Received May 3, 2018, accepted June 7, 2018, date of publication June 14, 2018, date of current version June 29, 2018.

*Digital Object Identifier 10.1109/ACCESS.2018.2847307*

# Efficient and Secure Top-k Queries With Top Order-Preserving Encryption

HANYU QUAN[1,2], BOYANG WANG[3], YUQING ZHANG[1,2], AND GAOFEI WU[1]
[1]School of Cyber Engineering, Xidian University, Xi'an 710071, China
[2]National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing 100049, China
[3]Department of Electrical Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221, USA

Corresponding author: Yuqing Zhang (zhangyq@ucas.ac.cn)

This work was supported in part by the National Key Research and Development Project of China under Grant 2016YFB0800703, in part by the National Natural Science Foundation of China under Grants 61272481, 61572460, and 61602361, in part by the Open Project of the State Key Laboratory of Information Security of China under Grant 2017-ZD-01, in part by the National Information Security Special Project of the National Development and Reform Commission of China under Grant (2012)1424, and in part by the China 111 Project under Grant B16037.

**ABSTRACT** Top-k queries can retrieve the most relevant tuples from massive datasets and have wide implementations, such as PageRank, healthcare analytics, and decision making. The increasing demands of outsourcing large datasets to public clouds with privacy concern expect new techniques to securely perform top-k queries on encrypted data on the cloud servers. Order-preserving encryption (OPE) can be used for answering top-k queries correctly and naturally. However, it is *over qualified* since it unnecessarily leaks too much information (i.e., orders of non-top-k values). In this paper, we propose a mutable top OPE (TOPE) to first enable top-1 (min or max) queries on encrypted data with minimized information leakage. Then, we extend this TOPE to support top-k queries in general. With TOPE, the ciphertexts of top-k values are still the top-k in the ciphertext domain, while the ciphertexts of non-top-k values are in meaningless order. In addition, we rigorously define and prove the security of TOPE with indistinguishability under top-ordered chosen-plaintext attacks. We implement our scheme on synthetic and real datasets to show its effectiveness and efficiency. The search performance of top-k queries on massive TOPE ciphertexts with our scheme is almost as fast as on the plaintexts.

**INDEX TERMS** Cloud computing, data privacy, order-preserving encryption, top-k query.

## I. INTRODUCTION

Top-k queries [1] can retrieve the most $k$ relevant tuples to users from massive datasets based on ranking. It has extensive applications, such as PageRank, healthcare analytics, and decision making, which make it a fundamental function for mining massive datasets in both SQL and NoSQL databases. For instance, a doctor can retrieve patients with the lowest blood sugar level in a medical dataset with `MIN()`, which is a top-1 query; a bank manager can retrieve the information of the most top-10 richest customers from thousands or millions of tuples in a financial dataset by querying `SELECT customer FROM table ORDER BY amount Limit 10`. Top-k queries also have been recently identified as one of the most critical techniques for studying and analyzing massive uncertain data [2], [3].

With the dramatic increase on the scale of datasets, a growing trend is for data owners to outsource their large-scale datasets to public cloud services in order to reduce local storage and query processing overhead. For instance, in healthcare field, with the growth in the use of Internet of Things (IoT) applications (e.g., wearable devices) and electronic heath record (EHRs), the volume of healthcare data grows exponentially each year [4]. Healthcare providers are starting to use clouds to manage and analyze the big data. For example, Philips, a Dutch company focuses on healthcare, is building its Philips HealthSuite digital platform on Amazon Web Services (AWS), which stores and analyzes 15 PB of patient data collected from studies, medical records and patient inputs [5]. On the other hand, due to legal and commercial issues, privacy of outsourced datasets on the cloud side is still a major concern. For example, an inside attacker who can see all cloud side data can easily reveal the sensitive data if an outsourced dataset is stored in plaintext (or encrypted by the cloud, which is referred to as *encryption at rest*) [6], [7]. While simply encrypting a dataset on the data owner side before outsourcing with traditional encryption (e.g., AES) can protect data privacy against an untrusted cloud server, but it inevitably losses rich search functionalities on the cloud side [8]. Specifically, top-k queries over outsourced data on the cloud side cannot be correctly performed if the dataset is encrypted by the data owner using traditional encryption.

The recent studies of Order-Preserving Encryption (OPE) [9]–[13] and Order-Revealing Encryption (ORE) [14]–[16], where the orders of ciphertexts are still consistently maintained as the orders of their plaintexts, can be obviously applied to ranking and answering top-k queries on encrypted data [17], [18]. However, OPE is *over-qualified* for top-k queries, which leaks too much information than secure top-k queries should be revealed. Specifically, to answer a top-k query over a number of $n$ tuples on encrypted data, where normally $k \ll n$, revealing only the orders of the top-k values is sufficient for answering the query while OPE unnecessarily leaks the orders of all the $n$ tuples in the dataset. This unnecessary leakage on the order of all the $n$ tuples in the dataset exposes too much information about datasets, especially considering the total number of tuples $n$ is normally large (e.g., millions) while $k$ is small (e.g., $k = 20$).

On the other hand, ranking-based searchable encryption [18], [19], also enables top-k queries in the ciphertext domain. However, compared to OPE, ciphertexts in searchable encryption cannot be ordered by themselves, which makes it incompatible with current SQL languages and softwares.

In this paper, we propose a new primitive, referred to as Top Order-Preserving Encryption (TOPE), which can naturally answer top-1 queries on encrypted data stored on an untrusted server. Then, we extend TOPE to support secure top-k queries in general with minimized leakage. Specifically, with our scheme, the ciphertexts of the top-k values from the plaintext domain are still the top-k in the ciphertext domain. In consequence, the encrypted data of plaintexts matching to top-k query can be easily and compatibly retrieved by existing SQL language (e.g., `SELECT * FROM table ORDER BY attribute LIMIT k`). More importantly, our scheme alleviates information leakage compared to OPE, where informally speaking, the orders of non-top-k values are not revealed in our scheme. A high-level comparison among TOPE, OPE and Deterministic Encryption (DE) in terms of privacy protection and search functionality is illustrated in Fig. 1, which implies TOPE is a better solution over OPE on top-k queries. Further details will be compared and discussed in Sec. VII and Sec. VIII. The main contributions of our work are summarized as follows:

- We first formally define and present a *mutable* Top Order-Preserving Encryption (TOPE), which can maintain top-1 order (i.e., min or max) on encrypted data while minimizing the leakage on orders of non-top-1 values. Specifically, we leverage *heaps* [20] as states in the encryption to maintain top-1 order, and this mutability indicates some of ciphertexts generated by TOPE may change over time in order to correctly answer top-1 queries. The main reason we leverage heaps is that heaps are *partially ordered* tree structures, which can naturally protect privacy of non-top-1 values. The security of TOPE is rigorously defined and proved with *indistinguishability under top-ordered chosen plaintext attacks* (IND-TOCPA).
- We then use this mutable TOPE as a stepping stone, and extend it to enable top-k queries on encrypted data with minimized privacy leakage. Our design is flexible in supporting dynamic dataset, and supports batch encryption in the setup phase, which can significantly reduce the running time of generating a massive ciphertexts. In addition, we discuss how to further preserve privacy by outputting probabilistic TOPE ciphertexts instead of deterministic ones.
- We implement our scheme with Java, and test its performance with both synthetic datasets and a real-world dataset. Encrypted data are stored in MySQL database and are searched with SQL queries. Experimental results show that, with a necessary tradeoff in terms of encryption time, our scheme is compatible with existing SQL language and extremely efficient, which is almost as fast as the search performance of top-k queries in plaintexts. More importantly, we leverage *Random Variables*, *Probability Mass Function* (PMF), and *Cumulative Distribution Function* (CDF) [21] to statistically analyze the advantages of TOPE over OPE in terms of privacy protection on the real-world dataset.

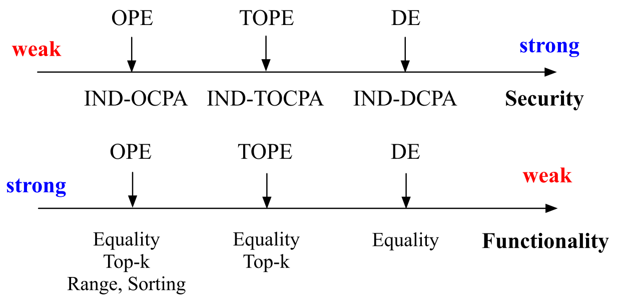


**FIGURE 1.** The comparison among OPE, TOPE, and DE in terms of security (i.e., indistinguishability) and search functionality on encrypted data.

## II. RELATED WORK

### A. ORDER-PRESERVING ENCRYPTION (OPE)

OPE was first formally defined and investigated by Boldyreva *et al.* [9], [10]. They rigorously proposed the *ideal security* of OPE, which is named as indistinguishability under ordered chosen-plaintext attacks (IND-OCPA). However, their designs failed to achieve the ideal security and at least half of the plaintext bits were leaked [10]. Moving a step forward, two mutable OPEs [11], [12] were proposed by leveraging binary search trees [20] to maintain proper states of ciphertexts on a server. These two mutable OPEs are both able to achieve the ideal security. As necessary trade-offs, the encryption algorithms in those two mutable

OPEs are *interactive* (between a client and a server) and some of the ciphertexts may need to be changed over time due to the change of the state of the encryption. Due to its high efficiency, OPE has been implemented in real secure applications such as CryptoDB [22].

The most important property of OPE is that, it maintains the orders of ciphertexts as the same as the orders of plaintexts. This information leakage is *perfect* (*i.e., revealing what we need but nothing more*) for sorting on encrypted data, but it is *over-qualified* for top-k queries. For instance, the server only needs to know which one is the ciphertext of the minimal value in a dataset to fulfill a min query but OPE unnecessarily tells the server the order of every pair of ciphertexts in the dataset.

Recently, an advanced variation of OPE, named Order-Revealing Encryption (ORE) was designed by Boneh *et al.* [14], and was optimized in [15] and [16]. Compared to OPE, which reveals order relations of two ciphertexts directly in the ciphertext domain, ORE leverages an additional evaluation algorithm (with a key) to reveal the order of two ciphertexts. ORE is more secure than OPE since it does not reveal the order relations directly. However, when applied to Top-k queries, it is still inevitably *over-qualified* as OPE. Besides, the calls of the additional evaluation algorithm make the efficiency of ORE less practical than the two mutable OPEs mentioned above.

### B. SEARCHABLE ENCRYPTION (SE)

Song *et al.* [8] first proposed SE to enable keyword search on encrypted data. Subsequent works of SE [6], [23]–[25] focus on improving search efficiency, allowing dynamic updates and supporting different types of queries. Some of these previous works can particularly perform comparisons and range queries, which achieve the same functionality on encrypted data as OPE. Specifically, Boneh and Waters [26] designed a public-key scheme to support comparisons and range queries. Shi *et al.* [27] also proposed a public-key approach, which focuses on multi-dimensional comparisons and range queries. Others utilized different tree structures to improve search efficiency [7], [28]. Recent works [24], [25], which support arbitrary Boolean queries, can also handle comparisons efficiently. Searchable schemes as [18], which can retrieve keywords or similar keywords based on ranking, can also be used for top-k queries on encrypted data.

Compared to OPE, SE provide a stronger security and privacy guarantee [11]. However, due to the different design methodologies between OPE and SE, the ciphertexts in searchable encryption cannot be ordered or ranked by themselves alone, which make SE much less compatible with current SQL languages and softwares [11].

## III. PROBLEM STATEMENT

### A. SYSTEM MODEL

In our system model (as described in Fig. 2), we have two entities, including a client (e.g., a company) and a server (e.g., a cloud service provider). The client outsources its massive dataset to the server in order to reduce local storage and query processing overhead. Moreover, the client also wants to query its outsourced data, and expects the server to return correct results. Particularly, in this paper, the client submits top-k queries on its data stored in the server. Normally, top-k queries are applied to numeric values (or messages that can be represented with numeric formats).

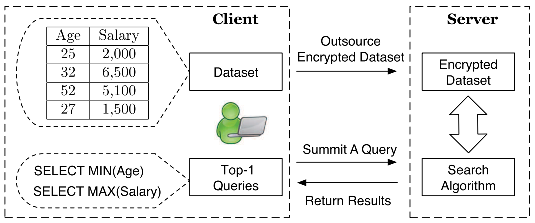


**FIGURE 2.** **The system model includes a client and a server.**

Due to privacy concern of the client, the server in this model is assumed to be *honest-but-curious* (also referred to as *semi-honest*), which indicates the server can provide reliable storage and query services but it is curious about the content of data stored in the cloud. As a result, the dataset of the client should be encrypted before being outsourced to this honest-but-curious server. Our major design objective is to enabling top-k queries on encrypted data but minimizing privacy leakage to the server. Without loss of generality, we will first focus on enabling top-1 queries (e.g., finding the minimal value or the maximal value) on encrypted data.

### B. DEFINITIONS OF MUTABLE TOPE

As stated above, we will first focus on defining our Top Order-Preserving Encryption (TOPE), which can search top-1 queries. Similar to some recent OPEs [11], [12], our encryption is also *mutable*, which means the ciphertexts (sometimes also referred to as encodings [11]) of some previous messages may be mutated while new ciphertexts are computed. In addition, the encryption algorithm is an *interactive* process between a client and a server, where the client has the secret key and the server maintains a state containing all the updated ciphertexts. For the ease of description, we use a function $\mathsf{top1}(\cdot)$ to represent the general form of a top-1 query (i.e., $\mathsf{MIN}(\cdot)$ or $\mathsf{MAX}(\cdot)$) in the rest of this paper. The formal definition of our mutable TOPE is described as below.

*Definition 1 (Mutable TOPE): A symmetric-key mutable Top Order-Preserving Encryption (TOPE) on plaintext domain $\mathcal{M}$ is a tuple of five polynomial-time algorithms $\Pi = (\mathsf{GenKey}, \mathsf{InitState}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Query})$ run by a client and a stateful server, where $\mathsf{Enc}$ is interactive:*

- $sk \leftarrow \mathsf{GenKey}(1^{\lambda})$*: is a probabilistic key generation algorithm that is run by the client to setup the scheme. It takes a security parameter $\lambda$ as input, and outputs a secret key $sk$.*

- $\mathtt{st}_0 \leftarrow \mathsf{InitState}(1^\lambda)$*: is a deterministic algorithm that is run by the server to initialize a state. It takes a security parameter* $\lambda$ *as input, and outputs an initial state* $\mathtt{st}_0$.
- $(c, \mathtt{st}') \leftarrow \mathsf{Enc}(sk, m, \mathtt{st})$*: is a deterministic (possibly probabilistic) algorithm that is interactively run between the client and the server. It takes a secret key sk, a message m, a state* $\mathtt{st}$ *as input, and outputs a ciphertext c and a new (updated) state* $\mathtt{st}'$.
- $m \leftarrow \mathsf{Dec}(sk, c)$*: is a deterministic algorithm that is run by the client. It takes a secret key sk, a ciphertext c as inputs, and outputs a message m.*
- $e \leftarrow \mathsf{Query}(\mathtt{st})$*: is a deterministic algorithm is run by the server to output the ciphertext of the top-1 (e.g., min or max) value. It takes a state* $\mathtt{st}$ *as inputs, and outputs a ciphertext e.*

The encryption and decryption algorithms can also be expressed as $\mathsf{Enc}_{sk}(m, \mathtt{st})$ and $\mathsf{Dec}_{sk}(c)$, respectively. The preceding scheme $\Pi$ is deterministic by default. And it is (possibly) probabilistic if $\mathsf{Enc}$ is probabilistic. We will show an approach to make it probabilistic in Sec. VI.

With the above scheme definition, we now explain its correctness. Assume the initial state of the scheme is $\mathtt{st}_0$, where $\mathtt{st}_0 \leftarrow \mathsf{InitState}(1^\lambda)$. Given a sequence of *distinct* messages $\mathtt{seq} = \{m_1, \ldots, m_n\}$, the scheme outputs *n successive* states $\{\mathtt{st}_1, \ldots, \mathtt{st}_n\}$ by computing $(c_i, \mathtt{st}_i) \leftarrow \mathsf{Enc}_{sk}(m_i, \mathtt{st}_{i-1})$, where $1 \leq i \leq n$. The correctness of the above mutable TOPE can be rigorously defined as below.

*Definition 2 (Correctness of Mutable TOPE): A mutable TOPE* $\Pi$ *on plaintext domain* $\mathcal{M}$ *is correct if, for every security parameter* $\lambda$*, for every sk* $\leftarrow \mathsf{GenKey}(1^\lambda)$*,*

1) *for every* $m \in \mathcal{M}$ *and for every state* $\mathtt{st}$*, for every c output by* $\mathsf{Enc}_{sk}(m, \mathtt{st})$*, we have* $\mathsf{Dec}_{sk}(c) = m$*;* **AND**
2) *for each sequence* $\mathtt{seq}_i = \{m_1, \ldots, m_i\} \in \mathcal{M}^i$*, where* $i \in [1, n]$*, there exists an* $m^* \in \mathtt{seq}_i$ *such that*

$$c^* = \mathsf{Query}(\mathtt{st}_i) \Longleftrightarrow \begin{cases} m^* = \mathsf{top1}(m_1, \ldots, m_i), \\ m^* = \mathsf{Dec}(sk, c^*). \end{cases}$$

Informally speaking, the preceding correctness of a mutable TOPE indicates 1) the decryption of a ciphertext should always be its original message being encrypted; 2) if a message is a top-1 (e.g., min or max) value among a set of plaintexts, then according to the state, its ciphertext should be the output of $\mathsf{Query}$ among the set of ciphertexts of these plaintexts.

### *C. IND-TOCPA SECURITY*

It is well-understood that a deterministic encryption is distinguishable (or insecure) under *standard* chosen-plaintext attacks (IND-CPA) [29]. Clearly, it is impossible for our deterministic design to achieve IND-CPA secure. However, our objective is to properly *weaken* standard IND-CPA as previous works did [9], [30], so that we are still able to define the (*best possible*) security for a TOPE scheme in a rigorous and reasonable manner. For instance, by restricting messages to be distinct in the security game, the security of a (classic) Deterministic Encryption can be relaxed to indistinguishability under distinct chosen-plaintext attacks (IND-DCPA) [30].

Correspondingly, the security of a mutable TOPE can be defined under a weak version of standard IND-CPA, which we denote as *indistinguishability under top-ordered chosen-plaintext attacks* (IND-TOCPA). We can think of it as a variation of *indistinguishability under ordered chosen-plaintext attacks* (IND-OCPA) being used for the ideal security of OPEs [9]–[12]. Informally, IND-TOCPA means given two messages $m_0$ and $m_1$, if they are respectively the top-1 value of two same-length sequences $\mathtt{seq}_0$ and $\mathtt{seq}_1$, then the ciphertexts of these two messages are computationally indistinguishable. In addition, this indistinguishability on ciphertexts should hold if they are respectively non-top-1 value of these two sequences. More precisely, given two same-length sequences of plaintexts $\mathtt{seq}_0 = \{m_{0,1}, \ldots, m_{0,n}\}$ and $\mathtt{seq}_1 = \{m_{1,1}, \ldots, m_{1,n}\}$, an adversary cannot distinguish the two ciphertext sequences $c_{0,1}, \ldots, c_{0,n}$ and $c_{1,1}, \ldots, c_{1,n}$ if for every integer $i \in [1, n]$, the two sub-sequences $m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ have the *same top-1 order*, which is expressed as

$$\begin{aligned} \big(m_{0,j} = \mathsf{top1}(m_{0,1}, \ldots, m_{0,i}) \wedge m_{1,j} \\ = \mathsf{top1}(m_{1,1}, \ldots, m_{1,i})\big) \\ \big(m_{0,j} \neq \mathsf{top1}(m_{0,1}, \ldots, m_{0,i}) \wedge m_{1,j} \\ \vee \neq \mathsf{top1}(m_{1,1}, \ldots, m_{1,i})\big) \end{aligned}$$

where $1 \leq j \leq i$, $\wedge$ denotes **AND**, and $\vee$ denotes **OR**.

The rigorous description of IND-TOCPA is similar to the one for IND-OCPA used in OPE. Specifically, an adversary is assumed to have (conditional) access to the encryption oracle and it can obtain all the views of states on the server. However, compared to IND-OCPA, which requires the distinct messages satisfying the *same orders* (e.g., $(m_{0,i} > m_{0,j}) \wedge (m_{1,i} > m_{1,j})$, for all $i, j \in [1, n]$) [9], IND-TOCPA only asks the distinct messages maintaining the same top-1 order, which makes it less restricted than IND-OCPA. In other words, sequences that satisfy same orders *must* also satisfy same top-1 order (or same top-k order in general), but sequences that satisfy same top-1 order *do not necessarily* satisfy same orders (for example, $\{1, 2, 3\}$ and $\{11, 13, 12\}$ only have the same top-1 (min) order), while less restrictions in the security game indicate higher security.

*Definition 3 (IND-TOCPA Security): Let* $\Pi = (\mathsf{GenKey}, \mathsf{InitState}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Query})$ *be a symmetric-key mutable TOPE over security parameter* $\lambda$*, the security game of it between a challenger* $\mathcal{CH}$ *and an adversary* $\mathcal{A}$ *is described as follows:*

1) *The challenger* $\mathcal{CH}$ *generates sk* $\leftarrow \mathsf{GenKey}(1^\lambda)$*, and chooses a random bit* $b \in \{0, 1\}$.
2) *The adversary* $\mathcal{A}$ *generates an initial state* $\mathtt{st}_0$*, where* $\mathtt{st}_0 \leftarrow \mathsf{InitState}(1^\lambda)$*, and has access to an encryption oracle* $\mathsf{Enc}_{sk}(\cdot)$ *by querying a number of q message pairs, where the i-th message pair is* $(m_{0,i}, m_{1,i})$*, for* $1 \leq i \leq q$.

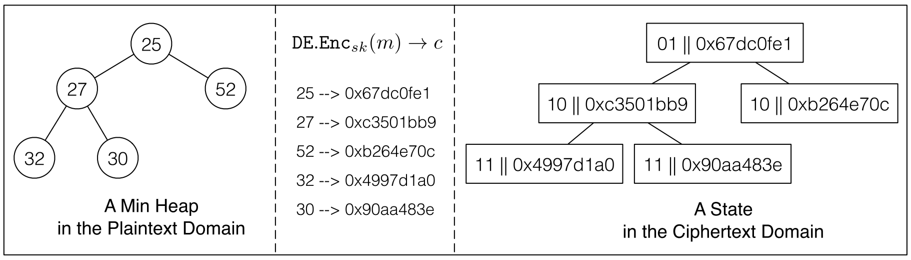


**FIGURE 3.** **A min heap in the plaintext domain, and its corresponding state in the ciphertext domain.**

3) *The challenger $\mathcal{CH}$ outputs a ciphertext $c_{b,i}$ and a state $\texttt{st}_{b,i}$, where $(c_{b,i}, \texttt{st}_{b,i}) \leftarrow \mathsf{Enc}_{sk}(m_{b,i}, \texttt{st}_{b,i-1})$, for message $m_{b,i}$ if*
   a) *$m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ are all distinct;* **AND**
   b) *$m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ have the same top-1 order*

   *and returns $(c_{b,i}, \texttt{st}_{b,i})$ to the adversary $\mathcal{A}$; otherwise, the challenger $\mathcal{CH}$ outputs $\perp$ and returns it to the adversary $\mathcal{A}$.*
4) *The adversary $\mathcal{A}$ outputs a ciphertext $e_{b,i}$ of the top-1 value by evaluating $\mathsf{Query}(\texttt{st}_{b,i})$, where $\texttt{st}_{b,i}$ is one of the states obtained from Step 3.*
5) *The adversary $\mathcal{A}$ outputs $b'$, which is its guess of $b$.*

*The adversary $\mathcal{A}$ wins this security game if $b' = b$. We say that scheme $\Pi$ is (ideal) IND-TOCPA secure if for any polynomial time adversaries in the above security game, it has at most negligible advantage to win the game:*

$$\mathbf{Adv}^{\mathbf{IND-TOCPA}}_{\Pi,\mathcal{A}}(1^\lambda) = \left|\Pr[b' = b] - \frac{1}{2}\right| \leq \texttt{negl}(\lambda) \quad (1)$$

*where $\texttt{negl}(\lambda)$ is a negligible function [29] in $\lambda$.*

Note that the above definition is the ideal security (i.e., best possible security) for a mutable TOPE. Namely, except the same top-1 order, no additional information is leaked. We will see later that by utilizing different types of heaps as states, a mutable TOPE sometimes may not be able to achieve the ideal security (*i.e., may need to further slightly weaken the ideal security by adding additional restrictions to the encryption oracle $\mathsf{Enc}_{sk}(\cdot)$ in Step 3*), but it is much more efficient for real applications (see discussions in Sec. V). Note that the security of these relaxed versions will still be stronger than the security of OPE. The definition and security of a mutable TOPE on top-k queries can be similarly presented by extending (i.e., relaxing) the corresponding restrictions above.

## IV. PRELIMINARIES

### A. HEAP

A heap is a *partially ordered* tree structure, where the root of the tree stores the top-1 (i.e., min or max) value [20]. An example of a heap is illustrated in Fig. 3. It has been extensively used for finding top-1 values among data. Based on the value stored at the root, a heap can be categorized as a *min heap* or a *max heap*. Nodes on the same path in a heap still maintain certain orders. However, siblings (i.e., nodes on the different paths) in a heap do not have particular orders. The most common type of heaps is *binary heaps*, which is a nearly complete binary tree. That is, the tree is completely filled on all levels except possibly the lowest, which is filled from left to right (the heap in Fig. 3 is a binary heap). A more general form is $d$-ary heap, in which the nodes have $d$ children instead of 2. We intend to utilize this partially-ordered property of heaps to minimize the information leakage on data compared to the use of binary search trees (i.e., fully-ordered) in OPE [11], [12], but still correctly and efficiently answer top-1 queries (and top-k queries in general) on encrypted data.

### B. DETERMINISTIC ENCRYPTION

Deterministic Encryption (DE) always generates a same ciphertext for a given message (assuming using the same key), which achieves weaker security than probabilistic encryption with IND-CPA security. However, by using this inherit property of DE, it is efficient to evaluate equality checking on encrypted data. A DE normally contains three algorithms $\{\mathsf{DE.GenKey}, \mathsf{DE.Enc}, \mathsf{DE.Dec}\}$, where $\mathsf{DE.GenKey}$ is probabilistic while the other two algorithms are deterministic. Specifically, we have

- $sk \leftarrow \mathsf{DE.GenKey}(1^\lambda)$: Given a security parameter $\lambda$, output a secret key $sk$.
- $c \leftarrow \mathsf{DE.Enc}(sk, m)$: Given $sk$ and a message $m \in \mathcal{M}$, output a ciphertext $c$.
- $m \leftarrow \mathsf{DE.Dec}(sk, c)$: Given $sk$ and a ciphertext $c \in \mathcal{C}$, output a message $m$.

Its correctness maintains as $\mathsf{DE.Dec}_{sk}(\mathsf{DE.Enc}_{sk}(m)) = m$, for all $sk$, and all $m \in \mathcal{M}$. As we mentioned in the last section, the security of a DE can be defined under IND-DCPA [30]. AES-ECB is a concrete example of DE.

## V. MUTABLE TOP ORDER-PRESERVING ENCRYPTION

### A. MAIN IDEA

In this section, we introduce the design of our mutable Top Order-Preserving Encryption. The main idea is to leverage a heap stored on the server to maintain the top-1 order of a dataset, and this heap represents a state $\texttt{st}$ described in the definition of mutable TOPE. Specifically, each node in the heap stored on the server contains only encrypted data (see Fig. 3). Similar as recent mutable OPEs [11], [12],

- $\mathsf{GenKey}(1^\lambda)$: Given security parameter $\lambda$, the client computes

$$sk \leftarrow \mathsf{DE.GenKey}(1^\lambda)$$

  and outputs a secret key $sk$.
- $\mathsf{InitState}(1^\lambda)$: Given security parameter $\lambda$, the server creates an empty min (or max) heap as the initial state $\mathtt{st}_0$, and $\mathtt{length}(\mathtt{st}_0) = 0$.
- $\mathsf{Enc}(sk, m, \mathtt{st})$: Given a secret key $sk$, a message $m$ and a state $\mathtt{st}$, the client first tells the server to increase the length of $\mathtt{st}$ by 1 by computing:

$$\mathsf{length}(\mathtt{st}) = \mathsf{length}(\mathtt{st}) + 1,$$

  and to set $\mathtt{st}[\mathtt{length}(\mathtt{st})] = \mathtt{null}$ and $i = \mathtt{length}(\mathtt{st})$. Then the client interacts with the server by running

$$i \leftarrow \mathsf{Traverse}(m, i)$$

  to locate the proper position of $m$ in the state. The client computes $c^p = \mathsf{DE.Enc}(sk, m)$ and sends it to the server. Finally, the server sets $\mathtt{st}[i].c^p = c^p$, runs $c^t = \mathsf{GenTranCipher}(\mathtt{st}[i], i)$, and outputs a ciphertext $c = (c^t, c^p) = \mathtt{st}[i]$ and the updated state as $\mathtt{st}'$.
- $\mathsf{Dec}(sk, c)$: Given a secret key $sk$ and a ciphertext $c$, where $c = (c^t, c^p)$, the client computes

$$m = \mathsf{DE.Dec}(sk, c^p)$$

  and outputs a message $m$.
- $\mathsf{Query}(\mathtt{st})$: Given a state $\mathtt{st}$, the server returns the ciphertext of the top-1 value as $e = \mathtt{st}[1]$.

**FIGURE 4. Details of mutable TOPE.**

a ciphertext $c$ at each node in our mutable TOPE includes two parts, a *transient ciphertext* $c^t$ and a *permanent ciphertext* $c^p$.

- A transient ciphertext (or transient encoding) $c^t$ is the binary encoding of the *tree level* of this corresponding node in the state (i.e., the heap), and it may change when new messages are encrypted due to the updates on its tree level and the structure change of the state;
- A permanent ciphertext $c^p$ is a ciphertext of a message stored at the corresponding node, where this ciphertext is computed by DE.Enc from a DE, and it does not change while encrypting new messages.

A transient ciphertext indicates the top-1 order (e.g., the root node of a state is the ciphertext of the top-1 value), while a permanent ciphertext protects the original message. Therefore, a combination of these two types of ciphertexts, representing as a mutable TOPE ciphertext, can indicate the top-1 order among encrypted data as the same as the top-1 order in the plaintext domain without revealing these original messages. For example, 25 is the root node of the min heap in Fig. 3, its permanent ciphertext is `0x67dc0fe1` (assume 128-bit security) and its transient ciphertext is `01` (the root node is at tree level 1, and its binary encoding is `01`), so the mutable TOPE ciphertext of 25 in Fig. 3 is

$$\mathtt{01||0x67dc0fe1}$$

where || is a concatenation operation. It is easy to see the top-1 order still correctly holds in the ciphertext domain with the help of a state. We reserve tree level 0 (e.g., transient ciphertext `00`) for the later implementation on top-k queries.

## B. SCHEME DETAILS

The detailed description of our mutable TOPE is explained as below and algorithms are also presented in Fig. 4. Note that, since a heap (i.e., a state) can be normally implemented as an array in practice [20], we use $n = \mathsf{length}(\mathtt{st})$ to denote the total number of elements in state $\mathtt{st}$, and we describe these elements as $\{\mathtt{st}[1], \mathtt{st}[2] \ldots, \mathtt{st}[n]\}$, where $\mathtt{st}[1]$ is the first element of the array and it stores the root node of a heap ($\mathtt{st}[0]$ is left to be $\mathtt{null}$). Since each element is a node storing a mutable TOPE ciphertext, we have $\mathtt{st}[i] = \{\mathtt{st}[i].c^t, \mathtt{st}[i].c^p\}$, where $\mathtt{st}[i].c^t$ and $\mathtt{st}[i].c^p$ is the transient ciphertext and the permanent ciphertext of $\mathtt{st}[i]$, respectively.

Without loss of generality, we assume a binary heap (i.e., $d = 2$) is used in the scheme description in Fig. 4. Since a binary heap is a nearly completed binary tree, given an index $i$ of $\mathtt{st}[i]$, its parent can be easily computed as $\mathtt{st}[\lfloor i/2 \rfloor]$, while its left child and right child can be calculated as $\mathtt{st}[i \cdot 2]$ and $\mathtt{st}[i \cdot 2 + 1]$, respectively. The tree level of an element $\mathtt{st}[i]$ can be obtained as $l = \lfloor \log_2(i) \rfloor + 1$.

To initialize the scheme, the client outputs a secret key in GenKey by running DE.GenKey, and the server creates an empty heap as its initial state by InitState. The client can decrypt a ciphertext by running Dec, and the server is able to query a state to return the ciphertext of the top-1 value (i.e., the root of the state) in Query. The encryption algorithm Enc is a relatively complicated process compared to other ones in the mutable TOPE, and several sub-algorithms leveraged in it are described in Fig. 5.

Essentially, generating a new ciphertext in Enc is similar to the process of inserting a new value into a heap in the plaintext domain [20]. The major difference in mutable TOPE is that we need to process it on encrypted data via interactions between the client and the server. Specifically, given a message $m$ in Enc, the client first tells the server to increase the length of the state by 1 (i.e., adding a new element at the end of the array), set the new element as `null`, and use the

- $\mathsf{Traverse}(m, i)$: Given a message $m$ and an index $i$,
  1) If $i = 1$, the client stops and returns $i$. Else, it retrieves $c_{parent} = \mathsf{Parent}(\mathtt{st}[i], i).c^p$ from the server.
  2) The client computes $m_{parent} = \mathsf{TOPE.Dec}(sk, c_{parent})$.
  3) The client calculates $\mathsf{top1}(m_{parent}, m)$,
     - If $m == \mathsf{top1}(m_{parent}, m)$, the client tells the server to swap $\mathtt{st}[i]$ with its parent $\mathsf{Parent}(\mathtt{st}[i])$, then the server runs $\mathsf{GenTranCipher}(\mathtt{st}[i], i)$ to update the transient ciphertext of $\mathtt{st}[i]$ (note that $\mathtt{st}[i]$ is currently the ciphertext of $m_{parent}$), and sets $i = \lfloor i/2 \rfloor$. The client and the server continue to interactively run $\mathsf{Traverse}(m, i)$.
     - Else (i.e., $m_{parent} == \mathsf{top1}(m_{parent}, m)$), the client stops and returns $i$.
- $\mathsf{Parent}(\mathtt{st}[i], i)$: Given a node $\mathtt{st}[i]$ and its index $i$, the server outputs its parent as $\mathtt{st}[\lfloor i/2 \rfloor]$.
- $\mathsf{GenTranCipher}(\mathtt{st}[i], i)$: Given a node $\mathtt{st}[i]$ and its index $i$, the server calculates $l$ (i.e., the level of this node in the state $\mathtt{st}$) as

$$l = \lfloor \log_2(i) \rfloor + 1,$$

computes the binary encoding of $l$, sets this binary encoding of $l$ as $\mathtt{st}[i].c^t$, and outputs $\mathtt{st}[i].c^t$.

**FIGURE 5. Details of some sub-algorithms used in mutable TOPE.**

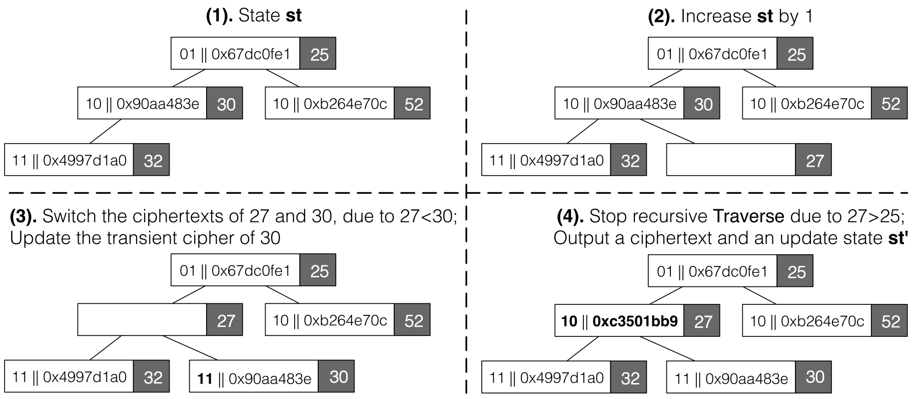


**FIGURE 6. An example of Enc in mutable TOPE, where top1(·) = MIN(·) and *m* = 27. Values in the gray parts (i.e., 25, 30, 52, 32, 27) are for the ease of illustration. They are not parts of a state stored on the server.**

index of this new element as a temporary position for storing the ciphertext of message $m$ (even though this ciphertext, especially the transient cipher of it, has not been computed yet at this moment).

Then, the client runs Traverse (see Fig. 5), in which it starts from the last element $\mathtt{st}[i]$ in the heap, retrieves the parent of node $\mathtt{st}[i]$, decrypts the parent, and compares the message $m_{parent}$ of this parent with the new message $m$, if $m = \mathsf{top1}(m, m_{parent})$ (i.e., $\mathtt{st}[i]$ and its parent currently do not maintain a proper order relation based on the property of a heap), the server swaps $\mathtt{st}[i]$ with its parent, updates the transient cipher of $\mathtt{st}[i]$ and index $i$ accordingly. The client recursively runs Traverse with the parent of the updated $i$ until there is no parent (i.e., reaching the root node) or if $m_{parent} = \mathsf{top1}(m, m_{parent})$ (i.e., $\mathtt{st}[i]$ and its parent maintain a correct relation based on the property of a heap). Finally, after the client and server interactively locate the proper index $i$ for the new message $m$, the client computes the permanent cipher $c^p$ with DE, the server computes the transient cipher $c^t$ based on the level of index $i$ and outputs an updated state $\mathtt{st}'$. An example of Enc, where $\mathsf{top1}(\cdot) = \mathsf{MIN}(\cdot)$ and $m = 27$ is presented in Fig. 6.

## *C. DISCUSSIONS*

As we presented, the encryption algorithm is interactive, because the client has the secret key while the server maintains an encrypted heap as the state. We will show in Sec. VI that leveraging a heap in our design is not only useful for running top-1 queries, but also necessary and *flexible* in supporting top-k queries in general.

Besides keeping the state on the server, an alternative design option would be letting the client maintain both the secret key and the state, which can avoid the implementation of an interactive encryption process. However, this alternative is less efficient, since the client has to spend huge storage cost on the state (e.g., the client maintains the entire heap locally in order to mutate ciphertexts properly, which costs at least as the same as the storage overhead on original messages). Moreover, if the client uses multiple devices to generate data, it needs to maintain multiple copies of the state (i.e., each device needs a copy of the state). On the other hand, allowing the server to keep both the secret key and the state will be obviously not secure, since the adversary can obtain all the views on the server (i.e., directly revealing the secret key to the adversary in this case).

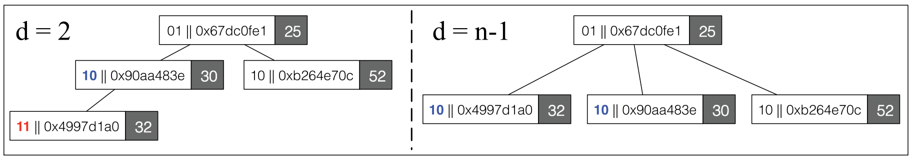


**FIGURE 7.** An example of the impact of $d$ on leakage. The binary heap reveals 30 < 32 while the ($n - 1$)-ary heap does not.

In the scheme description and security definitions of mutable TOPE, we assume no distinct messages are introduced as in OPE. In practice, in order to handle identical messages, the server can additionally build a hash table [20] to check whether the permanent ciphertext of a new message is already in the state, where this hash table is generated based on all the existing permanent ciphertexts in the state. An example of an item in the hash table is expressed as below

```
  HashValue     PermCipher    TranCipher
— — — — — — — — — — — — — —  — — — — — — —
   0x8db3a       0x90aa483e       10
```

where each hash value is calculated from its permanent ciphertext. Since permanent ciphertexts are deterministic (i.e., equality relations of messages still maintain over permanent ciphertexts), this hash table will work well as the same as the ones built from the plaintexts, where finding an entry in the hash table only requires $O(1)$ search time. If the permanent ciphertext of a new message is indeed in the hash table, the transient ciphertext of it can be directly obtained from the hash table without mutating the state. If the permanent ciphertext of a new message is not in the hash table, we need to insert it to the heap and mutate the state, where some of the transient ciphertexts in the state as well as the ones in the hash table should be updated correspondingly.

### D. IMPACTS OF PARAMETER d

In our mutable TOPE, the efficiency of the encryption algorithm is $O(\log_d(n))$ (in the worst case), which is the height of a state and is asymptotically as the same as the efficiency of an insertion in a heap [20]. Particularly, when generating a new mutable TOPE ciphertext, if $d = 2$, we have a state with $(\lfloor \log_2(n) \rfloor + 1)$ tree levels, where a number of $\lfloor \log_2(n) \rfloor$ retrieve-decrypt-compare operations (in Traverse) between the client and the server are needed in the worse case; and if $d = n - 1$, we only have a state with 2 levels, and only 1 retrieve-decrypt-compare operation is required.

Another impact of $d$ is that, the scheme is ideal secure iff $d = n - 1$ (see an example in Fig. 7). Specifically, for $d < n - 1$, some of the ciphertexts of non-top-1 values still maintain their meaningful order because they are on the same path in the state, which reveals additional information than what we have rigorously defined for the ideal IND-TOCPA security in Sec. III. Fortunately, we can further slightly limit the restrictions on the encryption oracle to maintain a *weak* IND-TOCPA security, where the restrictions in this case can be rigorously described as

1) $m_{0,1}, \ldots, m_{0,i}$, and $m_{1,1}, \ldots, m_{1,i}$ are all distinct; **AND**
2) The state of $m_{0,1}, \ldots, m_{0,i}$ and the state of $m_{1,1}, \ldots, m_{1,i}$ are isomorphic, i.e., $\mathtt{st}_{0,i} \simeq \mathtt{st}_{1,i}$.

The second restriction $\mathtt{st}_{0,i} \simeq \mathtt{st}_{1,i}$ indicates for every $1 \leq j \leq i$, the ciphertexts of $m_{0,j}$ and $m_{1,j}$ are located in the same position of $\mathtt{st}_{0,i}$ and $\mathtt{st}_{1,i}$, respectively. As a result, the transient ciphertexts of $m_{0,j}$ and $m_{1,j}$ will be the same throughout, which cannot be distinguished. Note that $\mathtt{st}_{0,i} \simeq \mathtt{st}_{1,i}$ implies $m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ have the same top-1 order, and this restriction is still looser than the *same orders* in OPE.

Although when $d < n - 1$, especially for $d = 2$, it is only weak IND-TOCPA secure, we will see in the next section that it is much more efficient for answering top-k queries.

## VI. EXTENSIONS OF MUTABLE TOPE

### A. TOP-k QUERIES

In the previous section, we use mutable TOPE to generate ciphertexts that can still maintain top-1 order. Leveraging it as a stepping stone, we now extend this mutable TOPE to answer top-k queries, which is the general form of top-1 queries.

At a high-level, instead of maintaining a single state, we can create multiple sub-states (e.g., $k$ heaps) while generating a ciphertext. Specifically, assuming using min heaps, the root node of the first sub-state is the ciphertext of the min value among $n$ values, and the root node of the second sub-state is the ciphertext of the min value among $(n - 1)$ values without considering the root node of the first sub-state. For instance, if $\{25, 27, 52, 32, 30\}$ are involved in the first sub-state, then only $\{27, 52, 32, 30\}$ will be considered in the second sub-state. Essentially, it is an approach to maintain a set of $k$ heaps to answer top-$k$ queries, where the sizes of each heap are $n$, $n - 1$, $\ldots$, $n - (k - 1)$, respectively. The roots of these $k$ heaps are simply the answers of top-$k$ queries. An example with two sub-states is described in Fig. 8, where the top-2 order is still maintained in the ciphertext domain while the orders of non-top-2 values are not revealed.

In practice, we can obtain the second sub-state by making a copy of the first sub-state and removing the root. More concretely, after making a copy of the first sub-state, the server swaps the position of the root and the last node in this copy, and reduces the size of this copy by 1. Then, the server moves this new root down if necessary by checking its order relation with its children. Essentially, it is the same as DELETE-MIN (or DELETE-MAX) operation in a min (or max) heap [20], except that each checking of the order relation between a

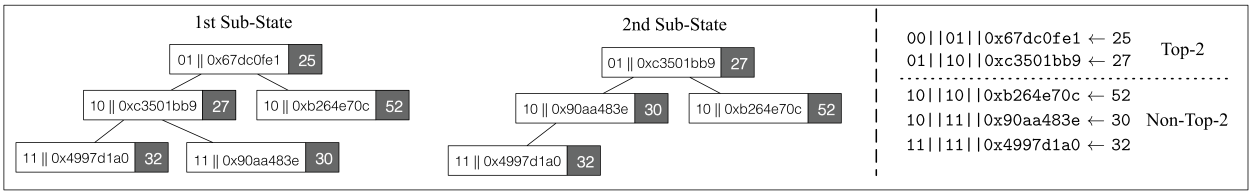


**FIGURE 8.** An example of TOPE with multiple sub-states, where a ciphertext is in the form of ($c^{t_2}||c^{t_1}, c^P$). $c^{t_2}$ is a transient ciphertext in the 2nd sub-state, $c^{t_1}$ is a transient ciphertext in the 1st sub-state, and $c^P$ is still a permanent ciphertext. If a plaintext (e.g., 25) is in the first sub-state but not in the second sub-state, its transient ciphertext in the 2nd sub-state is `00`.

parent and its children is done through interactions between the server and the client. Obviously, the third sub-state can be obtained after the second sub-state is built, and eventually the server can build $k$ sub-states for top-k queries by interacting with the client.

Due to the use of multiple sub-states, a ciphertext for answering top-k queries has multiple sub-transient ciphertexts (one for each sub-state), the length of a ciphertext will linearly increase with an increase of $k$. Moreover, it is well studied that the running time of DELETE-MIN (or DELETE-MAX) in a $d$-ary min (or max) heap is $O(d \log_d n)$ [20]. Thus, the generation time of a ciphertext in top-k queries is $O(kd \log_d n)$, which increases compared to $O(log_d n)$ in top-1 queries. Therefore, as we discussed before, although when $d = n - 1$, the mutable TOPE could achieve *ideal* IND-TOCPA security, it is much more efficient for top-k queries if $d$ is small (e.g., $d = 2$ in a binary heap). Another additional cost is that the server needs to spend more costs on keeping multiple heaps (i.e., $O(kn)$) compared to a single state (i.e., $O(n)$) in the original TOPE.

## B. DYNAMIC DATA

Our scheme also supports dynamic data, including insert, modify, or delete a plaintext from a state, which essentially allow a client to update its outsourced encrypted datasets in the cloud. Inserting a new value to the state with our scheme is obvious (i.e., generating a new ciphertext with our scheme). For modify and delete operations, they are not straightforward because a heap alone does not efficiently support Find (i.e., to modify or delete, we first need to find an existing element in a heap, which takes $O(n)$ time).

Fortunately, we can use the hash table we mentioned ahead to find an existing permanent ciphertext, and locate its corresponding index in the state with $O(1)$ time. Specifically, for each tuple in the hash table, we have an additional attribute to record each ciphertext's index (i.e., the position of it in the array of the heap), for instance,

| HashValue | PermCipher | TranCipher | Index |
|---|---|---|---|
| 0x8db3a | 0x90aa483e | 10 | 2 |

After locating the index, the server can replace the previous permanent ciphertext at this index with the new permanent ciphertext if it is a modify operation, and then move this new permanent ciphertext up/down if it does not maintain a proper order relation with its parent/children (again interactions between the client and the server are required to check this order relation). If it is a delete operation, then the server can swap the position of the located index with the last element in the array of the heap, reduce the state size by 1 and move the corresponding node down if it does not maintain a proper order relation with its children.

## C. BATCH ENCRYPTION

In some applications, where a huge number of plaintexts are introduced in a short period, it is necessary to efficiently encrypt these massive data in a relatively short time. Although our TOPE is based on lightweight DE, a large amount of interactions between the client and the server may still slow efficiency down while encrypting.

A common situation, where the idea of batch encryption can optimize the performance, is to use it in the setup phase. For instance, the client would like to encrypt and outsource its local large-scale dataset at once (instead of interactively encrypting each data one by one) in the setup phase. In this case, the batch encryption can be done by asking the client to build a heap in plaintext based on its local dataset, generate all the transient ciphertexts and all the permanent ciphertexts locally, and then outsource the encrypted heap at once. Then, the client can delete the copy of this initial encrypted heap from local storage, and it can generate more TOPE ciphertexts one-by-one subsequently via interactions between the client and the server (i.e., using the Enc in TOPE). It implies that the client only spends additional storage overhead of maintaining a heap in the setup phase. Once the encrypted heap is sent to the server, the client does not have to maintain any additional state (i.e. heap), where a new TOPE ciphertext will be generated instantly by following our scheme.

Note that in a binary heap, the running time of building an $n$-node bina ry heap is $O(n)$ [20]. Thus, the overall running time of batch encryption of $n$ messages is $O(n)$, and this batch encryption is a non-interactive operation.

## D. PROBABILISTIC TOPE

Up to the present, our scheme is described in a deterministic manner as previous OPE schemes [11], [12], where the same message always outputs the same TOPE permanent ciphertext. Efficient and scalable as it is in terms of encryption and updates, all types of deterministic ciphertexts leak certain

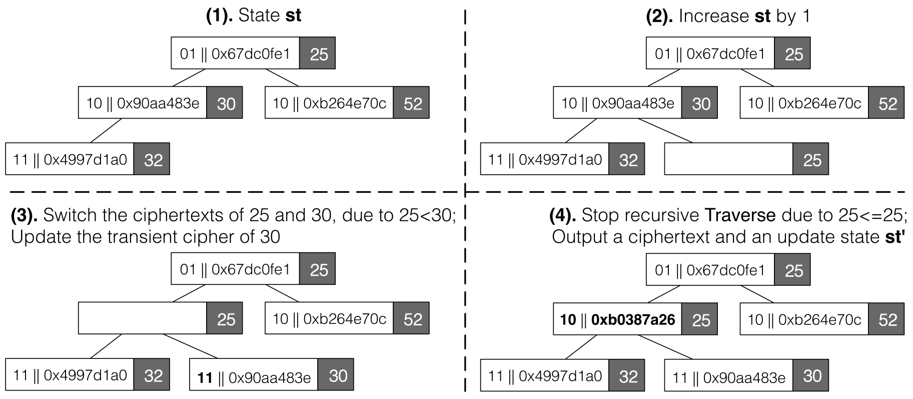

(1). State st
(2). Increase st by 1
(3). Switch the ciphertexts of 25 and 30, due to 25<30; Update the transient cipher of 30
(4). Stop recursive Traverse due to 25<=25; Output a ciphertext and an update state st'
01 || 0x67dc0fe1 25
10 || 0x90aa483e 30
10 || 0xb264e70c 52
11 || 0x4997d1a0 32
11 || 0x90aa483e 30
10 || 0xb0387a26 25

**FIGURE 9.** **An example of** Enc **with probabilistic TOPE, where** top1(·) = MIN(·) **and** $m = 25$**. Values in the gray parts (i.e., 25, 30, 52, 32) are for the ease of illustration. They are not parts of a state stored on the server.**

statistical information about datasets. Inspired by a recent work [13], we describe how to make those ciphertexts probabilistic while still using a heap as a state.

Specifically, given a new message $m$, the client will insert a probabilistic ciphertext of this message to the state no matter whether there is an identical message in the heap. Similar as in the deterministic version illustrated ahead, the server will still increase the state by 1, put this ciphertext of this new message in the last position of the state, and move it up level by level if necessary by interactively comparing this value with its parent.

However, when a child has an identical value as its parent, the client will randomly decide to swap these two positions. In other words, the order relation between a parent and a child in a state with deterministic ciphertexts is *strictly* followed as `parent < child` (assume using a min-heap). While in a state with probabilistic ciphertexts, the order relation between a parent and a child is *loosely* followed as `parent` $\leq$ `child`. By loose, we mean the swapping of the positions of a parent and child is optional and randomly decided when `parent == child`. An example of generating a probabilistic TOPE ciphertext is presented in Fig. 9, where encrypting 25 twice will output two probabilistic ciphertexts `01||0x67dc0fe1` and `10||0xb0387a26` respectively.

By making ciphertexts probabilistic, the client can easily preserve statistical information of an outsourced datasets against a curious server. However, on the other hand, the size of the state grows dramatically (since a state with probabilistic ciphertexts needs a number of $N$ nodes, where $N$ is the total number of data records, while a state with deterministic ciphertexts requires only a number of $n$ nodes, where $n$ is the total number of distinct values and $n \leq N$). This issue (i.e., the gap between $n$ and $N$) normally would be a huge cost in real datasets. For example, if the client encrypts people's age in a medical dataset, where the number of distinct values is $n = 120$ (i.e., `age` $\in [0, 120]$) in general, while the number of tuples in the medical dataset could be $N = 1$ million. Obviously, maintaining a state with a size of $n = 120$ is much more efficient than the one with a size of $N = 1 \times 10^6$. Another significant downside of having probabilistic ciphertexts is that modification and deletions on encrypted data are not provided, since equality checking over encrypted data is disabled with probabilistic ciphertexts.

## VII. SECURITY ANALYSIS

### A. SECURITY ANALYSIS OF MUTABLE TOPE

We now analyze the security of our mutable TOPE. The following proof is with the ideal IND-TOCPA security, while the weak IND-TOCPA security (i.e., $d < n - 1$) can be proved in the same way except using the tougher restriction we mentioned in Sec. V-D.

*Theorem 1: Our mutable TOPE scheme is IND-TOCPA secure as long as DE is IND-DCPA secure.*

*Proof:* At the high level, the security of our mutable TOPE can be interpreted as follows: first, we argue that transient ciphertexts only reveal the top-1 order; second, we can prove that, for two mutable TOPE ciphertexts generated from two distinct messages, when they have the same transient ciphertexts, if these two mutable TOPE ciphertexts are distinguishable, it is equivalent to say two DE ciphertexts (i.e., the two permanent ciphers of these two mutable TOPE ciphertexts) are distinguishable, which contradicts to the assumption that DE is IND-DCPA secure.

More specifically, if the DE is simply constructed based on a Pseudo Random Function (PRF), e.g., $\mathsf{DE.Enc}_{sk}(m) = \mathsf{PRF}_{sk}(m)$ [29], distinguishing those two mutable TOPE ciphertexts is equivalent to distinguishing a PRF and a random function, which is computationally infeasible [29]. Note that for two mutable TOPE ciphertexts with different transient ciphertexts, especially the ones on the same path, it is trivial to distinguish them. However, this does not contradict to the security of mutable TOPE. Because it is the natural information leakage of a mutable TOPE, which can be used for answering top-1 (e.g., min or max) queries. Details analyses are described below.

Assume we have an adversary $\mathcal{A}'$ from the IND-DCPA security game of the DE, where adversary $\mathcal{A}'$ has a negligible advantage in that game (i.e., $\mathbf{Adv}^{\textsf{IND-DCPA}}_{\textsf{DE},\mathcal{A}'}(1^\lambda) \leq \texttt{negl}(\lambda)$), we simulate the IND-OTCPA security game of our mutable TOPE with adversary $\mathcal{A}'$ (instead of adversary $\mathcal{A}$). Specifically, we have

1) The challenger $\mathcal{CH}$ generates $sk \leftarrow \textsf{DE.GenKey}(1^\lambda)$ and chooses a random bit $b \in \{0, 1\}$;
2) The adversary $\mathcal{A}'$ generates an initial state $st_0$ by creating an empty heap, and it is able to query a number of $q$ message pairs, where the $i$-th message pair is $(m_{0,i}, m_{1,i})$, for $1 \leq i \leq q$;
3) The challenger $\mathcal{CH}$ outputs a ciphertext $c_{b,i} = (c^t_{b,i}, c^p_{b,i})$ and a state $\texttt{st}_{b,i}$ for message $m_{b,i}$ by computing $(c_{b,i}, \texttt{st}_{b,i}) \leftarrow \textsf{Enc}_{sk}(m_{b,i}, \texttt{st}_{b,i-1})$, where $c^p_{b,i} \leftarrow \textsf{DE.Enc}_{sk}(m_{b,i})$, and return $(c_{b,i}, \texttt{st}_{b,i})$ to the adversary $\mathcal{A}'$ if
   a) $m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ are all distinct; **AND**
   b) $m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ have the same top-1 order.

   It is equivalent to say that the challenger $\mathcal{CH}$ returns $(c^t_{b,i}, c^p_{b,i}, \texttt{st}_{b,i})$ to the adversary $\mathcal{A}'$, where
   a) $m_{0,1}, \ldots, m_{0,i}$ and $m_{1,1}, \ldots, m_{1,i}$ are all distinct; **AND**
   b) $\texttt{st}_{0,i} \simeq \texttt{st}_{1,\texttt{i}}$ and $c^t_{0,i} = c^t_{1,i}$.

   Otherwise, the challenger $\mathcal{CH}$ outputs and returns $\perp$ to adversary $\mathcal{A}'$;
4) The adversary $\mathcal{A}'$ outputs a ciphertext $e_{b,i}$ of the top-1 value by choosing the root of state $\texttt{st}_{b,i}$, where $\texttt{st}_{b,i}$ is one of those states obtained from Step 3;
5) The adversary $\mathcal{A}'$ outputs $b'$ as its guess of $b$.

In the above, we successfully simulate the IND-TOCPA security game of a mutable TOPE with adversary $\mathcal{A}'$ from the IND-DCPA security game of a DE. Specifically, when given $c^t_{0,i} = c^t_{1,i}$ and $\texttt{st}_{0,i} \simeq \texttt{st}_{1,\texttt{i}}$, it does not provide any additional advantages for adversary $\mathcal{A}'$ to guess $b'$ based on $c^p_{b,i} \leftarrow \textsf{DE.Enc}_{sk}(m_{b,i})$, for each $i \in [1, q]$. Said differently, the IND-TOCPA security game is simulated by a number of $q$ sub-games, where each sub-game is an IND-DCPA security game. Therefore, adversary $\mathcal{A}'$ could win the above game with advantages as long as it could win any of the $q$ sub-games with advantages, then we have

$$\begin{aligned}\mathbf{Adv}^{\textsf{IND-TOCPA}}_{\textsf{TOPE},\mathcal{A}}(1^\lambda) &\leq q \cdot \mathbf{Adv}^{\textsf{IND-DCPA}}_{\textsf{DE},\mathcal{A}'}(1^\lambda) \quad \text{(Prop. 1)}\\ &\leq q \cdot \texttt{negl}(\lambda)\\ &\leq \texttt{negl}'(\lambda) \quad \text{(Prop. 2)}\end{aligned}$$

which proves the security of our mutable TOPE. The two propositions (i.e., Prop. 1 and Prop. 2) used in the above are shown after this proof.

For the security of TOPE on top-k queries in general, we can use a function $\textsf{topk}(\cdot)$ instead of $\textsf{top1}(\cdot)$ to capture the corresponding leakage in the security game, and follow the same approach above to prove its security, which also relies on the IND-DCPA security of DE. □

*Proposition 1 (Union Bound [29]):*

$$\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2]$$

*Proposition 2 (Properties of Negligible Functions [29]): Let* $\texttt{negl}_1$ *be negligible functions, for any positive polynomial p, the function* $\texttt{negl}_2(n)$ *defined by* $\texttt{negl}_2(n) = p(n) \cdot \texttt{negl}_1(n)$ *is negligible.*

Readers are referred to [29] for further details of these two above propositions.

## VIII. EXPERIMENTAL ANALYSIS

In this section, we first demonstrate the efficiency of our scheme, including encryption time and search time. Specifically, we implement our scheme based on a binary heap with Java 8, and utilize JDBC (Java Database Connectivity) to connect to MySQL (v5.5) to store and query datasets. Besides using different scales of random data, we also evaluate the performance of our scheme by leveraging a real-world diabetes dataset [31] from UCI Machine Learning Repository. More importantly, we also leverage statistic techniques, including Random Variables, Probability Mass Function and Cumulative Distribution Function, to analyze the actual advantages of TOPE over OPE in terms of privacy protection on this real-world diabetes dataset.

### A. PERFORMANCE EVALUATION ON SYNTHETIC DATA

We run our Java code on a Linux machine (Ubuntu 16.04, 3.2 GHz Intel Core i5, 4G memory). The permanent ciphertexts are implemented with AES-ECB model with 128-bit security. The size of transient ciphertexts is 64 bits. In order to flexibly evaluate the impact of networks with different delays (i.e., round-trip time, RTT), we use `Thread.sleep(RTT)` to simulate the interaction time between a client and a server for each round. The functions on the client side and the functions on the server side are all run on this single Linux machine in our experiments.

We first use some synthetic data to test the performance of our scheme with different scales of datasets. More specifically, we generate random *distinct* data values, read these raw data from files, encrypt them with our TOPE. These TOPE ciphertexts are then stored in MySQL via JDBC and ready for answering SQL queries. Besides encrypted heaps, another hash table as we mentioned before is also maintained in order to check non-distinct data values (i.e., avoid to generate two different transient ciphertexts for a same data value).

#### 1) IMPACT OF THE NUMBER OF DISTINCT MESSAGES

We first investigate the impact of the number of distinct messages on encryption time. As we can see from Fig. 10, where we assume RTT = 1 millisecond, the encryption time is almost linearly increasing with an increase of the number of distinct messages. Specifically, when the client has 1024 distinct messages, our scheme needs 14.59 seconds to generate all the 1024 TOPE ciphertexts. Since we insert each new distinct element one by one to the encrypted heap, where the overall complexity is $O(n)$ (which is as the same as the

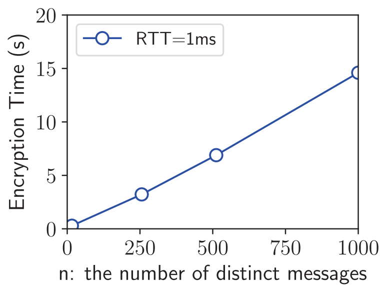


**FIGURE 10. The impact of $n$ on the encryption time (seconds) where RTT = 1ms and $k$ = 1.**

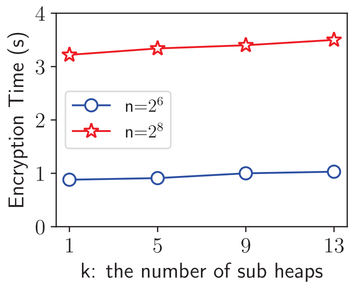


**FIGURE 11. The impact of $k$ on the encryption time (seconds) where RTT = 1ms and $k$ = 1.**

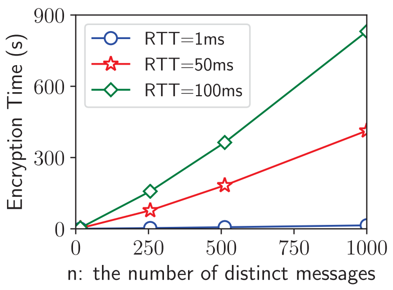


**FIGURE 12. The impact of $n$ on the encryption time (seconds) with different RTTs and $k$ = 1.**

**TABLE 1. The Impact of Batch Encryption on Encryption Time (seconds) where $n$ = 64.**

| | Batch | RTT=1 ms | RTT=50 ms | RTT=100 ms |
|---|---|---|---|---|
| $k = 1$ | 0.64 | 0.88 | 12.90 | 26.32 |
| $k = 5$ | 0.70 | 0.91 | 13.77 | 27.20 |
| $k = 9$ | 0.78 | 1.00 | 14.36 | 29.59 |
| $k = 13$ | 0.77 | 1.03 | 14.22 | 30.58 |

complexity of building a heap in plaintexts [20]), it is not surprising to see that the encryption time of an entire random dataset increases linearly. Besides, other operations, such as I/O (e.g., reading data from a file) and generating the hash table, which are linear as well, also together impact the entire encryption process in the implementation, which eventually leads to a linear increase overall.

### 2) IMPACT OF $k$

According to our design, our scheme should generate $k$ sub-heaps to answer top-$k$ queries in general. Specifically, our scheme first builds the 1st sub-heap, and outputs the rest of $k-1$ heaps one after another (i.e., the $i$-th sub-heap is generated from the $(i-1)$-th sub-heap). As illustrated in Fig. 11, when a client needs to support top-$k$ queries with a greater $k$, the encryption of our scheme slightly increases with $k$.

### 3) IMPACT OF RTT

Since our scheme is a mutable encryption, where one round of client-to-server interaction is needed while traversing from a parent node to a child node in an encrypted heap. Obviously, the network delay (i.e., RTT) plays a significant impact on the encryption performance of our scheme. As we can see from Fig. 12, when the network situation is almost ideal (i.e., RTT = 1 ms), our scheme is extremely efficient. While with the increase of RTT, the encryption time increases. For instance, if the network is relatively normal (e.g., with RTT = 50 ms), it takes around 413 seconds for our mutable encryption to output 1024 TOPE ciphertexts. On the other hand, if the delay in the network is high (e.g., with RTT = 100 ms), our scheme requires 831 seconds on average to generate a same number of TOPE ciphertexts. From the differences among different RTTs in Fig. 12, it is also easy to see that the overall encryption time linearly increases with an increase of RTT.

### 4) IMPACT OF BATCH ENCRYPTION

As we mentioned before, we can leverage batch encryption to avoid interactions and save a significant amount of encryption time in the setup phase, especially when network delay is high. More specifically, the client still uses binary heaps to produce TOPE ciphertexts, but instead of generating them one by one as in previous experiments, the client first generates heaps in plaintexts, then encrypt them, and produce transient ciphertexts and permanent ciphertexts before outsourcing them to the server. The encrypted heaps after performing batch encryption can still be used for dynamic operations (e.g., removing a data, modifying a data or inserting a new data) later. By doing this, the overall encryption time is independent with RTT. As presented in Table 1, the time saving is significant, especially when the network delay is in a bad condition. Specifically, it is over 40 times faster if batch encryption is leveraged while RTT = 100 milliseconds. Even if RTT is extremely good (e.g., RTT = 1ms), this type of batch encryption can still save over 20% of the overall encryption time.

## *B. PERFORMANCE EVALUATION ON REAL-WORLD DATA*

Besides distinct random data, we also test the performance of our scheme over a real-world dataset. Specifically, we leverage a diabetes dataset, and test the encryption time and search time of it while running our scheme. A brief description about this dataset is presented in the following.

### 1) THE INFORMATION OF THE DIABETES DATASET

Each tuple in this original dataset has four attributes, `Date`, `Time`, `Code` and `Value`. `Date` and `Time` are in the format

**TABLE 2. The Impact of $k$ on Encryption Time (seconds) over the Diabetes Dataset.**

| $k=1$ | $k=5$ | $k=9$ | $k=13$ |
|---|---|---|---|
| 6.31 | 6.33 | 6.41 | 6.54 |

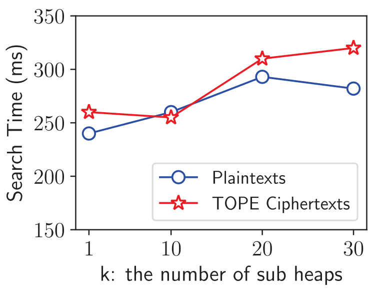


**FIGURE 13. The comparison of search time.**

of $\mathtt{MM-DD-YYYY}$ and $\mathtt{XX:YY}$, respectively. Attribute `Value` shows the blood glucose (BG) of a patient at a certain time, and attribute `Code` is the categorized code based on `Value`. An example of a tuple is presented as below

| Date | Time | Code | Value |
|---|---|---|---|
| 04-21-1991 | 9:09 | 58 | 100 |

We leverage TOPE to encrypt all the BG values in the dataset. There are a number of 441 distinct BG values and the range is from $[0, 501]$. The total number of (valid) tuples in the dataset is $29, 263$. We ignore some invalid tuples with incorrect or inconsistent data format.

### 2) PERFORMANCE ON THE DIABETES DATASET

Since we generate ciphertexts for the entire dataset as a whole, we leverage batch encryption here to boost the encryption performance. When $k = 1$, it takes 6.31 seconds on average to build an encrypted heap for those 441 distinct BG values and output ciphertexts for all the $29, 263$ tuples. The overall encryption time slightly increase when $k$ increases as shown in Table 2. It is easy to observe that our scheme is extremely efficient over this large-scale diabetes dataset in terms of encryption time. Obviously, this mainly depends the number of distinct values in a dataset.

In Fig. 13, we show that the search time of our scheme over diabetes dataset is almost good as the corresponding search time in plaintexts. The search efficiency of our scheme is actually not suprising, because MySQL still runs the same way to answer top-k queries by evaluating comparison operations. The only difference here is that the domain of transient ciphertexts is different from (more specifically, much greater than) the domain of plaintexts, which seems to be the reason that makes the search time over TOPE ciphertexts slightly slower than the one in plaintexts.

## *C. STATISTICAL ANALYSIS ON REAL-WORLD DATA*

Although we use formal definitions and rigorous proofs to properly capture the privacy leakage of our scheme, which is normally sufficient for proving the security of a cryptographic primitive, the actual impact of it on real-world datasets behind those formal security games remains obscure. Put differently, it is still not straightforward to see how well we can preserve the privacy of real-world datasets with TOPE in practice compared to the implementation of OPE. In order to fulfill this gap, we leverage Random Variables, Probability Mass Function and Cumulative Distribution Function [21] to observe the statistical information over real-world data's ciphertexts generated by Deterministic Encryption (i.e., AES-ECB [29]), Order-Preserving Encryption [11] and our Top Order-Preserving Encryption.

### 1) RANDOM VARIABLES

Given a set of distinct messages $\mathbf{m} = (m_1, \ldots, m_n)$, we define a random variable $X$ as

$$X(m_i) = \mathsf{rank}(m_i)$$

where $X(m_i) \in [1, n]$ and $\mathsf{rank}(m_i)$ indicates the order of $m_i$ in $\mathbf{m}$ after $\mathbf{m}$ is sorted. For example, given $\mathbf{m} = \{25, 52, 27, 32, 30\}$, we can first sort it as $\mathbf{m} = \{25, 27, 30, 32, 52\}$, then we have $X(25) = 1$ and $X(27) = 2$ since 25 and 27 are the smallest one and the second smallest one in $\mathbf{m}$ respectively.

Similarly, given a set of distinct ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$, we can also define a random variable $Y$ as

$$Y(c_i) = \mathsf{rank}(c_i)$$

where $Y(c_i) \in [1, n]$ and $\mathsf{rank}(c_i)$ indicates the order of $c_i$ in $\mathbf{c}$ after $\mathbf{c}$ is sorted.

Then, given a set of (distinct) message/ciphertexts pairs $\mathbf{z} = (z_1, \ldots, z_n)$, where $z_i = (m_i, c_i)$ and $c_i = \mathsf{Enc}(m_i)$, for $i \in [1, n]$, we can define a random vector $Z$ as

$$Z(z_i) = Z(m_i, c_i) = (X(m_i), Y(c_i))$$

where $Z(z_i) \in \mathbb{Z}_n^2$. Intuitively speaking, this random vector $Z$ is another way to show whether the orders of ciphertexts are consistent with the orders of their messages.

More specifically, the distribution of this random vector $Z$ generated by the real dataset using DE, OPE and TOPE are presented from Fig. 14 to Fig. 16. Here, for the ease of showing the major difference within the top-k part in these figures, we take $n = 20$ distinct message/ciphertexts pairs generated by the diabetes dataset (i.e., BG values from 0 to 19 and their corresponding ciphertexts). We can see from Fig. 14 and Fig. 15 that the distribution of $Z$ with DE is random (i.e., the orders of ciphertexts are not consistent with the orders of their messages) while the distribution of $Z$ with OPE is certain and predictable (i.e., the orders of ciphertexts are consistent with the orders of their messages). The reason is that DE does not reveal any meaningful order information while OPE intentionally reveals the orders of messages over ciphertexts. We can also observe from Fig. 16 that the distribution of $Z$ with our TOPE is only certain and predictable when $(0, 0) \leq Z(z_i) \leq (k, k)$ ($k = 5$ in this example), but

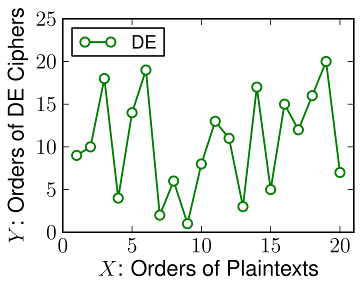


**FIGURE 14. The distribution of *Z* with DE.**

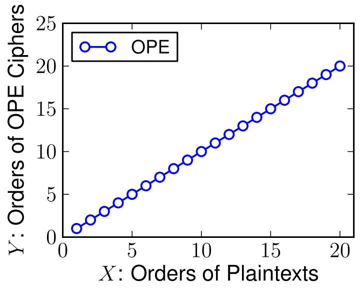


**FIGURE 15. The distribution of *Z* with OPE.**

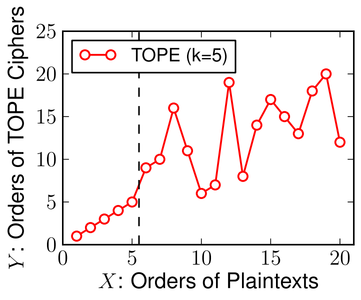


**FIGURE 16. The distribution of *Z* with TOPE.**

it is randomly distributed when $(k, k) \leq Z(z_i) \leq (n, n)$. The comparison between Fig. 15 and Fig. 16 vividly demonstrates the advantage of our TOPE over OPE when processing top-k queries on encrypted data.

### 2) PROBABILITY MASS FUNCTION

With the random variables above, we can further define Probability Mass Function (PMF) of a random variable to evaluate the statistical information of plaintexts and ciphertexts. More specifically, given random variable $X$, we define its PMF $F_X(x)$ as

$$F_X(x) = P(X(m_i) = x) = \frac{\texttt{count}(X(m_i))}{N} = \frac{\texttt{count}(m_i)}{N}$$

which intuitively indicates the number of a plaintext $m_i$ in a number of $N$ tuples. Similarly, given random variable $Y$, we can also define its PMF as

$$F_Y(y) = P(Y(c_i) = y) = \frac{\texttt{count}(Y(c_i))}{N} = \frac{\texttt{count}(c_i)}{N}$$

which intuitively describes the number of a ciphertext $c_i$ in a number of $N$ encrypted tuples.

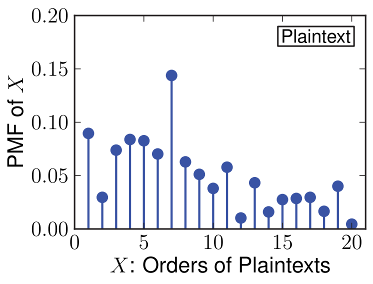


**FIGURE 17. PMF of *X* with plaintexts.**

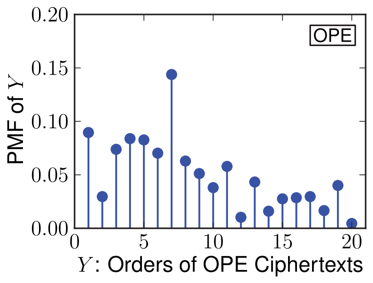


**FIGURE 18. PMF of *Y* with OPE ciphertexts.**

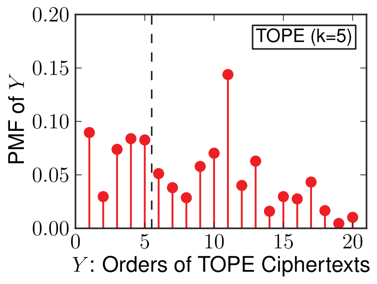


**FIGURE 19. The PMF of *Y* with TOPE ciphertexts.**

In order to evaluate these PMFs, we select all the messages/plaintexts in the diabetes dataset, where BG values are from 0 to 19 (as before, we again take $n = 20$ for the ease of showing the difference in the top-k part in figures). It turns out that the number of tuples satisfying this condition is 14, 204, which is about 48.5% of the entire dataset. The PMF of $X$ generated by plaintexts and the PMFs of $Y$ calculated from DE ciphertexts, OPE ciphertexts and TOPE ciphertexts are presented in Fig. 17 to Fig. 20.

As we can see from those figures, the PMF of $Y$ outputted by OPE ciphertexts is exactly the same as the PMF of $X$ generated from plaintext, which means by observing OPE ciphertexts only, a curious server can learn the exact PMF of the original plaintexts. However, with the use of our TOPE, the PMF of $Y$ shown in Fig. 19 is only consistent with the PMF of the original plaintexts when $y \leq k$ $(k = 5)$, while the actual PMF of the original plaintexts is preserved over TOPE ciphertexts (when $y > 5$). It is obvious to see from Fig. 20 that DE ciphertexts prevent a curious server from learning the entire PMF of the original plaintexts.

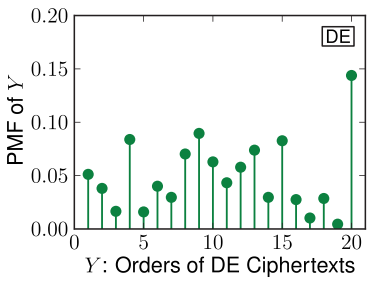


**FIGURE 20.** The PMF of $Y$ with DE ciphertexts.

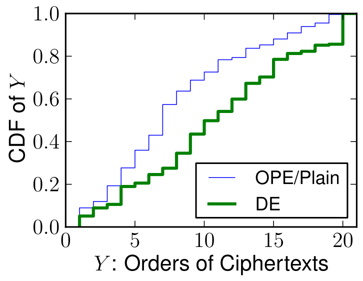


**FIGURE 21.** The CDFs of $Y$ with OPE and DE ciphertexts.

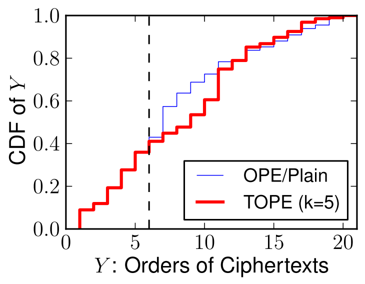


**FIGURE 22.** The CDFs of $Y$ with OPE and TOPE ciphertexts.

In other words, with the use of TOPE, a curious server only learns a permutation of the original PMF of the plaintexts by observing ciphertexts only, but it does not know which exact permutation it is among all the possible $(n-k)!$ cases. Similarly, with the implementation of DE, a curious server only reveals a permutation of the original PMF of the plaintexts, however, it cannot decide which particular permutation it is among all the possible $n!$ cases by analyzing ciphertexts only.

### 3) CUMULATIVE DISTRIBUTION FUNCTION

With a PMF of a random variable, we can further evaluate Cumulative Distribution Function (CDF) of this random variable. More concretely, given random variable $X$, its CDF $f_X(x)$ can be defined as

$$\begin{aligned} f_X(x) &= P(X(m_i) \leq x) = \sum_{X(m_i)=1}^{x} P(X(m_i)) \\ &= \sum_{X(m_i)=1}^{x} \frac{\texttt{count}(X(m_i))}{N} \end{aligned}$$

Similarly, given random variable $Y$, its CDF $f_Y(y)$ is

$$\begin{aligned} f_Y(y) &= P(Y(c_i) \leq y) = \sum_{Y(c_i)=1}^{y} P(Y(c_i)) \\ &= \sum_{Y(c_i)=1}^{y} \frac{\texttt{count}(Y(c_i))}{N} \end{aligned}$$

Essentially, CDF is another important approach to analyze the statistical information of data besides PMF.

Since the PMF of $Y$ obtained by OPE ciphertexts are exactly the same as the PMF of plaintexts, it is obvious that their CDFs are exactly the same. As illustrated in Fig. 21 and Fig. 22, we can see that the CDF of $Y$ calculated from TOPE ciphertexts is only as the same and consistent as the CDF of plaintexts when $y < 6$ ($k = 5$, note that we use $y < 6$ instead of $y \leq 5$ here since CDF is right-continuous [21]), while the CDF of $Y$ computed from DE ciphertexts is totally different to the CDF of plaintexts. The results in Fig. 21 and Fig. 22 indicate that, by analyzing ciphertexts only, a curious server can directly recover the entire CDF of plaintexts over OPE ciphertexts, but only a small partial CDF of plaintexts over TOPE ciphertexts. The comparisons among DE, OPE, and TOPE in Random Vectors, PMFs, and CDFs all statistical-wise demonstrate the relations of these three primitives describing in Fig. 1.

## IX. CONCLUSION

We propose a Top Order-Preserving Encryption to enable top-k queries over encrypted data. Our key approach is to utilize the partially-order property of heaps to balance the privacy and search functionality while applying a mutable encryption. As a result, it is able to alleviate privacy leakage of top-k queries compared to the use of OPE. Search efficiency and compatibility of our scheme with current SQL queries is obvious by conducting a set of extensive experiments on both random and real-world datasets. In addition, we properly and formally define its security with in a rigorous manner and statistically analyze its impact on real-world datasets.

## REFERENCES

[1] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-*k* query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, 2008, Art. no. 11.

[2] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-*k* query processing in uncertain databases," in *Proc. IEEE ICDE*, Apr. 2007, pp. 896–905.

[3] T. Ge, S. Zdonik, and S. Madden, "Top-*k* queries on uncertain data: On score distribution and typical answers," in *Proc. ACM SIGMOD*, 2009, pp. 375–388.

[4] *Cloud-Based Analytics: Supporting Healthcare's Digital Transformation*. Accessed: May 1, 2018. [Online]. Available: https://aws.amazon.com/health/resource-center/

[5] *Philips Healthcare Case Study—AWS*. Accessed: May 1, 2018. [Online]. Available: https://aws.amazon.com/solutions/case-studies/philips/

[6] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. ACM CCS*, 2006, pp. 895–934.

[7] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li, "Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *Proc. ACM ASIACCS*, 2014, pp. 111–122.

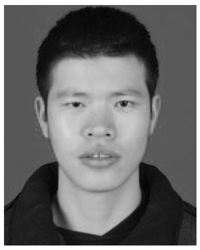
[8] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE SP*, May 2000, pp. 44–55.

[9] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neil, "Order-preserving symmetric encryption," in *Proc. EUROCRYPT*, 2009, pp. 224–241.
[10] A. Boldyreva, N. Chenette, and A. O'Neil, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. CRYPTO*, 2011, pp. 578–595.
[11] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. IEEE SP*, May 2013, pp. 463–477.
[12] F. Kerschbaum and A. Schropfer, "Optimal average-complexity ideal-security order-preserving encryption," in *Proc. ACM CCS*, 2014, pp. 275–286.
[13] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. ACM CCS*, 2015, pp. 656–667.
[14] 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 *Proc. EUROCRYPT*, 2015, pp. 563–594.
[15] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *Proc. FSE*, 2016, pp. 474–493.
[16] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. ACM CCS*, 2016, pp. 1167–1178.
[17] C.-M. Yu, G.-K. Ni, I.-Y. Chen, E. Gelenbe, and S.-Y. Kuo, "Top-*k* query result completeness verification in tiered sensor networks," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 1, pp. 109–124, Jan. 2014.
[18] F. Baldimtsi and O. Ohrimenko, "Sorting and searching behind the curtain," in *Proc. FC*, 2015, pp. 127–146.
[19] W. Sun *et al.*, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. ACM AISACCS*, 2013, pp. 71–82.
[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[21] R. D. Yates and D. J. Goodman, *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*, 3rd ed. Hoboken, NJ, USA: Wiley, 2014.
[22] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. ACM SOSP*, 2011, pp. 85–100.
[23] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM CCS*, 2012, pp. 965–976.
[24] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Proc. CRYPTO*, 2013, pp. 353–373.
[25] V. Pappas *et al.*, "Blind seer: A scalable private DBMS," in *Proc. IEEE SP*, May 2014, pp. 359–374.
[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. TCC*, 2007, pp. 535–554.
[27] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE SP*, May 2007, pp. 350–364.
[28] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," in *Proc. NDSS*, 2012, pp. 1–17.
[29] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.
[30] M. Bellare, T. Kohno, and C. Namprempre, "Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol," in *Proc. ACM CCS*, 2002, pp. 1–11.
[31] *UCI Machine Learning Repository: Diabetes Data Set*. Accessed: May, 1 2018. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Diabetes

**HANYU QUAN** received the B.S. degree in information security and the M.S. degree in cryptography from Xidian University, China, in 2011 and 2014, respectively, where he is currently pursuing the Ph.D. degree with the School of Cyber Engineering. He was a Visiting Ph.D. Student with the Department of Electrical and Computer Engineering, The University of Arizona. His research interests include privacy-enhancing technologies and applied cryptography.

**BOYANG WANG** received the B.S. degree in information security and the Ph.D. degree in cryptography from Xidian University in 2007 and 2013, respectively, and the Ph.D. degree in electrical and computer engineering from The University of Arizona in 2017. In 2015, he joined the Bosch Research and Technology Center as a Research Intern. He was a Visiting Student with the University of Toronto and also with Utah State University. He is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science, University of Cincinnati. He has authored over 20 research papers in top journals and conferences, including TIFS, TDSC, TSC, TPDS, INFOCOM, CNS, ACM ASIACCS, and ICDCS. His research interests include privacy-enhancing technologies, data privacy, and applied cryptography.

**YUQING ZHANG** received the Ph.D. degree in cryptography from Xidian University, China. He is currently a Professor with the School of Cyber Engineering, Xidian University, and the Director of the National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences. He has authored over 100 research papers in international journals and conferences, such as IEEE TDSC, IEEE TPDS, and ACM CCS. His research interests include network and system security. His research has been sponsored by NSFC, Huawei, Qihu360, and Google. He has served as a PC Member for more than 10 international conferences in networking and security, such as IEEE Globecom 16/17, IEEE CNS 17, and IFIP DBSec 17.



**GAOFEI WU** received the B.S. and Ph.D. degrees from Xidian University, China, in 2009 and 2015, respectively. During 2012–2014, he was a Visiting Ph.D. Student with the Department of Informatics, University of Bergen. He is currently a Lecturer with the School of Cyber Engineering, Xidian University. His research interests include cryptography and sequence design.

• • •