Fast *Unbalanced* Private Set Union Based on Leveled Fully Homomorphic Encryption

No Author Given

No Institute Given

Abstract. Private set union (PSU) allows two parties to compute the union of their sets without revealing anything except the union. However, most PSU protocols have been designed for the balanced case, where the two sets are rough of equal size, and their communication cost scales at least linearly with the size of the larger set.

In this paper, we use the leveled fully homomorphic encryption to construct a fast *unbalanced* PSU protocol with a small communication overhead that is secure against semi-honest adversaries. Our protocol has communication complexity linear in the size of the smaller set, and logarithmic in the larger set. More precisely, if the set sizes are $|Y| \ll |X|$, our protocol achieves a communication overhead of $O(|Y| \log |X|)$.

1 Introduction

PSU is a cryptographic technique that allows two parties holding sets X and Y respectively, to compute the union $X \cup Y$, without revealing anything else, namely what are the items in the union of X and Y. Recently, some work has been made on PSU, which has become considerably efficient and been deployed in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data [21,17,20], private ID [13] etc. However, most of the works on PSU are designed in the balanced case. These protocols typically perform only marginally better when one of the sets is much smaller than the other. In particular, their communication cost scales at least linearly with the size of the larger set.

The unbalanced PSU (uPSU) can be seen as a special case of PSU where (1) the receiver's set is significantly smaller than the sender's, and (2) the receiver (with the smaller set) has a low-power device. Chen et al. [8,6] first consider unbalanced private set operation and design an efficient private set intersection (PSI) based on the leveled fully homomorphic encryption (FHE) with communication complexity linear in the size of the smaller set, and logarithmic in the larger set, which breaks the bound of communication complexity linear in the size of the larger set. However, they only realize unbalanced PSI without considering the construction of the uPSU. Jia et al. [19] give a construction of uPSU, but their protocol still requires at least *linearly* with the size of the *larger* set. Therefore, how to design a fast uPSU, which breaks the bound of communication complexity linear with the size of the larger set is an open problem. Based on the above discussions, we ask the following natural question:

Is it possible to design a fast unbalanced PSU protocol which breaks the bound of communication complexity linear with the size of the larger set?

1.1 Contributions and Roadmap

In this paper, we give an affirmative answer to above question. We construct a fast unbalanced PSU protocol based on the leveled fully homomorphic encryption which has communication complexity linear in the size of the smaller set, and logarithmic in the larger set. In detail, our contribution and roadmap can be summarized as follows:

1. We first give a basic uPSU protocol based on fully homomorphic encryption with communication linear in the smaller set, achieving optimal communication that is on par with the naive solution. However, the basic protocol requires a high computational cost and a deep homomorphic circuit.
2. Then, we use an array of optimizations following [8] to significantly reduce computational cost and the depth of the homomorphic circuit, while only adding a logarithmic overhead to the communication. But using the optimized technique of [8] directly will lead to information leakage. That is, the sender could know that some of its subsets have the items in the intersection.
3. Next, we use permute and share technique [13,19] to fix above problem of information leakage and give a full fast uPSU protocol with communication complexity linear in the size of the smaller set and logarithmic in the larger set that is secure against semi-honest adversaries.
4. Finally, we realize our fast uPSU protocol.

1.2 Related Works

The balanced PSU. Kolesnikov et al. [20] propose a PSU protocol based on the reverse private membership test (RPMT) and oblivious transfer. In RPMT, the sender with input x interacts with a receiver holding a set Y, and the receiver can learn a bit indicating whether $x \in Y$, while the sender learns nothing. After that, both parties run OT protocol to let the receiver obtain $\{x\} \cup X$. RPMT requires $O(n \log^2 n)$ computation, and $O(n)$ communication. If the size of the sender's set is $|X| = n$, for computing the set union, the protocol runs RPMT n times independently, which requires $O(n^2)$ communication and $O(n^2 \log^2 n)$ computation. By using the bucketing technique, two parties can hash each set X or Y in m bin, each bin consists of ρ items. Computing (n, n)-PSU is changed into computing m (ρ, ρ)-PSU. The complexity can be reduced to $O(n \log n)$ communication and $O(n \log n \log \log n)$ computation. However, the bucketing technique leads to information leakage about the items in the intersection. In the ideal (n, n)-PSU, from the view of receiver, any item in Y could be an item in $X \cap Y$. But in each (ρ, ρ)-PSU, the receiver can know that the subsets with size ρ have items in $X \cap Y$.

Garimella et al. [13] give a PSU protocol based on oblivious switching and oblivious transfer. They first propose the permuted characteristic functionality based oblivious switching, in which the sender inputs the set $X = \{x_1, \cdots, x_n\}$ and get a random permutation π, the receiver inputs the set $Y = \{y_1, \cdots, y_n\}$ and gets a vector $\mathbf{e} \in \{0,1\}^n$, where $e_i = 1$ if $x_{\pi(i)} \in Y$, else, $e_i = 0$. Then two parties run OT to let the receiver obtain the set union. Their protocol requires $O(n \log n)$ communication and $O(n \log n)$ computation.

Jia et al. [19] propose a PSU with the shuffling technique and oblivious transfer. They use cuckoo hash technique to hash receiver's set Y into m bins and each bin consists of one item, and hash sender's set X into m bins and each bin consists of ρ items. And then, they use shuffling technique to permute and share receiver's bins, in which the sender inputs a permutation π and get the shuffled shares $\{s_{\pi(1)}, \cdots, s_{\pi(m)}\}$, and the receiver inputs its bins $\{a_1, \cdots, a_m\}$ and gets another shuffled shares $\{s_{\pi(1)} \oplus a_{\pi(1)}, \cdots, s_{\pi(m)} \oplus s_{\pi(m)}\}$. That is, for same bin i, if $x_{\pi(i)} \oplus s_{\pi(i)} = s_{\pi(i)} \oplus a_{\pi(i)}$, the hash pre-image of $x_{\pi(i)}$ belongs to Y. Then, the sender and receiver run multi-point oblivious PRF to compute all PRF values of $x_{\pi(i)} \oplus s_{\pi(i)}$ and $s_{\pi(i)} \oplus a_{\pi(i)}$, and for each bin, the sender sends its PRF values to receiver. And the receiver can test whether the item belongs to the union. Finally, two parties runs OT protocol to let the receiver the set union. Their protocol requires $O(n \log n)$ communication and $O(n \log n)$ computation.

Zhang et al. [29] recently give a generic framework of PSU based on multi-query reverse private membership test (mq-RPMT) and oblivious transfer. In the mq-RPMT, the sender inputs $X = \{x_1, \cdots, x_{n_x}\}$ and get nothing, and the receiver inputs $Y = \{y_1, \cdots, y_{n_y}\}$ and gets a bit vector $\mathbf{b} \in \{0,1\}^{n_x}$, satisfying $b_i = 1$ if and only if $x_i \in Y, i \in [n_x]$. And then two parties runs OT protocol to let the receiver the set union. They give two concrete PSU protocols based on symmetric-key encryption and general 2PC techniques, and re-randomizable public-key encryption techniques respectively. Both constructions lead to PSU with linear computation $O(n)$ and communication $O(n)$ complexity.

The unbalanced PSI/PSU. To our knowledge, the first unbalanced PSI is proposed by Chen et al. [8]. The unbalanced PSI of [8] is based on FHE. They first give a strawman protocol in which the receiver encrypts each item y_i in Y, and send the ciphertexts c_i to the sender; the sender chooses random non-zero plaintext item r_i, and homomorphically computes $r_i \Pi_{x \in X}(c_i - x)$, and returns to the receiver; the receiver can decrypt each ciphertext: if $r_i \Pi_{x \in X}(c_i - x) = 0$, it gets $y_i \in X \cap Y$ else, it gets a random item. The protocol requires communication linear in the smaller set, but it requires a high computational cost and a deep homomorphic circuit. And then they use cuckoo hashing, batching, windowing, partitioning, modulus switching, etc to optimize the strawman protocol and give a fast unbalanced PSI. Furthermore, Chen et al. [6] and Cong et al. [9] based on the above framework and give fast labeled unbalanced PSI.

Jia et al. [19] considers uPSU with shuffling technique. For the size of sender's set $m = |X|$ is smaller than the receiver's $n = |Y|$, the receiver chooses the permutation π and the sender inputs its hash bins. Thus the permute and share phase require the communication $O(m \log m)$. However, in the later phase, the receiver needs to send all PRF values of its large set to the sender, for the sender to test whether the item of each bin belongs to the union. Therefore, their protocol requires $O(n + m \log m)$ communication which is at least **linear** with the size of large set.

2 Preliminaries

2.1 Notation

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \cdots, n\}$. 1^λ denotes the string of λ ones. We use κ and λ to denote the computational and statistical security parameters, respectively. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. We denote a probabilistic polynomial-time algorithm by PPT. If S is a set then $s \leftarrow S$ denotes the operation of sampling an item s of S at random. For any permutation π of a set, we set $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\} = \pi(\{s_1, s_2, \cdots, s_n\})$. We denote the parties as the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, and their respective input sets as X and Y, set sizes $|X|$ and $|Y|$. In the unbalanced setting, we assume that $|Y| \ll |X|$.

2.2 Private Set Union

We review the ideal functionality of PSU in Figure 1.

Parameters: Two parties: the sender $\mathcal{S}$ with set X and receiver $\mathcal{R}$ with set Y.
Functionality:

1. Wait for an input $X = \{x_1, x_2, \cdots, x_n\} \subset \{0,1\}^*$ from sender $\mathcal{S}$, and an input $Y = \{y_1, y_2, \cdots, y_m\} \subset \{0,1\}^*$ from receiver $\mathcal{R}$.
2. Give output $X \cup Y$ to the sender $\mathcal{S}$.

Fig. 1: Ideal functionality $\mathcal{F}_{\text{PSU}}$ for private set union with one-sided output

2.3 Hashing

As mentioned in [8], two parties hash the items in their sets into two hash tables using some agreed-upon deterministic hash function, and they only need to perform a PSI for each bin, since items in different bins are necessarily different. We show that these optimization techniques can also be used in PSU, and review them here.

Simple Hashing. There are h hash functions $H_1, \cdots, H_h : \{0,1\}^* \to [m]$ used to map n items into m bins $\mathbf{B}_1, \cdots, \mathbf{B}_m$. Following [24], the maximum bin size B can be set to ensure that no bin will contain more than B items except with probability $2^{-\lambda}$ when hashing n items into m bins.

$$\Pr[\exists \text{ bin size} > B] \leq m \left[\sum_{i=B+1}^{n} \binom{n}{i} \cdot \left(\frac{1}{m}\right)^{i} \cdot \left(1 - \frac{1}{m}\right)^{n-i} \right]$$

Cuckoo hashing. Cuckoo hashing [25,10,12] can be used to build dense hash tables by many hash functions. There are h hash functions $H_1, \cdots, H_h$ used to map n items into $m = \epsilon n$ bins and a stash, where each bin at most one item. For an item x, we choose a random index i from $[h]$, and insert the tuple (x, i) at location $H_i(x)$ in the table. If this location was already occupied by a tuple (y, j), we replace (y, j) with (x, i), choose a random j' from $[h] \backslash \{j\}$, and recursively re-insert (y, j') into the table. The above procedure is repeated until no more evictions are necessary, or until the number of evictions has reached a threshold. In the latter case, the last item will be put in the stash. According to the analysis in [26], we can adjust the values of m and ϵ to reduce the stash size to 0 while achieving a hashing failure probability of 2^{-40}.

Note that following [8], we also let the receiver perform cuckoo hashing with $m \geq |Y|$ bins. The sender inserts each of its items into a two-dimensional hash table using all h hash functions $H_1, \cdots, H_h$, because there is no way for it to know which one of the hash functions the receiver eventually ended up using for the items.

Permutation-based hashing. The permutation-based hashing [1] is an optimization to reduce the length of the items stored in the hash tables by encoding a part of an item into the bin index. For simplicity, we assume m is a power of two. To insert a bit string x into the hash table, we parse it as $x_L || x_R$, where the length of x_R is equal to $\log_2 m$. The hash functions $H_1, \cdots, H_h$ are used to construct location functions as

$$\mathsf{Loc}_i(x) = H_i(x_L) \oplus x_R, 1 \leq i \leq h$$

Instead of inserting the entire tuple (x, i) into the hash table, we only insert (x_L, i) at the location specified by $\mathsf{Loc}_i(x)$. If $(x_L, i) = (y_L, j)$ for two items x and y, then $i = j$ and $x_L = y_L$. If in addition these are found in the same location, then $H_i(x_L) \oplus x_R = H_j(y_L) \oplus x_R = H_j(y_L) \oplus y_R$, so $x_R = y_R$, and hence $x = y$. The lengths of the strings stored in the hash table are thus reduced by $\log_2 m - \lceil \log_2 h \rceil$ bits. The hashing routine is specified in Figure 2.

2.4 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) [14] is a form of encryption schemes that allow arbitrary operations to be performed on encrypted data without requiring

Parameters: Two parties: the sender $\mathcal{S}$ with set X and receiver $\mathcal{R}$ with set Y. Both sets consist of bit strings of length σ. Number of bins m, and each bin size B. A set of hash functions $H_1, H_2, \cdots, H_h : \{0,1\}^{\sigma - \log_2 m} \to \{0,1\}^{\log 2m}$. The location functions Loc_i is defined with respect to H_i for $i \in [h]$.
Input: The sender inputs set X; The receiver inputs set Y.
Output:

1. [**Sender**] Let $\mathbf{B}_x$ be an array of m bins, each with capacity B, and value $\{(\perp, \perp)\}^B$. For each $x \in X$ and $i \in [h]$, the sender samples $j \leftarrow [B]$ s.t. $\mathbf{B}_x[\mathsf{Loc}_i(x)][j] = \perp$, and sets $\mathbf{B}_x[\mathsf{Loc}_i(x)][j] = (x_L, i)$. If the sampling fails due to a bin being full, the sender outputs $\perp$. Otherwise it outputs $\mathbf{B}_x$.
2. [**Receiver**] Let $\mathbf{B}_y$ be an array of m bins, each with capacity 1, and value $(\perp, \perp)$. For each $y \in Y$, the receiver
   (a) sets $w = y$, and $i \leftarrow [h]$;
   (b) defines and calls the function $\mathsf{Insert}(w, i)$ as follows: swap (w, i) with the entry at $\mathbf{B}_y[\mathsf{Loc}_i(w)]$. If $(w, i) \neq (\perp, \perp)$, recursively call $\mathsf{Insert}(w, j)$, where $j \leftarrow [h] \backslash \{i\}$.

   If for any $y \in Y$ the recursive calls to Insert exceeds the system limit, the receiver halts and outputs $\perp$. Otherwise it outputs $\mathbf{B}_y$.

Fig. 2: Hashing routine

access to the decryption key. For improved performance, the encryption parameters are typically chosen to support only circuits of a certain bounded depth (leveled fully homomorphic encryption), and we use this in our implementation following [8,6,9]. There are several FHE implementations that are publicly available. We use the homomorphic encryption library SEAL, which implements the variant of [2] of the Brakerski/FanVercauteren (BFV) scheme [11]. The core parameters of the BFV scheme are three integers: n, q, and t.

We also need some optimization techniques of FHE following [8], such as batching, windowing, partitioning, modulus switching, etc, and review them here.

Batching. Batching is a well-known and powerful technique in fully homomorphic encryption to enable SIMD (Single Instruction, Multiple Data) operations on ciphertexts [15,3,28,7,16]. The batching technique allows the sender to operate on n items from the receiver simultaneously, resulting in n-fold improvement in both the computation and communication. Since in typical cases n has size of several thousand, this results in a significant improvement over the basic protocol.

Windowing. We use a standard windowing technique to lower the depth of the arithmetic circuit that the sender needs to evaluate on the receiver's homomorphically encrypted data, resulting in a valuable computation-communication trade-off.

If the sender only has an encryption of y, it needs to compute at worst the product $y^{|X|}$, which requires a circuit of depth $\lceil \log_2(|X|+1) \rceil$. If the receiver sends encryptions of extra powers of y, the sender can use these powers to evaluate the same computation with a much lower depth circuit. More precisely, for a window size of l bits, the receiver computes and sends $c(i,j) = \mathrm{FHE}.\mathsf{Enc}(y^{i \cdot 2^{lj}})$ to the sender for all $1 \le i \le 2^l - 1$, and all $0 \le j \le \lfloor \log_2(|X|)/l \rfloor$. For example, when $l = 1$, the receiver sends encryptions of $y, y^2, y^4, \cdots, y^{2^{\lfloor \log_2 |X| \rfloor}}$. This technique results in a significant reduction in the circuit depth. To see this, we write

$$r + \Pi_{x \in X}(y - x) = r + a_0 + a_1 y + a_2 y^2 + \cdots + a_{|X|-1} y^{|X|-1} + y^{|X|}.$$

The cost of windowing is in increased communication. The communication from the receiver to the sender is increased by a factor of $(2^l - 1)(\lfloor \log_2(|X|)/l \rfloor + 1)$, and the communication back from the sender to the receiver does not change.

Partitioning. Another way to reduce circuit depth is to let the sender partition its set into α subsets. In the basic protocol, this reduces sender's circuit depth from $\lceil \log_2(|X|+1) \rceil$ to $\lceil \log_2(|X|/\alpha + 1) \rceil$, at the cost of increasing the return communication from sender to receiver by a factor of α. In the PSU, the sender needs to compute encryptions of all powers $y, \cdots, y^{|X|}$ for each of the receiver's items y. With partitioning, the sender only needs to compute encryptions of $y, \cdots, y^{|X|/\alpha}$, which it can reuse for each of the α partitions. Thus, with both partitioning and windowing, the sender's computational cost reduces by a factor of α.

Modulus Switching. We can employ modulus switching [4], which effectively reduces the size of the response ciphertexts. Modulus switching is a well-known operation in lattice-based fully homomorphic encryption schemes. It is a public operation, which transforms a ciphertext with encryption parameter q into a ciphertext encrypting the same plaintext, but with a smaller parameter $q' < q$. As long as q' is not too small, correctness of the encryption scheme is preserved. This trick allows the sender to "compress" the return ciphertexts before sending them to the receiver. Note that the security of the protocol is trivially preserved as long as the smaller modulus q' is determined at setup.

2.5 Oblivious Transfer

Oblivious Transfer (OT) [27] is a ubiquitous cryptographic primitive and is a foundation for almost all efficient secure computation protocols. In OT, a sender with two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. The result is that the receiver learns x_b without learning anything about x_{1-b}, while the sender learns nothing about b. We define the generalized primitive of 1-out-of-2 OT as follows.

Parameters: Two parties: the sender $\mathcal{S}$ and receiver $\mathcal{R}$.
Functionality $\mathcal{F}_{\mathbf{OT}}$:

1. Wait for input $\{x_0, x_1\}$ from $\mathcal{S}$; Wait for input $b \in \{0, 1\}$ from $\mathcal{R}$;
2. Give x_b to $\mathcal{R}$.

Fig. 3: 1-out-of-2 oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$

2.6 Permute and Share

We recall the permute and share functionality $\mathcal{F}_{ps}$ defined in Figure 4. Roughly speaking, P_1 possesses a set $X = \{x_1, \cdots, x_n\}$ of size n and P_2 picks a permutation π on n items. The goal of $\mathcal{F}_{ps}$ is to let P_1 learn the shares $\{s_{\pi(1)},\ s_{\pi(2)},\ \cdots,\ s_{\pi(n)}\}$ and P_2 learn nothing but the other shares $\{x_{\pi(1)} \oplus s_{\pi(1)}, x_{\pi(2)} \oplus s_{\pi(2)}, \cdots, x_{\pi(n)} \oplus s_{\pi(n)}\}$. As mentioned in [5], some earlier works [18,23] can also be used for securely realizing $\mathcal{F}_{ps}$. These solutions all have computation/communication complexity $O(n \log n)$.

Parameters: Two parties: the sender $\mathcal{S}$ and receiver $\mathcal{R}$; Set size n for $\mathcal{S}$; Length of item l.
Functionality $\mathcal{F}_{\mathbf{ps}}$:

1. Wait for input $X = \{x_1, \cdots, x_n\}$ from $\mathcal{S}$, abort if $|X| \neq n$, or $\exists\ x_i \in X$ such that $|x_i| > l$; Wait for input a permutation π from $\mathcal{R}$, abort if π is not a permutation on n items;
2. Give output shuffled shares $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\}$ to $\mathcal{S}$, and another shuffled shares $\{x_{\pi(1)} \oplus s_{\pi(1)}, x_{\pi(2)} \oplus s_{\pi(2)}, \cdots, x_{\pi(n)} \oplus s_{\pi(n)}\}$ to $\mathcal{R}$.

Fig. 4: Permute and share functionality $\mathcal{F}_{\mathrm{ps}}$

3 The Basic Protocol

We describe our basic uPSU protocol in Figure 5 as a strawman protocol. The receiver encrypts each of its items y, and sends them to the sender. For each y, the sender then evaluates homomorphically the product of differences of y with all of the sender's items x (computing a function $f = (x - x_1) \cdots (x - x_{|X|})$, s.t. $f(x) = 0$ for each $x \in X$), randomizes the product by adding it with differences uniformly random non-zero plaintext r, and sends the ciphertext c back to the receiver. The receiver decrypts c to $r + f(y)$ and sends $r + f(y)$ to the sender. If $r + f(y) = r$, $y \in X$, otherwise, $y \notin X$.

Input: The sender inputs set X of size $|X|$ and the receiver inputs set Y of size $|Y|$. Both sets consist of bit strings of length σ.
Output: The sender outputs $X \cup Y$ and the receiver outputs $\perp$.

1. [**Setup**] The sender and the receiver jointly agree on a fully homomorphic encryption scheme: The receiver generates a public-secret key pair for the scheme, and keeps the secret key itself.
2. [**Set encryption**] The receiver encrypts each item y_i in its set Y using the fully homomorphic encryption scheme, and sends the $|Y|$ ciphertexts $(c_1, \cdots, c_{|Y|})$ to sender.
3. [**Computation**] For each c_i, the sender
   (a) samples a random non-zero plaintext item r_i;
   (b) homomorphically computes $c'_i = \text{FHE}.\mathsf{Enc}(f(y_i) + r_i), i \in [|Y|]$ where for all $x \in X$, s.t. $f(x) = 0$.
   (c) sends $c'_i, i \in [|Y|]$ to receiver.
4. [**Decryption**] The receiver decrypt $c'_i, i \in [|Y|]$ to $m_i = f(y_i) + r_i$ and sends them to sender.
5. [**Output**] The sender checks all plaintext. If $m_i = r_i, i \in [|Y|]$, $y_i \in X$, otherwise, $y_i \notin X$. The sender and the receiver invoke the ideal functionality $\mathcal{F}_{OT}$. For $i \in [|Y|]$, the sender gets y_i, if $y_i \notin X$. And then sender outputs $X \cup Y$.

Fig. 5: Basic uPSU protocol

We give the following informal theorem with regards to the security and correctness of above basic protocol.

Theorem 1. *(informal). The protocol described in Figure 5 securely and correctly computes the private set union of X and Y in the semi-honest security model, provided that the fully homomorphic encryption scheme is IND-CPA secure with circuit privacy and the oblivious transfer is secure.*

Proof. (Proof sketch). For each index i, the sender gets the ciphertext $\mathsf{FHE}_{pk_R}(y_i)$, and then compute and randomize it to gain $\mathsf{FHE}_{pk_R}(r_i + f(y_i))$. The receiver decrypt it and send $m_i = r_i + f(y_i)$ to the sender. The sender can remove randomization by $m_i - r_i = f(y_i)$, and check that if $f(y_i) = 0$, $y_i \in X \cap Y$ [1], else, $y_i \in X \cup Y \backslash X$. Then, the sender runs oblivious transfer protocol with the receiver to gain the item $y_i \in X \cup Y$ (correctness).

Receiver's security is straightforward: the receiver sends an array of ciphertexts, which looks pseudorandom to the sender since the fully homomorphic encryption scheme is IND-CPA secure. For sender's security, we note that the receiver can decrypt ciphertexts, but only get an array of randomness, since the plaintext is randomized by the sender. Moreover, the oblivious transfer can help the sender get the item in $X \cup Y \backslash X$ and protect the security of $X \cap Y$.

[1] In this case, the sender only knows $y_i \in X$, and it dose not know y_i.

Therefore, the sender learns no additional information beyond the union $X \cup Y$ and the receiver learns nothing.

Compared to basic uPSI [8]. We review the basic unbalanced PSI protocol [8] as follows: the receiver encrypts each item y_i in Y, and send the ciphertexts c_i to the sender; the sender chooses random non-zero plaintext item r_i, and homomorphically computes $r_i \Pi_{x\in X}(c_i - x)$, and returns to the receiver; the receiver can decrypt each ciphertext: if $r_i \Pi_{x\in X}(c_i - x) = 0$, it gets $y_i \in X \cap Y$ else, it gets a random item.

In our basic uPSU protocol, the sender chooses random non-zero plaintext item r_i, and homomorphically computes $c'_i = r_i + \Pi_{x\in X}(c_i - x)$, and returns to the receiver; thus, the receiver decrypts each ciphertexts c'_i and gets an array of random plaintexts m_i and returns them to the sender. The sender knows i-th items in Y belongs to $X \cup Y \backslash X$, if $m_i \neq r_i$, otherwise, it belongs to $X \cap Y$. Then, the sender runs OT protocol with the receiver to gain all items $X \cup Y \backslash X$.

As we can see, the key different step between our uPSU and the PSI [8] is using the different randomization methods. In the PSI [8], they compute the product of randomness r and the polynomial value $f(y)$, if $f(y) = 0$, $rf(y) = 0$ denotes $y \in X$, else $y \notin X$, and the receiver only gets a randomness $rf(y) \neq 0$ which hides the information of f and X. In our uPSU, we compute the sum of randomness r and the polynomial value $f(y)$, the receiver decrypt the ciphertext and get the plaintext $r + f(y)$ which hides the information of f and X. Then the receiver sends the plaintext $r + f(y)$ to sender, if $f(y) = 0$, $r + f(y) = r$ denotes $y \in X$, else $y \notin X$, and the sender can gets $f(y)$. This method will leak some information of $y \notin X$, but this leakage does not cause any harm to the PSU, since the PSU protocol releases that value.

4 Full Unbalanced PSU and Security Proof

In this section, We start from our basic protocol in Figure 5 and give a fast full uPSU based on some optimization techniques.

4.1 Full uPSU Protocol

We detail the setup phase in Figure 6, 7, 8, given a secure fully homomorphic encryption scheme with circuit privacy and a secure OT protocol.

In the setup phase 6, the sender and the receiver agree on the hashing parameters and the FHE scheme parameters. After the setup phase, the sender and the receiver take advantage of the optimization techniques [8], such as Hashing, Batching, Windowing, Partitioning, Modulus Switching, etc, to pre-process the set X and Y offline 7, respectively. After offline pre-processing phase, the sender and the receiver begin the efficient online phase 8: first, the receiver sends the ciphertexts to the sender, and the sender homomorphically computes ciphertexts and returns them back. Then, the receiver decrypts the new ciphertexts and returns them back to the sender, the sender can check which position of the items

Input: Receiver inputs set $Y \subset \{0,1\}^\sigma$ of size $|Y|$; sender inputs set $X \subset \{0,1\}^\sigma$ of size $|X|$. κ and λ denote the computational and statistical security parameters, respectively.
Output: The sender outputs $Y \cup X$; the receiver outputs $\perp$.

1. [**Setup**] The sender and the receiver agree on the hashing parameters and the FHE scheme parameters.
   (a) [Hashing parameters] They agree on hashing parameters h, m, B, such that simple hashing $h|X|$ balls into m bins with max load B, and cuckoo hashing $|Y|$ balls into m bins succeed with probability $\geq 1 - 2^{-\lambda}$, and publicly choose hash functions $H_1, \cdots, H_h : \{0,1\}^\sigma \to [m]$.
   (b) [FHE parameters] They agree on parameters (n, q, t) for an IND-CPA secure FHE scheme with circuit privacy. They choose t to be large enough so that $\log_2 t > \sigma - \log_2 m + \lceil \log_2 h \rceil + 1$. The parties agree on the windowing parameter $l \in [1, \log_2 B]$ and partitioning parameter $\alpha \in [1, B]$ as to minimize the overall cost.

Fig. 6: Full uPSU protocol (setup phase)

in the set Y belongs to the union. Last, the sender and the receiver run OT protocol together, and the sender obtains the union $X \cup Y$ and outputs it.

4.2 Security Proof

We prove security in the standard semi-honest simulation-based paradigm. Loosely put, we say that the protocol Π_{PSU} of Figure 6, 7, 8 securely realizes the functionality $\mathcal{F}_{PSU}$, if it is correct, and there exist two simulators (PPT algorithms) $\mathsf{Sim}_\mathcal{S}$, $\mathsf{Sim}_\mathcal{R}$ with the following properties. The simulator $\mathsf{Sim}_\mathcal{S}$ takes the sender's set and the union as input, and needs to generate a transcript for the protocol execution that is indistinguishable from the sender's view of the real interaction. $\mathsf{Sim}_\mathcal{R}$ is similarly defined, with the exception of not taking the union as input. For a formal definition of simulation-based security in the semi-honest setting, we refer the reader to [22].

Theorem 2. *The protocol in Figure 6, 7, 8 is a secure protocol for $\mathcal{F}_{PSU}$ in the semi-honest setting.*

Proof. It is easy to see that the protocol correctly computes the union conditioned on the hashing routine succeeding, which happens with overwhelming probability $1 - 2^{-\lambda}$.

We exhibit simulators $\mathsf{Sim}_\mathcal{S}$ and $\mathsf{Sim}_\mathcal{R}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

We start with a corrupt sender, and show the existence of $\mathsf{Sim}_\mathcal{S}$. For easy of exposition, we will assume that the simulator/protocol is parameterized by $(h, m, B, n, q, t, \alpha, l, \{H_i\}_{1 \leq i \leq h})$, which are fixed and public. $\mathsf{Sim}_\mathcal{S}$ $(X = \{x_1, \cdots,$

2. [**Hashing**] Two parties take the parameters h, m, B and hash functions $H_1, \cdots, H_h : \{0,1\}^{\sigma - \log_2 m} \to \{0,1\}^{\log_2 m}$ as input. The sender runs Step 1 in Figure 2 with the set X to compute $\mathbf{B}_x$ and the receiver performs Step 2 in Figure 2 with the set Y to obtain $\mathbf{B}_y$.
3. [**Pre-process X**]
   (a) [Partitioning] The sender partitions its table $\mathbf{B}_x$ vertically (i.e. by columns) into α subtables $\mathbf{B}_{x,1}, \mathbf{B}_{x,2}, \cdots, \mathbf{B}_{x,\alpha}$. Each subtable has $B' = B/\alpha$ columns and m rows. Let $\mathbf{B}_x = [\mathbf{B}_{x,i,j}], \mathbf{B}_{x,i,j} = [x_1^{i,j}, x_2^{i,j}, \cdots, x_{B'}^{i,j}], i \in [\alpha], j \in [m]$.
   (b) [Computing coefficients] For each rows of a subtable $\mathbf{B}_{x,i,j} = [x_1^{i,j}, x_2^{i,j}, \cdots, x_{B'}^{i,j}]$, $i \in [\alpha], j \in [m]$, the sender computes the coefficients of the polynomial $f^{i,j}(x) = \Pi_{k=1}^{B'}(x - x_k^{i,j}) = a_0^{i,j} + a_1^{i,j} x + \cdots + a_{B'-1}^{i,j} x^{B'-1} + a_{B'}^{i,j} x^{B'}$, and then replaces each row $\mathbf{B}_{x,i,j}$, with coefficients of the polynomial $f^{i,j}(x)$, $\mathbf{A}_{i,j} = [a_0^{i,j}, a_1^{i,j}, \cdots, a_{B'}^{i,j}]$, $i \in [\alpha], j \in [m]$.
   (c) [Batching] For each subtable, the sender interprets each of its column as a vector of length m with items in $\mathbb{Z}_t$. Then the sender batches each vector into $\beta = m/n$ plaintext polynomials, and the coefficients are $\hat{\mathbf{A}}_{i,j} = [\hat{\mathbf{a}}_0^{i,j}, \hat{\mathbf{a}}_1^{i,j}, \cdots, \hat{\mathbf{a}}_{B'}^{i,j}], i \in [\alpha], j \in [\beta]$, where $\hat{\mathbf{a}}_k^{i,j} = [\hat{a}_{k,1}^{i,j}, \cdots, \hat{a}_{k,n}^{i,j}], k \in [0, B']$
4. [**Pre-process and Encrypt Y**]
   (a) [Batching] The receiver interprets $\mathbf{B}_y$ as a vector of length m with items in $\mathbb{Z}_t$. It batches this vector into $\beta = m/n$ plaintext polynomials $\bar{Y}_1, \cdots, \bar{Y}_\beta$.
   (b) [Windowing] For each batched plaintext polynomial $\bar{Y}$, the receiver computes the component-wise $i \cdot 2^j$-th powers $\bar{Y}^{i \cdot 2^{lj}}$, for $1 \le i \le 2^l - 1$ and $0 \le j \le \lceil \log_2(B')/l \rceil$.
   (c) [Encrypt] The receiver uses FHE to encrypt each such power, obtaining β collections of ciphertexts $\mathbf{C}_j, j \in [\beta]$. The receiver sends these ciphertexts to the sender.

Fig. 7: Full uPSU protocol (offline pre-processing)

$x_{|X|}\}$, $X \cup Y$) simulates the view of corrupt semi-honest sender. It executes as follows: $\mathsf{Sim}_{\mathcal{S}}$ computes the set $\hat{Y}^* = X \cup Y \backslash X$, and uses $|Y| - |\hat{Y}^*|$ items $\perp$ to pad $\hat{Y}$ to $|Y|$ items and permutes these items randomly. Let $\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_{|\hat{Y}|}\}$. Next it runs step 1-4 as real protocol, and encrypts $r'_{ij} = r_{ij} + f(\hat{y}_i)$, for $\hat{y}_i \in \hat{Y}^*$, $i \in [m]$, $j \in [\alpha]$, and encrypts $r'_{ij} = r_{ij} + 0$ for $\hat{y}_i = \perp, i \in [m], j \in [\alpha]$. $\mathsf{Sim}_{\mathcal{S}}$ runs PS simulator $\mathsf{Sim}_{\mathcal{S},\mathrm{PS}}(\pi, \{\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_m\})$ and gets $\{r_{\pi(1)} \oplus s_{\pi(1)}, r_{\pi(2)} \oplus s_{\pi(2)}, \cdots, r_{\pi(m\alpha)} \oplus s_{\pi(m\alpha)}\}$, and then $\mathsf{Sim}_{\mathcal{S}}$ sets all rows $\{\mathbf{R}'_1, \mathbf{R}'_2, \cdots, \mathbf{R}'_m\}$, $\mathbf{R}'_i = \{r'_{i1}, \cdots, r'_{i\alpha}\}$ as a vector and uses π to permute the vector to $\{r'_{\pi(1)}, r'_{\pi(2)}, \cdots, r'_{\pi(m\alpha)}\}$ and permutes $\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \cdots, \hat{\mathbf{y}}_m\}$ to $\{\hat{y}_{\pi(1)}, \hat{y}_{\pi(2)}, \cdots, \hat{y}_{\pi(m\alpha)}\}$, where $\hat{\mathbf{y}}_j = \{\hat{y}_j, \hat{y}_j, \cdots, \hat{y}_j\}$, $j \in [m]$ denotes α identical $\hat{y}_j$. Then $\mathsf{Sim}_{\mathcal{S}}$ computes $\{r'_{\pi(1)} \oplus s_{\pi(1)}, r'_{\pi(2)} \oplus s_{\pi(2)}, \cdots, r'_{\pi(m\alpha)} \oplus s_{\pi(m\alpha)}\}$. $\mathsf{Sim}_{\mathcal{S}}$ can simulate $\mathcal{S}$ to compute $r''_{\pi(i)} = r'_{\pi(i)} \oplus s_{\pi(i)} \oplus r_{\pi(i)} \oplus s_{\pi(i)}$, $i \in [m\alpha]$. If $r''_{\pi(i)} = 0$, it sets $b_{\pi(i)} = 0$, else, it sets $b_{\pi(i)} = 1$. Thus, $\mathsf{Sim}_{\mathcal{S}}$ gets a bit vector $D = [b_{\pi(1)}, \cdots, b_{\pi(m\alpha)}]$, where if

5. [**Computation**]
   (a) [Homomorphically compute encryptions of all powers] For each collection of ciphertexts $\mathbf{C}_j, j \in [\beta]$, the sender homomorphically compute encryptions of all powers $\hat{\mathbf{C}}_j = [\hat{\mathbf{c}}_0^j, \hat{\mathbf{c}}_1^j, \cdots, \hat{\mathbf{c}}_{B'}^j], j \in [\beta]$, where $\hat{\mathbf{c}}_k^j = [\hat{c}_{k,1}^j, \hat{c}_{k,2}^j, \cdots, \hat{c}_{k,n}^j], k \in [B']$.
   (b) [Homomorphically evaluate the dot product] The sender homomorphically evaluates

$$\mathbf{C}'_{i,j} = \hat{\mathbf{A}}_{i,j}\hat{\mathbf{C}}_j = \sum_{k=0}^{B'} \hat{a}_k^{i,j}\hat{c}_k^j, i \in [\alpha], j \in [\beta].$$

   optionally performs modulus switching on the ciphertexts $\mathbf{C}'_{i,j}, i \in [\alpha], j \in [\beta]$ to reduce their sizes, and sends them back to the receiver.
6. [**Decrypt**] For each $1 \leq i \leq \alpha, 1 \leq j \leq \beta$, the receiver decrypts all ciphertexts it receives and concatenates the resulting β matrixes into one matrix $\mathbf{R}'_{m,\alpha}$.
7. [**Permute and Share**]
   (a) The sender and the receiver run the permute and share protocol. The sender inputs all rows $\{\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_m\}$, $\mathbf{R}_i = \{r_{i1}, \cdots, r_{i\alpha}\}$ as a vector, and the receiver inputs a permutation π. Then the sender gets $\{r_{\pi(1)} \oplus s_{\pi(1)}, r_{\pi(2)} \oplus s_{\pi(2)}, \cdots, r_{\pi(m\alpha)} \oplus s_{\pi(m\alpha)}\}$ and the receiver gets shuffled shares $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(m\alpha)}\}$.
   (b) The receiver sets all rows $\{\mathbf{R}'_1, \mathbf{R}'_2, \cdots, \mathbf{R}'_m\}$, $\mathbf{R}'_i = \{r'_{i1}, \cdots, r'_{i\alpha}\}$ as a vector and uses π to permute the vector to $\{r'_{\pi(1)}, r'_{\pi(2)}, \cdots, r'_{\pi(m\alpha)}\}$ and permutes $\{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m\}$ to $\{y_{\pi(1)}, y_{\pi(2)}, \cdots, y_{\pi(m\alpha)}\}$, where $\mathbf{y}_j = \{y_j, y_j, \cdots, y_j\}, j \in [m]$ denotes α identical y_j. Then the receiver computes $\{r'_{\pi(1)} \oplus s_{\pi(1)}, r'_{\pi(2)} \oplus s_{\pi(2)}, \cdots, r'_{\pi(m\alpha)} \oplus s_{\pi(m\alpha)}\}$ and sends them to the sender.
   (c) The sender computes $r''_{\pi(i)} = r'_{\pi(i)} \oplus s_{\pi(i)} \oplus r_{\pi(i)} \oplus s_{\pi(i)}, i \in [m\alpha]$. If $r''_{\pi(i)} = 0$, it sets $b_{\pi(i)} = 0$, else, it sets $b_{\pi(i)} = 1$, and gains a bit vector $D = [b_{\pi(1)}, \cdots, b_{\pi(m\alpha)}]$.
8. [**Output**] The sender and the receiver run the OT protocol, in which the sender inputs $D = [b_{\pi(1)}, \cdots, b_{\pi(m\alpha)}]$ and the receiver inputs $\{y_{\pi(1)}, y_{\pi(2)}, \cdots, y_{\pi(m\alpha)}\}$. If $b_{\pi(i)} = 1$, the sender gets $y_{\pi(i)}$, else, it gets $\perp$. Thus, the sender can get the set $\hat{Y}^* = X \cup Y \backslash X$. Finally, the sender outputs

$$X \cup Y = \hat{Y}^* \cup X$$

Fig. 8: Full uPSU protocol (online phase)

$b_{\pi(i)} = 1$, $y_{\pi(i)} \in \hat{Y}^*$, else, $y_{\pi(i)} = \perp$. For $i \in [m\alpha]$, $\mathsf{Sim}_{\mathcal{S}}$ invokes OT simulator $\mathsf{Sim}_{\mathcal{S},\mathrm{OT}}(b_{\pi(i)}, \hat{y}_{\pi(i)})$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. In the simulation, the way $\mathcal{S}$ obtains the items in $\hat{Y}^* = X \cup Y \backslash X$ is identical to the real execution. By the IND-CPA security of the fully homomorphic encryption scheme and the security of the permute and share protocol and the OT protocol, this result is indistinguishable from the sender's view in the real protocol.

The case of a corrupt receiver is straightforward. The simulator $\mathsf{Sim}_{\mathcal{R}}(Y = \{y_1, \cdots, y_{|Y|}\})$ can generate new encryptions of randomness in place of the encryptions in step 5. This result is indistinguishable from the sender's view in the real protocol.

References

1. Arbitman, Y., Naor, M., Segev, G.: Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In: 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA. pp. 787–796. IEEE Computer Society (2010). https://doi.org/10.1109/FOCS.2010.80, https://doi.org/10.1109/FOCS.2010.80
2. Bajard, J., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: Avanzi, R., Heys, H.M. (eds.) Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10532, pp. 423–442. Springer (2016). https://doi.org/10.1007/978-3-319-69453-5_23, https://doi.org/10.1007/978-3-319-69453-5_23
3. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in lwe-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7778, pp. 1–13. Springer (2013). https://doi.org/10.1007/978-3-642-36362-7_1, https://doi.org/10.1007/978-3-642-36362-7_1
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 309–325. ACM (2012). https://doi.org/10.1145/2090236.2090262, https://doi.org/10.1145/2090236.2090262
5. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12493, pp. 342–372. Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_12, https://doi.org/10.1007/978-3-030-64840-4_12
6. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1223–1237. ACM (2018). https://doi.org/10.1145/3243734.3243836, https://doi.org/10.1145/3243734.3243836
7. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library - SEAL v2.1. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial

Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10323, pp. 3–18. Springer (2017). https://doi.org/10.1007/978-3-319-70278-0_1, https://doi.org/10.1007/978-3-319-70278-0_1
8. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1243–1255. ACM (2017). https://doi.org/10.1145/3133956.3134061, https://doi.org/10.1145/3133956.3134061
9. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 1135–1150. ACM (2021). https://doi.org/10.1145/3460120.3484760, https://doi.org/10.1145/3460120.3484760
10. Devroye, L., Morin, P.: Cuckoo hashing: Further analysis. Inf. Process. Lett. **86**(4), 215–219 (2003). https://doi.org/10.1016/S0020-0190(02)00500-8, https://doi.org/10.1016/S0020-0190(02)00500-8
11. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
12. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. In: Alt, H., Habib, M. (eds.) STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2607, pp. 271–282. Springer (2003). https://doi.org/10.1007/3-540-36494-3_25, https://doi.org/10.1007/3-540-36494-3_25
13. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 591–617. Springer (2021). https://doi.org/10.1007/978-3-030-75248-4_21, https://doi.org/10.1007/978-3-030-75248-4_21
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
15. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_49, https://doi.org/10.1007/978-3-642-32009-5_49
16. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd Inter-

national Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016), http://proceedings.mlr.press/v48/gilad-bachrach16.html
17. Hogan, K., Luther, N., Schear, N., Shen, E., Stott, D., Yakoubov, S., Yerukhimovich, A.: Secure multiparty computation for cooperative cyber risk assessment. In: IEEE Cybersecurity Development, SecDev 2016, Boston, MA, USA, November 3-4, 2016. pp. 75–76. IEEE Computer Society (2016). https://doi.org/10.1109/SecDev.2016.028, https://doi.org/10.1109/SecDev.2016.028
18. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012. The Internet Society (2012), https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols
19. Jia, Y., Sun, S., Zhou, H., Du, J., Gu, D.: Shuffle-based private set union: Faster and more secure. IACR Cryptol. ePrint Arch. p. 157 (2022), https://eprint.iacr.org/2022/157
20. Kolesnikov, V., Rosulek, M., Trieu, N., Wang, X.: Scalable private set union from symmetric-key techniques. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11922, pp. 636–666. Springer (2019). https://doi.org/10.1007/978-3-030-34621-8_23, https://doi.org/10.1007/978-3-030-34621-8_23
21. Lenstra, A.K., Voss, T.: Information security risk assessment, aggregation, and mitigation. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3108, pp. 391–401. Springer (2004). https://doi.org/10.1007/978-3-540-27800-9_34, https://doi.org/10.1007/978-3-540-27800-9_34
22. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. In: Lindell, Y. (ed.) Tutorials on the Foundations of Cryptography, pp. 277–346. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-57048-8_6, https://doi.org/10.1007/978-3-319-57048-8_6
23. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 557–574. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_33, https://doi.org/10.1007/978-3-642-38348-9_33
24. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995). https://doi.org/10.1017/cbo9780511814075, https://doi.org/10.1017/cbo9780511814075
25. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2161, pp. 121–133. Springer (2001). https://doi.org/10.1007/3-540-44676-1_10, https://doi.org/10.1007/3-540-44676-1_10

26. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. **21**(2), 7:1–7:35 (2018). https://doi.org/10.1145/3154794, https://doi.org/10.1145/3154794
27. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. p. 187 (2005), http://eprint.iacr.org/2005/187
28. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptogr. **71**(1), 57–81 (2014). https://doi.org/10.1007/s10623-012-9720-4, https://doi.org/10.1007/s10623-012-9720-4
29. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Optimal private set union from multi-query reverse private membership test. Cryptology ePrint Archive, Report 2022/358 (2022), https://ia.cr/2022/358