Efficient Unbalanced Private Set Intersection Cardinality and User-friendly Privacy-preserving Contact Tracing

Mingli Wu and Tsz Hon Yuen, *The University of Hong Kong*

https://www.usenix.org/conference/usenixsecurity23/presentation/wu-mingli

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Efficient Unbalanced Private Set Intersection Cardinality and User-friendly Privacy-preserving Contact Tracing

Mingli Wu
The University of Hong Kong

Tsz Hon Yuen
The University of Hong Kong

Abstract

An *unbalanced* private set intersection cardinality (PSI-CA) protocol is a protocol to securely get the intersection cardinality of two sets X and Y without disclosing anything else, in which $|Y| \ll |X|$. In this paper, we propose efficient unbalanced PSI-CA protocols based on fully homomorphic encryption (FHE). To handle the long item issue in PSI-CA protocols, we invent two techniques: *virtual Bloom filter* and *polynomial links*. The former can encode a long item into several independent shorter ones. The latter fragments each long item into shorter slices and builds links between them.

Our FHE-based unbalanced PSI-CA protocols have the lowest communication complexity $O(|Y|\log(|X|))$, which is much cheaper than the existing balanced PSI-CA protocols with $O(|Y|+|X|)$. When $|X| = 2^{28}$ and $|Y| = 2048$, our protocols are $172\times \sim 412\times$ cheaper than the best balanced PSI-CA protocol. Our protocols can be easily modified into unbalanced PSI protocols. Compared with Cong et al. (CCS'21), one of our unbalanced PSI protocols can save $42.04\% \sim 58.85\%$ communication costs and accelerate the receiver querying time.

We apply our lightweight unbalanced PSI-CA protocols to design a privacy-preserving contact tracing system. We demonstrate that our system outperforms existing schemes in terms of security and performance.

1 Introduction

Private set intersection cardinality (PSI-CA) allows two parties (i.e., a receiver who has a set Y and a sender who has a set X) to securely compute the intersection cardinality of their sets. Specifically, the receiver learns only the cardinality $|X \cap Y|$ without disclosing any item of Y to the sender; the sender gets no output and discloses only $|X \cap Y|$ to the receiver. When $|Y| \ll |X|$, we have *unbalanced* PSI-CA (uPSI-CA). Unbalanced PSI-CA has many applications (e.g., data mining [54], genome test [5]), in which COVID-19 contact tracing [20, 52] is a significant one to control the pandemic. Users (i.e., the receivers) can secretly detect the number of close contacts who are diagnosed with COVID-19. Then measurements can be taken to protect themselves and their acquaintances (e.g., their families, friends) in time.

Though many balanced PSI-CA protocols [16, 18, 22, 54] have been proposed, they are not efficient when the set sizes are unbalanced, especially in communication costs. Duong et al. [20] designed a user-friendly uPSI-CA protocol but relied on third parties. It is noted that a balanced PSI-CA protocol can also be used in the unbalanced case. However, the performance is poor, especially with expensive communication costs. For example, when $n = 2^{28}$, the communication cost of balanced PSI-CA protocol [16] can be as high as 2560 MB.

PSI-CA is closely related to private set intersection (PSI). The difference is that the receiver learns $X \cap Y$ in PSI rather than $|X \cap Y|$ in PSI-CA. Intuitively, designing PSI-CA is more difficult than designing PSI. Knowing the intersection, the receiver can easily know the intersection cardinality. However, only knowing the cardinality, the receiver usually cannot know the items in the intersection. Noticing this relation, to design a lightweight uPSI-CA protocol, we first investigate the related uPSI (i.e., unbalanced PSI) protocols and find uPSI protocols based on fully homomorphic encryption (FHE) are the best.

1.1 Naive FHE-enabled uPSI

CLR'17 [13] proposed a uPSI protocol with low communication complexity (i.e., $O(n_y \log(n_x))$) by FHE, where n_y and n_x are respectively the set size of the receiver and sender. The core idea is bin-wise matching. They first profile the smaller set Y into a Cuckoo hash table and insert the larger set X into a two-dimensional simple hash table by using the same hash functions. For each item y in the Cuckoo hash table, one only needs to compare it with the items $x_1, x_2, \cdots, x_b$ in the simple hash table at the same bin location, where b is the bin size (i.e., the number of items in the bin). Specifically, the sender first constructs a normal polynomial $P(\cdot)$ using $x_1, x_2, \cdots, x_b$:

$$P(x) = \prod_{i=1}^{b}(x - x_i) = c_0 + c_1 \cdot x + \cdots + c_b \cdot x^b,$$

where c_i $(i = 0, 1, \cdots, b)$ are the polynomial coefficients. Then the receiver can send the FHE ciphertext $[\![y]\!]$ to the sender. Upon receiving $[\![y]\!]$, the sender can compute

$$P([\![y]\!]) = c_0 + c_1 \cdot [\![y]\!] + \cdots + c_b \cdot [\![y^b]\!],$$

and returns it back to the receiver. The receiver decrypts the result ciphertext $P([\![y]\!])$ to get $P(y)$. If $P(y)$ is 0, then y is in the intersection; otherwise not.

Although the performance of CLR'17 [13] is good, it can only support items with bit length $\sigma \leq 32$. When naively increasing σ in their protocol, the related FHE parameters have to be substantially increased on par, resulting in high performance deterioration. Handling longer items in FHE-based PSI is challenging. To solve this issue, their following work [12] used general single instruction multiple data (SIMD) encoding scheme, which is difficult to be implemented in SEAL (an open-source FHE library). The latest work [15] instead took a simple but highly effective strategy: item slicing. To handle a long item, they simply slice a long item y into shorter items, e.g., $y^{(1)}||y^{(2)}||y^{(3)}$, where "||" is bit string concatenation. However, simple slicing has an obvious drawback: false positives. For example, to match an item $y = y^{(1)}||y^{(2)}||y^{(3)}$ of the receiver with three items $x_1 = x_1^{(1)}||x_1^{(2)}||x_1^{(3)}, x_2 = x_2^{(1)}||x_2^{(2)}||x_2^{(3)}, x_3 = x_3^{(1)}||x_3^{(2)}||x_3^{(3)}$ of the sender after slicing, the receiver may mistakenly take y as a matched item if $y^{(1)} = x_1^{(1)}$, $y^{(2)} = x_2^{(2)}$, and $y^{(3)} = x_3^{(3)}$. More details will be given in section 6.3.

In addition to the drawback, all the above FHE-based works [12, 13, 15] are prone to the deception attack, in which the sender deceives the receiver to accept an item not in $X \cap Y$ as a matched one without being noticed by the receiver. This attack is practical in real-world PSI applications. For example, in *private contact discovery* [29], a social service provider (e.g., Facebook, Signal) can deceive a user that he/she has many common friends who are using the same application; then the user will be motivated to choose this application too. The details are given in section 3.1.

1.2 Overview of Our Solutions

To overcome the above drawbacks, we propose three uPSI-CA protocols by using a cryptographic primitive *shuffled oblivious pseudorandom function* (shuffled OPRF) and inventing two techniques: virtual bloom filter (VBF) and polynomial links. In general, our protocols have two parts: shuffled OPRF preprocessing and securely getting the intersection.

Shuffled OPRF. A shuffled OPRF allows the receiver with inputs $Y = \langle y_1, y_2, \cdots, y_{n_y} \rangle$ to learn shuffled outputs $Y^o = \langle OPRF_k(y_{\pi(1)}), OPRF_k(y_{\pi(2)}), \cdots, OPRF_k(y_{\pi(n_y)}) \rangle$, where k is the sender's input key and $\pi(\cdot)$ is the sender's random input permutation. The sender learns nothing from this functionality. The receiver cannot find the corresponding OPRF value for a specific input. Since the sender knows the key k, he/she can trivially compute the OPRF outputs $X^o = \langle OPRF_k(x_1), OPRF_k(x_2), \cdots, OPRF_k(x_{n_x}) \rangle$ by inputting his/her items. Then the receiver can only know $|X^o \cap Y^o| = |X \cap Y|$ after implementing PSI with X^o and Y^o. It is noted that the receiver does not know the corresponding OPRF value of any item in his/her set after shuffling, and thus knows nothing about the items in $X \cap Y$ even when they know $X^o \cap Y^o$. After preprocessing X and Y by the shuffled OPRF, we can focus on the techniques for handling the long item issue.

VBF Encoding. Virtual Bloom Filter (VBF) can encode a long item into several shorter ones. Bloom Filter is a data structure that can map each item of a set into k_v bins of a hash table with size m by using k_v uniform hash functions. We propose VBF based on Bloom Filter. For a long item y, we encode it into k_v shorter one. Specifically, we represent y as its k_v bin indices, which have shorter bit lengths (i.e., $\lceil \log_2(m) \rceil$). For example, if y with bit length $\sigma = 128$ is mapped to bin # 2, 5, 9 out of a total of 16 bins, we encode it as $y := (2, 5, 9)$. Now we can represent a 128-bit item y as $3 \times \log_2 16$ bits. After VBF encoding, one can do the above bin-wise matching by FHE. If all the k_v VBF sub-items (e.g., the table indices 2,5,9) of an item are matched, then this item is in the intersection with desired probability. More details can be found in subsection 4.1.

Polynomial Links. In polynomial links (PoL), for each item x, we first slice it as $x = x^{(1)}||x^{(2)}||\cdots||x^{(k_s)}$. To make bin-wise matching, we then build links by using k_s polynomials $P_i(\cdot)$ $(i = 1, 2, \cdots, k_s)$, which are built by the corresponding slices in the sender's bin. We construct the first polynomial $P_1(\cdot)$ by utilizing the first slices of items in the sender's bin $x_1^{(1)}, x_2^{(1)}, \cdots, x_b^{(1)}$, such that $P_1(x_j^{(1)}) = 0$ for all j. Then we construct the other polynomials $P_i(x_j^{(1)}) = x_j^{(i)}$ $(i = 2, 3, \cdots, k_s)$ by using interpolation over the points $(x_1^{(1)}, x_1^{(i)}), (x_2^{(1)}, x_2^{(i)}), \cdots, (x_b^{(1)}, x_b^{(i)})$. To make bin-wise matching, for an item $y = y^{(1)}||y^{(2)}||\cdots||y^{(k_s)}$, the receiver can simply check whether $P_1(y^{(1)}) = 0$ and $P_i(y^{(1)}) = y^{(i)}$ $(i = 2, 3, \cdots, k_s)$. If all slices $y^{(i)}$ $(i = 1, 2, \cdots, k_s)$ are matched, then y is in the intersection. It is noted that $P_i(y^{(1)})$ can be securely computed by the sender using FHE. Here we neglect the FHE computation steps for simplicity. Since the slices can be much shorter than the original item, the long item issue can be resolved. Refer to subsection 4.2 for more details.

1.3 Our Contributions

In summary, we have the following contributions in this paper:

- We propose efficient uPSI-CA protocols without relying on any third party. Compared with the existing PSI-CA protocols, our protocols have many strengths.

Our FHE-based protocols have the lowest communication complexity $O(n_y \log(n_x))$. Most of previous two-party PSI-CA protocols only consider the balanced case and their best communication complexity is $O(n_y + n_x)$. However, when the set size n_x of the sender is large in the unbalanced case, their communication costs can be much larger than ours. For example, when $n_x = 2^{28}$ and $n_y = 2048$, the communication costs of our three protocols are respectively 14.82 MB, 6.21 MB, and 7.66 MB; however, Cristofaro et al. [16] needs 2560.13 MB, which is $172\times \sim 412\times$ as expensive as ours.

- Our uPSI-CA protocols can be easily modified to uPSI protocols. Compared with the state-of-the-art uPSI protocol [15], our first PoL based protocol eliminates false positives and has the lowest communication costs in all settings. It can save $42.04\% \sim 58.85\%$ communication costs compared with Cong et al. [15]. Also, the receiver in our PoL based protocol can get the intersection more quickly. Our second VBF+slicing based protocol has slightly better performances than [15] and can greatly alleviate the deception attack. Our third protocol combines both the PoL and VBF to get balanced performances between our first two protocols. Our PoL-based and VBF+PoL based protocols can inherently thwart the deception attack, while none of the previous FHE-based uPSI protocols [12, 13, 15] can.
- By utilizing our uPSI-CA protocol, we develop a user-friendly privacy-preserving contact tracing system. Our system can satisfy all the security and performance requirements while none of the existing designs can. For example, compared with the well-known contact tracing schemes Google&Apple [2] and DP-3T [53], we can provide strong privacy protection for the users by thwarting the linkage attacks. Compared with Catalic [20], we do not need non-colluded third parties when utilizing our PoL-based uPSI-CA protocol.

2 Related Work

In this section, we introduce the works mostly related with ours in PSI-CA, unbalanced-PSI, and contact tracing. Due to the page limit, we put other related works in Appendix A.

PSI-CA. Kacsmar et al. [28] exploited differential privacy to design their uPSI and uPSI-CA protocols, which have the communication complexity $O(n_y \log(n_x))$. However, their concrete communication costs are very expensive. When $n_x = 2^{20}$, both their uPSI and uPSI-CA protocols have the communication costs > 100 MB [25][1], which is $> 10\times$ more expensive than our uPSI and uPSI-CA protocols. The other existing PSI-CA protocols (e.g., [16, 23, 33, 54]) and circuit-based PSI protocols (e.g., [11, 43, 44, 47]) that can be used to compute the cardinality are *balanced* ones with at least $O(n_y + n_x)$ communication cost and it is not appropriate to be used in *unbalanced* cases, especially when n_x is large. It is noted that in *unbalanced* PSI-CA, the focus is on achieving low communication costs.

Cristofaro et al. [16] designed the first PSI-CA protocol with linear complexity $O(n_y + n_x)$ based on Decisional DH (DDH) assumption in the random oracle model (ROM) [27]. Suppose that β and k are respectively the receiver's key and the sender's key. The receiver first sends $H(y_i)^\beta$ ($\forall i \in [n_y]$) to the sender. The sender computes $(H(y_i)^\beta)^k$ and sends them to the receiver after shuffling. Then the receiver can exploit the commutative property of DH to compute $((H(y_i)^\beta)^k)^{1/\beta} = ((H(y_i)^k)^\beta)^{1/\beta} = H(y_i)^k$. The sender also sends $H(x_i)^k$ to enable the receiver to compute the intersection cardinality. For the computation cost, both the sender and the receiver need to compute two exponentiation for each item. The communication cost of their protocol is small. For example, when the set size $n = 2^{20}$, their protocol takes 74 MB. Similarly, Lv et al. [33] exploited commutative encryption to design their PSI-CA protocol. In their protocol, it takes 135.25 MB in communication when $n = 2^{20}$, which is $1.8\times$ as expensive as [16]. Vaidya et al. [54] utilized the commutative hash functions but with their focus on multi-party PSI-CA rather than two-party.

Miao et al. [34] targeted computing the intersection sum, though they could also get the cardinality. In addition to using DH related assumption (q-DHI assumption), they also used other primitives (e.g., Zero-Knowledge Argument of Knowledge, ElGamal encryption), which made it much less efficient than Cristofaro et al. [16]. Circuit-based PSI can be used to realize PSI-CA [43, 44][2]. The notion of circuit-based PSI was proposed by Huang et. al. [26], in which each party gets the correct shares for the intersection items rather than the intersection. Then computation can be done to realize the targeted functionality (e.g., PSI-CA) by using the specific circuit. Pinkas et. al. [43] proposed a circuit-based protocol by using their Oblivious Programmable Pseudorandom Function (OPPRF). To realize PSI-CA, in their previous work [44], they mentioned that Hamming weight circuit [8] could be used. Chandran et al. [11] achieved better performance than [43]. The latest circuit-based PSI protocol [47] only needed 115 MB (i.e., the lowest one), but still about $1.6\times$ as expensive as [16]. For the computation cost, it was $1.53\times$ as fast as [11]. With a similar idea, Garimella et al. [23] combined the OPPRF in [43] with oblivious switching in [36] to designed a PSI-CA protocol. The communication cost of their protocol is

[1] Kacsmar et al. [28] did not show the communication costs for their uPSI-CA protocol. He et al. [25] also used differential privacy to design their PSI and PSI-CA protocols and made comparisons with [28]. However, their communication complexity is $O(n_y + n_x)$. When $n_x = 2^{20}$, their communication costs are larger than [28].

[2] In [43, 44], the authors showed the performance of PSI-CAT protocols by using their circuit-based PSI protocols. In PSI-CAT, the receiver knows whether the cardinality is over a threshold. The costs of their PSI-CA protocols are slightly cheaper than their PSI-CAT protocols.

1155 MB when $n = 2^{20}$, which is 15.6× as expensive as [16].

All the above protocols are secure in the semi-honest model except that [34] is secure in the malicious model. Cristofaro et al. [16] achieves the lowest communication cost. Circuit-based PSI can be very fast (e.g., 15s in 1Gbps LAN setting when $n = 2^{20}$ [47]). By exploiting the circuit-based PSI protocol in [47], one can design a PSI-CA protocol that runs the fastest when the bandwidth is high (e.g., 1Gbps).

Unbalanced PSI. In above mentioned FHE-based uPSI protocols [12, 13, 15], the communication complexity is $O(n_y \log(n_x))$, which is cheap. However, the sender needs to compute $m\alpha$ normal polynomials, thus the computation complexity is $O(\frac{n_x^2}{m\alpha})$. Here m is the Cuckoo hash table size, and α is the partition number to partition a large bin into several smaller ones. The computation cost for the sender is large, especially when n_x is large. Except for the FHE-based ones, Kales et al. [29] presented two optimized uPSI protocols LowMC-GC and ECC-NR based on the Cuckoo filter, garbled circuit, and Oblivious Transfer extension (OTe) [4, 30]. Their protocols required an offline preprocessing phase in which the sender sends a compressed Cuckoo filter to the receiver and the receiver needs to prepare the storage. For performance, their LowMC-GC achieves the lowest sender offline computation cost. For example, when $n_x = 2^{24}$ and $n_y = 5535$, it only takes 87.85s for their LowMC-GC, which is about 8.8× as fast as the state-of-the-art [15]. Their ECC-NR protocol achieves lower communication costs than LowMC-GC. In the same set size, ECC-NR is 3.9× as cheap as LowMC-GC. However, compared with [15] that achieved the lowest communication cost, ECC-NR is about 2.9× more expensive. For security, both Kales et al. [29] and [15] can thwart a malicious receiver and are secure for a semi-honest sender.

Contact tracing. In contact tracing, a user has an identifier set Y and wants to know whether he/she is a close contact by comparing Y with the identifier set X of the Public Health Authority (PHA). For contact tracing, the designed protocol should be unbalanced because $|Y| \ll |X|$. Users usually utilize their cellphones to finish contact tracing. Since the cellphone is resource-constrained, a good contact tracing protocol is supposed to focus on the performance on the user-end. Also, to attract users' participation, the protocol should be strongly privacy-preserving. Apple and Google [2] jointly provided a contract tracing framework, which was adopted by many contact tracing apps, such as COVID Alert NY in New York and COVID Alert in Canada. However, the framework can only ensure limited privacy protection by relying on temporal identifiers. DP-3T [53] also designed the protocol relying on ephemeral identifiers. In their protocol, users need to download a Cuckoo filter that stores the ephemeral identifiers of the positive diagnosis, which is expensive in communication. Trieu et al. [52] designed three PSI-CA protocols to realize privacy-preserving contact tracing. Their first PSI-CA protocol used the DH-OPRF that is exactly the same as [16]. The other two protocols utilized multi-query Keyword private information retrieval [1] by respectively using one backend server or two non-colluded backend servers to reduce the communication costs. In addition to the above two-party protocols, Duong et al. [20] designed a delegated uPSI-CA protocol by using an oblivious distributed key pseudorandom function (Odk-PRF). Their design is prone to the collusion attack because of using multiple intermediate delegates. By using delegates, their protocol can achieve very low communication cost (i.e., 0.094MB) and computation cost (i.e., 0.002s) when $n_y = 2^{11}$ and $n_x \approx 2^{26}$, which are the lowest compared with the above protocols. However, the communication cost of the backend server on the PHA is very high (i.e., $\geq$ 1036.83 MB), which is very expensive in the real-world application. More details can be found in section 7.

3 Preliminaries

The receiver has a set Y with size n_y. The sender has a set X with size n_x. In addition, $[n]$ denotes a set $\{1, \cdots, n\}$. We also denote $\{x_i\}_n$ as a set $\{x_1, \cdots, x_n\}$. In FHE, a ciphertext is denoted as $[\![\mathbf{b}]\!]$, in which $\mathbf{b}$ is the corresponding batched plaintext. A batched plaintext is a vector containing many single plaintexts. All arithmetic operations in the protocols are in the finite field $\mathbb{F}_t$, where t is the FHE plaintext modulus.

3.1 Security Model

As previous FHE-based protocols (i.e., [12, 15]), in this paper, we consider security against malicious adversaries under a relaxed definition where one corruption case is simulatable and for the other only privacy (formalized through indistinguishability) is guaranteed [24]. In addition, we consider the deception attack. In the deception attack, the sender deceives the receiver to accept an item not in the intersection as a matched one without being noticed by the receiver. In prior FHE-based uPSI protocols [12, 13, 15], the receiver will accept an item if its matching result is '0'; then the malicious sender can deliberately return $[\![0]\!]$ to the receiver even $y \notin X \cap Y$. The receiver can only passively accept y as a matched one. In a special case, the malicious sender can simply return '0' results for all bins. Then the receiver has to accept his/her own set Y as the intersection. To measure the protocol's ability to counter the deception attack, we have attacking success probability which shows the probability of the sender deceiving the receiver, and the detecting probability which shows the probability of the receiver detecting that he/she is deceived.

3.2 Bloom Filter

Bloom Filter [7] is a probabilistic data structure that can profile a set. It maps each item of a set to a hash table with size m_v by using k_v hash functions $h_1(\cdot), h_2(\cdot), \cdots, h_{k_v}(\cdot) : \{0,1\}^* \mapsto [m_v]$. The Bloom Filter is initialized by setting all the bins as '0'. To insert an item e, one should map it to k_v

positions $h_1(e), h_2(e), \cdots, h_{k_v}(e)$ and set the corresponding bins as '1'. To query an item e, one only needs to check all its k_v hashed locations. If they are all '1's, then e is in the set; otherwise not. Though there is no false negative in Bloom Filter, false positives are possible. To insert n items, the false positive probability (FPP) is

$$\varepsilon = (1 - (1 - \frac{1}{m_v})^{k_v n})^{k_v}. \quad (1)$$

There are two differences between Bloom Filter and Cuckoo hashing (in Appendix B). Firstly, the bins in Bloom Filter contain bits instead of items. Secondly, each item in Bloom Filter is mapped to all the k_v bins rather than only one bin.

3.3 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) allows the evaluation of arbitrary functions on encrypted data [21]. Under FHE, one can easily outsource his/her private data to another party with powerful computation capability (e.g., the cloud) by encrypting the data. After computing on the encrypted data by the other party, only the one with the decryption key can decrypt and get the computation result. However, the computation cost for executing homomorphic operations is generally expensive, which makes FHE only practical in limited applications.

To enable cheaper computation, Chen et.al [13] and [12] utilized *leveled* FHE, which is bound by a pre-determined circuit depth. They used a leveled FHE library *SEAL* [51]. *SEAL* allows four basic homomorphic operations over integers:

Add: $[\![\mathbf{p}_1 + \mathbf{p}_2]\!] = [\![\mathbf{p}_1]\!] + [\![\mathbf{p}_2]\!]$,
Add Plain: $[\![\mathbf{p}_1 + \mathbf{p}_2]\!] = [\![\mathbf{p}_1]\!] + \mathbf{p}_2$,
Multiply: $[\![\mathbf{p}_1 \times \mathbf{p}_2]\!] = [\![\mathbf{p}_1]\!] \times [\![\mathbf{p}_2]\!]$,
Multiply Plain: $[\![\mathbf{p}_1 \times \mathbf{p}_2]\!] = [\![\mathbf{p}_1]\!] \times \mathbf{p}_2$,

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are two plaintexts. There are three important parameters in *SEAL*: the plaintext modulus t, the ring dimension n', and the ciphertext modulus q. A freshly generated ciphertext gains some noise budget $\log(2|\mu|)$ to execute FHE operations, where μ is the invariant noise [31]. When the noise grows over the noise budget of a ciphertext, the correctness of the decrypted plaintext is not guaranteed. In addition to the four basic operations, employing *modulus switching* [9] to a ciphertext can effectively reduce the ciphertext modulus q to a smaller q'. Therefore, the size of a resulting ciphertext can be reduced. For more details, the readers are suggested to refer to the work [31] and the source codes [51].

Batching, also known as Single Instruction Multiple Data (SIMD), is a technique that enables operations on two vectors rather than two integers. Batching is integrated into many FHE schemes, such as SEAL [51]. Taking addition operation as an example, given $\mathbf{p}_1 = \langle x_1, x_2, \cdots, x_n \rangle$ and $\mathbf{p}_2 = \langle y_1, y_2, \cdots, y_n \rangle$, one can get $\mathbf{p}_1 + \mathbf{p}_2 = \langle x_1 + y_1, x_2 + y_2, \cdots, x_n + y_n \rangle$ rather than computing $x_i + y_i$ for n times. By utilizing batching, one can have the above basic operations on vectors in FHE, thus greatly saving the computation and communication costs.

Since FHE operations are noise bounded, the receiver cannot simply multiply $[\![\mathbf{y}]\!]$ to get $[\![\mathbf{y}^i]\!]$ ($\forall i \in \{2, 3, \cdots, n_x\}$). Windowing is a technique proposed in CLR'17 [13] to find the minimum window set W, such that all ciphertexts $[\![y^i]\!]$ ($\forall i \in \{1, 2, \cdots, n_x\} \backslash W$) can be recovered by computing on the windowing ciphertexts $[\![\mathbf{y}^i]\!]$ for $\forall i \in W$ within given circuit depth ℓ. Windowing is used to reduce the number of ciphertexts from the receiver to the sender in PSI. For example, to enable the sender get $[\![\mathbf{y}]\!], [\![\mathbf{y}^2]\!], [\![\mathbf{y}^3]\!]$, the receiver only needs to send $[\![\mathbf{y}]\!], [\![\mathbf{y}^2]\!]$ to the sender. The sender then can compute $[\![\mathbf{y}^3]\!] = [\![\mathbf{y}]\!] \times [\![\mathbf{y}^2]\!]$. Cong et al. [15] reduced the windowing as a problem of finding extremal postage-stamp bases [10] and exploited Paterson-Stockmeyer algorithm [40] to save communication costs. It is noted that additive HE cannot be used in windowing because there are ciphertexts multiplications.

3.4 Shuffled DH-OPRF

Oblivious Pseudorandom Function (OPRF) enables the receiver to get the $OPRF_k(y)$ secretly, in which k is the secret key of the sender and y is the input of the receiver. Shuffled Diffie-Hellman-based OPRF (DH-OPRF) [16] enables an additional shuffled feather for the OPRF values. One advantage of the DH-OPRF is that the communication cost can be $O(n_y)$ (i.e., linear with the smaller set size). Specifically, assuming $H_1(\cdot)$ and $H_2(\cdot)$ are random oracles, there are three steps:

1. The receiver generates a random secret β for all items in Y and sends $H_1(y_i)^\beta$ ($\forall i \in [n_y]$) to the sender.
2. The sender generates a random key k. Upon receiving $H_1(y_i)^\beta$, he/she computes $H_1(y_i)^{\beta k}$ ($\forall i \in [n_y]$) and shuffles them to get $H_1(y_{\pi(i)})^{\beta k}$ ($\forall i \in [n_y]$), where $\pi(\cdot)$ is a random permutation. The sender sends the shuffled values to the receiver.
3. The receiver gets its outputs $OPRF_k(y_i) = H_2((H_1(y_{\pi(i)})^{\beta k})^{(1/\beta)}) = H_2(H_1(y_{\pi(i)})^k)$ ($\forall i \in [n_y]$).

Since the sender knows the OPRF key k, he/she can have the OPRF output for any input. To achieve an uPSI-CA protocol, the sender directly computes $U = \{H_2((H_1(x_i)^k)\}_{n_x}$ and sends it to the receiver. Then the receiver gets the intersection cardinality $|X \cap Y| = |U \cap V|$, where $V = \{H_2((H_1(y_{\pi(i)})^k)\}_{n_y}$. The drawback is that the sender needs to send n_x OPRF-processed items to the receiver, which is expensive in communication when n_x is large. In this paper, we utilize the shuffled DH-OPRF to preprocess the items to design our protocols. In [12, 15], the authors also utilized the DH-OPRF [27] to preprocess their items. There are two differences between their used DH-OPRF and the above shuffled DH-OPRF. First, in step 1, the receiver samples n_y random secrets $\{\beta_i\}_{n_y}$. In step 2, the sender does not shuffle the returned values.

By using n_y secrets $\{\beta_i\}_{n_y}$, the protocol is perfectly receiver-private [27]. By using one secret β, one needs DDH assumption to ensure the receiver's privacy [16] in the semi-honest model (i.e., against a semi-honest sender). In [27], to

be secure against a malicious sender, zero-knowledge proof is used to ensure the sender does the computation by using the same k for all $\{H_1(y_i)^\beta\}_{n_y}$. In the preprocessing phase of our uPSI(-CA) protocols, we only need the OPRF protocol to be secure against a semi-honest sender. Therefore, zero-knowledge proof is not needed as [12, 15]. The difference between n_y secrets and one secret only affects the privacy of the receiver. For the sender's privacy, in [16], the authors claimed their protocol was secure against a semi-honest receiver under the DDH assumption in ROM. However, the authors in [27] proved that the protocol was secure against a malicious receiver under the One-More Gap DH assumption in ROM. We also observe a slight difference between [27] and [16] to compute the OPRF value by using $H_2(\cdot)$. In [27], the receiver computes $H_2(H_1(y_i), H_1(y_i)^k)$ for $y_i \in Y$ rather than $H_2(H_1(y_i)^k)$ in [16]. Here we do not consider the shuffling for simplicity. Also, the receiver receives $H_2(H_1(x_i), H_1(x_i)^k)$ for $x_i \in X$ from the sender to finish PSI after DH-OPRF. The authors in [27] explained that they did so simply for better security reduction and it was also secure to compute $H_2(H_1(y_i)^k)$ and $H_2(H_1(x_i)^k)$, which was adopted in [12, 15]. In this paper, we follow [12, 15] to compute $H_2(H_1(y_i)^k)$ and $H_2(H_1(x_i)^k)$.

4 Our Approaches

In this section, we present our techniques to tackle the long item issue and improve performance. Since the item bit length σ has a great impact on the performance, we propose two techniques to turn a long item into several shorter ones.

4.1 Virtual Bloom Filter

To turn a long item into a short one, the existing protocols [12, 42, 45] naively hash it to get a shorter representation with length $\log(n_x) + \log(n_y) + \lambda$. In this paper, we take a different method by utilizing Bloom Filter. Bloom Filter can store each item in its k_v corresponding hashed bit bins. Noticing that the hash table size m_v is much smaller than 2^σ, we can represent a long item as the k_v shorter *indices*. Since we do not need a physical hash table to store the bits, we call it *Virtual* Bloom Filter (VBF). Having VBF, we can encode a long item x as k_v short VBF sub-items (i.e., $x := (h_1(x), h_2(x), \cdots, h_{k_v}(x))$). It is noted that the order of these sub-items does not matter, since all the k_v hash indices are independent. In equation 1, setting $n = n_x + n_y = 2^{24} + 5535$, $\lambda = 40$, and the FPP $\varepsilon \leq 2^{-\lambda}$, we can get different values of m_v for different k_v. The length of the short VBF sub-item $\sigma_1 = \log(m_v)$ is shown as Table 1.

Table 1: The bit length σ_1 of short VBF sub-items with different k_v, when $n = n_x + n_y = 2^{24} + 5535$, $\lambda = 40$.

k_v	2	3	4	5	6	7
σ_1	46	39	36	35	34	33

From table 1, with the increase of k_v, σ_1 decreases as expected; however, the decreasing trend of σ_1 becomes diminishing. For example, to encode an item with $\sigma = 128$, we need $2 \times 46 = 92$ bits for $k_v = 2$ and $3 \times 39 = 117 > 92$ bits for $k_v = 3$. When $k_v \geq 4$, we need at least $4 \times 37 = 148 > 128$, which is expensive. In this paper, we choose $k_v = 2$ to achieve better performance. When setting $k_v = 1$, we simply use the naive hashing strategy as [12, 42, 45].

Both VBF encoding and naive hashing can be combined with permutation-based hashing [3] to further save about $\log_2(m)$ bits per item, where m is the table size. Specifically, to store an item $x = x^{(1)}||x^{(2)}$ into the hash table, one just needs to store $x^{(2)}$ with bit length $\sigma - \log_2(m)$ at bin with index $x^{(1)} \oplus h(x^{(2)})$, where $h(\cdot)$ is a uniform hash function, $x^{(1)}$ is with bit length $\log_2(m)$. To check whether $y = x$, one only needs to find bin $y^{(1)} \oplus h(y^{(2)})$ and compares whether $y^{(2)} = x^{(2)}$. If yes, they are equal; otherwise not. For more discussion about permutation-based hashing, one can refer to [13, 42]. In the above example, by setting $k_v = 2$, one needs to handle $2 \times 46 = 92$ bits per item. By using naive hashing, one only needs to handle 80 bits per item. However, when combined with permutation-based hashing, VBF encoding can save about $k_v \times \log_2(k_v \times 8192) = 2 \times \log_2(2 \times 8192) = 28$ bits; naive hashing can only save $\log_2(8192) = 13$ bits. It is noted that the table size needs to be increased by $k_v\times$ as the number of the VBF sub-items is $k_v\times$ as the number of original items. Therefore, after permutation-based hashing, one only needs to handle $92 - 28 = 64$ bits per item in VBF encoding. In naive hashing, one needs to handle $80 - 13 = 67 > 64$ bits. The main advantage of VBF encoding over the naive hashing is the security benefit by encoding an item into several independent shorter ones. More details can be found in the discussion of the deception attack in subsection 5.4 and 6.3.

4.2 Polynomial Links

Though VBF can turn a long item to k_v short VBF sub-items with bit length σ_1, it is still expensive for FHE operations. For example, when setting $k_v = 2$, the VBF item bit length can be $\sigma_1 = 46$ from Table 1. However, $t \geq 2^{\sigma_1} = 2^{46}$ is still expensive for FHE. In Chen et al. [13], the item bit length is only limited in $\sigma \leq 32$ for performance considerations. Here we propose another optimization method called *polynomial links* (PoL). There are two steps in the PoL: slicing and building links. The general flow is shown in Figure 1.

In the first step, we simply slice a long item x into k_s (default ≥ 2) short slices $x^{(i)}$ with equal bit length σ_2 (i.e., $x = x^{(1)}||x^{(2)}||\cdots||x^{(k_s)}$). After simple slicing, we only need to handle k_s shorter slices than the long item x with bit length σ. However, it brings false positives. For example, by using normal polynomial, to match $y = y^{(1)}||y^{(2)}||\cdots||y^{(k_s)}$ with $x_i = x_i^{(1)}||x_i^{(2)}||\cdots||x_i^{(k_s)}$ $(i = 1, 2 \ldots, b)$, it is possible for the receiver to accept y as a common item by receiving a matching result $y^{(1)} = x_2^{(1)}$, $y^{(2)} = x_1^{(2)}$, $y^{(3)} = x_1^{(3)}$, $\cdots$, $y^{(k_s)} = x_1^{(k_s)}$

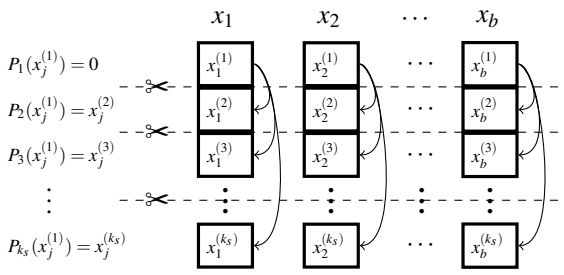

Figure 1: The general flow of the polynomial links.

but $y^{(1)} \neq x_1^{(1)}$. In this example, it is obvious that $y \neq x_1$, nor does it match any of the other items.

The reason why simple slicing causes false positives is that it cuts down the connections between the slices of an item. From the above example, we only know that all slices of y are matched, but not sure if these slices are matched to the same x_1. Therefore, we need to establish such a connection. We utilize interpolation polynomials to accomplish this goal. To build the connections, we take the first slices as roots and the other slices as their corresponding tags to build interpolation polynomials. Specifically, we take $(x_1^{(1)}, x_1^{(j)})$, $(x_2^{(1)}, x_2^{(j)})$, $\cdots$, $(x_b^{(1)}, x_b^{(j)})$ $(j = 2, 3, \cdots, k_s)$ as points to build $k_s - 1$ interpolation polynomials $P_j(\cdot)$ $(j = 2, 3, \cdots, k_s)$. Then taking $y^{(1)}$ as input, the receiver can get $P_j(y^{(1)})$ $(j = 2, 3, \cdots, k_s)$. If $y^{(1)}$ is equal to $x_i^{(1)} \in \{x_1^{(1)}, x_2^{(1)}, \cdots, x_b^{(1)}\}$, we can easily know $P_j(y^{(1)}) = x_i^{(j)}$ $(j = 2, 3, \cdots, k_s)$. Then the receiver can simply check $y^{(j)} = P_j(y^{(1)})$ $(j = 2, 3, \cdots, k_s)$. If the first slice is still equal, then $y = x_i$. To check whether $y^{(1)}$ is in the root set, the receiver still needs to build another normal polynomial $P_1(\cdot)$ by using $\{x_i^{(1)}\}_b$. If $y^{(1)}$ does not match, obviously, y also does not match either.

5 Our uPSI(-CA) Protocols

We classify our protocols into two phases: the *preparation phase* and the *online phase*. In the *preparation phase* in Figure 2, the sender can do the computation without interactions with the receiver. Therefore, the online time can be reduced. In the *online phase* in Figure 3, the receiver and the sender interact to securely get the intersection cardinality.

In the protocols, different steps play different functions. There are two functions for the shuffled DH-OPRF step. First, it can hide the final matched items of the receiver such that the receiver can only get the intersection cardinality rather than the intersection. Second, it hides the sender's input set. After DH-OPRF, all items in $X^o \backslash Y^o$ of the sender are pseudorandom to the receiver. VBF and PoL are used to handle the long item issue in FHE. Cuckoo hashing and simple hashing are used to store the processed items to make bin-wise matching. FHE can ensure the receiver discloses nothing to the sender.

5.1 Construction Details

In addition to the general protocol in Figure 2 and 3, we add some details in the construction, such as permutation-based hashing, the *partitioning* of the simple hashing table for the sender, and inserting spies.

Permutation-based hashing. In step I.(c) and step II.(c), permutation-based hashing can be used to further reduce about $\lfloor \log_2(m) \rfloor$ bits for each VBF sub-item or naively hashed item in both simple hashing and Cuckoo hashing.

Partitioning. In step I.(c), the sender partitions each bin into α sub-bins. There are two functions. First, in PoL, the sender needs to implement polynomial interpolation by taking the first slices as roots, which are supposed to be distinct. When the slice number k_s is large or the number of VBF sub-items in a bin is large, the collision probability for the first slices will be high. To handle this issue, we partition each bin into α sub-bins and assign the VBF sub-item with the same first slice into different sub-bins. The second function is for performance. After partitioning, the highest windowing power is $b = \lceil \frac{B}{\alpha} \rceil$ for the α sub-bins rather than the largest bin size B. The α sub-bins can share the same ciphertexts (i.e., $[\![(\mathbf{y}^{(1)})^1]\!], \cdots, [\![(\mathbf{y}^{(1)})^b]\!]$) after windowing in step II.(d).

Inserting spies. To avoid the deception attack, the receiver can put random dummy items (in the hash output domain) as spies in non-occupied bins of the Cuckoo hashing table in step II.(c). Then in step II.(f), the receiver can first check the results corresponding to the spies. If there is a '0' in these results, the receiver aborts the protocol and refuses to accept the intersection cardinality. It is noted that the number of non-occupied bins in the Cuckoo hashing table is small (i.e., $m - n_y$) and the probability that the inserted spies are in the sender's set is negligible. For example, to ensure negligible collisions, the output bit length of naive hashing should be respectively $\log_2(n_y n_x) + \lambda$ and $\log_2(m n_x) + \lambda$ without and with dummies. In concrete settings, we have $\lceil \log_2(m) \rceil = \lceil \log_2(n_y) \rceil$ (when $n_y = 5535, 11041$) and $\lceil \log_2(m) \rceil = \lceil \log_2(n_y) \rceil + 1$ (when $n_y = 1024, 2048$). All our output bit length settings can cover the bit increase for inserting dummies. Also, inserting dummies is not unusual in uPSI (e.g., [13]).

5.2 Variants and Extensions

The depicted protocol is a general version that uses both VBF and PoL. We define three variants of our uPSI-CA protocols:

1. *VBF+slicing*. In this variant, we combine VBF and simple slicing. We do not build links in I.(d) in this variant and simply compute normal polynomials for all slices. In II.(d), the receiver needs to send the windowing ciphertexts for all slices to the sender. In II.(e), we check the normal polynomial results for all slices. In this variant, the sender has relatively low offline computation costs because of using normal polynomials. However, the communication costs are relatively high. In security,

Input: The receiver and the sender input the set $Y \subset \{0,1\}^\sigma$ and $X \subset \{0,1\}^\sigma$ respectively. The size n_y (resp. n_x) of Y (resp. X) is known to the sender (resp. the receiver). The common inputs are the computational security parameter κ and the statistical security parameter λ.
Output: The receiver outputs $|Y \cap X|$. The sender outputs $\perp$.
Protocol:

I. **Preparation phase.**

(a) **[OPRF]** The sender samples a key k and preprocesses each item x_i as $OPRF_k(x_i)$. We denote $X^o = \{OPRF_k(x_i)\}_{n_x}$.

(b) **[VBF encoding]** The sender and the receiver agree upon the number of uniform hash functions k_v and the virtual hash table size 2^{σ_1} in VBF, making the failure probability in equation (1) negligible (i.e., $2^{-\lambda}$). Then they agree upon the uniform hash functions $h_1(\cdot), \cdots, h_{k_v}(\cdot) : [2^\sigma] \mapsto [2^{\sigma_1}]$. After that, the sender applies VBF encoding to all elements X^o to get the VBF sub-item set X^v. The encoding details go in subsection 4.1.

(c) **[Simple hashing]** The sender and the receiver agree upon the appropriate Cuckoo hash table size m and the number of hash functions (by default as 3) for inserting $k_v n_y$ VBF sub-items with negligible failure probability (i.e., $\leq 2^{-\lambda}$). The uniform hash functions are $g_1(\cdot), g_2(\cdot), g_3(\cdot) : \{0,1\}^{\sigma_1} \mapsto [m]$. The sender hashes his/her $|X^v| = k_v n_x$ VBF sub-items into the 2D simple hash table $\mathcal{T}$ with size m and bin size B. Then the sender partitions each bin $\mathcal{T}_i$ ($i \in [m]$) into α sub-bins $\mathcal{T}_{i,j}$ ($i \in [m], j \in [\alpha]$) and ensures that there are no duplicated first slices in all sub-bins. Here α is no less than the maximum number of VBF sub-items with duplicated first slices in all bins.

(d) **[PoL]** The sender divides each stored VBF sub-item into k_s slices. For each sub-bin $\mathcal{T}_{i,j}$, the sender constructs the normal polynomial $P^{(1)}_{i,j}(x) = c^{(1)}_{i,j,0} + c^{(1)}_{i,j,1}x + \cdots + c^{(1)}_{i,j,b}x^b$ by using the first slices $\mathcal{T}^{(1)}_{i,j}$s. Also, the sender constructs the other interpolation polynomials $P^{(u)}_{i,j}(x) = c^{(u)}_{i,j,0} + c^{(u)}_{i,j,1}x + \cdots + c^{(u)}_{i,j,b-1}x^{b-1}$ ($u = 2,3,\cdots,k_s$) by using points $\{(\mathcal{T}^{(1)}_{i,j}, \mathcal{T}^{(u)}_{i,j})\}_b$.

(e) **[FHE parameters]** The sender and the receiver agree upon the secure FHE parameters including the plain modulus t, the ring dimension n', and the ciphertext modulus q while ensuing $\kappa \geq 128$. The receiver can also generate a secret key for encryption and decryption in advance.

Figure 2: The preparation phase of our uPSI-CA protocol.

the sender is difficult to deceive the receiver; and even though, the receiver is possible to detect the attack. (see subsection 5.4 and Table 4 for more details).

2. *PoL*. In this variant, we take the naive hashing to turn each OPRF value after I.(a) and II.(a) to a shorter bit length (e.g., 80) rather than VBF encoding in I.(b) and II.(b). In this variant, the sender has the highest offline computation costs because of using interpolation polynomials. However, the communication costs are the lowest. In security, this variant can thwart the deception attack (see analysis in subsection 5.4).
3. *VBF+PoL*. In this variant, we keep both VBF (i.e., I.(b), II.(b)) and PoL (i.e., I.(d)) to get a performance balance of the above two variants. In security, this variant also can thwart the deception attack (see subsection 5.4).

To facilitate understanding, we also provide a compact overview of the three variants in Table 2. Our uPSI-CA protocols can be easily modified to their corresponding uPSI protocols. In the shuffled DH-OPRF, the sender can simply not shuffle the returned values to the receiver in step 2.

5.3 Correctness

The core idea of the protocol is bin-wise matching. We simply prove the correctness of our protocols by showing the correctness of VBF and PoL. For a specific bin index, the receiver stores y and the sender stores $\{x_1, x_2, \cdots, x_B\}$. Here we assume all stored items are distinct. To begin with, for the naive normal polynomial based matching, the collision probability is $\frac{B}{2^\sigma}$.

The correctness of matching based on VBF and normal polynomial is obvious. First, the FPP of VBF is the same as the traditional Bloom Filter, which is negligible (i.e., $\leq 2^{-\lambda}$). Second, to match a VBF sub-item of y with B random VBF sub-items, the collision probability is $\frac{B}{2^{\sigma_1}}$. Then for y, the collision probability is $(\frac{B}{2^{\sigma_1}})^{k_v} = \frac{B^{k_v}}{2^{\sigma_1 k_v}}$, which is negligible in our settings. From Table 1, by setting $k_v = 2$ and $\frac{B^{k_v}}{2^{\sigma_1 k_v}} < 2^{-40}$, we can get $B < 2^{26}$, which is far sufficient for all settings. For other k_v in Table 1, we can get similar high bounds (i.e., $B < 2^{25}$). In the above analysis, we do not take partition into the consideration. Partitioning can lower the FPP by constraining $b \leq B$. A similar analysis can be found in subsection 6.3 when we compare our protocols with Cong et al. [15].

For PoL-based matching, the collision probability of the first slice is $\frac{B}{2^{\sigma_2}}$, where σ_2 is the bit length of the slice. Noting that the interpolation polynomial outputs a specific output for y's root, the collision probability for any other slice is $\frac{1}{2^{\sigma_2}}$. Then the collision probability for y is $\frac{B}{2^{\sigma_2}} \times (\frac{1}{2^{\sigma_2}})^{k_s - 1} = \frac{B}{2^\sigma}$, which is the same as the naive normal polynomial. Hence the matching by combining both VBF and PoL has the same

II. **Online phase.**

(a) **[Shuffled DH-OPRF]** The receiver can get $OPRF_k(y_{\pi(i)})$ ($\forall i \in [n_y]$) after the interaction with the sender, where $\pi(\cdot)$ is the sender's secret permutation function. The details are in 3.4. For simplicity, we denote $Y^o = \{OPRF_k(y_{\pi(i)})\}_{n_y}$.

(b) **[VBF encoding]** The receiver applies VBF encoding to all elements in Y^o to get the VBF sub-item set Y^v.

(c) **[Cuckoo hashing]** The receiver inserts the VBF sub-item set Y^V in a Cuckoo hashing table $\mathcal{H}$ by uniform hash functions $g_1(\cdot), g_2(\cdot), g_3(\cdot) : \{0,1\}^{\sigma_1} \mapsto [m]$.

(d) **[FHE with windowing]** The receiver divides each stored VBF sub-item into k_s slices. Denote the stored VBF sub-item $\mathcal{H}_i$ as $\mathcal{H}_i^{(1)} \| \mathcal{H}_i^{(2)} \| \cdots \| \mathcal{H}_i^{(k_s)}$. The receiver first gets the window set W by inputting the circuit depth ℓ and the sub-bin size b. The windowing details are in 3.3. Then the receiver does windowing to compute the batched first slices $\mathcal{H}_i^{(1)}$s with powers in W and get the corresponding ciphertexts (i.e., $[\![(\mathbf{y}^{(1)})^j]\!]$, $\forall j \in W$). Here we denote $\mathbf{y}$ as the batched $\mathcal{H}_i^{(1)}$ ($\forall i \in [m]$). The receiver sends all the $|W|$ windowing ciphertexts for the roots (i.e., $[\![(\mathbf{y}^{(1)})^j]\!]$, $\forall j \in W$) to the sender.

(e) **[Intersect]** Upon receiving $[\![(\mathbf{y}^{(1)})^j]\!]$ ($\forall j \in W$), the sender computes on these ciphertexts and gets all the ciphertexts $[\![(\mathbf{y}^{(1)})^j]\!]$ ($\forall j \in [b]$). Now, the sender starts to compute the encrypted inner product as the result ciphertexts. For the first slice, the sender computes the normal polynomial result:

$$P_j^{(1)}([\![(\mathbf{y}^{(1)}]\!]) = \sum_{v=0}^{b} [\![(\mathbf{y}^{(1)})^v]\!] \times \mathbf{c}_{j,v}^{(1)}.$$

For the other slices, the sender computes the interpolation polynomial results:

$$P_j^{(u)}([\![(\mathbf{y}^{(1)}]\!]) = \sum_{v=0}^{b-1} [\![(\mathbf{y}^{(1)})^v]\!] \times \mathbf{c}_{j,v}^{(u)},$$

for $u = 2, \cdots, k_s$. Finally, the sender optionally does modulus switching to these result ciphertexts to reduce their sizes and returns them to the receiver.

(f) **[Decrypt and get results]** Finally, the receiver decrypts and unbatches the result ciphertexts $P_j^{(1)}([\![(\mathbf{y}^{(1)}]\!])$ ($\forall j \in [\alpha]$) to get the result plaintexts

$$z_{i,j}^{(u)} = P_{i,j}^{(u)}(\mathcal{H}_i^{(1)}), \qquad \forall i \in [m], j \in [\alpha], u \in [k_s].$$

To check whether an item $y \in Y^o$ is in the intersection $X^o \cap Y^o$ or not, the receiver needs to check all its k_v VBF sub-items. If all of them are in $X^v \cap Y^v$, then it is in the intersection. To check whether a VBF sub-item y' of y is in $X^v \cap Y^v$, the receiver first finds its stored bin $\mathcal{H}_{g_*(y')}$, where $g_*(\cdot)$ is the hash function that maps y' to its stored bin. Then he/she checks whether there exists a $j \in [\alpha]$, all the conditions meets: $z_{g_*(y'),j}^{(1)} = 0$, $z_{g_*(y'),j}^{(u)} = y'^{(u)}$ ($u = 2, \cdots, k_s$). If yes, then $y' \in X^v \cap Y^v$; otherwise not. Finally, the receiver gets the intersection cardinality $|X \cap Y| = |X^o \cap Y^o|$.

Figure 3: The online phase of our uPSI-CA protocol.

Table 2: An overview of the three uPSI-CA variants.

Variant	Sender offline computation cost	Communication cost	Deception attack
VBF+slicing	$O(n_x^2)$	$O(k_v k_s \|W\| + k_v k_s \alpha)$	Difficult and detectable
PoL	$O(n_x^2)$ or $O(n_x \log(n_x))$	$O(\|W\| + k_s \alpha)$	Resistant
VBF+PoL	$O(n_x^2)$ or $O(n_x \log(n_x))$	$O(k_v \|W\| + k_v k_s \alpha)$	Resistant

collision probability as the VBF-based one.

5.4 Security

The security of our protocol is guaranteed by FHE and shuffled DH-OPRF. First, in the shuffled DH-OPRF, the sender learns nothing about the receiver's input set. Second, the receiver only sends FHE ciphertexts to the sender and the FHE secret key is held by the receiver. Given the IND-CPA property, the ciphertexts are pseudorandom to the sender. Therefore, the sender cannot learn anything from the receiver.

As for the sender, he/she only returns the polynomial evaluation over the FHE ciphertexts. The roots of the polynomials are preprocessed by the OPRF and are pseudorandom to the

receiver. Therefore, the receiver cannot learn anything about $X^o \backslash Y^o$. Also, the receiver cannot inject more items into the original set to know extra information of X because he/she can only get the OPRF values Y^o for items in Y. For matched items in Y^o, the receiver cannot link them with the items in Y because of the shuffling. For security about the shuffled DH-OPRF, one can refer to [16, 27]. Chen et al. [12] and Cong et al. [15] also used DH-OPRF to preprocess their items. The differences between theirs and ours are just in algorithm aspects. Therefore, our security proof is similar to theirs.

Protection against Deception. Our protocols can inherently thwart the deception attack. In our VBF+slicing based protocol that only uses normal polynomials, to deceive the receiver to accept an item $y \notin X \cap Y$ as a matched one, a malicious sender needs to set y's k_v result ciphertexts as '0's, each of which corresponds to y's one VBF sub-item. The attacking success probability is

$$\frac{n}{m}(\frac{1}{m})^{k_v-1} = \frac{n}{m^{k_v}}. \tag{2}$$

Since the receiver has inserted random spies into the Cuckoo hashing table, he/she can also detect whether the sender is deceiving him/her by checking the spies' corresponding results. For a specific bin, the probability that a random dummy is matched is $\frac{B}{2^{\sigma_1}}$, which is quite low. If a receiver finds a dummy item is matched, he/she has $1-\frac{B}{2^{\sigma_1}}$ confidence that he/she is deceived. Then the detecting probability is

$$1-(\frac{n}{m})^{k_v}. \tag{3}$$

For our PoL based protocol, the malicious sender can only send '0' results for the first slice and can only randomly set results for other slices. The attacking success probability is $\frac{1}{2^{(k_s-1)\sigma_2}}$, which is negligible. For our VBF+PoL based protocol, the malicious sender can only set the first slices of the k_v VBF sub-items as '0', whose attacking success probability is

$$\frac{n}{m}\frac{1}{2^{(k_s-1)\sigma_2}}(\frac{1}{m}\frac{1}{2^{(k_s-1)\sigma_2}})^{k_v-1} = \frac{n}{m^{k_v}2^{(k_s-1)k_v\sigma_2}},$$

which is also negligible. Therefore, in both our PoL based and VBF+PoL based protocols, the attacker will not choose to deceive the receiver and inserting spies is not needed.

6 Experiments and Evaluations

We implement our protocols based on *APSI* [3], an open-sourced library for designing uPSI by FHE. The benchmark machine has 16 cores Intel(R) Xeon(R) Gold 6226R CPU@2.90GHz and 236G RAM. For communication, we simulate the network via the localhost network as CLR'17 [13]. The bandwidth is controlled by *wondershaper* [4].

[3] https://github.com/microsoft/APSI

[4] https://github.com/magnific0/wondershaper

We respectively set the computational and statistical security parameter as $\kappa = 128$ and $\lambda = 40$. For the polynomial interpolation in the sender's setup phase, after comparing the computation costs of four representative interpolation algorithms in Appendix D, we use Newton interpolation when the sub-bin size is small (i.e., $b < 500$) and MB [35] when the sub-bin size is large (i.e., $b \geq 500$). Because of the page limit, we only compare settings for $n_x = 2^{24}, 2^{28}$ in Table 3. For $n_x = 2^{20}$, one can refer to Table 6 in Appendix C.

6.1 VBF+slicing based vs PoL based

As is shown in Table 3, our PoL based protocol gains the advantages of lower communication cost and shorter online time compared with VBF+slicing based protocol. Specifically, VBF+slicing based protocol is $1.60 \sim 2.38\times$ more expensive than PoL based protocol in communication. With the increase of set sizes (i.e., n_x, n_y), more communication costs can be saved. The online time is the querying time of the receiver. The shorter it is, the faster the receiver can get the result. When the sender has a very large set (i.e., $n_x = 2^{28}$), by using PoL based protocol, a receiver with set size $n_y = 2048$ can get the result by taking 24.01s, which saved 70.47% more time than using VBF-slicing based one. The strength of VBF-slicing based protocol is the lower sender offline cost. When $n_x = 2^{28}$ and $n_y = 2048$, the sender in PoL based protocol needs to do $1.47\times$ more offline computation than VBF-slicing based protocol. When $n_x = 2^{24}$ and $n_y = 11041$, the sender offline time of PoL based protocol can be about $7\times$ as expensive as the VBF+slicing based one. Our third protocol (i.e., VBF+PoL based) combines both VBF and PoL to balance the communication cost and sender offline computation cost.

6.2 Comparisons with PSI-CA protocols

Since there are no comparable two-party uPSI-CA protocols, we compare our protocols with the balanced PSI-CA protocol (i.e., Cristofaro et al. [16]) that can achieve the lowest communication cost in Table 3.

Generally, balanced PSI-CA protocols have a communication complexity at least $O(n_x + n_y)$. Therefore, in the unbalanced case, with the increase of n_x, the communication cost also greatly increases. Their advantage is that they can have lower computation costs. As is shown in Table 3, compared with our uPSI-CA protocols, Cristofaro et al. [16] can have faster online time when the bandwidth is high (e.g., ≥ 10 Gbps) and lower sender offline time. However, the communication cost is much higher than ours. When $n_x = 2^{28}$ and $n_y = 1024$, the communication cost of Cristofaro et al. [16] is over $172\times$ as expensive as our protocols. In real-world applications (e.g., contact tracing), a user (i.e., the receiver) needs to pay about 2.5 GB data traffic to make one single query by using their protocol, which is very expensive.

Table 3: The comparison results between our protocols and other protocols in different settings. The last metric "OC&RS" is the offline communication and receiver storage. "-" indicates data unavailable due to long running time. The best results are marked in bold. When $n_x = 2^{28}$, all protocols are tested with 32 threads (each core runs 2 threads) except for LowMC-GC [29] with a single thread. In all other settings, we run the experiments with a single thread. The network latency is 0.05 ms.

Cardinality		**Protocol**	**Communication Cost** (MB)	**Online Time** (s)				**Sender offline (s)**	**OC&RS** (MB)
n_x	n_y			10 Gbps	100 Mbps	10 Mbps	1 Mbps		
2^{28}	2048	Cristofaro et al. [16]	2560.13	10.95	229.73	2220.88	-	**186.614**	0
		LowMC-GC [29]	47.10	**5.35**	13.06	83.21	795.17	1426.39	1072.14
		ECC-NR [29]	-	-	-	-	-	-	-
		Cong et al. [15]	15.09	8.42	8.55	11.14	83.48	1786.89	0
		Ours (VBF+slicing)	14.82	8.69	8.44	10.51	81.30	1788.59	0
		Ours (PoL)	**6.21**	8.27	**8.39**	**8.49**	**24.01**	3635.81	0
		Ours (VBF+PoL)	7.66	8.52	8.75	8.80	39.43	2955.24	0
	1024	Cristofaro et al. [16]	2560.06	11.06	229.64	2220.06	-	**186.55**	0
		LowMC-GC [29]	23.57	**4.64**	8.37	43.49	396.39	1426.83	1072.14
		ECC-NR [29]	-	-	-	-	-	-	-
		Cong et al. [15]	12.56	8.21	8.23	10.50	81.50	2032.99	0
		Ours (VBF+slicing)	12.30	8.20	8.25	10.00	78.75	2067.89	0
		Ours (PoL)	**5.22**	8.00	**7.91**	**8.51**	**24.15**	3297.69	0
		Ours (VBF+PoL)	7.60	8.19	8.34	9.32	37.32	3248.64	0
2^{24}	11041	Cristofaro et al. [16]	160.67	**3.12**	16.34	140.89	1405.54	326.74	0
		LowMC-GC [29]	253.75	12.43	53.46	413.13	4248.98	**85.49**	67.01
		ECC-NR [29]	65.24	4.85	**12.09**	114.12	1154.91	2048.49	67.01
		Cong et al. [15]	14.58	26.78	27.25	27.72	39.12	735.78	0
		Ours (VBF+slicing)	11.76	21.35	21.42	22.01	34.58	660.13	0
		Ours (PoL)	**6.15**	16.43	16.42	**17.22**	**33.65**	5143.96	0
		Ours (VBF+PoL)	8.28	18.31	18.45	18.98	34.21	3034.08	0
	5535	Cristofaro et al. [16]	160.17	2.69	16.34	140.89	1405.54	326.74	0
		LowMC-GC [29]	127.22	5.13	25.82	215.32	2133.24	**87.85**	67.01
		ECC-NR [29]	32.71	**2.64**	**8.99**	86.23	871.78	2418.46	67.01
		Cong et al. [15]	8.49	20.87	20.75	21.11	28.35	769.05	0
		Ours (VBF+slicing)	6.86	16.73	16.83	17.17	25.75	683.48	0
		Ours (PoL)	**4.29**	14.51	14.55	**15.00**	**22.16**	4918.41	0
		Ours (VBF+PoL)	5.04	14.97	15.00	15.29	22.59	2922.14	0

6.3 Comparisons with uPSI protocols

Our protocols can be easily modified to uPSI protocols. Here we also compare our uPSI protocols with the state-of-the-art uPSI protocols in Table 3.

Generally, our protocols outperform LowMC-GC and ECC-NR [29] in all aspects except the sender offline time. In their setup phase, the server (i.e., the sender) needs to profile its set into a Cuckoo filter and send it to the client (i.e., the receiver). This step makes their offline communication costs in linear with the large set size n_x. Also, the client needs to store the received Cuckoo filter. This will put pressure on the clients who are resource constrained, especially when n_x is large. For example, when $n_x = 2^{28}$, the cost is as large as 1072 MB. In terms of security, LowMC-GC and ECC-NR [29] can be modified to be secure against a malicious client by exploiting a malicious OTe [4, 30] in the offline communication phase. Since a malicious sender can return an arbitrary result set to the receiver, both protocols need to assume the sender is semi-honest. Their LowMC-GC protocol can be prone to the deception attack if the sender can arbitrarily craft the garbled circuit. For their ECC-NR, it is non-trivial for the sender to do the deception. In this paper, after the DH-OPRF preprocessing, as previous FHE-based works [12, 15], our uPSI protocols also guarantee security against a malicious receiver and privacy against a malicious sender[5] [24]. Additionally, we consider the receiver deception attack.

Since our protocols are FHE-based, we put our focus on comparing with Cong et. al [15], which has the best performance among existing FHE-based uPSI protocols. We compare our protocols with theirs in receiver performance and security. Additionally, we make comparisons in the false positive in Appendix E. To get their performance data, we directly use the parameters in their source code.

[5] In [12], in addition to showing their formal protocol, the authors also made some discussions about extending their protocol into a fully malicious case in which the sender is allowed to force the receiver to accept a subset of $X \cap Y$ by using circuit leakage.

Table 4: The attacking success probability (SP) and detecting probability (DP) in the deception attack for previous uPSI protocols and our VBF+slicing based uPSI protocol with different receiver set sizes.

SP/DP	$n_y = 1024$	$n_y = 2048$	$n_y = 5535$	$n_y = 11041$
Previous FHE-based [12, 13, 15]	0.5/0.5	0.5/0.5	0.676/0.324	0.674/0.326
Ours (VBF+slicing)	2.441×10^{-4}/0.75	1.22×10^{-4}/0.75	2.062×10^{-5}/0.543	1.028×10^{-5}/0.546

Receiver performance. The naive slicing method in [15] inherently decreases the load factor of the Cuckoo hashing table by k_s times. In another word, for inserting the same number of items, the Cuckoo hashing table has to be increased by $k_s\times$. In consequence, the number of ciphertexts from the receiver to the sender is also increased by $k_s\times$, resulting in worse communication and computation costs for the receiver. Our VBF+slicing based protocol has the same deficiency by enlarging the Cuckoo hashing table by $k_v k_s$ times. In our PoL based protocol, the receiver only needs to encrypt the first slices and send the ciphertexts, while the receiver in [15] needs to encrypt all the k_s slices and send all the ciphertexts. Therefore, the communication costs from the receiver to the sender of our PoL based protocol are generally only $\frac{1}{k_s}$ of theirs. The encryption and decryption time for the receiver also can be cut down by $k_s\times$.

As Table 3 shows, our VBF+slicing protocol has slightly better performances than Cong et al. [15]. However, the security is greatly improved, which will be discussed later. Our PoL based protocol (resp. VBF+PoL based) can save $42.04\% \sim 58.85\%$ (resp. $28.70\% \sim 49.23\%$) communication costs compared with theirs. Also, the receiver can get the querying result more quickly. When the sender set size is very large (i.e., $n_x = 2^{28}$), the receiver set size is small (i.e., $n_y = 2048$), and the bandwidth is low (i.e., 1 Mbps), the receiver in our PoL based protocol can get the result after 24.01s, which is much faster (i.e., $3.48\times$) than theirs.

A deficiency of our PoL based protocol is high interpolation costs for the sender when the sub-bin size is small compared with using normal polynomials in [15]. However, it is worthwhile. In the following section 7, one can find the advantages of our uPSI-CA protocols in real-world applications because the sender offline computation only needs to be done once. For interpolation polynomial, Cong et al. [15] also exploited it. However, they used it to realize their *labeled* uPSI protocol, which is different from uPSI. In *labeled* uPSI, the sender has an item-label set and the receiver has an item set. If an item is matched, its corresponding label will be returned to the receiver; otherwise, a random label will be returned. In their uPSI protocol, they did not use the interpolation polynomial. In this paper, we only focus on uPSI.

Security. None of the previous FHE based uPSI protocols can thwart the deception attack, and neither does Cong et. al [15]. Our protocols can handle this attack almost at no extra cost (see subsection 5.4). The detailed attacking success probability and detecting probability can be calculated by equations 2 and 3 and are listed in Table 4. From this table, we can easily know that our VBF+slicing based protocol can greatly improve the security compared with previous ones. For example, when $n_y = 5535$, the SP of our VBF+slicing based uPSI is only 2.062×10^{-5}, while previous protocols have 0.676 (i.e., $32,784\times$ large); the DP of our VBF+slicing based uPSI is 0.543, while previous ones have 0.324 (i.e., $1.67\times$ small). We do not list our PoL based and VBF+PoL based protocols because the SP in both protocols are negligible.

7 Application: Contact tracing

In our uPSI-CA based privacy-preserving contact tracing system, there are two roles: users and the Public Health Authority (PHA). Each user plays the role of a receiver. The PHA (who owns the backend server) acts as the sender. Users open the Bluetooth of their smartphones and exchange ephemeral identifiers with other users via Bluetooth beacons at a frequency (e.g., 10 minutes) when they are in proximity. The identifiers of infected users will be uploaded to the PHA's backend server. For COVID-19 exposure checking, a user simply makes a query to the PHA by matching his/her contact set Y with the set X of infected identifiers in the backend server.

In the contact tracing system, each user has $14 \times 24 \times 60/10 = 2016$ ephemeral identifiers and receives $n_y = 2^{11}$ identifiers during the past 14 days. The daily positive diagnosis number is 2^{15} and the backend server has a database with size $n_x = 2^{15} \times 2016 \approx 2^{26}$. The settings follow Catalic [20]. In the above sections, we only discuss uPSI-CA between a receiver and a sender. Therefore, we can set the naive hashing output bit length as $\log(n_x) + \log(n_y) + \lambda$ in our PoL-based uPSI-CA protocol, such that the hash collision is negligible (i.e., $< 2^{-\lambda}$). However, it is not enough for real-world deployment. To support real-world deployment, we assume there are $n_1 = 10^7$ users [6] and each user queries $n_2 = 100$ times. To ensure negligible false positives (i.e., a user is not a close contact but detected as one), we set the hash output bit length as $\log_2(n_1 n_2 n_y) + \log_2(n_x) + \lambda \approx 107$. Therefore, even with 10^9 queries, we can still guarantee negligible FPP (i.e. $< 2^{-\lambda}$). We compare our contact tracing design with four representative ones in Table 5 from users' perspective. To motivate users to use the application, security (i.e., linkage attack, collusion attack) and user-end performance (i.e., querying time, computation costs, communication costs) are two crucial factors.

A good contact tracing system first needs to guarantee the privacy of users. However, most of the existing contact

Table 5: Comparing our contact tracing system with other representative ones in a single thread. In two party designs, the bandwidth between the user and the backend server is 10 Mbps. In the delegated design Catalic [20], the bandwidth between the delegates and the user (resp. the backend server) is set as 10 Mbps (resp. 100 Mbps). The network latency is 0.05ms. For Epione [52], data in the left/right are respectively for the cases *not using*/*using* server caching.

Protocols	Linkage attack	Third parties	Query time(s)	User computation(s)	User communication(MB)
Google&Apple [2]	Yes	No	**6.640**	24.322	7.00
DP-3T [53]	Yes	No	387.736	0.384	448.00
Epione [52]	No	No	268.5/140.14	2.088/2.217	226.13/65.65
Ours (PoL)	No	No	60.524	0.366	5.98
Catalic [20]	No	Yes	97.79	**0.002**	**0.094**

tracing protocols simply utilize dynamic or ephemeral identifiers to ensure security, such as Google&Apple [2] and DP-3T [53]. However, the ephemeral identifiers in their designs can be linked with specific persons. In the example of Google&Apple, Alice has a conversation with Bob in a park and records Bob's identifier(s). Later, if Alice makes a query to PHA and finds this identifier is matched, then she will know Bob is infected, which results in the *linkage attack*. This also explains why we cannot simply apply uPSI in contact tracing because Alice will know the matched identifier(s). In our uPSI-CA based protocol, Alice cannot distinguish Bob's identifier from others'. Therefore, Bob's privacy is preserved. In Catalic [20], the authors designed their delegated uPSI-CA protocol. However, they need to involve multiple delegates (i.e., third parties), which brings potential privacy disclosure risks (e.g., collusion attacks). Our uPSI-CA based system can thwart the linkage attack and does not need any third parties, thus providing strong user privacy protection.

A good contact tracing system also needs to have good performance, especially on the user side. We measure the querying time (same as the online time in Table 3), user computation and communication costs in Table 5. In Google&Apple [2], a user needs to download $2^{15} \times 14$ diagnosis keys to generate $2^{15} \times 14 \times 144$ ephemeral identifiers. Each diagnosis key is with bit length 128. Then he/she locally compares his contact list with these derived identifiers. Google&Apple has the shortest querying time compared with other protocols. For DP-3T [53], a user needs to download a Cuckoo filter profiling the backend identifier set and does the match. Therefore, the communication cost of the user can be as high as 448 MB, which is far larger than others. For the single server PSI-CA based contact tracing in Epione [52], the author considered two cases based on whether the backend server caches the previous user queries or not. When the backend server does the caching, a user who queried yesterday only needs to add $2^{11}/14 = 146$ identifiers generated today into the cache, thus saving communication costs.

Catalic [20] has very low user communication and computation costs by shifting the costs from the user to the delegates. Specifically, after inserting the identifiers into a Cuckoo hashing table, the user sends an XOR share of the Cuckoo hash table to each delegate. Then the non-colluded delegates perform the Odk-PRF protocol with the backend server to get result shares containing the intersection cardinality information. One of the delegates combines the result shares and sends the combined information to the user. Finally, the user can get the intersection cardinality. In the above interactions, the expensive cryptographic operations are done by the delegates rather than the user. Therefore, the costs of the user can be very low. From Table 5, the user computation and communication costs of our two-party protocol are respectively about 183× and 64× as expensive as Catalic. However, there is one obvious drawback of Catalic. For a single query from the user, both the delegates and the backend server need to pay at least 1036.83 MB, which is very expensive in real-world applications. To handle 10^9 queries, the communication cost of the backend server is $10^9 \times 1036.83$ MB, which is $10^9 \times (1036.83 - 5.98) = 10^9 \times 1030.85$ MB larger than ours. For the backend server offline computation cost, Catalic takes 1.12 hours, which is 5.9× as cheap as ours.

Our PoL based system gets the lowest user computation and communication costs compared with other two-party designs. One drawback of our system is that PHA needs to pay large offline pre-computation costs in the preparation phase. For a single thread, it takes PHA about 6.6 hours to do the computation; for 32 threads, it takes about 0.49 hours. The sender's computation costs are much larger than Google&Apple [2] (with < 1s), DP-3T [53] (with 1.5s), and Epione [52] (with 233s). However, the offline computation in our system only needs to be done once. Also, compared with the performance advantages, this cost is worthwhile. For example, compared with Google&Apple [2], our PoL based system can save 1 MB per query. For $n_1 n_2 = 10^9$ queries, PHA can save 10^9 MB. For each user, he/she can save $n_2 = 100$ MB. Therefore, our system benefits both the PHA and the users.

8 Conclusion

In this paper, we design efficient uPSI-CA protocols without relying on any third party. These uPSI-CA protocols can be easily modified to uPSI protocols. The core techniques in our protocols are VBF and PoL, which are used to resolve

the long item issue. Not only can our protocols effectively thwart the deception attack, but also have great performance, especially in communication cost and online time. Our protocols are user-friendly and can be used in many applications. By applying our PoL based uPSI-CA protocol in the contact tracing system, we can meet all the security requirements and enable good user performance.

Acknowledgments

This research is supported by the HKU SCF Fintech Academy under grant No. 260910193; by Huawei International Pte. Ltd. under grant No. TC20210528016. We appreciate the constructive reviews from the anonymous reviewers. Specially, we thank the shepherd who proposes many useful suggestions to improve this paper. We also thank Kim Laine for useful discussions.

References

[1] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In *USENIX Security 2021*, pages 1811–1828. USENIX Association, 2021.

[2] Privacy-preserving contact tracing. https://covid19.apple.com/contacttracing, 2020. Accessed: 2022-8-16.

[3] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *FOCS 2010*, pages 787–796. IEEE Computer Society, 2010.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 673–701. Springer, 2015.

[5] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *CCS 2011*, pages 691–702. ACM, 2011.

[6] James Bell, David Butler, Chris Hicks, and Jon Crowcroft. Tracesecure: Towards privacy preserving contact tracing. *CoRR*, abs/2004.04059, 2020.

[7] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[8] Joan Boyar and René Peralta. Concrete multiplicative complexity of symmetric functions. In Rastislav Kralovic and Pawel Urzyczyn, editors, *MFCS 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 179–189. Springer, 2006.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014.

[10] Michael F Challis and John P Robinson. Some extremal postage stamp bases. *Journal of Integer Sequences*, 13(2):3, 2010.

[11] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch OPPRF. *Proc. Priv. Enhancing Technol.*, 2022(1):353–372, 2022.

[12] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *CCS 2018*, pages 1223–1237. ACM, 2018.

[13] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *CCS 2017*, pages 1243–1255. ACM, 2017.

[14] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In Dario Catalano and Roberto De Prisco, editors, *SCN 2018*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.

[15] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS 2021*, pages 1135–1150. ACM, 2021.

[16] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS 2012*, pages 218–231. Springer, 2012.

[17] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, pages 261–278. Springer, 2017.

[18] Sumit Kumar Debnath and Ratna Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *ISC 2015*, pages 209–226. Springer, 2015.

[19] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Function secret sharing for PSI-CA: with applications to private contact tracing. *CoRR*, abs/2012.13053, 2020.

[20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In *Advances in Cryptology - ASIACRYPT 2020*, volume 12493 of *Lecture Notes in Computer Science*, pages 870–899. Springer, 2020.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[22] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[23] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In Juan A. Garay, editor, *PKC 2021*, volume 12711 of *Lecture Notes in Computer Science*, pages 591–617. Springer, 2021.

[24] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography, TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer, 2008.

[25] Yuanyuan He, Xinyu Tan, Jianbing Ni, Laurence T. Yang, and Xianjun Deng. Differentially private set intersection for asymmetrical ID alignment. *IEEE Trans. Inf. Forensics Secur.*, 17:3479–3494, 2022.

[26] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, 2012.

[27] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *SCN 2010*, volume 6280 of *Lecture Notes in Computer Science*, pages 418–435. Springer, 2010.

[28] Bailey Kacsmar, Basit Khurram, Nils Lukas, Alexander Norton, Masoumeh Shafieinejad, Zhiwei Shang, Yaser Baseri, Maryam Sepehri, Simon Oya, and Florian Kerschbaum. Differentially private two-party set operations. In *EuroS&P, 2020*, pages 390–404. IEEE Computer Society, 2020.

[29] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security 2019*, pages 1447–1464. USENIX Association, 2019.

[30] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741. Springer, 2015.

[31] Kim Laine. Simple encrypted arithmetic library 2.3. 1. *Microsoft Research. https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf*, 2017.

[32] Joseph K. Liu, Man Ho Au, Tsz Hon Yuen, Cong Zuo, Jiawei Wang, Amin Sakzad, Xiapu Luo, and Li Li. Privacy-preserving COVID-19 contact tracing app: A zero-knowledge proof approach. *IACR Cryptol. ePrint Arch.*, 2020:528, 2020.

[33] Siyi Lv, Jinhui Ye, Sijie Yin, Xiaochun Cheng, Chen Feng, Xiaoyan Liu, Rui Li, Zhaohui Li, Zheli Liu, and Li Zhou. Unbalanced private set intersection cardinality protocol with low communication cost. *Future Generation Computer Systems*, 102:1054–1061, 2020.

[34] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[35] R. Moenck and Allan Borodin. Fast modular transforms via division. In *SWAT 1972*, pages 90–96. IEEE Computer Society, 1972.

[36] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 557–574. Springer, 2013.

[37] G. Sathya Narayanan, T. Aishwarya, Anugrah Agrawal, Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In *CANS 2009*, pages 21–40. Springer, 2009.

[38] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[39] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *SIGMOD 2017*, pages 775–787. ACM, 2017.

[40] Mike Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[41] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

[42] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security 2015*, pages 515–530. USENIX Association, 2015.

[43] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *Advances in Cryptology - EUROCRYPT 2019*, pages 122–153. Springer, 2019.

[44] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.

[45] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX Security 2014*, pages 797–812. USENIX Association, 2014.

[46] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):7:1–7:35, 2018.

[47] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *CCS 2022*, pages 2505–2517. ACM, 2022.

[48] Amanda Cristina Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In *FC 2018*, volume 10957 of *Lecture Notes in Computer Science*, pages 203–221. Springer, 2018.

[49] Amanda Cristina Davi Resende and Diego de Freitas Aranha. Faster unbalanced private set intersection in the semi-honest setting. *J. Cryptogr. Eng.*, 11(1):21–38, 2021.

[50] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

[51] Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL, March 2022. Microsoft Research, Redmond, WA.

[52] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 43(2):95–107, 2020.

[53] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda Gurses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. Decentralized privacy-preserving proximity tracing. *IEEE Data Eng. Bull.*, 43(2):36–66, 2020.

[54] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4):593–622, 2005.

[55] Haohuang Wen, Qingchuan Zhao, Zhiqiang Lin, Dong Xuan, and Ness Shroff. A study of the privacy of covid-19 contact tracing apps. In *SecureComm 2020*, pages 297–317. Springer, 2020.

[56] Åke Björck and Victor Pereyra. Solution of vandermonde systems of equations. *Mathematics of Computation*, 24(112):893–903, 1970.

A Related Work (others)

In this section, we introduce the other related works about PSI-CA, unbalanced PSI, and contact tracing schemes.

PSI-CA. Freeman et al. [22] showed a linear lower bound for communication overhead for approximating the cardinality of the set intersection. They discussed the multi-party case, which was realized in [37]. Debnath and Dutta [18] also presented their protocol in linear complexity by using Bloom Filter and Goldwasser-Micali (GM) Encryption based on the Quadratic Residuosity (QR) assumption. Davidson and Cid [17] presented a private set union protocol with linear complexity by using Bloom Filter and addictive homomorphic encryption. They claimed their protocol can be adapted to compute PSI-CA with minor changes. All the above protocols are expensive. For example, in [18], even by setting $n_x = 2^{20}$ and $n_y = 5535$, the communication cost is already as high as 7425.59 MB. For the circuit-based PSI, in addition to the protocols [11, 26, 43, 47] we introduce in section 2, there are also others [14, 42, 46, 50]. However, their performance is worse than [47].

Unbalanced PSI. Kales et al. [29] proposed two optimized uPSI protocols LowMC-GC and ECC-NR based on previous protocols. In both of their protocols, there is a setup phase in which the server (acting as the sender) needs to profile a set into a Cuckoo filter and send it to the client (acting as the receiver), which costs $O(n_x)$ in the offline communication and the client needs to prepare the storage for the Cuckoo filter. Pinkas et al. [46] designed a PSI protocol supporting both

balanced and unbalanced cases based on Cuckoo hashing and OT extension. However, their protocol has a communication cost $O(n_x + n_y)$, which is expensive. Aranha and Resende [48] also need the offline communication phase and used the Cuckoo filter as Kales et al. [29]. Their work has an even larger false positive probability. Their later work [49] used another structure called Rank- and Select-based Quotient filter [39] to get a larger compression rate. The expensive offline phase is still needed. When the sender set size $n_x = 10^9$, the offline communication costs can be as large as 7186.19 MB.

Contact tracing. Wen et al. [55] investigated 41 contact tracing Apps and found some of them were vulnerable to fingerprinting and user identity tracking due to the adoption of static identifiers. They suggested using dynamic or temporal identifiers to enhance security. However, releasing personal identifiers brings potential privacy exposure risks, even when the identifiers are dynamic. Because these identifiers are linked with both users' temporal and spatial information, which can be private. Bell et al. [6] leveraged secret sharing and additive homomorphic encryption to hide the infected status of users. Though they claimed they guaranteed privacy from contacts, they cannot because they do not protect the temporal identifiers. Liu et al. [32] designed a privacy-preserving protocol by using zero-knowledge proof to convince a doctor that a user is a close contact but does not disclose his/her information. One deficiency of their approach is that users need to download all close contact information from a bulletin board and make comparisons by themselves. Dittmer et al. [19] exploited Function Secret Sharing to realize PSI-CA. In their design, they need two non-colluding backend servers as Trieu et al. [52]. However, it is impractical to assume a party has 2 non-colluded backend servers.

B Cuckoo Hashing

Cuckoo hashing [38] enables inserting each item e of a set into a bin of a hash table with size m by using $k > 1$ hash functions $g_1(\cdot), g_2(\cdot), \cdots, g_k(\cdot) : \{0,1\}^* \mapsto [m]$. Specifically, to insert an item e into a bin, one first maps it to location $g_i(e)$ where i is randomly drawn from $[k]$. If this location is occupied by e', then kick out e' from the occupied bin and insert e; otherwise, simply insert e. Denoting the hash index of e' in the evicted location is i', then map e' to location $g_j(e')$ where $j \in [k] \backslash \{i'\}$ and does the eviction again. The evicting process is executed recursively until an empty bin is found to insert the last evicted item. When there is an evicting loop (i.e., no empty bin can be found), the common practice is to put an item in the loop into an extra data structure *stash* to break the loop. In [13], the authors found Cuckoo hashing with $k = 3$ performed very well with even no stash. To achieve the statistical security level $\lambda = 40$, they claimed that one could insert 5535 (resp. 11041) items into a hash table with size 8192 (resp. 16384). The corresponding load factor of the hash table is approximately 67%. To query an item e, one just needs to check all the k bins $g_1(e), g_2(e), \ldots, g_k(e)$.

C Performance comparisons when $n_x = 2^{20}$

The performance of different protocols when $n_x = 2^{20}$ is as shown in Table 6. From this table, one can find similar comparison results as $n_x = 2^{24}, 2^{28}$ in Table 3. Our PoL based uPSI(-CA) protocol still gains the lowest communication costs when $n_y = 5535, 11041$. When the bandwidth is low (e.g., ≤ 10 Mbps), it runs the fastest online. Our VBF-based protocol gains slight communication and online time advantages over the Cong et. al. [15]. For the sender offline computation, LowMC-GC is still the fastest one. Our VBF-based protocol gains lower sender offline time than the PoL based one. Our VBF+PoL based protocol can get a performance balance between the PoL based one and VBF based one.

D Interpolation algorithms selection

In the setup phase of our protocols, the sender needs to do polynomial interpolation for the sub-bins to get the corresponding polynomial coefficients, which accounts for most of his/her setup time. There are many algorithms for computing the interpolation polynomials, such as the Newton polynomial with computation complexity $O(b^2)$. In 1970, Björck and Pereyra (BP) [56] developed an algorithm with $O(b^2)$. Later in 1972, Moenck and Borodin (MB) [35] designed a cheaper one with $O(b\log(b))$, which is the one used in the PSI protocol *Spot-Light* [41]. We also find another algorithm with $O(b^2)$ in NTL[6], a popular library for doing number theory. We implement these algorithms and compare the running time costs in Figure 4. From Figure 4, we have two find-

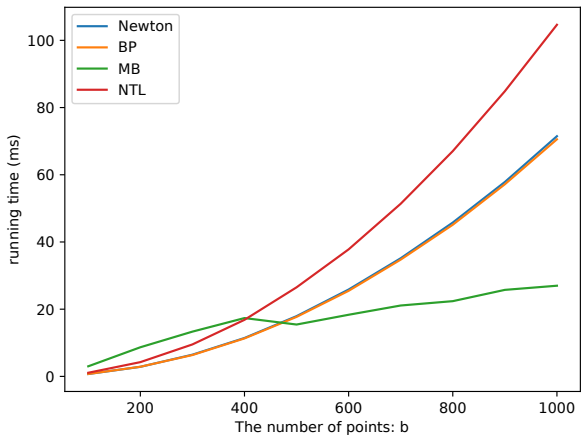

Figure 4: The computation costs for interpolating b points by using different algorithms.

[6] https://libntl.org/

Table 6: The comparison results between our protocols and other protocols when $n_x = 2^{20}$. The last metric "OC&RS" is the offline communication and receiver storage. The best results are marked in bold. All protocols are tested with a single thread. The network latency is 0.05 ms.

Cardinality		**Protocol**	**Communication Cost** (MB)	**Online Time** (s)				**Sender offline (s)**	**OC&RS** (MB)
n_x	n_y			10 Gbps	100 Mbps	10 Mbps	1 Mbps		
2^{20}	11041	Cristofaro et al. [16]	10.67	**1.05**	**1.98**	10.24	98.70	20.31	0
		LowMC-GC [29]	253.75	11.61	52.79	430.21	4359.08	**5.48**	4.19
		ECC-NR [29]	65.24	4.52	6.77	60.20	609.01	151.05	4.19
		Cong et al. [15]	8.95	4.96	5.06	5.61	26.33	26.53	0
		Ours (VBF+slicing)	8.84	5.01	5.09	5.64	25.18	26.56	0
		Ours (PoL)	**4.50**	3.85	3.99	**4.66**	**24.46**	88.04	0
		Ours (VBF+PoL)	6.26	4.32	4.43	5.07	25.11	49.42	0
	5535	Cristofaro et al. [16]	10.34	**0.62**	**1.15**	9.51	91.74	20.36	0
		LowMC-GC [29]	127.22	4.89	25.56	215.23	2136.56	**5.35**	4.19
		ECC-NR [29]	32.71	2.35	3.64	31.93	323.71	150.67	4.19
		Cong et al. [15]	5.40	3.64	3.67	3.95	22.92	26.21	0
		Ours (VBF+slicing)	5.30	3.63	3.68	3.94	18.68	26.27	0
		Ours (PoL)	**3.13**	3.15	3.21	**3.63**	**18.30**	93.09	0
		Ours (VBF+PoL)	3.85	3.25	3.28	3.64	18.45	50.25	0

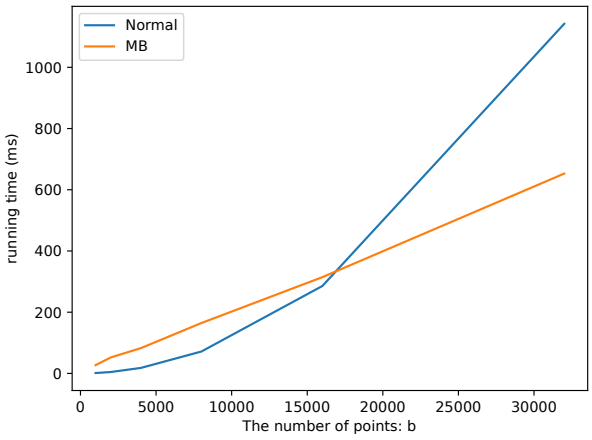

Figure 5: The computation costs for building a normal polynomial and an interpolation polynomial with b points.

ings. First, it is surprising that the BP [56] and Newton [56] have highly similar performance, though they are different algorithms. Second, no perfect interpolation algorithm can outperform others in all numbers of point settings. When b is small (e.g., $b < 500$), BP [56] and Newton [56] run the fastest. When b is large (e.g., $b \geq 500$), MB [35] with better time complexity runs the fastest. Therefore, for a protocol with small sub-bin sizes, we suggest using BP [56] or Newton [56]; for those with large sub-bin sizes, MB [35] is suggested.

In addition to comparing the interpolation polynomials, we also compare MB [35] with the normal polynomial. When building a normal polynomial, we only use the data in the x-coordinate of the b points. As is shown in Figure 5, the interpolation cost can be lower than building a normal polynomial after a threshold (e.g., $b > 18000$). Since MB [35] is with computation complexity $O(b\log(b))$ and a normal polynomial is with $O(b^2)$, this result is not without expectation.

E False positive comparisons with [15]

To make the bin-wise comparison, Cong et. al. [15] has a false positive probability (FPP) $1-(1-(\frac{b}{2^{\sigma_2}})^{k_s})^{\alpha}$. In their work, they do not consider α and simply take $(\frac{b}{2^{\sigma_2}})^{k_s}$ as the FPP, which is inaccurate. A high FPP impairs the correctness of the protocol. In their work, the parameters need to be selected carefully. The basic idea of handling a long item is to make it into shorter ones. Then k_s is supposed to be large to make σ_2 small. However, from their FPP, there is an interesting finding: given parameters b, α, and a fixed FPP (e.g., 2^{-40}), when we increase k_s, σ_2 also needs to be increased. Therefore, it is not suggested to simply select a large k_s, which may result in performance deterioration because the total number of bits to be handled per item increases. Their work needs to consider both the false positive and the performance issues. Since there is no false positive concern in PoL based and VBF+PoL based protocols, we can simply put our focus on improving the performances. It is noted that the authors [15] do not use permutation-based hashing because it only brings a marginal gain in performance in their work. Same as [15], we also do not use permutation-based hashing in our PoL based protocol to show its strength. Without using permutation-based hashing, we set $k_s = 4$ in PoL to make the slice bit length in the twenties (i.e., $20 \sim 29$) after naive hashing. If using permutation-based hashing, the performance can be improved.