UNIVERSITY OF LIMOGES
FACULTY OF SCIENCE AND TECHNOLOGY

INTERNSHIP REPORT

Private Set Intersection From Pseudo-random Correlation Generators

Supervisor:

Geoffroy COUTEAU
CNRS research scientist at IRIF
Université de Paris

Submitted By:

Thi-Thuy-Dung BUI

This work was done at IRIF, Université de Paris

February 18, 2024

Contents

Abstract

In this report, we study Private Set Intersection (PSI), a specific scenario in secure multi-party computation, allowing parties to jointly compute the intersection of their inputs without revealing anything more. Private set intersection is not only significant theoretically, but it is also widely used in practice. It has been motivated for many real-life applications. We first revisit the state of the art of PSI, go through all types of PSI protocols while we deeply analyze some breakthrough PSI protocols. Second, we introduce our new PSI protocol, which is very efficient and can be competitive with other current fastest protocols. The idea behind our PSI is constructing a batch obvious pseudorandom function (OPRF) based on subfield oblivious linear-function evaluation (subVOLE) and hashing techniques.

This report is divided into 4 main parts.

- Chapter 1 gives the introduction of both secure multi-party computation and private set intersection, which includes the definition, the motivation, and the brief overview.
- Chapter 2 is used as a description of some important cryptographic primitives used commonly in the private set intersection. Especially, the main technique in our PSI protocol i.e subfield oblivious linear-function evaluation (subVOLE) is presented in detail in section 2.2.7.
- Chapter 3 describes the state-of-art of PSI by classifying all the types of PSI protocols and reviews some currently efficient PSI protocols.
- Finally, in chapter 4, we propose a PSI protocol against a semi-honest adversary and its theoretical comparison with other current PSI protocols.

Acknowledgements

It took me a long time to write down these sentences to express my gratitude to all of my encouragers and supporters.

First of all, I would like to thank my supervisor Geoffroy Couteau for the support he gives me. Under his guidance, my six-month internship became very fruitful. He is a fantastic guide and I learn from him not only the academic knowledge but also the research method. I appreciate the time he gives every week to discuss, analyze problems, and answer my questions. Thank you for all of your encouragement and insightful comments. This advice helps me learn a lot after 6 months.

Secondly, I would like to express my appreciation to Duong Hieu Phan, who brings me to cryptography and also is my supervisor during the first-year summer internship. When I first came to France, everything was difficult when I did not know French, but his advice and encouragement gave me more confidence and strength to be able to complete the CRYPTIS program well.

It is impossible not to mention the lecturers in the CRYPTIS program, who taught me the background and fundamental knowledge about cryptography. Moreover, I would like to thank all of my friends in Limoges, especially the Vietnamese friends. We have a great and unforgettable time together.

Last but not least, I would like to give my appreciation to my family, those who have always silently supported and cheered on every decision in my life. For me, family is one of the most important and valuable gifts of my life.

Chapter 1

Introduction

1.1 Secure Computation

Secure multiparty computation (MPC) protocols allow a group of parties to interact and calculate a joint function of their private inputs while exposing only the output. As a result, secure multi-party computation has progressed from a theoretical interest to a critical tool for developing large-scale privacy-preserving technologies, from biometric identification [YY13] in national security applications and face recognition [OPJM10], to computing market-clearing prices in Denmark [BCD+09]. MPC, was introduced in the 1980s by Yao for the two-party case (FOCS 1986) [Yao86] and by Goldreich, Micali and Wigderson for the multiparty case (STOC 1987) [GMW87], has become prospective research topic. MPC has recently been efficient enough to be utilized in practice, transitioning from a theoretical research to a technology used in industry.

In a secure computation protocol, there are a given number of parties (denoted $P_1, P_2, \cdots, P_n$) and they own their private input respectively $x_1, x_2, \cdots, x_n$. They wish to join and compute a public function $f(x_1, x_2, \cdots, x_n)$ without revealing their inputs.

Informally speaking, the most fundamental figures that a multi-party computation protocol aims to ensure are:

- **Input Privacy:** The messages sent during the protocol's execution provide no information about the parties' private data $(x_1, x_2, \cdots, x_n)$. The only information about the private data that can be deduced is what can be inferred from seeing the function's output only.
- **Correctness:** Any proper subset of adversarial colluding parties willing to share information or deviate from the instructions during the protocol execution should not be able to force honest parties to output an incorrect result. This correctness goal comes in two flavors: either the honest parties are guaranteed to compute the correct output (a "robust" protocol), or they abort if they find an error (a multi-party computation protocol "with abort").

More technically, a protocol's security is determined by comparing the outcomes of a real protocol execution to the outcomes of an ideal computation [BPW04]. In spite of decades of improvements, secure computation remained until about a decade ago mainly in the realm of theoretical research. In recent years, however, the amazing progresses both in secure computation, and in computational power, have brought secure computation to the real world (as witnessed by the creation of several companies offering secure computation solutions, such as Unbound Security[1] and Sharemind[2]). However, the real world use of secure computation is currently *limited* either to scenarios where the parties have a lot of computational power and time (e.g. major companies), or to scenarios where the target functions are very simple and run on small inputs.

[1] `www.unboundtech.com`
[2] `sharemind.cyber.ee`

1.1.1 Additional Definition Parameters

Adversarial power

The security of multi-parties protocol is defined by against the attack of adversary. We will discuss the power of the adversary that attacks a executing protocol and classify these adversaries. There are two main types of adversaries divided by allowed adversarial behaviour:

- *Semi-honest adversaries:* In the semi-honest setting, the corrupted parties correctly follow the protocol description but from the internal states (including all messages received) these corrupted parties try to learn information which is private. Semi-honest adversaries is called 'honest but curious' or 'passive'.
- *Malicious adversaries:* In this model, the corrupted parties can arbitrarily deviate from the protocol specification, according to the adversary's instructions. If a protocol is security against malicious adversaries then no attack can violate the security of it. This type of adversarial model is called 'active'.

Feasibility of MPC

Powerful feasibility findings have been obtained, proving that in the presence of malicious adversaries, every distributed computing function may be safely calculated [GMW87]. We will now summarize the most important of these findings. Let n indicate the number of participating parties and t denote a bound on the number of corrupted parties (where the identity of the corrupted parties is unknown):

1. For $t < n/3$ secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function with computational security assuming a synchronous point-to-point network with authenticated channels.
2. For $t < n/2$ secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function with computational and information-theoretic security, assuming that the parties also have access to a broadcast channel.
3. For $t \geq n/2$ secure multiparty protocols (without fairness or guaranteed output delivery) can be achieved.

In summary, secure multiparty protocols exist for any distributed computing task, but this result is *theoretical*. We do not consider about the efficiency.

1.1.2 Secure computation paradigms.

Secure multi-party computation essentially comes in two flavors. The first approach is typically based upon secret sharing and operates on an arithmetic or a boolean circuit representation of the computed function, such as in the BGW [BOGW88], the CCD [CCD88] protocols or the GMW (Goldreich, Micali, and Wigderson) multi-party protocol [GMW87]. While the security of BGM and CCD bases on the information-theoretic, GMW relies on OT primitive with computational hardness. Secret-sharing-based protocols achieve a lower communication, but require a large number of rounds of interaction, which increases with the circuit depth of the function. In this method, the parties secretly share their inputs before evaluating the circuit gate by gate while maintaining privacy and correctness. They are very efficient but the high number round complexity makes them unsuitable for the networks, where each party is far away. An alternative approach represents the function as a garbled circuit. This approach was used in the original two-party garbled circuit construction of Yao [Yao82] . In the two-party setting, one of the parties "encrypts" the circuit being evaluated, whereas the other party privately evaluates it. Beaver, Micali and Rogaway [BMR90] protocol is a variant of Yao's protocol for multi-party setting.
Yao's original protocol ensures the privacy of each party's input and the correctness of the output under the semi-honest model, in which both parties follow the protocol honestly. The first implementation of Yao's protocol appears in Fairplay systems [BDNP08] by decomposing the

function into a sequence of atomic operation - typically, boolean operations such as exclusive ORs and ANDs. In the years following the introduction of Fairplay, many improvements to Yao's basic protocol have been created, in the form of both efficiency improvements and techniques for active security. These include techniques such as the free XOR method ,cut-and-choose technique [PSSW09], [KSS12], [FN13], Obvious transfer [IPS08], [LOP11] in which AES was used as a benchmark for performance tests.
However, despite these improvements, each secure execution of a circuit still requires a *large* amount of communication between the participants. We stress that this is an extremely strong limitation, which kept secure computation for decades in the realm of interesting theoretical objects with limited hopes of ever becoming practical. With high communication costs, the secure computation cannot possibly be deployed and used on a large scale for data analysis computations, which are our main targets in this project. Therefore, the *communication cost* which is also one of the main bottlenecks of secure computation should be tacked for adapting to large-scale input databases.

1.2 Private Set Intersections

*Private Set Intersection (*PSI*)* computation is a specific scenario in secure multi-party computing (MPC) applications. It not only has important theoretical significance but also has great practical usage. It have been motivated for many *real-life applications* such as Google runs PSI together with third-party data providers to find target audiences for advertising and marketing campaigns [IKN$^+$19] or contact discovery [CLR17]. Private set intersection allows parties to jointly compute the set of all common elements between the data sets of all parties. In the end of PSI protocol, one or both parties should get the correct intersection and will get nothing about other's data sets outside the set intersection. The ideal functionality of 2-party PSI is present as figure 1.1.

PARAMETERS:

- Two parties: a sender and a receiver.
- Set size n for honest parties and n' for corrupt parties.

FUNCTIONALITY:

1. Wait for input $Y = \{y_1, y_2, \dots, \}$ from the receiver. Abort if the receiver is corrupt and $|Y| > n'$.
2. Wait for input $X = \{x_1, x_2, \dots, \}$ from the sender. Abort if the sender is corrupt and $|X| > n'$.
3. Given output $X \cap Y$ to the receiver or sender.

Figure 1.1: Ideal functionality of 2-party PSI

1.2.1 Applications

PSI in Privacy-Preserving Analysis of Medical Data

There are various efficient techniques based on PSI that contributes to the development of the healthcare system. We can not deny the importance of PSI applications for human genomes testing [BBC$^+$11],to comparing patient information, contact tracing [DPT20] to prevent the spread of diseases.

Sharing patient information. As for improving service and facilitating diagnosis, healthcare providers often wish to exchange information about their common patients. However, due to privacy regulations such as the Health Insurance Portability and Accountability Act (HIPAA), they can only share a patient's record if both providers have obtained the patient's consent. Moreover, the two providers may be competitors, and may not wish to leak the identity of the patients who are not in the set intersection. The malicious PSI protocols or Authorized Private Set Intersection (APSI) can be used too address this issue [SSS12]. PSI protocols against malicious adversaries make sure each corrupted provider can not deviate from the protocol, e.g., changing the information of patients in the input, and output sets. About Authorized Private Set Intersection (APSI), it ensures that each providers can only use elements certified by a trusted authority in the intersection protocol.

Privacy-preserving genomics. Genetic testing is a type of medical test that identifies changes in chromosomes, genes, or proteins. Genomic testing is used to diagnose, monitor, treat, predict and prevent disease, as well as promote good health in individuals, across communities, and whole populations. There are 3 important applications of genetic testing: Paternity Tests, Personalized Medicine, Genetic Compatibility Tests [BBC+11]. Implementing these applications in practice has been posing a threat to genomic privacy. To address this challenge on privacy, a set of efficient techniques based on the private set intersection is used efficiently.

Contact tracing. Another concrete application of PSI is for contact tracing to repel an epidemic such as COVID- 19. Global lockdown measures have been imposed all around the world and will cause severe social and economic problems. To relax the lockdown measures while keeping the ability to control the spread of the disease, technical tools for contact tracing have been introduced [DPT20]. The resulting applications try to determine and notice the smartphone owner when exposed to the infectors without disclosing the privacy data of users who have been tested positive. Specifically, PSI is taken advantage of to find the number of elements in the intersection between the data set of the infected patients from healthcare providers and the users' data indicating who already are approximately close to them.

1.2.2 Overview

Several techniques have been proposed that realize the PSI functionality, such as the efficient but insecure naive hashing solution (where the parties hash all entries of their databases and share the hashes), protocols that require a semi-trusted third party, or two-party PSI protocols (which have no trusted third party, and no leakage of information like in the naive solution). The earliest proposed two-party PSI protocols were special-purpose solutions based on public-key cryptography as the Diffie-Hellman assumption [Sha80, Mea86], which had a prohibitive computational cost. Later, solutions were proposed using circuit-based generic techniques for secure computation, that are mostly based on symmetric cryptography [HEK12a] and used hashing technique for achieving linear communication, [PSTY19, PRTY19]. The most recent and efficient developments are PSI protocols that are based on oblivious transfer (OT) alone[3], and combine the efficiency of symmetric cryptographic primitives with special purpose optimizations [PSZ14, KKRT16, RR17, KRTW19, PSWW18, PRTY19, PRTY20, CM20]. Among these protocol, recent works based on OT [CM20, PRTY19] tried to balance between communication and computation and construct a lightweight multi-point Obvious Peusdorandom Function (OPRF [FIPR05]), particularly suited for cloud computing settings. Additionally, there are some semi-honest two-parties protocols [KKRT16, PRTY19] that take advantage of advanced hashing methods such as Cuckoo hashing and Bloom filter to reduce the communication. Recently, there has been active work on efficient secure semi-honest PSI protocol with fast implementations that can process millions of items in seconds. However, malicious protocols are much slower, the works [RR17, PRTY20], are the best of in terms of concrete efficiency.

[3] OT is a secure computation primitive which can be realized extremely efficiently and is used as a building block in larger protocols.

Chapter 2

Preliminaries

2.1 Notation and Security

2.1.1 Notation

Throughout the paper we use the following notation: let κ, λ denote the computational and statistical security parameters, respectively. We write $[m]$ to denote a set $\{1, 2, \ldots, m\}$. A function $\mu(.)$ is negligible if for every positive polynomial $p(.)$ and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$.
The notation $\binom{2}{1}\mathsf{OT}_m^t$ is used to denote t instances of 1-out-of-2 string OT where strings are m bits long. The field is denoted $\mathbb{F}_q$ field extension of an arbitrary subfield base $\mathbb{F}_p$, where $q = p^r$, p is a power of prime number. All operations in our paper is computed over the field.
For a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, x_i is the i'th position in $\mathbf{x}$. Given a matrix T, t^i and t_j are corresponding to the $i-$th column and $j-$th row of matrix T. If $\mathbf{a} = a_1||\ldots||a_n \in \{0,1\}^n$ and $\mathbf{b} = b_1||\ldots||b_n \in \{0,1\}^n$ are two vector then we define $\oplus$ and . as follows. We use the notion $\mathbf{a} \oplus \mathbf{b}$ to denote the vector $(a_1 \oplus b_1)||\ldots(a_n \oplus b_n)$. Similarly, the notion $\mathbf{a}.\mathbf{b}$ denotes the vector $(a_1.b_1)||\ldots||(a_n.b_n)$. For $c \in \{0,1\}$, $c.\mathbf{a}$ denotes the vector $(c.a_1)||\ldots||(c.a_n)$. We use $||x||$ to denote the Hamming weight of a binary string x.
In this report, all PSI protocol are considered in the 2-party private set intersection PSI with semi-honest security. The sender Alice and the receiver Bob have 2 input sets $X = \{x_1, \ldots, x_{n_1}\}$, $Y = \{y_1, \ldots, y_{n_2}\}$ respectively.

2.1.2 Security model

Simulation paradigm. PSI is a special case of secure computation. The proof of two-party PSI protocol is followed by security of two-party computation. Security of secure computation protocol is proved via the *simulation* paradigm, which is a method of comparing what occurs in the "real world" to what occurs in a "ideal world" in which the primitive in question is secure by definition. Simulation paradigm builds a simulator in the alternative world that is secure by definition, and it creates a view in the actual world for the adversary that is *computationally indistinguishable* from its real view.

Definition 2.1 (Computationally indistinguishable) *Let $\{X_n\}, \{Y_n\}$ be sequences of distribution with $\{X_n\}, \{Y_n\}$ ranging over $\{0,1\}^{l(n)}$ for some $l(n) = n^{O(1)}$. $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable (notation: $X_n \approx Y_n$) if for every non-uniform polynomial-time algorithm A there exists a negligible function $\mu(.)$ such that for sufficiently large n*

$$\big|Pr[\mathsf{A}(X_n) = 1] - Pr[\mathsf{A}(Y_n) = 1]\big| < \mu(n)$$

Two-party computation. A two-party protocol problem is defined by determining a possibly random procedure that assigns a pairs of inputs to a pairs of outputs (one for each party). This procedure is referred to as *functionality*, and it is denoted by $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0,1\}^*$, the output is a pair of random

variables $(f_1(x,y), f_2(x,y))$. One party (with input x) wishes to learn $f_1(x,y)$ and the other (with input y) wishes to learn $f_2(x,y)$.

Simulation for semi-honest adversary. The model considered is a two-party computing paradigm with static semi-honest adversaries. One of the parties is controlled (statically, at the beginning of computation) by an adversary who strictly follows to the protocol specification. By examining the messages it has received and its internal state, it may, nevertheless, attempt to discover more information than is permitted. Even if the adversary chooses its random tape in an un-random approach, it may be able to totally disrupt the protocol. Nevertheless, a protocol that is safe in the presence of semi-honest adversaries ensures that no information is accidentally leaked. Defining and proving security for semi-honest adversaries is considerably easier than malicious adversaries since we know exactly what the adversary will do.

Semi-honest security. We begin with the following notation:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality and let π be a two-party computation protocol for evaluating f.
- The view of the ith party ($i \in \{1,2\}$) during an execution of π on (x,y) and security parameter n is denoted by $\mathbf{view}_i^\pi(x,y,n)$ and equals $(w, r^i; m^i, 1, \ldots, m_t^i)$, where $w \in \{x,y\}$ (its input depending on the value of i), r^i equals the contents of the ith party's internal random tape, and m^i, j represents the jth message that it received.
- The output of the i'th party during an execution of π on (x,y) and security parameter n is denoted by $\mathbf{output}_i^\pi(x,y,n)$ and can be computed from its own view of the execution. We denote the joint output of both parties by $\mathbf{output}^\pi(x,y,n) = \mathbf{output}_1^\pi(x,y,n), \mathbf{output}_2^\pi(x,y,n))$.

Definition 2.2 *Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f against semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that*

$$\{S_1(1^n, x, f_1(x,y)), f(x,y))\}_{x,y,n} \approx \{(\boldsymbol{view}_1^\pi(x,y,n), \boldsymbol{output}^\pi(x,y,n))\}_{x,y,n},$$
$$\{S_2(1^n, y, f_2(x,y)), f(x,y))\}_{x,y,n} \approx \{(\boldsymbol{view}_2^\pi(x,y,n), \boldsymbol{output}^\pi(x,y,n))\}_{x,y,n},$$

where $x, y \in \{0,1\}^*$ such that $|x| = |y|$, and $n \in \mathbb{N}$

2.2 Preliminaries

2.2.1 Obvious transfer (OT)

OT was introduced by M.Rabin [Rab05], is a central cryptographic primitive in the area of secure computation, in which the receiver learns nothing more than one string s_r from two input strings s_0, s_1 of the sender and sender learn nothing about bit r. An oblivious transfer (OT) protocol is a type of protocol in which a sender transfers one of potentially many pieces of information to a receiver, but remains oblivious as to what piece (if any) has been transferred. The cost for constructing OT is expensive so the idea of expanding a small number of OT to obtain large instances of OT has been developed [Bea96].
When executing m invocations of 1-out-2 OT on l-bit strings (denoted $\binom{2}{1}\mathsf{OT}_l^m$), the sender S holds m message pairs $(x_{j,0}, x_{j,1})$ with $x_{j,0}, x_{j,1} \in \{0,1\}^l$, while the receiver R holds an m-bit choice vector $\mathbf{r}$. At the end of the protocol, R receives x_{j,r_j} but learns nothing about $x_{j,1-r_j}$ and S learns nothing about $\mathbf{r}$. The ideal functionality of OT_l^m is presented in figure 2.1. Many OT protocols have been proposed in [CO15], [NP01].
To make the oblivious transfer is more efficient, we can use OT **extension** [IKNP03] and **random** OT [NNOB12] [ALSZ13].

INPUTS:

- The sender S holds m message pairs $(x_{j,0}, x_{j,1})$ $1 \leq j \leq m$ with $x_{j,0}, x_{j,1} \in \{0,1\}^l$.
- The receiver R holds an m-bit choice vector $\mathbf{r} = (r_1, \ldots, r_m)$.

OUTPUTS:

- R outputs x_{j,r_j} for $1 \leq j \leq m$.
- S has no output.

Figure 2.1: Ideal functionality of $\binom{2}{1}\mathsf{OT}_l^m$

OT extension

In recent years, OT extension is efficiently deployed to construct PSI protocol with very practical both communication and computation cost. The first truly practical OT extension is proposed in [IKNP03]. protocol is state of art for PSI in a semi-honest setting, it is the core idea adapted to the several protocol [PSZ14, KKRT16, KRTW19, PRTY19, PRTY20, CM20] and obtain the extremely fast PSI protocols with a balance between communication and computation cost. The $\binom{2}{1}\mathsf{OT}_t^\kappa$ $(t > \kappa)$ can be directly implemented from $\binom{2}{1}\mathsf{OT}_\kappa^\kappa$: the sender associates two κ-bit keys to each pair of messages and obliviously transfer one key of each pair to the receiver. Then, the receiver stretches two t-bit strings from the two keys of each pair, using a pseudo-random generator, and sends the XOR of each of these strings and the corresponding message to the receiver. Then, the $\binom{2}{1}\mathsf{OT}_m^t$ can be obtained by one call $\binom{2}{1}\mathsf{OT}_t^\kappa$ under the assumption of correlation-robust hash function (section 2.2.3). The total communication cost for reduction from $\binom{2}{1}\mathsf{OT}_m^t$ to $\binom{2}{1}\mathsf{OT}_\kappa^\kappa$ is $2tm + 2t\kappa$ bits.
An optimization to the protocol of [IKNP03] was proposed in [ALSZ13] (and discovered independently in [KK13]. [KK13] protocol replaces the repeating code in [IKNP03] with Walsh-Hadamard (WH) code to construct OT extension for short strings. This protocol reduce $\binom{2}{1}\mathsf{OT}_m^t$ to $\binom{2}{1}\mathsf{OT}_\kappa^\kappa$ with the with $t(2\kappa/\log n + nm)$ bits of communication, n being a parameter that can be chosen arbitrarily so as to minimize this costs. While [ALSZ13] describe the OT extension on inputs satisfying some particular conditions. In particular, the communication of the OT extension protocol can be reduced from $2tm + 2t\kappa$ bits to $tm + t\kappa$ bits when the inputs to each OT are correlated.

2.2.2 The random oracle model

A random oracle is an oracle (a theoretical black box) that responds to every unique query with a (truly) random response chosen uniformly from its output domain. If a query is repeated, it responds the same way every time that query is submitted. Stated differently, a random oracle is a mathematical function chosen uniformly at random, that is, a function mapping each possible query to a (fixed) random response from its output domain. Random oracles as a mathematical abstraction are typically used when the proof cannot be carried out using weaker assumptions on the cryptographic hash function. A system that is proven secure when every hash function is replaced by a random oracle is described as being secure in the random oracle model, as opposed to secure in the standard model of cryptography.
The efficiency gain in using the random oracle model is particularly true with regards to protocols for private set intersection. The public-key-based protocols (based on Diffie-Hellman and blind-RSA) use a hash function $\mathsf{H}()$ that must be modeled as a random oracle, or modeled using another non-standard assumption. The other protocols (the generic protocol, as well as the protocol based on Bloom filters and the new OT-based protocol) can be implemented without a random oracle assumption, but in order to speedup the computation of OT in these protocols we must use random OT extension, whose efficient implementation relies on a function that is typically modeled as a random oracle.

2.2.3 Correlation robustness

Our OPRF is proven secure under correlation robustness on the hash function. Correlation robustness was firstly introduced in [IKNP03], later generalized in [KKRT16,PRTY19,KK13,CM20] but this assumption considered in Hamming weight while our assumption works over any finite field. We say that $\mathsf{H} : \{0,1\}^k \to \{0,1\}$ is *correlation robust* if for a random and independent choice of (polynomial many) strings $s, t_1, \ldots, t_m\{0,1\}^k$, the joint distribution $(\mathsf{H}(t_1 \oplus s), \ldots, \mathsf{H}(t_m \oplus s))$ is pseudorandom given $t_1, \ldots, t_m$.

Definition 2.3 (Hamming Correlation Robustness) *Let H be a hash function with input length n. Then H is d-Hamming correlation robust if, for any $a_1, \ldots, a_m, b_1, \ldots, b_m \in \{0,1\}^n$ with $\| b_i \|_{\mathsf{H}} \geq d$ for each $i \in [m]$, the following distribution, induced by random sampling of $s \leftarrow \{0,1\}^n$ is pseudorandom. Namely,*

$$(\mathsf{H}(a_1 \oplus [b_1.s]), \ldots, \mathsf{H}(a_m \oplus [b_m.s])) \overset{c}{\approx} (\mathsf{F}(a_1 \oplus [b_1.s]), \ldots, \mathsf{F}(a_m \oplus [b_m.s])$$

where F is a random function.

This is a weaker assumption than random oracle model. The [IKNP03] protocol uses this assumption with $n = d = \kappa$. In that case, the only valid choice for b_i is 1^κ and the distribution simplifies to $\mathsf{H}(a_1 \oplus s), \ldots, \mathsf{H}(a_m \oplus s)$.

Definition 2.4 ($(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-Correlation robustness.) *Let $\mathbb{F}_q$ be a finite field, $q = p^r \approx O(2^{\kappa + \log n})$ where p is power of prime, H be a hash function $\mathsf{H} : \{0,1\}^* \times \mathbb{F}_q \to \{0,1\}^v$. Then H is a $\mathbb{F}_q$ correlation robust if for any distinct strings $t_1, t_2, \ldots, t_n \in \{0,1\}^*$; $u_1, u_2, \ldots, u_n \in \mathbb{F}_p \backslash \{0\}$, $v_1, v_2, \ldots, v_n \in \mathbb{F}_q$, the following distribution, induced by random sampling of $\Delta \leftarrow \mathbb{F}_q$, is pseudorandom:*

$$\mathsf{H}(t_1, v_1 - \Delta u_1), \mathsf{H}(t_2, v_2 - \Delta u_2), \ldots, \mathsf{H}(t_n, v_n - \Delta u_n)$$

We state this definition in form of attack game. Given a $(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-correlation robust hash function $\mathsf{H} : \{0,1\}^* \times \mathbb{F}_q \to \{0,1\}^v$ and for a given an adversary $\mathcal{A}$. A PPT $\mathcal{A}$ is unbounded adversary, can make at most Q queries. The hash function H is secure if the advantage of any PPT $\mathcal{A}$ in the following game is negligible:

1. $\mathcal{A}$ chooses a distinct sequence $t_1, t_2, \ldots, t_n \in \{0,1\}^*$; $u_1, u_2, \ldots, u_n \in \mathbb{F}_p \backslash \{0\}$, $v_1, v_2, \ldots, v_n \in \mathbb{F}_q$ and then submits this sequence to challenger.
2. Challenger chooses randomly $\Delta \leftarrow \mathbb{F}_q$ then gets sequence of strings $z_i = (t_i, v_i - \Delta u_i)_{i \leq n} \in \mathbb{F}_q$ and tosses the coin $b \leftarrow \{0,1\}$.
   - If $b = 0$, challenger returns to $\mathcal{A}$ n strings: $\mathsf{H}(z_1), \mathsf{H}(z_2), \ldots, \mathsf{H}(z_n)$.
   - If $b = 1$, challenger chooses random from uniform distribution n strings of length v and outputs them to $\mathcal{A}$.
3. The $\mathcal{A}$ gets sequence $w_1, w_2, \ldots, w_n \in \{0,1\}^v$. Note that the $\mathcal{A}$ can make Q queries then outputs a bit b'. He *wins* if $b = b'$. The advantage of $\mathcal{A}$ is defined by:

$$\mathrm{Adv}(\mathcal{A}, \mathsf{H}) = \Pr(\mathcal{A} \textit{ wins}) - 1/2 = \epsilon$$

The adversary can make Q queries to the random oracle in which if a query x is already queried then it returns $\mathsf{H}(x)$ if not a random value is returned. The adversary $\mathcal{A}$ queries Q times, each time $\mathcal{A}$ queries an arbitrary sequence of n string in the domain of H. Let L be the list of queries from $\mathcal{A}$. We can see that ϵ is negligible under an PPT adversary $\mathcal{A}$. Indeed,

$$\begin{aligned}
\Pr[\mathcal{A} \textit{ wins}] &= \Pr[\mathcal{A} \textit{ wins} \mid \exists i \in [n],\ z_i \in L] + \Pr[\mathcal{A} \textit{ wins} \mid \nexists i \in [n],\ z_i \in L] \\
&\leq 1 \,.\, \sum_{i \leq n} \Pr[z_i \in L] \ + \ \frac{1}{2} \,.\, 1 \\
&\leq \frac{1}{2} \ + \ \sum_{1 \leq n} \sum_{i \leq Q} \Pr[z_i = q_j] \\
&= \frac{1}{2} \ + \ \frac{nQ}{q} \ \leq \ \frac{1}{2} \ + \ \frac{Q}{2^\kappa}
\end{aligned}$$

2.2.4 Obvious pseudorandom function (OPRF)

An oblivious pseudo-random function [FIPR05] is a function $F : (\{0,1\}^\kappa \times \{0,1\}^\sigma) \to (\perp, \{0,1\}^l)$ that, given a key k from P_1 and an input element x from P_2, computes and outputs $F_k(x)$ to P_2. P_1 obtains no output and learns no information about x while P_2 learns no information about k. We describe the ideal functionality of OPRF in figure 2.2.

PARAMETERS:

- A pseudorandom function $\mathbb{F}$.
- The receiver R inputs set $(x_1, \ldots, x_n)$ and the sender S inputs nothing.

PROTOCOL:

- The sender S learns a key k.
- The receiver R learns the PRF output $(\mathsf{PRF}_k(x_1), \ldots, \mathsf{PRF}_k(x_n))$

Figure 2.2: The ideal functionality of OPRF

OPRF is the common approach for PSI, which can construct a balance between communication and computation cost. The receiver inputs x and obtain the $F_k(x)$ where key k is given by sender and $\mathbb{F}$ is a PRF. The sender has the key so the sender can compute PRF value on any input. A PSI protocol based on OPRF is constructed by firstly receiver computes the set OPRF for all of his input, permutes this set and sends to sender, the sender can compare with her PRF set then output their intersection. The construction of OPRF for PSI protocol can be divided into single point OPRF and multi-point OPRF. These approaches is discussed detailed in section 3.4. There exist several installations for OPRF, described in [FIPR05]: based on generic secure computation techniques (using an AES circuit), based on the Diffie-Hellman assumption, or based on OT.

2.2.5 Hashing techniques

Hashing techniques has provided an important role in constructing PSI protocol. Using hash functions, the elements in a set can be divided into small bins. Then, the parties can just compare elements in each bin and output the intersection set. Therefore, for pairwise comparisons, the average number of comparisons between items can be decreased from $O(n^2)$ to $O(n)$. In this section, we revisit the simple hashing, Cuckoo hashing schemes and also describe how to use both hashing schemes in the context of PSI. The detailed practical parameter of these hashing schemes is presented in [PSZ18].

Simple hashing

In the context of PSI, a simple hashing scheme is the simplest but it is very efficient when corporate with other hashing schemes such that Cuckoo hashing. Given a input set X, a hash function $\mathsf{H} : \{0,1\}^* \to [1,b]$ is chosen independently of the input elements and a hash table consists of b empty bins $B_1, \cdots, B_b$, this hashing scheme assigns each element $x \in X$ into a bin $B_{\mathsf{H}(x)}$. Then a bin can contain more than one element.
For applying simple hashing in PSI, both parties can map their input set to b bins and then compare between the elements in each bin. To hire the information of items mapped to bin, we must conceal the number of items mapped to a bin, the parties must pad their bins with dummy elements to contain $\max_b$ elements. When $n = b$, $\max_b = \frac{\ln n}{\ln \ln n}(1 + o(1))$.

Cuckoo hashing

Cuckoo hashing scheme [PR04] uses hash functions to map elements into bin such that each bin contains at most one element. Specifically, it uses k hash functions $h_1, \cdots, h_k : \{0,1\}^* \to [1,b]$, which are chosen randomly and independently of the input elements, to map n elements to $b = \epsilon n$

bins. A element x firstly is assigned to bin $B_{h_1(x)}$. If this bin contains another element y then y is evicted to a new bin $B_{h_i(y)}$ where $i \in [k]$ is chosen randomly and $h_i(y) \neq h_1(y)$. The procedure is continued until no further evictions are required, or until a certain number of relocations have been performed. In the latter case, the last elements are placed in a stash s.
A lookup in this method is highly efficient since it simply compares x to the k items in k bin: $B_{h_1(x)}, \cdots, B_{h_k(x)}$ and to the s items in the stash. The size of the hash table is determined by the number of hash functions k and the stash size s. The greater the value of k selected, the more likely the insertion procedure will succeed, and therefore the number of bins b will be reduced. The greater the value of s, on the other hand, the more insertion failures may be allowed.
There is a problem when using Cuckoo hashing scheme for PSI: the relocations in scheme is random so an item applies Cuckoo hashing schemes twice with the same parameters can be assigned to different bins. Therefore, if both parties the S and R use Cuckoo hashing to hash their inputs then they can not compare and obtain the intersection set. To deal with this problems, the solution is that one party uses Cuckoo hashing and other uses simple hashing scheme for the same number of hash functions used in Cuckoo hashing. In addition, we must ensure that the parameter chosen for Cuckoo hashing is appropriate such that the failure probability is less than $2^{-\lambda}$ so the information of input set is security. Moreover, as mentioned above, when one uses simple hashing it requires to pad each bin to size max_b using dummy elements and all dummy elements are distinct.
The best performance is obtained when the size of input set of the sender and receiver is equal. All protocols combining Cuckoo hashing and simple hashing ([KKRT16, PSSZ15] uses 2 hash functions to map n elements into $2.4n$ bins with small size of stash s for the failure probability $O(2^{-\lambda})$.

Bloom Filter-Based PSI

The protocol [DCW13] is based on OT and a novel two-party computation approach , which makes use of a new variant of Bloom filters that is called garbled Bloom filters. In general, there is no difference between a garbled Bloom filter and a Bloom filter: it encodes a set of n elements in an array of length m and it has no false negative and negligible false positive. To add an item, it is mapped into k index numbers using k randomly uniform hash functions, and the relevant array positions are set. To query an element, the element is mapped into k index numbers using the same k hash functions, and the associated array positions are verified.
A Bloom filter has a set of k independent uniform hash functions $\mathsf{H} = \{h_0, \ldots, h_{k-1}\}$, each of which h_i uniformly maps items to index numbers in the range $[0, m-1]$. We use BF_S to represent a Bloom filter parameterized by (m, n, k, H), $\mathrm{BF}_S[i]$ to denote the bit at index i in BF_S.
Firstly, in the array of length $[m]$, all of the array's bits are set to 0. To enter an element $x \in S$ into the filter, the element is hashed using the k hash functions to obtain the k index numbers. At all of these indices, the bits in the bit array are set to 1 i.e. set $\mathrm{BF}_S[h_i(x)] = 1$ for $0 \leq i \leq k-1$. To determine if an item y is in S, y is hashed using the k hash algorithms, and all places y hashes to are examined. If any of the bits at the places are 0, y is not in S; otherwise, y is most likely in S.
So a Bloom filter never returns a false negative since hash functions are deterministic and if y is encoded in the filter then in the query phase every $\mathrm{BF}_S[h_i(y)]$ must be 1. Although it is conceivable for a false positive to occur, for example if y is not in the set S, yet all $\mathrm{BF}_S[h_i(y)]$ are set to 1. In the Bloom filter, the chance of a specific bit being 1 in a particular case is given by $p = 1 - (1 - 1/m)^{kn}$, and the upper bound of the false positive probability is:

$$\epsilon = p^k \times \left(1 + O\left(\frac{k}{p}\sqrt{\frac{lnm - klnp}{m}}\right)\right)$$

which is negligible in k.
If we have two (m, n, k, H)-Bloom filters that each encode a set of S_1 and S_2, we can get another (m, n, k, H)-Bloom filter $\mathrm{BF}_{S_1 \cap S_2}$ by bit-wise ANDing BF_{S_1} and BF_{S_2}. The resultant Bloom filter, on the other hand, generally includes more 1 bits than a Bloom filter created from start with $S_1 \cap S_2$. Then we'll need to use a technique called Garbled Bloom Filter (GBF).
To minimize unexpected information leakage while utilizing Bloom filters for PSI, the creators

of [DCW13] created the Garbled Bloom Filter, a variation of the BF (GBF). A GBF G applies the same k hash functions as a BF, but instead of single bits, it keeps shares of length l at each place $G[i]$, for $1 \leq i \leq m$. These shares are selected uniformly at random, subject to the requirement that $\bigoplus_{j=0}^{k} G[h_j(x)] = x$ for every element x contained in the filter G.
To represent a set X with a GBF G, all locations of G are designated as empty at first. After that, each element $x \in X$ is put as follows. First, the insertion algorithm looks for a hash function $tin[1, \ldots, k]$ that is unoccupied by $G[h_t(x)]$ (the probability of not finding such a function is equal to the probability of a false positive in the BF, which is negligible due to the choice of parameters). All other unoccupied positions $G[h_j(x)]$ are assigned to l-bit shares at random. Last but not least, to achieve a valid sharing of x, $G[h_t(x)]$ is set to $G[h_t(x)] = x \oplus (\bigoplus_{j=1, j\neq t}^{k} G[h_j(x)])$. We highlight that the construction of the GBF cannot be entirely paralleled since existing shares must be re-used.
In the semi-honest secure PSI protocol of [DCW13], P_1 generates a m-bit GBF G_X from its set X and P_2 generates a m-bit BF F_Y from its set Y . P_1 and P_2 then perform OT_l^m , where for the i-th OT P_1 acts as a sender with input $(0, GX[i])$ and P_2 acts as a receiver with choice bit $F_Y[i]$. Thereby, P_2 obtains an intersection GBF$G_{(X \wedge Y)}$, for which $G(X \wedge Y)[i] = 0$ if $F_Y[i] = 0$ and $G(X \wedge Y)[i] = G_X[i]$ if $F_Y[i] = 1$. P_2 can check whether an element y is in the intersection by checking whether $\bigoplus_{i=1}^{k} {}_{X \wedge Y} G[h_j(x)]$ is equal to y or not. (Note that P_2 cannot perform this check for any value which is not in its input set, since the probability that it learns all GBF locations associated with that value is equal to the probability of a false positive, which is negligible due to the choice of parameters).

2.2.6 Oblivious Key-Value Store (OKVS)

Many recent private set intersection protocols use obvious key value store to hide their input. [PRTY19] encode input sets as a polynomial or [PRTY20, RS21] use a linear solver technique which is called PaXoS for mapping a set of input to corresponding value. Informally speaking, obvious key value store (OKVS) [GPR+21] is a data structure that compactly represents a desired mapping $k_i \to r_i$. When the v_i values are random, the OKVS data structure hides the k_i values that were used to generate it. The simplest (and size-optimal) OKVS is a polynomial p that is chosen using interpolation such that $p(k_i) = v_i$.

Definition 2.5 *A key-value store is parameterized by a set of $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of function* H*, and consists of two algorithms:*

- $\mathsf{Encode}_{\mathsf{H}}$ *takes as input a set of (k_i, v_i) key-value pairs and outputs an object S (or, with statistically small probability, an error indicator $\perp$.*
- $\mathsf{Decode}_{\mathsf{H}}$ *takes as input an object S, a key k and outputs a value v.*

A KVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(k, v) \in A \quad \textit{and} \quad \perp \neq S \leftarrow \mathsf{Encode}_{\mathsf{H}}(A) \implies \mathsf{Decode}_{\mathsf{H}}(S, k) = v$$

There exists some approaches OKVS for PSI protocols e will discuss these method and their cost. Normally, given n pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, the Encode algorithm will output the vector S of length m. We use $r = m/n$ to denote the rate encoding which describes how compact the encoding is, i.e n items can be encoded as m element vector.

Interpolation polynomial

The most well-know method for encoding the input is based on polynomial. This method heavy replies on interpolation polynomial over a arbitrary field $\mathbb{F}_p$ which constructs a polynomial P with n coefficients in $\mathbb{F}_q$ such that $P(x_i) = y_i$ for all $i \in [n]$. Hence, S is presented by n coefficient of P and the rate encoding $r = 1$. This rate is optimal and can help protocol achieve a better communication overhead when sending the polynomial P.
However, finding polynomial P by polynomial interpolation requires $O(n \log^2 n)$ time and decoding n points also requires $O(n \log^2 n)$ time. Some available methods reduce the time for interpolation around $O(n \log n)$ but it still can be expensive for large n.

Garble Bloom Filters

A GFB is an array $GFB[1,\ldots,N]$ of strings, associated with a set of hash functions $h_1,\ldots,h_k : \{0,1\}^* \to [N]$. For an appropriate parameter, the GFB implements a key-value store, where the value associated with key x_i is:

$$\bigoplus_{j=1}^{k} GFB[h_j(x_i)] = y_i \quad \text{for all} \quad i \in [n]$$

To limit the probability by $2^{-\lambda}$, a table with $N = n\lambda \log e$ entries can be used to store n items. In such situation, λ is the ideal number of hash functions. If we set $\lambda = 40$, the table size is about $60n$ and the number of hash functions is $k = 40$. Furthermore, by using less hashing [KM08], each insert requires just two hash functions $h_1(x)$ and $h_2(x)$. Taking $h_i(x) = h_1(x) + i \times h_2(x)$ simulates the extra $k-2$ hash functions $h_i(x), i \in [3,k]$.
The Bloom filter-based construction has an advantage over the polynomial-based construction in that the insertion method runs in time $O(n)$ rather than $O(n^2)$, and it is also highly efficient in practice. The communication is still $O(n)$, but the constant coefficient is huge (the actual communication is $60n$ items rather than n), and therefore communication might be a bottleneck, particularly on slow networks.

PaXoS

The most efficient linear solver approach is Probe-and-XOR of Strings (PaXoS), is firstly introduced in [PRTY20]. Its implementation based on Cuckoo graph with efficient encoding and decoding algorithms. The ideal of PaXoS scheme is combined between Garble Bloom Filter and Cuckoo hashing to obtain a encoding and decoding algorithms with linear time Recent work [RS21] has proposed an optimal version of PaXoS.
We generalize the PaXoS algorithms for n pairs $\{(x_1,y_1),\ldots,(x_n,y_n)\}$ in a finite group $\mathbb{G}$. Given an integer $m \le n$, security parameter λ and a matrix $M \in \{0,1\}^{n\times m}$ is chosen dependently of the input set. The Encode algorithm will outputs $\mathbf{P} \in \mathbb{G}^m$ s.t.

$$M\mathbf{P}^{\mathsf{T}} = (y_1,\ldots,y_n)^{\mathsf{T}}$$

The target value $(y_1,\ldots,y_n) \in \mathbb{G}$ can be arbitrary. It requires the solver to output a solution with probability $1 - O(2^{-\lambda})$.
The matrix M includes n rows such that i-th row is defined by a random function $\mathtt{row}(x_i; r)$, where $r \in \{0,1\}^\kappa$ as a random seed. We obtain $\langle \mathtt{row}(x_i,r), \mathbf{P}\rangle = y_i$ then $\mathsf{Decode}(\mathbf{P}, x_i, r) := \langle \mathtt{row}(x_i,r), \mathbf{P}\rangle$.
We now present the PaXoS solver [PRTY20] in detail. The matrix $M \in \{0,1\}^{n\times m}$ is constructed on Cuckoo graph. Recall that Cuckoo hashing uses 2 hash functions h_1 and h_2 to assign a item x into either $h_1(x)$ or $h_2(x)$ position. The matrix M consists of 2 sub-matrices $M = M'|M*$ such that $M' \in \{0,1\}^{n\times m'}$ where $m' = 2.4n$. The sub-matrix M' is constructed by $2.4n$ first columns of matrix M and the weight of each row of M' is exactly 2. In $i-$th row of matrix M', two position $h_1(x_i)$ and $h_2(x_i)$ is assigned to 1 while other positions is 0. Let $\mathcal{G}$ be the graph consisting of m' vertices $\mathcal{V} = [m']$ and the edge set $\mathcal{E} = \{(c_0,c_1) | i \in [n] \wedge M'_{i,c_0} = M'_{i,c_1} = 1\}$. That is, for each constraint $y_i = \langle \mathtt{row}(z_i,r), \mathbf{P}\rangle = P_{c_0} + P_{c_1} + \ldots$ there is an edge between vertices $(c_0; c_1) = e_i$. $\mathcal{G}$ is called the Cuckoo-graph.
First, let us assume that $\mathcal{G}$ has no cycles and therefore consists of one or more trees. This case can be solved by doing a linear pass over the nodes along tree edges, and assigning values on the way. In particular:

1. Initialize $P_i := 0$ for $i \in [m]$.
2. Let $I \subseteq \mathcal{V}$ s.t. each tree in $\mathcal{G}$ has a single vertex in I and $I := V \setminus I$.
3. Pick an $i \in I$ and for each edge $(j;i) \in \mathcal{E}$ such that $j \in \bar{I}$, identify $e_k \in \mathcal{E}$, i.e. $M'_{k,i} = M'_{k,j} = 1$, and update $P_j := y_k - P_i$. Note that because $\mathcal{G}$ is acyclic, P_i will not change value later. Update $I := I \cup \{j\}$.

4. Finally, define $I := I \setminus \{i\}$, $\bar{I} := \bar{I} \cup \{i\}$ and if $I \neq \emptyset$, go back to (3).

This approach does not work if $\mathcal{G}$ contains a cycle, because P_j will have already been modified at some time in step (3). To address this, the solver first determines the so-called 2-core graph $\tilde{\mathcal{G}}$, which is a subgraph of $\mathcal{G}$ that only contains the cycles and any paths between them. It is worth noting that the graph produced by $\mathcal{G} \setminus \tilde{\mathcal{G}}$ is acyclic.
The solver uses Gaussian elimination to solve the constraints contained in $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ with the use of the $m - m'$ additional columns of M. In particular, Pinkas et al. [PRTY20] show that for $m' = 2.4n$, the size of $\tilde{\mathcal{E}}$ is bounded by $d = O(\lambda)$ with overwhelming probability. Let the actual number of edges in $\tilde{\mathcal{G}}$ be $d < \tilde{d}$. They then consider the submatix formed by the last $m - m'$ columns of M and the rows corresponding to edges in $\tilde{\mathcal{G}}$. In their parameterization they set $m = d + \lambda + m'$. As such $\tilde{M}$ is a $(d + \lambda) \times \tilde{d}$ random binary matrix. With probability $1 - O(2^{-\lambda})$ there exists an invertible $\tilde{d} \times \tilde{d}$ submatrix $\tilde{M}*$ within . The constraints in $\tilde{\mathcal{G}}$ can then be solved for using Gaussian elimination on $\tilde{M}*$ which requires $O(\tilde{d}^3) = O(\lambda^3)$ time. The remaining P_i values corresponding to $\tilde{\mathcal{G}}$ and are assigned the value zero, and the remaining constraints in $\mathcal{G} \setminus \tilde{\mathcal{G}}$ can then be solved using the linear time algorithm described above.
The encode and decode algorithms both run in linear time. The rate encoding of PaXoS is $r \approx 2.4$, compared to Grable Bloom Filter, this rate is very efficient.

2.2.7 Subfield VOLE

The notion of a *pseudorandom correlation generator (PCG)*, recently proposed and studied by Boyle et al. [BCGI18,BCG+19a], enables to generate and store the correlated randomness strings. The goal of a PCG is to compress long sources of correlated randomness without violating security. More concretely, the sender and receiver in a (two-party) PCG scheme hold pair of short correlated keys, and then they can locally expand these keys without interactions to obtain a pair of long correlated strings.
In recent works of Boyle et al. [BCGI18, BCG+19b], a concrete type of pseudorandom correlation generator i.e oblivious linear function evaluation (OLE) correlation [ADI+17, NP06, IPS09] is proposed for the goal of secure computation with silent preprocessing. This PCG is constructed based on variants of the Learning Parity with Noise assumption. The OLE functionality allows a receiver to learn a secret linear combination of two field elements held by a sender. A useful extension of OLE is vector OLE (VOLE), allowing the receiver to learn a linear combination of two vectors held by the sender.
The approach for generating a VOLE correlation is via reduction to random string OT. A first implementation of a primal VOLE generator was provided in [SGRR19], while concurrently, Boyle et al. [BCG+19b] provide an implementation of dual VOLE over binary fields. While [SGRR19] give a efficient technique for multi-point distributed point function (DPF) and construct VOLE without replying on hardness of LPN assumption, [BCG+19b] constructs not only VOLE based on LPN assumption but also OT correlation with two rounds.
The ideal functionality of subfield VOLE is presented in figure 2.3. The functionality of subfield VOLE is identical with VOLE except the sender learns a vector $\mathbf{u}$ in a base field. VOLE, is recently introduced in [BCG+19b, BCGI18, SGRR19, BCG+19a], enables MPC with silent preprocessing. VOLE can be constructed with sublinear communication complexity by LPN assumption and a puncturable pseudorandom function (PPRF).
The construction distributes a pair of seeds to parties allows to locally expand these seeds to the large instance of VOLE. PPRF enables the party to compute the value of PRF F at any point except one. Given t is the number of positions non-zero of an error sparse vector $e \in \mathbb{F}_p^N$, [BCG+19a] uses t times PPRF tree to obtain the secret sharing of (v_0, v_1) such that:

$$v_1 = v_0 + \Delta e \in \mathbb{F}_q^N \tag{2.1}$$

where $v_1 \in \mathbb{F}_q^N, \Delta \in \mathbb{F}_q$ is given to a party and $e, v_0 \in \mathbb{F}_q^N$ is held by other party.
puncturable pseudorandom function (PPRF) primitives is a PRF F such that given an input x, and a PRF key k, one can generate a punctured key $k\{x\}$ which allows evaluating $\mathbb{F}$ at every point except for x, and does not conceal any information about the value $\mathsf{F}(k, x)$. A PPRF can be built from any length-doubling pseudorandom generator, using a binary tree-based construction [BCG+19b].

PARAMETERS:

- 2 parties a sender and receiver.
- A finite field $\mathbb{F}_q$ where $q = p^r$, p is a power of prime,r integer.
- An integer n the size of output vector.

FUNCTIONALITY:

- The sender gets 2 random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$.
- The receiver receives $\Delta \in \mathbb{F}_q$ and a vector $\mathbf{w}$ such that $\mathbf{w} = \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$.

Figure 2.3: Ideal functionality $\mathcal{F}_{\mathsf{svole}}$

We will give the sender a random key k and x, and give to the receiver a random point $\alpha \in [N]$, a punctured key $k\{x\}$, and the value $z = \mathsf{F}(k, \alpha) + x$. Given these seeds, the sender and receiver can now define the expanded outputs, for $i \in [n]$:

$$v_0[i] = \mathbb{F}(k, i), \quad v_1[i] = \begin{cases} \mathsf{F}(k, i) & i \neq \alpha \\ z & \text{otherwise} \end{cases}$$

These immediately satisfy 2.1, with e as the α-th unit vector. To obtain sharing of sparse e with, say, t non-zero coordinates, as needed to use LPN, we repeat this t times and XOR together all t sets of outputs. This correlation 2.1 can be converted to randomness using multiplication each vector by a public matrix $H \in \mathbb{F}_p^{N \times n}$ where $N = O(n)$ [BCG$^+$19a]. When $p = 2$ to get n instances $\mathsf{subVOLE}$ the number of OTs needed is just $t \log N$ with the input string of OT are κ bits long. For $p > 2$, the implementation requires in addition a single subfield-reverse VOLE on vector of length t.
Note that in VOLE the sender inputs $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$, the receiver inputs $x \in \mathbb{F}_q$ and gets $\mathbf{w} = \mathbf{u}x + \mathbf{v} \in \mathbb{F}_q^n$. However, in the reverse VOLE functionality, the sender inputs $(\mathbf{v}, x) \in \mathbb{F}_q^n \times \mathbb{F}_q$ while the receiver sends $\mathbf{u} \in \mathbb{F}_p^n$ and receives $\mathbf{w} = \mathbf{u}x - \mathbf{v} \in \mathbb{F}_q^n$. The reverse VOLE can be implemented in several approaches. A direct approach is via homomorphism encryption schemes, which can be instantiated using Pallier encryption [EFG$^+$09] or (ring)-LWE [DPSZ12]. Another approach is by reduction from VOLE to string- OTs, first introduced by Gilboa [Gil99] and implemented by [KOS16]. The idea of Gilboa uses a bit-decomposition to reduce an OLE over the field with l-bit elements to l instances of OT. There is one efficient approach [ADI$^+$17] uses the same idea of reduction of Gilboa, based on arithmetic version of "LPN-style" assumptions and an arithmetic version of a local polynomial-stretch PRG.
The construction of a PCG for OT correlations based on a PCG for subfield VOLE is presented in figure 2.4. The PCG produces a set of n random 1-out-of-p OTs based on a correlation robust hash function (section 2.2.3 and the LPN assumption over $\mathbb{F}_p$. For each position $i \in [n]$, the sender receives a pair (u_i, v_i') where v_i' is the OT value corresponding to u_i and the receiver p OTs values. Protocol 2.4 is security under the security of subfield VOLE and correlation hash function. Our OPRF in section 4 is motivated from this PCG OTs correlation.

PARAMETERS:

- Security parameter 1^κ, integers n, r and prime power p with $p^r = O(2^\kappa)$.
- An $(n, \mathbb{F}_p, \mathbb{F}_{p^r}$ correlation-robust function $\mathsf{H} : \{0,1\}^\kappa \times \mathbb{F}_{p^r} \to \{0,1\}^\kappa$.
- A subfield VOLE model with parameters (n, p, r).

PROTOCOL:

1. The sender S and receiver R join the subfield VOLE, then :
   - The sender gets 2 random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v}' \in \mathbb{F}_q^n$.
   - The receiver receives $\Delta \in \mathbb{F}_q$ and a vector $\mathbf{w}'$ such that $\mathbf{w} = \Delta\mathbf{u} + \mathbf{v}' \in \mathbb{F}_q^n$.
2. S computes
   $$v_i \leftarrow \mathsf{H}(i, v_i') \quad \text{for} \quad i = 1, \ldots, n$$
   and outputs (u_i, v_i).
3. R obtains $\mathbf{x} \in \mathbb{F}_q, \mathbf{w}' \in \mathbb{F}_q^n$ compute:
   $$w_{i,j} \leftarrow \mathsf{H}(i, w_i' - j.x) \quad \text{for} \quad i = 1, \ldots, n, \forall j \in \mathbb{F}_p$$
   and outputs $\{w_{i,j}\}_{i,j}$

Figure 2.4: PCG for n sets 1-out-p random OT

Chapter 3

State of the art of PSI

Securely intersecting two sets without leaking any information is one of the most prominent problems in secure computation. Recent works have developed some PSI protocols with an efficient time in both communication and computation which can be truly used in practice. In this chapter, we revisit all the common approaches for the PSI problem. Several techniques have been applied to deal with the PSI problem as an efficient but insecure hashing method (both parties hash their input and compare their hash results). The earliest method used relies on public-key cryptosystems, such that Diffie-Hellmann or Blind RSA. This method has a small communication but it requires the parties have a large computation capacity. On the other side, some works try to adapt the garbled circuit method i.e a generic approach of secure computation to PSI problem, and then these works can bring a competitive result under a careful construction of garbled circuits. The most recent and efficient technique is using OT extension to obtain a secure obvious pseudorandom function OPRF, which can directly deal with PSI problem. We discussed in detail some fastest PSI protocols based on OTs.

3.1 A naive solution

The easiest approach to deal with PSI problem is applying a cryptographic hash function. Each party hashes their input and then compare the result of hashing. Although this protocol is very efficient but it is insecure if the input set is small and has low entropy. One party can run a brute force attack for all possible values of input set and then compare to the received hashes so the information of input set of other parties will be revealed.

3.2 Public-key- Based PSI

The major advantage of Public-key based protocols is their simplicity, which makes them comparably easy to implement. However the cost for implement public key is expensive and it requires a large computation capabilities so this method seems to be efficient when we just consider about communication.

3.2.1 DH based protocol

A PSI protocol relied on the Diffie-Hellmann (DH) key agreement scheme was firstly proposed in [HFH99]. This protocol (figure 3.1) is based on the commutative properties of the DH function and was used for private preference matching. This protocol is very simple to implementation, can be based on elliptic-curve crypto and has a low communication complexity. However, this protocol has to compute $4n$ exponentiation so with the computational security parameter $\kappa = 128$, it requires a high computing power.

PARAMETERS:

1. Alice has the set $\{x_1, \ldots, x_n\}$ and Bob has the set $\{y_1, \ldots, y_n\}$.
2. H is modeled as a random oracle.
3. p is a 1024 bits prime number.

PROTOCOL:

1. A and B choose randomly 2 number a and b.
2. A permutes set $\mathsf{H}(x_1)^a, \ldots, \mathsf{H}(x_n)^a \pmod p$ and sends it to B.
3. Similarly, B sends $\mathsf{H}(y_1)^b, \ldots, \mathsf{H}(y_n)^b \pmod p$ to A.
4. A computes and sends $\mathsf{H}(y_1)^{ba}, \ldots, \mathsf{H}(y_n)^{ba} \pmod p$ to B.
5. B also computes $\mathsf{H}(x_1)^{ab}, \ldots, \mathsf{H}(x_n)^{ab} \pmod p$ and send to A.
6. Each party can now count the matches.

Figure 3.1: DH protocol

3.2.2 Blind RSA-based PSI Protocol

Another PSI protocol that uses public-key cryptography (blind-RSA operations) to minimize the communication cost is represented in [DCT09]. This protocol is efficient for unequal input sets, in which one party with large computation power as server while other is client with small input set and limitation about computation. In this protocol, both sender and receiver can locally do some pre-online computations. The client does not execute any exponentiation throughout the online phase, but only $O(v)$ on-line modular multiplications in step 4 (Figure 3.2).

To check the correction of protocol, consider that:$K_{y_j} = (hy_j)^d$ in Step 1, and, in Step 6:

$$K_{x_i} = y_i'/R_{x_i} = (hx_i).(R_{x_i})^e)^d/R_{x_i} = (hx_i)^d$$

Then we obtain $K_y = K_x$ iff $y = x$.
This protocol's performance is comparable to that of DH-based protocols, however in the online phase, just the owner of the private key does all of the heavy work.

3.3 Circuit-Based PSI

The generic secure computing protocol enables the secure evaluation of arbitrary functions defined as Boolean or Arithmetic circuits. Intensive effort has been spent over the last several years to creating custom protocols for PSI based on homomorphic encryption cryptosystem and other public-key techniques. The majority believe that the methods based on generic approaches would be unfeasible. The paper [Yao86, HEK12b] shows that this belief could be not inappropriate. It creates three types of protocols aimed at different set sizes and domains, all of which are based on Yao's general garbled-circuit technique [Yak17]. The results demonstrate that using garbled circuits carefully leads to solutions that can operate on million-element sets on ordinary PCs and compete with the fastest customized protocols.

Yao's Garbled Circuits

The millionaire's problem proposed by Yao laid the foundation of secure computation. The idea behind using circuit is that express the evaluated function into Boolean or Arithmetic circuits and takes the advantage of cryptography primitives to secret sharing. The protocol consists of 6 steps as follows:

PARAMETERS:

1. Common input: n, e, d parameters of RSA, two hash functions H; H'.
2. $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_n^*$ is a full-domain hash function.
3. Alice's input: $\mathcal{C} = \{hx_1; \ldots; hx_v\}$, where: $hx_i = \mathsf{H}(x_i)$.
4. Bob's input: $\mathcal{S} = \{hy_1; \ldots; hy_w\}$, where: $hy_j = \mathsf{H}(y_j)$.
5. Alice as client and knows public key e while Bob holds secret key d.

PROTOCOL:
Offline:

1. Bob:

$$\text{For } j \in [w], \text{ compute: } K_{y_j} = (hy_j)^d \pmod{n} \text{ and } t_j = \mathsf{H}'(K_{y_j})$$

2. Alice:

$$\text{For } i \in [v], \text{ compute: } R_{x_i} \leftarrow \mathbb{Z}_n^* \text{ and } y_i = hx_i.(R_{x_i})^e \pmod{n}$$

Online:

1. Alice sends to Bob: $\{y_1, \ldots, y_v\}$
2. Bob:

$$\forall i, \text{ compute } y_i' = (y_i)^d \pmod{n}$$

3. Bob then send to Alice $\{y_1', \ldots, y_v'\}, \{t_1, \ldots, t_w\}$
4. Alice:

$$\forall i, \text{ compute } K_{x_i} = y_i'/R_{x_i} \text{ and } t_i' = \mathsf{H}'(K_{x_i})$$

5. Alice computes the intersection of $\{t_1', \ldots, t_v'\}$ and $\{t_1, \ldots, t_w\}$ and outputs the intersection of two input sets.

Figure 3.2: Blind RSA protocol

1. A Boolean circuit with two input gates is used to explain the evaluating function. Both parties are known the circuit. This step can be completed in advance by a third party.
2. The circuit is garbled (encrypted) by Alice. Alice is known as the garbler.
3. Along with Alice's encrypted input, Alice transmits the garbled circuit to Bob.
4. Bob must also garble his own input in order to compute the circuit. To that aim, he requires Alice's help. Because only the garbler understands how to encrypt. Finally, Bob may use oblivious transfer to encrypt his input. According to the preceding description, Bob is the receiver and Alice is the sender in this oblivious transfer.
5. Bob evaluates the circuit (decrypts it) and receives the encrypted outputs. Bob is referred to as the evaluator.
6. Alice and Bob interact in order to determine the output.

A circuit-based protocol

All of the circuits are constructed in [Yao86] using Yao's garbled-circuit approach, which takes any Boolean circuit and produces a secure protocol for calculating this circuit. The most involved protocols all use a Sort-Compare-Shuffle architecture. The complexity of computation for these protocols is $O(n \log n)$ with small constant factors. The primary concept is for each party to sort their set locally before merging their sorted sets into a single sorted list (privately). Then, obliviously, each adjacent pair of items is compared, with the value maintained if the elements in the pair are equivalent and a dummy value replaced otherwise. Finally, before revealing the complete list of matching/dummy items, the resultant list of matching/dummy elements is obliviously mixed. This shuffling phase is needed because the information about position of the matched items may leak information of other items in set.

3.4 OT based Protocol

OT is a secure computation primitive which can be realized extremely efficiently and is used as a building block in larger protocols. However, the cost for constructing OT is expensive so the idea of expanding a small number of OT to obtain large instances of OT has been developed [Bea96]. OT extension has been applied into many PSI protocol and brings significant improvement to private set intersection aspects. Almost the currently fastest PSI protocol ([PSZ14, KKRT16, RR17, KRTW19, PSWW18, PRTY19, PRTY20, CM20, RS21]) take advantage from OT extension. The main construct of these protocol is using OT extensions to form a efficiently security OPRF then apply some masking techniques for the input set to get a security protocol. In this section, we discuss about some typical PSI protocols based on OT extension, which have best performance and obtains a balance between computation and communication.

3.4.1 PSI from single point OPRF

Single point OPRF

OT extension became extremely practical by the work of Y. Ishai et al. [IKNP03] to reduce OT_m^l to OT_m^κ. [KKRT16] deployed and developed the idea of Y.Ishai et al. to obtain a single point OPRF. We firstly present the functionality of single point OPRF in figure 4.1.

PARAMETERS:

- There are two parties involved: a sender S and a receiver R.
- The receiver has a input set $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in \mathbb{F}_q$ for $i \in [n]$.

FUNCTIONALITY:

- Choose a random seed for functionality $F : k*, k_1, k_2, \ldots, k_n$ to the sender.
- Give the set $x' = \{F((k, k_1)x_1), F((k, k_2), x_2), \ldots, F((k, k_n), x_n)\}$ to the receiver.

Figure 3.3: Ideal Functionality $\mathcal{F}_{\mathsf{oprf}}$ of batched Obvious PRF

The construction of single point OPRF [KKRT16] is evaluated as below.
Let $\mathcal{C}(\kappa, \epsilon)$ be a pseudorandom code that produces a pseudorandom string such that with probability at most $2^{-\epsilon}$ the hamming distance between two codewords is less than or equal to d. and H be κ hamming correlation robust function. 3.4 is the detail protocol of [KKRT16]. This type OPRF is called batch related key since we can compute m OPRF values of m input by executing this protocol one time and each input is related to corresponding key.
The R forms 2 $m \times k$ binary matrices T_0, T_1 such that the XOR of each $i-$th column of 2 matrices is equal to $\mathcal{C}(x_i)$. S and R join the OT_m^k with the input of sender is vector $s \in \{0,1\}^k$ while the R

INPUT: R has input set $x_1, x_2, \ldots, x_m$ where $x_i \in \{0,1\}^*$.
PARAMETERS:

- A $\mathcal{C}(\kappa, \epsilon)$ pseudorandom code with the output length $k > \kappa$.
- A κ-hamming correlation robust function H.
- An ideal OT_m^k primitive.

PROTOCOL:

1. S chooses random a string $s \leftarrow \{0,1\}^k$.
2. R forms $m \times k$ matrices T_0, T_1 in the following way:
   - T_0 is random and $t_{1,j} = \mathcal{C}(x_j) \oplus t_{0,j}$ for $j \in [m]$.

   Let t_0^i, t_1^i the $i-$th column of matrices T_0, T_1.
3. S and R interact with OT_m^k in the following way:
   - S is consider as receiver and inputs the vector $s_{i\,i\in[k]}$.
   - R is sender with input k pair of m bit strings $t_0^i, t_{1\,i\in[k]}^i$.
   - S receivers output ${q^i}_{i\in[k]}$.

   S obtains $m \times k$ matrix Q by taking the vector q^i as $i-$th column of Q. Let q_j denote the $j-$th row of Q.
4. For all $j \in [m]$ S takes (q_j, s) as the key k_j and OPRF at a point x is established by:
$$\mathsf{OPRF}_{k_j}(x) = \mathsf{H}(j, q_j \oplus [\mathcal{C}(x).s])$$
5. For all $j \in [m]$ R outputs $\mathsf{H}(j, t_{0,j})$.

Figure 3.4: OPRF protocol from [KKRT16]

inputs k pair of columns of 2 matrices T_0, T_1. Then, S gets a $m \times k$ matrix Q such that:

$$q_j = (t_{0,j} \oplus t_{1,j}).s) \oplus t_{0,j} = t_{0,j} \oplus [\mathcal{C}(x_j).s]$$

for all $j \in [m]$. Each row of matrix Q is considered as a key to construct a OPRF value and we always have:

$$t_{0,j} = q_j \oplus [\mathcal{C}(x_j).s]$$

The most heavy cryptographic techniques of this single point OPRF is OT, which is implemented by public key cryptosystems but this protocol can form m-RK-PRF using only $k \approx 3.5\kappa$ OTs with length of input for each OT m bits where $m >> k$. k OTs can be efficiently instantiated by using OT extension. The number of OTs needed is not proportional to the input size so it is very efficient when we requires a large number of OPRF. However, [KKRT16] is secure when the pseudorandom code $\mathcal{C}$ has enough Hamming weight.

From RK-OPRF to PSI

Ideal functionality of batch single point OPRF shows that when R inputs a set of n elements then each input of R will be assigned with a OPRF values corresponding to a specific key and the S holds those n related key. Therefore, before using single point OPRF to PSI protocol, we need to know the corresponding key of each input and make sure each input will be mapped to at most one key. This problem is solved by approach using linear decoding techniques to map each input with exactly one key. One of the most common decoding techniques used is hashing techniques,

specific Cuckoo hashing since both the communication and computational cost of linear decoding by Cuckoo hashing is linear $O(n)$ with a small factor. The first usage of Cuckoo hashing is presented in [PSSZ15].
The construction PSI protocol from batch RK-OPRF and hashing techniques is mainly described as figure 3.5. In step 3, we can see that each element of input is evaluated on multiple PRF, using the specified the number of hash functions using in Cuckoo hashing plus the size of stash s. This work leads to increase the overhead of both communication and computation. We can get rid of this obstacle by building a multiple point OPRF.

PARAMETERS:

- Alice and Bob have respectively input set $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$

PROTOCOL:

1. Alice uses Cuckoo hashing to map the input set X into b bins such that each element in X is assigned with at most one bin. Then Alice obtains a set X_b.
2. Alice and Bob join the single point OPRF, Alice as receiver with input set is X_b while Bob is sender. Alice will get the set X' of b PRF values and Bob receives b corresponding PRF key.
3. Now for each bin,using simple hashing, Bob computes PRF value on all possible elements in that bin then permutes and send this whole set to Alice.
4. Alice compares and outputs the intersection set.

Figure 3.5: PSI protocol from batch RK-OPRF

3.4.2 PSI from multi-point OPRF

Multi-point OPRF

Multi-point OPRF allows the sender Alice to learns a pseudorandom function (PRF) F and the receiver Bob learns the $F(y_i)$ for each element in his set $\{y_1, y_2, \ldots, y_n\}$. Alice will compute the PRF value on each element in her input set $\{x_1, x_2, \ldots, x_n\}$ and send them to Bob. For an appropriate choice of parameters for $\mathbb{F}$ with high probability we have $F(x_i) = F(y_j)$ if and only if $x_i = y_j$ then Bob can output the intersection of two input sets. Since F is pseudorandom function, for $x_i \in \{x_1, x_2, \ldots, x_n\}$ the corresponding $F(x_i)$ is indistinguishable to Bob. As a result, Bob receives no information about items which are not in the intersection set.
As compare to the single point OPRF, the multi-point PRF does not requires to assign each element to corresponding key. With such a multi-point OPRF it is trivial to achieve PSI. There are some efficient PSI protocols ([PRTY19, PRTY20, CM20, RS21] which construct multi-point OPRF from OT extension and obtain a balance between communication and computation. We will discuss some of them to analyse their detail techniques as well their strength and weakness.

[PRTY19] protocol

is the work of Pinkas et al. to construct a PSI protocol from multi-party OPRF. The idea of [PRTY19] is based on the overall construction of the IKNP protocol [IKNP03]. [PRTY19] proposes a technique called sparse OT extension. The sender has a N random secret values (N can be exponential), the receiver can enable to pick obviously a subset k values out of N value without the disclosing of subset k to sender, with communication costs just proportional to k. The main protocol of [PRTY19] is present in figure 3.6. We can see for any $x \in [N]$:

$$Q(x) = T(x) \oplus s.(T(x) \oplus U(x)) = T(x) \oplus s.R(x)$$

PARAMETERS:

- The sender Alice and the receiver Bob have 2 input sets $X = \{x_1, \ldots, x_{n_1}\}$, $Y = \{y_1, \ldots, y_{n_2}\} \subseteq [N]$ respectively.
- The size $l := \log \mathbb{F}$ is chosen such that protocol is security under interpolation polynomial over $\mathbb{F}$.
- A κ Hamming correlation robust function $\mathsf{H} : \{0,1\}^l \to \{0,1\}^{\lambda + \log(n_1 n_2)}$.
- A PRF F: $\{0,1\}^\kappa \times [N] \to \{0,1\}$.

PROTOCOL:

1. S picks random a string $s \leftarrow \{0,1\}^l$.
2. S and R interact with l instances of random OT with input length κ bits in this way:
   - S is consider as receiver and inputs the vector $s_{ii \in [l]}$.
   - R is sender with input k pair of strings $t_i, u_i \in \{0,1\}^\kappa$.
   - S receivers output q_i.
3. For $y \in Y$, Bob computes $R(y) = T(y) \oplus U(y)$, where:
$$T(y) := F(t_1, y) \parallel F(t_2, y) \parallel \ldots \parallel F(t_l, y)$$
$$U(y) := F(u_1, y) \parallel F(u_2, y) \parallel \ldots \parallel F(u_l, y)$$
4. Bob evaluates a polynomial $P(x)$ is interpolation of $\{y, R(y)\}_{y \in Y}$, and sends its coefficients to Alice.
5. Alice defines Q as follows:
$$Q(x) := F(q_1, x) \parallel F(q_2, x) \parallel \ldots \parallel F(q_l, x)$$
and sends $\mathcal{O} = \{\mathsf{H}(Q(x) \oplus s.P(x)) \mid x \in X\}$ randomly permuted to Bob.
6. Bob outputs $\{y \in Y \mid \mathsf{H}(T(y))) \in \mathcal{O}$

Figure 3.6: [PRTY19] PSI protocol

and in the step 5, Alice compute:

$$Q(x) \oplus s.P(x) = T(x) \oplus s.(P(x) \oplus R(x))$$

Let consider $x \in X \cap Y$ then $P(x) = R(x)$. Hence, $Q(x) = T(x)$ for all $x \in X \cap Y$.
Ignoring the communication cost in primitive l instance of random OT,the communication of [PRTY19] includes sending an interpolation polynomial of n distinct points and a set of n OPRF value on each element in the input set of sender. Hence, this type of protocol gains a better communication compared to [KKRT16] protocol. Looking at the parameter, we note that both [KKRT16] protocol and [PRTY19] protocol use a parameter l which is the width of the OT extension matrix, but in [KKRT16] this parameter is usually a little bigger.
This protocol relies mainly on computing high-degree polynomials over large finite fields. [PRTY19] protocol achieves a multi-point OPRF by interpolation and evaluation over large field so this protocol costs a asymptotically $O(n \log^2 n)$ computation. To reduce this dominant computation cost, Pinkas et al. proposes in [PRTY19] 2 version called spot-low (as in figure 3.6)and spot-fast. Spot-low obtain a optimal communication cost version presented in figure 3.6 but the running time is slow because it requires to interpolate large polynomial, in which Bob sends the coefficient of a

large polynomial and Alice sends the set of OPRF value on each item in her input set otherwise spot-fast obtain a better running time but its communication cost is larger than spot-low in which Bob uses 2-choice hashing and Alice sends two OPRF values for each element in her input set.

[CM20] protocol

In the single point construction OPRF (figure 3.4), the sender will receive a random $s \in \{0,1\}^\kappa$ and different PRF keys k_j. However, regardless of which s is picked, $\mathsf{OPRF}_{k_j}(x) = \mathsf{H}(j, t_{0,j})$. Melissa Chase et al. [CM20] extends this idea to obtain a multi-point OPRF. The protocol of [CM20] is presented in figure 3.7.

PARAMETERS:

- The sender Alice and the receiver Bob have 2 input sets $X = \{x_1, \ldots, x_{n_1}\}$, $Y = \{y_1, \ldots, y_{n_2}\}$ respectively.
- Two hash functions $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{2\kappa}$ collision resistant hash function, $\mathsf{H}_2 : \{0,1\}^w \to \{0,1\}^{\lambda + \log n_1 n_2}$ a κ-Hamming correlation robust.
- PRF $F : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to [m]^w$.

PROTOCOL:

1. Precomputation:
   (a) Alice samples a random string $s \in \{0,1\}^w$.
   (b) Bob does the following:
   - Initialize am $m \times w$ binary matrix D to all $1's$. Denote its columns vectors by $D_1, D_2, \ldots, D_w$. Then $D_1 = D_2 = \ldots = D_w = 1^m$.
   - Sample a uniformly random PRF key $k \in \{0,1\}^\kappa$.
   - For each $y \in Y$, compute $v = F_k(\mathsf{H}_1(y))$. Set $D_i[v[i]] = 0$ for all $i \in [m]$.
2. Obvious Transfer:
   (a) Bob randomly samples an $m \times w$ binary matrix $A \in \{0,1\}^{m \times w}$. Compute matrix $B = A \oplus D$.
   (b) Alice and Bob join OT_m^w in the following way:
   - Bob is the sender with input $\{A_i, B_i\}_{i \in [w]}$.
   - Alice is the receiver with input $s \in \{0,1\}^w$.
   - Alice obtains w number of $m-$ bit strings as the column vectors of matrix C.
3. OPRF Evaluation
   (a) Bob sends the PRF key k to Alice.
   (b) For each $x \in X$, Alice computes $v = F_k(\mathsf{H}_1(x))$ and its OPRF value $\phi = \mathsf{H}_2(C_1[v[1]] \parallel \ldots \parallel C_w[v[w]]$ and sends ϕ to Bob. Let $\mathcal{O}$ be the set of these OPRF value.
   (c) For each $y \in Y$, Bob computes $v = F_k(\mathsf{H}_1(y))$ and its OPRF value $\phi = \mathsf{H}_2(A_1[v[1]] \parallel \ldots \parallel A_w[v[w]]$ and outputs y if $\phi \in \mathcal{O}$.

Figure 3.7: [CM20] PSI protocol

The new PRF key in [CM20] includes a matrix $C \in \{0,1\}^{m \times w}$. To compute the PRF on input x, [CM20] compute $F_k(\mathsf{H}_1(x)$ which produces a vector $v \in [m]^w$ where F pseudorandom function

and H_1 a hash function. Hence, the OPRF value is evaluated as follows:

$$\mathsf{OPRF}_C(x) = \mathsf{H}_2(C_1[v[1]] \parallel \ldots \parallel C_w[v[w]]$$

H_2 κ correlation robust hash function and for all $i \in [w]$ C_i is the i'th column of matrix C.
As in figure 3.7, we can see that for $x \in X \cap Y$, let $v = F_k(\mathsf{H}_1(y))$ then

$$A_1[v[1]] \parallel \ldots \parallel A_w[v[w] = B_1[v[1]] \parallel \ldots \parallel B_w[v[w]$$

since $B = A \oplus D$ and $D_i[v[i]] = 0$ for all $i \in [m]$.
Then, for all possible value of $s \in \{0,1\}^w$, we deduce that $x \in X \cap Y$ if and only if:

$$\mathsf{OPRF}_C(x) = \mathsf{OPRF}_A(x)$$

The parameter m and w is chosen such that if F is a random function and $\mathsf{H}_1(x)$ is different for each $x \in X \cup Y$. They take $m = n$ and the number of OTs base is $w \approx 4.7\kappa$.
Compare to [KKRT16] [PRTY19], [CM20], it's PSI protocol is faster than all the other. [CM20] has a larger number of base OTs than [KKRT16] but it's PSI protocol do not requires any hashing techniques to map each item to bin then the communication cost consists of only one OPRF value of each element in input set. [PRTY19] relies on the interpolation polynomial over large field $\mathbb{F}$ which makes this protocol lower computation than [CM20].

[RS21] protocol

presents the construction for a batched Oblivious Pseudorandom Function (OPRF) relied on Vector-OLE and the PaXoS linear solver, then use it for achieving PSI protocol from an OPRF.
Firstly, PaXoS is a approach for linear system solvers. The main of PaXoS is to encode the input sets $\{z_1, \ldots, z_n\} = Z$ and values $\{v_1, \ldots, v_n\} = V$ as a vector $\mathbf{P} \in \mathbb{F}^m$. There will exist a function Decode such that $\text{Decode}(\mathbf{P}, z_i) = v_i$ for $i \in [n]$ and is linear with respect to $\mathbf{P}$. PaXoS method is firstly presented by Pinkas et al. [PRTY20], it obtains $O(n)$ running times but it costs $m = 2.4n$ to output a solution with probability of success $1 - O(2^{-\lambda})$.
Second technique is used is VOLE, is constructed by OTs with sublinear communication cost. Given a large finite field $\mathbb{F}_q$, to guaranteeing security the size of $\mathbb{F}_q$ will be $\approx 2^\kappa$. The functionality of their VOLE give parties random vectors $\mathbf{A}', \mathbf{B}, \mathbf{C} \in \mathbb{F}_q^m$ and an element $\Delta \in \mathbb{F}_q$ such that $\mathbf{C} := \Delta\mathbf{A}' + \mathbf{B}$. The PSI receiver will hold $(\mathbf{A}', \mathbf{C})$, while the sender will hold $(\mathbf{B}, \Delta)$.
The receiver has input set $Y = \{y_1, y_2, \ldots, y_n\}$ defines a matrix $M \in \{0,1\}^{n \times m}$ such that:

$$M = \begin{bmatrix} \mathsf{row}(y_1, r) \\ \mathsf{row}(y_2, r) \\ \cdots \\ \mathsf{row}(y_n, r) \end{bmatrix}$$

where $\mathsf{row} : \mathbb{F}_q \times \{0,1\}^\lambda \to \{0,1\}^m$ is random function and $r \in \{0,1\}^\lambda$ is random seed.
The receiver use (Xo)PaXoS to solve the linear equation:

$$\mathbf{MP} = (\mathsf{H}(y_0), \mathsf{H}(y_1), \ldots, \mathsf{H}(y_n))^T$$

for the unknown $\mathbf{P} \in \mathbb{F}_q^m$ and H random oracle. The protocol continues by having receiver sends $\mathbf{A} := \mathbf{A}' + \mathbf{P} \in \mathbb{F}_q^m$ to the sender and then sender defines: $\mathbf{K} = \mathbf{B} + \Delta\mathbf{A} \in \mathbb{F}_q^m$.
The sender computes their the PRF function as:

$$X' = \{\mathsf{H}(\text{Decode}(\mathbf{K}, x, r) - \Delta\mathsf{H}^{\mathbb{F}}(x) + w, x) \mid x \in X\}$$

The receiver outputs the value:

$$Y' := \{\mathsf{H}(\text{Decode}(\mathbf{C}, y) + w, y) \mid y \in Y\}$$

We consider the correctness of [RS21] protocol as figure 3.8. For $x \in X \cap Y$, we see that:

$$\begin{aligned}
\text{Decode}(\mathbf{K}, x, r) - \Delta\mathsf{H}^{\mathbb{F}}(x) &= \text{Decode}(\mathbf{B} + \mathbf{P}\Delta + \mathbf{A}'\Delta, x) - \Delta\mathsf{H}^{\mathbb{F}}(x) \\
&= \text{Decode}(\mathbf{B} + \mathbf{A}'\Delta + \mathbf{P}\Delta, x) - \Delta\mathsf{H}^{\mathbb{F}}(x) \\
&= \text{Decode}(\mathbf{C} + \mathbf{P}\Delta, x) - \Delta\mathsf{H}^{\mathbb{F}}(x) \\
&= \text{Decode}(\mathbf{C}, x) + \Delta\text{Decode}(\mathbf{P}, x) - \Delta\mathsf{H}^{\mathbb{F}}(x) \\
&= \text{Decode}(\mathbf{C}, x) + \Delta\mathsf{H}^{\mathbb{F}}(x) - \Delta\mathsf{H}^{\mathbb{F}}(x) \\
&= \text{Decode}(\mathbf{C}, x).
\end{aligned}$$

Finally, we obtain an OPRF by breaking up the linear correlation using the hash function H.

PARAMETERS:

- The sender Alice and the receiver Bob have 2 input sets $X = \{x_1, \ldots, x_{n_1}\}$, $Y = \{y_1, \ldots, y_{n_2}\}$ respectively, $x_i, y_j \in \mathbb{F}$ for $i \in [n_1], j \in [n_2]$.
- A hash function H

PROTOCOL: Upon input (sender; sid; X) from the Sender and (receiver; sid; Y) from the Receiver, the protocol specifies the following:

1. Alice samples randomly $w^s \leftarrow \mathbb{F}$ and sends $c^s := \mathsf{H}^{\mathbb{F}}(w^s)$ to the receiver.
2. Bob samples $r \leftarrow \{0,1\}^\kappa, w^r \leftarrow \mathbb{F}$ and solves the systems:

$$\begin{bmatrix} \mathsf{row}(y_1, r) \\ \mathsf{row}(y_2, r) \\ \cdots \\ \mathsf{row}(y_n, r) \end{bmatrix} \mathbf{P} = (\mathsf{H}^{\mathbb{F}}(y_1), \ldots, H^{\mathbb{F}}(y_n)$$

   for vector $\mathbf{P}$ as a function of their set $Y \in \mathbb{F}$.
3. The Sender sends (sender; sid) and the Receiver sends (receiver; sid) to $\mathcal{F}_{\mathsf{vole}}$ with dimension m and $|\mathbb{F}| \approx 2^\kappa$. The parties respectively receive $\Delta, \mathbf{B}$ and $\mathbf{C} := \mathbf{A}'\Delta + \mathbf{B}, \mathbf{A}'$.
4. The Receiver sends $r; w^r; \mathbf{A} := \mathbf{A}' + \mathbf{P}$ to the Sender who defines $\mathbf{K} := \mathbf{B} + \mathbf{A}\Delta$..
5. The Sender sends w^s to the Receiver who aborts if $c^s \neq \mathsf{H}^{\mathbb{F}}(w^s)$. Both parties define $w := w^r + w^s$.
6. The Receiver outputs the set of OPRF values $Y' := \{\mathsf{H}(\text{Decode}(\mathbf{C}, y) + w, y) \mid y \in Y\}$.
7. The Sender computes the set $X' = \{\mathsf{H}(\text{Decode}(\mathbf{K}, x, r) - \Delta\mathsf{H}^{\mathbb{F}}(x) + w, x) \mid x \in X\}$ and sends X' to receiver.
8. Receiver compares X', Y' and outputs the intersection of their set.

Figure 3.8: [RS21] PSI protocol

Chapter 4

Our contribution

4.1 Our protocol

4.1.1 Our contributions

A batch-related key OPRF.

We construct a batch OPRF based on subVOLE. The communication cost of VOLE is negligible when n is sufficiently large and we only need to send around $(\lambda + \log n_1)$ bits per element so our batch-related key OPRF achieves a small communication per OPRF evaluation on random input. Recently, the approach of Rindall et al. [RS21] combines two cryptographic primitives, VOLE and a linear solver e.g PaXoS, into a high efficient OPRF and PSI protocol. Using PaXoS gains better computation but it induces a higher communication cost. Given a large finite field $\mathbb{F}_q$, to guaranteeing security the size of $\mathbb{F}_q$ will be $O(2^{\kappa+\log n})$. The functionality of their VOLE give parties random vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{F}_q^m$ and an element $\Delta \in \mathbb{F}_q$ such that $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v}$. The PSI receiver will hold $(\mathbf{u}, \mathbf{w})$, while the sender will hold $(\mathbf{v}, \Delta)$.
The receiver has input set $X = \{x_1, x_2, \ldots, x_n\}$ defines a matrix $M \in \{0,1\}^{n\times m}$ such that:

$$M = \begin{bmatrix} \mathsf{row}(x_1, r) \\ \mathsf{row}(x_2, r) \\ \cdots \\ \mathsf{row}(x_n, r) \end{bmatrix}$$

where $\mathsf{row} : \mathbb{F}_q \times \{0,1\}^\lambda \to \{0,1\}^m$ is random function and $r \in \{0,1\}^\lambda$ is random seed.
The receiver use (Xo)PaXoS to solve the linear equation:

$$\mathbf{Mp} = (\mathsf{H}(x_0), \mathsf{H}(x_1), \ldots, \mathsf{H}(x_n))^T$$

for the unknown $\mathbf{p} \in \mathbb{F}_q^m$ and H random oracle. The protocol continues by having receiver sends $\mathbf{u}+\mathbf{p} \in \mathbb{F}_q^m$ to the sender. For the PaXoS linear solver the value of $m \approx 2.4n$, so the communication cost of this protocol is around $2.4\kappa n$ and plus the overhead cost to implement a VOLE with dimension $2.4n$. Compare with this, we obtain a batched OPRF on n input in any sufficiently large field by installing only a subVOLE hybrid model with dim n and sending a random vector $\mathbf{t} \in \mathbb{F}_p^n \subset \mathbb{F}_q^n$.
Comparing to the batch OPRF [KKRT16, PRTY19] constructing from OT extension, our OPRF is identical with this of [KKRT16] when obtaining a large instance OPRF values with related key for each input. However, [KKRT16] uses the assumption of d-Hamming hashing robustness which leads to the number of OT bases is from 3κ to 4κ even the size of input is small. While [PRTY19] protocol,is an optimization of [KKRT16], uses less OT bases but it relies heavily on manipulating high-degree polynomials over large finite fields. Our construction uses only $t \log N$ OT bases to implement subVOLE then it gains a significant cost compare with [KKRT16] and its variant.

A new PSI.

We present a new semi-honest PSI protocol from our OPRF which we believe achieves better communication and computation as compared with other efficient PSI protocols based on OTs. Our PSI relying on subfield VOLE and hashing techniques are most closely to [KKRT16, PSSZ15] using Cuckoo hashing to assign each element of set into the bins and compute OPRF for each bin. However, our protocol just requires sending $\lambda + \log n_1$ bits per hash element then we use 3 random functions in Cuckoo hashing without stash and trade-off by increasing the number of bins. We also propose a PSI version with slightly revealing information of input but it can be efficient in practice.

4.1.2 Our Fast OPRF

Our OPRF is proven secure under $(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-*Correlation robustness* on the hash function.

Our variant OPRF

Our OPRF is a variant of OT extension which associate a input $x \in \mathbb{F}_p$ and a random key k with a pseudorandom value $F(k, x)$. We present the ideal functionality of $\mathcal{F}_{\mathsf{oprf}}$ in figure 4.1. Our batched OPRF allows the receiver gets the PRF value of each input x related to a different random key while the sender holds these key can compute PRF value on any input.

PARAMETERS:

- There are 2 parties the sender and a receiver.
- The receiver has a input set $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in \mathbb{F}_q$ for $i \in [n]$.

FUNCTIONALITY:

- Choose a random seed for functionality $F : k, k_1, k_2, \ldots, k_n$ to the sender.
- Give the set $x' = \{F((k, k_1)x_1), F((k, k_2), x_2), \ldots, F((k, k_n), x_n)\}$ to the receiver.

Figure 4.1: Ideal Functionality $\mathcal{F}_{\mathsf{oprf}}$ of batched Obvious PRF

We now present our OPRF construction based on $\mathcal{F}_{\mathsf{svole}}$ model. Our construction is detail in figure 4.2 and securely realizes the functionality $\mathcal{F}_{\mathsf{oprf}}$ in figure 4.2. We use a $(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-correlation robust: $\mathsf{H} : \mathbb{F}_p \times \mathbb{F}_q \to \{0,1\}^v$ to construct the pseudorandom output of OPRF. We observe that if two parties join to subVOLE with inverse role then receiver will get two random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$, other holds $\Delta \in \mathbb{F}_q, \mathbf{w} = \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$. Therefore, for all $i \in [n]$:

$$v_i = w_i - \Delta u_i \in \mathbb{F}_q$$

We assign a element $x \in \mathbb{F}_p$ with a position $j \in [n]$ then:

$$\mathsf{H}(x, v_j) = \mathsf{H}(x, w_j - \Delta u_j)$$

Consider the pair (Δ, w_j) which is known by sender is the associated key k_j of x, the $\mathsf{OPRF}(k_j, x) = \mathsf{H}(x, v_j)$. Another property is that the sender holds the key can compute the PRF on any input and receive the same value for input x. To deal with this, receiver sends to other party the value $t_j = u_j - x$. The vector $\mathbf{u}_j \in \mathbb{F}_p$ is random which lead to the randomness of t_j. Therefore, the vector $\mathbf{t}$ will not reveal any information about $\mathbf{u}$ and the receiver's set to input.
Subsequently, the sender can compute PRF value on any input related to the key $k_j = (\Delta, w_j)$ by

$$\mathsf{PRF}(k_j, y) = \mathsf{H}(y, w_j - (y + t_j)\Delta)$$

By the assumption of hash function H, the security is a guarantee.

PARAMETERS:

- $(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-Correlation robustness: $\mathsf{H} : \mathbb{F}_p \times \mathbb{F}_q \to \{0,1\}^v$.
- The sender and receiver inputs a set $X = \{x_1, x_2, \ldots, x_n\}$ considered as a vector $\mathbf{x}$ in $\mathbb{F}_q^n$.

PROTOCOL:

1. The receiver sends (receiver, sid) and sender sends (sender, sid) to $\mathcal{F}_{\mathsf{svole}}$ dimension n with inverse role. The receiver gets two random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$ and the sender receives $\Delta \in \mathbb{F}_q$, $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$.
2. The receiver sends a vector $\mathbf{t} := \mathbf{u} - \mathbf{x} \in \mathbb{F}_p^n$ to sender.
3. For $j \in [n]$ the receiver outputs:
$$\mathsf{OPRF}((\Delta, w_j), x_j) = \mathsf{H}(x_j, v_j)$$
.
4. For $j \in [n]$ the sender output $\Delta, (w_j, j, t_j)$. Subsequently, for each input $y \in \mathbb{F}_q$, sender obtains:
$$\mathsf{PRF}(\Delta, w_j), y) = \mathsf{H}(y, w_j - (t_j + y)\Delta)$$

Figure 4.2: Our batched OPRF Π_{oprf} based on $\mathsf{subVOLE}$

Theorem 1 *The protocol Π_{oprf} securely realizes the ideal functionality $\mathcal{F}_{\mathsf{oprf}}$ against a semi-honest adversary in the correlation robustness hash function over field, $\mathcal{F}_{\mathsf{svole}}$ hybrid model when the parameter is chosen as described in 4.1.3.*

Proof.
Correctness:
By the correction of the protocol $\mathcal{F}_{\mathsf{vole}}$, we obtain:

$$w_i = \Delta u_i + v_i \quad \text{for} \quad i \in [n]$$

Therefore,

$$\begin{aligned}\mathsf{OPRF}((\Delta, i, w_i), x_i) &= \mathsf{H}(x_i, v_i) = \mathsf{H}(x_i, w_i - \Delta u_i) \\ &= \mathsf{H}(x_i, \Delta(u_i - x_i + x_i) = \mathsf{H}(x_i, \Delta(t_i + x_i))\end{aligned}$$

The receiver and sender compute the equal value of OPRF for the same x_i related to the one random key.
Security:
When using the ideal functionality of OPRF and the properties of correlation robustness hashing function over field, the security is proved by simulation model.
Corrupted Sender.

- The simulator S interacts with the sender as follows:
- S plays the role of $\mathcal{F}_{\mathsf{svole}}$ to get $\Delta \in \mathbb{F}_q$,$w \in \mathbb{F}_q^n$ and the outputs Δ, w_j, t_j for $j \in [n]$
- On behalf of the receiver, S sends a random vector $\mathbf{t} = (t_1, t_2, \ldots, t_n) \in \mathbb{F}_p^n$ to the sender. The simulation is perfect.

Corrupted Receiver.

- The simulator R has input $(x_1, x_2, \ldots, x_n)$ and receives two random vector $\mathbf{u} \in \mathbb{F}_p^n$ and $\mathbf{v} \in \mathbb{F}_q^n$.

- Protocol is abort if there exist $i \in [n]$ such that $u_i = 0$. The size of $\mathbb{F}_p$ is chosen such that if there exists a $i \in [n]$ such that $u_i = 0$ then the aborting probability is negligible $O(2^{-\lambda})$.
- R takes a uniformly random $\Delta \in \mathbb{F}_q$ then compute: $\mathbf{w} = \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q$ then as sender R outputs $\Delta, (w_j, j, t_j)$ for $j \in [n]$.

4.1.3 PSI protocol

Alice and Bob have two input sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. ur protocol PSI figure 4.3 from OPRF is identical with protocol in [PSSZ15] and based on the optimization of [KKRT16]. However, our protocol just requires sending $\lambda + \log n_1$ bits per hash element then we use 3 random functions in Cuckoo hashing without stash and trade-off by increasing the number of bins. Cuckoo hashing maps each element of the input set X to the empty bin $\mathcal{B}$ by random functions. Each element in X is assigned to exactly one bin and each bin has to contain one element. Therefore, we add the dummy element to the bin which does not consist of any element of set X.

PARAMETERS:

- An arbitrary subfield $\mathbb{F}_p \subseteq \mathbb{F}_q$.
- Alice and Bob have respectively input set $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ considered as vector dimension n in $\mathbb{F}_p$.
- 3 random functions $h_1, h_2, h_3 : \{0,1\}^* \to [1.27n]$.

PROTOCOL:

1. Alice uses Cuckoo hashing with 3 given random function to map elements in set X to the 1.27 bins.
2. Alice joins the protocol Π_{oprf} as a receiver with input set $X_{\mathcal{B}} = \{r_1, r_2, \ldots, r_{1.27n}\}$ defined as follows:
   for $j \in [1.27n]$, if bin $\#j$ is empty, then set r_j to a dummy value; otherwise if x is in bin $\#j$ then set $r_j = x \parallel i$ where $j = h_i(x), i \in \{1,2,3\}$.
3. Alice obtains $1.27n$ instances OPRF:
$$X' = \left\{\mathsf{OPRF}(k_j, r_j) \mid r_j \in X_{\mathcal{B}}\right\}$$
4. Bob computes the set of $3n$ PRF output:
$$H_i = \left\{\mathsf{PRF}(k_j, y \parallel i) \mid y \in Y, j = h_i(y)\right\} \quad \text{for} \quad i \in \{1,2,3\}$$
   Then Bob randomly permutes each set and sends them to Alice.
5. Alice finds the intersection:
   if $x \in X$ is mapped to bin j by function h_i then check whether $\mathsf{PRF}(k_j, x \parallel i) \in H_i$.
   Alice outputs the intersection and sends them to Bob.

Figure 4.3: PSI protocol

Specifically, in our protocol, Alice uses Cuckoo hashing [PR04] with 3 random functions: $h_1, h_2, h_3 : X \to [m]$, an empty bin $\mathcal{B}[1, \ldots, m]$. By a high probability all element in set X is mapped to exactly one bin $\mathcal{B}[h_i(x)]$ for $i \in \{1,2,3\}$. We add dummy value to bin which is not assigned by any element. If the size of set X is n then $m = 1.27n$ and no stash needed [PSZ18].
After mapping, Alice expanded her set X to the ordered set contains $1.27n$ elements. Alice joins the protocol Π_{oprf} as a receiver with input set $X_{\mathcal{B}} = \{r_1, r_2, \ldots, r_{1.27n}\}$.

Alice obtains $1.27n$ instances OPRF:

$$X' = \{\mathsf{OPRF}(k_j, r_j) \mid r_j \in X_{\mathcal{B}}\}$$

Bob as a sender receives the set of key $(k, k_1, \ldots, k_{1.27n})$ then can compute OPRF in any input. He uses simple hashing with same 3 random functions h_1, h_2, h_3 to map each element y in his set to 3 bins $h_1(y), h_2(y), h_3(y)$. Therefore, Bob computes the set of $3.n$ PRF output:

$$H_i = \{\mathsf{PRF}(k_j, y \parallel i) \mid y \in Y, j = h_i(y)\} \quad \text{for} \quad i \in \{1, 2, 3\}$$

Then Bob randomly permutes each set and sends them to Alice. Alice finds common elements and output the intersection $X \cap Y$ as follows: if $x \in X$ is mapped to bin j by function h_i then check whether $\mathsf{PRF}(k_j, x \parallel i) \in H_i$.
Obviously, this protocol is secure with semi-honest adversary by the security of OPRF under the assumption $(1.27n, \mathbb{F}_p, \mathbb{F}_{p^r})$-correlation robust of hash function H. For each $y \in Y \backslash X$ the corresponding output of function $\mathsf{PRF}(k_i, y)$ are pseudorandom. The correction of protocol based on the resistant collision of hashing function H i.e $F(k_i, x) \neq F(k_j, y)$ for $x \neq y$. This requirement leads to choosing parameters in section 4.1.3.

Theorem 2 *The protocol* PSI *(figure 4.3) is secure against a semi-honest adversary in the* $\mathcal{F}_{\mathsf{oprf}}$ *hybrid-model.*

Proof. Consider a semi-honest sender. The simulator interacts with sender as follow:

- The simulator plays the role of $\mathcal{F}_{\mathsf{oprf}}$. The simulator observes and gets the set of key $k, k_1, \ldots, k_{1.27n}$ then gives these keys to the receiver.
- Wait for the receiver to send $3n$ PRF values and sends back the intersection set $X \cap Y$ to the receiver.

Consider a semi-honest receiver. The receiver only sends to the sender $3n$ PRF value of the input set Y (each element in Y is computed 3 times PRF values). The simulator interacts with receiver as follows:

- The receiver as the sender in Π_{oprf} gets set of PRF keys $k, k_1, \ldots, k_{1.27n}$.
- Let denote $Z = X \cap Y$. For $\forall z \in Z$, $|Z| = n_z$, simulator computes 3 hash values of z: $(i_1, i_2, i_3) \leftarrow (h_1(z), h_2(z), h_3(z))$ and then adds $\mathsf{PRF}_{k,k_{i_j}}$ to the set H_j for $j \in [1, 2, 3]$.
- Add $(n - n_z)$ randomly values to each set H_j for $j \in [1, 2, 3]$ such that $3n$ values obtained are distinct.
- Simulator permutes and sends each set H_j to the sender.

Choosing Parameters

Under the assumption of $(n, \mathbb{F}_p, \mathbb{F}_{p^r})$-Correlation robustness, the size of our extended field $\mathbb{F}_q$ is $\approx 2^{\kappa + \log n}$. As discuss in section 4.1.2 the size of $\mathbb{F}_p$ need to be large enough for guaranteeing the probability existence $i \in [n]$ such that $u_i = 0$ is approximate $O(2^{-\lambda})$. i.e

$$\Pr\big(\ \exists i \in [n] :\ u_i = 0\big) \ = \ 1 - \left(\frac{p-1}{p}\right)^n \leq \frac{n}{p} \ \approx \ O(2^{-\lambda})$$

where n is the output size of $\mathcal{F}_{\mathsf{svole}}$. Then the size of $\mathbb{F}_p \approx 2^{\log(n) + \lambda}$.
In the construction of PSI protocol, the Cuckoo hashing is used to map n items to m bins by 3 hash functions. Our PSI protocol requires Cuckoo hashing without stash so the receiver can compute exactly 3 OPRF values per each element in the input set. To obtain no stash, the number of bins need to increase to $1.27n$ for failure probability $2^{-\lambda}$ [PSZ18].
The correction of PSI protocol required there is no collision in PRF i.e for $x \neq y$ then $\mathsf{PRF}(k_i, x) \neq \mathsf{PRF}(k_j, y)$. The overall probability of a collision is $2^{-\lambda}$ so we take the output domain of PRF is $\{0,1\}^v$ where $v = \lambda + 2\log_2(n)$.

4.2 Efficient analysis

4.2.1 Theoretical Comparison

In the table 4.1 we show the theoretical communication complexity of our protocol compared with the KKRT protocol [KKRT16], SpOT protocol [PRTY19], Chase-Miao protocol [CM20], Rindal-Schoppmann protocol [RS21] in semi-honest setting. This comparison measures how much communication the protocols require on an idealized network where we do not care about protocol metadata, realistic encoding, byte alignment, etc.

Protocol	Communication	$n = n_1 = n_2$				
		2^{10}	2^{12}	2^{16}	2^{20}	2^{24}
[KKRT16]	$1.2ln_1 + (3+s)(\lambda + \log(n_1 n_2))n_2$	$1078n$	$1114n$	$1042n$	$1018n$	$977n$
[PRTY19] SpOT-low	$ln_1 + 1.02(\lambda + \log(n_1) + 2)n_2$	$502n$	$504n$	$488n$	$500n$	$512n$
[PRTY19] SpOT-fast	$l(1+1/\lambda)n_1 + 2(\lambda + \log(n_1 n_2)n_2$	$580n$	$587n$	$583n$	$609n$	$634n$
[CM20]	$4.8\kappa n_1 + ((\lambda + \log(n_1 n_2))n_2$	$675n$	$679n$	$687n$	$695n$	$703n$
[RS21]	$2.4\kappa n_1 + ((\lambda + \log(n_1 n_2))n_2 + 2^{17}\kappa n_1^{0.05}$	large	large	$825n$	$424n$	$398n$
Our	$1.27(\lambda + \log(n_1))n_1 + 3((\lambda + \log(n_1 n_2))n_2 + t\log(1.27n_1)256 + 1152t$	$487n$	$323n$	$292n$	$317n$	$346n$

Table 4.1: Theoretical communication cost of PSI protocols (in bits), using the computational security $\kappa = 128$ and the statistical security $\lambda = 40$. Ignores the cost of base OTs (in [KKRT16, PRTY19, CM20]) which is not dependent on the length of input sets, n_1, n_2 are input sizes of sender and receiver respectively. l is width of OT extension matrix (depends on n_1 and each type of protocol). s is the size of stash using in cuckoo hashing (table 4.2). The parameter of our protocol is taken from [BCG+19a] where t is error weight.

For the set size range from 2^{20} to 2^{24}, our protocol has the least communication of any protocol we consider: $\approx 50\%$ less than KKRT, less than Chase-Miao and SpOT-fast protocol, and nearly similar to others. While Rindal-Schoppmann based on a linear solver method to encode the input set into a linear system and others [KKRT16, PRTY19, CM20] have to use a constant number of OT extension, $\approx 3.5\kappa$, our protocol simply sends a random vector which considered in a subfield $\mathbb{F}_p$ size $\lambda + \log(n_1)$. By the simplicity, we believe that our protocol gains a competitive running time.
The communication cost of our protocol is $1.27(\lambda + \log(n_1))n_1 + 3((\lambda + \log(n_1 n_2))n_2$ plus the communication for performing subVOLE size $1.27n_1$. The implementation of VOLE with dimension n need $t \log n$ OTs where the input of OTs with κ bits length and in addition one single call a reverse VOLE length t over $\mathbb{F}_q$. The most efficient protocol for reverse VOLE need to send v field elements and do v OTs. [ADI+17] implements the OTs directly from 1–2 OT which means an OT costs communication of 2 field elements and 1 bit. So we get a total of $3v$ field elements (plus v bits, which we can ignore when the field is large). The parameter v is roughly $3t$, so they have $9t$ field elements to send. Therefore the total communication cost for our protocol is $1.27(\lambda + \log(n_1))n_1 + 3((\lambda + \log(n_1 n_2))n_2 + t \log n_1 256 + 9t.128$ bits.

n	2^{12}	2^{16}	2^{20}	2^{24}
s	6	4	3	2

Table 4.2: The size of stash for failure probability $2^{-\lambda}$ [PSZ18].

[KKRT16, PRTY19] derived from [IKNP03], using a small number of OTs to obtains a constant l OT extension and then constructing a single point OPRF. These protocol use similar parameter l, around 3.5κ but the parameter in [KKRT16] is slightly larger. They also require sending 3.5κ

bits per element in OPRF. Cuckoo hashing approach in [KKRT16] gains a better running time compared with the linear solver method but it results in the need to send $(3+s)$ values of OPRF per element in Y. [PRTY19] has two variants for communication-optimized and speed-optimized but both of them rely heavily on the interpolation encoding method.
[CM20] transforms the idea of single point OPRF in [KKRT16] to multi-point OPRF. This protocol requires sending around 4.8κ per element, slight higher than [KKRT16] but the sender just needs to send one value OPRF $\lambda + \log(n_1 n_2)$ bits per element in set Y.
[RS21] also working on VOLE size κn_1 gains an efficient communication cost. They use PaSoX linear solver with a linear encoding rate is 2.4 to obtain a better computation but it induces a higher communication cost. They need to implement a VOLE with dimension $2.4n_1$ and then they shows the experimentally overhead cost to implementing VOLE size $2.4n_1$ is $2^{17}\kappa \sqrt[20]{n_1}$. This protocol requires sending κ bits per element while our protocol just slightly larger than $\log(X)$., Since the VOLE is highly sublinear then the overhead cost per element will decrease when we have a sufficiently large input size as $n_1 \geq 2^{16}$. [RS21] is not efficient for $n \leq 2^{16}$ while our is still efficient until $n = 2^{10}$.
Overall, our protocol based on two main techniques: cuckoo hashing and subVOLE. We use Cuckoo hashing with 3 hash functions and no stash to assign the input set into $1.27n_1$ bins and then protocol requires sending $\lambda + \log n_1$ bits for each hash element, which is slightly larger than the size bit of the element. Cuckoo hashing is more efficient compared with other linear solvers since it obtains small running times. So finding a solution with no stash and minimize the number of bins can make our protocol better.

4.2.2 Leakage version

We now present a PSI protocol with a weaker security. The communication of our PSI protocol (section 3) is contributed by sending a masking vector $\mathbf{t}$ with length $1.27n$ in $\mathbb{F}_p$ from one party and $3n$ OPRF values from another. We propose a protocol called leakage version to reduce the communication cost by minimizing as much as possible the large field base $\mathbb{F}_p$. In our leakage version, the size of $\mathbb{F}_p$ is slightly larger than the input set size.
Then we describe the ideal functionality of leakage OPRF in figure 4.4. Beyond the receiving the set of PRF keys, the sender gets $\overline{X}$. The set $\overline{X}$ is randomly samples related to the set X such that $|\overline{X}| = n_1$ and for every $\overline{x_i} \in \overline{X}$ we have $\overline{x_i} \neq x_i$ for $i \in [1, \ldots, n_1]$.

PARAMETERS:

- There are 2 parties the sender and a receiver.
- The receiver has a input set $X = \{x_1, x_2, \ldots, x_n\}$, $x_i \in \mathbb{F}_q$ for $i \in [n]$.

FUNCTIONALITY:

- For every $x_i \in X$ randomly samples $\overline{x_i} \in \mathbb{F} \setminus \{x_i\}_{i \leq n_1}$. Let denote $\overline{X} = \{x_i\}_{i \leq n_1}$.
- Choose a random seed for PRF $F : k*, k_1, k_2, \ldots, k_n$ and give these keys and the set $\overline{X}$ to the sender.
- Give the set $X' = \{F((k, k_1)x_1), F((k, k_2), x_2), \ldots, F((k, k_n), x_n)\}$ to the receiver.

Figure 4.4: The ideal functionality of $\mathcal{F}_{\text{loprf}}$

As mentioned in section 4.1.2, we must ensure that all the position in vector $\mathbf{u} \in \mathbb{F}_p^n$ are different from 0. Then for the statistical parameter security $\lambda = 40$, the bit length of each element of $\mathbb{F}_p$ is around $\log(n) + \lambda$. Then, we design the leakage version of our OPRF as follow:

1. Firstly, the receiver sends (receiver, sid) and sender sends (sender, sid) to $\mathcal{F}_{\text{svole}}$ dimension n with inverse role. The receiver gets two random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$ and the sender receives $\Delta \in \mathbb{F}_q$, $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$.

2. The receiver checks to make sure that all position in vector $\mathbf{u}$ are different from 0 then the protocol continues as protocol 4.2. If not, receiver will ask the sender to restart step 1.

Following the leakage OPRF, the ideal functionality of leakage PSI is presented in figure 4.5. Leakage PSI protocol is secure if the sender learns only about the intersection set, the receiver learns more than the sender information of set $\overline{X}$ and nothing more.

PARAMETERS:

- Given a finite field $\mathbb{F}$. The sender S has a set $X = \{x_1, \ldots, x_{n_1}\} \in \mathbb{F}^{n_1}$ and the receiver R has a set $Y = \{y_1, \ldots, y_{n_2}\} \in \mathbb{F}^{n_2}$.

FUNCTIONALITY:

1. The protocol waits until gets a set X from S and a set Y from R.
2. For every $x_i \in X$ randomly samples $\overline{x_i} \in \mathbb{F} \setminus \{x_i\}_{i \leq n_1}$. Let denote $\overline{X} = \{x_i\}_{i \leq n_1}$.
3. Output $X \cap Y, |Y|$ to S and $X \cap Y, \overline{X}$ to R.

Figure 4.5: The ideal functionality of leakage PSI

Our leakage PSI protocol uses leakage OPRF which realizes the ideal functionality of $\mathcal{F}_{\text{loprf}}$ instead of the construction of main OPRF (section 4.1.2. We will sketch the proof of security of our leakage PSI directly from the security of our main PSI.

Theorem 3 *The leakage* PSI *semi-honest adversary securely realizes the ideal functionality* $\mathcal{F}_{\text{lPSI}}$ *against a semi-honest adversary in the correlation robustness hash function over field,* $\mathcal{F}_{\text{loprf}}$ *hybrid model.*

Sketch proof of security.
Observe the proof of security of PSI protocol (theorem 2), the proof is described as follows:

- The simulator for the receiver is unchanged.
- However, for corrupted sender, the simulator plays the role in leakage OPRF to get not only the set of PRF keys but also the set $\overline{X}$. After that, the simulator sends all of these values to the receiver, and the process to simulate is the same.

Instead of choosing $\mathbb{F}_p$ for the probability existence $i \in [n]$ such that $u_i = 0$ is approximate $O(2^{-\lambda})$, we chooses a higher probability 2^{-k} where $k < \lambda$. This work leads to a smaller size of $\mathbb{F}_p$.

- The probability for the existence of $u_i = 0,\ i \in [n]$ is 2^{-k}, using linear property of expectation, we averagely need to recall $\mathcal{F}_{\text{svole}}$ for $1 + 2^{-k}$ times. Hence, the number of times for restarting $\mathcal{F}_{\text{svole}}$ is sufficient small and the communication cost for subVOLE is sublinear so we efficiently gain a reduce in communication.
- Because this version requires to recall $\mathcal{F}_{\text{svole}}$ with high probability, the information of the input set of the receiver will be revealed. Indeed, the sender gets more information about the vector $\mathbf{u}$ because all u_i is not equal to 0. Therefore, when sender receives the vector $\mathbf{t} = \mathbf{u} - \mathbf{x} \in \mathbb{F}_p^n$ as step 3 in protocol 4.2, sender learns the value of $x_i \neq -t_i$ for all $i \in [n]$. This leakage version can not be totally security but information disclosed is not negligible.

Bibliography

[ADI+17] B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 223–254. Springer, Heidelberg, August 2017.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

[BBC+11] P. Baldi, R. Baronio, E. D. Cristofaro, P. Gasti, and G. Tsudik. Countering gattaca: Efficient and secure testing of fully-sequenced human genomes. *CoRR*, abs/1110.2478, 2011.

[BCD+09] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BCG+19a] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG+19b] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Heidelberg, August 2019.

[BCGI18] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BDNP08] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: A system for secure multi-party computation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, page 257–266, New York, NY, USA, 2008. Association for Computing Machinery.

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 479–488, New York, NY, USA, 1996. Association for Computing Machinery.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Theory of Cryptography*, pages 336–354, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CCD88] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 11–19, New York, NY, USA, 1988. Association for Computing Machinery.

[CLR17] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

[CM20] M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO 2020, Part III*, *LNCS* 12172, pages 34–63. Springer, Heidelberg, August 2020.

[CO15] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, page 40–58, Berlin, Heidelberg, 2015. Springer-Verlag.

[DCT09] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptology ePrint Archive*, 2009:491, 01 2009.

[DCW13] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, page 789–800, New York, NY, USA, 2013. Association for Computing Machinery.

[DPSZ12] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, *LNCS* 7417, pages 643–662. Springer, Heidelberg, August 2012.

[DPT20] T. Duong, D. H. Phan, and N. Trieu. Catalic: Delegated psi cardinality with applications to contact tracing. Cryptology ePrint Archive, Report 2020/1105, 2020. `https://eprint.iacr.org/2020/1105`.

[EFG+09] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, pages 235–253, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[FIPR05] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography*, pages 303–324, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[FN13] T. K. Frederiksen and J. B. Nielsen. Fast and maliciously secure two-party computation using the gpu. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, page 339–356, Berlin, Heidelberg, 2013. Springer-Verlag.

[Gil99] N. Gilboa. Two party RSA key generation. In *CRYPTO'99*, *LNCS* 1666, pages 116–129. Springer, Heidelberg, August 1999.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[GPR+21] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. Springer-Verlag, 2021.

[HEK12a] Y. Huang, D. Evans, and J. Katz. 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.

[HEK12b] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols?, 2012.

[HFH99] B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. *ACM International Conference Proceeding Series*, 09 1999.

[IKN+19] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, M. Raykova, S. Saxena, K. Seth, D. Shanahan, and M. Yung. On deploying secure computing: Private intersection-sum-with-cardinality. *Journal of Cryptology*, 2019. `https://eprint.iacr.org/2019/723`.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, *LNCS* 2729, pages 145–161. Springer, Heidelberg, August 2003.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer — efficiently. In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, page 572–591, Berlin, Heidelberg, 2008. Springer-Verlag.

[IPS09] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, *Lecture Notes in Computer Science* 5444, pages 294–314. Springer, 2009.

[KK13] V. Kolesnikov and R. Kumaresan. Improved ot extension for transferring short secrets. In *Advances in Cryptology – CRYPTO 2013*, pages 54–70, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KKRT16] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

[KM08] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better bloom filter. *Random Struct. Algorithms*, 33(2):187–218, September 2008.

[KOS16] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

[KRTW19] V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT 2019, Part II*, *LNCS* 11922, pages 636–666. Springer, Heidelberg, December 2019.

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, page 14, USA, 2012. USENIX Association.

[LOP11] Y. Lindell, E. Oxman, and B. Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, page 259–276, Berlin, Heidelberg, 2011. Springer-Verlag.

[Mea86] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134, 1986.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. *CoRR*, abs/1202.3052, 2012.

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. pages 448–457, 01 2001.

[NP06] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35:1254–1281, 2006.

[OPJM10] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. Scifi - a system for secure face identification. In *2010 IEEE Symposium on Security and Privacy*, pages 239–254, 2010.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122 – 144, 2004.

[PRTY19] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 401–431. Springer, Heidelberg, August 2019.

[PRTY20] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT 2020, Part II*, *LNCS* 12106, pages 739–767. Springer, Heidelberg, May 2020.

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT 2009*, pages 250–267, 12 2009.

[PSSZ15] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, Washington, D.C., August 2015. USENIX Association.

[PSTY19] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT 2019, Part III*, *LNCS* 11478, pages 122–153. Springer, Heidelberg, May 2019.

[PSWW18] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT 2018, Part III*, *LNCS* 10822, pages 125–157. Springer, Heidelberg, April / May 2018.

[PSZ14] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

[PSZ18] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), January 2018.

[Rab05] M. O. Rabin. How to exchange secrets with oblivious transfer, 2005. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005.

[RR17] P. Rindal and M. Rosulek. Malicious-secure private set intersection via dual execution. In *ACM CCS 2017*, pages 1229–1242. ACM Press, October / November 2017.

[RS21] P. Rindal and P. Schoppmann. Vole-psi: Fast oprf and circuit-psi from vector-ole. *IACR Cryptol. ePrint Arch.*, 2021:266, 2021.

[SGRR19] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

[Sha80] A. Shamir. On the Power of Commutativity in Cryptography. In *Proceedings of the Seventh Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 85, pages 582–595. Springer-Verlag, 1980.

[SSS12] E. Stefanov, E. Shi, and D. Song. Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In *Public Key Cryptography – PKC 2012*, pages 413–430, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Yak17] S. Yakoubov. A gentle introduction to yao ' s garbled circuits. 2017.

[Yao82] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[YY13] J. Yuan and S. Yu. Efficient privacy-preserving biometric identification in cloud computing. In *2013 Proceedings IEEE INFOCOM*, pages 2652–2660, 2013.