Modular Order-Preserving Encryption, Revisited

Charalampos Mavroforakis
Boston University
cmav@cs.bu.edu

Nathan Chenette
Rose-Hulman Institute of Technology
chenett1@rose-hulman.edu

Adam O'Neill
Georgetown University and NIST
adam@cs.georgetown.edu

George Kollios
Boston University
gkollios@cs.bu.edu

Ran Canetti
Boston University
canetti@bu.edu

ABSTRACT

Order-preserving encryption (OPE) schemes, whose ciphertexts preserve the natural ordering of the plaintexts, allow efficient range query processing over outsourced encrypted databases without giving the server access to the decryption key. Such schemes have recently received increased interest in both the database and the cryptographic communities. In particular, *modular* order-preserving encryption (MOPE), due to Boldyreva *et al.* [8], is a promising extension that increases the security of the basic OPE by introducing a secret modular *offset* to each data value prior to encrypting it. However, executing range queries via MOPE in a naïve way allows the adversary to learn this offset, negating any potential security gains of this approach.

In this paper, we systematically address this vulnerability and show that MOPE can be used to build a practical system for executing range queries on encrypted data while providing a significant security improvement over the basic OPE. We introduce two new query execution algorithms for MOPE: our first algorithm is efficient if the user's query distribution is well-spread, while the second scheme is efficient even for skewed query distributions. Interestingly, our second algorithm achieves this efficiency by leaking the *least-important* bits of the data, whereas OPE is known to leak the *most-important* bits of the data. We also show that our algorithms can be extended to the case where the query distribution is adaptively learned online. We present new, appropriate security models for MOPE and use them to rigorously analyze the security of our proposed schemes. Finally, we design a system prototype that integrates our schemes on top of an existing database system and apply query optimization methods to execute SQL queries with range predicates efficiently. We provide a performance evaluation of our prototype under a number of different database and query distributions, using both synthetic and real datasets.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2749455 .

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption

Keywords

order preserving encryption; range queries; database encryption; database security model

1. INTRODUCTION

Cloud computing and modern networking and virtualization infrastructures have made the idea of database outsourcing to a third party not only a possibility, but sometimes a necessity. Following the seminal papers on database outsourcing [20, 19], a number of systems have been proposed [4, 26, 18, 11, 1, 10, 12] that use cloud services for this task. However, despite the enormous benefits of this approach, it has been recognized that there are a number of important issues that need to be addressed for it to be viable, the most critical of which is security. There have been three main issues related to security in this context: confidentiality, integrity (verifiability), and availability [2].

In this paper, we focus on the confidentiality problem, i.e., how a client can keep the database contents private from the third party. A typical approach is that the client encrypts each record before sending it to the third party. Ideally, one would like to use an encryption scheme that provides very strong security guarantees, such as semantic security [17], while maintaining the ability to execute queries. Remarkable recent work on fully homomorphic encryption has shown that, in theory, we can in fact build such encryption schemes with polynomial-time overhead [14]. However, while this is an active area of research, such schemes remain far from practical and are not expected to be anywhere close to being used in real systems in the near future.

To this end, new encryption schemes have been proposed that, although providing weaker security guarantees, allow for much more efficient query processing. In this paper we consider one of the most important such schemes, order-preserving encryption (OPE) [3, 7, 8, 27]. In an OPE scheme, any two values x and y with a natural order, e.g., $x < y$, get encrypted as $Enc(x) < Enc(y)$. In other words, the encryption scheme preserves the original order of the plaintexts. This is a very useful primitive because it allows the database system to make comparisons between the ciphertexts and still get the same results as if it had operated on the plaintexts. Therefore, the database system can still

build index structures, like B+-trees, on the encrypted attributes to efficiently answer exact match and range queries, the same way as with un-encrypted data.

We focus on a promising extension to OPE introduced by Boldyreva *et al.* [8] called *modular* OPE (MOPE). The idea behind MOPE is simple: a secret *modular offset* j is included in the secret key and the encryption of x becomes $MOPE(x) = OPE(x + j)$, where OPE is the underlying OPE scheme. At a first glance, this seems to provide a substantial improvement in security — given just the encrypted database, the adversary cannot deduce any information about the locations of the underlying plaintexts. In constrast, basic OPE leaks about half of the most-important bits of the locations [8]. We want to point out this is in fact crucial for applications where a database table contains a column that takes consecutive values, e.g., a date. In this case, the plaintexts might cover the complete domain and if their order is revealed, so are their values. (This is actually the case for some of the attributes in the TPC-H benchmark that we used in the experiments.)

However, there is a catch: As Boldyreva *et al.* [8] note, if the adversary, i.e., the third party database server, observes the user's queries, they can divulge the secret offset j and negate any security gain. This is because the queries will never intersect the interval in the ciphertext-space between the encryptions of the last and the first plaintext value. This gap (see Figure 1) allows the adversary to orient the space and get a better estimate of j.

1.1 Contributions

In this paper, we revisit *modular* OPE and investigate the security and efficiency of systems that use it to execute range queries on encrypted data. The goal of our schemes is to hide the user's query distribution by *mixing* it with another distribution. Our main contributions are summarized below.

Uniform Query Algorithm: In our first scheme, the perceived query distribution from the server's point of view is uniform across all possible queries, including "wrap around" queries. Specifically each time the user makes a query, our scheme will repeatedly flip a biased coin to decide whether to execute it or sample a fake query from an appropriate *completion* distribution, repeating until the former happens. The coin's bias and the completion distribution are carefully chosen so that the expected number of fake queries is minimized, while at the same time ensuring that the perceived query distribution is uniform. As a result, we perform about μM additional fake queries for every real query, where μ is the maximum probability of any particular query and M is the domain size.

For better efficiency, our scheme also transforms every range query to a set of one or more queries with some fixed size $k \leq M$. In this case, the space requirement for storing the query distribution is M values instead of M^2. As shown by our security analysis, the scheme completely hides the *location* of the plaintext values, however still reveals the $\log(M + qk)/2$ most-significant bits of the distance between any two plaintexts, where q bounds the total number of queries executed by the system.

Periodic Query Algorithm: Our first scheme becomes inefficient if the user's query distribution is highly skewed. For example, if μ is a large constant, the system performs roughly M fake queries for every user query, which is obviously prohibitive. Our second scheme addresses this inefficiency by mixing the user's query distribution with a different completion distribution, such that the perceived query distribution is ρ-*periodic* instead of uniform. In a ρ-periodic distribution the probability of a query depends only on its congruence class modulo ρ — recall that we transform queries into a set of queries with fixed size k, so in effect each query is represented by a single domain element —, where ρ is a parameter of the system. Again, by carefully choosing the coin's bias and the ρ-periodic completion distribution, we minimize the expected number of fake queries. Namely, the expected number of fake queries is $\mu_\rho M$, where μ_ρ is the average over the congruence classes modulo ρ of the maximum probability of a query in each class. As $\mu_\rho \leq 1/\rho$, this quantity can be small even when μ is high, resulting in much better efficiency than our first scheme in such a case. As shown by our security analysis, the scheme achieves this efficiency by revealing $\log \rho$ least-significant bits of the data. Tuning ρ balances between security and efficiency.

Learning the Query Distribution: A drawback of the above schemes is that, in order to calculate the appropriate coin bias and completion distribution, they both assume that the system knows the user's query distribution *a priori*. We show that they can be extended to learn this distribution *adaptively online*.

The key idea is to maintain a buffer of the user's queries and, at each step, use it to estimate their query distribution based on the queries seen so far. Then, each time a query is to be executed, the coin's bias and completion distribution are updated to reflect this estimate. As the user makes more queries, this estimate improves and so does the performance of the system. While the efficiency will be low in the beginning (for example, after the user makes the first query, the system estimates that the query distribution is entirely concentrated on this point and, almost always, runs a fake query), it improves over time and converges to the performance of the schemes discussed above. Remarkably, our experiments show that, for many real world datasets, convergence happens very quickly.

System Prototype and Evaluation: We develop a simple system prototype that implements the proposed schemes. In our system architecture, we have three parties: the client (it could be more than one), the proxy, and the server, as show in Figure 4. The client issues queries to the proxy, which processes them (and buffers them, in the online adaptive case) and issues real and fake queries to the server according to one of our schemes. The responses that are relevant to the client's queries are forwarded by the proxy to the client and the rest are dropped. We use a PostgreSQL system as a back-end database system. Furthermore, an important advantage of our approach (similar to CryptDB [31]) is that the underlying database server does not need to be modified and we can use any existing DBMS to execute the actual queries. Therefore, we can take advantage of query optimization techniques at the back-end to process many fake queries together very efficiently.

We provide a comprehensive evaluation, where we study experimentally how our prototype performs under a number of different database and query distributions. The experiments use datasets and SQL queries from the TPC-H benchmark. The results show that the proposed approaches are quite practical.

Security Analysis: In order to analyze the security of our schemes, we propose new security definitions that allow the adversary to see both the encrypted database *and* the user's encrypted queries. Previous security models for OPE did not make this distinction and essentially just allowed the adversary to see the encrypted database. Our security definitions extend those of [8] to capture both the *location privacy* (how much about the plaintext location is revealed) and *distance privacy* (how much about the pairwise distances between the plaintexts is revealed).

Under our new definitions, we show that MOPE with our uniform query algorithm achieves perfect location security, while in the worst case leaking the $\log(M+qk)/2$ high-order bits of the plaintext distances. Despite the fact that the perceived distribution of the queries is uniform, the analysis here is non-trivial because, as explained above, we transform the queries to have a fixed size k. This means that the adversary gets to see encryptions of plaintexts that it knows are distance k apart, a situation not considered in prior analyses of OPE. We also show that MOPE with our periodic query algorithm leaks at most the $\log \rho$ least-important bits of the data. These results also hold in the case where the user's query distribution is learned adaptively online. This means that learning the query distribution affects only the efficiency and not the security of the system.

These security results should be contrasted with the basic OPE, which was shown to leak half of the most-important bits of both the plaintext values and their pairwise distances [8]. In particular, we want to draw the reader's attention to the fact that using MOPE with our periodic query algorithm only leaks the *least-important* bits of the data, which seems much more desirable.

2. BACKGROUND ON (MODULAR) OPE

For a randomized algorithm A, we denote by $y \leftarrow\!\!\$\, A(x)$ that y is the output of running A on input x and a fresh random tape. When A is deterministic, we denote this by $y \leftarrow A(x)$ instead. All algorithms in the paper are required to be efficient unless otherwise mentioned. For a number n we denote by $[n]$ the set $\{1, \ldots, n\}$.

2.1 Order-Preserving Encryption

An *order-preserving symmetric encryption* (OPE) scheme with plaintext-space $[M]$ and ciphertext space $[N]$ is a tuple of algorithms $\mathsf{OPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ where:

- The randomized key-generation algorithm Kg outputs a key K.
- The deterministic encryption algorithm Enc on inputs a key K and a plaintext m outputs a ciphertext c.
- The deterministic decryption algorithm Dec on inputs a key K and a ciphertext c outputs a plaintext m.

In addition to the usual correctness requirement that

$$\mathsf{Dec}(\mathsf{Enc}(K, m)) = m$$

for every plaintext m and key K, we require that

$$m_1 \le m_2 \text{ if and only if } \mathsf{Enc}(K, m_1) \le \mathsf{Enc}(K, m_2)$$

for all plaintexts m_1, m_2 and every key K.

For notational convenience, we extend encryption notation to sets. That is, if X is a set then $\mathsf{Enc}(K, X)$ denotes the set $\{\mathsf{Enc}(K, x) \mid x \in X\}$.

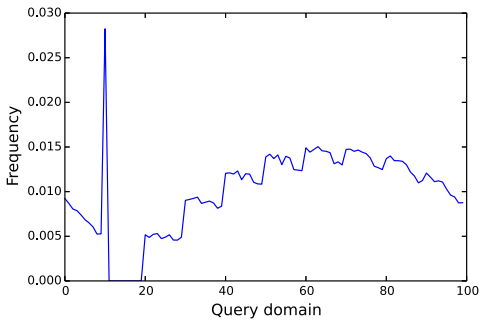


Figure 1: The gap in the query distribution reveals the displacement.

2.2 Modular Order-Preserving Encryption

A *modular order-preserving encryption* (MOPE) scheme is an extension to OPE that increases its security. Instead of defining such a scheme in general, we define a transformation to obtain it from a given OPE scheme.

THE TRANSFORMATION. Let $\mathsf{OPE} = (\mathsf{Kg}', \mathsf{Enc}', \mathsf{Dec}')$ be an OPE scheme. We define the associated *modular OPE scheme* $\mathsf{MOPE}[\mathsf{OPE}] = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ where

- Kg generates $K \leftarrow\!\!\$\, \mathsf{Kg}'$ and $j \leftarrow\!\!\$\, [M]$; it outputs (K, j).
- Enc on inputs a key K and a plaintext m outputs $\mathsf{Enc}'(K, m + j \bmod M)$.
- Dec on inputs a key K and a ciphertext c outputs $\mathsf{Dec}(K, c) - j \bmod M$.

Above, the value j in the secret key of $\mathsf{MOPE}[\mathsf{OPE}]$ is called the *secret offset* or *displacement*.

3. NEW QUERY ALGORITHMS FOR MOPE

The Problem: With MOPE, as is also the case for OPE, it is very easy for the client to execute range queries: to make a range query $[m_L, m_R]$ for which $1 \le m_L \le m_R \le M$, the client computes the pair $(c_L, c_R) = (\mathsf{Enc}(K, m_L), \mathsf{Enc}(K, m_R))$ and sends it to the server. The server then returns all database ciphertexts found in the range $[c_L, c_R]$. Note that in an MOPE scheme, this ciphertext interval could "wrap around" the space, i.e., if $c_R < c_L$ then the server returns database ciphertexts found in $[c_L, N] \cup [1, c_R]$. For simplicity, we will use the notation $[c_L, c_R]$ independent of whether or not the interval wraps around.

However, as pointed out by [8], there is a security vulnerability introduced by making range queries with MOPE. Note that all valid range queries, i.e., those $[m_L, m_R]$ for which $1 \le m_L \le m_R \le M$, when encrypted may "cover" every value in the ciphertext-space $[N]$ *except* from those ciphertexts lying between $\mathsf{Enc}(K, M)$ and $\mathsf{Enc}(K, 1)$. Therefore, after observing enough queries, the adversary can get a better idea of where this gap lies, increasing its probability to predict j. In Figure 1 we show the histogram of the start values of a set of range queries for a small domain and displacement $j = 20$. In particular, we assume that the domain (and the range) is $[0, 100]$ and that all possible valid range queries with length 10 ($k = 10$) are generated and executed. In that case, the adversary will observe that there are no

queries that start between the values 10 and 20, which correspond to the end of the domain before the displacement. Therefore, it can easily infer that the displacement is 20.

One natural step toward avoiding this attack is to introduce *wrap-around* queries [8]. A wrap-around range query corresponds to a pair m_L, m_R for which $m_R < m_L$. The desired interval of values wraps around the space, and is $[m_R, M] \cup [1, m_L]$. MOPE schemes naturally support wrap-around range queries in the same manner as standard range queries. These queries are not practically useful, but can be used as "dummy queries" in fooling the gap attack. However, [8] did not rigorously analyze the attack or explain how to implement dummy queries. Moreover, the one-wayness bounds of [8] did not consider the effect of observed range queries. A formal security model, constructions, and analysis are needed to address this scenario, and we take this as the starting point of our work.

3.1 Uniform Query Algorithm

The idea behind our basic query algorithm is as follows. We suppose that the user's queries come from some distribution $\mathcal{Q}$. Each time we need to make a query, we flip an α-biased coin, which returns heads with probability α, to decide whether to make a query from $\mathcal{Q}$ or from some other distribution $\tilde{\mathcal{Q}}$. We will choose α and $\tilde{\mathcal{Q}}$ such that the convex combination $\alpha\mathcal{Q} + (1-\alpha)\tilde{\mathcal{Q}} = \mathcal{U}$, where $\mathcal{U}$ is the uniform distribution on the *entire* query space considered, including wrap-around queries in the case of MOPE). This solves two problems: (1) $\mathcal{Q}$ may not be uniform, even on standard (non-wrap-around) queries, invalidating the techniques for the security analysis developed in [8], and (2) $\mathcal{Q}$ depends on the secret offset, while $\mathcal{U}$ clearly does not, and consequently hides all information about it.

Completion of a distribution: To define our algorithm formally, we start with notation and a definition. Let $\mathcal{D}$ be a probability distribution on a finite set S. Define $\mu_{\mathcal{D}} = \max_{i \in S} \mathcal{D}(i)$, and define the *completion* of $\mathcal{D}$, denoted $\tilde{\mathcal{D}}$, as

$$\tilde{\mathcal{D}}(i) = \frac{\mu_{\mathcal{D}} - \mathcal{D}(i)}{\mu_{\mathcal{D}}|S| - 1}$$

for all $i \in S$.

Representation of the query distribution: To represent the query distribution we use a histogram on the query domain. However, since each query can have different start location and length, to represent all possible queries, we need to keep $O(M^2)$ values. In order to address this problem, we pick a fixed query length $k \geq 1$ and we decompose every query into a set of queries with length k. So, if an original user query has length smaller than k, we use a single fixed query that starts at the same location as the original query. On the other hand, if the original user query is larger than k, we split the query into a set of range queries of size k, where again the first fixed query starts from the start location of the original query. This approach guarantees that we need only $O(M)$ values to represent the query histogram. Notice that if a user's query q is decomposed into multiple fixed length queries, the results of all the decomposed queries should be returned to the user.

The query algorithm: Now we are ready to describe the algorithm, which we call QueryU ("U" for uniform), which processes a new query q. The algorithm is initialized with the completion distribution $\tilde{\mathcal{Q}}$, the maximum query probability $\mu_{\mathcal{Q}}$, and a fixed length query k. We denote by $\mathsf{Bern}(\alpha)$ the Bernoulli distribution with parameter α, i.e., $\mathsf{Bern}(\alpha) = 1$ with probability α. Note that the input to the algorithm is just the start location of the query, since the query has a fixed length.

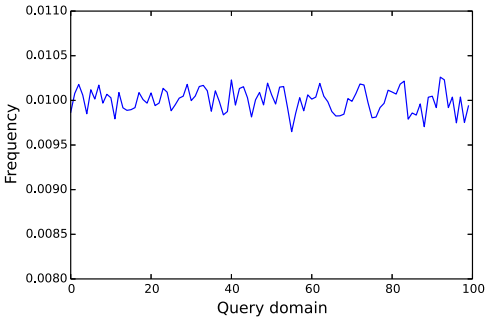


Figure 2: The query distribution after we add the fake queries. The real queries are obfuscated and the displacement gap is hidden.

Algorithm $\mathsf{QueryU}(q)$:
Until q is executed do:
$\mathsf{coin} \leftarrow\!\!\$\ \mathsf{Bern}\,(1/(\mu_{\mathcal{Q}} M))$
If $\mathsf{coin} = 1$ then execute q
Else $(\mathsf{coin} = 0)$
$q_f \leftarrow\!\!\$\ \tilde{\mathcal{Q}}$; Execute q_f

Above, "executing" a query q means that the ciphertexts $\mathsf{Enc}(K, q), \mathsf{Enc}(K, q+k)$ are issued, where Enc is the encryption algorithm for the MOPE scheme and K is the encryption key. To sample from the completion distribution $\tilde{\mathcal{Q}}$ we use the inversion method, i.e., inversion sampling [13]. Again, we just have to sample the start location of the query. In Figure 2 we show the perceived query distribution after we add the fake queries. In Section 7 we formally analyze the security of our scheme.

3.2 Periodic Query Algorithm

Note that the uniform query algorithm can be very inefficient when the user's distribution $\mathcal{Q}$ is highly skewed. In particular, on expectation $\mu_{\mathcal{Q}} M \in \Omega(M)$ fake queries are needed for every user query, which is no longer sub-linear to the size of plaintext domain.

We now present a periodic query algorithm that avoids this inefficiency, at the cost of leaking the least-significant bits of the secret offset in the MOPE scheme. In fact, it is a generalization of both QueryU and the straightforward algorithm that forwards all user queries directly; it offers any level of security vs. functionality trade-off, lying between these two extremes. Furthermore, we reduce the size of the histogram that is used to generate the fake queries from M to the size of the period.

Let $\mathcal{Q}$, as before, be the user's query distribution. Our periodic algorithm works like that of Section 3 with one exception: the distribution $\tilde{\mathcal{Q}}$ is chosen so that $\alpha\mathcal{Q} + (1-\alpha)\tilde{\mathcal{Q}} = \mathcal{P}_\rho$, where $\mathcal{P}_\rho$ is a periodic distribution on the query space with period ρ, an integer that divides M. That is, $\mathcal{P}_\rho(x) = \mathcal{P}_\rho(x+\rho)$ for all $x \in [M]$, where the addition is modular, i.e. it "wraps around" the query space. In partic-

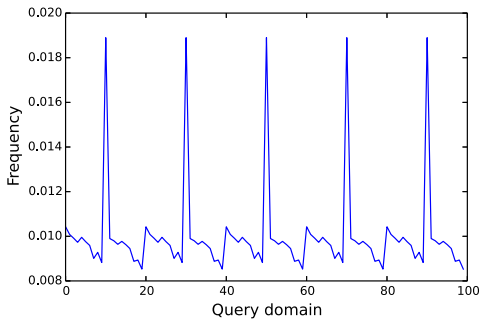


Figure 3: The periodic query distribution after we add the fake queries.

ular, we pick the appropriate $\tilde{\mathcal{Q}}$ to maximize α, which we formalize below.

ρ-periodic completion of a distribution: Let $\mathcal{D}$ be a probability distribution on a finite set $S = [M]$. Let ρ be an integer dividing M. Let sets $S_1, \ldots, S_\rho$ partition S into congruency classes modulo ρ. Define $\eta_j = \max_{i \in S_j} \mathcal{D}(i)$ for $j = 1, \ldots, \rho$ and $\eta_{\mathcal{D}} = \frac{1}{\rho} \sum_{j=1}^{\rho} \eta_j$. The *$\rho$-periodic completion* of $\mathcal{D}$, denoted $\tilde{\mathcal{D}}_\rho$, is

$$\tilde{\mathcal{D}}_\rho(i) = \frac{\eta_j - \mathcal{D}(i)}{\eta_{\mathcal{D}} M - 1}$$

for all $i \in S$.

The periodic query algorithm: The periodic query algorithm, $\mathsf{QueryP}[\rho]$, is the same as the one presented in Section 3 except that the coin is taken from $\mathsf{Bern}\,(1/(\eta_{\mathcal{Q}} M))$ and the ρ-periodic completion $\tilde{\mathcal{Q}}_\rho$ is used in place of $\tilde{\mathcal{Q}}$. In Figure 3 we show the query distribution histogram for the example that we used before using a period of size $\rho = 20$.

Choosing ρ to balance efficiency and security: Using QueryU, we expect to see $\mu_{\mathcal{Q}} M$ fake queries for every real query, where $\mu_{\mathcal{Q}}$ is the maximum probability in $\mathcal{Q}$. Using QueryP instead, we expect $\eta_{\mathcal{Q}} M$ where $\eta_{\mathcal{Q}}$ is the average of the maximum probabilities of the congruency classes of $\mathcal{Q}$ modulo ρ. As $\eta_{\mathcal{Q}} \leq 1/\rho$, the ratio of fake to real queries is at most M/ρ, so even for skewed distributions we can achieve sublinear (in M) number of fake queries per real query by taking, e.g., $\rho = \sqrt{M}$.

On the other hand, notice that the ρ-periodic algorithm cannot protect the least-significant-bit information of the secret offset, as the cumulative distribution of real and fake queries is non-uniform. An adversary that knows the user's distribution can easily construct the cumulative distribution and match this to the perceived distribution of query ciphertexts. However, the adversary cannot determine which of the ρ possible shifts of this match is correct; i.e. the adversary can guess the least-significant bits of the encryption's secret offset, but cannot guess the most-significant bits. For example, if $\rho = \sqrt{M}$ then the most-significant half of the bits of the secret offset are hidden.

Thus, ρ is a parameter that enables us to fine-tune the efficiency-security trade-off: a larger ρ results in fewer fake queries but leaks more bits of the secret shift; a smaller ρ has the opposite effects. Notice that when $\rho = 1$, there is only one congruence class and the "periodic" algorithm is just the standard algorithm. When $\rho = M$, each value has its own congruence class and no fake queries are generated, so the user's distribution is exposed. In Section 7 we formally analyze the security of this scheme.

4. LEARNING THE QUERY DISTRIBUTION

In our schemes above, we assume that the user's query distribution is known *a priori*. Now we consider whether it can be learned adaptively online. The main idea is at each point to use the queries seen so far as the estimate of the user's query distribution. A naïve algorithm for this setting would then run the QueryU (or QueryP) algorithm to process each query with the updated histogram. However, this would result in extremely inefficient performance — we would need to execute roughly all possible range queries before the first real query is even executed. Instead, we maintain a buffer of the queries seen so far, which we use to represent the histogram, and issue only a single query before updating the buffer with the next query. For example, here is our adaptive online version of QueryU, which we call $\mathsf{AdaptiveQueryU}$, to process a query q.

Algorithm $\mathsf{AdaptiveQueryU}(q)$:
  $\mathsf{buffer.add}(q)$
  Compute $\mu_{\mathcal{Q}}$ and $\tilde{\mathcal{Q}}$ from buffer
  $\mathsf{coin} \leftarrow\!\!\$\ \mathsf{Bern}\,(1/(\mu_{\mathcal{Q}} M))$
  If $\mathsf{coin} = 0$ then
    $q_f \leftarrow\!\!\$\ \tilde{\mathcal{Q}}$; Execute q_f
  Else ($\mathsf{coin} = 1$)
    $q_r \leftarrow\!\!\$\ \mathsf{buffer}$; Execute q

The above algorithm should be repeated until all of the user's queries have been executed. Note that the second line regards the buffer as a histogram, in the obvious way. Furthermore, the last line randomly selects an item from the buffer, but leaves the buffer itself unmodified. An analogous adaptive version of QueryP, call it $\mathsf{AdaptiveQueryP}$, can be obtained from the above.

In Section 7, we show that the adaptive online version of our algorithms achieve the same security guarantees as their counterparts in the basic model. For example, in the case of $\mathsf{AdaptiveQueryU}$, the idea is that each individual query executed is uniformly distributed. To see this, note that in every execution of the while loop, sampling randomly from the buffer is identical to sampling randomly from the current $\mathcal{Q}$. Ensuring this identity is why we sample randomly from the buffer (rather than taking an arbitrary element from it).

Note that at some point the algorithm could declare the user's query distribution "learned" and delete any executed queries from the buffer (without changing a separate histogram $\mathcal{Q}$) until the buffer is empty, then switching to the basic QueryU or QueryP algorithms. We leave determining such a "cross-over" point for future work.

5. SYSTEM ARCHITECTURE

We now turn to developing a concrete system for executing queries on encrypted data using our algorithms. We consider a typical architecture used in database outsourcing systems [19], including CryptDB [31]. The system consists of a set of clients, a proxy server, and a (database) server as shown in Figure 4. The database server is an un-modified DBMS (in our experiments with TPC-H described later, we used a PostgreSQL server). The proxy is a trusted party

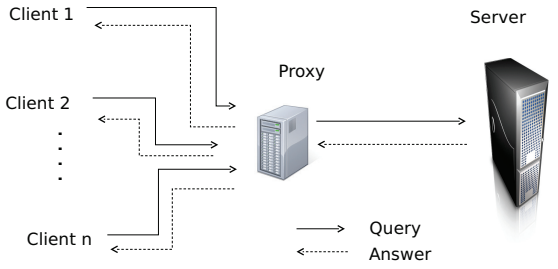


Figure 4: The architecture of the system.

and holds both the basic and the periodic completion distributions $\tilde{\mathcal{Q}}$ and $\tilde{\mathcal{Q}}_\rho$ respectively, as well as the encryption key for the MOPE scheme.

Initially, the data owner encrypts the database, using MOPE to encrypt attributes that support range query predicates and outsources the encrypted database to the untrusted database server. After that, the proxy starts sending queries (which are a mix of real and fake queries) to the server, which are executed using the DBMS, and the encrypted results are sent back to the proxy. We assume that the server executes queries correctly and returns the correct results. The proxy gets the results back, filters out the fake queries, and returns the requested results to each client. Next, we discuss how the proxy generates the fake queries and the query mix.

As we discussed before, our algorithms assume that all the queries have the same length k. However, an incoming query may have arbitrary length. Therefore, the proxy decomposes an original query q into a set of queries of size k that cover completely q. Assume that the set of these queries is $\tau_k(q)$. According to the previous algorithms we need to use a set of Bernoulli trials to generate a set of queries for each query in $\tau_k(q)$. However, we can directly compute the number of fake queries that we need each time, using the geometric distribution Geom. In particular, for each query in $\tau_k(q)$, we generate a value $\ell = \mathsf{Geom}\,(1/(\mu_{\mathcal{Q}} M))$ and sample ℓ fake queries from $\tilde{\mathcal{Q}}$ using inversion sampling. Let the union of all the fake queries be $F(q)$. Finally, the proxy permutes all the queries in $F(q) \cup \tau_k(q)$ and sends them to the server.

Of course, the server may be able to detect the real queries if the number of fake queries is small and the real queries arrive in non-regular time intervals. We thus have the proxy issue queries to the server at fixed regular time intervals. This is a standard approach to hide real from fake queries that has been used in other systems, like in a practical ORAM implementation [25].

5.1 Multiple Range Query Execution

An advantage of our MOPE schemes compared to another recently proposed scheme that offers increased security over basic OPE [30] is that we do not need to modify the underlying DBMS when we execute the range queries. Thus, any optimization that the server can use for a regular database workload can also be used here. In particular, the fake queries are just some additional queries of the same form. Therefore, a simple and very efficient optimization is to consider multiple range predicates and execute them in a single query. For example, we can have many range query predicates connected with ORs in the WHERE clause of a single SQL query statement. This is an example of multiple query optimization [32] that allows sharing index and data access across multiple range queries. We can therefore execute multiple range queries at once and return the results to the proxy. This means of course that the proxy has to filter the query results and compute the tuples that satisfy the real queries. However, this slight extra work at the proxy can be justified if the performance at the server and the communication overhead (since the same record can be shared by multiple queries) improves significantly.

5.2 Other Practical Considerations

The completion distribution $\tilde{\mathcal{Q}}$ needs $O(M)$ space to be stored. If M is not very large, this is not a problem. On the other hand, it can be a non-negligible overhead if M is large. However, notice that M is actually the effective domain of the possible values of the encrypted attribute and not the domain of the attribute *per se*. In many applications, this can be much smaller than the potential complete value of M. For the periodic query algorithm the space is not a problem since the histogram needs only $O(\rho)$ space, where ρ is the period that we use in this approach.

Furthermore, another interesting point is that the proxy does not really execute any queries; it just creates the fake queries and keeps track of the state of the actual queries. Therefore, it is possible to implement the proxy using secure hardware, for example secure co-processors [6, 5]. This has a number of benefits. The operations executed from the proxy can be easily executed inside a secure hardware platform. Moreover, the proxy can be placed very close to the server and the communication overhead between the server and the proxy can be minimized.

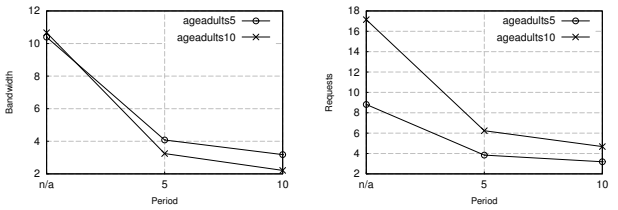
6. EXPERIMENTS

In this section, we experimentally evaluate the uniform and periodic query algorithms, as well as their extensions that learn the user's query distribution adaptively online. We aim to (i) clearly illustrate the cost of these algorithms in different settings, (ii) measure the gain in performance when using the QueryP algorithm, (iii) explore the performance of an end-to-end system that runs range queries using MOPE, (iv) showcase the ability to deploy query optimization methods, and (v) evaluate the algorithm for learning the user's query distribution.

Query distributions: As we discussed in the previous sections, the performance of a query algorithm is determined by the user's behavior, i.e., the query distribution $\mathcal{Q}$. In order to evaluate our proposed algorithms with practical workloads, we simulate a user's query distribution in the following way: (i) We use a combination of real and synthetic datasets to define a distribution over the range of values in the database. This distribution will determine the position of each query, implicitly assuming that a user is more interested in querying records that are densely represented in the dataset. (ii) In order to account for variable query lengths, we sample the length of each range query from the normal distribution $N(0, \sigma^2)$ for different values of σ.

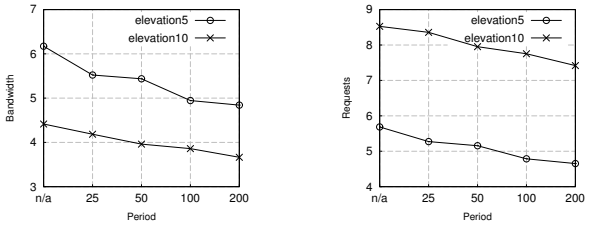
By combining the query's center and length, we generate the query distribution against which we will evaluate our algorithms.

The datasets that we used for step (i) above are: `Uniform`, `Zipf`, `Adult`, `Covertype`, `SanFran`. Details about these datasets can be found in the Appendix.



(a) (b)

Figure 5: `Bandwidth`(5a) and `Requests`(5b) costs for the `Adult` query distribution with $\sigma = 5$ and 10.



(a) (b)

Figure 6: `Bandwidth`(6a) and `Requests`(6b) costs for the `Covertype` query distribution with $\sigma = 5$ and 10.

Each of the resulting query distributions is defined on a different domain. As such, we treat them differently during the experiments, by picking values for the period size and the query length that make sense.

In Section 6.3, we test our system against the popular TPC-H benchmark. Instead of generating simulated user queries, we use the ones that the benchmark provides. Specifically, we chose queries `Q4`, `Q6` and `Q14`, since they are the only ones to query ranges.

Notation: We denote by $\mathcal{R}$ the set of the user's real queries. The first step of both our algorithms is to break each query $q \in \mathcal{R}$ into a set of transformed queries $\tau_k(q)$, all of which are of length k and cover at least the same range of values as q. We define $\mathcal{T}$ to be the multi-set $\bigcup_{q\in\mathcal{R}} \tau_k(q)$. Finally, $\mathcal{F}$ will denote the set of fake queries.

Cost evaluation: In practice, cloud-service providers have different pricing schemes for the number of requests and the bandwidth that is being used. To reflect this, we introduce two different cost functions.

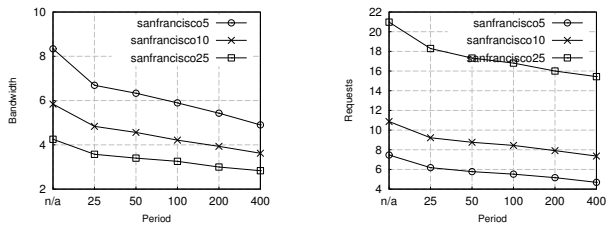
The first one captures the *overhead* in the number of records that we need to retrieve from the database and we formally define it as

$$\texttt{Bandwidth}(\mathcal{R},\mathcal{F}) = \frac{\sum_{q\in\mathcal{F}} |q| + \sum_{q\in\mathcal{R}} (|q| \mod k)}{\sum_{q\in\mathcal{R}} |q|}$$

where $|q|$ denotes the size of the answer to query q (number of records returned). This cost depends on the number of fake records that are queried and the number of excess records requested due to the transformation of the original queries.

The second measure captures the *relative increase* in the number of requests to the service provider. We define it as

$$\texttt{Requests}(\mathcal{R},\mathcal{T},\mathcal{F}) = \frac{|\mathcal{T}| + |\mathcal{F}|}{|\mathcal{R}|}$$



(a) (b)

Figure 7: `Bandwidth`(7a) and `Requests`(6b) costs for the `SanFran` query distribution with $\sigma = 5$, 10 and 25.

6.1 The Cost of Security

In our first experiment, we aim to measure the cost of using each query algorithm, both in terms of `Bandwidth` and `Requests`. For each of the query distributions that we introduced above, we generate queries to simulate a user's behavior. We try different values for σ that make sense for the underlying database. Then, each query is handed over to the proxy, who runs one of the two algorithms, either QueryU or QueryP, with a fixed value $k = 10$. We let the proxy execute the queries and observe the sets $\mathcal{T}$ and $\mathcal{F}$. We report the results in Figures 5, 6 and 7. We got similar results for the `Uniform` and `Zipf` distributions, but we do not report them for brevity. We also repeated this experiment for multiple values of k; however the trend was similar.

6.1.1 The Uniform Query Algorithm

First, we consider QueryU. The results for QueryU correspond to the value of each of the cost functions for Period=`n/a`. As expected, for a constant k, the `Bandwidth` when the queries in $\mathcal{R}$ are short ($\sigma = 5$) is bigger than when they are longer. This can be attributed to the fact that the denominator, $\sum_{q\in\mathcal{R}} |q|$, becomes smaller as σ decreases.

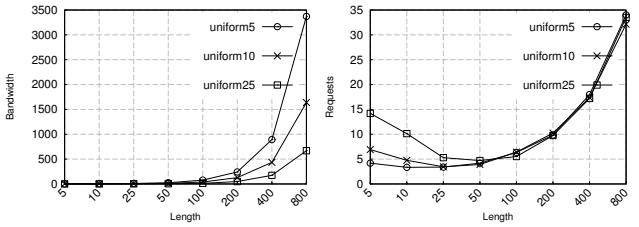
On the other hand, `Requests` behaves oppositely. Longer queries in $\mathcal{R}$ mean that, for a fixed k, the set $\mathcal{T}$ will be strictly bigger and it will dominate the cost function.

6.1.2 The Periodic Query Algorithm

Next, we use the QueryP algorithm with different choices for the period size. It is clear that the cost functions decrease as we increase the size of the period. However, this comes with a price — larger periods leak more bits of information. The aim of this experiment is to measure the trade-off between the security and the performance of our scheme. Notice that the behavior of QueryP depends strongly on the user's query distribution. We observe that the cost for query distributions similar to `Adult` and `SanFran` is greatly decreased, even with just a small sacrifice in the security. On the other hand, the cost for query distributions such as `Covertype` seems to be much harder to change.

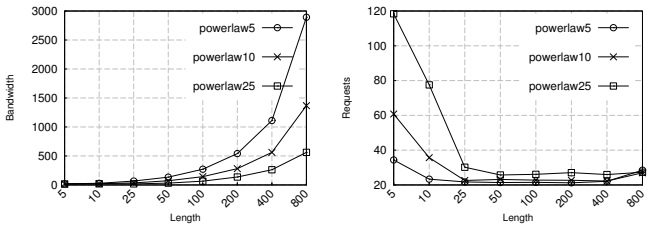
6.2 Choosing the Query Length

One important parameter of both QueryU and QueryP is the fixed length k of the transformed queries in $\mathcal{T}$. The reason of its importance is the fact that larger values for k lead to increased `Bandwidth` costs, but also tend to decrease the `Requests`. In this experiment, we explore how this choice affects the cost for the different query patterns that we use, for the case of the QueryP algorithm with period size $\rho = 25$. However, the trend of the cost functions is similar for other



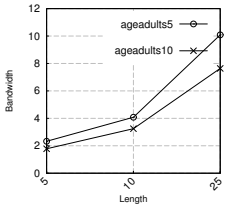
(a) (b)

Figure 8: `Bandwidth`(8b) and `Requests`(8b) costs for `Uniform` query pattern and different k.



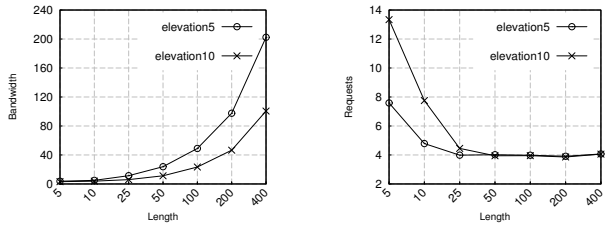
(a) (b)

Figure 9: `Bandwidth`(9a) and `Requests`(9b) costs for `Zipf` query pattern and different k.



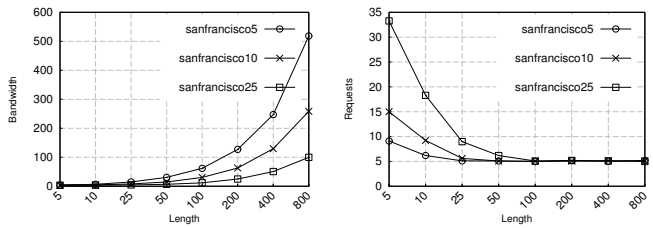
(a) (b)

Figure 10: `Bandwidth`(5a) and `Requests`(5b) costs for `Adult` query pattern and different k.



(a) (b)

Figure 11: `Bandwidth`(6a) and `Requests`(6b) costs for `Covertype` query pattern and different k.



(a) (b)

Figure 12: `Bandwidth`(7a) and `Requests`(6b) costs for `SanFran` query pattern and different k.

period sizes, as well as for QueryU. We report the results of this experiment in Figures 8, 9, 10, 11 and 12.

We observe that, in order to minimize both costs, one should pick k to be above the median length of the queries in $\mathcal{R}$.

6.3 The TPC-H Benchmark

As part of this work, we built a prototype proxy that connects to a PostgreSQL database server and implements both QueryU and QueryP algorithms. We chose to test these algorithms against the TPC-H benchmark, a widely-used database performance benchmark. There are only a handful query templates in TPC-H that execute range queries. Specifically, `Q1`, `Q4`, `Q6` and `Q14` generate range queries on a date attribute of the database. This attribute represents the full range of dates between the years 1992 and 1998, inclusive. Apart from `Q1`, all of the queries are of fixed length (3 months, 1 year and 1 month for each query type respectively) and only cover the range of dates between 1993 and 1997. `Q1` runs range queries that retrieves almost the whole database, so we chose not to include it in the experiments. We used a Scale Factor (SF) equal to 1 for generating the TPC-H workload. Thus, the LINEITEM table contains 6M tuples, ORDERS 1.5M tuples, and PART 200K tuples.

We start by creating an encrypted version of the tables that these queries are executed on, i.e. we encrypt the date values using MOPE. Then, for each query that TPC-H generates, the proxy encrypts it and runs it on the encrypted tables of the database. We pick k equal to the fixed (original) query size and measure the performance of the algorithms for different period sizes. The times we report are for 1000 client queries. The baseline we are comparing against is the time it took the client to execute those 1000 queries, without using any encryption or requesting the help of the proxy's algorithms. Conveniently enough, this time is almost the same for both `Q6` and `Q14` (1.3 and 1.1 seconds respectively), so, for ease of presentation, we consider them to be equal. The results of this experiment for `Q6` and `Q14` are summarized in Figure 13. We observe that, if we allow the adversary to know where within a quarter the client's query is located, the time it takes to run 1000 queries drops to under 20 seconds (or about 20 msec for each query). We also need to add here that the naïve approach to return the whole database every time – a strategy that would result in perfect hiding – would require 8 hours 20 minutes and 14 hours, for `Q14` and `Q6` respectively, in order to execute all 1000 queries. This means that for `Q14` even the QueryU algorithm is more than 660 times faster than the naïve approach and for `Q6` the QueryP with period 1 month is more than 800 times faster.

We consider the case of `Q4` separately. A single, unencrypted, query of this type takes almost 4 seconds to run. This makes it a little bit more expensive to handle it the same way as `Q1` and `Q14`. So, instead of reporting the running time, we skip the execution of the queries and just measure the `Requests` cost for the proxy. The results of this experiment are summarized in Figure 14.

6.4 Multiple Range Query Execution

As we discussed in Section 5, a big advantage of our MOPE schemes is that there is no need to modify the under-

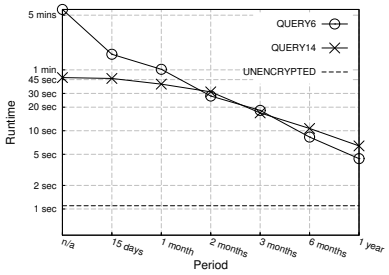


Figure 13: The time required to run 1000 queries generated by the `Q6` and `Q14` query templates using different period sizes.

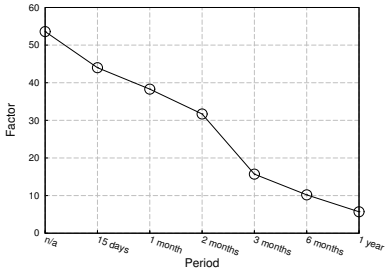


Figure 14: The `Bandwidth` cost when trying to hide the query pattern for `Q4`. A single query takes around 4 seconds to execute, so we can predict the actual running time.

lying DBMS in order to use it. As a result, practical query optimization methods are still applicable. To showcase this advantage, we do the following experiment: instead of executing the range queries one at a time, we combine a number of them into a disjunctive query (the WHERE clause is the logical disjunction of multiple ranges) and execute this instead. The setting is the same as in the previous section: the client executes 1000 encrypted queries of type `Q6` and `Q14` to the TPC-H database using the proxy's QueryU algorithm. Figure 15 shows that, as we combine more ranges, the time required for the proxy to execute all these queries is dramatically decreased. Notice that for the case of `Q14`, executing queries of 250 ranges at a time makes QueryU run in time comparable with the non-encrypted execution!

6.5 Learning the Query Distribution

For AdaptiveQueryU, we experimented with both a query distribution, i.e. `SanFran`, as well as the benchmark TPC-H (`Q14`). We measure the number of fake queries that need to be executed for each set of 10 unique real queries. Notice that because the sampling from the buffer is done with replacement, we may sample again a real query that has been executed before. In that case, we do not count this query as part of the unique queries but we count it as a fake query. In the beginning, when we do not know anything about the clients's query distribution, this number is expected to be large. However, as we observe more real queries, AdaptiveQueryU converges close to the performance of QueryU. Figure 16 summarizes these results. Indeed, each unique real query executed takes on average about 10 fake queries in the `SanFran` query pattern. For the case of `Q14`, what we observe is that the algorithm converges rapidly, since the distribution is practically over only 84 elements (one per month, for 7 years). The deviations we observe are due to randomness. We obtained similar results for all the other datasets.

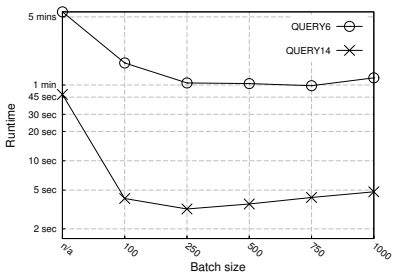
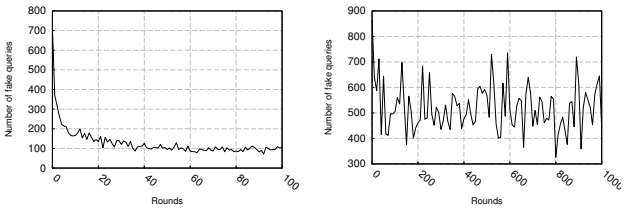


Figure 15: Combining multiple ranges into a single query results in dramatic speedups of the QueryU algorithm. The times reported are for 1000 queries.



(a) (b)

Figure 16: The number of fake queries executed for round of 10 real queries in `SanFran`10(16a) and `Q14` of TPC-H(16b). The AdaptiveQueryU converges really fast, especially for `Q14`.

7. SECURITY ANALYSIS

We follow the paradigm of *provable security*, the standard in modern cryptography. The first step is to capture the security of a scheme via a precise definition that specifies what an adversary is allowed to do when attacking the scheme and what its goals are. Such definitions are formulated via a probabilistic *experiment* that the adversary runs on some inputs, which is itself a randomized algorithm. For example, in a simple experiment the adversary may be given the encryptions of some messages and then tries to guess one of these messages. The adversary's *advantage* is defined to capture the probability that an adversary is able to violate the security according to the definition.

We can then analyze our candidate schemes under these definitions in order to assess their security, i.e., bound any efficient adversary's advantage against the scheme (usually under some well-accepted computational assumptions). It is important, especially in our context, to realize that the security of a scheme is not an all-or-nothing property, meaning that a scheme should not really be viewed as "either secure or not." Our schemes are such that some information about the data intentionally leaks in order to allow efficient query processing, and we strive to minimize information leakage beyond that. The provable security paradigm allows us to precisely capture and measure such a leakage.

7.1 Background on (M)OPE Security

We recall the security models for OPE and MOPE introduced in [7, 8]. Below, we fix a symmetric encryption scheme $\mathsf{SE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$. The definitions that follow do not assume that the scheme is (modular) order-preserving, but this is an important special case.

POPF security: We first define the notion of *pseudorandom order-preserving function* (POPF) security introduced in [7]. Intuitively, this notion asks that oracle access to the encryption algorithm of the scheme is indistinguishable from that to a truly random order-preserving function with the same domain and range (the "ideal object"). Let $\mathsf{OPF}[M, N]$ denote the set of all monotonically-increasing functions from $[M]$ to $[N]$. For every adversary A define its *POPF-advantage* as

$$\mathbf{Adv}^{\mathrm{popf}}_{\mathsf{SE}}(A) \;=\; \Pr[A^{\mathsf{Enc}(K,\cdot)} \text{ outputs } 1 \,:\, K \leftarrow\!\!\$\, \mathsf{Kg}] - \Pr[A^{f(\cdot)} \text{ outputs } 1 \,:\, f \leftarrow\!\!\$\, \mathsf{OPF}[M, N]] \;.$$

The work of [7] provides a POPF-secure OPE construction based on any standard blockcipher such as AES.

The related notion of *pseudorandom modular order preserving function* (PMOPF) security was introduced in [8]. Now, the "ideal object" is a random *modular* order-preserving function with the same domain and range. Let $\mathsf{MOPF}[M, N]$ denote the set of all modular order-preserving functions from $[M]$ to $[N]$ (that is, all order-preserving functions appended with all possible modular shifts). For every adversary A define its *PMOPF-advantage* as

$$\mathbf{Adv}^{\mathrm{pmopf}}_{\mathsf{SE}}(A) \;=\; \Pr[A^{\mathsf{Enc}(K,\cdot)} \text{ outputs } 1 \,:\, K \leftarrow\!\!\$\, \mathsf{Kg}] - \Pr[A^{f(\cdot)} \text{ outputs } 1 \,:\, f \leftarrow\!\!\$\, \mathsf{MOPF}[M, N]] \;.$$

[8] showed that one can easily extend a POPF-secure OPE (like that of [7]) to a MPOPF-secure MOPE by introducing a secret (pseudo)random shift to encryption and decryption.

WOW-L and WOW-D security: Next, we characterize the data privacy provided by a POPF-secure OPE scheme. Fix a parameter n, which is the number of ciphertexts in the database, and w, which is the "window size." For a set S let $\mathcal{S}_k(S)$ denote the set of all k-element subsets of S, for any $1 \le k \le |S|$.

We consider security experiments in which a database is sampled randomly and then the encrypted database is given to the adversary, who in turn tries to infer something about the plaintexts. Specifically, we use the notions of *window one-way wayness for location* (WOW-L) and *window one-way wayness for distance* (WOW-D) from [8]. Intuitively, WOW-L asks that it should be hard to specify a consecutive interval (i.e., a "window") of at most a certain length in which the decryption of some ciphertext lies. Analogously, WOW-D asks that it should be hard to specify a window of at most a certain length in which the distance between the decryption of some two ciphertexts lies. For more details we refer to [8].

Note that if OPE met the standard notion of semantic security, then the WOW-L and WOW-D advantage of any adversary would be upper-bounded by nw/M, since the adversary would get no useful information about the plaintexts and should just guess at random. (In general we could allow the adversary to choose a non-consecutive window in which a plaintext or distance could lie, but then in the case of OPE it is to the adversary's advantage to choose a consecutive one anyway.) Roughly, for OPE we get a $\sqrt{M}$ in the denominator instead. This means that roughly the *upper-half* of the bits of each plaintext, as well as the *upper-half* of the bits of the distance between each pair of plaintexts, are leaked by POPF-secure OPE.

Security bounds for POPF-secure MOPE: We now give the bounds from [8] for MOPE.

THEOREM 1. *Let* $\mathsf{OPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ *be a POPF-secure OPE scheme with* $N \ge 8M$ *and let* MOPE *be the associated MOPE scheme. Then for every adversary* A *there is an adversary* A' *such that*

$$\mathbf{Adv}^{\text{wow-}\ell}_{\mathsf{MOPE},n,w}(A) \;\le\; \mathbf{Adv}^{\mathrm{pmopf}}_{\mathsf{MOPE},n}(A') + \frac{nw}{M} \;.$$

The running-time of A' *is that of* A.

THEOREM 2. *Let* $\mathsf{OPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ *be a POPF-secure OPE scheme with* $N \ge 8M$ *and let* MOPE *be the associated MOPE scheme. Then for every adversary* A *there is an adversary* A' *such that*

$$\mathbf{Adv}^{\text{wow-d}}_{\mathsf{MOPE},n,w}(A) \;\le\; \mathbf{Adv}^{\mathrm{pmopf}}_{\mathsf{MOPE},n}(A') + \frac{4n(n-1)w}{\sqrt{M-n+1}} \;.$$

The running-time of A' *is that of* A.

Intuitively, the above says that in the case of one-wayness for location, the security of MOPE is the same as for semantic security, i.e., no information about the location is revealed at all. However, in the case of distance, it is the same as for OPE.

7.2 New Security Models with Queries

As shown by the gap attack, in an appropriate security model for MOPE we need to take into account the queries seen by the adversary. Accordingly, we will now denote a MOPE scheme by $\mathsf{MOPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Query})$. Here the stateful and randomized algorithm Query on input a key K, an un-encrypted (standard or wrap-around) query (m_L, m_R), and state St outputs an encrypted query (c_L, c_R) and updated state St'. The un-encrypted query may also be the empty string ε, which indicates that there is no new query to process but that the next query in the encrypted query sequence should be returned. Note that in our general model the Query algorithm is otherwise arbitrary; in our analyses in Sections 7.3 and 7.4, it will be replaced by (appropriate modifications of) the algorithms from Sections 3 and 4.

For any choice of Query, it will be useful to define the following algorithm that takes as input a key K and an un-encrypted query sequence $Q = (m_L^1, m_R^1), \ldots, (m_L^q, m_R^q)$, and outputs a corresponding encrypted query sequence that contains the real encrypted queries embedded in it:

Algorithm $\mathsf{MakeQueries}(K, Q)$:
  $St \leftarrow \varepsilon$
  For $i = 1$ to q do: // process queries in Q
    $(c_L^i, c_R^i, St) \leftarrow\!\!\$\, \mathsf{Query}(K, (m_L^i, m_R^i), St)$
  While $St \ne \varepsilon$ do: // output remaining encrypted queries
    $i \leftarrow i + 1$; $(c_L^i, c_R^i) \leftarrow\!\!\$\, \mathsf{Query}(K, \varepsilon, St)$
  $C_Q \leftarrow ((c_L^1, c_R^1), \ldots, (c_L^i, c_R^i))$
  Return C_Q

We require that for any Q, $\mathsf{MakeQueries}(K, Q)$ terminates with overwhelming probability over $K \leftarrow\!\!\$\, \mathsf{Kg}$ and the coins of the algorithm itself.

WOW-L and WOW-D with queries: As before, let n be the database size and w be the window size. Let $\mathcal{Q}$ denote a distribution on queries; i.e., $\mathcal{Q}$ is a distribution on the set

$$\{(m_L, m_R) \mid 1 \le m_L \le m_R \le M\} \;.$$

Experiment $\mathbf{Exp}^{\text{wow}^*\text{-}\ell}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A)$:
/* Sample encrypted database */
$K \leftarrow\$ \mathsf{Kg}$
$D \leftarrow\$ \mathcal{S}_n([M])$; $C_D \leftarrow\$ \mathsf{Enc}(K, D)$
$m \leftarrow\$ D$; $c \leftarrow\$ \mathsf{Enc}(K, m)$
$Count \leftarrow 0$
/* Now adversary can ask for queries */
$x \leftarrow\$ A^{\mathsf{NextQuery}(q,Count)}(C_D, c)$
If $m \in [x, x+w]$
Then return 1
Else return 0

Experiment $\mathbf{Exp}^{\text{wow}^*\text{-d}}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A)$:
/* Sample encrypted database */
$K \leftarrow\$ \mathsf{Kg}$
$D \leftarrow\$ \mathcal{S}_n([M])$; $C_D \leftarrow\$ \mathsf{Enc}(K, D)$
$\{m_1, m_2\} \leftarrow\$ \mathcal{S}_2(D)$
$\{c_1, c_2\} \leftarrow\$ \mathsf{Enc}(K, \{m_1, m_2\})$
$Count \leftarrow 0$
/* Now adversary can ask for queries */
$x \leftarrow\$ A^{\mathsf{NextQuery}(q,Count)}(C_D, c_1, c_2)$
If $|m_1 - m_2| \in [x, x+w]$
Then return 1
Else return 0

Oracle $\mathsf{NextQuery}(q, Count)$:
If $Count \geq q$
Return $\perp$
$(m_L, m_R) \leftarrow\$ \mathcal{Q}$
$(c_L, c_R, St) \leftarrow\$ \mathsf{Query}(K, m_L, m_R, St)$
$Count \leftarrow Count + 1$
Return (c_L, c_R)

Figure 17: The security experiments (left and middle) give the adversary access to an oracle (right).

Intuitively, we want to consider window one-wayness when the adversary also sees queries. We want to protect privacy of both the data and the queries. The basic set-up is as follows: We give the adversary the encrypted database but also give it the ability to *request the next query* via an oracle that samples a query from an appropriate distribution. It can query this oracle (which takes no input) as much as it likes to receive more queries, up to some query limit q (which will play a role in our security bounds). Formally, we consider the experiments given in Figure 17. For an adversary A define its *WOW*-L advantage* (one-wayness for location with queries) as

$$\mathbf{Adv}^{\text{wow}^*\text{-}\ell}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A) = \Pr\left[\mathbf{Exp}^{\text{wow}^*\text{-}\ell}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A) \text{ outputs } 1\right].$$

Similarly, for an adversary A define its *WOW*-D advantage* (one-wayness for distance with queries) as

$$\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A) = \Pr\left[\mathbf{Exp}^{\text{wow}^*\text{-d}}_{\mathsf{SE},n,w,q,\mathcal{Q}}(A) \text{ outputs } 1\right].$$

Note that we do *not* require the plaintext distances in the encrypted queries to be hidden; that is, we allow the adversary to learn the length of the requested plaintext ranges. This admits a much more practical solution.

Notice also a further change in the experiments $\mathbf{Exp}^{\text{wow}^*\text{-}\ell}$, $\mathbf{Exp}^{\text{wow}^*\text{-d}}$ from their non-query counterparts: we require that the adversary invert a *specific* plaintext or distance between two plaintexts (chosen at random). This change in definition allows us to consider applications where the database size n is large, and one is concerned with the ability of an adversary to uncover a specific message or distance—a more practical scenario. And while this weakens the security notions themselves, our later security results lose none of their potency from a practical standpoint.

7.3 The Uniform Query Algorithm

We can easily fit QueryU and AdaptiveQueryU into our general model above. For example, we can modify QueryU to maintain a buffer as in AdaptiveQueryU and return the next query to be executed instead of executing many queries until the input is executed. We slightly abuse notation and use the same names for the appropriately modified algorithms. Below we prove results about QueryU; analogous results hold for AdaptiveQueryU. The first result says that when using this algorithm to make range queries (regardless of the fixed query length) with MOPE, one-wayness for location is the best possible (no information is revealed at all).

THEOREM 3. *Let* OPE *be a POPF-secure OPE scheme on domain* $[M]$*, range* $[N]$*. Let* $\mathsf{MOPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{QueryU})$ *be the associated MOPE scheme using the* QueryU *algorithm obtained from Section 3.1. Let* $\mathcal{Q}$ *be a distribution on* $[M]$*. Then for any query limit* q*, for every adversary* A *there is an adversary* A' *such that*

$$\mathbf{Adv}^{\text{wow}^*\text{-}\ell}_{\mathsf{MOPE},n,w,q,\mathcal{Q}}(A) \;\leq\; \mathbf{Adv}^{\text{pmopf}}_{\mathsf{MOPE},n}(A') + \frac{w}{M}\,.$$

The running-time of A' *is that of* A*.*

The next result is the most technically challenging to prove, and says that for distance one-wayness, using QueryU to make range queries with MOPE and fixing the query length to k leaks roughly at most the $\log(M + qk)/2$ most-significant bits of the distance between any two plaintexts.

THEOREM 4. *Let* $\mathsf{OPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ *be an OPE scheme on domain* $[M]$*, range* $[N]$*, with* $N \geq 16M$*. Let* $\mathsf{MOPE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{QueryU})$ *be the associated MOPE scheme using the* QueryU *algorithm obtained from Section 3.1. Let* $k \geq 1$ *be the fixed query length, with* $k \ll M$*,* q *be the query limit. Then for any adversary* A *there is an adversary* A' *such that*

$$\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{MOPE},n,w,q,\mathcal{Q}}(A) \;\leq\; \mathbf{Adv}^{\text{pmopf}}_{\mathsf{MOPE},n}(A') + \frac{8w}{\sqrt{M - qk - 1}}\,.$$

The running-time of A' *is that of* A*.*

The proofs for the Theorems 3 and 4 appear in the Appendix.

7.4 The Periodic Query Algorithm

Again, we can easily fit QueryP and AdaptiveQueryP into our model above. We present our results for QueryP, but analogous results hold for AdaptiveQueryP. Below we show that using this algorithm to make range queries with MOPE leaks roughly at most the $\log \rho$ *least-significant* bits of the plaintexts.

THEOREM 5. *Let* $\mathcal{Q}$ *be a distribution on* $[M]$ *and let* ρ *be an integer that divides* M*. Let* OPE *be a POPF-secure OPE scheme on domain* $[M]$*, range* $[N]$*, and let* $\mathsf{MOPE}(\rho) = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Query})$ *be the associated MOPE scheme, with* $\mathsf{Query}(\rho)$ *defined as the periodic algorithm above. Then for any query limit* q*, for every adversary* A *there is an adversary* A' *such that*

$$\mathbf{Adv}^{\text{wow}^*\text{-}\ell}_{\mathsf{MOPE}(\rho),n,w,q,\mathcal{Q}}(A) \;\leq\; \mathbf{Adv}^{\text{pmopf}}_{\mathsf{MOPE}(\rho),n}(A') + \frac{\rho w}{M}\,.$$

The running-time of A' *is that of* A*.*

The proof of Theorem 5 is also in the Appendix.

Notice that, in general, the periodic scheme has poor one-wayness for distance. In fact, depending on how the period ρ relates to the input distribution $\mathcal{Q}$, it may be quite easy for an adversary to guess distances based on knowledge of these parameters alone. For instance, suppose $\mathcal{Q}$ has equal spikes at a certain set of messages spaced ρ apart and is zero elsewhere. Then only distances $k\rho$ for integers k are possible, and thus low-order-bit information is leaked about distance (in addition to high-order-bit information about distance, which is always leaked by an RMOPF) resulting in poor distance one-wayness.

8. RELATED WORK

The idea of OPE was proposed by Agrawal et al. in [3] were they formally defined the problem and they provided a scheme to address it. Interestingly, this idea had already appeared before in [27, 19]. However, none of these works defined a formal security model as the one that we discuss here and no formal cryptographic security guarantees were discussed. The first to present a more formal security study of the problem was Boldyreva et al [7, 8].

A stronger notion of security for OPE would be that *only* order (or modular order, in the case of MOPE) among the plaintext is revealed. Unfortunately, [7] show such a definition is impossible to achieve. However, [30] (and an improvement by [23]) shows that this can be achieved by an "interactive" scheme that modifies existing encryptions in the database depending on the insertion of new values, and where encryption and decryption requires the interaction between the server and the client. However, here we work in the standard "non-interactive" setting. Remarkably, it has recently been shown that such security can even be achieved in the non-interactive setting by a "generalized" form of OPE that allows an arbitrary comparison operation on ciphertexts [16, 9], but such schemes are far from practical.

There are a number of works that use bucketization to store encrypted records for answering range queries, where each bucket corresponds to a given partition of the domain and each record is associated with a given partition. Given a range query, the buckets that intersect with the query must be identified by the client and an exact match query is issued at the server that retrieves all the records in these buckets. The records themselves are encrypted with a semantic security scheme. This idea has been used for both one dimensional [19, 22] and multidimensional range queries [21]. However, these works have weaker security models than ours and are susceptible to query pattern attacks.

Another work, that is related to our approach to obfuscate real queries using fake queries, is the work by Pang et al. [28], where they try to hide the terms in text search queries by introducing decoy terms. In a subsequent work [29], the same authors enhance a text query with a set of fake queries that can obfuscate the real query.

Some notable recent works on building database systems that run queries over encrypted data are CryptDB [31] and Monomi [34]. Both of them use an OPE scheme as one of their main encryption strategies that allows to execute database queries efficiently. Furthermore, two systems that use secure hardware and provide stronger security and better functionality are TrustedDB [6] and Cipherbase [5].

9. DISCUSSION AND FUTURE WORK

Relation to oblivious RAM: Oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky [15] (see [33] and references therein for more recent work) is a technique that hides all information about what positions in an outsourced database are accessed by the client, by continually shuffling around and re-encrypting the data. In principle ORAM could be used generically to solve the problem of hiding the user's access pattern in our modular OPE setting. However, ORAM is less efficient than our solution and is actually an overkill: We need to hide the *distribution* of access locations by the client, not the locations themselves.

Thus, we achieve efficiency by taking in account the *actual distribution* of the user's queries, whereas in the ORAM model the access pattern of the client is arbitrary and fixed. It is an interesting future direction whether such an approach can be beneficial in the ORAM setting as well.

A note on the security model: We note that obtaining the security benefits of modular OPE over basic OPE crucially relies on the fact that we only allow the adversary a so-called "ciphertext-only" attack. In particular, the adversary in our security models is not allowed to see known plaintext-ciphertext pairs; if it could, it would be able to orient the dataset despite the use of the modular offset.

We envision that in many applications such an attack is unlikely — in any case, security will degrade back only the case of basic OPE (it cannot be *worse* than using basic OPE), so in some sense there is no reason not to use modular OPE anyway. However, it is interesting question for future research whether something can be done to mitigate the effect of plaintext-ciphertext pair exposure, such as re-encrypting portions of the data at regular intervals.

Acknowledgements

We thank Raluca Ada Popa, Kobbi Nissim, and Nickolai Zeldovich for discussions. Part of this work was done while Adam O'Neill was at Boston University, supported by NSF grants CNS-1012910 and CNS-0546614. George Kollios and Ran Canetti were supported by an NSF SaTC Frontier Award CNS-1414119.

10. REFERENCES

[1] D. Agrawal, A. El Abbadi, B. C. Ooi, S. Das, and A. J. Elmore. The evolving landscape of data management in the cloud. *IJCSE*, 7(1):2–16, 2012.

[2] D. Agrawal, A. El Abbadi, and S. Wang. Secure data management in the cloud. In *DNIS*, pages 1–15, 2011.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *SIGMOD Conference*, pages 563–574, 2004.

[4] Amazon. Amazon RDS. http://aws.amazon.com/rds/.

[5] A. Arasu, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, and R. Ramamurthy. Transaction processing on confidential data using cipherbase. In *ICDE Conference*, 2015.

[6] S. Bajaj and R. Sion. Trusteddb: A trusted hardware based outsourced database engine. *PVLDB*, 4(12):1359–1362, 2011.

[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pages 224–241, 2009.

[8] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, pages 578–595, 2011.

[9] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Advances in Cryptology - EUROCRYPT*, 2015.

[10] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *SIGMOD Conference*, pages 251–264, 2008.

[11] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: a database service for the cloud. In *CIDR*, pages 235–240, 2011.

[12] S. Das, D. Agrawal, and A. El Abbadi. Elastras: An elastic, scalable, and self-managing transactional database for the cloud. *ACM Trans. Database Syst.*, 38(1):5:1–5:45, Apr. 2013.

[13] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag New York, 1986.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

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

[16] 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 *Advances in Cryptology - EUROCRYPT*, pages 578–602, 2014.

[17] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[18] Google. Google Cloud SQL. https://cloud.google.com/products/cloud-sql.

[19] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD Conference*, pages 216–227, 2002.

[20] H. Hacigümüs, S. Mehrotra, and B. R. Iyer. Providing database as a service. In *ICDE Conference*, pages 29–38, 2002.

[21] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *VLDB J.*, 21(3):333–358, 2012.

[22] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[23] F. Kerschbaum and A. Schropfer. Optimal average-complexity ideal-security order-preserving encryption. In *ACM SIGSAC Conference on Computer and Communications Security, CCS'14*, pages 1–12, 2014.

[24] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *Advances in Spatial and Temporal Databases, 9th International Symposium, SSTD, Proceedings*, pages 273–290, 2005.

[25] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song. PHANTOM: practical oblivious computation in a secure processor. In *ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 311–324, 2013.

[26] Microsoft. SQL Azure. http://www.windowsazure.com/en-us/develop/net/fundamentals/cloud-storage/.

[27] G. Özsoyoglu, D. A. Singer, and S. S. Chung. Anti-tamper databases: Querying encrypted databases. In *DBSec*, pages 133–146, 2003.

[28] H. Pang, X. Ding, and X. Xiao. Embellishing text search queries to protect user privacy. *Proc. VLDB Endow.*, 3(1-2):598–607, Sept. 2010.

[29] H. Pang, X. Xiao, and J. Shen. Obfuscating the topical intention in enterprise text search. In *ICDE Conference*, pages 1168–1179, 2012.

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

[31] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.

[32] T. K. Sellis. Global query optimization. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 191–205, 1986.

[33] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: an extremely simple oblivious ram protocol. In *ACM Conference on Computer and Communications Security*, pages 299–310, 2013.

[34] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5), 2013.

APPENDIX

A. SECURITY PROOFS

First, we prove a short lemma. Let RMOPF be the "ideal" (inefficient) MOPE scheme that selects a random element from $\mathsf{MOPF}[M, N]$ as its key, and samples this function and its inverse for encryption and decryption.

LEMMA 1. *Given an adversary A to experiment WOW* (can be WOW*-L or WOW*-D) on scheme MOPE, there exists PMOPF adversary A' with the same running time such that*

$$\mathbf{Adv}^{\text{wow}^*}_{\mathsf{MOPE},n,w}(A) - \mathbf{Adv}^{\text{wow}^*}_{\mathsf{RMOPF},n,w}(A) = \mathbf{Adv}^{\text{pmopf}}_{\mathsf{MOPE},n}(A').$$

(where wow* *is either* wow*-ℓ *or* wow*-d.*)*

PROOF. Let A' be an algorithm for the PMOPF experiment that simulates experiment $\text{WOW}^*_{\mathsf{SE},n,w}$ for adversary A, using its oracle access to either MOPE's actual encryption function or a truly random MOPF on the same domain and range as "encryption" for SE, and outputs the result of the experiment. The PMOPF advantage of A' is equal to the difference in success probabilities for WOW* between using MOPE's actual encryption function versus a truly random MOPF. The former success probability is $\mathbf{Adv}^{\text{wow}^*}_{\mathsf{MOPE},n,w}(A)$ and since a truly random MOPF is equivalent to the encryption function of RMOPF, the latter success probability is $\mathbf{Adv}^{\text{wow}^*}_{\mathsf{RMOPF},n,w}(A)$. The result follows. □

PROOF OF THEOREM 3. By Lemma 1, it is enough to prove that

$$\mathbf{Adv}^{\text{wow}^*\text{-}\ell}_{\mathsf{RMOPF},n,w}(A) \le \frac{w}{M} .$$

Recall that a random MOPF can be viewed as a random shift on the message space, followed by a random OPF. Notice then that even given the entire set of ciphertexts seen by the adversary, each ciphertext equally likely came from any element of the message space (because of the random shift.) Furthermore, all the adversary sees is ciphertexts. Thus, when the adversary guesses a window of size w, each ciphertext's likelihood to have come from a message in the window is simply w/M. □

PROOF OF THEOREM 4. In this proof, consider addition, subtraction, and "betweenness" of plaintexts and ciphertexts to be modular, i.e., they wrap around the plaintext and ciphertext spaces. Also, recall from [8] the definitions of *most likely plaintext* (m.l.p.) and *most likely plaintext distance* (m.l.d.): informally, the m.l.p. of a given ciphertext c is the plaintext m_c that most likely (over all keys and given knowledge) produced ciphertext c; the m.l.d. of given ciphertext pair c, c' is the plaintext distance $d_{c,c'}$ that most likely separates the plaintexts whose encryptions produced c_1 and c_2.

Invoking Lemma 1 and the trivial fact $\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{RMOPF},n,w}(A) \le w \cdot \mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{RMOPF},n,1}(A)$, it is left to prove that

$$\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{RMOPF},n,1}(A) \le \frac{8}{\sqrt{M}} .$$

$\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{RMOPF},n,1}(A)$ is bounded by the probability of the most likely plaintext distance (m.l.d.) between random messages m_1, m_2 when c_1, c_2 are known to the adversary as well as ciphertexts (c^i_L, c^i_R) for $i \in [q]$, each pair of which comes from some uniformly random query m^i_L, $m^i_R = m^i_L + k - 1$.

This task is equivalent to the following experiment: given N balls, numbered 1 through N (the ciphertext space), we are told that a random M of them are black (representing which are actual ciphertexts under the key) and the rest are white. Also, balls c_1 and c_2 are black (as they are valid ciphertexts.) Furthermore, for $i \in [q]$, the balls c^i_L and c^i_R are black and there are precisely k total black balls between c^i_L and c^i_R, inclusive. The goal is to guess how many black balls lie between c_1 and c_2. (Notice that the MOPE scheme's secret shift is irrelevant in determining plaintext distance.)

We prove the bound by showing a more powerful adversary's advantage is bounded by the quantity desired. The more powerful adversary is given slightly more information in addition to the above:

- When two range queries overlap, i.e. $c^i_L < c^j_L < c^i_R$ for some $i, j \in [q]$, the adversary knows how many black balls lie in each subinterval. That is, the adversary is given the number of black balls between c^i_L and c^j_L, between c^j_L and c^i_R, and between c^i_R and c^j_R.
- When a challenge ciphertext lies in a range query, i.e. $c^i_L < c_1 < c^i_R$ (or analogously, $c^i_L < c_2 < c^i_R$), the adversary knows how many black balls lie in each subinterval. That is, the adversary is given the number of black balls between c^i_L and c_1, and between c_1 and c^i_R.

Let $\mathcal{C}$ be the set of ciphertexts within a range query, but not including c_1 or c_2.

All told, the adversary knows that some number $r \le kq$ of black balls lie within $\mathcal{C}$. Of these, say r_{in} lie strictly in between c_1 and c_2 and r_{out} lie outside, so that $r_{in} + r_{out} = r$. Of the remaining $M - r$ black balls, the adversary knows there are black balls at c_1 and c_2 but knows nothing of the others, other than that they are uniformly distributed among the other balls in $\mathcal{R}' = [N] \setminus \mathcal{C}$. Thus, the adversary's advantage is bounded by its ability to guess how many of these remaining balls between c_1 and c_2 are black. (This guess would be combined with r_{in} to find the adversary's best guess.) Let $N' = |\mathcal{R}'|$. Then this is equivalent to guessing the m.l.d. between two random ciphertexts on a domain of size $M - r$ and range of size N'. Notice that $|\mathcal{C}| \approx \frac{N}{M} r = \frac{N}{M} kq \ll N$. Thus $N' \ge N/2 \ge 8M \ge 8(M - r)$ and the bound of Theorem 2 applies, giving

$$\mathbf{Adv}^{\text{wow}^*\text{-d}}_{\mathsf{RMOPF},n,1}(A) \le \frac{8}{\sqrt{M - r - 1}} \le \frac{8}{\sqrt{M - kq - 1}}.$$

Combining with the initial reductions, we achieve the result. □

PROOF OF THEOREM 5. Lemma 1 applies here, so it is enough to prove that

$$\mathbf{Adv}^{\text{wow}^*\text{-}\ell}_{\mathsf{RMOPF}(\rho),n,w}(A) \le \frac{\rho w}{M}$$

where $\mathsf{RMOPF}(\rho)$ is the scheme that uses a completely random MOPF as encryption function and $\mathsf{Query}(\rho)$ as a query algorithm. Notice that the query algorithm samples messages from the ρ-periodic distribution $\mathcal{P}_\rho$, and in the random MOPF a random shift is applied in encryption. Thus, the set of ciphertexts seen by the adversary has equal probability of coming from one of M/ρ different shifts of a certain set of messages. Hence, when the adversary guesses a window of size w, each ciphertext's likelihood to have come from a message in the window is simply $\rho w/M$. □

B. DATASETS

Here we describe in more details that datasets that we used in the experimental evaluation:

- `Uniform`: The most basic distribution of queries is when every record has equal chance of being the center of a query. We use a domain size of 10000.
- `Zipf`: Often, user behavior resembles a power law. Certain ranges in the database are accessed far more frequently than the rest. Again, the domain size is 10000.
- `Adult`: We generate the distribution of the query center based on the *age* attribute of the `Adult`[1] dataset. The range of values of *age* is between 17 and 90. The probability of each record in the database is equal to the frequency with which that record appears in the dataset.
- `Covertype`: This is a real dataset based on the *elevation* attribute of the `Covertype`[2]. The range of values here is between 1859 and 3858.
- `SanFran`: This is a spatial dataset[3] of the California Road Network's nodes[24]. We use the *longitude* information of the nodes, after binning the dataset in 10000 bins. The probability of each record then is equal to the probability of picking the corresponding bin at random.

[1] `https://archive.ics.uci.edu/ml/datasets/Adult`
[2] `https://archive.ics.uci.edu/ml/datasets/Covertype`
[3] `http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`