, it is possible to construct ORE from either symmetric or public-key encryption [3, 16] or bilinear maps [42], but these constructions are far less efficient compared to our small-domain ORE from Section 3, which just relies on PRFs.

Searching on encrypted data. Numerous techniques, such as searchable symmetric encryption (SSE) [53, 24, 20], property-preserving encryption (PPE) [9, 48, 21], fully homomorphic encryption (FHE) [29], hidden vector encryption [15], oblivious RAMs (ORAM) [32], and others have been proposed for tackling the general problem of searching and querying on encrypted data. While tools such as FHE or ORAM can be used for searching on encrypted data [11, 57], these methods are prohibitively expensive for nearly all real-world deployments. On the more practical side, numerous SSE schemes [53, 30, 19, 24, 20, 37, 47] have been proposed in the last 15 years, but these past works are limited to exact keyword searches, and generally do not handle the efficient computation of complex queries (such as *range queries*) over encrypted data. More recently, several works [18, 17, 49, 25] describe constructions of SSE schemes that are able to handle more expressive queries. We survey these works below.

Cash et al. [18] give the first SSE scheme that supports Boolean queries (in time sublinear in the size of the database) with a small amount of leakage and security from the decisional Diffie-Hellman (DDH) assumption. Subsequently, Cash et al. [17] extend the construction to allow for updates to the encrypted database as well as support multiple, potentially dishonest clients. Handling updates requires the client to maintain a small amount of state (or requires additional rounds of communication and leads to increased leakage). Boolean queries alone, however, do not suffice for range queries, so in another follow-up work, Faber et al. [25] show how the Cash et al. SSE scheme can be leveraged for range queries. Their resulting construction leaks some additional information about the database contents, namely the number of values that fall into certain subintervals within the requested range. Moreover, due to the use of universal covers, the size of the server's response set to a range query may be up to 66% larger than the size of the true response set. We do not know of any existing SSE scheme that can efficiently support range queries with optimal (minimal) leakage.

Concurrent to the work of Cash et al., Pappas et al. [49] introduce BlindSeer, a private database management system that can support a wide-range of queries in sublinear time over an encrypted database. Their construction leverages generic two-party computation tools such as Yao's garbled circuits [56], and their construction provides security in the semi-honest model.

Comparison to our techniques. To conclude, we highlight some of the key differences between existing SSE methods and our ORE-based construction for implementing range queries over an encrypted database:

- Like other PPE-based constructions, our ORE-based construction integrates well with *existing* database management systems—we just need to implement a custom comparator. With SSE, we would have to deploy a new, and oftentimes, complex database management system. This lacks legacy compatibility, which is a barrier to deployment in existing systems. Our approach provides a fast, simple, and direct solution for supporting range queries on encrypted data *without* requiring significant infrastructural changes.
- We explicitly model and analyze the leakage of our range query protocol assuming adaptive updates to the database.
- Our construction only requires symmetric primitives and does not require more expensive primitives such as public-key cryptography or oblivious transfer.

Acknowledgments

We thank Dan Boneh, Mark Zhandry, and Joe Zimmerman for insightful discussions about this work. We thank the members of the 2015 Stanford Theory Retreat for initiat-

ing our study of new OPE lower bounds. This work was supported by the NSF, DARPA, the Simons foundation, a grant from ONR, and an NSF Graduate Research Fellowship. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

9. REFERENCES

[1] R. Abelson and J. Creswell. Data breach at anthem may forecast a trend. *The New York Times*, 2015.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *ACM SIGMOD*, 2004.

[3] P. Ananth and A. Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 2012.

[5] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE SP*, 2013.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[7] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *ACM SIGMOD*, 2009.

[8] T. Boelter, R. Poddar, and R. A. Popa. A secure one-roundtrip index for range queries. Cryptology ePrint Archive, Report 2016/568, 2016.

[9] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.

[10] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[11] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In *ACNS*, 2013.

[12] 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 *EUROCRYPT*, 2015.

[13] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.

[14] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 2003.

[15] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.

[16] Z. Brakerski, I. Komargodski, and G. Segev. From single-input to multi-input functional encryption in the private-key setting. *IACR Cryptology ePrint Archive*, 2015.

[17] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, 2014.

[18] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.

[19] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.

[20] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594, 2010.

[21] S. Chatterjee and M. P. L. Das. Property preserving symmetric encryption revisited. In *ASIACRYPT*, 2015.

[22] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *FSE*, 2016.

[23] J. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, 2013.

[24] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, 2006.

[25] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS*, 2015.

[26] J. Finkle and D. Volz. Database of 191 million u.s. voters exposed on internet: researcher. *Reuters*, 2015.

[27] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[28] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[29] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[30] E. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003.

[31] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 1986.

[32] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 1996.

[33] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In *EUROCRYPT*, 2014.

[34] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 1984.

[35] T. Granlund and the GMP development team. GNU MP: The GNU Multiple Precision Arithmetic Library. http://gmplib.org/, 2012.

[36] S. Gueron and N. Mouha. Simpira v2: A family of efficient permutations using the AES round function. *IACR Cryptology ePrint Archive*, 2016.

[37] S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM CCS*, 2013.

[38] H. Kadhem, T. Amagasa, and H. Kitagawa. A secure and efficient order preserving encryption scheme for relational databases. In *KMIS*, 2010.

[39] G. Kelly. ebay suffers massive security breach, all users must change their passwords. *Forbes*, 2014.

[40] F. Kerschbaum. Frequency-hiding order-preserving encryption. In *ACM CCS*, 2015.

[41] F. Kerschbaum and A. Schröpfer. Optimal average-complexity ideal-security order-preserving encryption. In *ACM CCS*, 2014.

[42] S. Kim, K. Lewi, A. Mandal, H. W. Montgomery, A. Roy, and D. J. Wu. Function-hiding inner product encryption is practical. *IACR Cryptology ePrint Archive*, 2016.

[43] K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. *IACR Cryptology ePrint Archive*, 2016:612, 2016.

[44] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 1988.

[45] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, and R. Canetti. Modular order-preserving encryption, revisited. In *ACM SIGMOD*, 2015.

[46] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM CCS*, 2015.

[47] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *IEEE SP*, 2014.

[48] O. Pandey and Y. Rouselakis. Property preserving symmetric encryption. In *EUROCRYPT*, 2012.

[49] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. Bellovin. Blind seer: A scalable private DBMS. In *IEEE SP*, 2014.

[50] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE SP*, 2013.

[51] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *ACM SOSP*, 2011.

[52] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich. POPE: partial order-preserving encoding. *IACR Cryptology ePrint Archive*, 2015.

[53] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE SP*, 2000.

[54] I. Teranishi, M. Yung, and T. Malkin. Order-preserving encryption secure beyond one-wayness. In *ASIACRYPT*, 2014.

[55] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, 2003.

[56] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982.

[57] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *CCSW*, 2013.