Efficient VOLE based Multi-Party PSI with Lower Communication Cost

Shuqing Zhang
Key Laboratory of Mathematics Mechanization, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, People's Republic of China
zhangshuqing17@mails.ucas.ac.cn

Abstract

We present a new method for doing multi-party private set intersection against a malicious adversary, which reduces the total communication cost to $O(nl\kappa)$. Additionally, our method can also be used to build a multi-party Circuit-PSI without payload. Our protocol is based on Vector-OLE(VOLE) and oblivious key-value store(OKVS). To meet the requirements of the protocol, we first promote the definition of VOLE to a multi-party version. After that, we use the new primitive to construct our protocol and prove that it can tolerate all-but-two malicious corruptions.

Our protocol follows the idea of [21], where each party encodes the respective set as a vector, uses VOLE to encrypt the vector, and finally construct an OPRF to get the result. When it comes to multi-party situation, we have to encrypt several vectors at one time. As a result, the VOLE used in [21] and follow-up papers is not enough, that brings our idea of an multi-party VOLE.

1 Introduction

Private set intersection(PSI) is a classic research object in SMPC, and is widely used in practical applications such as machine learning. In recent years, two-party PSI has been studied fully in [12], [15], [21], [18], and the fastest protocol can process a million items in 0.37 seconds. Along with the development of two-party PSI, other types of PSI are also receiving increasing attention. [21], [18] further consider circuit-PSI following the idea of [16], which makes further applications more convenient. [8] introduces structure-aware PSI, which contributes to fuzzy matching. [19] considers PSI in the unbalanced setting, obtains better efficiency. In our article we mainly consider another kind of PSI, multi-party PSI.

The problem of multi-party PSI was first introduced in [6]. Subsequently, a large number of theoretical protocols ([5], [9], [11], [22], [23]) are proposed until [13] become the first practical one. It is secure against semi-honest adversary in the dishonest-majority setting. Later, efficient multi-party PSI protocols are further developed. [1] is the first concretely efficient maliciously secure multi-party PSI protocol, based on [10] which uses garbled bloom filter(GBF) and [20]. [4] improves [13] and extends it to circuit-PSI and Quorum PSI. [24] achieves maliciously secure multi-party PSI, but in a model in which the two servers P_0 and P_1 do not collude. [14] makes use of OKVS and forms an efficient protocol of semi-honest MP-PSI, while in cases with collusion, more expensive OPPRF and additional Zero-sharing are needed. [7] is more efficient but still needs zero-share and multi calls of OPRF. We note that [17] uses VOLE-based OPRF from [21], and raises an MP-VOLE construction. However, it needs calling standard VOLE n times which is still expensive. [3] gives a different definition of MP-VOLE with construction, but there is no MP-PSI construction.

MP-PSI	Communication	Bandwidth
[1]	$O(nl\kappa^2 + nl\kappa log(l\kappa))$	$O(nl\kappa^2 + nl\kappa log(l\kappa))$
[7]	$O(n^2\kappa + nl(\kappa+\rho+logl))$	$O(nl(\kappa+\rho+logl))$
[17]	$O(n^2\kappa + nl(\kappa+\rho+logl))$	$O(l(\kappa+\rho+logl))$
this work	$O(l(n\kappa+\rho+logl))$	$O(l(n\kappa+\rho+logl))$

Table 1: **Comparison between our protocol and recent MP-PSI protocols.** n is the number of parties, l is the size of each set, κ and ρ are the computational and statistical security parameters respectively.

In our work, we give a more natural definition of MP-VOLE which is almost as efficient as two-party situation, and use it to form a MP-PSI protocol. In Table 1, we compare our protocol with several recent MP-PSI protocols in the dishonest-majority setting and observe significant improvements in communication cost. We also prove that our protocol is secure in the presence of a malicious adversary who may corrupt up to all but two parties.

We will discuss related articles in 1.1 below, and introduce our protocol briefly in 1.2. In section 2, we will mention some of the cryptographic primitives used. In section 3, we will present our core contributions, first promoting MP-VOLE, then proposing our MP-PSI protocol, and finally involving the generalization to Circuit-PSI.

1.1 Related Works

In this subsection, we introduce some most relevant works.

1.1.1 *Rindal et al.*

[21] is the first work to achieve PSI based on VOLE and OKVS (to ensure consistency, in this subsection and 1.2, we will use the notation in [21]). The most common version of VOLE allows two parties to sample random vectors $\vec{A}, \vec{B}, \vec{C} \in \mathbb{F}^m$ and an element $\Delta \in \mathbb{F}$, such that $\vec{C} = \vec{A}\Delta + \vec{B}$. The receiver will hold $\vec{A}, \vec{C}$ while the sender will hold $\vec{B}, \Delta$. VOLE works very similar to OT extension, but more quickly and efficient. OKVS encodes the data held by one party into a short vector through a fast solution method of a system of linear equations. In this work, the receiver uses a hash function H' to map the set X into a random matrix $M \in \mathbb{F}^{n\times m}$ and solves $M\vec{P}^\intercal = (H(x_1), H(x_2), \ldots, H(x_n))^\intercal$ for the unknown $\vec{P} \in \mathbb{F}^m$(assume that the receiver holds the set $X := \{x_1, x_2, \ldots, x_n\}$ and M is constructed to make solving linear systems simple).

Based on the tools above, we introduce the protocol in [21] briefly. The receiver send $\vec{A} + \vec{P}$ to the sender who calculates $\vec{K} := \vec{B} + \Delta(\vec{A} + \vec{P})$. Now a crucial observation is that,

$$M\vec{K}^\intercal = M\vec{B}^\intercal + \Delta(M\vec{A}^\intercal + M\vec{P}^\intercal) = M\vec{C}^\intercal + \Delta M\vec{P}^\intercal.$$

For $x \in X$,

$$H'(x)\vec{K}^\intercal = H'(x)\vec{C}^\intercal + \Delta H(x),$$

where $H'(x)$ is one of the rows in M. An OPRF can therefore be obtained by having the receiver calculate $H'(x)\vec{C}^\intercal$ while the sender computes $H'(x)\vec{K}^\intercal - \Delta H(x)$. Further development from OPRF to PSI is trivial.

Another contribution of this article is that the authors present an extension to Circuit-PSI, which allows the parties to perform a subsequent MPC protocol after the outputs. Recently, [18] follows the idea of it, proposes a more efficient OKVS scheme, and achieves the most powerful PSI protocol we know so far.

1.1.2 *Qiu et al.*

[17] proposes an MP-VOLE protocol mainly aimed at reducing the number of LPN decodings. The protocol running between P_i and a pivot P_0 works as follows: each P_i and P_0 calls standard VOLE functionality $\mathcal{F}_{VOLE}$ but with the same $\vec{A}$(realized by programming the input/output of P_0), thus P_0 does not need to decode $\vec{A}$ for n times. P_0 then uses its own set to compute an OKVS $\vec{S}$ and sends $\vec{d} = \vec{S} - \vec{A}$ to each P_i. This, along with the standard VOLE, creates an OPRF between each party and the pivot. Then the article follows the idea of [7], and finishes their MP-PSI protocol. It is worth noting that this article also proposes a tree structure to reduce the communication bandwidth of the central party P_0.

Compared with this article, our protocol further reduces the number of LPN decodings. We also note that another computational efficiency bottleneck in performing n standard VOLE is the n calls of the FSS functionality. In our MP-VOLE protocol, it only needs to call the FSS functionality once, thus improving the efficiency.

1.1.3 *Nevo et al.*

[14] can implement a very efficient MP-PSI protocol without collusion. The main idea of this protocol is to reduce the multi-party PSI situation to the already well-studied two-party PSI situation. Taking three-party PSI as an example, party P_1 selects a random key k and sends it to P_2. Then P_1 applies the corresponding pseudo-random function $F_k(x)$ to its own set, obtains corresponding values, generates an OKVS structure S by them, and sends it to P_3. Now P_2 and P_3 can generate two new sets through $F_k(x)$ and S, which satisfy their intersection is the intersection of three parties. As a result, the three-party PSI is simplified to two parties, and only relies on symmetric-key primitives.

However, when there is collusion, [14] becomes much more complicated. Expensive OPPRF is needed, and additional zero-sharing is performed. In comparison, our protocol also simplifies the multi-party situation to two parties, but does not require so many complex primitives. We can further implement multi-party Circuit-PSI with a small cost in efficiency. However, if we follow the idea of [14], it will be difficult to achieve.

1.2 Overview of Our Results

We aim to construct a maliciously secure multi-party PSI protocol. We make use of two main building blocks: vector-OLE(VOLE) and oblivious key-value store(OKVS). They are the basis of nowadays fastest two-party PSI protocol, and after some adjustments, they can be used in our protocol.

We now introduce our protocol briefly in this subsection. Our core tool is a promotion of VOLE. Based on 1.1.1, our MP-VOLE allows n parties $P_1, P_2, \ldots, P_n$ to sample $n+1$ random vectors $\vec{A}_0, \vec{A}_1, \ldots, \vec{A}_n \in \mathbb{F}^m$ and $n-1$ random elements $x_1, x_2, \ldots, x_{n-1} \in \mathbb{F}$, such that $\vec{A}_0 = \sum_{i=1}^{n} \vec{A}_i x_i$, where $x_n = 1$. A direct idea is to imitate the distribution method before: the party P_1 will hold $\vec{A}_0, \vec{A}_1$, while other P_i will hold $\vec{A}_i, x_{i-1}$ respectively. It is a fair construction plan, and is secure under collusion of any strict subset of the parties. However, the problem is that if we follow the idea of [21], this plan is difficult to be used to construct a secure MP-PSI protocol. Considering the simplified protocol in 1.1.1, one can find that there should be one party who knows all the x_i in order to compute all the

$-x_i H(\cdot)$ parts. Therefore, we adjust the distribution method, let P_n hold all the x_i and $\vec{A}_n$, P_1 remain the same and other P_i only get corresponding $\vec{A}_i$. It seems that P_n knows too much but later we prove that the scheme is secure enough for our MP-PSI. After MP-VOLE is carried out, each $(P_i)_{i\in[1,n-1]}$ encodes its set into a short vector $(\vec{Q}_i)_{i\in[1,n-1]}$ using the same hash functions by OKVS system, then sends $(\vec{A}_i+\vec{Q}_i)_{i\in[1,n-1]}$ to P_n. P_n calculates $\vec{A}'_n := \vec{A}_n + \sum_{i=1}^{n-1} x_i(\vec{A}_i+\vec{Q}_i)$. As before we can observe that

$$M\vec{A}'^{\intercal}_n = M\vec{A}_n^{\intercal} + \sum_{i=1}^{n-1} x_i(M\vec{A}_i^{\intercal} + M\vec{Q}_i^{\intercal}) = M\vec{A}_0^{\intercal} + \sum_{i=1}^{n-1} x_i M\vec{Q}_i^{\intercal},$$

and for $x \in \cap_{i=1}^{n} X_i$,

$$H'(x)\vec{A}'^{\intercal}_n = H'(x)\vec{A}_0^{\intercal} + (\sum_{i=1}^{n-1} x_i)H(x_i).$$

An OPRF can therefore be obtained by having P_0 calculate $H'(x)\vec{A}_0^{\intercal}$ while P_n computes $H'(x)\vec{A}'^{\intercal}_n - (\sum_{i=1}^{n-1} x_i)H(x_i)$. Further development from OPRF to MP-PSI and MP-C-PSI is trivial.

Note that in our construction, the part of solving OKVS can be operated in parallel, the method of obtaining the OPRF at the end and further performing the intersection is also similar to [21]. Therefore, except for the MP-VOLE part, the efficiency of our protocol is approximate to the efficiency of a two-party PSI.

2 Preliminaries

2.1 Private Set Intersection

The elementary form of Private Set Intersection (PSI) primarily addresses the problem that two parties possess distinct datasets, seek to get their intersection, while remaining entirely ignorant about the elements not shared between them. In this work, we mainly focus on the investigation of MP-PSI, which extends the number of participating parties. We will also mention circuit-PSI, which allows further computation on the intersection.

We present the ideal functionality of MP-PSI in Figure 1. Generally speaking, the elements in the respective sets X_i of each party should be of the same size. In practice we can make use of a collision-resistant hash function to reach the request.

Parameters: There are n parties $P_1, P_2, \ldots, P_n$ and an adversary $\mathcal{A}$.
Functionality: Upon receiving X^i from each party, the functionality waits for the adversary to send $abort \in \{0,1\}$. If $abort = 0$, it outputs the intersection $\cap_{i=1}^{n} X^i$ to each party, otherwise it outputs $\perp$ to each party.

Figure 1: Ideal functionality $\mathcal{F}_{MP-PSI}$

2.2 Function Secret Sharing

Informally, a function secret sharing (FSS) scheme splits a function $f : I \to G$ into two functions f_0 and f_1 such that $f_0(x) + f_1(x) = f(x)$ for every input x, and each $(f_b)_{b\in\{0,1\}}$ computationally hides f except acceptable leakage. Moreover, the evaluation of the sub function should also hide f, which refuses the trivial idea of simply split the coefficient of the function. If the function f is a point function, the scheme is known as **distributed point function(DPF)**, while in this work we rely on efficient constructions of FSS schemes for multi-point functions. Here we present the definition of FSS and multi-point function from [2].

Definition 1 (FSS)**.** *A 2-party function secret sharing (FSS) scheme for a class of functions $\mathcal{F} = \{f : I \to \mathbb{G}\}$ with input domain I and output domain an abelian group $(\mathbb{G},+)$, is a pair of PPT algorithms FSS = (FSS.Gen, FSS.Eval) with the following syntax:*

- *$FSS.Gen(1^\lambda, f)$, given security parameter λ and description of a function $f \in \mathcal{F}$, outputs a pair of keys (K_0, K_1);*

- *$FSS.Eval(b, K_b, x)$, given party index $b \in \{0,1\}$, key K_b, and input $x \in I$, outputs a group element $y_b \in \mathbb{G}$.*

Given an allowable leakage function $Leak : \{0,1\}^* \to \{0,1\}^*$, the scheme FSS should satisfy the following requirements:

- **Correctness**. For any $f : I \to \mathbb{G}$ in $\mathcal{F}$ and $x \in I$, we have $Pr[(K_0,K_1) \overset{R}{\leftarrow} FSS.Gen(1^\lambda, f) : FSS.Eval(0,K_0,x) + FSS.Eval(1,K_1,x) = f(x)] = 1$.

- **Security**. For any $b \in \{0,1\}$, there exists a PPT simulator Sim such that for any polynomial-size function sequence $f_\lambda \in \mathcal{F}$, the distributions $\{(K_0,K_1) \overset{R}{\leftarrow} FSS.Gen(1^\lambda, f_\lambda) : K_b\}$ and $\{K_b \overset{R}{\leftarrow} Sim(1^\lambda, Leak(f_\lambda))\}$ are computationally indistinguishable.

Next we consider multi-point functions. It is a natural generalization of point functions. A t-point function evaluates to 0 everywhere, except on t specified points. When specifying multi-point functions we often view the domain of the function as $[n]$ for $n = 2^l$ instead of $\{0,1\}^l$. Formally:

Definition 2 (Multi-Point Function)**.** *An (n,t)-multi-point function over an abelian group $(\mathbb{G},+)$ is a function $f_{S,\vec{y}} : [n] \to \mathbb{G}$, where $S = \{s_1, s_2, \ldots, s_t\}$ is a subset of $[n]$ of size t, $\vec{y} = (y_1, y_2, \ldots, y_t) \in \mathbb{G}^{kt}$, and $f_{S,\vec{y}}(s_i) = y_i$ for any $i \in [t]$, and $f_{S,\vec{y}}(x) = 0$ for any $x \in [n] \setminus S$.*

Parameters: There are two parties, a Sender and a Receiver. Let $\mathbb{F}$ be a field, m be the size of output vectors.
Functionality: Upon receiving (sender,sid) from the sender and (receiver,sid) from the receiver.

- If the Receiver is malicious, wait for them to send $C, A \in \mathbb{F}^m$. Sample $\Delta \leftarrow \mathbb{F}$ and compute $B := C - A\Delta$. Otherwise,
- If the Sender is malicious, wait for them to send $B \in \mathbb{F}^m, \Delta \in \mathbb{F}$. Sample $A \leftarrow \mathbb{F}^m$ and compute $C := B + A\Delta$. Otherwise,
- Sample $A, B \leftarrow \mathbb{F}^m, \Delta \leftarrow \mathbb{F}$ and compute $C := B + A\Delta$.

The functionality outputs Δ, B to the Sender and C, A to the Receiver.

Figure 2: Ideal functionality $\mathcal{F}_{VOLE}$

A Multi-Point Function Secret Sharing (MPFSS) is an FSS scheme for the class of multi-point functions, where a multi-point function $f_{S,\vec{y}}$ is represented in a natural way.

Finally, the application of MPFSS in our work requires applying the evaluation algorithm on all inputs. Given an MPFSS (MPFSS.Gen, MPFSS.Eval), we denote by MPFSS.FullEval an algorithm which, on input a bit b, and an evaluation key K_b, outputs a list of $|I|$ elements of $\mathbb{G}$ corresponding to the evaluation of $FSS.Eval(b, K_b, \cdot)$ on every input $x \in I$ (in some arbitrary specified order).

2.3 Vector OLE

The VOLE functionality, as we introduced in 1.1.1, is a two-party functionality that selects a pair of vectors randomly, provides it to P_0, and enables P_1 to learn a random linear combination of these vectors. We present the VOLE functionality in Figure 2.

In the description of this functionality, it is noteworthy that all parameters are generated in a random manner. However, there are certain situations in which it is beneficial to give a party the capability to select specific parameters. Formally, the VOLE scheme we present is called a random vector OLE. In this work, we only focus on this particular variant of VOLE.

Furthermore, it is important to highlight that the VOLE proposed in [2] also offers a significant advantage. It can operate on short correlation vectors $(seed_1, seed_2)$ online and then expand them locally on both parties to produce much longer target outputs $(Expand(seed_1), Expand(seed_2))$. Our promotion also preserves this advantage, ensuring that our protocol remains competitive.

- For $i \in [m]$:choose uniform $v_i \leftarrow \mathcal{V}$, where $\mathcal{V}$ represent the value domain.
- Return $\mathcal{A}(Encode(\{(k_1, v_1), (k_2, v_2), \ldots, (k_m, v_m)\}))$.

Figure 3: Obliviousness Experiment $Exp^{\mathcal{A}}(\mathcal{K} = (k_1, k_2, \ldots, k_m))$

2.4 Oblivious Key-Value Store

Oblivious Key-Value Store(OKVS) is a data structure which is often used to hide meaningful keys. It consists of two algorithms (here we use the definition from [14]):

- Encode takes as input a set of (k_i, v_i) key-value pairs and outputs an object S(or, with negligible probability, an error indicator).
- Decode takes as input an object S, a key k and outputs a value x.

The algorithms should satisfy:

- **Correctness.** If $(k, v) \in S$, then $Decode(S, k) = v$.
- **Obliviousness.** For all key domains $\mathcal{K}_1, \mathcal{K}_2$ of size m and all PPT adversaries $\mathcal{A}$:$|Pr[Exp^{\mathcal{A}}(\mathcal{K}_1)] - Pr[Exp^{\mathcal{A}}(\mathcal{K}_2)]| \leq negl(\kappa)$, where the experiment $Exp^{\mathcal{A}}(\mathcal{K})$ is presented in Figure 3.

Additionally, in our work we need the OKVS construction to be *linear*, which means:
For any k, $Decode(S, k)$ can be expressed as a linear combination of S.

Therefore it is easily observed that, for any $S = \sum S_i$,

$$Decode(S, k) = \sum Decode(S_i, k).$$

There are several ways to form an OKVS structure in previous studies [21], [18]. They all satisfy linearity, and we will use the construction in [18].

2.5 Learning Parity with Noise

Our construction relies on the Learning Parity with Noise (LPN) assumption. It is like the LWE assumption, except that the noise here is required to have a small Hamming weight and satisfy the Bernoulli distribution instead of discrete Gaussian distribution. For a finite field $\mathbb{F}$, we denote by $Ber_r(\mathbb{F})$ the Bernoulli distribution obtained by sampling a uniformly random element of $\mathbb{F}$ with probability r, and 0 with probability $1 - r$. We define the LPN assumption over a field $\mathbb{F}$ formally below.

Definition 3 (LPN assumption)**.** *Let $\boldsymbol{C}$ be a probabilistic code generation algorithm such that $\boldsymbol{C}(k,q,\mathbb{F})$ outputs (a description of) a matrix $A \in \mathbb{F}^{k\times q}$. For dimension $k = k(\lambda)$, number of queries (or block length) $q = q(\lambda)$, and noise rate $r = r(\lambda)$, the LPN(k,q,r) assumption with respect to $\boldsymbol{C}$ states that for any polynomial-time non-uniform adversary $\mathcal{A}$, it holds that*

$$Pr\begin{bmatrix} \mathbb{F} \leftarrow \mathcal{A}(1^\lambda), \\ A \overset{R}{\leftarrow} \boldsymbol{C}(k,q,\mathbb{F}), \\ e \overset{R}{\leftarrow} Ber_r(\mathbb{F})^q, : \mathcal{A}(A,b) = 1 \\ s \overset{R}{\leftarrow} \mathbb{F}^k, \\ b \leftarrow s\cdot A + e \end{bmatrix} \approx$$

$$Pr\begin{bmatrix} \mathbb{F} \leftarrow \mathcal{A}(1^\lambda), \\ A \overset{R}{\leftarrow} \boldsymbol{C}(k,q,\mathbb{F}), : \mathcal{A}(A,b) = 1 \\ b \overset{R}{\leftarrow} \mathbb{F}^q \end{bmatrix}$$

By default, we assume that C outputs a uniformly random matrix.

3 Multi-Party Private Set Intersection

Next we turn to our main contributions. We will define our MP-VOLE generator first and then use it to construct our MP-PSI protocol.

3.1 Multi-Party VOLE

We start from promoting the VOLE protocol. First, we introduce the ideal functionality of MP-VOLE in Figure 4. Its design idea is consistent with what we mentioned before in 1.1.1.

Next, we give a formal definition of MP-VOLE. We follow the idea of [2] and allow one party to decide all the x_i in the definition, thus bringing possibilities for other subsequent applications. In our article, however, these x_i are just randomly selected.

Within the definition, security is the most noteworthy point. It is evident that party P_n holds the most substantial amount of information. As a result, our security discussion revolves around this central party. We divide the discussion into two distinct situations: one where P_n is corrupted and the other where P_n remains honest. In both cases, we show that the information possessed by the adversary along with the parameters generated by the protocol in the real world is indistinguishable from the case where these authentic parameters are replaced by randomly generated ones.

For simplicity, we multiply $\vec{A}_n$ by an 1.

Definition 4 (Pseudorandom MP-VOLE Generator)**.** *A Pseudorandom MP-VOLE Generator is a pair of algorithms (Setup,Expand) with the following syntax:*

Parameters: There are n parties $P_1, P_2, \ldots, P_n$ and an adversary. Let $\mathbb{F}$ be a field. Let $\mathcal{C}$ be the corrupted parties.
Functionality: Upon receiving sid from P_i.

1. If P_1 is corrupted, wait for the adversary to send $\vec{A}_0, \vec{A}_1 \in \mathbb{F}^m$. Otherwise sample $\vec{A}_0, \vec{A}_1 \leftarrow \mathbb{F}^m$.
2. For $i \in [2, n-1]$, if P_i is corrupted, wait for the adversary to send $\vec{A}_i \in \mathbb{F}^m$. Otherwise sample $\vec{A}_i \leftarrow \mathbb{F}^m$.
3. If P_n is honest, sample $\vec{x} \leftarrow \mathbb{F}^{n-1}$, where $\vec{x} := (x_1, x_2, \ldots, x_{n-1})$, and compute $\vec{A}_n := \vec{A}_0 - \vec{A}_1 x_1 - \vec{A}_2 x_2 - \ldots - \vec{A}_{n-1} x_{n-1}$. Otherwise,
4. Wait for the adversary to send $\vec{x} \in \mathbb{F}^{n-1}$ and $\vec{A}_n \in \mathbb{F}^m$. If $\exists i \in [2, n-1], P_i \notin \mathcal{C}$, recompute $\vec{A}_i := (\vec{A}_0 - \vec{A}_1 x_1 - \vec{A}_2 x_2 - \ldots - \vec{A}_{i-1} x_{i-1} - \vec{A}_{i+1} x_{i+1} - \ldots - \vec{A}_{n-1} x_{n-1} - \vec{A}_n)/x_i$. Otherwise,
5. Resample $\vec{A}_1 \leftarrow \mathbb{F}^m$, recompute $\vec{A}_0 := \vec{A}_1 x_1 + \vec{A}_2 x_2 + \ldots + \vec{A}_{n-1} x_{n-1} + \vec{A}_n$.

The functionality outputs $\vec{A}_0, \vec{A}_1$ to P_1, $\vec{A}_i$ to P_i for $i \in [2, n-1]$, $\vec{A}_n$ and $\vec{x}$ to P_n.

Figure 4: Ideal functionality $\mathcal{F}_{MP-VOLE}$

- *Setup$(1^\lambda, \mathbb{F}, m, x_1, x_2, \ldots, x_{n-1})$ is a PPT algorithm that given a security parameter λ, field $\mathbb{F}$, output length m, and scalars $x_i \in \mathbb{F}$ outputs n seeds seed$_1, \ldots,$ seed$_n$, where seed$_n$ includes $x_1, x_2, \ldots, x_{n-1}$.*

- *Expand$(\sigma, seed_\sigma)$ is a polynomial-time algorithm that given party index $\sigma \in [1,n]$ and a seed seed$_\sigma$, outputs a pair $(\vec{A}_0, \vec{A}_1) \in \mathbb{F}^m \times \mathbb{F}^m$ if $\sigma = 1$, or a vector $\vec{A}_i \in \mathbb{F}^m$ if $\sigma = i, i \in [2,n]$.*

The algorithms (Setup, Expand) should satisfy the following:

- **Correctness.** For any field $\mathbb{F}$ and $x_i \in \mathbb{F}$, for any $seed_i, i \in [1,n]$ in the image of $Setup(1^\lambda, \mathbb{F}, m, x_1, x_2, \ldots, x_{n-1})$(for some m), denoting $(\vec{A}_0, \vec{A}_1) \leftarrow Expand(1, seed_1)$, and $\vec{A}_i \leftarrow Expand(i, seed_i)$, it holds that $\vec{A}_0 := \vec{A}_1 x_1 + \vec{A}_2 x_2 + \ldots + \vec{A}_{n-1} x_{n-1} + \vec{A}_n$.

- **Security.** For any (stateful, nonuniform) polynomial-time adversary $\mathcal{A}$, corrupted parties $\mathcal{C}$, $U_1 := \{j | P_j \in \mathcal{C}\}$, it holds that

– if $n \notin U_1$,

$$Pr\left[\begin{array}{r}(\mathbb{F},1^m,\vec{x}) \leftarrow \mathcal{A}(1^\lambda),\\ (seed_i)_{i\in[1,n]} \xleftarrow{R} Setup(1^\lambda,\mathbb{F},m,\vec{x}),\\ V_1 \leftarrow \{Expand(i,seed_i)\}_{i\in[1,n]\setminus U_1}\\ : \mathcal{A}(V_1,\vec{x},(seed_j)_{j\in U_1}) = 1\end{array}\right] \approx$$

$$Pr\left[\begin{array}{r}(\mathbb{F},1^m,\vec{x}) \leftarrow \mathcal{A}(1^\lambda), \vec{x}' \xleftarrow{R} \mathbb{F}^{n-1},\\ (seed_i)_{i\in[1,n]} \xleftarrow{R} Setup(1^\lambda,\mathbb{F},m,\vec{x}),\\ V_2 \leftarrow \{Expand(i,seed_i)\}_{i\in U_1},\\ U_2 \leftarrow \{l|\vec{A}_l \notin V_2\}, u \leftarrow max(U_2),\\ (\vec{A}_l \xleftarrow{R} \mathbb{F}^m)_{l\in U_2\setminus\{u\}},\\ \vec{A}_u := (\vec{A}_0 - \sum_{i=1}^{u-1}\vec{A}_i x'_i - \sum_{i=u+1}^{n}\vec{A}_i x'_i)/x'_u\\ : \mathcal{A}((\vec{A}_l)_{l\in U_2},\vec{x}',(seed_j)_{j\in U_1}) = 1\end{array}\right]$$

– if $n \in U_1$,

$$Pr\left[\begin{array}{r}(\mathbb{F},1^m,\vec{x}) \leftarrow \mathcal{A}(1^\lambda),\\ (seed_i)_{i\in[1,n]} \xleftarrow{R} Setup(1^\lambda,\mathbb{F},m,\vec{x}),\\ V_1 \leftarrow \{Expand(i,seed_i)\}_{i\in[1,n]\setminus U_1}\\ : \mathcal{A}(V_1,\vec{x},(seed_j)_{j\in U_1}) = 1\end{array}\right] \approx$$

$$Pr\left[\begin{array}{r}(\mathbb{F},1^m,\vec{x}) \leftarrow \mathcal{A}(1^\lambda),\\ (seed_i)_{i\in[1,n]} \xleftarrow{R} Setup(1^\lambda,\mathbb{F},m,\vec{x}),\\ V_2 \leftarrow \{Expand(i,seed_i)\}_{i\in U_1},\\ U_2 \leftarrow \{l|\vec{A}_l \notin V_2\}, u \leftarrow max(U_2),\\ (\vec{A}_l \xleftarrow{R} \mathbb{F}^m)_{l\in U_2\setminus\{u\}},\\ \vec{A}_u := (\vec{A}_0 - \sum_{i=1}^{u-1}\vec{A}_i x_i - \sum_{i=u+1}^{n}\vec{A}_i x_i)/x_u\\ : \mathcal{A}((\vec{A}_l)_{l\in U_2},\vec{x},(seed_j)_{j\in U_1}) = 1\end{array}\right]$$

where $\vec{x} = (x_1,x_2,\ldots,x_{n-1},x_n := 1), \vec{x}' = (x'_1,x'_2,\ldots,x'_{n-1},x'_n := 1)$.

Next we present our MP-VOLE generator construction. We use a "spreading function" just as [2] does. The function $spread_m$ takes as input a subset $S = \{s_1,s_2,\ldots,s_{|S|}\}$ of $[m]$ (with $s_1 < s_2 < \ldots < s_{|S|}$) and a vector $\vec{y} = (y_1,y_2,\ldots,y_{|S|}) \in \mathbb{F}^{|S|}$, such that $spread_m(S,\vec{y})$ is the vector $\vec{z}$ satisfying $z_j = 0$ for any $j \in [m]\setminus S$, and $z_{s_i} = y_i$ for $i \in [1,|S|]$. Note that the function $spread_m$ is a linear function. Our construction is given in Figure 5.

Now we need to show that our construction satisfies the correctness and security defined previously. It is easy to find that when the number of corrupt parties reaches $n-1$, the

Parameters: There are n parties $P_1,P_2,\ldots,P_n$. Let $\mathbb{F}$ be a field, m be the expanded length, λ be the security parameter, $k := k(\lambda)$ be the dimension and $t := t(\lambda)$ be the noise parameter of the LPN assumption.
Building blocks: A code generator **C**, where $\mathbf{C}(k,m,\mathbb{F})$ defines a public matrix $C_{k,m} \in \mathbb{F}^{k\times m}$. A multi-point function secret sharing MPFSS=(MPFSS.Gen, MPFSS.Eval, MPFSS.FullEval).

- $\mathcal{G}_{MP-VOLE}.Setup(1^\lambda,\mathbb{F},m,x_1,x_2,\ldots,x_{n-1})$**:** Pick a random size-t subset S of $[m]$,n random vectors $(\vec{a}_0,\vec{a}_1,\ldots,\vec{a}_{n-1}) \xleftarrow{R} (\mathbb{F}^k)^n$, and $n-1$ random vectors $(\vec{y}_1,\vec{y}_2,\ldots,\vec{y}_{n-1}) \xleftarrow{R} (\mathbb{F}^t)^{n-1}$. Let $s_1 < s_2 < \ldots < s_t$ denote the elements of S. Set $\vec{a}_n := \vec{a}_0 - \vec{a}_1x_1 - \vec{a}_2x_2 - \ldots - \vec{a}_{n-1}x_{n-1}$. Compute $(K_0,K_1) \xleftarrow{R} MPFSS.Gen(1^\lambda,f_S,x_1\vec{y}_1+x_2\vec{y}_2+\ldots+x_{n-1}\vec{y}_{n-1})$. Set $seed_1 \leftarrow (\mathbb{F},m,K_0,S,\vec{y}_1,\vec{a}_0,\vec{a}_1)$, $seed_i \leftarrow (\mathbb{F},m,S,\vec{y}_i,\vec{a}_i)$ for $i \in [2,n-1]$, $seed_n \leftarrow (\mathbb{F},m,K_1,x_1,x_2,\ldots,x_{n-1},\vec{a}_n)$.
- $\mathcal{G}_{MP-VOLE}.Expand(\sigma,seed_\sigma)$**:**
  - If $\sigma = 1$, parse $seed_1$: compute $\vec{v}_0 \leftarrow MPFSS.FullEval(0,K_0),\vec{\mu}_1 \leftarrow spread_m(S,\vec{y}_1)$. Output $(\vec{A}_0,\vec{A}_1) \leftarrow (\vec{a}_0\cdot C_{k,m}+\vec{v}_0,\vec{a}_1\cdot C_{k,m}+\vec{\mu}_1)$.
  - If $\sigma = i, i \in [2,n-1]$, parse $seed_i$: compute $\vec{\mu}_i \leftarrow spread_m(S,\vec{y}_i)$. Output $\vec{A}_i \leftarrow \vec{a}_i\cdot C_{k,m}+\vec{\mu}_i$.
  - If $\sigma = n$, parse $seed_n$: compute $\vec{v}_1 \leftarrow MPFSS.FullEval(1,K_1)$. Output $\vec{A}_n \leftarrow \vec{a}_n\cdot C_{k,m}-\vec{v}_1$.

Figure 5: MP-VOLE generator $\mathcal{G}_{MP-VOLE}$ which realizes $\mathcal{F}_{MP-VOLE}$

system is insecure: say $p_1, p_3, p_4, \dots, p_{n-1}, p_n$ are corrupted, $\vec{A}_2$ held by P_2 can be easily calculated by $\vec{A}_2 = (\vec{A}_0 - \vec{A}_1 x_1 - \sum_{i=3}^{n} \vec{A}_i x_i)/x_2$. Luckily, we can prove that the generator is secure when the number of corrupting parties is no more than $n-2$.

Theorem 5. *$\mathcal{G}_{MP-VOLE}$ is a secure MP-VOLE generator against $n-2$ party collusion given a secure MPFSS scheme.*

Correctness. By the MPFSS correctness, it holds that $MPFSS.FullEval(0,K_0) + MPFSS.FullEval(1,K_1) = spread_m(S, x_1\vec{y}_1 + x_2\vec{y}_2 + \dots + x_{n-1}\vec{y}_{n-1}) = \vec{\mu}_1 x_1 + \vec{\mu}_2 x_2 + \dots + \vec{\mu}_{n-1} x_{n-1}$. Therefore, $\vec{A}_1 x_1 + \vec{A}_2 x_2 + \dots + \vec{A}_{n-1} x_{n-1} + \vec{A}_n = (\vec{a}_1 \cdot C_{k,m} + \vec{\mu}_1) x_1 + (\vec{a}_2 \cdot C_{k,m} + \vec{\mu}_2) x_2 + \dots + (\vec{a}_{n-1} \cdot C_{k,m} + \vec{\mu}_{n-1}) x_{n-1} + \vec{a}_n \cdot C_{k,m} - \vec{\nu}_1 = (\vec{a}_1 + \vec{a}_2 + \dots + \vec{a}_{n-1} + \vec{a}_n) C_{k,m} + MPFSS.FullEval(0,K_0) + MPFSS.FullEval(1,K_1) - MPFSS.FullEval(1,K_1) = \vec{a}_0 \cdot C_{k,m} + \vec{\nu}_0 = \vec{A}_0$.
Security. We start by proving the first security requirement through a sequence of games.

- **Game 0.** Compute $(seed_i)_{i\in[1,n]} \xleftarrow{R} Setup(1^\lambda, \mathbb{F}, m, \vec{x})$, set $V_1 \leftarrow \{Expand(i, seed_i)\}_{i\in[1,n]\setminus U_1}$, and send $(V_1, \vec{x}, (seed_j)_{j\in U_1})$ to $\mathcal{A}$. Denote β_0 the output of $\mathcal{A}$ in this game.
- **Game 1.** In this game, compute the input of $\mathcal{A}$ as before except that K_0 is computed solely from $(\mathbb{F}, m, t)$ using the simulator for the secrecy of the MPFSS. Note that in this game, K_0 carries no information whatsoever about x_i and $(\vec{\mu}_i)_{i\in[2,n-1]}$. Denote β_1 the output of $\mathcal{A}$ in this game, by the secrecy of the MPFSS, $|Pr[\beta_1 = 1] - Pr[\beta_0 = 1]| = negl(\lambda)$.
- **Game 2.** In this game, pick $\vec{x}' \xleftarrow{R} \mathbb{F}^n, (\vec{A}_l \xleftarrow{R} \mathbb{F}^m)_{l\in U_2\setminus\{u\}}$ and set $\vec{A}_u := (\vec{A}_0 - \sum_{i=1}^{u-1} \vec{A}_i x'_i - \sum_{i=u+1}^{n} \vec{A}_i x'_i)/x'_u$. There are two differences between this game and the previous one. The first is that we replace $\vec{x}$ by a uniformly random $\vec{x}'$. Note that the only part of $(seed_j)_{j\in U_1}$ which depends on x is, if P_1 is corrupted, $K_0 \in seed_1$, while in **Game 1** we have made it irrelevant. The other difference is that we replaced several $\vec{A}_l = \vec{a}_l \cdot C_{k,m} + \vec{\mu}_l$ by uniformly random $\vec{A}'_l$. Observe that $\vec{A}_l$ is exactly a noisy linear encoding of $\vec{a}_l$, using the linear code $C_{k,m} \in \mathbb{F}^{k(\lambda)\times m}$, with noise vectors $\vec{\mu}_l$. Since $seed_j$ carries no information about $\vec{\mu}_l$, $\vec{A}_l$ is therefore a noisy linear encoding of $\vec{a}_l$, where the number of noisy coordinates is exactly $t(\lambda)$, and each noisy coordinate is masked by a uniformly random element of $\mathbb{F}$. Therefore, distinguishing **Game 2** from **Game 1** is equivalent to breaking $|U_2|-1$ independent LPN assumptions of dimension $k(\lambda)$ over $\mathbb{F}$, with m samples and a noise rate $t(\lambda)/m$: denoting β_2 the output of $\mathcal{A}$ in this game, under the LPN$(k(\lambda), m, t(\lambda)/m)$ assumption over $\mathbb{F}$, $|Pr[\beta_2 = 1] - Pr[\beta_1 = 1]| = negl(\lambda)$. This concludes the proof of the first security requirement.

Parameters: There are n parties $P_1, P_2, \dots, P_n$ with respective sets $X_1, X_2, \dots, X_n$. Let $\mathbb{F}$ be a field, κ be the security parameter, $\mathcal{F}_{MP-VOLE}$ be the MP-VOLE functionality.
Protocol: Upon input (sid, X_i):

1. P_1 samples and reveals $r, r' \xleftarrow{R} \{0,1\}^\kappa$.
2. Each $(P_i)_{i\in[1,n-1]}$ computes $\vec{Q}_i := Encode(L_i, r)$, where $L_i := \{(x, H^{\mathbb{F}}(x, r')) | x \in X_i\}$.
3. Each P_i sends sid to $\mathcal{F}_{MP-VOLE}$ with dimension $m := |\vec{Q}_i|$ and $\mathbb{F}$. The parties respectively receive $(\vec{A}_0, \vec{A}_1)$, $(\vec{A}_i)_{i\in[2,n-1]}$ and $(\vec{x} := (x_1, x_2, \dots, x_{n-1}), \vec{A}_n)$.
4. For $i \in [1, n-1]$, each P_i sends $\vec{A}'_i := \vec{A}_i + \vec{Q}_i$ to P_n, who defines $\vec{A}'_n = \vec{A}_n + \vec{A}'_1 \cdot x_1 + \vec{A}'_2 \cdot x_2 + \dots + \vec{A}'_{n-1} \cdot x_{n-1}$.
5. P_n sends $Y := \{H(Decode(\vec{A}'_n, y, r) - (x_1 + x_2 + \dots + x_{n-1}) H^{\mathbb{F}}(y, r')) | y \in X_n\}$ to P_1 in a random order.
6. P_1 outputs $\{x \in X_1 | H(Decode(\vec{A}_0, x, r)) \in Y\}$.

Figure 6: Malicious protocol Π_{MP-PSI} which realizes $\mathcal{F}_{MP-PSI}$

The proof for the second requirement is entirely consistent and even simpler. Therefore, we choose to omit it.

□

3.2 Multi-Party PSI

After completing the construction of the tool, we move on to the construction of the MP-PSI protocol itself. We present our MP-PSI protocol in Figure 6. Due to the limitations of MP-VOLE, our MP-PSI protocol can only resist $n-2$ corruption parties. We give the proof of correctness and security in Theorem 6.

Theorem 6. *Π_{MP-PSI} realizes the $\mathcal{F}_{MP-PSI}$ functionality against a malicious adversary in the random oracle, $\mathcal{F}_{MP-VOLE}$ − hybrid model.*

Correctness.

- x in the intersection. We just need to prove that $Decode(\vec{A}_0, x, r) = Decode(\vec{A}'_n, x, r) - (x_1 + x_2 + \dots + x_{n-1}) H^{\mathbb{F}}(x, r')$. Note that for each P_i, $(x, H^{\mathbb{F}}(x, r'))$ is encoded to $\vec{Q}_i$. Therefore, $Decode(\vec{Q}_i, x, r) = H^{\mathbb{F}}(x, r')$. Then by the linearity of OKVS and correctness of MP-VOLE, we have $Decode(\vec{A}'_n, x, r) - (x_1 + x_2 + \dots + x_{n-1}) H^{\mathbb{F}}(x, r') =$

$Decode(\vec{A}_n + \vec{A}'_1 \cdot x_1 + \vec{A}'_2 \cdot x_2 + \ldots + \vec{A}'_{n-1} \cdot x_{n-1}, x, r) - (x_1 + x_2 + \ldots + x_{n-1})H^{\mathbb{F}}(x, r') = Decode(\vec{A}_n + \vec{A}_1 \cdot x_1 + \vec{A}_2 \cdot x_2 + \ldots + \vec{A}_{n-1} \cdot x_{n-1}, x, r) + Decode(\vec{Q}_1, x, r) \cdot x_1 + Decode(\vec{Q}_2, x, r) \cdot x_2 + \ldots + Decode(\vec{Q}_{n-1}, x, r) \cdot x_{n-1} - (x_1 + x_2 + \ldots + x_{n-1})H^{\mathbb{F}}(x, r') = Decode(\vec{A}_0, x, r)$, which concludes this situation.

- $x \notin X_1$ or $x \notin X_n$. Note that in the 6th step of the protocol, P_1 only considers the elements in X_1; in the 5th step of the protocol, P_n only considers the elements in X_n. Therefore, x is not output as part of the intersection.

- $x \notin (X_i)_{i\in[2,n-1]}$. In this situation, due to the structure of OKVS, $Decode(\vec{Q}_i, x, r)$ is overwhelmingly not equal to $H^{\mathbb{F}}(x, r')$. Thus the values $Decode(\vec{A}_0, x, r)$ and $Decode(\vec{A}'_n, x, r) - (x_1 + x_2 + \ldots + x_{n-1})H^{\mathbb{F}}(x, r')$ would not match, so x is not output as part of the intersection.

Security.

Note that in the case where multiple parties are corrupted, the method of extracting input is the same as in the case where one party is corrupted. Therefore, we only construct simulators for the latter case.

- **Corrupted** P_1**.** The simulator $\mathcal{S}$ interacts with P_1 as follows:

  1. $\mathcal{S}$ waits for $\mathcal{A}$ to send r, r'.
  2. $\mathcal{S}$ plays the role of $\mathcal{F}_{MP-VOLE}$ and receives $\vec{A}_0, \vec{A}_1$ from $\mathcal{A}$.
  3. When $\mathcal{A}$ sends $\vec{A}'_1$, $\mathcal{S}$ computes $\vec{Q}_1 := \vec{A}'_1 - \vec{A}_1$. For each of the previous $H^{\mathbb{F}}(x, r')$ queries made by $\mathcal{A}$, $\mathcal{S}$ checks if $Decode(\vec{Q}_1, x, r) = F^{\mathbb{F}}(x, r')$ and if so adds x to set X_1. $\mathcal{S}$ sends (sid, X_i) to $\mathcal{F}_{MP-PSI}$.
  4. $\mathcal{S}$ outputs whatever P_1 outputs.

- **Corrupted** $(P_i)_{i\in[2,n-1]}$**.** The simulator $\mathcal{S}$ interacts with P_i as follows:

  1. $\mathcal{S}$ samples $r, r' \xleftarrow{R} \{0,1\}^{\kappa}$ in stand of P_1 and sends it to $\mathcal{A}$.
  2. $\mathcal{S}$ plays the role of $\mathcal{F}_{MP-VOLE}$ and receives $\vec{A}_i$ from $\mathcal{A}$.
  3. When $\mathcal{A}$ sends $\vec{A}'_i$, $\mathcal{S}$ computes $\vec{Q}_i := \vec{A}'_i - \vec{A}_i$. For each of the previous $H^{\mathbb{F}}(x, r')$ queries made by $\mathcal{A}$, $\mathcal{S}$ checks if $Decode(\vec{Q}_i, x, r) = F^{\mathbb{F}}(x, r')$ and if so adds x to set X_i. $\mathcal{S}$ sends (sid, X_i) to $\mathcal{F}_{MP-PSI}$.

- **Corrupted** P_n**.** The simulator $\mathcal{S}$ interacts with P_n as follows:

  1. $\mathcal{S}$ samples $r, r' \xleftarrow{R} \{0,1\}^{\kappa}$ in stand of P_1 and sends it to $\mathcal{A}$.
  2. $\mathcal{S}$ plays the role of $\mathcal{F}_{MP-VOLE}$ and receives $\vec{x} := (x_1, x_2, \ldots, x_{n-1})$ and $\vec{A}_n$ from $\mathcal{A}$.
  3. $\mathcal{S}$ sends $\vec{A}'_i$ to $\mathcal{A}$ in stand of $(P_i)_{i\in[1,n-1]}$.
  4. When $\mathcal{A}$ sends Y, $\mathcal{S}$ computes $\vec{A}'_n = \vec{A}_n + \vec{A}'_1 \cdot x_1 + \vec{A}'_2 \cdot x_2 + \ldots + +\vec{A}'_{n-1} \cdot x_{n-1}$. For each of the previous $H^{\mathbb{F}}(y, r')$ queries made by $\mathcal{A}$, $\mathcal{S}$ checks if $H(Decode(\vec{A}'_n, y, r) - (x_1 + x_2 + \ldots + x_{n-1})H^{\mathbb{F}}(y, r')) \in Y$ and if so adds y to set X_n. $\mathcal{S}$ sends (sid, X_n) to $\mathcal{F}_{MP-PSI}$.

□

Performance Evaluation

We theoretically evaluate the performance of our protocol. In our protocol, each party sends an encrypted OKVS vector to the pivot, which are $nm\kappa$ bits in total, here m is a small multiple of l (if using OKVS in [21], $m \approx 2.4l$). The pivot needs to send a set of size $(\rho + 2logl)l$ bits. Our protocol also requires performing an MP-VOLE of size m, but since only short seeds need to be sent in the communication, we can ignore this part. Therefore, the total communication cost is $O(l(n\kappa + \rho + logl))$. Every communication goes through the pivot, so the bandwidth is also $O(l(n\kappa + \rho + logl))$. In terms of running time, since the operations of each party can be executed in parallel, the overall time should be consistent with the two-party PSI protocol.

Multi-party Circuit-PSI without payload

We make use of the Circuit-PSI protocol of [21], but only if there is no payload. Note that in essence, our multi-party PSI protocol ultimately relies on the two-party PSI protocol of P_1 and P_n. Therefore, using the same method as [21], we can establish an OPPRF between P_1 and P_n, using P_1 as the sender to establish the cuckoo hash. In the absence of payload, we only needs P_1 to give the intersection element to the circuit to perform subsequent operation. However, once each party has some payload, it is undoubtedly insecure to reconstruct the payload of each party through the input of only P_1 and P_n. We leave it as an interesting future work.

References

[1] Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. Psimple: Practical multiparty maliciously-secure private set intersection. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 1098–1112, 2022.

[2] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector ole. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 896–912, 2018.

[3] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom

correlation generators: Silent ot extension and more. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 489–518. Springer, 2019.

[4] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty psi and extensions to circuit/quorum psi. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1182–1204, 2021.

[5] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 95(8):1366–1378, 2012.

[6] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer, 2004.

[7] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*, pages 395–425. Springer, 2021.

[8] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In *Annual International Cryptology Conference*, pages 323–352. Springer, 2022.

[9] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *IACR international workshop on public key cryptography*, pages 175–203. Springer, 2017.

[10] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In *International Conference on Security and Cryptography for Networks*, pages 235–252. Springer, 2018.

[11] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.

[12] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[13] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1257–1272, 2017.

[14] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1151–1165, 2021.

[15] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: fast, malicious private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 739–767. Springer, 2020.

[16] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 122–153. Springer, 2019.

[17] Zhi Qiu, Kang Yang, Yu Yu, and Lijing Zhou. Maliciously secure multi-party psi with lower bandwidth and faster computation. In *International Conference on Information and Communications Security*, pages 69–88. Springer, 2022.

[18] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2505–2517, 2022.

[19] Amanda C Davi Resende and Diego F Aranha. Faster unbalanced private set intersection. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 203–221. Springer, 2018.

[20] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–259. Springer, 2017.

[21] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 901–930. Springer, 2021.

[22] Yingpeng Sang and Hong Shen. Privacy preserving set intersection protocol secure against malicious behaviors. In *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007)*, pages 461–468. IEEE, 2007.

[23] Yingpeng Sang and Hong Shen. Privacy preserving set intersection based on bilinear groups. In *Proceedings of the thirty-first Australasian conference on Computer science-Volume 74*, pages 47–54. Citeseer, 2008.

[24] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. Efficient multi-party private set intersection against malicious adversaries. In *Proceedings of the 2019 ACM SIGSAC conference on cloud computing security workshop*, pages 93–104, 2019.