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

Xinle Cao
Zhejiang University
Hangzhou stateChina
xinle@zju.edu.cn

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

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

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

Kui Ren
Zhejiang University
Hangzhou stateChina
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. We 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 often ignored in OPE.

1 INTRODUCTION

Order-preserving encryption (OPE) [9–11, 21, 22, 30, 35] is a widely used technique in encrypted databases for performing range queries [5, 31, 36]. It preserves plaintext order in ciphertexts to allow efficient sorting and comparison operations over encrypted data. Moreover, it does not require any modification to existing database engines. 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, 17, 28]. For example, Grubbs et al. [17] propose a frequency-based inference attack that recovers over 80% of plaintexts protected by deterministic OPE schemes.

To this end, some *frequency-hiding* order-preserving encryption (FH-OPE) schemes [21, 25, 32] are proposed to protect plaintext frequency. The core idea is to randomize ciphertexts of repeated plaintexts such that each plaintext is encrypted to a unique ciphertext. For example, plaintexts $(0, 0, 0, 1)$ are encrypted to ciphertexts $(64, 32, 96, 112)$. There are three different ciphertexts of 0 and all of them are smaller than the ciphertext of 1.

Up to now, there have been three FH-OPE schemes proposed. 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 existing FH-OPE schemes is the *binomial attack*, which is based on only plaintext order. But it is considered an ineffective attack since it recovers at most 30% of plaintexts in [17] and 15% of plaintexts in [18]. As a result, FH-OPE schemes are still recommended for use in encrypted databases by various studies [17, 18, 25].

Our contribution. However, we found that 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. Our analysis and attacks exploit the leakages of non-uniform ciphertext distribution and ciphertext insertion orders, highlighting

*Jian Liu is the corresponding author.

the importance of avoiding these leakages to achieve frequency-hiding. Notably, the leakage of ciphertext insertion orders is often overlooked in OPE despite being crucial to the security of the schemes [21, 22].

We summarize our contributions as follows:

(1) We revisit the security of Kerschbaum's FH-OPE scheme and show that its non-uniform ciphertext distribution leaks plaintext frequency. We present a frequency-revealing attack named *density attack* that recovers frequency 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 based on 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).
(4) We discuss some possible scenarios and directions for improving the security of FH-OPE schemes and further analyze the impact of recent work [33] on FH-OPE, which tries to enhance the security of FH-OPE with *differential privacy* (§ 7).

2 PRELIMINARIES

Notation. For positive integer n, $[n]$ is the set $\{1, ..., n\}$, and $|I|$ is the cardinality (number of elements) of set I. $r \overset{\$}{\leftarrow} I$ 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.

2.1 OPE and inference attacks.

Order-preserving encryption (OPE). OPE preserves plaintext order in ciphertexts to achieve both data privacy and functionality in cloud computing. It is widely used in encrypted databases for range queries including CryptDB [31], Cipherbase [5] and Monomi [36].

A (stateful) OPE scheme OPE = (KeyGen, Encrypt, Decrypt) consists of the following 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 $c_1 = c_2$.

Most existing OPE schemes [9, 10, 30**?**] are designed to be deterministic. They generically leak both plaintext order and frequency. The leakages incur two kinds of frequency-based inference attacks: *sorting attack* and *frequency-analyzing attacks*.

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. Attackers recover 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, 17, 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.

These inference attacks show, with plaintext frequency revealed, an OPE scheme is insecure, e.g., in [29], the sorting attack recovers 100% dense plaintexts and a frequency-analyzing attack named *cumulative attack* recovers more than 80% low-density plaintexts.

3 THREAT MODEL

3.1 Security model

To describe the plaintext order in a non-deterministic OPE scheme, Kerschbaum presents the notion of *randomized order*.

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 $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_n\}$ ($\forall i.1 \le \gamma_i \le n, \forall i, j.i \ne j \Longrightarrow \gamma_i \ne \gamma_j$) of sequence V, it holds that*

$$\forall i, j.v_i > v_j \Longrightarrow \gamma_i > \gamma_j$$

and

$$\forall i, j.\gamma_i > \gamma_j \Longrightarrow v_i \ge v_j$$

The randomized order actually assigns order relations between repeated plaintexts in V. So a plaintext sequence can have multiple randomized orders. For example, let the plaintext sequence $V = \{1, 2, 2, 3\}$, then possible randomized orders are $\Gamma_1 = \{1, 2, 3, 4\}$ and $\Gamma_2 = \{1, 3, 2, 4\}$. Besides, multiple plaintext sequences can have the same randomized order even if they have different plaintext frequencies, e.g., plaintext sequences $V_1 = \{2, 2, 2, 3\}$, $V_2 = \{1, 2, 1, 3\}$, and $V_3 = \{1, 1, 1, 1\}$ have a common randomized order $\Gamma = \{1, 3, 2, 4\}$. Based on the randomized order, Kerschbaum presents the notion of *frequency-hiding* OPE (FH-OPE) with a formal security guarantee named *indistinguishability under frequency-analyzing ordered chosen-plaintext attack* (IND-FAOCPA) [21].

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[1]. He sends them to the challenger.

[1]Multiple common randomized orders are possible and allowed

(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., $\mathsf{st}_i, c'_i \leftarrow \Pi.\mathsf{Encrypt}(\mathsf{st}_{i-1}, v_i^b)$.
(3) The adversary $\mathcal{A}$ outputs a guess b^* of b, i.e. which of the two sequences it is.

Then the IND-FAOCPA is defined as below:

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 $Game_{\mathcal{A},\Pi}^{FAOCPA}(\lambda)$ is negligible in λ, i.e.*

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

IND-FAOCPA implies an OPE scheme leaks the randomized order of the plaintexts, i.e., two plaintext sequences with the same randomized order - but different plaintext frequency - should be indistinguishable [21]. Kerschbaum claims the randomized order does not contain any frequency information because each value (order) always occurs exactly once. Therefore, an OPE scheme achieving IND-FAOCPA secure is called a FH-OPE scheme.

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 sorted 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 sorted 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 [17, 28] observes one snapshot of the encrypted database. It can get only ciphertexts and their orders. i.e., sorted ciphertexts $\boldsymbol{c} = (c_1, ..., c_n)$. The multi-snapshot attacker [4, 12] has access to multiple snapshots of the encrypted database. Each snapshot is interspersed with a batch of (insertion) 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.*

The multi-snapshot attacker additionally observes the leakage of *partial insertion orders*, i.e., for two ciphertexts observed in two distinct moments, it knows which one is inserted earlier. This attacker captures an adversary who gets access to the encrypted database periodically [4, 12], e.g., a malicious employee steals a snapshot of the company's encrypted database each month.

Snapshot vs Persistent. The multi-snapshot attacker is much weaker than the persistent attacker. Its power depends on the number of snapshots it has access to. It can be as strong as the persistent attacker only if it gets a snapshot every time an operation happens. In our experiments, we assume the multi-snapshot attacker gets no more than 11 snapshots while there are millions of operations.

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

Algorithm 1: Encrypt in Kerschbaum's FH-OPE scheme

```
Input: $v, t, min, max$
Output: $c$
State: Binary search tree $T$ of nodes $\{t\}$ in client
if $T$ is empty then                                  initialize $T$
    $T.root = Node(v, min + \lceil \frac{max-min}{2} \rceil)$
    return $T.root.cipher$
end
if $v = t.plain$ then                   randomize repeated plaintexts
    $coin$ = RandomCoin() $\in \{0, 1\}$
else
    $coin = \perp$
end
if $v > t.plain$ or $coin = 1$ then
    if $t.right \neq$ NULL then
        return Encrypt($v, t.right, t.cipher, max$)
    else
        $t.right = Node(v, t.cipher + \lceil \frac{max-t.cipher}{2} \rceil)$
        return $t.right.cipher$
    end
end
if $v < t.plain$ or $coin = 0$ then
    if $t.left \neq$ NULL then
        return Encrypt($v, t.left, min, t.cipher$)
    else
        $t.left = Node(v, min + \lceil \frac{t.cipher-min}{2} \rceil)$
        return $t.left.cipher$
    end
end
```

$\boldsymbol{\phi}$ 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 revisit the security of Kerschbaum's FH-OPE scheme under the single-snapshot attacker. We make some observations about its data structure and ciphertext distribution. Based on them, we present a frequency-revealing attack named *density attack*. It recovers most plaintext frequency hidden by this scheme with only ciphertexts.

4.1 Review

Encryption. In Kerschbaum's FH-OPE scheme, the client maintains a binary search tree T to record mapping between plaintexts and OPE ciphertexts: each node in T corresponds to a plaintext and its ciphertext, and these nodes are sorted according to plaintext order. To encrypt a new plaintext v, the client finds its order by searching v in T, and then maps v to the ciphertext which is the mean value of

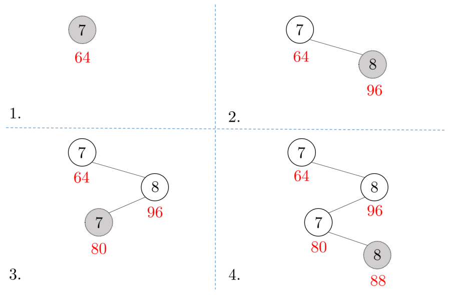

Figure 1. Let plaintext sequence be $V = \{7, 8, 7, 8\}$ and ciphertext space be $\mathbb{C} = [0, 128]$, then plaintexts are encrypted to $\{64, 96, 80, 88\}$ with RandomCoin() outputs 1 and 0 in subfigure 3 and 4 respectively.

the ciphertexts for the next smaller plaintext and the next greater plaintext, e.g., let plaintexts $\{0, 2\}$ be encrypted to $\{0, 100\}$, then a new plaintext 1 is encrypted to the ciphertext 50.

We formally describe the encryption in Algorithm 1. It is designed recursively to search v in T. We denote the binary search tree T as a set of nodes $\{t\}$ and each node consists of a plaintext (*plain*), a ciphertext (*cipher*) and pointers to its child nodes. The input t is for the current node searched on. The inputs *min* and *max* are the lower and upper limits for the ciphertext. Initially, the client calls the encryption function with plaintext v, the root of T, and the minimal and maximal values in ciphertext space $\mathbb{C}$. Throughout this paper, we assume $|\mathbb{C}|$ is a power of 2 for convenience in the calculation.

Frequency-hiding. In order 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. We give an example of Kerschbaum's FH-OPE scheme in Figure 1. We show the growth of T with four subfigures. There are two calls for RandomCoin(): When inserting 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 inserting 8 for the second time in subfigure 4, it outputs 0 so the first 8 is greater.

4.2 Observations

Kerschbaum's FH-OPE scheme ensures each plaintext is mapped to a unique ciphertext, preventing attackers from determining plaintext frequency through repeated ciphertexts. However, our observations regarding the distribution of the ciphertexts suggest that there may be some leakages about frequency. 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 $\mathbb{C}$ in Figure 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. 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. It recovers the plaintext frequency although it does not know the two distinct plaintexts are 7 and 8. In some cases, the closet neighboring ciphertexts may have the same underlying plaintext, 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 still finds most ciphertexts of 7.

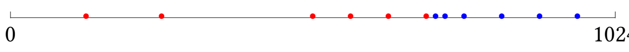

Figure 2. The ciphertext distribution in the example. The ciphertexts of 7 and 8 are represented by red and blue points, respectively.

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. We first introduce a basic notion used in the observations.

Definition 5 (Leaf nodes of v^i). *In the binary tree T, we call a node of plaintext v^i as a leaf node of v^i if there is no node of v^i in its subtrees.*

Observations overview. We present three observations that demonstrate how the ciphertext distribution of plaintext v^i leaks its frequency. First, we observe that the leaf nodes of v^i are expected to have almost the same number of ancestors corresponding to v^i, providing a useful insight into the structure of the tree. Furthermore, we describe how the depths of nodes of v^i are affected by nodes of other plaintexts, which reflects the depth distribution of leaf nodes of v^i. Finally, we connect the depth distribution to the ciphertext distribution, revealing the potential for leakages in the encryption scheme. By examining these observations, we gain a deeper understanding of the limitations of the FH-OPE scheme and the potential for attacks based on its ciphertext distribution. The following lemmas are used in the proof of our observations. We introduce and prove them here. They are also important properties of T.

Lemma 1. *For any node of v^i in T, it is expected that the number of nodes of v^i in its left and right subtree differs no more than 1.*

Proof. For any node of v^i in T, the order relations between it and nodes of v^i to be inserted into its subtrees are determined by calling RandomCoin(). It is expected that the number of 0 and 1 outputted by the function differs no more than 1, which concludes the lemma. □

Lemma 2. *Suppose there are h nodes of plaintext v^i in a path of T. Then, it is expected that the number of nodes of plaintext v^i in T lying on this path is at least 2^{h-1} and at most $2^h + 2^{h-1} - 2$.*

Proof. We assume that there is a node P in T corresponding to v^i, and it has a subtree with k nodes of v^i. According to Lemma 1, the other subtree of P is expected to have at least $k - 1$ and at most $k + 1$ nodes of v^i. To express the expected number of nodes of v^i in the subtree rooted at P, we define functions $f(k)$ and $g(k)$ as the lower and upper bounds, respectively. We can calculate $f(k)$ and $g(k)$ as follows:

$$f(k) = k + (k - 1) + 1 = 2k,$$

$$g(k) = k + (k + 1) + 1 = 2k + 2.$$

For a path with h nodes of v^i, we can recursively call f and g for $h - 1$ times by traversing the path from the second lowest node of v^i to the highest node of v^i and calling the functions at each node of v^i encountered. Therefore, the expected number of nodes of v^i in T can be expressed as

$$f^{h-1}(1) = 2^{h-1},$$

$$g^{h-1}(1) = 2^h + 2^{h-1} - 2$$

for the lower and upper bound, respectively. □

Observation 1. *It is expected that the number of ancestors with plaintext v^i for any two leaf nodes of v^i differs no more than* 1*.*

Proof. We prove this observation by showing that the number of nodes of v^i in any two paths is expected to differ no more than 1. Suppose there are k nodes of v^i expected in T, and two paths in T have τ and $\tau + \delta$ ($\delta > 0$) nodes of v^i, respectively. Then with Lemma 2, the two paths imply it is expected that:

$$k \le 2^\tau - 2^{\tau-1} + 2 \quad \text{and} \quad k \ge 2^{\tau+\delta-1}.$$

Assume $\delta \ge 2$, we have

$$k \le 2^\tau + 2^{\tau-1} - 2 < 2^{\tau+\delta-1} \le k.$$

There is a contradiction that $k < k$, so we conclude our proof. □

Observation 2. *Denote the first inserted node of plaintext v^i in T as* P*. Let* P_1 *and* P_2 *be two nodes of v^i in T. If* $\mathsf{P}_1 < \mathsf{P}_2 \le \mathsf{P}$ *or* $\mathsf{P} \le \mathsf{P}_2 < \mathsf{P}_1$*, then* P_1 *is likely to have more ancestors with plaintexts other than v^i than* P_2*.*

Proof. We note a simple but important fact: If a node of plaintext v is inserted into the subtrees of a node with v^i, then it only can be inserted into the left (right) subtree when $v < v^i$ ($v > v^i$) because of the order relation. Then we discuss the case where $\mathsf{P}_1 < \mathsf{P}_2 \leq \mathsf{P}$.

In this case, the fact implies P_1 and P_2 have the same number of ancestors with plaintexts larger than v^i as P. So we only need to prove P_1 is likely to have more ancestors with plaintexts smaller than v^i than P_2. We first consider that the lowest common ancestor of the two nodes is one of them:

(1) P_1 is in the left subtree of P_2. All ancestors of P_2 are also ancestors of P_1. Besides, nodes of plaintexts smaller than v^i can appear in the left subtree of P_2 and even can be ancestors of P_1. So P_1 is likely to have more ancestors with plaintexts smaller than v^i.
(2) P_2 is in the right subtree of P_1. Nodes of plaintexts smaller than v^i cannot appear in the right subtree of P_1. So the two nodes have the same number of ancestors of plaintexts smaller than v^i.

When the lowest common ancestor of the two nodes is another node P'. Clearly, $\mathsf{P}'.plain = v^i$ as

$$v^i = \mathsf{P}_1.plain \leq \mathsf{P}'.plain \leq \mathsf{P}_2.plain = v^i.$$

Then nodes of plaintexts smaller than v^i can only appear in the left subtree of P' and even can be ancestors of P_1. This also implies P_1 is likely to have more ancestors with plaintexts smaller than v^i. So we conclude our conclusion with $\mathsf{P}_1 < \mathsf{P}_2 \leq \mathsf{P}$. The case where $\mathsf{P} \leq \mathsf{P}_2 < \mathsf{P}_1$ can be analyzed similarly. □

Observation 3. *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 higher node of them as* H, *then it holds that:*

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

where $\eta = depth(\mathsf{H})$. *Especially, it is expected that one of the following two cases holds:*

(1) H *is a leaf node of* v^i*;*
(2) The subtrees of H *have only one node of* v^i*, which is also a leaf node of* v^i*.*

Proof. The tree structure ensures that P_2 is the smallest node in P_1's right subtree or P_1 is the greatest node in P_2's left subtree. In the first case, P_2 is the higher node and $\eta = depth(\mathsf{P}_2)$. To insert P_2 into T, the client recursively executes Algorithm 1 for η times. In the last recursion, a ciphertext limit with size $\frac{|\mathbb{C}|}{2^{\eta-1}}$ is assigned to P_2 and the next smaller ciphertext (min) is $\mathsf{P}_1.cipher$. Therefore, it holds that:

$$\mathsf{P}_2.cipher = \mathsf{P}_1.cipher + \frac{1}{2} \cdot \frac{|\mathbb{C}|}{2^{\eta-1}}.$$

Besides, there is no node in the left subtree of P_2. According to Lemma 1, there is at most one node of v^i expected in the right subtree of P_2. So it is expected that P_2 is either a leaf node of v^i or its right subtree has only one node of v^i.

The second case can be analyzed similarly as P_1 is the greatest node in P_2's left subtree. The same equation holds with $\eta = depth(\mathsf{P}_1)$. It is expected that P_1 is a leaf node of v^i or its left subtree has only one node of v^i. □

Putting it all together. We combine our observations to gain insight into the distribution of ciphertexts. As we saw in Observation 3, if two nodes of v^i have neighboring ciphertexts, then the distance between their ciphertexts depends on the depth of the higher node.

The two cases in observation 3 imply that the number of ancestors with v^i of the higher node and one leaf node of v^i is expected to differ by no more than 1. Recall Observation 1 tells us that all leaf nodes of v^i are expected to have roughly the same number of ancestors with v^i. So the higher node is expected to have almost the same number of ancestors with v^i as leaf nodes of v^i. Now we consider the ancestors with other plaintexts of the higher node to study its the depth.

Observation 2 is used to complete the final step. It shows the higher node is likely to have more ancestors with other plaintexts if it is farther away from the first inserted node of v^i. Therefore, when the two nodes of v^i with neighboring ciphertexts are farther away from the first inserted node of v^i, the higher node is likely to have a greater depth. As a result, the distance between the neighboring ciphertexts is likely to be smaller.

Revealing frequency. 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 imply there are two stages in the traverse:

(1) *Increase stage.* As we traverse towards the first inserted node of v^i, the distance is expected to increase.
(2) *Decrease stage.* As we traverse away the first inserted node of v^i, the distance is expected to decrease.

It is important to note that our analysis assumes that the output of RandomCoin() produces roughly the same number of 0s and 1s, with a difference of no more than 1 as expected. However, due to the inherent randomness of the function, this assumption may not always hold true in practice, leading to some slight fluctuations in the real distance in a local field. To mitigate the impact of this randomness, we sum consecutive distances to observe the overall distance distribution. We refer to this notion as the α-ciphertext density, which provides a more robust characterization of the distribution of ciphertexts.

Definition 6 (α-ciphertext density). *For n ordered ciphertexts* $\mathbf{c} = \{c_1, ..., c_n\}$ *in Kerschbaum's FH-OPE scheme, the* α*-ciphertext density of* $c_j (\alpha < j < n - \alpha)$ *is denoted as* $d_{\alpha,j}$ *and is calculated as below:*

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

When we traverse sorted ciphertexts of the same plaintext and calculate the α-ciphertext density, the two stages described earlier also apply. This provides the potential of recovering frequency by finding the decrease stage of ciphertexts of v^i and the increase stage of ciphertetxs of v^{i+1}. To illustrate this, we give an example of the α-ciphertext distribution in Figure 3. We show the decrease stage of ciphertexts of v^i and the increase stage of v^{i+1}. Clearly, the density of the maximal ciphertext of v^i, denoted by d_{α,π_i} lies between the two stages and keeps a very small value. This implies the potential for finding π and revealing the frequency of plaintext v^i.

Tree depth. Note the non-uniform distribution is independent of the depth of the search tree T. It results from the gaps between the depths of nodes corresponding to the same plaintext. Moreover, the

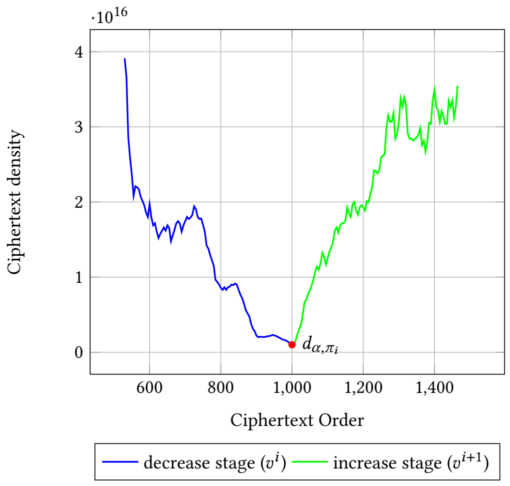

Figure 3. The α-ciphertext density of plaintexts v^i and v^{i+1} ($\alpha = 20$).

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.

4.3 Density Attack

Based on the observations and analyis, we present a frequency-revealing attack named *density attack* in Algorithm 2. It takes only ciphertexts and attacking parametwes (α, γ) as inputs and outputs an order set $\boldsymbol{\pi'}$, which is an estimation of $\boldsymbol{\pi}$. This attack works by traversing the sorted ciphertexts and calculating their α-ciphertext density. When it traverse 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 approximation 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.

Attacking parameters (α, γ)**.** α is used to calculate ciphertext density and γ is a threshold for noticing the two cases of conversion.

Algorithm 2: Density attack

```
Input: Ciphertexts $\boldsymbol{c} = \{c_1, ..., c_n\}, \alpha, \beta$
Output: An order set $\boldsymbol{\pi'} = \{\pi'_1, ..., \pi'_{N'}\}$
$stage = increase$
$density = -1$
$order = -1$
$\boldsymbol{\pi'} = \{\}$
for $j = \alpha + 1 \rightarrow n - \alpha$ do          traverse ciphertexts
    $d = c_{j+\alpha} - c_{j-\alpha}$
    if $stage = increase$ then
        if $d > density$ then
            $density = d$
            $order = j$
        end
        if $density > \beta \cdot d$ 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 $\beta \cdot density < d$ then                    Case 2
            $stage = increase$
            $\boldsymbol{\pi'}.add(order)$
            $density = d$
            $order = j$
        end
    end
end
```

Large α and γ can effectively remove the randomness brought by RandomCoin(). For example, in Figure 3, α-ciphertext density also slightly decreases (with the order from 1300 to 1350) in the increase stage. As $d_{1300} < \gamma \cdot d_{1350}$ still holds under a large γ, it will not be regarded as a decrease stage.

Efficiency and accuracy. Density attack succeeds by traversing ciphertext. Its computation is simple and the computational complexity is only $O(n)$. Meanwhile, in our experiments, it achieves over 96% accuracy in recovering plaintext frequency hidden by Kerschbaum's FH-OPE scheme.

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

In this section, we review the other two FH-OPE schemes: POPE and the FH-OPE scheme proposed by Li et al. Differ from Kerschbaum's FH-OPE scheme, they encrypt 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

Algorithm 3: Encrypt in POPE

```
Input: $v, sk$
State: Sorted binary tree $T$ of nodes $\{t\}$ in server
Client encrypts $v + r$ with semantically secure encryption:
    $c = \text{S.Encrypt}(sk, v + r)$ where $r \xleftarrow{\$} \{0,1\}^l/2^l$.
if $T$ is empty then
    Server sets $T.root = Node(c)$
else
    Client retrieves $T.root$ as $t$
end
while 1 do                    client searches $v + r$ in $T$
    if $v + r > \text{S.Decrypt}(sk, t.cipher)$ then
        if $t.right = \text{NULL}$ then
            Client tells server to set $t.right = Node(c)$;break
        else
            Client retrieves $t.right$ as $t$
        end
    else
        if $t.left = \text{NULL}$ then
            Client tells server to set $t.left = Node(c)$;break
        else
            Client retrieves $t.left$ as $t$
        end
    end
end
```

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.

In both schemes, a single-snapshot attacker can only observe plaintext order and semantically secure ciphertexts, making the density attack unfeasible. However, under a multi-snapshot attacker (cf. Definition 3), we show that the two schemes are still vulnerable to frequency-revealing attacks. Our analysis and attacks demonstrate the leakage of the ciphertext insertion order is essential for achieving frequency-hiding. To the best of our knowledge, *this is a new finding as this leakage is always ignored in OPE* [8, 22].

Batch encryption. The multi-snapshot attacker observes ordered ciphertexts $C = \{c_1, ...c_n\}$ at $\mu + 1$ distinct ordered moments $T = \{t_0, t_1, ..., t_\mu\}$, with an indicator sequence $\{\text{part}(\text{id}_1), ..., \text{part}(\text{id}_n)\}$ where $\text{part}(\text{id}_i) \in [\mu] \cup \{0\}$ indicates that c_i is first observed in $t_{\text{part}(\text{id}_i)}$. 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 $C^0 = \{c^0_1, ...c^0_{n_0}\}$.
(2) *Insertion batches.* The ith batch consists of all ciphertexts firstly observed in t_i, denoted as ordered ciphertexts:
$$\forall i \in [\mu], C^i = (c^i_1, ..., c^i_{n_i}) \ .$$
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 $(1, 1, 1, 2)$, 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 $(1.70, 1.32, 1.45, 2.04)$ and the randomized order is $(3, 1, 2, 4)$. *In this way, there is no plaintext with the same order in POPE*[2].

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 $r \xleftarrow{\$} \{0,1\}^l/2^l$ 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. We describe the encryption function in Algorithm 3. It takes the plaintext v and a secret key sk as inputs. We denote the binary search tree T as a set of nodes $\{t\}$. When this function is completed, the ciphertext of $v + r$ is inserted into T.

In fact, searching $v + r$ in T requires client and server to interact $O(\log n)$ rounds: In each round, the client retrieves the node compared from the server, decrypts its ciphertext and then compares its plaintext with $v + r$ to chose the next node compared. Therefore, POPE actually requires $O(\log n)$ rounds of interaction between client and server per range query, which makes it impractical.

The FH-OPE scheme proposed by Li et al. 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 in memory. 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 $rd \in [l, u]$ where $l = \sum_{v^i < v} fre(v^i)$ and $u = \sum_{v^i \leq v} fre(v^i)$. The ciphertext of v will be inserted between c_{rd} and c_{rd+1}.

We describe the encryption function in the client side in Algorithm 4. It takes a new plaintext v, the secret key sk, and the local table LT as inputs and returns a ciphertext c and a random order rd. The client uploads c and rd to the server and then the server inserts c into the search tree such that the position of c is between positions of c_{rd} and c_{rd+1}.

Other techniques. There are also some other important techniques in the two schemes. We refer the reader to [25, 32] for details. Here we just briefly introduce them as they have little effect on the property of frequency-hiding. POPE reduces the leakage of plaintext

[2]The probability of repeated random components is negligible in l.

Algorithm 4: Encrypt (client-side) in the FH-OPE scheme presented by Li et al.

```
Input: $sk, v, LT = \{(v^i, fre(v^i))\}$
Output: $c, rd$
$c$ = S.Encrypt$(sk, v)$   semantically secure encryption
$l = 0, u = 0$
for $i = 1 \rightarrow |LT|$ do
    if $v^i < v$ then
        $l$ += $fre(v^i), u$ += $fre(v^i)$
    else if $v^i = v$ then
        $u$ += $fre(v^i)$
end
if $v$ in $LT$ then
    $fre(v)$ += 1
else
    $LT.add((v, 1))$
end
$rd \xleftarrow{\$} [l, u]$
return $c, rd$
```

order by delaying the insertion of ciphertexts: The client first buffers some ciphertexts in the server. It leaks their insertion positions in T only when there are range queries for them. However, the client has to download these buffered ciphertexts to get their insertion positions, which incurs a huge communication volume overhead for range queries. In this paper, we focus on the ciphertexts whose insertion positions are public. We note range queries are common and frequent in plenty of real-world scenarios so there are only a small fraction of ciphertexts in POPE whose insertion positions can be protected[3]. Moreover, even if we consider that the range queries are rare, the multi-snapshot attacker fits POPE well as the attacker can access the encrypted database periodically to get the ciphertexts whose insertion positions are revealed by range queries during the last period of time.

For the FH-OPE scheme proposed by Li et al., the search tree in the scheme is named *coding tree* because it encodes ciphertexts according to their positions. These encodings are *order-preserving* so ciphertexts can be compared efficiently by using their encodings instead of positions. We note the coding method is similar to Kerschbaum's FH-OPE scheme and possibly leaks ciphertext insertion orders, which can leak more information to the multi-snapshot attacker than we used in this paper.

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.

[3]As the authors of POPE said, hiding partial order information is only suitable for cases where insertions are common and range queries are rare.

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 $\{0.19, 0.58, 0.42, 0.81\}$.

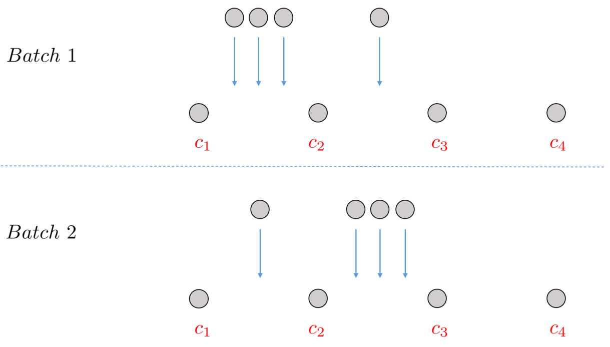

Figure 4. An example in POPE.

We visualize the insertion process in Figure 4. 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.
- 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., S.Decrypt$(sk, c_1) \neq$ 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 S.Decrypt$(sk, c_1) \neq$ 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 have the same underlying plaintext, this insertion pattern happens 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. Recall the two schemes leak only semantically secure ciphertexts and the orders of their underlying plaintexts. For any 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 the two schemes. 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 4. *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:

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

Proof. POPE preserves orders of distinct plaintexts, so $c = \mathsf{S.Encrypt}(v+r)$ where r is a random fraction between 0 and 1. POPE determines orders of repeated plaintexts with random components, we have

$$\boxed{\begin{array}{c} c < c_2 \Longleftrightarrow r < r_2, \\ c_1 < c < c_3 \Longleftrightarrow r_1 < r < r_3. \end{array}}$$

Therefore, the conditional probability is calculated as

$$\Pr(c < c_2 | c_1 < c < c_3) = \frac{r_2 - r_1}{r_3 - r_1}.$$

which indicates the Bernoulli distribution. □

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 *Fisher exact test* to complete the verification.

Fisher exact test [14] is a statistical test. Suppose there are two samples under binomial distributions as below:

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

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 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:

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

(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 in Figure 5. 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.

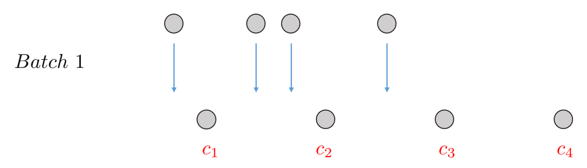

Figure 5. An example in the FH-OPE scheme in VLDB '21.

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 unusal: 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 5. *In the FH-OPE scheme proposed by Li et al., 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:

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

Proof. Recall the FH-OPE scheme proposed by Li et al. preserves orders of distinct plaintexts, so the underlying plaintext of c is v. Then the scheme determines its order with a random order rd. Similar to POPE, we have

$$\boxed{\begin{array}{c} c < c_2 \Longleftrightarrow rd < rd_2, \\ c_1 < c < c_3 \Longleftrightarrow rd_1 \leq rd < rd_3. \end{array}}$$

So the conditional probability is calculates as

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

□

Differ from the observation about POPE, Observation 5 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 0.5. So it is practical to verify if $\{c_1, c_2, c_3\}$ have the same underlying plaintext by testing if Y follows the binomial distribution $\mathsf{Bin}(n', 0.5)$.

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 $(n', 0.5)$ as inputs and returns a probability:

$$\boxed{pro = \mathsf{BinTest}(Y, n', 0.5)}$$

where pro is the probability of $Y \sim \mathsf{Bin}(n', 1/2)$. 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 do not. Besides, as 0.5 is only an estimation of p'_ts, we set a large γ to promise that ciphertexts would not fail to pass the verification because of the estimation error.

5.2.3 *Summary.* Here we explain why ciphertexts having distinct underlying plaintexts cannot pass the verifications w.h.p. We still consider the simple insertion case but assume that $\{c_1, c_2, c_3\}$ have distinct underlying plaintexts:

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

where $v^1 < v^2 < v^3$ and $\lfloor \cdot \rfloor$ is used for deleting random components in POPE. In this case, we show the binomial distribution Bin' that X follows violates the requirements in our verification.

In short, our verification process requires two conditions: 1) in POPE, Bin' is static and remains the same across insertion batches; 2) in the FH-OPE scheme proposed by Li et al., Bin' depends only on the order of ciphertexts. However, in the given case above, Bin' strongly depends on the distribution of plaintexts. Specifically, for a newly inserted ciphertext c, if its underlying plaintext is v^1, then $c < c_2$, and if its underlying plaintext is v^3, then $c > c_2$. As a result, Bin' is highly dependent on the distribution of $\lfloor \mathsf{S.Decrypt(sk, c)} \rfloor$, i.e., the plaintext distribution, which violates the second requirement.

Furthermore, the distribution of plaintexts is often dynamic [16, 19], making it difficult to ensure that the plaintext distribution remains the same across insertion batches. Consequently, Bin' is also dynamic and cannot remain the same across insertion batches, violating the first requirement. Therefore, we can expect our attacks to perform better with μ increasing because dynamic distributions are more likely to occur.

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 proposed by Li et al, respectively.

Sliding window. A multi-snapshot attacker is assumed to have a setup batch of ciphertexts $C^0 = (c^0_1, ...c^0_{n_0})$ 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.

To recover the plaintext frequency in C^0, we use a sliding window approach. For each three-tuple $\{c^0_{i-\alpha}, c^0_i, c^0_{i+\alpha}\}$, we verify whether ciphertexts from $c^0_{i-\alpha}$ to $c^0_{i+\alpha}$ have the same underlying plaintext. We record the order interval $[i - \alpha, i + \alpha]$ if the tuple fails the verification. If there are any overlapping intervals, we intersect them to obtain a shorter interval as the final interval. The sliding window and intersection are illustrated in Figure 6. 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 5 describes 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

Algorithm 5: Fisher exact test attack

```
Input: Setup batch $C^0 = \{(c_1^0, ..., c_{n_0}^0\}$,
       Attacking parameters $(\alpha, \gamma)$
       $\mu$ insertion batches $C^j = \{c_1^j, ..., c_{n_j}^j\}$ $(\forall j \in [\mu])$
Output: An order vector $\boldsymbol{\pi}' = (\pi_1', ..., \pi_{N'}')$
$a = 1, I = [1, n_0]$
for $i = \alpha + 1 \to n_0 - \alpha$ do                    sliding window
    for $j = 1 \to \mu$ do                                  get samples
        $Y_j = 0, n^{(j)} = 0$
        for $k = 1 \to n_j$ do
            if $c_{i-\alpha}^0 < c_k^j < c_i^0$ then $Y_j$ += 1;
            if $c_{i-\alpha}^0 < c_k^j < c_{i+\alpha}^0$ then $n^{(j)}$ += 1;
        end
    end
    $pro_i = \min\{\underset{\forall j_1, j_2 \in [\mu]}{\text{Fisher}}(Y_{j_1}, n^{(j_1)}, Y_{j_2}, n^{(j_2)})\}$
    if $pro_i < 1/\gamma$ then
        if $I \cap [i - \alpha, i + \alpha] \neq \emptyset$ then        intersect
            $I = I \cap [i - \alpha, i + \alpha]$
        else
            $order = \underset{\forall \theta \in I}{\arg\min}\, pro_\theta$
            $\pi_a' = order,\ a$ += 1
            $I = [i - \alpha, i + \alpha]$
        end
    end
end
```

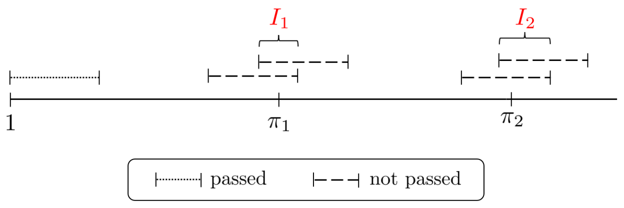

Figure 6. The Sliding window and intersection in our attacks. The intersection intervals I_i s are used to estimate π_is.

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 in Algorithm 6. 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.

There are also two parts of calculation in the algorithm: 1) the calculation of Y and n' (line 3-7); 2) the calculation of binomial test

Algorithm 6: Binomial test attack with one insertion batch

```
Input: Setup batch $C^0 = \{c_1^0, ..., c_{n_0}^0\}$,
       Attacking parameters $(\alpha, \gamma)$,
       Insertion ciphertexts $C^1 = \{c_1^1, ..., c_{n_1}^1\}$
Output: An order vector $\boldsymbol{\pi}' = (\pi_1', ..., \pi_{N'}')$
$a = 1, I = [1, n_0]$
for $i = \alpha + 1 \to n_0 - \alpha$ do                    sliding window
    $Y = 0, n' = 0$
    for $k = 1 \to n_1$ do                                  get sample
        if $c_{i-\alpha}^0 < c_k^1 < c_i^0$ then $Y$ += 1;
        if $c_{i-\alpha}^0 < c_k^1 < c_{i+\alpha}^0$ then $n'$ += 1;
    end
    $pro_i = \text{BinTest}(Y, n', 0.5)$
    if $pro_i < \frac{1}{\gamma}$ then
        if $I \cap [i - \alpha, i + \alpha] \neq \emptyset$ then        intersect
            $I = I \cap [i - \alpha, i + \alpha]$
        else
            $order = \underset{\forall \theta \in I}{\arg\min}\, pro_\theta$
            $\pi_a' = order,\ a$ += 1
            $I = [i - \alpha, i + \alpha]$
        end
    end
end
```

attack. The former costs $O(n_0 \cdot n_1)$ computation while the latter requires $O(n_0)$ times binomial tests.

Attacking parameter (α, γ)**.** In the two attacks, α is used to choose ciphertexts as inputs for Fisher exact test and binomial test. γ is a threshold for our verifications. It is also set large to promise low *false positive rate*, i.e., ciphertexts that have the same underlying plaintext are thought to have distinct underlying plaintexts with a probability close to $1/\gamma$.

6 EXPERIMENTS

In this section, we evaluate our frequency-revealing attacks by answering the following questions[4]:

1. How do our attacks choose suitable attacking parameters to ensure effectiveness? (§ 6.2)
2. How many plaintext frequencies can our attacks reveal under a small estimation error? (§ 6.3)
3. Is it possible to combine our attacks and existing inference attacks? (§ 6.4)

6.1 Experimential 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

[4]The code is available at https://github.com/XinleCao/Frequency-revealing-Attack

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 [17, 20, 29]. In this context, we select Apls because age and gender are typical attributes that FH-OPE schemes aim to protect [21, 32]. We also select Births and PBN as representative examples of dense and low-density datasets, respectively. The concepts of dense and low-density data are inherited from inference attacks [17, 28], as discussed in § 2.1.

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 greatly reflects the relative estimation error for revealing frequency in C (See additional experimental results in Appendix).

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 Appdenix A.1 so we can efficiently find suitable values. Besides, we also provide more experimental results in Appdenix A.1 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 in Figure 7-9. 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. Figure 7 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. Figure 8-9 show how accuracy and FP changes with α and γ increasing. When α and γ are larger than 15, the accuracy is still close to 1 but FP decreases to 0. We use a similar approach to select parameter values for all of our attacks on Births and other datasets.

The relation between γ and μ. For the tests in Fisher exact test attack and binomial test attack, γ determines the confidence for the test results, while μ affects the probability of ciphertexts with

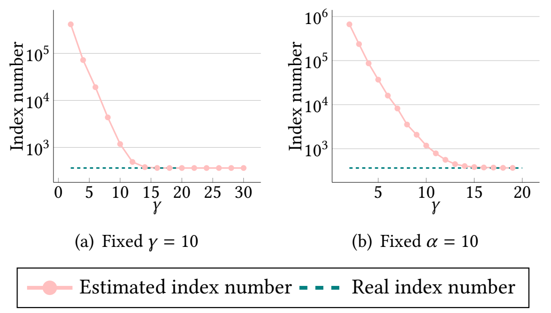

Figure 7. With α and γ increasing, the results outputted by the density attack become stable. The number of the estimated index (N') is close to the real index number (N).

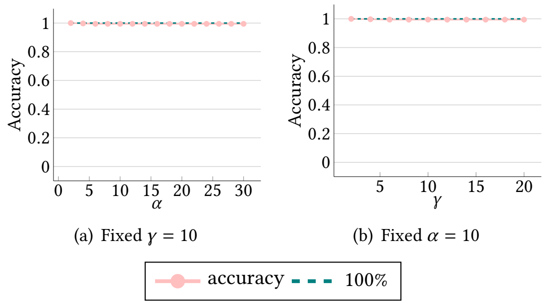

Figure 8. With α and γ increasing, the accuracy of the density attack is always close to 1.

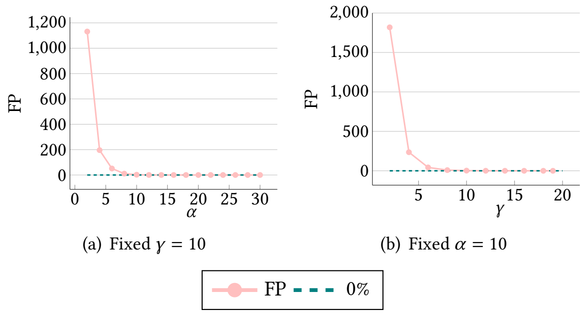

Figure 9. With α and γ increasing, the FP of the density attack decreases and is finally close to 0%.

distinct underlying plaintexts failing the tests. *To guarantee the test results can be accepted with high confidence, we only need a large γ and have no requirement for μ.* Given different γ values, the test results are accepted with different confidence levels. Under a larger γ and higher confidence level, it is harder for ciphertexts having distinct plaintexts to fail the tests, which decreases the *accuracy*. At that time, μ is required to be larger for increasing the accuracy. However, there are no specific requirements for the value of μ since the desired value is affected by the plaintext distribution in real-world scenarios. In our experiments, as we choose the value of γ in an adaptive manner instead of imposing a very high threshold directly, our attacks can achieve accuracy over 90% with $\mu \leq 10$.

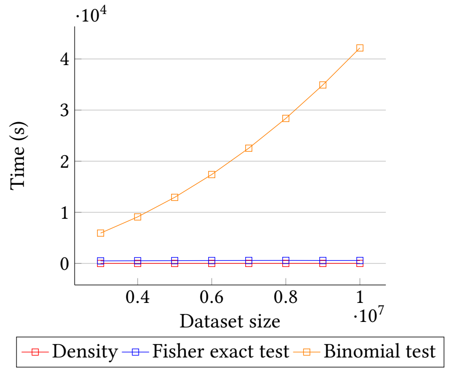

Figure 10. Attacking time of frequency-revealing attacks ($\mu = 1$).

#Batch (μ)	Fisher		Binomial		Density
	Min (%)	**Ave** (%)	**Min** (%)	**Ave** (%)	**Min** (%)
2	18.6	24.1	58.4	62.2	97.7
3	51.6	55.1	72.2	75.3	97.7
4	77.7	80.0	84.6	86.6	97.7
5	85.7	87.2	87.9	91.4	96.6
6	89.8	91.3	86.8	90.8	98.8
7	92.3	94.7	86.8	91.4	97.7
8	97.8	98.8	86.8	93.0	96.6
9	97.8	99.1	86.8	93.0	96.6
10	97.8	99.1	86.8	93.0	97.7
Total ($\mu = 10$)	98.4		92.6		99.5

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.

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 10, 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. 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 diving the ciphertexts of the entire dataset into multiple subsets based on their order. Once the attacks on the 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

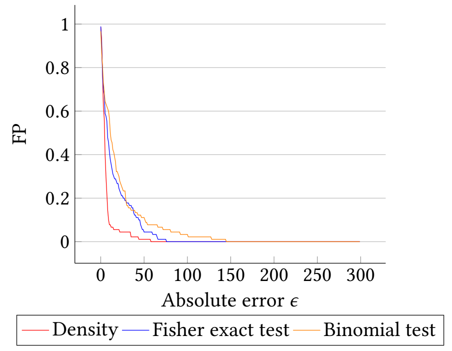

Figure 11. False positive rate of frequency-revealing attacks on Births ($\mu = 11$).

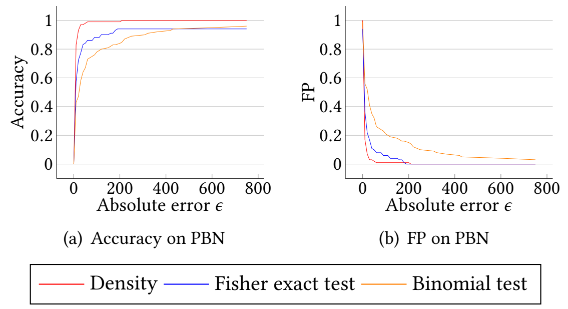

Figure 12. Performance of frequency-revealing attacks on PBN.

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 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 in Figure 11. 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 in Figure 12.

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

Table 2. The recovery rate of combined attack.

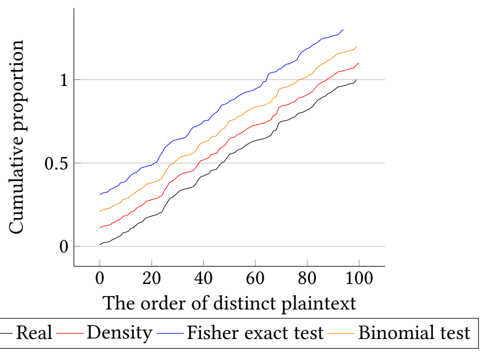

Figure 13. Cumulative plaintext distributions recovered by our attacks and real plaintext distribution on PBN. We give a vertical shift to each line for comparing them.

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 § 2.1 we 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. These attacks were considered useless to FH-OPE schemes with plaintext frequency hidden.

Sorting attack. Our attacks demonstrate 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 in Table 2, demonstrate that we are able to recover almost 100% of plaintexts protected by all existing FH-OPE schemes. These results highlight that FH-OPE schemes are ineffective in protecting gender and age datasets, which are precisely the types of datasets that FH-OPE was designed to protect.

Frequency-analyzing attacks. 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, in Figure 13. This finding suggests that frequency-analyzing attacks can potentially be applied to the results obtained from our attacks.

However, the combination of frequency-revealing attacks and inference attacks is not a trivial task. For sorting attacks, our attacks may not find exact $|\mathbb{M}|$ distinct underlying plaintexts when $|\mathbb{M}| > 50$. Therefore, we only conducted sorting attacks on the birthdays in January in Table 2. For frequency-analyzing attacks, it is important to standardize the adversarial power since all existing inference attacks adopt a single-snapshot attacker.

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* [16, 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. However, there are some inherent shortages in the technique, limiting its deployment. First, it requires that the client knows F_0 in advance, which can be out of reality. Second, it can result in unacceptable costs to simulate F with F_0 and F_1.

We note our attacks show that in FH-OPE the frequency leakage is somehow inherent. FH-OPE tries to randomize ciphertexts of repeated plaintexts with predefined algorithms, which makes the order relations between these ciphertexts *predictable.* However, the order relations between inserted ciphertexts of distinct plaintexts highly depend on plaintext distribution, which cannot be *exactly predicted.* The inconsistency makes the attacker possibly distinguish the two cases. Removing the gaps between the two cases is very challenging since plaintext distribution can be arbitrary and in plenty of scenarios cannot be known in advance. A well-known technique that can work under arbitrary unknown plaintext distributions is oblivious RAM (ORAM) [15, 34]. However, ciphertexts in ORAM cannot preserve plaintext order, losing the utility of OPE.

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. 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

After Kerschbaum [21] presented the notion of IND-FAOCPA and the first FH-OPE scheme, there has been much discussion on FH-OPE [7, 17, 18, 21, 25, 32].

Kerschbaum's FH-OPE scheme. As the first FH-OPE scheme, Kerschbaum's FH-OPE scheme has been studied a lot. Maffei et al. [26] analyze the security of Kerschbaum's FH-OPE scheme. They notice a contradiction between RandomCoin() and IND-FAOCPA. 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 snapshot attacker. They claim some attacks based on insertion order may exist in the future.

Binomial attack. The best known published attack against FH-OPE schemes is the *binomial attack* [17] proposed by Grubbs et al. It uses only the leakage of plaintext order to recover plaintexts. Roughly speaking, let sorted ciphertexts be $\boldsymbol{c} = (c_1, ..., c_n)$. To recover a distinct plaintext v^i, the attacker gets the lower and upper bound order of v^i by calculating $\phi_i' = n \cdot \Pr(v < v^i) + 1$ and $\pi_i' = n \cdot \Pr(v \leq v^i)$. Then the attacker regards the underlying plaintext of $(c_{\phi_i'}, ..., c_{\pi_i'})$ as v^i.

Based on only plaintext order, the binomial attack can be applied to any OPE scheme. However, it requires auxiliary public information about plaintexts: $\Pr(v < v_i)$ and $\Pr(v \leq v_i)$ but our attacks do not. Besides, it performs poorly in real-world datasets, i.e., in [17], it recovers no more than 30% plaintexts protected by Kerschbaum's FH-OPE scheme. Therefore, Grubbs et al. still suggest deploying FH-OPE schemes in reality as a countermeasure of inference attacks.

Attacks based on queries. There are also some attacks [13, 20, 23, 24] based on the leakages of queries: access pattern and communication volume. Access pattern indicates which records satisfy queries and communication volume indicates how many records satisfy queries. Kellaris et al. [20] prove OPE schemes cannot defend *reconstruction attacks* based on the two leakages of range queries. Kornaropoulos et al. [23] present new reconstruction attacks with range queries and k-nearest-neighbor (k-NN) queries. These construction attacks are effective to any generic encrypted database including those using OPE [5, 31, 36] and FH-OPE [21, 25, 32]. However, these attacks require plenty of range queries or k-NN queries for recovering plaintexts. Besides, these attacks assume the persistent attacker to get the leakages of queries.

Comparison. Compared with the attack against Kerschbaum's FH-OPE scheme and the binomial attack, 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 the security of Kerschbaum's FH-OPE scheme is greatly overestimated as the scheme provides little protection for plaintext frequency even under a single-snapshot attacker.

Compared with attacks based on queries, 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. Additionally, attacks based on queries require observing at least $O(N \log N)$ range queries [24] that are under some specific known distributions, such as uniform distribution. In contrast, our attacks require only ciphertexts and their partial insertion orders, making them effective even without any range queries observed."

Our work provides a starting point for future inference attacks on FH-OPE schemes. While the binomial attack actually shows that it is challenging to break FH-OPE schemes with only the leakage of plaintext order, we aim to explore other leakages in FH-OPE schemes that could provide more sensitive information for inference attacks. We exploit the leakages of non-uniform ciphertext distribution and ciphertext insertion orders, highlighting the significance of avoiding them, even if the leakage of ciphertext insertion orders is often disregarded in OPE [21, 22].

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. We evaluate the effectiveness of our attacks on three real-world datasets and show that we can recover over 90% of plaintext frequency in FH-OPE schemes. Moreover, we discuss the potentiality of applying inference attacks on existing FH-OPE schemes. 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.

[2] [n.d.]. https://www.osi.ca.gov/CalHEERS.html.

[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] 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

[12] 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

[13] 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

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

[15] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (may 1996), 431–473. https://doi.org/10.1145/233551.233553

[16] 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.

[17] 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

[18] 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

[19] Chip Huyen. 2022. *Designing Machine Learning Systems.* " O'Reilly Media, Inc.".

[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] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM* 65, 4, Article 18 (apr 2018), 26 pages. https://doi.org/10.1145/3177872

[35] 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.

[36] 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

A ADDITIONAL EXPERIMENTAL RESULTS

The additional experimental results consist of three parts: 1) the experiments and techniques about choosing attacking parameters; 2) the insertion of new distinct plaintexts; 3) more evaluation of relative estimation error.

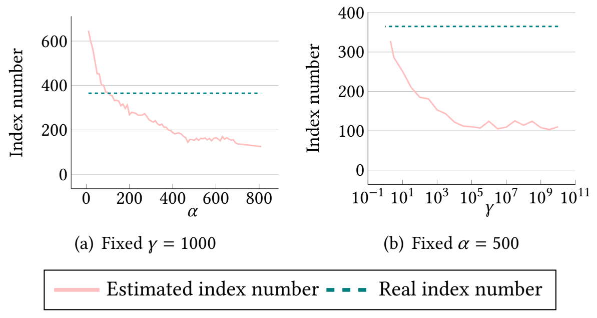

Figure 17. Strict values of α and γ in the binomial test attack make the estimated index number (N') stable. It is not close to the real index number (N) as we only use one insertion batch ($\mu = 1$).

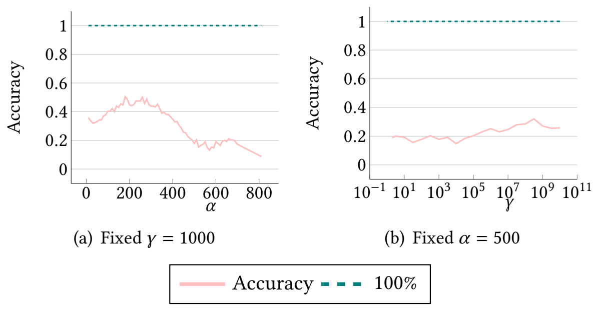

Figure 18. With α and γ increasing, the accuracy of the binomial test attack first increases and then decreases.

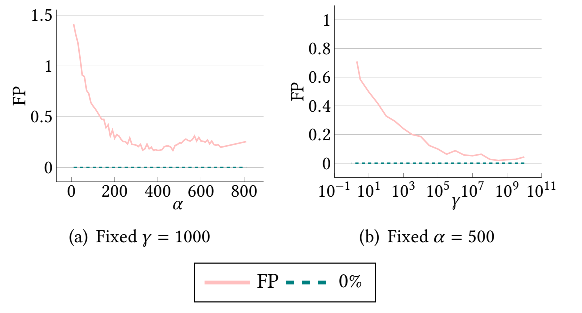

Figure 19. With α and γ increasing, the FP of the binomial test attack decreases ($\mu = 1$).

A.1 Choosing attacking parameters

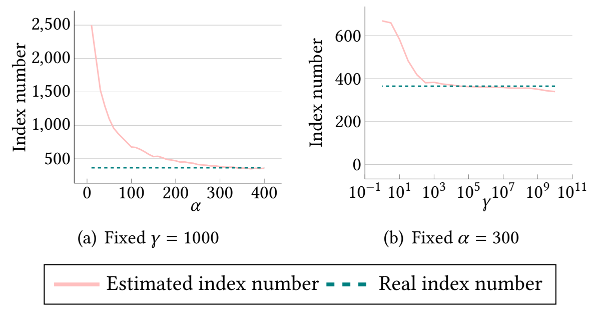

Figure 14. Strict values of α and γ in Fisher exact test attack make the estimated index number (N') stable and close to the real index number (N) under $\mu = 10$.

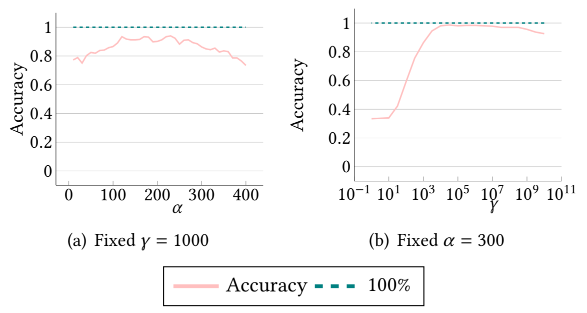

Figure 15. With α and γ increasing, the accuracy of the Fisher exact test attack first increases and then decreases a little under $\mu = 10$.

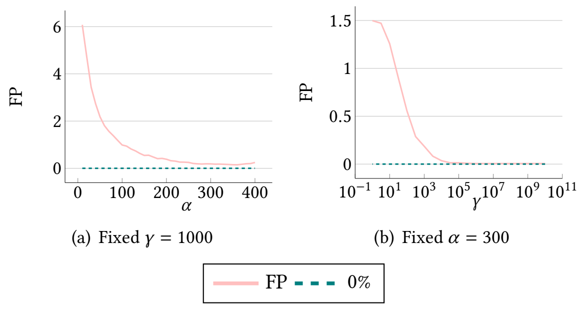

Figure 16. With α and γ increasing, the FP of the Fisher exact test attack decreases and finally is close to 0% under $\mu = 10$.

We introduce some optimizations for the initial values and values increasing about the attacking parameters (α, γ) as follows:

- *Initial values.* In Fisher exact test attack and binomial test attack, $1/\gamma$ can be described as the probability of ciphertexts having the same underlying plaintext failing in the tests. So if we require that this probability is less than some values like 0.1%, then we should initiate γ with a value no more than 1000.
- *Traverse with a fixed parameter.* In general, as we initiate γ with a value like 1000, we fix γ and increase α from 1 to a value α' such that the results outputted are stable. Finally, we fix α with α' and increase γ until the results outputted are stable. This is much more efficient than traversing each non-negative integer pair although the latter definitely works and possibly can find more suitable values.
- *Adaptive jump.* We traverse the integer pairs for (α, γ) in different step sizes to quickly choose suitable parameter values. For example, we initiate $\alpha = 1$ and then adds it with $step = 1000$, which means $\alpha := \alpha + step$. If we find the values deserved are in an interval like $[1, 1001]$. Then we traverse this interval with $step = 100$. In this way, we can find the suitable values much faster than traversing each integer.

Besides, we also provide more experimental results here to illustrate the effect and parameter selection process in Figure 14-19. To show as many groups of experiments as possible, we randomly select a subset of the Births dataset to conduct experiments. In Figure 14-19, the unit of α is 10, and the γ scales by multiplying by 10. These figures show, with α and γ increasing, the results outputted by attacks become stable, the accuracy increases and the FP decreases. We explain some details:

(1) In Figure 7-9, as the density attack is more robust, we can find values for (α, γ) at the same time. So we just fix one parameter with 10 and increase another parameter until the attack results become stable.
(2) In Figure 14-19, Fisher exact test attack and binomial test attack are more sensitive to the parameter values, so they have to find parameter values one by one. They first fix $\gamma = 1000$ to guarantee a relatively strict threshold, i.e., ciphertexts having the same underlying plaintext fail the test with a probability of nearly 99.9%. Then they increase α until the attack results become stable ($\alpha = 300$ in Fisher exact test attack and $\alpha = 500$ in binomial test attack). Finally, with α fixed, they increase γ until the attack results become stable ($\gamma = 10^5$ in both the two attacks).
(3) In Figure 17-19, as we only apply one insertion batch for the binomial test attack, the accuracy is not close to 100%. With more insertion batches, the accuracy will increase and finally is more than 90% in our experiments.
(4) Except for the density attack, the results of the other two attacks show that too larger values for α and γ incur some loss in accuracy. That's why in Figure 14-19, with α and γ continually increasing the stable results are reduced again and the accuracy decreases a little. In reality, we choose the value at the first stable stage to guarantee the values for α and γ will not be too large. On the contrary, it is clear that too small values of (α, γ) result in unacceptable FP.

Here we note that the experimental results in Figure 14-19 are based on a subset of Birth because we want to show results under as much different parameter values as possible but the Fisher exact test and Binomial test attacks are relatively expensive and the dataset size is very large. Plaintexts in the subset are uniformly chosen from the whole dataset. The subset size is 1/10 of the whole dataset size. We believe results on the subset are still sufficient for showing the effect of parameter values and the parameter-chosen process. In reality, we actually only conduct a small number of parameter values to find suitable parameter values as the parameter chosen process only relies on a rough trend instead of the detailed results under different parameter values (which implies the unit of α and γ can be larger). Therefore, the parameter-chosen process cannot be used for defending against our attacks.

B INSERTION OF NEW DISTINCT PLAINTEXTS

Our discussion in § 7.1 presents the insertion of new distinct plaintexts is too dangerous for protecting plaintext frequency. Even if there are only a few distinct plaintexts inserted, the frequency of plaintexts can be directly revealed. Specifically, the insertion of new distinct plaintexts enables another very simple and efficient attack under the multi-snapshot attacker: The attacker can notice the insertion and reveal frequency by observing if there are significant ciphertexts inserted between two neighboring ciphertexts. Here we give additional experiments to illustrate and verify this simple attack. This attack does not belong to the main contributions of this paper but should be paid attention to.

For each of the three FH-OPE scheme, we pick the PBN dataset and insert the insertion batch C^1 to the setup batch C^0. We calculate the number of ciphertexts inserted between two neighboring ciphertexts in C^0 and show the results in Figure 20. For each non-negative integer i, we calculate the number of pairs of neighboring ciphertexts between which there are no less than i ciphertexts in C^1 inserted. The results show for any pair of neighboring ciphertexts in C^0, there are at most 60 ciphertexts in C^1 inserted between them even if the size of C^1 is very large ($|C^0| \approx 5591$K and $|C^1| \approx 9199$K). Therefore, if some new distinct plaintexts with a frequency of over 60 (e.g., 100) are encrypted and inserted when inserting C^1 into C^0, the multi-snapshot attacker observes a significant number of ciphertexts inserted between some pairs of neighboring ciphertexts and guess the insertion of new distinct plaintexts happens. Then the frequency of ciphertexts belonging to the new distinct plaintexts and some ciphertexts in C^0 and C^1 can be both directly revealed.

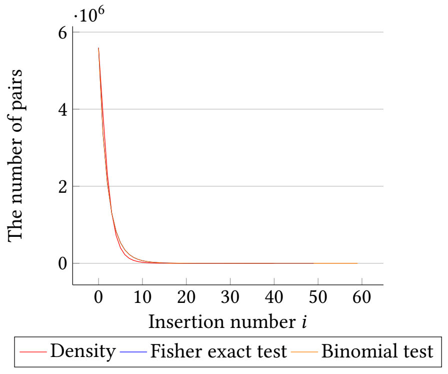

Figure 20. The insertion of ciphertexts without those of new distinct plaintexts. For each non-negative integer i, we show the number of pairs of neighboring ciphertexts in C^0 between them there are no less than i ciphertexts in C^1 inserted between them.

C RELATIVE ESTIMATION ERROR

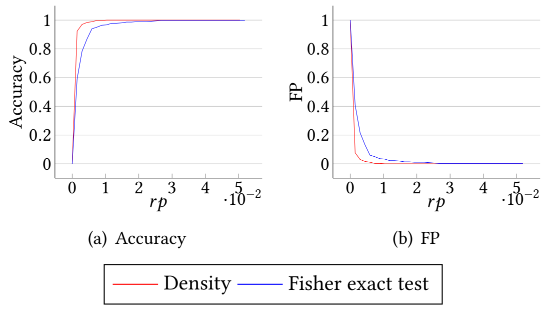

Figure 21. Performance of frequency-revealing attacks on C^0 of Births.

In this paper, we use an absolute error ϵ to evaluate the estimation error and set ϵ as a small value to guarantee a small relative estimation error. Here we give more direct results about the relative estimation error for further understanding the proportion of

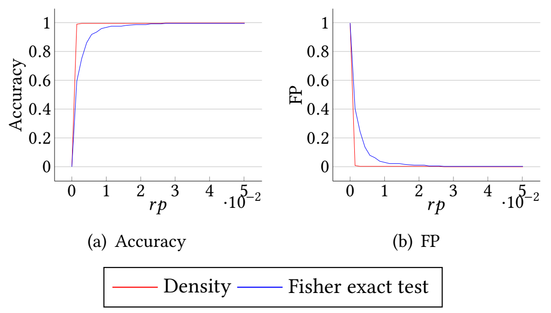

Figure 22. Performance of frequency-revealing attacks on C of Births.

ciphertexts in FH-OPE affected and threaten by our attacks. We define the relative estimation error as

$$rp = \frac{\epsilon}{\min_{\forall i \in [N]} \{\pi_i - \phi_i\}}.$$

Here rp is a strict upper bound of relative estimation error as it picks the smallest frequency to calculate the relative error. If the order range $[\phi_i, \pi_i]$ of plaintext v^i is revealed under a relative estimation error rp, then at least $1 - 2rp$ of ciphertexts of v^i are founded correctly. The attacker knows these ciphertexts have the same underlying plaintext although it may not know the value of v^i. For example, under $rp = 3\%$, the attacker finds at least 94% of ciphertexts of v^i.

Here we show the performance of our attacks under different relative errors on revealing frequency in both C^0 and C as the Fisher exact test attack and binomial test attack adopt a multi-snapshot attacker. We note when the attacker reveals the frequency of v^i in C^0 under a relative estimation error rp, then it also reveals the frequency of v^i in C under a similar relative estimation error rp. This is because the ciphertexts of v^i in insertion batches are inserted proportionally, e.g., suppose the attacker finds ciphertexts c_{ϕ_i} and c_{π_i} in C^0 where 96% of ciphertexts of v^i in C^0 are between them, then in each insertion batch, there are also around 96% ciphertexts of v^i inserted between the two ciphertexts. We show the accuracy and FP under different relative estimation error in Births in Figure 21 and Figure 22. The results show under $rp = 5\%$ (corresponding to 90% ciphertexts of distinct plaintexts), our attacks still can achieve over 90% accuracy and a FP less than 1%.