Frequency-revealing attacks against Frequency-hiding Order-preserving Encryption

Xinle Cao
Zhejiang University
Hangzhou, China
xinle@zju.edu.cn

Jian Liu*
Zhejiang University
Hangzhou, China
liujian2411@zju.edu.cn

Yongsheng Shen
Hang Zhou City Brain Co., Ltd
Hangzhou, China
sys@cityos.com

Xiaohua Ye
Hang Zhou City Brain Co., Ltd
Hangzhou, China
Veraye926@163.com

Kui Ren
Zhejiang University
Hangzhou, China
kuiren@zju.edu.cn

ABSTRACT

Order-preserving encryption (OPE) allows efficient comparison operations over encrypted data and thus is popular in encrypted databases. However, most existing OPE schemes are vulnerable to inference attacks as they leak plaintext frequency. To this end, some *frequency-hiding* order-preserving encryption (FH-OPE) schemes are proposed and claim to prevent the leakage of frequency. FH-OPE schemes are considered an important step towards mitigating inference attacks.

Unfortunately, there are still vulnerabilities in all existing FH-OPE schemes. In this work, we revisit the security of all existing FH-OPE schemes. We are the first to demonstrate that plaintext frequency hidden by them is recoverable. We present three ciphertext-only attacks named *frequency-revealing attacks* to recover plaintext frequency. We evaluate our attacks in three real-world datasets. They recover over 90% of plaintext frequency hidden by any existing FH-OPE scheme. With frequency revealed, we also show the potentiality to apply inference attacks on existing FH-OPE schemes.

Our findings highlight the limitations of current FH-OPE schemes. Our attacks demonstrate that achieving frequency-hiding requires addressing the leakages of both non-uniform ciphertext distribution and insertion orders of ciphertexts, even though the leakage of insertion orders is always ignored in OPE.

PVLDB Reference Format:
Xinle Cao, Jian Liu, Yongsheng Shen, Xiaohua Ye, and Kui Ren. Frequency-revealing attacks against Frequency-hiding Order-preserving Encryption. PVLDB, 16(11): 3124-3136, 2023.
doi:10.14778/3611479.3611513

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/XinleCao/Frequency-revealing-Attack.

*Jian Liu is the corresponding author.
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611513

1 INTRODUCTION

Order-preserving encryption (OPE) [9, 10, 12, 21, 22, 30, 34] is a widely used technique in encrypted databases for performing range queries [5, 31, 35]. It preserves plaintext order in ciphertexts to allow efficient comparison operations over encrypted data. However, the security of OPE has been debated for a long time. Boldyreva et al. [9] present the *ideal-security* of OPE: not to reveal any other information besides plaintext order. But most existing OPE schemes [9, 22, 30] additionally leak plaintext frequency because they are designed to be deterministic, i.e., the same plaintext is always encrypted to the same ciphertext. The leakage of frequency makes OPE vulnerable to plenty of frequency-based inference attacks [6, 14, 18, 28].

To this end, some *frequency-hiding* order-preserving encryption (FH-OPE) schemes [21, 25, 32] are proposed to protect plaintext frequency. Kerschbaum presents the first FH-OPE scheme [21]. The client in this scheme maintains the mapping between plaintexts and ciphertexts so it can encrypt each plaintext to a unique ciphertext. However, this scheme is hard to deploy as the mapping requires $O(n)$ storage space in the client where n is the number of plaintexts. Roche et al. also present a FH-OPE scheme named POPE [32] and reduce the storage cost in the client to $O(1)$. But the client in POPE has to interact with the server $O(\log n)$ rounds for each range query, which makes the scheme still unrealistic. The state-of-the-art FH-OPE scheme [25] is recently proposed by Li et al. in VLDB '21. It achieves only $O(N)$ storage space in the client and 1 interaction per range query, where N is the number of distinct plaintexts.

Motivation. These FH-OPE schemes have been recognized as a crucial advancement in mitigating inference attacks, as they conceal the frequency of plaintext values, making them impervious to all frequency-based inference attacks. The best-known published inference attack against FH-OPE schemes is the *binomial attack*, which is based on only plaintext order. But it is an ineffective attack since it recovers at most 30% of plaintexts in [18] and 15% of plaintexts in [19]. As a result, FH-OPE schemes are still recommended for use in encrypted databases by various studies [18, 19, 25].

Our contribution. However, we found there are still vulnerabilities in all existing FH-OPE schemes. In this work, we revisit these schemes and expose the overestimation of their security. Surprisingly, our analysis reveals that plaintext frequency in all existing FH-OPE schemes is recoverable, which is a new finding to the best

of our knowledge. We present three novel ciphertext-only attacks named *frequency-revealing attacks* to recover plaintext frequency. We summarize our contributions as follows:

(1) We revisit the security of Kerschbaum's FH-OPE scheme and present a frequency-revealing attack named *density attack* to recover plaintexts with only ciphertexts (§ 4).
(2) We revisit the security of POPE and the state-of-the-art FH-OPE scheme. We present two frequency-revealing attacks named *Fisher exact test attack* and *binomial test attack* against them, respectively. The two attacks are based on only ciphertexts and their partial insertion orders. As far as we know, they are the first attacks exploiting the leakage of ciphertexts insertion orders [7] (§ 5).
(3) We validate our attacks on three real-world datasets. In our experiments, our attacks recover more than 90% of plaintext frequency hidden by any existing FH-OPE scheme. With frequency revealed, we also evaluate the potential for inference attacks on existing FH-OPE schemes (§ 6).

2 PRELIMINARIES

Notation. For positive integer n, $[n]$ is the set $\{1, ..., n\}$, and $|I|$ is the number of elements in set I. $r \leftarrow\$ I$ means sampling an element uniformly at random from I. $[a, b]$ and (a, b) denote integer sets $\{a, a+1, ..., b\}$ and $\{a+1, a+2, ..., b-1\}$, respectively. $\mathsf{Ber}(p)$ is the Bernoulli distribution, returning 1 with probability p and 0 with probability $1-p$. $\mathsf{Bin}(n, p)$ is the Binomial distribution, representing the number of successes in n independent trials, each with probability p of success.

Order-preserving encryption (OPE). A (stateful) OPE scheme $\mathsf{OPE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ consists of three algorithms:

- $\mathsf{st} \leftarrow \mathsf{KeyGen}(1^\lambda)$: Generates a secret state st according to the security parameter λ.
- $\mathsf{st}', c \leftarrow \mathsf{Encrypt}(\mathsf{st}, v)$: Computes the ciphertext c for plaintext v and updates the state from st to st′.
- $v \leftarrow \mathsf{Decrypt}(\mathsf{st}, c)$: Computes the plaintext v for ciphertext c based on state st.

It satisfies: for any plaintexts v_1, v_2, valid state st, and $\mathsf{st}', c_i = \mathsf{Encrypt}(\mathsf{st}, v_i)$, if $v_1 > v_2$ then $c_1 > c_2$. A deterministic OPE scheme additionally satisfies if $v_1 = v_2$ then $c_1 = c_2$.

Inference attacks. Most existing OPE schemes [9, 10, 30?] are deterministic and inherently leak both plaintext order and frequency. The leakages incur dangerous frequency-based inference attacks. We provide a brief overview of the two primary types of attacks. They can recover most plaintexts (> 80%) protected by a deterministic OPE scheme.

- *Sorting attack.* It is presented in [29] for dense data (e.g., age, gender), where each distinct plaintext in plaintext space $\mathbb{M}$ is encrypted at least once. In this case, the number of distinct OPE ciphertexts is equal to the number of distinct plaintexts. The attacker recovers plaintexts by mapping sorted distinct ciphertexts one-to-one to the sorted distinct plaintexts. So this attack requires that the attacker knows the plaintext space $\mathbb{M}$.
- *Frequency-analyzing attacks.* These attacks [6, 18, 29] are suitable for low-density data where only some plaintexts in $\mathbb{M}$ are encrypted. They apply both frequency and order leakages to find the mapping between ciphertexts and plaintexts such that the distributions of ciphertexts and plaintexts are close. So these attacks require that the attacker estimates plaintext distribution in advance with public auxiliary information.

Randomized order. Kerschbaum defines *randomized order* [21] to describe the plaintext order in FH-OPE.

Definition 1 (randomized order). *Let n be the number of not necessarily distinct plaintexts in sequence $V = \{v_1, v_2, ..., v_n\}$. For a randomized order $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_n\}$ $(\forall i.1 \le \gamma_i \le n, \forall i, j. i \ne j \implies \gamma_i \ne \gamma_j)$ of sequence V, it holds that*

$$\forall i, j \in [n], v_i > v_j \implies \gamma_i > \gamma_j$$

and

$$\forall i, j \in [n], \gamma_i > \gamma_j \implies v_i \ge v_j$$

3 SECURITY PROPERTIES

3.1 Security model

Kerschbaum defines frequency-hiding OPE (FH-OPE) with a formal security guarantee named *indistinguishability under frequency-analyzing ordered chosen-plaintext attack* (IND-FAOCPA) [21]. An OPE scheme achieving IND-FAOCPA secure is a FH-OPE scheme.

IND-FAOCPA security game. The security game $Game^{FAOCPA}_{\mathcal{A},\Pi}(\lambda)$ between a challenger and an adversary $\mathcal{A}$ for an OPE scheme Π with security parameter λ proceeds as follows:

(1) The adversary $\mathcal{A}$ chooses two sequences V_0 and V_1 of n not necessarily distinct plaintexts, such that they have at least one common randomized order Γ. He sends them to the challenger.
(2) The challenger flips an unbiased coin $b \in \{0, 1\}$, executes the key generation $\Pi.\mathsf{KeyGen}(1^\lambda)$, and encrypts $V_b = \{v_1^b, ..., v_n^b\}$ as $C' = \{c'_1, ..., c'_n\}$, i.e., $\mathsf{st}_i, c'_i \leftarrow \Pi.\mathsf{Encrypt}(\mathsf{st}_{i-1}, v_i^b)$. He sends the ciphertexts C' to the adversary.
(3) The adversary $\mathcal{A}$ outputs a guess b^* of b.

Definition 2. *An OPE encryption scheme Π is IND-FAOCPA secure against frequency-analyzing ordered chosen plaintext attack if the adversary $\mathcal{A}$'s advantage of outputting b in $Game^{FAOCPA}_{\mathcal{A},\Pi}(\lambda)$ is negligible in λ, i.e.*

$$Pr[Game^{FAOCPA}_{\mathcal{A},\Pi}(\lambda) = b] < \frac{1}{2} + \frac{1}{poly(\lambda)}$$

Adversary power. *It is clear that FH-OPE encrypts V by inserting the ciphertexts of V one by one.* The adversary in the game above observes C', which preserves 1) all the order-preserving ciphertexts of V including the entire ciphertext distribution; 2) the exact insertion order of each ciphertext, i.e., the insertion order of c'_i is i. Therefore, suppose the adversary sorts ciphertexts in C' according to their orders and get ordered ciphertexts $C = \{c_1, ..., c_n\}$, we can formally describe the two leakage profiles as

$$\mathcal{L}_0(V) = \{(c_1, \mathsf{id}_1), ..., (c_n, \mathsf{id}_n)\}$$

where id_i is the exact insertion order of c_i and $\forall i, j \in [n], c_i < c_j$ iff $i < j$. In reality, this adversary captures a *passive* (honest-but-curious) attacker: *persistent attacker*. It does not deviate from the protocol specified or access the client to issue queries but can get any information available in the server, e.g., query execution and results. So the persistent attacker can get all ciphertexts and the exact insertion order of each ciphertext, i.e., $\mathcal{L}_0(V)$.

3.2 Threat model

In this paper, we consider an adversary which is weaker than the adversary assumed in FH-OPE. It still has access to all ciphertexts but is limited to obtaining only partial information about the insertion orders. Given ordered ciphertexts $C = \{c_1, ..., c_n\}$ of V, we refer to the leakage profiles obtained by this adversary as $\mathcal{L}_1(V)$ and describe them as follows:

$$\mathcal{L}_1(V) = \{(c_1, \mathsf{part}(\mathsf{id}_1)), ..., (c_n, \mathsf{part}(\mathsf{id}_n)\}$$

where $\mathsf{part}(\mathsf{id}_i)$ represents partial information about the insertion order id_i and $\forall i, j \in [n], \mathsf{part}(\mathsf{id}_i) \geq \mathsf{part}(\mathsf{id}_j)$ if $\mathsf{id}_i > \mathsf{id}_j$. For example, with $\mathcal{L}_0(V) = \{(c_1, 1), (c_2, 3), (c_3, 2)\}$, we may have $\mathcal{L}_1(V) = \{(c_1, 1), (c_2, 2), (c_3, 1)\}$ which implies c_1 and c_3 are ciphertexts inserted prior to c_2 but it is unknown which of c_1 and c_3 was inserted earlier. This adversary captures a very weak attacker in reality: *snapshot attacker*. It only tries to steal *one* or *multiple* snapshots of the encrypted database and cannot access any in-memory information related to the execution in the server. According to the number of snapshots, we divide the snapshot attacker as *single-snapshot attacker* and *multi-snapshot attacker*. The single-snapshot attacker [18, 28] observes one snapshot of the encrypted database. It can get only ciphertexts and their orders. i.e., ordered ciphertexts $C = \{c_1, ..., c_n\}$. The multi-snapshot attacker [4, 13] has access to multiple snapshots of the encrypted database. Each snapshot is interspersed with a batch of (insertion) operations. To guarantee a very weak multi-snapshot attacker, in this paper, we assume the multi-snapshot attacker gets **no more than** 11 **snapshots** while there are millions of operations.

Definition 3 (multi-snapshot attacker). *A multi-snapshot attacker accesses the server at $\mu + 1$ ($\mu \geq 1$) ordered distinct moments $T = \{t_0, t_1, ..., t_\mu\}$ and observes the encrypted database. In the end, it gets ordered ciphertexts $C = \{c_1, ..., c_n\}$ and an indicator sequence $\{\mathsf{part}(\mathsf{id}_1), ..., \mathsf{part}(\mathsf{id}_n)\}$ where $\mathsf{part}(\mathsf{id}_i) \in [\mu] \cup \{0\}$ indicates that c_i is firstly observed in $t_{\mathsf{part}(\mathsf{id}_i)}$ by the attacker.*

3.3 Frequency-revealing attacks

Our attacks are the first trying to recover plaintext frequency in FH-OPE. We define *frequency-revealing attacks* by finding the *order range* of ciphertexts whose underlying plaintexts are the same. Roughly speaking, given ordered ciphertexts $C = \{c_1, ..., c_n\}$, we reveal the frequency of v by finding the minimal and maximal orders of its ciphertexts, i.e., we find $[\phi, \pi]$ such that the underlying plaintext of ciphertexts $\{c_\phi, c_{\phi+1}, ..., c_\pi\}$ is v and the underlying plaintexts of all other ciphertexts in C are not v. For example, let $\{7, 7, 7, 8\}$ be encrypted to $\{c_1, c_2, c_3, c_4\}$. We find the ciphertexts of plaintext 7 are $\{c_1, c_2, c_3\}$ and corresponding order range is $[1, 3]$. So for plaintext 7 we have $[\phi, \pi] = [1, 3]$. Now we formally define frequency-revealing attacks as below:

Definition 4 (Frequency-revealing attack). *In a FH-OPE scheme Π, let $C = \{c_1, ..., c_n\}$ be an ordered list of (randomized) ciphertexts for N ordered distinct plaintexts $\{v^1, ..., v^N\}$. A frequency-revealing attack* Attack *works by finding two sorted order vectors:*

$$(\boldsymbol{\phi}, \boldsymbol{\pi}) \leftarrow \mathsf{Attack}(C, \mathsf{aux})$$

where $\boldsymbol{\phi} = (\phi_1, ..., \phi_N)$ and $\boldsymbol{\pi} = (\pi_1, ..., \pi_N)$ ($\forall i \in [N], 1 \leq \phi_i \leq \pi_i \leq n$) and aux *denotes some (possible) auxiliary information like partial insertion orders of ciphertexts. It holds that*

$$j \in [\phi_i, \pi_i] \iff v^i \leftarrow \Pi.\mathsf{Decrypt}(\mathsf{st}, c_j).$$

Example. Let plaintexts $\{7, 7, 7, 8, 8, 9\}$ be encrypted to ordered ciphertexts $\{c_1, ..., c_6\}$. The ciphertexts of plaintexts 7, 8 and 9 are $\{c_1, c_2, c_3\}$, $\{c_4, c_5\}$ and $\{c_6\}$, respectively. The corresponding order ranges are $[1, 3]$, $[4, 5]$ and $[6, 6]$, respectively. Therefore, the vector $\boldsymbol{\phi}$ consisting of minimal orders in these order ranges is $(1, 4, 6)$ while the vector $\boldsymbol{\pi}$ consisting of maximal orders in these order ranges is $(3, 5, 6)$. Throughout this paper, we only try to find $\boldsymbol{\pi}$ since $\boldsymbol{\phi}$ can be calculated based on $\boldsymbol{\pi}$: $\phi_{i+1} = \pi_i + 1$ ($i \in [N - 1]$) and $\phi_1 = 1$.

4 ATTACKING KERSCHBAUM'S FH-OPE SCHEME

In this section, we analyze the security of Kerschbaum's FH-OPE scheme under the single-snapshot attacker and identify key properties of its ciphertext distribution. Based on the properties, we present a novel frequency-revealing attack named *density attack*. It recovers most plaintext frequency hidden by this scheme with only ciphertexts.

4.1 Review

Encryption. Kerschbaum's FH-OPE scheme utilizes a binary search tree T to store the mapping between plaintexts and OPE ciphertexts. Each node in T represents a plaintext and its corresponding ciphertext, and the nodes are sorted according to plaintext order. To encrypt a new plaintext v, the client searches for its order in T and then assigns it to the ciphertext that is the average of the ciphertexts of the next smaller plaintext and the next greater plaintext. For example, if plaintexts $\{0, 2\}$ are encrypted to $\{0, 100\}$, then a new plaintext 1 is encrypted to the ciphertext 50.

Frequency-hiding. To map repeated plaintexts to different ciphertexts, the client randomly determines their order relations by calling a function named RandomCoin(), which outputs 0 and 1 with the same probability 1/2. We give an example of Kerschbaum's FH-OPE scheme inFigure 1. We show the growth of T with four subfigures. There are two calls for RandomCoin(): When encrypting plaintext 7 for the second time in subfigure 3, it outputs 1 so the second 7 is regarded as greater than the first 7; When encrypting 8 for the second time in subfigure 4, it outputs 0 so the first 8 is greater.

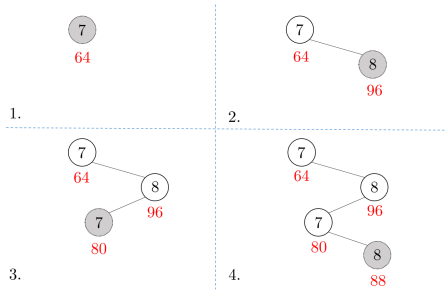

Figure 1. Let $\mathbb{C} = [0, 128]$, plaintexts $V = \{7, 8, 7, 8\}$ are encrypted to $\{64, 96, 80, 88\}$ as RandomCoin() outputs 1 and 0 in subfigure 3 and 4, respectively.

4.2 Observations

Kerschbaum's FH-OPE scheme guarantees each plaintext is mapped to a unique ciphertext, preventing repeated ciphertexts leaking frequency. However, it does not consider the leakage from *ciphertext distributions*. We give examples to introduce the leakage intuitively and then formally describe it with some observations.

Examples. The ciphertext distribution in Kerschbaum's FH-OPE scheme often exhibits non-uniformity, thereby leaking plaintext frequency. Suppose $\mathbb{C} = [0, 1024]$, consider the following example. We divide this example into three phases for ease of presentation. In each phase, we make RandomCoin() output the same number of 0 and 1 to guarantee fair randomness.

(1) The client first encrypts $\{7, 8, 7, 8\}$ to $\{512, 768, 256, 896\}$. It indicates RandomCoin() is called twice and outputs $\{0, 1\}$.
(2) Then the client encrypts new plaintexts $\{7, 7, 7, 8\}$ and gets new ciphertexts $\{640, 128, 704, 736\}$. RandomCoin() is called six times and outputs $\{1, 0, 0, 1, 1, 0\}$.
(3) Finally, the client encrypts new plaintexts $\{8, 8, 8, 7\}$ and gets new ciphertexts $\{960, 832, 720, 576\}$. RandomCoin() is called eight times and outputs $\{1, 1, 1, 0, 0, 0, 1, 0\}$.

We visualize all the ciphertexts in $\mathbb{C}$ inFigure 2, revealing the non-uniform ciphertext distribution and an important phenomenon: *In general, larger ciphertexts of* 7 *are closer than smaller ciphertexts of* 7*, while smaller ciphertexts of* 8 *are closer than larger ciphertexts of* 8. Consequently, the single-snapshot attacker can guess the ciphertexts belong to two distinct plaintexts although it does not know the values of the two plaintexts. It estimates the maximal ciphertext of the smaller plaintext and the minimal ciphertext of the larger plaintext with the closest neighboring ciphertexts such as 704 and 720. Then it recovers the plaintext frequency without knowing the plaintexts are 7 and 8. In some cases, the closet neighboring ciphertexts are not exactly correct for the guess, e.g., in the third phase, $\{8, 8, 7, 7\}$ are encrypted to $\{960, 832, 720, 576\}$ so the maximal ciphertext of 7 is 720 but the attacker incorrectly guesses 704. However, it is still dangerous as the attacker finds most ciphertexts having the same underlying plaintext and recover the frequency under a small error.

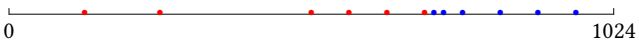

Figure 2. The ciphertext distribution in the example. The ciphertexts of 7 and 8 are represented by red and blue points, respectively.

We also provide an example where all plaintexts are distinct. Consider $\mathbb{C} = [0, 128]$ and encrypting the plaintexts $\{1, 4, 2, 6, 7, 5, 9\}$ produces $\{64, 96, 80, 112, 120, 104, 124\}$. The single-snapshot attacker knows 64 is the first encrypted ciphertext because it occupies the mean value in $\mathbb{C}$. It observes that all ciphertexts are no smaller than the first encrypted ciphertext 64. However, if all ciphertexts had the same underlying plaintext, half of the ciphertexts (excluding 64) would be expected to be smaller than 64 due to RandomCoin(). Hence, the attacker infers there are distinct plaintexts in the sequence, i.e., ciphertexts 64 and 124 have distinct underlying plaintexts w.h.p. Similarly, It can further deduce that 80 and 124 have distinct underlying plaintexts w.h.p. by examining the set of ciphertexts larger than 64, e.g., if all ciphertexts in the set have the same underlying plaintexts, then half of them (excluding 80) are expected to be smaller than 80. Now we formally describe and explain the unusual ciphertext distributions with our two observations.

Observation 1. *For any two nodes* P_1 *and* P_2 $(\mathsf{P}_2 > \mathsf{P}_1)$ *in* T*, suppose their ciphertexts are neighboring, i.e.,* $\mathsf{P}_2.cipher$ *is the next greater ciphertext of* $\mathsf{P}_1.cipher$*. Denote the depth of a node in* T *as* depth()*, then it holds that:*

$$\mathsf{P}_2.cipher - \mathsf{P}_1.cipher = \frac{|\mathbb{C}|}{2^{\eta}}.$$

where $\eta = max(\mathsf{depth}(\mathsf{P}_1), \mathsf{depth}(\mathsf{P}_2))$.

Observation 2. *Denote the first encrypted node of plaintext* v^i *as* P*. For any two nodes of* v^i *whose ciphertexts are neighboring, the maximal depth of them is likely to be greater if they are farther away from the first node* P.

The proofs and detailed explanations for the two observations are available in the full version [11]. Observation 1 builds a connection between the ciphertext distribution with the structure of the search tree T. Specifically, it indicates that the distance between neighboring ciphertexts is determined by the depths of their corresponding nodes. Observation 2 investigates the distribution of node depths to gain insight into the ciphertext distribution.

Ciphertext distribution. We combine the observation above to draw conclusions about the ciphertext distribution. Firstly, Observation 2 shows for any two nodes of v^i whose ciphertexts are neighboring, the maximal depth of them denoted by η is likely to be larger when they are farther away from the first encrypted node of v^i. Next, Observation 1 suggests the distance between ciphertexts of the two nodes depends on η: a larger η results in a smaller distance. Therefore, when the two nodes are farther away from the first encrypted node of v^i, the distance between their ciphertexts is likely to be smaller. Suppose we traverse from the smallest node of v^i to the second-greatest node of v^i. For each node, we calculate the distance between its ciphertexts and the next greater ciphertext. Our analysis implies there are two overall stages in the traverse:

(1) *Increase stage.* As we traverse towards the first encrypted node of v^i, the value of η overall decreases, the distance increases.
(2) *Decrease stage.* As we traverse away the first encrypted node of v^i, the value of η overall increases, the distance decreases.

Tree depth. Note the non-uniform distribution is independent of the depth of search tree T. It results from the gaps between the depths of nodes corresponding to the same plaintext. Moreover, the presence of these gaps is a consequence of multiple distinct plaintexts being encrypted within this scheme. Thus, in our experiments, the accuracy of our attack applying the non-uniform ciphertext distribution in this scheme is always more than 96% even if the tree depth varies between 32 and 57 with database size varying from $\sim$ 196K to $\sim$ 532239K.

Reducing randomness. The randomness introduced by RandomCoin() also affects the depth of nodes of v^i, although its impact is limited. As a result, while the distance distribution can generally be divided into two overall stages, there may be some local fluctuations in each stage. To reduce the randomness impact, we provide a more robust representation of the ciphertext distribution, which we call the *α-ciphertext density*.

Algorithm 1: Density attack

Input: Ordered ciphertexts $C = \{c_1, ..., c_n\}, \alpha, \gamma$

Output: An order vector $\boldsymbol{\pi'} = (\pi'_1, ..., \pi'_{N'})$

```
stage = increase
density = −1, order = −1
a = 1
for j = α + 1 → n − α do                    traverse ciphertexts
    d = c_{j+α} − c_{j−α}
    if stage = increase then
        if d > density then
            density = d, order = j
        end
        if density > γ · d then                          Case 1
            stage = decrease
            density = d, order = j
        end
    end
    if stage = decrease then
        if d < density then
            density = d, order = j
        end
        if γ · density < d then                          Case 2
            stage = increase
            π'_a = order, a += 1
            density = d, order = j
        end
    end
end
```

Definition 5 (α-ciphertext density). *For n ordered ciphertexts $C = \{c_1, ..., c_n\}$ in Kerschbaum's FH-OPE scheme, the α-ciphertext density of $c_j (\alpha < j \leq n - \alpha)$ is denoted as $d_{\alpha,j}$ and is calculated as:*

$$d_{\alpha,j} = c_{j+\alpha} - c_{j-\alpha}.$$

When we traverse ordered ciphertexts of the same plaintext and calculate α-ciphertext density, the two overall stages described earlier still apply, and the local fluctuations are more slight. This provides the potential of recovering frequency by finding the decrease stage of ciphertexts of v^i and the increase stage of ciphertexts of v^{i+1}.

4.3 Density Attack

Based on the observations and analysis, we present a frequency-revealing attack named *density attack* inAlgorithm 1. It takes only ordered ciphertexts and attacking parameters (α, γ) as inputs and outputs a vector $\boldsymbol{\pi'}$, which is an estimation of $\boldsymbol{\pi}$. This attack works by traversing the ordered ciphertexts and calculating their α-ciphertext density. When it traverses from the decrease stage of ciphertexts of v^i to the increase stage of ciphertexts of v^{i+1}, the minimal density is considered as an estimation of d_{α,π_i}.

Converting stages. When the attack traverses ciphertexts, it uses *stage* to record the stage currently traversed. It stores the maximal (minimal) density in the increase (decrease) stage and corresponding order in *density* and *order*, respectively. There are two cases for the attack to convert stages.

- Case 1. In the increase stage, *density* records the maximal density in this stage. If current density is much smaller than the maximal density ($density > \gamma \cdot d$), which violates the expectation for the increase stage, the attack knows the increase stage is over and converts the stage into decrease.
- Case 2. Similarly, in the decrease stage, *density* records the minimal density in this stage. If current density is much larger than the minimal density ($\gamma \cdot density < d$), the attack knows the decrease stage is over and converts the stage into increase.

The conversion in Case 2 indicates the traverse from the decrease stage of ciphertexts of v^i to the increase stage of ciphertexts of v^{i+1}. Therefore, the attack outputs the value of *order* in Case 2 as the estimation of π_i.

5 ATTACKING POPE AND THE FH-OPE SCHEME IN VLDB '21

In this section, we revisit the other two FH-OPE schemes: POPE and the FH-OPE scheme in VLDB '21. They guarantee uniform ciphertext distribution by encrypting plaintexts with a semantically secure encryption scheme S = (S.KeyGen, S.Encrypt, S.Decrypt). In these schemes, the client uploads ciphertexts along with the orders of their corresponding plaintexts, allowing the server to compare ciphertexts based on their orders. The encryption process in these schemes for a plaintext v involves the following steps:

(1) The client uploads v's semantically secure ciphertext c and the order of v to the server.
(2) The server always organizes ciphertexts as a search tree T according to the orders of their underlying plaintexts. It inserts c into T based on the order of v.

The two schemes are secure under a single-snapshot attacker as they leak only plaintext order to this attacker. However, under a multi-snapshot attacker (cf.Definition 3), we show that the two schemes are still vulnerable to frequency-revealing attacks.

Batch encryption. The multi-snapshot attacker observes ordered ciphertexts $C = \{c_1, ...c_n\}$ at $\mu + 1$ distinct ordered moments. For simplicity, we divide the ciphertexts into batches:

(1) *Setup batch.* This initial batch consists of all ciphertexts observed in t_0, denoted as ordered ciphertexts $C^0 = \{c_1^0, ...c_{n_0}^0\}$.
(2) *Insertion batches.* The ith batch consists of all ciphertexts firstly observed in t_i, denoted as ordered ciphertexts:

$$\forall i \in [\mu], C^i = (c_1^i, ..., c_{n_i}^i) .$$

These batches are inserted into the setup batch. The sum of the sizes of all insertion batches is $n - n_0$.

5.1 Review

POPE. POPE randomizes orders of repeated plaintexts by adding a random fractional component between 0 and 1 on each plaintext. For example, to encrypt $\{7, 7, 7, 8\}$, the client samples random components $\{0.70, 0.32, 0.45, 0.04\}$ and adds them on plaintexts in order. The final plaintexts that the client encrypts are

$\{7.70, 7.32, 7.45, 8.04\}$ and the randomized order is $\{3, 1, 2, 4\}$. *In this way, there is no plaintext with the same order in POPE.*

In POPE, the server organizes semantically secure ciphertexts as a search tree called POPE tree, which is similar to a standard B tree. Ciphertexts can be compared according to their positions in the POPE tree. For ease of presentation, we just describe the POPE tree as a binary search tree T. To encrypt a new plaintext v, the client first samples a random component $r \leftarrow\$ \{0,1\}^l/2^l$ and encrypts $v + r$ with semantically secure encryption. Then the client interacts with the server to search the insertion position of $v + r$ in T and tells the server where to insert the ciphertext.

The FH-OPE scheme in VLDB '21. In this scheme, the server also organizes semantically secure ciphertexts as a search tree. It achieves $O(1)$ interaction by requiring the client to store plaintext frequency locally. In detail, the client records plaintext frequency with a local table $LT = \{(v^i, fre(v^i))\}$ where $fre(v^i)$ denotes the frequency of v^i. With this table, the client obtains the order of v by calculating the number of plaintexts smaller than v. To achieve *frequency-hiding*, client assigns v with a random order $rd \leftarrow\$ [l, u]$ where $l = \sum_{v^i<v} fre(v^i)$ and $u = \sum_{v^i \le v} fre(v^i)$. When $rd > 0$, the ciphertext of v will be inserted between c_{rd} and c_{rd+1}. When $rd = 0$, this ciphertext will be inserted as the smallest ciphertext.

There are also some other important techniques in the two schemes. POPE presents the idea of partial order-preserving and the FH-OPE scheme in VLDB '21 applies a coding tree. We discussed them in the full version [11] and here we focus on the frequency-hiding property.

5.2 Observations

Under the multi-snapshot attacker, both POPE and the FH-OPE scheme in VLDB '21 leak partial insertion orders of ciphertexts, which can subsequently result in the leakage of plaintext frequency. For each of the two schemes, we provide simple examples to show the frequency leakage and then describe the leakage with a formal observation. While these examples provided are too small to fully illustrate our frequency-revealing attacks, they are sufficient to show the frequency leakage and basic intuition behind the attacks.

5.2.1 Leakage in POPE. Consider an example in POPE where the multi-snapshot attacker observes three batches of ciphertexts.

(1) *Setup batch.* Suppose $\{7, 8, 7, 8\}$ are encrypted with random components $\{0.11, 0.80, 0.96, 0.92\}$. Let the resulting **ordered** ciphertexts be $\{c_1, c_2, c_3, c_4\}$ corresponding to $\{7.11, 7.96, 8.80, 8.92\}$.
(2) *Insertion batch 1.* Plaintexts $\{7, 7, 7, 8\}$ are encrypted and inserted with random components $\{0.22, 0.69, 0.78, 0.31\}$.
(3) *Insertion batch 2.* Plaintexts $\{8, 8, 8, 7\}$ are encrypted and inserted with random components $\{0.19, 0.58, 0.42, 0.81\}$.

We visualize the insertion process inFigure 3. Notably, an unusual insertion pattern emerges: the majority of ciphertexts (3/4) in insertion batch 1 are inserted between c_1 and c_2, while only a small portion of ciphertexts (1/4) in insertion batch 2 fall between the two ciphertexts. From the attacker's view, *if all plaintexts in the setup batch are the same*, then it has two *contradictory* conclusions:

- Insertion batch 1 indicates a **significant** difference between the random components of c_1 and c_2, resulting in most new ciphertexts being inserted between them.

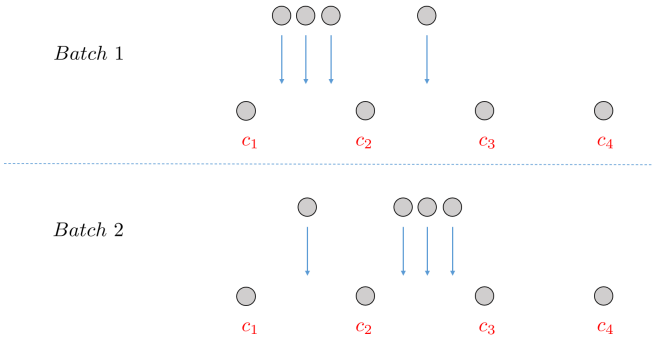

Figure 3. An example in POPE.

- Insertion batch 2 suggests a **small** difference between the random components of c_1 and c_2, leading to most new ciphertexts not being inserted between them.

The most plausible explanation for the contradictory results is there are distinct plaintexts in the setup batch, i.e., $\mathsf{S.Decrypt}(sk, c_1) \neq \mathsf{S.Decrypt}(sk, c_4)$. Clearly, the results reveal some frequency leakage. To further extract the leakage, the attacker can selectively observe subsets of the setup batch. For instance, by first assuming $\{c_1, c_2, c_3\}$ have the same plaintext and then trying to deduce contradictory results, the attacker also can infer that $\mathsf{S.Decrypt}(sk, c_1) \neq \mathsf{S.Decrypt}(sk, c_3)$ holds w.h.p.

We also give an example where all plaintexts are distinct. Consider an example with setup batch $\{1\}$ and one insertion batch $\{4, 2, 6, 7, 5, 9\}$. Denote the ciphertext of $\{1\}$ as $\{c_1\}$. All ciphertexts of the insertion batch are inserted larger than c_1, which is also unusual: if all the seven ciphertexts had the same underlying plaintext, this insertion pattern would happen with a small probability of only 1/7. Thus, the attacker can deduce the minimal ciphertext and maximal ciphertext of the seven ciphertexts have distinct underlying plaintexts w.h.p. Now we formally explain the leakage with our observation about a simple case.

A simple case. POPE preserves semantically secure ciphertexts and the orders of their underlying plaintexts. For two semantically secure ciphertexts c_1 and c_2, we use $c_1 < c_2$ to denote the order of c_1's underlying plaintext is smaller than that of c_2's underlying plaintext. Then we focus on a simple insertion case in POPE. Given three semantically secure ciphertexts $c_1 < c_2 < c_3$, a new ciphertext c is inserted and satisfies $c_1 < c < c_3$, then what is the probability of $c < c_2$? We define a variable X to calculate the probability:

$$X = \begin{cases} 1, & c < c_2, \\ 0, & c > c_2. \end{cases}$$

Observation 3. *In POPE, suppose $\{c_1, c_2, c_3\}$ have the same underlying plaintext v:*

$$c_i = \mathsf{S.Encrypt}(sk, v + r_i), i = 1, 2, 3$$

where $0 < r_1 < r_2 < r_3 < 1$, then X follows a Bernoulli distribution:

$$X \sim \mathsf{Ber}(\frac{r_2 - r_1}{r_3 - r_1}).$$

The proof is available in the full version [11]. This observation can be extended to a more complex and general case in POPE:

With n' ciphertexts inserted between c_1 and c_3, denote the number of ciphertexts smaller than c_2 as Y, if $c_i = \mathsf{S.Encrypt}(sk, v + r_i)$ $(i = 1, 2, 3)$, then $Y \sim \mathsf{Bin}(n', \frac{r_2 - r_1}{r_3 - r_2})$ where Bin denotes the binomial distribution. This is because each ciphertext inserted can be regarded as one independent instance in the simple case, i.e., a trial has probability $\frac{r_2 - r_1}{r_3 - r_2}$ of success. Furthermore, as the multi-snapshot attacker can have multiple insertion batches, we assume μ batches of ciphertexts are inserted between c_1 and c_3. In the jth insertion batch, denote the number of inserted ciphertexts as $n^{(j)}$ and the number of ciphertexts smaller than c_2 as Y_j. If $c_i = \mathsf{S.Encrypt}(sk, v + r_i)$ $(i = 1, 2, 3)$, then it holds

$$\forall j \in [\mu], Y_j \sim \mathsf{Bin}(n^{(j)}, \frac{r_2 - r_1}{r_3 - r_1}).$$

Fisher exact test. The extension above provides a method for the multi-snapshot attacker to verify if $\{c_1, c_2, c_3\}$ have the same underlying plaintext: It records the value of Y_j and $n^{(j)}$ in each insertion batch and verifies if the binomial distributions that Y_js follow have the same success probability $\frac{r_2 - r_1}{r_3 - r_1}$. Here we apply a statistical test *Fisher exact test* [16] to complete the verification. Suppose there are two samples under binomial distributions:

$$s_1 \sim \mathsf{Bin}(N_1, p_1), s_2 \sim \mathsf{Bin}(N_2, p_2).$$

Fisher exact test takes (s_1, N_1) and (s_2, N_2) as inputs and returns the probability of $p_1 = p_2$. Denote the test as $\mathsf{Fisher}()$, the verification can be done as following:

(1) We use *pro* to estimate the probability of the binomial distributions Y_is follow have the same success probability. It is calculated as follows:

$$pro = \min\{\underset{\forall j_1, j_2 \in [\mu]}{\mathsf{Fisher}}(Y_{j_1}, n^{(j_1)}, Y_{j_2}, n^{(j_2)})\}.$$

(2) Given a threshold $1/\gamma$, if $pro > 1/\gamma$, we say $\{c_1, c_2, c_3\}$ pass the verification, i.e., they are thought to have the same underlying plaintext, otherwise, we say they fail to the verification and have distinct underlying plaintexts.

5.2.2 Leakage in the scheme in VLDB '21. We first use examples to introduce the frequency leakage in the FH-OPE scheme in VLDB '21. Consider the following example:

(1) *Setup batch.* Suppose $\{7, 8, 7, 8\}$ are encrypted. Let the resulting **ordered** ciphertexts be $\{c_1, c_2, c_3, c_4\}$ corresponding to the ordered plaintexts $\{7, 7, 8, 8\}$.
(2) *Insertion batch 1.* Plaintexts $\{7, 7, 7, 8\}$ are encrypted and inserted. We simply introduce the insertion process with the first inserted plaintext. The ciphertext of the first plaintext 7 is inserted with a random order $rd \leftarrow\$ [0, 2]$. If $rd = 0$, it is inserted smaller than c_1. If $rd = 1$, it is inserted between c_1 and c_2. If $rd = 2$, it is inserted between c_2 and c_3.

Suppose the insertion pattern of the ciphertexts in insertion batch 1 aligns with that illustrated inFigure 4. The attacker can observe these ciphertexts are inserted in a non-uniformly manner, with all of them inserted smaller than c_3. However, if all ciphertexts have the same underlying plaintext, the insertion of ciphertexts should be nearly uniform due to uniformly sampled random orders. The uniformity implies there is at least one ciphertext inserted larger than c_3 w.h.p. Therefore, the attacker encounters contradictory results and deduces the existence of distinct plaintexts in the setup batch, i.e., $\mathsf{S.Dec}(c_1) \neq \mathsf{S.Dec}(c_4)$. Also, the inserted ciphertext smaller than c_1 has distinct underlying plaintexts with c_4.

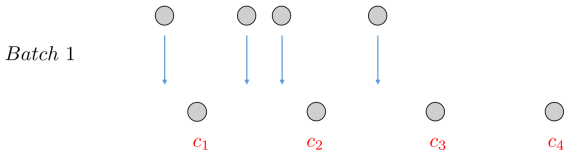

Figure 4. An example in the FH-OPE scheme in VLDB '21.

We also give an example where all plaintexts are distinct. Consider the example with the setup batch $\{1\}$ and the insertion batch $\{4, 2, 6, 7, 5, 9\}$. Denote the ciphertext of $\{1\}$ as $\{c_1\}$. In this case, all ciphertexts of the insertion batch are larger than c_1, which is non-uniform and unusual: if all the seven ciphertexts of the two batches have the same underlying plaintext, this insertion pattern happens with a small probability of only 1/7. Therefore, the attacker can deduce the minimal ciphertext and maximal ciphertext among the seven ciphertexts have distinct underlying plaintexts w.h.p. This example can also be extended by considering with the setup batch $\{1, 4, 2\}$ and the insertion batch $\{6, 7, 5, 9\}$, where the insertion pattern remains non-uniform and unusual, and thus leaks frequency. Now we formally describe the leakage and explain it with an observation about the same simple case defined in§ 5.2.1.

Observation 4. *In the FH-OPE scheme in VLDB '21, denote the initial order (index) of c_i as rd_i $(i = 1, 2, 3)$, if $\{c_1, c_2, c_3\}$ have the same underlying plaintext v:*

$$v = \mathsf{S.Decrypt}(sk, c_i), i = 1, 2, 3,$$

then X follows a Bernoulli distribution:

$$X \sim \mathsf{Ber}(\frac{rd_2 - rd_1}{rd_3 - rd_1}).$$

The proof is available in the full version [11].Observation 4 cannot be directly extended to more complex cases because the insertion of c changes both the orders of ciphertexts and the conditional probability. For example, with c inserted between c_1 and c_2, the orders of c_2 and c_3 are updated to rd_2+1 and rd_3+1, respectively. So for a newly inserted ciphertext c', we have

$$\Pr(c' < c_2 | c_1 < c' < c_3) = \frac{rd_2 - rd_1 + 1}{rd_3 - rd_1 + 1}.$$

However, it's still possible to apply binomial distributions to verify if ciphertexts in this scheme have the same underlying plaintext. Suppose there are n' ciphertexts to be inserted between c_1 and c_3. For each inserted ciphertext c_t $(\forall t \in [n'])$, it follows a Bernoulli distribution $\mathsf{Ber}(p'_t)$ and

$$\frac{rd_2 - rd_1}{rd_3 - rd_1 + n'} < p'_t < \frac{rd_2 - rd_1 + n'}{rd_3 - rd_1 + n'}.$$

The lower and upper bound indicates the extreme cases where all inserted ciphertexts are larger (smaller) than c_2. We set $rd_3 - rd_2 = rd_2 - rd_1$ and denote the number of inserted ciphertexts smaller than c_2 as Y. With $n' \ll rd_3 - rd_2$, p'_ts can be estimated with 1/2. So it is

Algorithm 2: Fisher exact test attack

Input: Setup batch $C^0 = \{c_1^0, ..., c_{n_0}^0\}$,
Attacking parameters (α, γ),
μ insertion batches $C^j = \{c_1^j, ..., c_{n_j}^j\}$ $(\forall j \in [\mu])$

Output: An order vector $\boldsymbol{\pi'} = (\pi'_1, ..., \pi'_{N'})$

1 $a = 1, I = [1, n_0]$
2 **for** $i = \alpha + 1 \rightarrow n_0 - \alpha$ **do** `sliding window`
3 **for** $j = 1 \rightarrow \mu$ **do** `get samples`
4 $Y_j = 0, n^{(j)} = 0$
5 **for** $k = 1 \rightarrow n_j$ **do**
6 **if** $c_{i-\alpha}^0 < c_k^j < c_i^0$ **then** Y_j += 1;
7 **if** $c_{i-\alpha}^0 < c_k^j < c_{i+\alpha}^0$ **then** $n^{(j)}$ += 1;
8 **end**
9 **end**
10 $pro_i = \min\{\underset{\forall j_1, j_2 \in [\mu]}{\text{Fisher}}(Y_{j_1}, n^{(j_1)}, Y_{j_2}, n^{(j_2)})\}$
11 **if** $pro_i < 1/\gamma$ **then**
12 **if** $I \cap [i-\alpha, i+\alpha] \neq \emptyset$ **then** `intersect`
13 $I = I \cap [i-\alpha, i+\alpha]$
14 **else**
15 $order = \underset{\forall \theta \in I}{\arg\min}\, pro_\theta$
16 $\pi'_a = order,\ a$ += 1
17 $I = [i-\alpha, i+\alpha]$
18 **end**
19 **end**
20 **end**

practical to verify if $\{c_1, c_2, c_3\}$ have the same underlying plaintext by testing if Y follows the binomial distribution $\text{Bin}(n', 1/2)$.

Binomial test. *Binomial test* [3] is a statistical test. It can be used to calculate the probability that a sample follows a given binomial distribution. In our verification, it takes the sample value of Y and $(n', 1/2)$ as inputs and returns a probability:

$$pro = \text{BinTest}(Y, n', 1/2)$$

where pro is the probability of $Y \sim \text{Bin}(n', 1/2)$. We also give a threshold of $1/\gamma$ for the verification. If $pro > 1/\gamma$, we say $\{c_1, c_2, c_3\}$ pass the verification, i.e., they are thought to have the same underlying plaintext, otherwise, we say they fail the verification and have distinct underlying plaintexts.

5.3 Attacks.

In this section, we present frequency-revealing attacks named *Fisher exact test attack* and *binomial test attack* against POPE and the FH-OPE scheme in VLDB '21, respectively.

Sliding window. A multi-snapshot attacker is assumed to have a setup batch of ciphertexts $C^0 = \{c_1^0, ...c_{n_0}^0\}$ and μ insertion batches of ciphertexts. As discussed in§ 5.2, given any three ciphertexts in C^0, we can verify whether they have the same underlying plaintext by observing the orders of ciphertexts in the insertion batches.

Algorithm 3: Binomial test attack with one insertion batch

Input: Setup batch $C^0 = \{c_1^0, ..., c_{n_0}^0\}, (\alpha, \gamma)$,
Insertion ciphertexts $C^1 = \{c_1^1, ..., c_{n_1}^1\}$

Output: An order vector $\boldsymbol{\pi'} = (\pi'_1, ..., \pi'_{N'})$

1 $a = 1, I = [1, n_0]$
2 **for** $i = \alpha + 1 \rightarrow n_0 - \alpha$ **do** `sliding window`
3 $Y = 0, n' = 0$
4 **for** $k = 1 \rightarrow n_1$ **do** `get sample`
5 **if** $c_{i-\alpha}^0 < c_k^1 < c_i^0$ **then** Y += 1;
6 **if** $c_{i-\alpha}^0 < c_k^1 < c_{i+\alpha}^0$ **then** n' += 1;
7 **end**
8 $pro_i = \text{BinTest}(Y, n', 0.5)$
9 **if** $pro_i < 1/\gamma$ **then**
10 **if** $I \cap [i-\alpha, i+\alpha] \neq \emptyset$ **then** `intersect`
11 $I = I \cap [i-\alpha, i+\alpha]$
12 **else**
13 $order = \underset{\forall \theta \in I}{\arg\min}\, pro_\theta$
14 $\pi'_a = order,\ a$ += 1
15 $I = [i-\alpha, i+\alpha]$
16 **end**
17 **end**
18 **end**

To recover the plaintext frequency in C^0, we use a sliding window approach. For each three-tuple $\{c_{i-\alpha}^0, c_i^0, c_{i+\alpha}^0\}$, we verify whether ciphertexts from $c_{i-\alpha}^0$ to $c_{i+\alpha}^0$ have the same underlying plaintext. We record the order interval $[i-\alpha, i+\alpha]$ if the tuple fails to the verification. If there are any overlapping intervals recorded, we intersect them to obtain a shorter interval as the final interval denoted by I. Using the intersection interval I, we estimate the maximal order π of distinct plaintext with the order whose corresponding verification probability pro is the smallest in I.

Fisher exact test attack. Algorithm 2describes the Fisher exact test attack against POPE. It takes the setup batch C^0, attacking parameters (α,γ) and μ insertion batches of ciphertexts C^js as inputs. It returns an order vector $\boldsymbol{\pi'}$ with length N', which reveals plaintext frequency in C^0. Based on the Fisher exact test which requires two samples, it requires there are at least 2 insertion batches, i.e., $\mu \geq 2$. The calculation in this algorithm consists of two parts: 1) the calculation of Y_js and $n^{(j)}s$ (line 3-9); 2) Fisher exact test (line 10). The former costs $O(n_0 \cdot \sum_{j=1}^{\mu} n_j)$ computation while the latter requires no more than $O(n_0 \cdot \mu^2)$ times Fisher exact test.

Binomial test attack. Differing from Fisher exact test attack, the binomial test attack can succeed with only one insertion batch because the binomial test requires only one sample. We describe the binomial test attack with one insertion batch inAlgorithm 3. It takes the setup batch C^0, attacking parameter (α, γ) and one insertion batch of ciphertexts C^1. The multi-snapshot attacker can repeat the algorithm with more insertion batches to recover more plaintext frequency, e.g., it may regard $C^0 \cup C^1$ as the new setup batch and C^2 as the new insertion batch to repeat the algorithm.

6 EXPERIMENTS

6.1 Experimental Setup

Datasets. We use the following datasets:

- *Births* [1]. This is the US birth data during 2004-2014. We initially encrypt birthdays in 2014 and then insert births in 2004-2013 yearly (∼ 45664K births for 366 different birthdays).
- *PBN* [1]. This is the US popular baby name data during 1971-2020. We initially encrypt names during 2011-2020 and then insert records according to decades (∼ 53239K births for 101 different birthdays)
- *Apls* [2]. This is the age and gender data of insurance applications during 2016-2020 in California. We separately use the age and gender attributes to encrypt records. We encrypt the records in 2020 and then insert other records according to years and the other attribute (∼ 1125K for 2 genders and 7 age groups).

Datasets consisting of birthdays, names, age, and gender are widely used in OPE [21, 25] and attacks against OPE [18, 20, 29].

Metrics. Our attacks represent the first successful attempts to recover plaintext frequency in FH-OPE schemes. They produce an order vector $\boldsymbol{\pi'} = (\pi'_1, ..., \pi'_{N'})$ as an estimate of the true order vector $\boldsymbol{\pi} = (\pi_1, ...\pi_N)$, which exactly describes the plaintext frequency. To evaluate the effectiveness of our attacks, we introduce new metrics that measure the similarity between $\boldsymbol{\pi'}$ and $\boldsymbol{\pi}$.

- *Estimation error* ϵ. For any $i \in [N]$, we call π_i is *recovered* if there exists a $j \in [N']$ such that $|\pi_i - \pi'_j| \leq \epsilon$. We set ϵ as a non-negative integer and its unit is 10.
- *Accuracy.* Denote the number of order π_i recovered by our attacks as N_r, we define the accuracy as N_r/N.
- *False positive rate* (FP). It represents the proportion of ciphertexts that have the same underlying plaintext but are thought to have distinct underlying plaintexts by our attacks. It is calculated as $(N' - N_r)/N$.

ϵ serves as an indicator of the estimation error level in our attacks. It is set small to promise a very small relative estimation error. Clearly, a great frequency-revealing attack should keep *both a high accuracy and a low FP under a small* ϵ. In the experiments, ϵ in density attack is the estimation error for revealing frequency in C. For Fisher exact test attack and binomial test attack, ϵ is the estimation error for revealing frequency in C^0, which reflects the estimation error for revealing frequency in C [11].

Configuration. All experiments are conducted on a machine with 48 2.5GHz vCPU and 128GB memory running Ubuntu Server 20.04.3. We implement the three FH-OPE schemes and our attacks in Python 3.8.12. The *Fisher exact test* and *binomial test* functions are token from Python scipy.stats library. We encrypt each dataset with the three FH-OPE schemes. For Kerschbaum's FH-OPE scheme, the ciphertext space for OPE ciphertexts is set as 120 bits, which is larger than that in [21]. The depth of the search tree in Apls, PBN and Births are 44, 52 and 57, respectively. In some experiments, we conduct this scheme on some subsets of these datasets, and the minimal tree depth on the subsets is 32 (the corresponding subset size is ∼ 196K).

6.2 Choosing Parameters for Attacks

Range and effect. There are two important attacking parameters in our attacks (α, γ). We have no predefined numerical ranges for α and γ, except that they should be positive integers. This is because there is no direct mathematical relationship between them and the accuracy and FP of our attacks. Their effects on our attacks can be briefly explained below:

- α determines the number of ciphertexts in each test in our attacks. It controls the impact of randomness in FH-OPE on our attacks. When α is small, a larger proportion of randomness affects the test results, resulting in a higher FP. When α is large, it leads to a coarser granularity in selecting ciphertexts for the tests, causing the loss of some frequency and resulting in lower accuracy.
- γ determines the threshold for the tests and the confidence for the test results. A larger γ assigns a stricter threshold and higher confidence, which can lead to lower accuracy and FP. Conversely, a smaller γ allows a more relaxed threshold and lower confidence, which results in higher accuracy but also a higher FP.

We conclude that loose parameter values (i.e., small α and γ) can achieve high accuracy but result in an unacceptable FP (e.g., $> 100\%$). Conversely, choosing excessively strict values sacrifices accuracy to achieve a low FP.

Setting values. We develop a simple and effective method to find suitable parameter values that achieve both high accuracy ($> 90\%$) and low FP ($< 10\%$). The method begins by initializing the parameters (α, γ) with small values like $(1, 1)$. We then gradually increase these values until the results outputted by our attacks stabilize. This approach is effective because the results consist of two types: 1) "incorrect results" caused by ciphertexts failing the tests due to randomness, and 2) "correct results" caused by ciphertexts failing the tests due to having distinct underlying plaintexts. When we increase α and γ, the impact of randomness is smaller and the threshold for the tests is stricter, allowing the ciphertexts that previously failed the tests due to randomness to pass the tests much more easily. However, the ciphertexts failing due to having distinct underlying plaintexts are **less sensitive** to the increase in α and γ. These enable us to filter out a majority of the incorrect results while preserving most of the correct results. The stability of outputted results under the increasing parameter values indicates that the remaining results are mostly correct, and thus the impact of further increasing α and γ becomes less significant.

Although the minimal initial values of (α, γ) can be $(1, 1)$, there are some optimizations for the initial values and values increasing presented in the full version [11] so we can efficiently find suitable values. Besides, we also provide more experimental results in the full version [11] to illustrate the effect and parameter selection process. Here we show the results of the parameter selection process for the density attack on the Births dataset inFigure 5. We fix one parameter with a loose value (e.g., 10) and observe the change in the number of indexes with the other parameter being stricter. The figure shows that the number of indexes becomes stable when α and γ are larger than 14. Therefore, we choose $(\alpha, \gamma) = (15, 15)$ for the density attack on Births. We use a similar approach to select parameter values for all of our attacks on Births and other datasets.

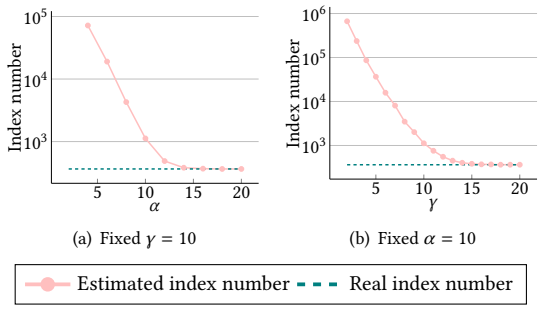

Figure 5. Strict values of α and γ make the estimated index number (N') stable and close to the real index number (N).

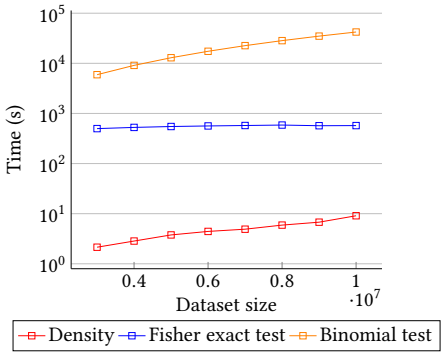

Figure 6. Attacking time of frequency-revealing attacks.

Table 1. Accuracy of frequency-revealing attacks on the Births dataset under different μ ($\epsilon = 100$). **Min** and **Ave** separately indicate the minimum and average accuracy on the four subsets.

#Batch (μ)	Fisher		Binomial		Density
	Min (%)	**Ave** (%)	**Min** (%)	**Ave** (%)	**Min** (%)
2	18.4	24.1	58.4	62.2	97.7
3	45.5	49.5	72.2	75.3	97.7
4	66.3	73.7	84.6	86.6	97.7
5	82.6	85.7	87.9	91.4	96.6
6	86.6	90.6	86.8	90.8	98.8
7	90.1	93.4	86.8	91.4	97.7
8	96.7	98.3	86.8	93.0	96.6
9	97.8	98.6	86.8	93.0	96.6
10	97.7	98.3	86.8	93.0	97.7
Overall ($\mu = 10$)	98.4	92.6	99.5		

6.3 Revealing plaintext frequency

Time usage. To demonstrate the efficiency of our attacks, we conduct experiments to measure the time required to execute each of the three attacks using a fixed number of records. We select a subset of Births consisting of 10^6 records in each batch and attack the setup batch with 2-10 insertion batches to observe the time usage under different dataset sizes. Our results, presented in Figure 6, show that the time usage of the density attack and Fisher exact test attack scales linearly with the dataset size. The density attack is the most efficient, taking no more than 10s to attack 10^7 records.

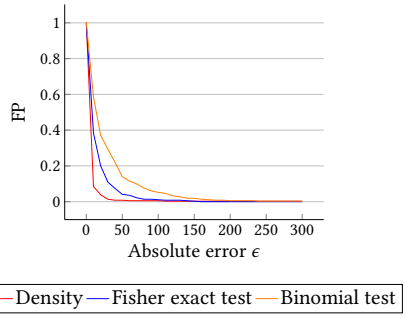

Figure 7. FP of frequency-revealing attacks on Births ($\mu = 10$).

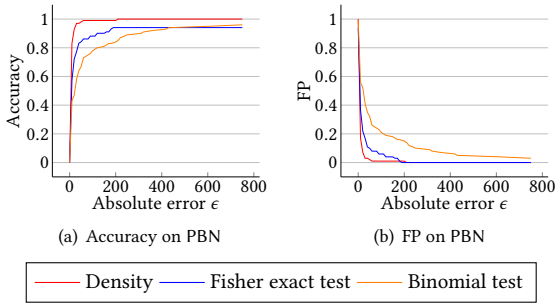

Figure 8. Performance of frequency-revealing attacks on PBN.

On the other hand, the binomial test attack is the most expensive, as it is designed for one insertion batch, and when applied with μ insertion batches, it has to be repeated μ times. However, its cost is still acceptable. To attack a dataset with 10^7 records, it takes no more than 5×10^4s, which is approximately 14 hours.

Parallelism. All of our attacks operate by traversing ordered ciphertexts in ascending order. As a result, they can be run in parallel by driving the ciphertexts of the entire dataset into multiple subsets based on their order. Once the attacks on the ciphertext subsets are completed, we combine their outputs to obtain the frequency of the entire dataset. This approach allows us to leverage the computational resources available in a parallel processing environment and speed up the attack process. Combing results on subsets needs some checking work. For example, it is unknown if the maximal ciphertext in the first subset and the minimal ciphertext in the second subset have the same underlying plaintext. Therefore, suppose in the first subset our attacks **think** the largest i_1 ciphertexts have the same underlying plaintext and in the second subset the smallest i_2 ciphertexts have the same underlying plaintext. Then we conduct our attacks on the $i_1 + i_2$ ciphertexts to reveal possible frequency information between these ciphertexts. In this way, the works conducted in parallelism are identical to those in one process.

Performance with varying μ. We evaluate the performance of our frequency-revealing attacks with different numbers of insertion batches (μ) on the Births dataset. Our attacks apply 2-10 insertion batches and the results are shown in Table 1. To show the stability of our attacks, we partition the dataset into four nearly equal subsets and attack each one separately. The results on subsets are presented in rows 2-10. We also attack the overall dataset. We conduct the

Table 2. The recovery rate of combined attack.

Data	Distinct (N)	Density (%)	Fisher (%)	Binomial (%)
Gender	2	99.99	99.19	99.84
Age	7	99.98	98.40	99.85
January	31	99.99	99.82	99.80

attacks in parallel by encrypting the overall dataset, dividing the ciphertexts into four equal subsets, attacking each ciphertext subset, and combing attack results with some checking processes. The last row shows the attack accuracy on the overall dataset.

We observe that the accuracy of both the Fisher exact test and binomial test attacks generally increases with more insertion batches. When $\mu > 6$, both attacks achieve more than 90% accuracy in most cases. In contrast, the accuracy of the density attack is independent of μ, and it consistently achieves an accuracy of no less than 96.6%. We also show FP on Births inFigure 7. With $\mu = 10$ and $\epsilon = 100$, we observe that the FP is no more than 5%. Especially, the density attack and Fisher exact test attack achieve a FP of 0%.

Performance with varying ϵ. We evaluate our attacks under different values of ϵ on the PBN dataset. We attack this dataset with 4 insertion batches and plot the experimental results inFigure 8. As ϵ increases, the accuracy of our attacks increases, and the FP decreases. This indicates that our attacks are successful in identifying the majority of indexes in $\boldsymbol{\pi}$ with some estimation errors. When we set $\epsilon = 430$, all of our attacks achieve an accuracy of over 90% and a FP of no more than 5%. Considering that the minimum frequency is 17124 (i.e., $|\pi_{i+1} - \pi_i| \geq 17124$ for any $i \in [N-1]$), the value of $\epsilon = 430$ corresponds to a very small relative error of only 2.5% for estimating any π_i. Notably, the density attack and Fisher exact test attack achieve a FP of 0% when $\epsilon = 210$ which indicates a relative estimation error of only 1.2%.

6.4 Combing Inference Attacks

We have shown plaintext frequency in any existing FH-OPE scheme is recoverable. Next, we explore what information our attacks can provide to inference attacks. Recall in§ 2we introduce two types of inference attacks: the sorting attack and frequency-analyzing attacks. To conduct them, the sorting attack requires that the dataset is dense and the plaintext space $\mathbb{M}$ is known while frequency-analyzing attacks require an estimation of plaintext distribution from auxiliary public information.

First, our attacks show that sorting attacks can still be used in FH-OPE schemes. If plaintexts encrypted by FH-OPE schemes are dense and $\mathbb{M}$ is public, we can apply our frequency-revealing attacks to recover plaintext frequency and then use sorting attacks. We conducted a combined attack on the Apls and Births datasets, recovering the gender and age attributes of records for the former, and the birthdays of records in January for the latter. The experimental results, shown inTable 2, demonstrate that we are able to recover almost 100% of plaintexts protected by all existing FH-OPE schemes. Second, our attacks can be used to reveal real plaintext distribution and provide a starting point for frequency-analyzing attacks. We show the cumulative plaintext distribution that our attacks recovered, which is almost identical to the real plaintext distribution, inFigure 9. This finding suggests that frequency-analyzing attacks can potentially be applied to the results obtained from our attacks.

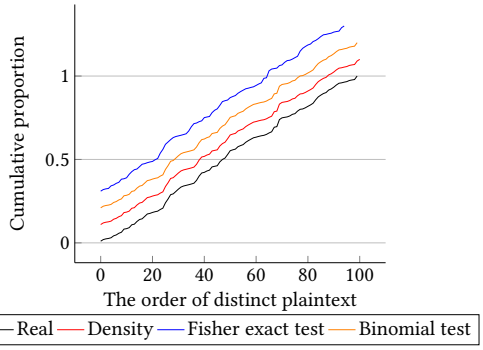

Figure 9. Cumulative plaintext distributions recovered by our attacks and real plaintext distribution on PBN. We give a vertical shift to each line for comparing them.

7 DISCUSSION

7.1 Secure Scenarios.

Our attacks show that all existing FH-OPE schemes are unable to prevent the frequency leakage. However, as FH-OPE is a clearly valuable direction of OPE, we discuss the scenarios where existing FH-OPE schemes may be secure, i.e., there is no leakage of plaintext frequency. Recall our attacks apply insertion patterns (the distribution of inserted ciphertexts) to recover frequency in POPE and the FH-OPE scheme in VLDB '21. We explore the requisitions for insertion patterns that are free from our attacks.

- *Stable.* In POPE, the proportion of ciphertexts inserted between any two ciphertexts has to be constant, e.g., if there are 10% ciphertexts in insertion batch 1 inserted between two ciphertexts c_1 and c_2, then there are still 10% ciphertexts inserted between them in any batch.
- *Uniform.* In the scheme in VLDB '21, if the difference between orders of two ciphertexts is d and there are total n ciphertexts, then the next inserted ciphertext is inserted between the two ciphertexts with the probability of $|d|/(n+1)$.

Both of the two insertion patterns above indicate that the plaintext distribution must be identical in each insertion batch. Moreover, the multi-snapshot attacker can arbitrarily determine the setup and insertion batches since it can access the server at any point during the scheme execution. So the plaintext distribution has to be identical *in each individual insertion*. Therefore, to defend against our attacks, each plaintext in the database has to be independently sampled from the same distribution. This strict condition is only feasible in *very limited real-world scenarios*. Notably, Kerschbaum's FH-OPE scheme still cannot protect frequency even under this strict condition mentioned above because the ciphertext distribution in this scheme is still non-uniform and leak plaintext frequency.

Limited insertion. Although the security model in FH-OPE leaks the exact insertion order of each ciphertext, it is practical to consider the scenario where the multi-snapshot attacker observes only limited insertions in a large database. Unfortunately, we discovered that all three FH-OPE schemes can still leak plaintext frequency in such scenarios. A common and dangerous case is when new distinct plaintexts are inserted into the database. For instance, if we consider an encrypted database with ciphertexts of plaintexts 7 and

9, each having a frequency of 10^7, and we insert 10 ciphertexts of value 8, they will all be inserted between the same two ciphertexts: the maximum ciphertext of 7 and the minimum ciphertext of 9. This insertion pattern is highly specific, as there are a total of $2 \times 10^7 + 1$ positions available for each insertion, yet only one specific position is chosen for all the newly inserted ciphertexts. As a result, the attacker can infer the insertion of new distinct plaintexts based on this distinctive insertion pattern.

7.2 Enhanced Security.

Fake queries. *Fake queries* [17, 27] is a technique in encrypted databases to hide range and access query distributions. It potentially can be used to indeed achieve frequency-hiding in OPE. Denote the (dynamic) real query distribution as F_0. Then the client may hope to hide F_0 by making the distribution observed by the attacker be F. For example, Grubbs et al. hide the real access distribution by making the attacker always observe a uniform access query distribution. To achieve that, the client calculates a query distribution F_1 such that $F = F_0 + F_1$. Then the client performs both real queries sampled from F_0 and fake queries sampled from F_1 at the same time to hide F_0. This idea can be used in FH-OPE: to hide the real insertion patterns, the client can perform both real insertions and fake insertions, making the total insertion patterns seem uniform and cannot be used by our attacks.

Differential privacy. We introduce the notable work by Chowdhury et al. [33], which integrates FH-OPE with *differential privacy* (DP). They relax the order-preserving property to improve the security. Roughly speaking, given two plaintexts v_1 and v_2 ($v_1 < v_2$), the probability that the ciphertext of v_1 is smaller than the ciphertext of v_2 is less than 1. This probability is calculated using the DP algorithms, and it increases as $v_2 - v_1$ becomes larger. In this way, the adversary cannot distinguish the ciphertexts of v_1 and v_2 when $v_2 - v_1$ is small enough. For example, for two age values v_1, v_2, under a suitable parameter value for DP, this work guarantees the ciphertexts of v_1 and v_2 are indistinguishable when $v_2 - v_1 \leq 8$.

We use a simple example to illustrate the intuition of the work and its relation with our attacks. Consider a dataset consisting of plaintexts $\{1, 2, 3\}$. There are three plaintext groups $\{o_1, o_2, o_3\}$ for dividing the plaintexts. For any i and j in $\{1, 2, 3\}$, we use $p_{i,j}$ to denote the probability of plaintext i falling into the group o_j. The probabilities depend on the difference $|i - j|$, e.g., plaintext 1 may fall into $\{o_1, o_2, o_3\}$ with probabilities $\{70\%, 20\%, 10\%\}$. To encrypt this dataset with FH-OPE, there are two steps:

(1) *Division.* For each plaintext v, the client determines the group that v falls into. If v falls into group o_j, the client treats v as plaintext j in the encryption algorithm of FH-OPE even if $v \neq j$.
(2) *Encryption.* The client uses FH-OPE to encrypt plaintexts in the dataset and gets three ciphertext groups denoted as $\{cg_1, cg_2, cg_3\}$ corresponding to the ciphertext set of $\{o_1, o_2, o_3\}$. Note FH-OPE guarantees the ciphertexts in the same ciphertext group are distinct to hide the frequency of each ciphertext group.

As shown above, the work adds a division step to mix plaintexts but the encryption step is identical to the encryption process in FH-OPE. Therefore, our frequency-revealing attacks still work against the encryption step if one of the three FH-OPE schemes is applied. This implies our attacks can reveal the three ciphertext groups $\{cg_1, cg_2, cg_3\}$ but cannot ensure the exact underlying plaintext of ciphertexts in each ciphertext group. In short, our attacks have no impact on the security provided by DP but still break the security provided by FH-OPE. So our attacks potentially reduce the security level of this work to that of a combination of deterministic OPE and DP. But the inclusion of DP indeed limits the accuracy of inference attacks on individual ciphertexts and enhances the security of OPE.

8 RELATED WORK

Maffei et al. [26] analyze the security of Kerschbaum's FH-OPE scheme. They present an attack that shows an adversary in the scheme may win the IND-FAOCPA security game with non-negligible probability. They are the first to claim Kerschbaum's FH-OPE cannot hide plaintext frequency. However, their attack cannot recover plaintext frequency, which makes the security of the scheme still unclear. At the same time, Bogatov et al. [8] point out that Kerschbaum's FH-OPE scheme leaks insertion orders of ciphertexts even under a single-snapshot attacker. They claim some attacks based on insertion orders may exist in the future.

The best known published attack against FH-OPE schemes is the *binomial attack* [18] proposed by Grubbs et al. It uses only plaintext order to recover plaintexts and thus can be applied to any OPE scheme. However, it requires auxiliary public information about plaintexts but our attacks do not. Besides, it performs poorly in real-world datasets [18, 19]. There are also some attacks [15, 20, 23, 24] based on the leakages of range queries. These attacks are effective to any generic encrypted database including those using OPE [5, 31, 35] and FH-OPE [21, 25, 32]. However, these attacks require plenty of range queries and assume the persistent attacker to get the leakages of queries.

Comparison. Firstly, our density attack is the first to recover plaintext frequency in Kerschbaum's FH-OPE scheme and does not require any auxiliary information. Our work clarifies that Kerschbaum's FH-OPE scheme provides little protection for plaintext frequency even under a single-snapshot attacker. Secondly, all of our attacks assume a snapshot attacker. We limit the number of snapshots obtained by a multi-snapshot attacker to 11, making our attackers much weaker than the persistent attacker.

9 CONCLUSION

In this paper, we provide a comprehensive analysis of the security of all existing FH-OPE schemes. Our observations lead to the conclusion that these schemes leak plaintext frequency, which motivated us to present three frequency-revealing attacks against them. Our work demonstrates that avoiding the leakages of non-uniform ciphertext distribution and ciphertext insertion orders is essential for achieving frequency-hiding.

ACKNOWLEDGMENTS

The authors thank Jingyu Li and the anonymous reviewers at VLDB 2023 for their helpful comments and suggestions. The work was supported in part by National Natural Science Foundation of China (Grant No. 62002319, U20A20222), Hangzhou Leading Innovation and Entrepreneurship Team (TD2020003), and Zhejiang Key R&D Plan (Grant No. 2021C01116).

REFERENCES

[1][n.d.]. https://www.ssa.gov/. Accessed in July 2023.

[2][n.d.]. https://www.osi.ca.gov/CalHEERS.html. Accessed in July 2023.

[3] 2008. *Binomial Test.* Springer New York, New York, NY, 47–49. https://doi.org/10.1007/978-0-387-32833-1_36

[4] Ghous Amjad, Seny Kamara, and Tarik Moataz. 2018. Breach-resistant structured encryption. *Cryptology ePrint Archive* (2018).

[5] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase. In *CIDR*.

[6] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. 2018. The Tao of Inference in Privacy-Protected Databases. *Proc. VLDB Endow.* 11, 11 (jul 2018), 1715–1728. https://doi.org/10.14778/3236187.3236217

[7] Dmytro Bogatov, George Kollios, and Leonid Reyzin. 2019. A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols. *Proc. VLDB Endow.* 12, 8 (April 2019), 933–947. https://doi.org/10.14778/3324301.3324309

[8] Dmytro Bogatov, George Kollios, and Leonid Reyzin. 2019. A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols. *Proc. VLDB Endow.* 12, 8 (apr 2019), 933–947. https://doi.org/10.14778/3324301.3324309

[9] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. 2009. Order-Preserving Symmetric Encryption. In *Advances in Cryptology - EUROCRYPT 2009*, Antoine Joux (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 224–241.

[10] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. 578–595. https://doi.org/10.1007/978-3-642-22792-9_33

[11] Xinle Cao, Jian Liu, Yongsheng Shen, Xiaohua Ye, and Kui Ren. 2023. Frequency-revealing attacks against Frequency-hiding Order-preserving Encryption. Cryptology ePrint Archive, Paper 2023/1122. https://eprint.iacr.org/2023/1122 https://eprint.iacr.org/2023/1122.

[12] Zhihao Chen, Qingqing Li, Xiaodong Qi, Zhao Zhang, Cheqing Jin, and Aoying Zhou. 2022. BlockOPE: Efficient Order-Preserving Encryption for Permissioned Blockchain. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1245–1258. https://doi.org/10.1109/ICDE53745.2022.00098

[13] Yang Du, Daniel Genkin, and Paul Grubbs. 2022. Snapshot-Oblivious RAMs: Sub-Logarithmic Efficiency Fornbsp;Short Transcripts. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 152–181. https://doi.org/10.1007/978-3-031-15985-5_6

[14] F. Betül Durak, Thomas M. DuBuisson, and David Cash. 2016. What Else is Revealed by Order-Revealing Encryption?. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1155–1166. https://doi.org/10.1145/2976749.2978379

[15] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. 2020. Full Database Reconstruction in Two Dimensions. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 443–460. https://doi.org/10.1145/3372297.3417275

[16] Ronald Aylmer Fisher. 1954. *Statistical methods for research workers; 20th ed.* Oliver and Boyd, Edinburgh. https://cds.cern.ch/record/724001

[17] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency smoothing for encrypted data stores. In *29th USENIX Security Symposium (USENIX Security 20)*. 2451–2468.

[18] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. 2017. Leakage-Abuse Attacks against Order-Revealing Encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 655–672. https://doi.org/10.1109/SP.2017.44

[19] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. 2018. Practical and Secure Substring Search. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 163–176. https://doi.org/10.1145/3183713.3183754

[20] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1329–1340. https://doi.org/10.1145/2976749.2978386

[21] Florian Kerschbaum. 2015. Frequency-Hiding Order-Preserving Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 656–667. https://doi.org/10.1145/2810103.2813629

[22] Florian Kerschbaum and Axel Schroepfer. 2014. Optimal Average-Complexity Ideal-Security Order-Preserving Encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 275–286. https://doi.org/10.1145/2660267.2660277

[23] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp40000.2020.00029

[24] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*. 297–314. https://doi.org/10.1109/SP.2018.00002

[25] Dongjie Li, Siyi Lv, Yanyu Huang, Yijing Liu, Tong Li, Zheli Liu, and Liang Guo. 2021. Frequency-Hiding Order-Preserving Encryption with Small Client Storage. *Proc. VLDB Endow.* 14, 13 (sep 2021), 3295–3307. https://doi.org/10.14778/3484224.3484228

[26] Matteo Maffei, M. Reinert, and Dominique Schröder. 2017. On the Security of Frequency-Hiding Order-Preserving Encryption. In *CANS*.

[27] Charalampos Mavroforakis, Nathan Chenette, Adam O'Neill, George Kollios, and Ran Canetti. 2015. Modular Order-Preserving Encryption, Revisited. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 763–777. https://doi.org/10.1145/2723372.2749455

[28] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 644–655. https://doi.org/10.1145/2810103.2813651

[29] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 644–655. https://doi.org/10.1145/2810103.2813651

[30] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich. 2013. An Ideal-Security Protocol for Order-Preserving Encoding. *2013 IEEE Symposium on Security and Privacy* (2013), 463–477.

[31] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566

[32] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. 2016. POPE: Partial Order Preserving Encoding. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2976749.2978345

[33] Amrita Roy Chowdhury, Bolin Ding, Somesh Jha, Weiran Liu, and Jingren Zhou. 2022. Strengthening Order Preserving Encryption with Differential Privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2519–2533. https://doi.org/10.1145/3548606.3560610

[34] Isamu Teranishi, Moti Yung, and Tal Malkin. 2014. Order-Preserving Encryption Secure Beyond One-Wayness. In *Advances in Cryptology – ASIACRYPT 2014*, Palash Sarkar and Tetsu Iwata (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 42–61.

[35] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proc. VLDB Endow.* 6, 5 (March 2013), 289–300. https://doi.org/10.14778/2535573.2488336