File-injection Attack and Forward Security for Order-revealing Encryption

Xingchen Wang and Yunlei Zhao

Abstract

Order-preserving encryption (OPE) and order-revealing encryption (ORE) are among the core ingredients for encrypted database (EDB) systems as secure cloud storage. In this work, we study the leakage of OPE and ORE and their forward security.

We propose generic yet powerful file-injection attacks (FIAs) on OPE/ORE, aimed at the situations of possessing *order by* and range queries. The FIA schemes only exploit the *ideal* leakage of OPE/ORE (in particular, no need of data denseness or frequency). We also improve its efficiency with the frequency statistics using a hierarchical idea such that the high-frequency values will be recovered more quickly. Compared with other attacks against OPE/ORE proposed in recent years, our FIA attacks rely upon less demanding conditions and are more effective for attacking the systems with the function of data sharing or transferring like encrypted email system. We executed some experiments on real datasets to test the performance, and the results show that our FIA attacks can cause an extreme hazard on most of the existing OPE and ORE schemes with high efficiency and 100% recovery rate.

In order to resist the perniciousness of FIA, we propose a practical compilation framework for achieving forward secure ORE. The compilation framework only uses some simple cryptographical tools like pseudo-random function, hash function and trapdoor permutation. It can transform most of the existing OPE/ORE schemes into forward secure ORE schemes, with the goal of minimizing the extra burden incurred on computation and storage. We also present its security proof and execute some experiments to analyze its performance.

Index Terms

Order-revealing Encryption; Order-preserving Encryption; File-injection Attack; Forward Security.

I. Introduction

Due to the increased data created in every industry moment by moment, database-as-a-service (DaaS) model has been regarded as a frontier idea with large development space. The basic DaaS model consists of the users, the clients and the server. Because of the support like the *Precise Query Protocols* (PQPs) and the *bucketization* approach, the DaaS model was designed to support the users in storing, operating and analyzing their data on the cloud.

In recent years, many property-preserving encryption (PPE) schemes and property-revealing encryption (PRE) schemes have been proposed with increased efficiency or/and security. This condition promotes the occurrence of encrypted database (EDB) systems. CryptDB [30] has been proposed by Popa et al. as the first practical EDB system for executing data manipulations on encrypted data. Because of its onion encryption model and its proxy architecture, CryptDB supports most of the basic operations on ciphertexts with acceptable efficiency. Based on or inspired by the idea of CryptDB, an increasing number of DaaS systems have been created as new web services. Adopting the idea of CryptDB, *Google Encrypted Bigquery* [2] and *SAP SEEED* [19] are both designed to provide a cloud database service for user to store and analyze numerous data with reasonable security.

As a kind of PPE, order-preserving encryption (OPE) has been gaining more and more attention and studies because of the new application area on EDB. OPE was first proposed for numeric data by Agrawal et al. [5], where the order of plaintexts can be obtained by comparing their ciphertexts directly. Later, order-revealing encryption (ORE) was proposed by Boneh et al. [8] as the generalization of OPE, where the ciphertexts reveal their order by a special algorithm rather than comparing themselves directly. Five formal leakage profiles of OPE/ORE were proposed and studied by Boldyreva et al. [6] and Chenette et al. [14]. Frequency-hiding OPE schemes, which achieve *stronger-than-ideal* security, were recently proposed in [24] and [31].

Even though OPE and ORE aim at leaking nothing other than the order of ciphertexts, many attacks have been proposed against OPE and ORE in recent years. Islam et al. [23] proposed a general inference attack on PQPs based on access pattern disclosure in the context of secure range queries. Naveed et al. [28] proposed several inference attacks against the deterministic encryption (DTE) and OPE in CryptDB. Durak et al. [16] showed that some ORE schemes, whose security is discussed on uniform inputs, could make the plaintext recovery of some known attacks more accurate on nonuniform data. They also proposed an attack, aiming at multiple encrypted columns of correlated data, which reveals more information than prior attacks against columns individually. Grubbs et al. [20] proposed new leakage-abuse attacks that achieve high-correctness recovery on OPE-encrypted data. They also presented the first attack on frequency-hiding OPE proposed in [24].

Xingchen Wang and Yunlei Zhao are with School of Software, Fudan University, Shanghai, China (e-mail: {xingchenwang16, ylzhao}@fudan.edu.cn). Yunlei Zhao is the corresponding author.

A. Our Contributions

In this paper, we demonstrate the power of file-injection attacks (FIAs) on OPE/ORE, by developing two categories of FIA schemes (to the best of our knowledge, the first such attacks) against OPE/ORE. The FIA attacks are improved using a hierarchical idea such that the high-frequency values will be recovered more quickly. Our FIA attacks are generic and powerful, in the sense that they only exploit the *ideal* leakage of OPE/ORE. Specifically, for our FIA attacks to work, the adversary only possesses the plaintext space, some old *order by* or range queries and the corresponding cipher result sets returned from EDB. In particular, the adversary does not need either the ability of comparing ciphertexts with the ORE comparison algorithm, or that of obtaining the ciphertexts outside of the result sets for *order by* and range queries. After the introduction of our basic FIA schemes, we also discuss several approaches to further increasing efficiency with some extra leakage (e.g., possessing both *order by* and range queries). In comparison with other attacks against OPE/ORE proposed in recent years, our FIA attacks rely upon less demanding conditions and are more effective for attacking the systems with the function of data sharing or transferring like encrypted email system. For example, compared with the attacks against OPE/ORE proposed in [20] and [28], our FIA attacks have the following features simultaneously: (1) no need of data denseness or frequency, and (2) generic against any OPE/ORE with ideal leakage. Furthermore, we detailedly illustrate the advantages of our attacks over the chosen-plaintext attack and the inference attack in Section III-F.

Next, we present some experiments about our FIAs on the OPEs and OREs with *ideal* security. The results show that our FIAs can cause an extreme hazard on most of the existing OPE and ORE schemes with high efficiency and 100% recovery rate when the plaintext space is unbroken.

Finally, we propose a compilation framework for achieving forward secure ORE schemes against FIA attacks. Specifically, the compilation framework is applicable to most of the existing OPE/ORE schemes to transform them into forward secure ORE ones. The resultant forward secure schemes leak nothing about newly inserted data that match the previous *order by* or range queries. Moreover, the compilation framework is constructed with the goal of minimizing the extra burden incurred on computation and storage. In particular, the compilation only uses some simple cryptographical tools like pseudo-random function (PRF), hash function and trapdoor permutation (TDP). After the security analysis, we also present two approaches to reducing the storage complexity of client. Finally, we discuss the computation incremental cost abstractly and execute some experiments to analyze the additional cost caused when applying our compilation framework to some prominent OPE/ORE schemes developed in recent year.

B. Related Work

1) Order-preserving encryption (OPE): Agrawal et al. [5] first proposed an OPE scheme for numeric data. Afterwards, OPE was formally studied by Boldyreva et al. [6], where, in particular, two leakage profiles were introduced. Boldyreva et al. [7] analyzed the one-wayness security of OPE, and showed that any OPE scheme must have immutable large ciphertexts if the scheme is constructed for leaking only order and frequency information. Popa et al. [29] proposed an OPE scheme in order tree structure, which is the first OPE scheme achieving the security of IND-OCPA (indistinguishability under ordered chosen-plaintext attack). Kerschbaum [24] proposed a frequency-hiding OPE scheme, which supports the security of IND-FA-OCPA (indistinguishability under frequency-analyzing ordered chosen-plaintext attack) for the first time. Later, a partial order preserving encryption (POPE), with a method for frequency-hiding, was developed by Roche et al. [31]. The POPE scheme proposed in [31] is mainly for the application scenarios where the system executes more insertion operations than order operations, and achieves even stronger security called IND-FA-POCPA (indistinguishability under frequency-analyzing partial ordered chosen-plaintext attack).

2) Order-revealing encryption (ORE): ORE was first generalized from OPE by Boneh et al. [8]. Their ORE scheme is built upon multilinear maps, which provides better security but at the cost of worse efficiency. Chenette et al. [14] proposed the first practical ORE, which achieves a simulation-based security w.r.t. some leakage functions that precisely quantify what is leaked by the scheme. Recently, Cash et al. [12] presented a general construction of ORE with reduced leakage as compared to [14], but at the cost of using a new type of "property-preserving" hash function based on bilinear maps.

3) File-injection attack on searchable symmetric encryption (SSE): File-injection attack was named by Zhang et al. [33], but it was first proposed by Cash et al. [11] who called it the known-document attack. Islam et al. [22] initiated this study of SSE security by showing that even a curious service provider can recover most of the keywords-search queries with high accuracy. Their attack is based on the condition of possessing the plaintext space, and uses the *L1* leakage named in [11] that contains the query pattern (i.e., the contents and repetition times of queries) and the file-access pattern (i.e., all the returned files as response to each query in timing order).

Cash et al. [11] further improved the power of the attack initiated in [22], by assuming less knowledge about the files of clients even in a larger plaintext space. In addition, they showed how the attack effects can be significantly improved if the adversary gets more leakages. Except the encrypted email systems like Pmail [3], they also discussed how their active attacks (e.g., query recovery attacks, partial plaintext recovery attacks, FIAs) might be used to break through other systems such as the systems in [21] and [25].

Zhang et al. [33] showed that FIA can recover the keywords-search queries with just a few injected files even for SSE of low leakage. Their attacks outperform the attacks proposed in [11] and [22] in efficiency and in the prerequisite of adversary's prior knowledge.

4) Forward secure encryption (FSE): Forward security of PPE/PRE was first considered for SSE. Stefanov et al. [32] presented the formal definition of forward security for SSE, and proposed a dynamic forward secure SSE scheme with sublinear complexity. Tracing back to earlier years, Chang et al. [13] proposed an SSE scheme with linear complexity which actually achieves forward security but without a clear and formal treatment. Nevertheless, the constructions of SSE in [13] and [32] cause heavy consumption of bandwidth or storage. In order to reduce the cost, Bost [9] proposed an SSE scheme with forward security named $\Sigma o\phi o\varsigma$, which is efficient with additional client storage $O(W \log D)$ where *W* is the number of distinct keywords and *D* is the number of documents. Furthermore, Bost et al. [10] also proposed an efficient and dynamic SSE, which includes a forward secure scheme derived from [32].

In general, FSE can be constructed from Oblivious RAM (ORAM) [18], which also enjoys many other desirable security features. Unfortunately, because of the generality of ORAM, the ORAM-based forward secure scheme named TWORAM proposed in [17] has a large bandwidth overhead, multiple data round-trips, and/or large client storage consumption. As noted in [27], ORAM is currently not suitable for SSE. Hence, making use of ORAM for forward secure OPE/ORE is impractical as well.

II. Preliminaries

In this section we introduce some fundamental knowledge of ORE, OPE, FIA, TDP and forward security. We use standard notations and conventions below for writing probabilistic algorithms, experiments and interactive protocols. If $\mathcal{D}$ denotes a probability distribution, $x \leftarrow \mathcal{D}$ is the operation of picking an element according to $\mathcal{D}$. If $\mathbf{S}$ is a finite set then $|\mathbf{S}|$ is its cardinality and $x \xleftarrow{\$} \mathbf{S}$ is the operation of picking an element uniformly at random from $\mathbf{S}$. If $\mathbf{S}$ is an set then for any *k*, $0 \leq k \leq |\mathbf{S}| - 1$, $\mathbf{S}[k]$ denotes the $(k{+}1)$-th element in $\mathbf{S}$. If α is neither an algorithm nor a set then $x \leftarrow \alpha$ is a simple assignment statement. If *A* is a probabilistic algorithm, then $A(x_1, x_2, \cdots; r)$ is the result of running *A* on inputs $x_1, x_2, \cdots$ and coins r. We let $A(x_1, x_2, \cdots) \rightarrow y$ denote the experiment of picking r at random and letting y be $A(x_1, x_2, \cdots; r)$. By $\mathbb{P}[R_1; \cdots; R_n : E]$ we denote the probability of event E, after the ordered execution of random processes $R_1, \cdots, R_n$.

A. Order-Revealing Encryption

Definition 1 (Order-Revealing Encryption): A secret-key encryption scheme is an order-revealing encryption (ORE), if the scheme can be expressed as a tuple of algorithms ORE = (ORE.Setup, ORE.Encrypt, ORE.Compare) which is defined over a well-ordered domain $\mathcal{M}$ with the following properties:

- ORE.Setup$(1^\lambda) \rightarrow sk$. On input a secure parameter λ, the setup algorithm outputs a secret key which is used for encrypting plaintexts afterward.
- ORE.Encrypt$(sk, m) \rightarrow c$. By making use of the secret key sk generated before, the encryption algorithm encrypts the input plaintext m to a ciphertext c that can reveal the correct order with other ciphertexts.
- ORE.Compare$(c_1, c_2) \rightarrow b$. On input two ciphertexts c_1, c_2, the comparison algorithm returns a bit $b \in \{0, 1\}$ as the result of order.

Decryption algorithm. In the description of ORE above, we only focus on the basic definition of ORE, without introducing many other parameters and components like clients for interactive queries (as our FIAs are w.r.t. the generic OPE/ORE structure). In particular, the ORE.Decrypt algorithm is actually absent. We omit the decryption algorithm, because it is not an essential part of an ORE scheme. The data owners can execute some binary search over the ciphertexts with their secret key to infer the corresponding plaintexts of the ciphertexts in the result set.

Leakage profiles. The *ideal* leakage profile, the *random order-preserving function* profile, the *most significant-differing bit* profile, the *RtM* profile and the *MtR* profile are five leakage profiles that have been proposed in the literature. The first two were described by Boldyreva et al. [6], and the else were described by Chenette et al. [14].

We remark that, in Section III, our FIAs are generic in the sense that they are constructed only with the *ideal* leakage profile. The *ideal* leakage profile just reveals the order and the frequency of the plaintexts. The frequency-hiding OPE proposed by Kerschbaum [24] and the POPE scheme with frequency-hiding strategy proposed by Roche et al. [31] achieve both *stronger-than-ideal* security actually.

An adversary is said to be adaptive, if it is allowed to adaptively select data to be encrypted by the clients and then stored back to the server. Roughly speaking, an ORE scheme is said to be $\mathcal{L}$-adaptively-secure, if any probabilistic polynomial-time (PPT) adaptive adversary cannot learn more than the leakage as described according to the leakage profile $\mathcal{L}$.

B. Order-Preserving Encryption

Order-preserving encryption (OPE) is a simplified case of ORE. The ciphertext domain $\mathcal{C}$ of OPE needs to be well-ordered exactly as the plaintext domain $\mathcal{M}$. The order result of the comparison algorithm only relies upon the order relation among ciphertexts. In other words, the correct order of ciphertexts has been preserved from their plaintexts when they were encrypted before.

Definition 2 (Order-Preserving Encryption): A secret-key encryption scheme is an order-preserving encryption (OPE), if the scheme can be expressed as a tuple of algorithms OPE = ($\mathsf{OPE.Setup}$, $\mathsf{OPE.Encrypt}$), which is defined over a well-ordered plaintext domain $\mathcal{M}$ and a well-ordered ciphertext domain $\mathcal{C}$ with the following properties:

- $\mathsf{OPE.Setup}(1^\lambda) \to sk$. On input a secure parameter λ, the setup algorithm outputs a secret key sk which is used for encrypting plaintexts afterward.
- $\mathsf{OPE.Encrypt}(sk, m) \to c$. By making use of the secret key sk generated before, the OPE-encrypt algorithm encrypts the input plaintext m to a ciphertext c that preserves the correct order with other ciphertexts.

C. File-Injection Attacks

Aimed at SSE, file-injection attack (FIA) was proposed as a kind of query-recovery attack. Extending the definition of FIA, files do not only represent the data elements in NoSQL database, but also mean any kind of specific formatted data which fit the target system.

During an FIA, an adversary forges some data and sends them to the client from the server. After being encrypted by the client, the resultant ciphertexts of the forged data are sent back to the server for storing. Then, the adversary infers the responses, from the database management system (DBMS), to some old queries with the leakage of newly inserted data. The data will be recovered successfully when the adversary obtains enough leakage by file-injecting and querying continuously.

This kind of attack was proposed by Cash et al. [11], which is named as “known-document attack”. In some application scenarios like encrypted email system (e.g., Pmail [3]) or the systems in [21] and [25], FIA can be easily executed. Assuming that the server has already responded many email-search requests and recorded many encrypted data manipulation statements, the adversary can forge some emails and send to the client. When the new emails are encrypted and sent back to the DBMS, the adversary can take advantage of the entire set of ciphertexts, as well as the old queries, to collect more leakage and infer the corresponding plaintexts. In this work, we extend the concept of FIA, and show that it is more powerful when attacking OPE/ORE.

Actually, FIA only considers the leakage of the queries and the result sets of ciphertexts. As usual, FIA has three assumptions as following:

- The target system has a dependable component used for data-sharing or data-transmitting, just like an email server.
- The adversary possesses some old encrypted queries and can obtain the correct result sets from the server.
- The adversary possesses the plaintext space of the target ciphertexts, and can store correct ciphertexts by sending some forged data to the client without suspicion.

It is worth noting that the adversary can only get the ciphertexts included in the result sets. If the plaintext injected by the adversary does not match the queries, the corresponding ciphertext will not be known to it. Additionally, the basic assumption emphasizes that the adversary is unable to forge queries or execute any PPE/PRE algorithm.

D. Trapdoor Permutation

Trapdoor permutation (TDP) is a special case of one-way permutation.

Definition 3 (Trapdoor Permutation): A tuple of polynomial-time algorithms (KeyGen, $\mathsf{\Pi}$, Inv) over a domain $\mathcal{D}$ is a family of trapdoor permutations (or, sometimes, a trapdoor permutation informally), if it satisfies the following properties:

- $\mathsf{KeyGen}(1^\lambda) \to (I, \mathrm{td})$. On input a secure parameter λ, the parameter generation algorithm outputs a pair of parameters (I, td). Each pair of the parameters defines a set $\mathcal{D}_I = \mathcal{D}_{\mathrm{td}}$ with $|I| \geqslant \lambda$. Informally, I (resp., td) is said to be the public key (resp., secret key) of TDP.
- $\mathsf{KeyGen}_1(1^\lambda) \to I$. Let KeyGen_1 be the algorithm that executes KeyGen and returns I as the only result. Then (KeyGen_1, $\mathsf{\Pi}$) is a family of one-way permutations.
- $\mathsf{Inv}_{\mathrm{td}}(y) \to x$. Inv is a deterministic inverting algorithm such that, for every pair of (I, td) output by $\mathsf{KeyGen}(1^\lambda)$ and any $x \in \mathcal{D}_{\mathrm{td}} = \mathcal{D}_I$ and $y = \mathsf{\Pi}_I(x)$, it holds $\mathsf{Inv}_{\mathrm{td}}(y) = x$. For presentation simplicity, we also write the algorithm $\mathsf{Inv}_{\mathrm{td}}$ as $\mathsf{\Pi}_{\mathrm{td}}^{-1}$, and denoted by

$$\mathsf{\Pi}_I^k(x) = \overbrace{\mathsf{\Pi}_I(\mathsf{\Pi}_I(\cdots \mathsf{\Pi}_I(x) \cdots))}^{k \text{ TDPs}}$$

  for some integer $k \geq 1$.

TABLE I
NOTATIONS IN SECTION III

Notation	Meaning
m, c	Instance variables of plaintext, ciphertext.
$\mathcal{M}, \mathcal{C}, \overline{\mathcal{M}}$	Ordered spaces of plaintexts and ciphertexts. $\overline{\mathcal{M}}$ represents the ordered sub-space that contains high-frequency data.
$\mathbf{M}, \mathbf{C}$	Set of plaintexts and set of ciphertexts.
ω	Adversary makes at most ω file-injections.
$\mathbf{R}_q^i, \mathbf{R}_q$	The result set of query q before the $(i+1)$-th file-injection, and the current result set of query q.
$\mathbf{EDB}$	The encrypted database which contains its own data, encryption schemes and management system.
$c \xleftarrow{file\ injection} m$	Send the forged plaintext m to client, then send and store the resultant ciphertext c (by client) to the **EDB**.
$\mathsf{mid}(a,\ b)$	An arbitrary scheme for efficient median calculation, regardless of the round-off method.
x_l, x_r	The left and right boundary values of range condition in a state x, where x is a plain state *m* or a cipher state c.
$\mathbf{d}$, $\mathbf{dqueue}$	A structural body contains two indices $(a,\ b)$ and a queue of the structural body.

E. Forward Security

Forward security is a strong property of the dynamic SSE leakage profile. For a dynamic SSE scheme, its forward security means that: the previous data manipulations do not cause any leakage of the newly inserted data. Stefanov et al. [32] proposed this notion informally. Stefanov et al. [32] also proposed the concept of backward security, which ensures that the previous data manipulations do not leak any information about the newly deleted data. In this work, we extend this concept from SSE to OPE/ORE. Specifically, we give the definitions of forward security and backward security for ORE, as following:

Definition 4 (Forward Security): An $\mathcal{L}$-adaptively-secure ORE scheme is forward (resp., backward) secure if the leakage of update operation $\mathcal{L}_{\mathsf{update}}$ for $\mathsf{update} = \mathsf{add}$ (resp., $\mathsf{update} = \mathsf{delete}$) can be described as following:

$$\mathcal{L}_{\mathsf{update}}(\mathsf{update}, \mathbf{W}_{\mathsf{update}}) = \mathcal{L}(\mathsf{update}, \mathbf{IND}_{\mathsf{update}})$$

where add (resp., delete) denotes the addition (resp., deletion) of data. $\mathbf{W}_{\mathsf{update}}$ is the data set of the update operations, in which the data have their own data storage structure, indices, and constraints according to the database. $\mathbf{IND}_{\mathsf{update}}$ is a set that only describes the modified table (in SQL database) or the document (in NoSQL database) and the number of updated data.

III. File-injection Attacks on OPE/ORE

Unlike the FIA schemes against SSE, the FIA schemes against OPE/ORE are data-recovery attacks, which are more powerful. Moreover, the forged data are less likely to be detected because of the smaller forged part. Two FIA algorithms are presented in this section, which have a common basis on algorithm construction – binary search. The difference between the two FIA algorithms lies in the search types they employ: one uses the traditional binary search like the depth first search, and another uses the breadth first search. We first give the meaning of some notations in Table I, which is helpful to comprehend the two FIA algorithms presented in this section. We use the composite notation to represent the main part which is related to the additional part. For instance, m_c represents the plaintext of the ciphertext c, $\mathcal{M}_{x,y}$ represents the plaintext space between x and y, $\mathbf{d}.a$ represents the parameter a in the structural body $\mathbf{d}$.

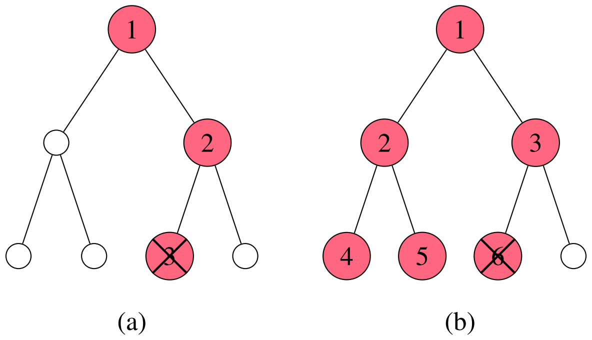
A. Binary Search

In this paper, we extend the definition of binary search which is also known as half-interval search or logarithmic search. The traditional binary search is a kind of (depth-first like) search algorithm, which finds the position of a target value within a sorted array by testing the order of the target value and the median value. In this work, we import the idea of breadth first search in the second FIA algorithm, with which we can get the relatively near data (around the target) that does not match the range condition. We regard it as a breadth first binary search like the breadth traversal of binary tree.

We show two types of binary search in Figure 1, where the colored nodes are the passed nodes with their order marked, and the crosses mark the target nodes. For the traditional binary search as described in part (a), we need just a little time in complexity $O(\log N)$ where N is the number of nodes in the binary tree, because we only need to find the target with a few data comparisons. But in part (b) of Figure 1, we have to traverse each data in every layer of the binary tree. Our FIA attacker, with the range query determined by $(m_l,\ m_r)$, needs to find a value m_1 matching the range condition, and a pair of relatively near unmatched values (m_2, m_3) in the file-injected dataset, such that $m_2 < m_l < m_1 < m_r < m_3$.

B. Basic Hierarchical FIA with order by Operation

Our FIA attacks use two kinds of order queries: *order by* queries and range queries. The *order by* operation is one of the Data Manipulation Languages (DMLs) that are based on the order of data, and the other one is the range query with relational operators like "<", ">" and so on. In Section III-B and III-C, we present the attack models and the FIA algorithms, assuming



(a) (b)

Fig. 1. Depth first binary search (a) and breadth first binary search (b).

the attacker possesses these two kinds of order queries respectively. In Section III-D, we also make some discussions about possessing the composition of these two kinds of order queries.

The attack model of basic hierarchical FIA, with *order by* operation, consists of the adversarial information (i.e., leakage) and the adversarial goal. As to the adversarial information, we limit the power of adversaries in order for more practical attacks in practice. Specifically, the adversary only possesses, as adversarial information, the plaintext space $\mathcal{M}$, the set $\mathbf{Q}$ of old *order by* queries, and the result sets of those queries with forged data. In particular, they do not have any information of the data not in the result sets of the old queries. About the adversarial goal, we partition it into two types: recovering the plaintext of a single ciphertext, and recovering the plaintexts of all the ciphertexts in the result sets. This partition facilitates the discussion of time complexity as we show later. We formalize the attack model as following:

$$\textbf{Leakage}: \qquad \mathcal{L}(\mathcal{M},\ \mathbf{Q},\ \mathbf{R}_{\mathbf{Q}} = \{ \bigcup_{q\in\mathbf{Q}, 0\le i\le\omega} \mathbf{R}^i_{q|\text{ordered}} \})$$

$$\textbf{Goal}: \qquad m_c\ (c \in \mathbf{R}_{q|\text{ordered}},\ q \in \mathbf{Q})\ \textit{or}\ \mathbf{M}_{\mathbf{C}}\ (\mathbf{C} = \bigcup_{q\in\mathbf{Q}} \mathbf{R}_q)$$

where $\mathbf{R}^i_{q|\text{ordered}}$ denotes the *ordered* result set for each *order by* query $q \in \mathbf{Q}$ before the $(i+1)$-th file-injection, $\mathbf{M}_{\mathbf{C}}$ ($\mathbf{C} = \bigcup_{q\in\mathbf{Q}} \mathbf{R}_q$) denotes the plaintext set $\mathbf{M}_{\mathbf{C}}$ corresponding to the ciphertext set $\mathbf{C}$ in the current result sets for all the queries in $\mathbf{Q}$. Here, $\mathbf{M}_{\mathbf{C}}$ can also be viewed as a mapping relation precisely, denoted $\mathcal{T}_{(\mathbf{M},\mathbf{C})}$, between all the ciphertexts in $\mathbf{C}$ (which includes all the original and forged data) and their corresponding plaintexts in $\mathbf{M}_{\mathbf{C}}$.

For ease of comprehension, **Algorithm 1** describes the elementary FIA based on utilizing a single *order by* query over an entire dataset.

Algorithm 1

$m_c \leftarrow \mathsf{FIA\_Orderby}(\overline{\mathcal{M}},\ \mathbf{EDB},\ c \in \mathbf{C}_{\mathbf{EDB}})$

```
$a \leftarrow -1,\ \phi \leftarrow 0,\ b \leftarrow |\overline{\mathcal{M}}|$
for $i \leftarrow 1$ to $\infty$ do
    $c_i \xleftarrow{file\ injection} \overline{\mathcal{M}}[\mathsf{mid}(a,\ b)]$
    if $(\mathsf{Comp}(c_i,\ c) = 0)$
        $\phi \leftarrow 1$, break
    else if $(\mathsf{mid}(a,\ b) = a$ or $\mathsf{mid}(a,\ b) = b)$ break
    else if $(\mathsf{Comp}(c_i,\ c) = 1)$
        $b \leftarrow \mathsf{mid}(a,\ b)$
    else $a \leftarrow \mathsf{mid}(a,\ b)$
    end if
end for
if $(\phi = 1)$ return $m_c \leftarrow \overline{\mathcal{M}}[\mathsf{mid}(a,\ b)]$
else return $m_c \leftarrow \mathsf{FIA\_Orderby}(\mathcal{M}_{a,\,b},\ \mathbf{EDB},\ c)$
```

In **Algorithm 1**, a and b are auxiliary indices, and ϕ is a flag variable. $\overline{\mathcal{M}}[\mathsf{mid}(a,\ b)]$ denotes the $(k{+}1)$-th element in $\overline{\mathcal{M}}$ for $k = \mathsf{mid}(a,\ b)$. The record of counter i is used for efficiency analysis in our experiments. The adversary will continually detect the plaintext of the target ciphertext c with an old query q by file-injection. We use $\mathsf{Comp}(c_i,\, c)$ to express the order result of query q about the target ciphertext c and the i-th injected ciphertext c_i, where the result expresses as following:

$$\mathsf{Comp}(c_i,\, c) = \begin{cases} 0 & c_i = c \\ 1 & c_i > c \\ -1 & c_i < c. \end{cases}$$

Because the plaintext space may be very large, and different values may have widely different frequency distributions in different fields, we adopt the idea of hierarchical plaintext space. In **Algorithm 1**, we divide the plaintext space into two hierarchies. We propose that the basic plaintext space $\overline{\mathcal{M}}$ should contain all the high-frequency plaintexts. If we cannot recover the correct plaintext of the target ciphertext from $\overline{\mathcal{M}}$, we recursively execute **Algorithm 1** with lower-frequency plaintext space. Because of the assumption that the union of all the plaintext spaces contains all of the possible values, our FIA algorithm can guarantee correctness and 100% recovery rate in theory.

Hierarchical plaintext space. In our basic FIA scheme, we use a hierarchical idea on the plaintext space to boost the attack efficiency. As the distributions that most of the plaintext utilization rates follow are heterogeneous, in general we cannot analyze the target ciphertexts by directly using general frequency statistics. In this work, we propose a solution to divide the entire plaintext space into several levels, which is relatively more precise and useful compared to those based on the general statistical data of word frequency.

Under the basic assumption of FIA, the adversary knows the semantic field and the plaintext space about the target ciphertexts. If there is not any perfect frequency statistics of the values in the target context, we adopt the method below with Normalized Google Distance (NGD) as the word semantic similarity measure. NGD is proposed by Cilibrasi et al. [15], and has drawn much attention in recent years because of its simplicity, low computational complexity, solid theoretical foundation, and the ability to achieve decent correlation levels with the human judgment of similarity. The main assumption of the NGD method is that the statistical frequency of data in the Web reflects their current similarity status in the target circumstance. The following steps show the details of this method:

- First, we calculate the NGD with the plaintexts, and the target context name n or the set $\mathbf{T}$ of its feature tags. For n or every $t \in \mathbf{T}$, we calculate $\mathsf{NGD}(n \text{ or } t,\, m)$ with every plaintext $m \in \mathcal{M}$ as following:

  $$\frac{\mathsf{max}(\log(|\mathbf{R}_{n \text{ or } t}|), \log(|\mathbf{R}_m|)) - \log(|\mathbf{R}_{n \text{ or } t} \cap \mathbf{R}_m|)}{\log(|\mathbf{R}|) - \mathsf{min}(\log(|\mathbf{R}_{n \text{ or } t}|), \log(|\mathbf{R}_m|))}$$

  where $\mathbf{R}_{n \text{ or } t}$ (resp., $\mathbf{R}_m$) is the query result set of the target context (resp., each plaintext) returned by Google search engine, and $|\mathbf{R}_{n \text{ or } t} \cap \mathbf{R}_m|$ is the co-occurrence times of $(n \text{ or } t,\, m)$, $|\mathbf{R}|$ is the number of searchable elements in the Web. We convert this distance value NGD into similarity value using

  $$\mathsf{p}(n \text{ or } t,\, m) = e^{-2\mathsf{NGD}(n \text{ or } t,\, m)}.$$

- Second, we cluster the plaintexts according to their semantic correlation indices $\mathsf{p}(n \text{ or } t,\, m)$. We suggest that the number of clusters should be two or three, as three hierarchies will make the plaintext space small enough in each hierarchy for even enormous plaintext space. Then we obtain three hierarchies of plaintext space, where the sub-hierarchy is divided into many parts by each plaintext in the super-hierarchy.

Time complexity. The time complexity of **Algorithm 1** is $O(\log|\mathcal{M}|)$ obviously in the worst condition for recovering one plaintext. When the adversarial goal is to recover all the N nonrepetitive ciphertexts in the entire result set, the time complexity is $O(N\log|\mathcal{M}| - N\log N)$ in the worst case. This means, in this case, the average time complexity of recovering a single ciphertext becomes smaller because the order of a ciphertext can be used for both sides. In other words, a file-injection for a target will reveal some order information about other target ciphertexts as well.

In **Algorithm 1**, we only take advantage of the leakage $\mathcal{L}_1(\mathcal{M},\ q,\ \mathbf{R}'_q)$, where $\mathbf{R}'_q = \mathbf{R}_q \setminus \mathbf{R}^0_q$ is the result set after file-injections excluding the original result set. Because the leakage of the original result set $\mathbf{R}^0_q$ is in the *ideal* leakage profile, we can only get some order information between the target ciphertext and other ciphertexts. In other words, we can rewrite the original result set as

$$\mathbf{R}^0_q = \{\mathbf{C}^-_{\text{ordered}},\, c_{\text{target}},\, \mathbf{C}^+_{\text{ordered}}\}$$

where $\mathbf{C}^-_{\text{ordered}}$ is the set of ordered ciphertexts which are smaller than the target, and $\mathbf{C}^+_{\text{ordered}}$ is the set of ordered ciphertexts which are greater than the target. Under the assumption of knowing nothing about the original ciphertexts except their order information, we can only take advantage of $|\mathbf{C}^-_{\text{ordered}}|$ and $|\mathbf{C}^+_{\text{ordered}}|$ to curtail the plaintext space. We delete the first $|\mathbf{C}^-_{\text{ordered}}|$ plaintexts and the last $|\mathbf{C}^+_{\text{ordered}}|$ plaintexts from the ordered plaintext space $\mathcal{M}$, and then we get a smaller new plaintext space $\mathcal{M}'$ for the target c_{target}. Thus, the time complexity of recovering a single ciphertext becomes $O(\log|\mathcal{M}'|)$ which is even smaller now.

In this way, the adversary can adaptively curtail the plaintext space according to the number of ciphertexts on both sides after each file-injection.

An improved method with extra leakage. If the adversary additionally knows the plaintext-ciphertext mapping relation of the OPE/ORE scheme, our FIA schemes will recover the target ciphertext faster. Specifically, the adversary can curtail the probable plaintext space immensely, by using the mapping function, the minimum ciphertext and the maximum ciphertext.

C. FIA with Range Queries

The attack model of FIA with range queries also consists of the adversarial information and the adversarial goal. As to the adversarial information, the adversary just has the plaintext space $\mathcal{M}$, the old range queries in $\mathbf{Q}$, and the result sets of those queries without inner order. In this condition, the leakage is less than that with *order by* operations, because the adversary only knows the result set matching the range conditions without knowing the inner order. As to the adversarial goal, the adversary needs to recover the boundary plaintexts of the range conditions as well as all the plaintexts matching the range conditions. We formalize the attack model as following:

$$\textbf{Leakage}: \qquad \mathcal{L}(\mathcal{M},\ \mathbf{Q},\ \mathbf{R_Q} = \{ \bigcup_{q\in\mathbf{Q},0\leq i\leq\omega} \mathbf{R}_q^i \})$$

$$\textbf{Goal}:\ \mathbf{M}_l,\ \mathbf{M}_r,\ \mathbf{M_C}\ (\mathbf{C} = \{c \mid q.c_l < c < q.c_r,\ q \in \mathbf{Q}\})$$

where $\mathbf{M_C}$ can be viewed as the mapping relation precisely, denoted $\mathcal{T}_{(\mathbf{M},\mathbf{C})}$, between all the ciphertexts in $\mathbf{C}$ (which includes all the original and forged data) and their plaintexts in $\mathbf{M_C}$, $\mathbf{R}_q$ is not ordered, $\mathbf{M}_l$ and $\mathbf{M}_r$ contain all the boundary plaintexts of the range queries in $\mathbf{Q}$. In our construction, we design 3 steps to achieve the goal as following:

- First, the adversary must find a plaintext matching the range condition, whether its ciphertext is in the original **EDB** or not.
- Second, the adversary recovers the boundary plaintexts using **Algorithm 1**.
- Third, the adversary recovers all the plaintexts of the ciphertexts matching the range condition by several file-injections.

Here, to describe the FIA scheme briefly, **Algorithm 2** is based on utilizing a single range query without any *order by* operation. In the following descriptions, q represents the range query with the boundary ciphertexts denoted $q.c_l$ and $q.c_r$ respectively.

Algorithm 2

$m_l,\ m_r,\ \mathbf{M}_{\mathbf{C}_q} \leftarrow \mathsf{FIA\_Rangequery}(\mathcal{M},\ \mathbf{EDB},\ q)$

```
$a \leftarrow -1,\ b \leftarrow |\mathcal{M}|,\ \mathbf{d} \leftarrow (a,\ b)$
insert $\mathbf{d}$ into the queue $\mathbf{dqueue}$
while $\mathbf{dqueue} \neq \emptyset$
    take out the first $\mathbf{d}$ in $\mathbf{dqueue}$, $a \leftarrow \mathbf{d}.a,\ \ b \leftarrow \mathbf{d}.b$
    $c \xleftarrow{file\ injection} \mathcal{M}[\mathsf{mid}(a,\ b)]$
    if $|\mathbf{R}'_q| \neq |\mathbf{R}_q|$ break
    if $(\mathsf{mid}(a,\ \mathsf{mid}(a,\ b)) \neq a$ and $\mathsf{mid}(a,\ \mathsf{mid}(a,\ b)) \neq \mathsf{mid}(a,\ b))$
        $\mathbf{d} \leftarrow (a,\ \mathsf{mid}(a,\ b))$, insert $\mathbf{d}$ into the queue $\mathbf{dqueue}$
    end if
    if $(\mathsf{mid}(\mathsf{mid}(a,\ b),\ b) \neq b$ and $\mathsf{mid}(\mathsf{mid}(a,\ b),\ b) \neq \mathsf{mid}(a,\ b))$
        $\mathbf{d} \leftarrow (\mathsf{mid}(a,\ b),\ b)$, insert $\mathbf{d}$ into the queue $\mathbf{dqueue}$
    end if
$m_l \leftarrow \mathsf{FIA\_Orderby}(\mathcal{M}_{a,\mathsf{mid}(a,b)},\ \mathbf{EDB},\ q.c_l)$
$m_r \leftarrow \mathsf{FIA\_Orderby}(\mathcal{M}_{\mathsf{mid}(a,b),b},\ \mathbf{EDB},\ q.c_r)$
$\mathbf{M}_{\mathbf{C}_q} \leftarrow$ do file-injections from $m_l$ to $m_r$ and get their mapping table or corresponding plaintext set briefly
return $m_l,\ m_r,\ \mathbf{M}_{\mathbf{C}_q}$
```

In **Algorithm 2**, we adopt the breadth first search, because under the assumption of FIA the adversary does not know the order between file-injected data and the boundary ciphertexts in case the file-injected data do not match the range condition. With this limitation, the breadth first search is beneficial to find a plaintext matching the condition, and to get the relatively near unmatching plaintexts that are necessary for recovering the boundary plaintexts. Then, the boundary plaintexts m_l and m_r are recovered by calling **Algorithm 1**. Finally, the mapping table are constructed by several file-injections over the entire plaintext set matching the condition.

Most of the boundary values are very special in practice. For instance, the numbers, which are the multiple of $10^{\gamma}(\gamma = 0, 1, 2...)$, are frequently used for range query over numerical data; and the 26 letters are used for the same purpose over string data. Based on the different frequency of the plaintexts which are between every two adjacent common boundary plaintexts, the adversary may recover them more rapidly by several file-injections instead of the first step.

Time complexity. As the method of the first step is aimed at finding a plaintext matching the range condition, the time complexity of this part is just like a random search over an interval. Therefore, the time complexity of the first step is $O(\frac{|\mathcal{M}|}{d})$, where d is the number of plaintexts that match the range condition in the entire plaintext space $\mathcal{M}$.

If z is the times of file-injections in the first step, the entire time complexity is

$$O\left(\frac{|\mathcal{M}|}{d} + \log\frac{|\mathcal{M}|}{2^{\lfloor \log z \rfloor}} + d\right)$$

where $\lfloor\ \rfloor$ represents the round down method. To be precise, the adversary actually does not need to file-inject all the plaintexts between m_l and m_r in the third step, as some of them must have been file-injected during the execution of the two steps before.

The calculation of time complexity is based on the polynomial size message spaces. Just like other OPE/ORE/DET-attack works in the literature, we do not consider the scenario where the goal is recovering the plaintexts of long texts.

D. Discussions on FIA with both order by Queries and Range Queries

In this section, we discuss the extra leakage when both *order by* operations and range operations are compounded or co-occurred in the query set **Q**, and its impact on FIA. We consider the following three cases.

In the first case, when a query statement contains both the *order by* operator and the range condition, the *order by* operation additionally causes the leakage of inner order (particularly, the order information between every pair of inner ciphertexts). The extra leakage can be used to curtail the plaintext space when calling **Algorithm 1** in the second step of **Algorithm 2**.

In the second case, when the range of an *order by* operation contains the range of a range query, the range query with small range leaks the feature of the boundary data. Some special values like the 26 letters discussed above may be recovered as the boundary plaintexts of the range conditions, which divide the recovering puzzle into several small puzzles with smaller plaintext spaces.

In the third case, the adversary has multiple range query statements. If their ranges do not have any intersection, they will not cause extra leakage other than dividing the puzzle to be solved by FIA into multiple irrelevant smaller puzzles. However, when they have an intersection and at least one boundary ciphertext of a range query is included in the result set of another one, the adversary will know the order of several sets of ciphertexts. In the latter case, the boundary ciphertexts can be recovered more quickly by the eigenvalue analysis over the plaintext space.

E. On FIA against Frequency-Hiding OPE

Our FIA attacks can be applied to the frequency-hiding OPE schemes proposed in [24] and [31], but possibly with different difficulties. In this case, after a series of file-injections, the puzzle faced by the FIA attacker turns into inferring the plaintext of the target ciphertext from two adjacent plaintext candidates. In general, we can adjust our FIAs to inject more data until two frequency-hiding ciphertexts (for any one of the two plaintext candidates) are stored on the different sides of the target value.

Both the scheme in [24] and that in [31] are based on interactive encryption (but in different phases). In [24], the order between two ciphertexts (one existing and one updating) of the same value is randomly claimed by the client to the server. And in [31], a random data suffix is used for ensuring frequency-hiding. Hence, for the frequency-hiding OPE construction proposed in [31], the random space is fixed; while for that in [24], the random space will enlarge if more equivalent data are inserted. In other words, for the OPE scheme proposed in [24], the recovery difficulty will increase in case the data have a high repetitive rate; while for that in [31], the recovery difficulty with our FIAs will not increase when we continuously inject the forged data. The increased recovery difficulty will not decrease the accuracy, but more file-injections are demanded for data recovering. Here, our FIA against the frequency-hiding POPE scheme in [31] is based on the assumption that the interactive processes between the server and the client are executed automatically without client authorization. We suggest that this is the natural and more common application scenario of OPE/ORE in practice, as otherwise the client may be overburdened or cumbersome.

F. Comparison among FIA and other generic attacks against OPE/ORE

Known- and chosen-plaintext attacks (CPAs) have been considered in many OPE/ORE works. To the best of our knowledge, the latest discussion of CPA is in [20]. Besides these attacks against OPE/ORE, inference attack (IA) is also a kind of powerful generic attack which has been described in [28] detailedly. In this subsection, we make brief comparisons among FIA, CPA and IA.

About the adversarial prerequisite, these three attacks all need an unbroken auxiliary dataset as the plaintext space, but only IA needs the data-frequency statistics. About the source of leakage information, CPA only utilizes the data which are chosen by

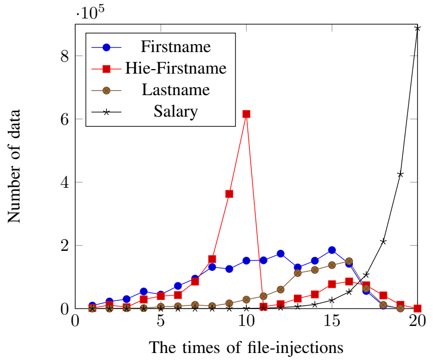


Fig. 2. The times of file-injections for recovering a single datum.

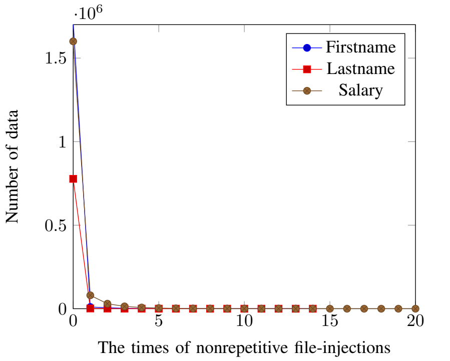


Fig. 3. The times of nonrepetitive file-injections for recovering entire dataset.

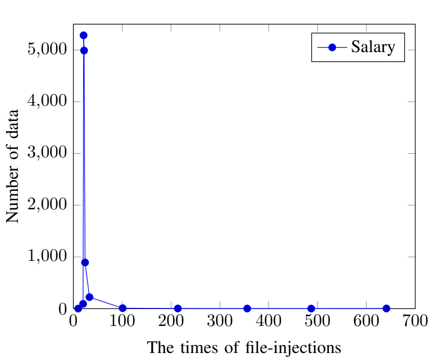


Fig. 4. The times of file-injections on frequency-hiding POPE.

the adversary and encrypted by the encryption scheme in the system; IA only utilizes the original ciphertexts in the EDB; but FIA can utilize both the main forged data (which are chosen by the adversary and encrypted by the system) and the secondary original ciphertexts in the EDB. Specially, the leakage information in FIA is obtained through old queries. In other words, the ciphertexts, which are not included in the result sets of the old queries, are not required in the adversarial prerequisite. As for the comparison algorithm, CPA and IA must use it; FIA only calls it normally through the old queries. About the performance, 100% accuracy can be achieved easily with either CPA or FIA, but it is difficult to achieve with IA. Additionally, CPA and FIA can attack the frequency-hiding OPE, but IA cannot do this without the decline of accuracy and applicability.

Overall, FIA and CPA require less auxiliary information, while IA needs more auxiliary information. CPA needs more adversarial abilities, but FIA and IA need less adversarial abilities. Because of the utilization rate of leakage, FIA is more efficient than CPA, while the efficiency of IA depends on the size of plaintext space overly.

G. *Experiments of FIAs*

In this subsection, we present the results of our FIA experiments against the OPE/ORE with *ideal* leakage as well as a frequency-hiding OPE. The FIA algorithms are implemented in C++, and our experiments were performed using a single core on a machine with an Intel Pentium G2020 2.9GHz CPU and 4GB available RAM. The FIA experiment with range queries relies closely upon the range width d in reality, which is, however, not easy to be obtained or evaluated. If we regard the range width as a variable, the results will be four-dimensional and inenarrable. As a consequence, our FIA implementations are aimed at the situation of only possessing *order by* queries in this paper. Moreover, the experimental results of FIA with range queries will be included and analyzed detailedly in the extended version of this paper.

Datasets selection. We used the California public employee payroll data from 2014 [1] as the target dataset. We did some preprocessing because there are some invalid data in this dataset; for example, "not provided" or a single letter in the name set, and negative values in the salary set. After the filtration, we got the valid datasets of firstname, lastname and salary, which contain 1739637 firstnames, 1736490 lastnames and 1739634 salary data respectively. We also need some public auxiliary datasets as the plaintext spaces of target data. For salary space, we do not need auxiliary dataset and just set 0 to 1000000 as the range of salary. For the firstname space and the lastname space, we select the union set of the statistics for baby names gathered by the US Social Security Administration [4] from 1951 to 1996 (so that the parties selected are of age older than 18). The union set contains 56680 unique names that have been used more than 4 times in a certain year.

Experiment results. We first executed our FIA schemes on the *ideally* secure OPE proposed by Popa et al. [29]. We remark that the experiment results will be the same for other OPE/ORE schemes without forward security, as we only leverage the order leakage. The results show that we got 100% recovery rate on firstname and salary datasets, but the recovery rate of lastnames is 44.96%. The reason of the lower recovery rate of lastnames is that our auxiliary dataset itself does not contain the unrecovered lastnames which are too rare to record in [4] or belong to the persons who were not born in the United States from 1951 to 1996.

Because the recovery rate is only connected with the integrity of the adversarial auxiliary dataset, we judge the experiment results through the times of file-injections. Moreover, the adversary will be detected less likely with fewer times of file-injections. We mainly executed two experiments about the OPE/ORE with *ideal* leakage profile as following:

- The first experiment is aimed at recovering every datum in the target datasets isolatedly.
- The second experiment is aimed at recovering all the data in the target datasets synchronously. Most of the file-injections in the first experiment are repeatedly used for revealing the order information of disparate data, but they will not be executed repeatedly in the second experiment.

TABLE II
NOTATIONS IN SECTION IV

Notation	Meaning
e	An instance of intermediate ciphertext output by the original ORE or OPE scheme in $\mathbf{EDB}$.
Π, sk, pk	A TDP scheme and its secret key, public key.
σ	All the variables which play a role in the original OPE or ORE scheme.
$\mathrm{OT}, \mathbf{OT}, \mathcal{OT}$	An instance of order token, the set of order token stored on the client and the domain of order token.
i	The counter of order tokens, which is equal to the number of order tokens minus one.
s	The id of an order space, which ensures that the data in different order spaces cannot be ordered.

According to the times of file-injections for recovering every datum, we count the number of data and draw the charts in Figure 2 and Figure 3. In other words, the x-axis means the number of injected files that we used for recovering each datum, and the y-axis means the statistical magnitude, which is the result of counting all attacked data basing on x-axis.

Figure 2 (which includes all the repetitive injected files used for recovering different data) shows the times of file-injections for recovering a single datum, where the approximate average times for recovering a firstname is 11.013, the approximate average times for a successfully recovered lastname is 13.801, and the approximate average times for recovering a salary is 19.022. We used the hierarchical idea for recovering firstnames. But we did not use this idea for recovering lastnames and salaries, because this idea does not well fit the task for recovering intensive high-frequency data like salary or the task for recovering unapparent-frequency data like lastname. Specifically, it will cause too many layers of the attack order tree in these situations. We set nine hundred as the size of $\overline{\mathcal{M}}$ and set two as the number of hierarchies for this hierarchical FIA experiment. The result of this hierarchical attack is showed as the line chart named Hie-Firstname in Figure 2, and the approximate average times for recovering a firstname is 10.320. As we discussed in Section III-B, the adversary always targets at the entire dataset, and each file-injection can actually reveal some order information among all the set. Hence, the times of file-injections can be much less in the actual attack situation. Figure 3 shows the times of nonrepetitive file-injections for recovering entire dataset, where the approximate average times for recovering a firstname is 0.023, the approximate average times for recovering a lastname is 0.011, and the approximate average times for recovering a salary is 0.150.

We also executed our FIAs on the frequency-hiding POPE proposed in [31], under the assumption that the interactive processes between client and server are automatically run without client authorization (as discussed at the end of Section III-E). To the best of our knowledge, this is the first attack against POPE with frequency-hiding. The frequency-hiding method of POPE in [31] is to add a random fractional part to each plaintext prior to encrypting. In our experiment, the random fractional part has two decimal places. We selected the first 15814 salary data as the attack target that contains values from 1 to 756351. Figure 4 is the result of file-injection times, which shows that the approximate average times of file-injections against the frequency-hiding POPE in [31] are 29.220. For simplicity, not all points are drawn in this line chart. Because of the randomness of the encryption algorithm in POPE, the experiment result is not deterministic.

IV. A Compilation Framework for Forward Secure ORE

To the best of our knowledge, all the existing OPE and ORE schemes in the literature do not have forward security *precisely*. Here, we use "precisely" with the only special case of POPE [31] (discussed in Section III-E) in mind. In [31], there is not any statement about whether the interactive processes need a client authorization or not. For the common application scenarios of OPE/ORE in practice, there is not any client authorization for querying. However, if the client authorization is mandated, POPE has forward security.

In the general case, the ciphertexts in **EDB** does not cover the entire ciphertext space. In other words, the ciphertexts in **EDB** are not dense usually. Thus, it is difficult to recover all the stored ciphertexts correctly with the limited leakage of OPE/ORE. However, according to our FIA constructions and experiments, FIA schemes are powerful and effective in recovering data encrypted by OPE/ORE without forward security in practice. Thus, it is desirable to have practical forward secure OPE/ORE schemes. In this section, we present a practical compilation framework that transforms most of the existing OPEs/OREs into forward secure ORE schemes which is based on trapdoor permutations (TDPs). To ease the comprehending of the framework, we first give the meaning of some notations in Table II.

A. Basic Ideas

With forward security, the *add* operation should leak nothing to server. In other words, the server should not distinguish between the ciphertexts output by a forward secure ORE and the ciphertexts encrypted by a perfect encryption scheme, when they are just inserted to the database before undergoing any search operation. In order to realize this goal, the ciphertexts should be salted. And we use TDP to link the salts to reduce the bandwidth consumption.

The salt is a hash value of an order token OT in our construction. To insert a new datum to EDB (say, the $(i+1)$-th insertion, $i \geq 0$), the client generates an order token OT_i based on the TDP scheme Π, its secret key *sk*, and the last order token OT_{i-1}. If OT_i $(i = 0)$ is the first order token in the order space, it will be randomly selected from the domain of order token. In order to reduce the client storage, the client only stores the latest order token and the corresponding counter in our basic construction. When an order query needs to be executed, the client sends the current token and the counter to the server. The server can then calculate all the order tokens with the public key *pk*, and gets the original OPE/ORE ciphertexts by desalting operations. At last, the client will receive the correct comparison result which is calculated with the comparison algorithm of the original OPE/ORE by the server.

B. The Compilation Framework

Given any OPE or ORE scheme Γ, which is briefly denoted by $\Gamma = (\mathsf{ORE\_Setup}, \mathsf{ORE\_Encrypt}, \mathsf{ORE\_Compare})$, the compiled ORE scheme is described as **Algorithm 3**, which is denoted by $\Gamma_{fp} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Compare})$. In **Algorithm 3**, the parts of the original OPE/ORE are described briefly.

In our construction, we use s to represent the order space of the related data on which order queries may be executed. For SQL databases, s can represent a column of a table. And for NoSQL databases, s can represent a set of documents. For each order space s, the key k_s of hash function H is calculated by pseudo-random function $\mathsf{PRF}_{k_0}[s]$. The order tokens are calculated with TDP one by one in sequence, and the hash values of these tokens will xor the original OPE/ORE ciphertexts to generate the final ciphertexts without extra storage consumption at the server side. The salt of the final ciphertext is of λ bits, and will be desalted in the comparison algorithm. In the comparison algorithm, c_{s_α} and c_{s_β} are two ciphertexts to be compared in the order space s with their indices α and β respectively.

Algorithm 3 Γ_{fp}

```
Setup(1^λ)
  $\mathsf{ORE\_Setup}(1^\lambda)$
  $\mathbf{OT} \leftarrow$ empty map
  $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $k_0 \xleftarrow{\$} \{0,1\}^\lambda$
  return $(pk, (sk, \mathbf{OT}))$
Encrypt(add, σ, EDB, m, s)
  Client :
  $k_s \leftarrow \mathsf{PRF}_{k_0}[s]$
  $(\mathrm{OT}_i, i) \leftarrow \mathbf{OT}[s]$
  if $(\mathrm{OT}_i, i) = \perp$ $\{i \leftarrow -1,\ \mathrm{OT}_{i+1} \xleftarrow{\$} \mathcal{OT}\}$
  else $\mathrm{OT}_{i+1} \leftarrow \Pi_{sk}^{-1}(\mathrm{OT}_i)$
  end if
  $\mathbf{OT}[s] \leftarrow (\mathrm{OT}_{i+1}, i+1)$
  $c_{i+1} \leftarrow \mathsf{H}_{k_s}(\mathrm{OT}_{i+1}) \oplus \mathsf{ORE\_Encrypt}(\mathsf{add}, \sigma, m)$
  Send $c_{i+1}$ to server.
  Server :
  $\mathbf{EDB} \Leftarrow c_{i+1}$                // Insert $c_{i+1}$ into EDB
Compare(σ, EDB, c_{s_α}, c_{s_β}, s)
  Client :
  $k_s \leftarrow \mathsf{PRF}_{k_0}[s]$
  $(\mathrm{OT}_i, i) \leftarrow \mathbf{OT}[s]$
  if $(\mathrm{OT}_i, i) = \perp$ return $\emptyset$
  Send $(\mathrm{OT}_i, i, k_s)$ to server.
  Server :
  $e_{s_\alpha} \leftarrow c_{s_\alpha} \oplus \mathsf{H}_{k_s}(\Pi_{pk}^{i-\alpha}(\mathrm{OT}_i))$
  $e_{s_\beta} \leftarrow c_{s_\beta} \oplus \mathsf{H}_{k_s}(\Pi_{pk}^{i-\beta}(\mathrm{OT}_i))$
  $b \leftarrow \mathsf{ORE\_Compare}(\sigma, e_{s_\alpha}, e_{s_\beta}, s)$
  Send the result $b$ to client
```

For data deletion, the first method is to store the deleted data in another **EDB**. Then a checking procedure should be added into the comparison algorithm to ensure that the ordered data have not been deleted. When the computing and bandwidth resource are sufficient and the server does not receive any query, the system can execute a **refresh** operation, which deletes all the deleted data from both **EDB** and recalculate all the order tokens and ciphertexts in sequence for curtailing storage and lifting efficiency. In this case, the scheme also achieves backward security. The second method is to insert the sequence numbers into **EDB** when inserting data. Then, for the situation where some data have been deleted, the server can calculate the salts of the remaining data by executing TDP exact times according to the interval leaked by sequence numbers.

For batch encryptions, we can simply arrange all the elements in random order, and run the Encrypt algorithm in sequence. If batch encryptions are common in the system, we can add an extra batch index in the database for each datum, and use the same calculated order token for encrypting all the plaintexts in a batch. This solution reduces the average computational complexity at the expense of leaking some information (eg. equality) of the elements in a batch.

C. Analysis of Forward Security

The following theorem shows that, given any OPE/ORE scheme, the complied scheme enjoys forward security, while preserving the security of the given OPE/ORE scheme.

Theorem 1 (Forward Security of Γ_{fp}): Assuming Γ is an OPE/ORE scheme with a leakage profile $\mathcal{L}_\Gamma$, the leakage profile $\mathcal{L}_{\Gamma_{fp}}$ of the compiled OPE/ORE scheme Γ_{fp} has two parts as following:

$$\mathcal{L}_{\mathsf{update}}(\mathsf{add},\, m,\, s) = (\mathsf{add},\, j,\, s)$$

$$\mathcal{L}_{\mathsf{compare}}(c_1,\, c_2,\, s) = (\mathbf{op}(s),\, \mathbf{Hist}(s),\, \mathcal{L}_\Gamma)$$

where j is a timestamp initially set to 0, the order pattern $\mathbf{op}(s)$ of an order space s includes the leakage tuple $(j,\, s)$, the adding history $\mathbf{Hist}(s)$ of s includes the leakage tuple $(j,\, \mathsf{add},\, e)$ of all the ciphertexts and the indices of s.

In our framework, the ciphertexts output by the original OPE/ORE xor one-way-generated salts. Hence, the newly inserted data leak nothing to the server if they have not been queried. Once the data have been queried and desalted, the ciphertexts turn into the security level of the original OPE/ORE scheme for the adversary with continuous monitoring. Hence, the security of the composite forward secure ORE cannot be weaker than that of the original OPE/ORE. On the other hand, our compilation framework is powerful against FIAs, because the forged data will not leak any information with the old queries. The data need a new credible order query from the client to desalt.

Proof: In order to give the entire proof of the theorem, we derive a game **G** and a simulator **S** from the security model of forward secure ORE in the real world. We let **A3** denote **Algorithm 3**. Moreover, the presentation of our proof follows the paradigm proposed in [9].

FS-OREReal. The security model of forward secure ORE in the real world, which is denoted by $\textbf{FS-OREReal}_A^{\mathbf{A3}}(\lambda)$ w.r.t. a security parameter λ, a PPT adversary A and our compilation framework **Algorithm 3**.

Game G. In the construction below, we generate the keys of order spaces randomly and independently, and store them in a map **KEY** instead of calling PRF. We also store all the order tokens generated by TDP Π in the map **OT**. Instead of generating the salts by hash function H, the game randomly generates the salts and stores them in the map **H**. The function H' is used for ensuring never generating two different salts for the inputs of a same tuple (k, OT_x). The function H' will randomly generate the result if the map **H** does not include the tuple (k, OT_x). But if the order token OT_x gets a collision with another order token, the flag *Error* will be set to 1, and the function will return the corresponding salt of the equivalent token. We label this part with a box, because it is not included in the intermediate game **G'**. Another part removed from **G**, also labelled with box, for the intermediate game **G'** is in the Encrypt algorithm. Specifically, the intermediate game $\mathbf{G}'$ is gotten by removing the boxed parts from **G**. When the salt of a new order token is existing, the flag *Error* will be set to 1.

Because of this, the outputs of the function H' in game **G** and the outputs of keyed hash function H are perfectly indistinguishable. Hence, if an adversary can distinguish between **FS-OREReal** and **G**, we can construct a reduction to distinguish between PRF and a truly random function. Specifically, we can make a formal reduction with an efficient adversary A_1 as following:

$$|\mathbb{P}[\textbf{FS-OREReal}_A^{\mathbf{A3}}(\lambda) = 1] - \mathbb{P}[\mathbf{G} = 1]| \leq \mathbf{Adv}_{\mathsf{PRF},A_1}^{\mathrm{PR}}(\lambda).$$

where PR denotes pseudo-randomness

According to the difference between **G** and **G'** described above, we have: the advantage of distinguishing between **G** and **G'** is smaller than the probability that the flag *Error* is set to 1 in **G**. Specifically:

$$|\mathbb{P}[\mathbf{G} = 1] - \mathbb{P}[\mathbf{G}' = 1]| \leq \mathbb{P}[Error \text{ is set to 1 in } \mathbf{G}].$$

Note that, the flag *Error* is set to 1 in **G**, only if an efficient adversary A_2 breaks the one-wayness of TDP Π. About the first boxed part in the game **G**, the error occurs only when the collision of at least two order tokens in an order space happens. In other words, the error occurs when the values generated by TDP Π form a token ring without one-wayness. About the second boxed part in the game **G**, the error occurs when the order token causes a collision with another token in certain of the order spaces. Hence, the advantage of distinguishing **G** and **G**' can be reduced to that of breaking the one-wayness of TDP with A_2, as following:

$$|\mathbb{P}[\mathbf{G} = 1] - \mathbb{P}[\mathbf{G}' = 1]| \leq \mathcal{N} \cdot \mathbf{Adv}_{\Pi,A_2}^{\text{one-wayness}}(\lambda)$$

where $\mathcal{N}$ is the total number of the data in all the order spaces.

```
Game G
Setup($1^\lambda$)
  ORE_Setup($1^\lambda$)
  $\mathbf{OT}, \mathbf{KEY}, \mathbf{H} \leftarrow$ empty map
  $(sk, pk) \leftarrow$ KeyGen($1^\lambda$)
  $Error \leftarrow 0$
  return $((pk, \mathbf{H}), (sk, \mathbf{OT}, \mathbf{KEY}))$
Encrypt(add, $\sigma$, $\mathbf{EDB}$, $m$, $s$)
    Client :
  $k_s \leftarrow \mathbf{KEY}[s]$
  if $k_s = \perp$  $\mathbf{KEY}[s] \leftarrow k_s \xleftarrow{\$} \{0,1\}^\lambda$
  $(\mathrm{OT}_0, \dots, \mathrm{OT}_i, i) \leftarrow \mathbf{OT}[s]$
  if $(\mathrm{OT}_i, i) = \perp$  $\{i \leftarrow -1,\ \mathrm{OT}_{i+1} \xleftarrow{\$} \mathcal{OT}\}$
  else $\mathrm{OT}_{i+1} \leftarrow \Pi_{sk}^{-1}(\mathrm{OT}_i)$
  end if
  $H_{i+1} \xleftarrow{\$} \{0,1\}^\lambda$
  if $\mathbf{H}[k_s, \mathrm{OT}_{i+1}] \neq \perp$ then
      $Error \leftarrow 1,\ H_{i+1} \leftarrow \mathsf{H}'(k_s, \mathrm{OT}_{i+1})$
  end if
  $\mathbf{H}[k_s, \mathrm{OT}_{i+1}] \leftarrow H_{i+1}$
  $\mathbf{OT}[s] \leftarrow (\mathrm{OT}_0, \dots, \mathrm{OT}_{i+1}, i+1)$
  $c_{i+1} \leftarrow \mathbf{H}[k_s, \mathrm{OT}_{i+1}] \oplus$ ORE_Encrypt(add, $\sigma$, $m$)
  Send $c_{i+1}$ to server.
    Server :
  $\mathbf{EDB} \Leftarrow c_{i+1}$
Compare($\sigma$, $\mathbf{EDB}$, $c_{s_\alpha}$, $c_{s_\beta}$, $s$)
    Client :
  $k_s \leftarrow \mathbf{KEY}[s]$
  if $k_s = \perp$  return $\emptyset$
  $(\mathrm{OT}_0, \dots, \mathrm{OT}_i, i) \leftarrow \mathbf{OT}[s]$
  if $((\mathrm{OT}_0, \dots, \mathrm{OT}_i, i) = \perp$  return $\emptyset$
  Send $((\mathrm{OT}_0, \dots, \mathrm{OT}_i, i, k_s)$ to server.
    Server :
  $e_{s_\alpha} \leftarrow c_{s_\alpha} \oplus \mathsf{H}'(k_s, \mathrm{OT}_\alpha)$
  $e_{s_\beta} \leftarrow c_{s_\beta} \oplus \mathsf{H}'(k_s, \mathrm{OT}_\beta)$
  $b \leftarrow$ ORE_Compare($\sigma$, $e_{s_\alpha}$, $e_{s_\beta}$, $s$)
  Send the result $b$ to client
$H \leftarrow \mathsf{H}'(k, \mathrm{OT}_x)$
  $H \leftarrow \mathbf{H}[k, \mathrm{OT}_x]$
  if $H = \perp$ then
      $H \xleftarrow{\$} \{0,1\}^\lambda$
      if $\exists s, i$ s.t. $\mathrm{OT}_x = \mathrm{OT}_i \in \mathbf{OT}[s]$ then
          $Error \leftarrow 1,\ H \leftarrow \mathbf{H}[\mathbf{KEY}[s], \mathrm{OT}_i]$
      end if
      $\mathbf{H}[k, \mathrm{OT}_x] \leftarrow H$
  end if
  return $H$
```

Simulator S. We use the simulator **S** and its leakage function $\mathcal{L}_{\mathbf{S}}$ to describe the ideal forward security of ORE constructions. Compared to the game **G'**, the simulator uses the counter $\bar{s}$ uniquely mapped from order space s with the leakage function. As the code for the oracle H' is useless now, it is removed from the description of the simulator. In addition, the leakage of the order information is only revealed when the ciphertexts are going to execute the algorithm Compare.

We show that the game **G'** and the simulator **S** are indistinguishable. For data encryption, it is immediate as the scheme is outputting a fresh random bit string for each update in **G'**. For data searching, using the adding history **Hist**, the simulator constructs the oracle H' which is subject to revealing the order correctly with the corresponding order token generated from OT_0. Hence,

$$\mathbb{P}[\mathbf{G}' = 1] - \mathbb{P}[\textbf{FS-OREIdeal}^{\mathbf{A3}}_{A,\mathbf{S},\mathcal{L}_\mathbf{S}}(\lambda) = 1] = 0.$$

```
Simulator S
Setup($1^\lambda$)
  $\mathsf{ORE\_Setup}(1^\lambda)$
  $\mathbf{OT}, \mathbf{KEY}, \mathbf{C} \leftarrow$ empty map
  $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $counter \leftarrow 0$
  return $((pk, \mathbf{C}), (sk, \mathbf{OT}, \mathbf{KEY}))$
Encrypt()
    Client :
  $\mathbf{C}[counter] \xleftarrow{\$} \{0,1\}^\lambda$
  Send $\mathbf{C}[counter]$ to server.
  $counter \leftarrow counter{+}1$
    Server :
  $\mathbf{EDB} \Leftarrow \mathbf{C}[counter]$
Compare($\mathbf{op}(s)$, $\mathbf{Hist}(s)$, $\sigma$, $\mathbf{EDB}$, $\mathbf{C}[\alpha]$, $\mathbf{C}[\beta]$, $s$)
    Client :
  $\bar{s} \leftarrow \min \mathbf{op}(s)$
  if $\bar{s} = \perp$ then
      $k_{\bar{s}} \xleftarrow{\$} \{0,1\}^\lambda$
      $\mathrm{OT}_0 \xleftarrow{\$} \mathcal{OT}$
  else
      $k_{\bar{s}} \leftarrow \mathbf{KEY}[\bar{s}]$
      $\mathrm{OT}_0 \leftarrow \mathbf{OT}[\bar{s}]$
  end if
  Parse $\mathbf{Hist}(s)$ as $[(counter_0, \mathsf{add}, e_0 \leftarrow \mathsf{ORE\_Encrypt}(\mathsf{add}, \sigma, m_0)), \ldots, (counter_i, \mathsf{add}, e_i \leftarrow \mathsf{ORE\_Encrypt}(\mathsf{add}, \sigma, m_i))]$
  if $i = 0$ return $\emptyset$
  Program $\mathsf{H}'$ s.t. $\mathsf{H}'(k_{\bar{s}}, \mathrm{OT}_\alpha \leftarrow \Pi_{sk}^{-\alpha}(\mathrm{OT}_0)) \leftarrow \mathbf{C}[\alpha] \oplus e_\alpha$ and $\mathsf{H}'(k_{\bar{s}}, \mathrm{OT}_\beta \leftarrow \Pi_{sk}^{-\beta}(\mathrm{OT}_0)) \leftarrow \mathbf{C}[\beta] \oplus e_\beta$
  Send $((\mathrm{OT}_\alpha, \mathrm{OT}_\beta, k_{\bar{s}})$ to server.
    Server :
  $e_{s_\alpha} \leftarrow \mathbf{C}[\alpha] \oplus \mathsf{H}'(k_{\bar{s}}, \mathrm{OT}_\alpha)$
  $e_{s_\beta} \leftarrow \mathbf{C}[\beta] \oplus \mathsf{H}'(k_{\bar{s}}, \mathrm{OT}_\beta)$
  $b \leftarrow \mathsf{ORE\_Compare}(\sigma, e_{s_\alpha}, e_{s_\beta}, s)$
  Send the result $b$ to client
```

Conclusion. Combining all the reductions above, there exists two efficient adversaries $A_1,\ A_2$ such that

$$|\mathbb{P}[\textbf{FS-OREReal}^{\mathbf{A3}}_{A}(\lambda) = 1] - \mathbb{P}[\textbf{FS-OREIdeal}^{\mathbf{A3}}_{A,\mathbf{S},\mathcal{L}_\mathbf{S}}(\lambda) = 1]|$$

$$\leq \mathbf{Adv}^{\mathrm{PR}}_{\mathsf{PRF},A_1}(\lambda) + \mathcal{N} \cdot \mathbf{Adv}^{\text{one-wayness}}_{\Pi,A_2}(\lambda)$$

where PRF is a pseudo-random function and TDP Π is a one-way permutation. ■

D. Analysis of Storage Complexity

Because the length of ciphertext resulted from our compilation framework is the same as that of original OPE/ORE ciphertext, our framework does not enlarge the storage complexity of the server. Because of the map **OT** which contains order tokens and counters, the storage complexity of client will increase $O(|\mathbf{S}|(\log|\mathcal{OT}| + \log|\mathbf{C}_s|))$, where $|\mathbf{S}|$ is the number of order spaces and $|\mathbf{C}_s|$ is the number of ciphertexts in every order space. We present two solutions to reduce the storage consumption as following:

- For every order space s, we can generate the first order token OT_0 by pseudo-random function $\mathsf{PRF}(s)$. The client only stores the counter *i* and calculates OT_i from OT_0 by TDP when the order query is going to be executed. Fortunately, it

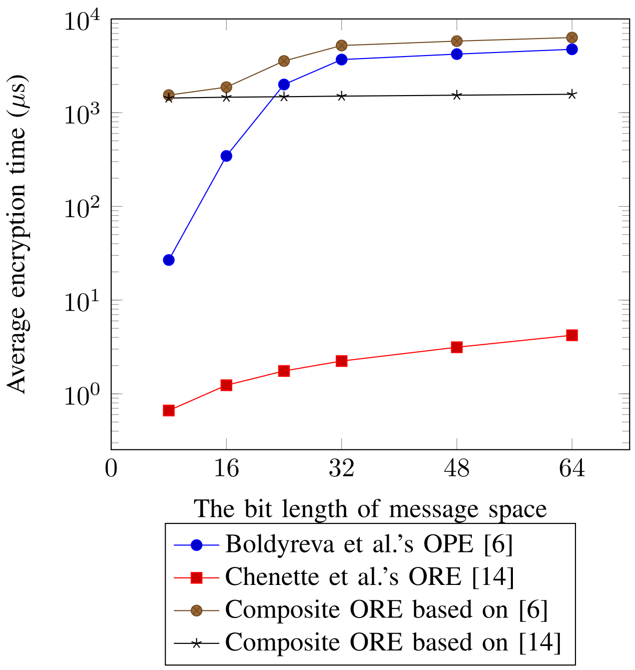


Fig. 5. The comparison of average encryption time between two existing OPE/ORE schemes and the composite forward secure ORE schemes with our compilation framework.

is not hard to perform TDP several times for popular TDPs like RSA. If (u, v, w) and (ψ, ϵ) are respectively the secret keys and the public keys, $\mathrm{OT}_i = \Pi_{sk}^{-i}(\mathrm{OT}_0)$ can be easily calculated as following:

$$f \leftarrow w^i \text{ mod } (u-1)(v-1), \quad \mathrm{OT}_i \leftarrow \mathrm{OT}_0^f \text{ mod } \psi.$$

Thus, the additional storage complexity of client will decrease to $O(|\mathbf{S}|\log|\mathrm{C}_s|)$ in this way.

- We can execute a count query every time before an order query in order to obtain the counter of an order space. Then the additional storage complexity of client becomes $O(1)$, no matter there are order spaces or not. And the order tokens can be calculated by TDP and PRF with the counter.

All the solutions above cause an extra pressure of calculation, but they are good options for the applications with limited storage at clients. In a way, **Algorithm 3** is the best construction for balancing the time complexity and the storage complexity.

E. Applicability and Experimental Analysis

Our compilation framework can be applied to all the OPE/ORE schemes except the OPE schemes, like the *ideally* secure schemes proposed in [24] and [29], which store the ciphertexts in an order tree. Their OPE schemes leak all the ciphertext order from the tree structure. Hence, the salting of our framework is useless in this condition.

For most of the OPE/ORE schemes in the literatures, our compilation needs to add a salt which is a hash value of an order token for every original ciphertext. Hence, the additional computational complexity of the resultant ORE with forward security depends on the server storage complexity of the original OPE/ORE. For instance, if a practical OPE (like the scheme in [6]) needs $O(N)$ storage on the server for a dataset of N items, the additional computational complexity caused by composite encryption and that by composite comparison are both $O(N)$. And for ORE schemes (like those in [12]), which encrypt every plaintext into T parts, the additional computational complexity caused by composite encryption and that by composite comparison are both $O(TN)$.

We finally designed some experiments to analyze the additional cost of our compilation framework. We combined the OPE/ORE schemes in [6] and [14] with our forward secure framework. The experiments are implemented in C/C++, and our experiments were performed using a single core on a machine with an Intel Pentium G2020 2.9GHz CPU and 4GB available RAM. We operate at 128-bits of security ($\lambda = 128$). We use HMAC as the PRF and the keyed hash function, and we use the RSA implementation (with 2048 bits RSA keys) in OpenSSL's BigNum library as the TDP. We use Blake2b as the underlying hash function. For our basic implementation of Boldyreva et al.'s OPE scheme, we use the C++ implementation from CryptDB [30], and for the implementation of Chenette et al.'s ORE scheme, we use the C-implemented FastORE mentioned in [26]. We use the datasets mentioned in Section III-G as the experimental plaintext sets.

Figure 5 shows the comparison results of the average encryption time between the original schemes and the composite schemes. We respectively used 100000 data for testing each of the points in Figure 5 and calculated the average results.

About the additional average encryption time, the composite forward secure ORE schemes demand about 1.5ms for each datum encryption. Moreover, as to the additional average comparison time, the composite forward secure ORE schemes demand about 47μs. Because we chose two of the most practical OPE/ORE schemes as the contrasts, the composite forward secure ORE schemes seem slower. However, the additional comparison time is still at the microsecond level. Hence, our scheme is still practical and useful for the most common systems which collect cipher data from numerous users instantaneity.

References

[1] "California public employee payroll data," 2014. [Online]. Available: http://transparentcalifornia.com/downloads/

[2] Google, "Encrypted Bigquery Client." [Online]. Available: https://github.com/google/encrypted-bigquery-client

[3] "Pmail." [Online]. Available: https://github.com/tonypr/Pmail

[4] "US social security name statistics." [Online]. Available: https://www.ssa.gov/OACT/babynames/

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, *SIGMOD '04*. ACM, 2004, pp. 563-574.

[6] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. "Order-preserving symmetric encryption," in A. Joux, editor, *RUROCRYPT 2009*, vol. 5479 of *LNCS*. Springer, Heidelberg, Apr. 2009, pp. 224-241.

[7] A. Boldyreva, N. Chenette, and A. O'Neill. "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in P.Rogaway, editor, CRYPTO 2011, vol. 6841 of LNCS. Springer, Heidelberg, Aug. 2011, pp. 578-595.

[8] 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 E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part ii*, vol. 9057 of LNCS. Springer, Heidelberg, Apr. 2015, pp. 563-594.

[9] R. Bost. "$\Sigma o\phi o\varsigma$ - forward secure searchable encryption," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, *ACM CCS 2016*. ACM Press, Oct. 2016, pp. 1143-1154.

[10] R. Bost, P. A. Fouque, and D. Pointcheval. "Verifiable dynamic symmetric searchable encryption: Optimality and forward security," Cryptology ePrint Archive, Report 2016/062, 2016. http://eprint.iacr.org/2016/062.

[11] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, *ACM CCS 2015*. ACM Press, Oct. 2015, pp. 668-679.

[12] D. Cash, F. H. Liu, A. ONeill, and C. Zhang. "Reducing the leakage in practical order-revealing encryption," Cryptology ePrint Archive, Report 2016/661, 2016. http://eprint.iacr.org/2016/661.

[13] Y. C. Chang and M. Mitzenmacher. "Privacy preserving keyword searches on remote encrypted data," in J. Ioannidis, A. Keromytis, and M. Yung, editors, *ACNS 05*, vol. 3531 of *LNCS*. Springer, Heidelberg, Jun. 2005, pp. 442-455.

[14] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. "Practical order-revealing encryption with limited leakage," in *International Conference on Fast Software Encryption*. Springer, Heidelberg, Mar. 2016, pp. 474-493.

[15] R. Cilibrasi and P. Vitanyi. "The google similarity distance," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 370-383, 2007.

[16] F. B. Durak, T. M. DuBuisson, and D. Cash. "What else is revealed by order-revealing encryption?" in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM Press, Oct. 2016, pp. 1155-1166.

[17] S. Garg, P. Mohassel, and C. Papamanthou. "TWORAM: Round-optimal oblivious RAM with applications to searchable encryption," ser. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563-592.

[18] O. Goldreich and R. Ostrovsky. "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43(3) of *JACM 1996*, pp. 431-473.

[19] P. Grofig, I. Hang, M. Härterich, F. Kerschbaum, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert. "Privacy by encrypted databases," in Annual Privacy Forum. Springer International Publishing, May 2014, pp. 56-69.

[20] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. "Leakage-abuse attacks against order-revealing encryption," In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 655-672.

[21] W. He, D. Akhawe, S Jain, E. Shi, and D. Song. "Shadowcrypt: Encrypted web applications for everyone," in *Proceeding of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM Press, 2014, pp. 1028-1039.

[22] M. S. Islam, M. Kuzu, and M. Kantarcioglu. "Access pattern disclosure on searchable encryption: Ramifcation, attack and mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[23] M. S. Islam, M. Kuzu, M. Kantarcioglu. "Inference attack against encrypted range queries on outsourced databases," in *Proceedings of the 4th ACM conference on Data and application security and privacy*, *CODASPY 2014*. ACM Press, Mar. 2014, pp. 235-246.

[24] F. Kerschbaum. "Frequency-hiding order-preserving encryption," in I. Ray, N. Li, and C. Kruegel editors, *ACM CCS 14*. ACM Press, Nov. 2014, pp. 275-286.

[25] B. Lau, S. Chung, C. Song, Y. Jang, W. Lee, and A. Boldyreva. "Mimesis aegis: A mimicry privacy shield-a systems approach to data privacy on public cloud," in *Proceeding of the 23rd USENIX conference on Security Symposium*. USENIX Association, 2014, pp. 33-48.

[26] K. Lewi and D. J. Wu. "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM Press, Oct. 2016, pp. 1167-1178.

[27] M. Naveed. "The fallacy of composition of oblivious RAM and searchable encryption," Cryptology ePrint Archive, Report 2015/668, 2015. http://eprint.iacr.org/2015/668.

[28] M. Naveed, S. Kamara, and C.V. Wright. "Inference attacks on property-preserving encrypted databases," in I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*. ACM Press, Oct. 2015, pp. 644-655.

[29] 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*. IEEE Computer Society Press, May 2013, pp. 463-477.

[30] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, Oct. 23-26, 2011*, pp. 85-100.

[31] D. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich. "POPE: Partial order-preserving encoding," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, *ACM CCS 2016*. ACM Press, Oct. 2016, pp. 1131-1142.

[32] E. Stefanov, C. Papamanthou, and E. Shi. "Practical dynamic searchable encryption with small leakage," in *21th Annual Network and Distributed System Security Symposium*, *NDSS 2014*. The Internet Society, Feb. 2014.

[33] Y. Zhang, J. Katz, and C. Papamanthou. "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *the Proceedings of the 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pp. 707-720.