Lecture note 9: Privacy-Enhancing Technologies 3 Secure Multiparty Computation

Donghui Dai, Huaien Zhang

April 10, 2023

In this lecture, we first discuss the concept and definitions for secure computation of the real-ideal world paradigm. Technically, we introduce two security adversary models under this paradigm, i.e., semi-honest and malicious security. Second, we give a detailed introduction to the general constructions for secure multi-party computation. Specifically, Yao's protocol is illustrated coupled with the GMW. Finally, we introduce the custom protocol: private set intersection.

1 Secure Computation: Concepts & Definitions

In an informal context, Multi-Party Computation (MPC) is concerned with enabling a collective of participants to acquire knowledge regarding the accurate output of a jointly agreed upon function applied to their individual, confidential inputs, without disclosing any additional information. Two or more parties need to perform some agreed-upon computation while guaranteeing "security" against "adversarial behavior". Specifically, a general security requirement should take privacy, correctness, independence of inputs and fairness into consideration. A conventional method of establishing security is to devise a catalogue of actions or events that signify a breach of security. However, directly leveraging conditions to confine the security properties is cumbersome and error-prone since we cannot guarantee the list could be considered complete. Therefore, we need to define security formally in a novel *real-ideal paradigm* that forms the conceptual core of secure computation.

1.1 Real-Ideal Paradigm

The *real-ideal paradigm* circumvents the potential pitfall mentioned above entirely through the introduction of an "ideal world" that implicitly encompasses all security assurances, thereby enabling the definition of security in relation to this ideal world [SS84]. The *real-ideal paradigm* consists of two worlds, i.e., the ideal world and the real world.

Ideal World In the ideal scenario, the function $\mathcal{F}$ is computed securely by the involved parties, who confidentially transfer their respective inputs to a completely trusted third party, designated as the functionality $\mathcal{T}$. Each party has his/her own input x_i, which is transmitted to $\mathcal{T}$. Next, the $\mathcal{T}$ simple computes $\mathcal{F}(x_1, ..., x_n)$ and return the

result to all parties. An adversary in the ideal world may take control over any of the parties P_i instead of $\mathcal{T}$, the simplicity of the ideal world makes the attack could be recognized easily since the adversary's choice input is independent of the honest parties'.

Real World In the real world, no trusted party exists. As an alternative, all parties must engage in communication through a prescribed protocol. Protocol π specifies for each party P_i a "next message" function π_i. The function accepts several parameters, including the security parameter, the party's private input x_i, a randomized tape, and a record of the messages that the party P_i has received so far. Subsequently, π_i generates an output, wherein it either issues a directive for the party to send the next message along with its destination or alternatively terminates the communication and yields specific outputs as per the instruction. In the real world, an adversary can corrupt parties.

Security Definition For every real-world adversary A, there exists an ideal adversary A' s.t. joint distribution (HonestOutput, AdvOutput) is indistinguishable. Namely, there exists a *simulator* that could simulate the real world in an ideal world.

Remark 1. *In essence, the security of the actual protocol π is ascribed to it if any impact that an adversary can generate in the real world can also be replicated by a corresponding adversary in the ideal world. The goal of a protocol is to provide security in the real world that is equivalent to that in the ideal world.*

1.2 Semi-Honest Security

A semi-honest adversary is an entity that engages in the corruption of parties, yet adheres to the specified protocol. In other words, the corrupt parties run the protocol honestly but they may try to learn as much as possible from the messages they receive from other parties. Multiple colluding corrupted parties may collaborate to pool their perspectives, enabling the acquisition of confidential information. Semi-honest adversaries are also commonly called passive honest-but-curious since they cannot take any actions other than attempting to learn private information by observing execution results.

Informally, suppose π is the protocol and $\mathcal{F}$ is a functionality. Let C symbolize the collection of corrupted parties, whilst *Sim* refers to a simulator algorithm. The ensuing distribution of random variables may be defined as follows:

- $\boldsymbol{Real}_{\pi}(\kappa, C; x_1, ..., y_n)$: Execute the protocol with a security parameter κ, wherein each party P_i operates with integrity by employing its own private input x_i. Denote the ultimate perspective of party P_i as V_i, and its resultant output as y_i, we have the output in the form of $\{ V_i \mid i \in C \}$, $(y_1, ..., y_n)$.

- $\boldsymbol{Ideal}_{\mathcal{F},Sim}(\kappa, \mathrm{C}; x_1, \ldots, x_n)$: Compute $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$
  Output $Sim(C, (x_i, y_i) \mid i \in C), (y_1, \ldots, y_n)$.

A protocol is considered secure against semi-honest adversaries if the views of corrupted parties in the real world are statistically indistinguishable from their views in the ideal world.

Definition 1. *A protocol π securely realizes function $\mathcal{F}$ in the presence of semi-honest adversaries if there exists a simulator Sim such that for every subset of corrupted parties C and all inputs $x_1, \ldots, x_n$, the distributions*

$$\boldsymbol{Real}_{\pi}(\kappa, C; x_1, ..., x_n)$$

and

$$\boldsymbol{Ideal}_{\mathcal{F},Sim}(\kappa, C; x_1, \ldots, x_n)$$

are ***indistinguishable*** *(in κ).*

1.3 Malicious Security

A malicious adversary, also known as an active adversary, may attempt to violate security by causing corrupted parties to deviate arbitrarily from the prescribed protocol. Unlike a semi-honest adversary, a malicious adversary has the power to take any actions during protocol execution in addition to analyzing the protocol execution. Note that this includes an adversary who has the ability to control, manipulate, and inject messages arbitrarily onto the network. Compared to the previous one, security in this setting have two important additions to consider:

Effect on honest outputs. When the corrupted parties deviate from the protocol, it introduces the possibility of affecting the outputs of honest parties

Extraction. In a secure protocol, honest parties follow a well-defined input, while the input of a malicious party is not well-defined in the real world. The simulator chooses inputs for the corrupt parties in the ideal world, which should achieve the same effects as in the real world. This process is known as extraction.

Informally, suppose $\mathcal{A}$ to denote the adversary program and $corrupt(\mathcal{A})$ to denote the set of corrupted parties. Similarly, $corrupt(Sim)$ represents the set of corrupted parties by the ideal adversary Sim. To define a secure protocol, we establish distributions for the real and ideal worlds, where the protocol makes these distributions indistinguishable. This is done in a similar manner to the definition of security against semi-honest adversaries:

- $\boldsymbol{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i \mid i \notin \text{corrupt}(\mathcal{A})\})$: The protocol is execute with security parameter κ, where each honest party P_i(for $i \notin corrupt(\mathcal{A})$) runs the protocol honestly using the given private input x_i, and the messages of corrupt parties are chosen according to $\mathcal{A}$ (where A is considered as the protocol's next-message function for a collection of parties). The output of each honest party P_i is denoted by y_i, and the final view of party P_i is denoted by V_i. The output is $(\{V_i \mid i \in corrupt(\mathcal{A}), \{y_i \mid i \notin corrupt(\mathcal{A})\})$.
- $\boldsymbol{Ideal}_{F,Sim}(\kappa; \{x_i \mid i \notin corrupt(\mathcal{A})\})$: Run Sim until it outputs a set of inputs $\{x_i \mid i \in corrupt(\mathcal{A})\}$. Compute $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$. Then, give $\{y_i | i \in corrupt(\mathcal{A})\}$ to Sim. Let $V^\star$ denote the final output of Sim, we can have:
  Output $(V^\star, y_i | i \notin corrupt(Sim))$.

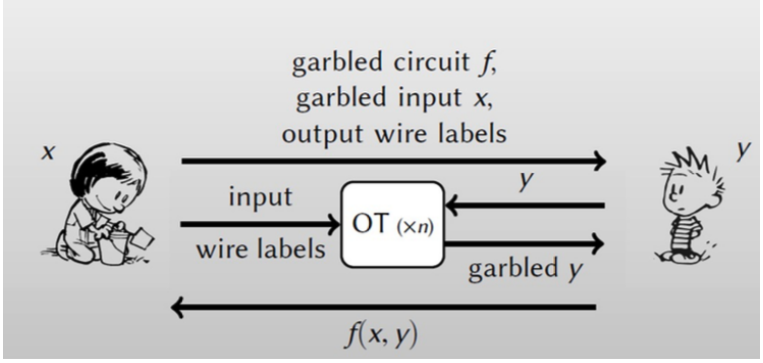

Figure 1: Yao's Protocol from lecture slide p56.

A protocol is considered secure against semi-honest adversaries if the views of corrupted parties in the real world are statistically indistinguishable from their views in the ideal world.

Definition 2. *A protocol π securely realizes function $\mathcal{F}$ in the presence of malicious adversaries if for every real-world adversary $\mathcal{A}$ there exists a simulator Sim with $corrupt(\mathcal{A}) = corrupt(Sim)$ such that, for all inputs for honest parties $\{x_i \mid i \notin corrupt(\mathcal{A})\}$, the distributions*

$$\boldsymbol{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i \mid i \notin corrupt(\mathcal{A})\})$$

and

$$\boldsymbol{Ideal}_{\mathcal{F},Sim}(\kappa; \{x_i \mid i \notin corrupt(Sim)\})$$

*are **indistinguishable** (in κ).*

Note that the definition quantifies only over the inputs of honest parties $\{x_i \mid i \notin corrupt(\mathcal{A})\}$.

2 Yao's Protocol for Secure MPC

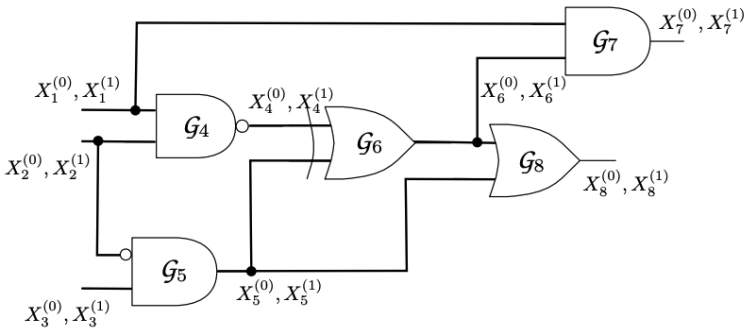

Figure 2: A garbled circuit example.

Generally, every computation of function could be transferred to computing a Boolean circuit. Yao's protocol is the most popular semi-honest secure (2-party) computation for

Boolean circuits. The main idea behind Yao's GC approach is quite natural. Fig. 1 demonstrates the basic workflow of Yao's protocol. At a very high level, the idea can be described as follows:

- P_1 will produce a "garbled encoding" of the circuit and transmit it to P_2. An example of the garbled circuit is shown in Fig. 2
- P_1 and P_2 subsequently engage in a specialized interactive subprotocol that enables P_2 to acquire 'garbled encodings' of their respective inputs, without disclosing any information about P_1's inputs to P_2, and vice versa. This subprotocol ensures that neither party gains any insight into the other's inputs.
- After obtaining the 'garbled encodings' of the circuit and inputs, P_2 proceeds to execute a dedicated evaluation algorithm locally, enabling it to calculate the 'garbled encodings' of the outputs. An example of the oblivious evaluation of the garbled circuit is shown in Fig. 3
- P_2 subsequently transmits the resulting 'garbled encodings' of the outputs to P_1, which enables it to calculate the actual outputs while keeping everything else private.
- Ultimately, P_1 forwards the computed actual outputs to P_2.

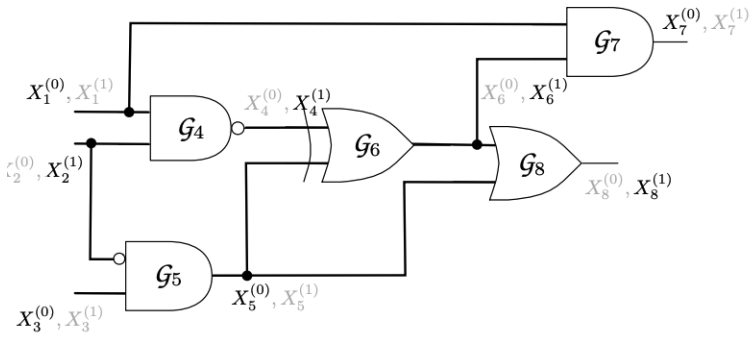

Figure 3: A oblivious evaluation example of a garbled circuit.

As an initial step in elaborating on the idea outlined above, we provide a precise and formal definition of the syntax that characterizes a garbling scheme as shown in Fig 4. By doing so, we aim to establish a clear framework and set of guidelines that can be used to design and implement secure cryptographic protocols based on the concept of garbled circuits.

Definition 3. ***Yao's Garbling scheme.*** *A garbling scheme G consists of four polynomial-time algorithms (Gb, En, Ev, De) respectively [Yak17]:*

- *The probabilistic circuit* ***garbling algorithm****, known as Garble, is called upon as part of the process:*

$$(\mathcal{F}, e, d) \xleftarrow{R} Garble(f)$$

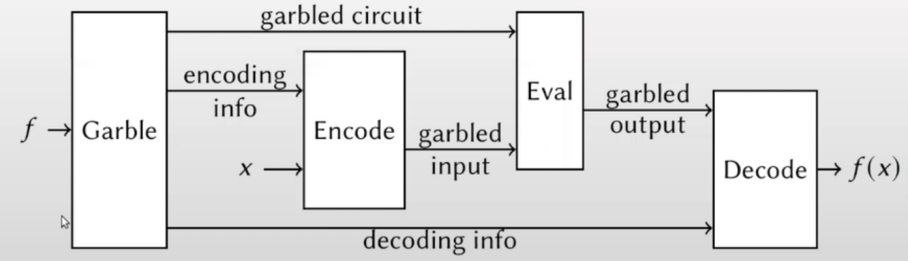

Figure 4: Yao's Protocol

where the input f is a boolean circuit. The result $\mathcal{F}$ is called a ***garbled encoding*** *of f, the result e is called the* ***input encoding data****, and the result d is called the* ***output decoding data.***

- *The deterministic* ***input encoding algorithm****, called Encode, is invoked during the procedure:*

$$X \leftarrow Encode(e, x)$$

where e is the input encoding data, and x is a vector of bits. The result X is called a garbled encoding of x.

- *A deterministic garbled* ***circuit evaluation algorithm*** *Evaluation that is invoked as*

$$\mathcal{Y} \leftarrow Evaluation(\mathcal{F}, X)$$

where $\mathcal{F}$ is a garbled encoding of a circuit and X is a garbled encoding of an input vector. The result $\mathcal{Y}$ is called a garbled output.

- *A deterministic* ***output decoding algorithm*** *Decode that is invoked as:*

$$y \leftarrow Decode(d, \mathcal{Y})$$

where d is the output decoding data and $\mathcal{Y}$ is a garbled output. The result y is either the special symbol reject or a vector of bits.

Therefore, we can re-express our high-level concept, presented earlier, in the terminology of garbling schemes as follows [BS20]:

- P_1 runs $(\mathcal{F}, e, d) \xleftarrow{R} Garble(f)$ and sends $\mathcal{F}$ to P_2.
- P_1 and P_2 then execute a special interactive subprotocol that lets P_2 obtain $X := Encode(e, x)$, where x is the vector comprising both P_1's and P_2's inputs. The P_2 knows nothing about P_1' inputs in this step and only X is share between the P_1 and P_2.
- P_2 executes $\mathcal{Y} \leftarrow Evaluation(\mathcal{F}, X)$ and send the evaluated $\mathcal{Y}$ to P_1 for decoding.
- P_1 runs $y \leftarrow Decode(d, \mathcal{Y})$ and send y to P_2. The P_1 learns nothing other than y.

This is the mathematical expression of Yao's Protocol.

3 Goldreich-Micali-Wigderson (GMW) Protocol

Computation related to encryption can be viewed as an operation on shared data in secrecy. In Yao's Garbled Circuit (GC), the active wire value is secretly shared by letting one player, called the generator, possess two potential wire labels w_i^0 and w_i^1, and the other player, called the evaluator, holding the active label w_i^b. On the other hand, the GMW protocol [MGW87] directly splits the wire value into additive shares among the players. Unlike Yao's GC, the GMW protocol (or simply "GMW") can naturally extend to more than two parties without requiring novel techniques.

3.1 GMW Intuition

The GMW protocol is capable of operating on both Boolean and arithmetic circuits. We begin by presenting the two-party Boolean variant of the protocol, followed by a brief explanation on how it can be extended to more than two parties. In Yao's protocol, it can be inferred that players P_1 and P_2 with inputs x and y respectively, have reached an agreement on the Boolean circuit C that represents the computed function $\mathcal{F}(x, y)$.

It is assumed that the communication between the parties takes place over an asynchronous communication network denoted as $\mathcal{C}$. We also assume that the network provides secure point-to-point channels, which ensure both message privacy and integrity. The GMW protocol, which is a two-party version, can be described as follows:

1. For each input bit $x_i \in \{0, 1\}$ of the input $x \in \{0, 1\}^n$, Party P_1 generates a random bit $r_i \in \{0, 1\}$ and transmits all r_i to Party P_2.

2. Party P_1 then obtains a secret sharing of each x_i between itself and P_2 by setting its share to be $x_i \oplus r_i$.

3. Similarly, Party P_2 generates random bit masks for its inputs y_i and sends them to P_1, secret sharing its input in a similar manner.

Following this step, Party P_1 and Party P_2 can proceed to evaluate the circuit C gate by gate. For instance, let us consider a gate G with input wires w_i and w_j and output wire w_k. The input wires can be split into two shares, such that $s_x^1 \oplus s_x^2 = w_x$. In this case, Party P_1 holds shares s_i^1 and s_j^1 on wires w_i and w_j, while Party P_2 holds shares s_i^2 and s_j^2 on the corresponding wires. For the purpose of this discussion, we assume that the circuit C contains only NOT, XOR and AND gates, without loss of generality.

Interaction is unnecessary for evaluating a NOT or XOR gate. To evaluate a NOT gate, P_1 can flip its share of the wire value, resulting in a flipped shared wire value. For an XOR gate on wires w_i and w_j, players compute their output shares by xor-ing the shares they hold. P_1 computes its output share as $s_k^1 = s_i^1 \oplus s_j^1$, and P_2 computes its output share as $s_k^2 = s_i^2 \oplus s_j^2$. The computed shares, s_k^1 and s_k^2, represent the shares of the active output value: $s_k^1 \oplus s_k^2 = (s_i^1 \oplus s_j^1) \oplus (s_i^2 \oplus s_j^2) = (s_i^1 \oplus s_i^2) \oplus (s_j^1 \oplus s_j^2) = w_i \oplus w_j$.

On the other hand, evaluating an AND gate requires interaction and the use of a basic primitive called a 1-out-of-4 OT. From P_1's perspective, its shares s_i^1 and s_j^1 are fixed, and P_2 has two boolean input shares, resulting in four possible input options for P_2. If P_1 knew P_2's

shares, then evaluating the gate under encryption would be easy: P_1 could reconstruct the active input values, compute the active output value, and secret-share it with P_2. However, P_1 cannot do that, so it instead prepares a secret share for each of P_2's possible inputs and runs a 1-out-of-4 OT to transfer the corresponding share.

To accomplish this, P_1 chooses a random mask bit $r \in_R 0, 1$ and prepares a table of OT secrets:

$$T_G = \begin{pmatrix} r \oplus S(0,0) \\ r \oplus S(0,1) \\ r \oplus S(1,0) \\ r \oplus S(1,1) \end{pmatrix}$$

Here, $S = S_{s_i^1, s_j^1}(s_i^2, s_j^2) = (s_i^1 \oplus s_j^2) \wedge (s_j^1 \oplus s_j^2)$ is the function computing the gate output value from the shared secrets on the two input wires. P_1 and P_2 then run an 1-out-of-4 OT protocol, with P_1 playing the role of the sender and P_2 playing the role of the receiver. P_1 uses table rows as each of the four input secrets, and P_2 uses its two bit shares as the selection to choose the corresponding row. P_1 keeps r as its share of the gate output wire value, and P_2 uses the value it receives from the OT execution.

Due to the construction of the oblivious transfer (OT) inputs, players in a two-party setting obtain a secret sharing of the gate output wire, without learning anything about the other player's inputs or intermediate computation values. In effect, only Player 2 (P_2) receives messages, and based on the OT guarantee, it cannot obtain any information about the three OT secrets it did not select. The only thing P_2 learns is its OT output, which is its share of a random sharing of the output value, and thus leaks no information about the plaintext value on that wire. Similarly, P_1 learns nothing about the selection of P_2.

After evaluating all gates, players reveal to each other the shares of the output wires to obtain the output of the computation.

We now propose an approach for generalizing this to a setting where n players $P_1, P_2, \cdots, P_n$ evaluate a Boolean circuit $\mathcal{F}$. As before, player P_j secret-shares its input by selecting $r_i \in_R \{0, 1\}$ for all $i \neq j$ and sending r_i to each P_i. The parties $P_1, P_2, \cdots, P_n$ proceed to evaluate C gate-by-gate, as follows:

- For an XOR gate, the players locally add their shares. No interaction is required, and correctness and security are guaranteed.
- For an AND gate $c = a \wedge b$, let $a_1, \cdots, a_n$ and $b_1, \cdots, b_n$ denote the shares of a and b, respectively, held by the players. Consider the identity:

$$\begin{aligned} c = a \wedge b &= (a_1 \oplus \cdots \oplus a_n) \wedge (b_1 \oplus \cdots \oplus b_n) \\ &= (\bigoplus_{i=1}^{n} a_i \wedge b_i) \oplus (\bigoplus_{i \neq j} a_i \wedge b_j) \end{aligned}$$

Each player P_j computes $a_j \wedge b_j$ locally to obtain a sharing of $\oplus_{i=1}^{n} a_i \wedge b_i$. Further, each pair of players P_i and P_j jointly computes the shares of $a_i \wedge b_j$ using the two-party GMW protocol as described earlier. Finally, each player outputs the XOR of all obtained shares as the sharing of the result $a \wedge b$.

3.2 BGW protocol

An effective multi-party protocol for secure computation was proposed by Ben-Or, Goldwasser, and Wigderson [WOG88], referred to as the "BGW" protocol. Concurrently, Chaum, Crépau, and Damgård published a somewhat similar protocol [CCD88], and the two are often considered in tandem. For the sake of clarity, we outline the BGW protocol for n parties here, which is relatively easy to grasp.

The BGW protocol facilitates the evaluation of an arithmetic circuit over a field $\mathbb{F}$, which includes addition, multiplication, and multiplication-by-constant gates. The protocol is heavily dependent on Shamir secret sharing [Sha79], which utilizes the homomorphic property of Shamir secret shares in a unique way—the underlying shared value can be manipulated obliviously through the appropriate manipulations to the individual shares.

The process of Shamir secret sharing involves representing a value v in the field $\mathbb{F}$ as $[v]$, denoting the fact that the parties possess Shamir secret shares of v. This is achieved by a dealer who chooses a random polynomial p of degree at most t, such that $p(0) = v$. Each party, denoted by P_i, receives a share $p(i)$, which together with the shares of other parties forms the complete secret sharing of v. The parameter t represents the threshold of the sharing, implying that a group of up to t parties cannot obtain any information about the original value v.

The key feature of the BGW protocol is that for every wire w in the arithmetic circuit, the parties hold a secret-sharing $[v_w]$ of the value v_w on that wire. This invariant is maintained throughout the protocol, and forms the foundation for secure computation. To elaborate on the protocol, we provide a brief outline of its execution, which involves performing operations on the secret shares while ensuring the invariant is preserved at each step.

Addition Gate: To compute the output of an addition gate, parties collectively hold shares of the incoming wires, $[v_\alpha]$ and $[v_\beta]$, and aim to get a sharing of $[v_\alpha + v_\beta]$. The corresponding polynomials $p_\alpha(x)$ and $p_\beta(x)$ can be locally added by each party to produce a new set of shares $p_\gamma(i) = p_\alpha(i) + p_\beta(i)$, where $p_\gamma(x)$ is a polynomial with degree at most t. Since $p_\gamma(0) = v_\alpha + v_\beta$, the resulting values comprise a valid sharing of $[v_\alpha + v_\beta]$. Remarkably, this computation requires no communication among the parties.

Multiplication Gate: This kind gate is more complicated. To obtain a sharing of $[v_\alpha \cdot v_\beta]$, parties hold shares of the incoming wires $[v_\alpha]$ and $[v_\beta]$. Each party can locally multiply their shares to obtain a point on the polynomial $q(x) = p_\alpha(x) \cdot p_\beta(x)$. However, the resulting polynomial can have a degree as high as $2t$, which is not acceptable. The BGW protocol uses "Beaver triples" to address this issue, where the parties jointly generate triples (a_i, b_i, c_i) such that $c_i = a_i \cdot b_i$. Using these triples, a new polynomial with degree at most t can be constructed to share the same value as the original polynomial on the input points. This technique requires some communication among the parties but is still practical. Once parties obtain a valid sharing of $[v_\alpha \cdot v_\beta]$, they can move on to the next gate in the circuit.

To address the problem of excessive degree in the secret sharing scheme, the parties take a crucial step of degree reduction. Each party P_i has a value $q(i)$, which is part of a polynomial of degree at most $2t$. The objective is to obtain a valid secret-sharing of $q(0)$, but with the correct threshold.

An important observation is that $q(0)$ can be expressed as a linear combination of the shares held by the parties. Specifically,

$$q(0) = \sum_{i=1}^{2t+1} \lambda_i q(i)$$

where λ_i denotes the appropriate Lagrange coefficients. Therefore, the degree-reduction step proceeds as follows.

First, each party P_i generates and distributes a threshold-t sharing of $[q(i)]$. It is noteworthy that each party selects a polynomial of degree at most t, whose constant coefficient is $q(i)$. Next, the parties perform local computations to compute $[q(0)] = \sum_{i=1}^{2t+1} \lambda_i [q(i)]$. It is important to note that this expression involves addition and multiplication-by-constant operations applied to secret-shared values.

Overall, this degree-reduction step enables the parties to obtain a valid secret-sharing of $q(0)$ with the appropriate threshold, thereby avoiding the problem of excessive degree in the secret sharing scheme.

To obtain a valid secret-sharing of $q(0)$ with the desired threshold, a degree-reduction step is necessary. Each party P_i holds a value $q(i)$ from a polynomial of degree at most $2t$. To reduce the degree of the polynomial, the parties observe that $q(0)$ can be expressed as a linear function of the party's shares. Specifically, the Lagrange coefficients λ_i can be used to obtain $q(0)$ as a summation of each party's share.

In the degree-reduction step, each party generates and distributes a threshold-t sharing of $[q(i)]$, where $q(i)$ is a polynomial of degree at most t with constant coefficient $q(i)$. The parties then use local computations to sum the shared values and obtain $[q(0)]$. It is important to note that since the values $[q(i)]$ were shared with threshold t, the final sharing of $[q(0)]$ also has threshold t.

However, it is important to remember that the BGW protocol's multiplication gates require communication and interaction, in which parties must send shares of $[q(i)]$. To ensure the protocol's security, it is necessary to have $2t + 1 \leq n$, as otherwise, $q(0)$ may have degree $2t$, and the n parties may not have enough information to determine its value. Hence, the BGW protocol is secure against t corrupt parties, provided that $2t < n$ (i.e., an honest majority).

When it comes to output wires, the parties will eventually hold shares of the value $[v_\alpha]$ on the wire α. Each party can simply broadcast its share of this value to enable all parties to learn v_α.

4 Custom Protocols

Until now, we have discussed generic circuit-based protocols as the main class of secure computation protocols. However, circuit-based protocols are limited by the linear bandwidth cost in the size of the circuit, which can become prohibitively expensive for large computations. Moreover, the overhead associated with circuit-based computation on large data structures is significant when compared to using a Random Access Machine (RAM) representation. To address this, one approach is to incorporate sublinear data structures into generic circuit-based protocols.

Alternatively, a customized protocol may be designed for a specific problem. However, this approach has some significant disadvantages over using a generic protocol. Firstly,

it requires the design and proof of security for a custom protocol. Secondly, it may not integrate with generic protocols, so even if an efficient custom protocol for computing a specific function exists, it may not be possible to use it without also developing methods for connecting it with a generic protocol. Finally, while generic protocols have hardening techniques, it may not be possible to (efficiently) harden a customized protocol to work in a malicious security setting.

Despite these disadvantages, several specialized problems benefit from tailored solutions, and the performance gains possible with custom protocols can be substantial. In this work, we briefly review one such practically important problem: private set intersection.

4.1 Private Set Intersection (PSI)

The objective of Private Set Intersection (PSI) is to enable a group of parties to jointly compute the intersection of their input sets without revealing any additional information about the sets, except for the upper bounds on their sizes. While protocols for PSI can be constructed based on generic Secure Multi-Party Computation (MPC) [HEK12], custom protocols that leverage the structure of the problem can achieve better efficiency.

First, we use a simple and vivid case shown in the figure 5 to illustrate the PSI. In this figure, Bob wants to know whether Alice has a number y_i in the set X. Hence, they send holding sets to PSI and calculate the intersection set to solve Bob's problem. Obviously, we can use hash function to calculate the key values and send to PSI, but this approach is not secure, especially when the numbers have a low entropy, e.g., phone numbers. Another promising approach is using Diffie-Hellman algorithm, but its drawback is a little time-consuming.

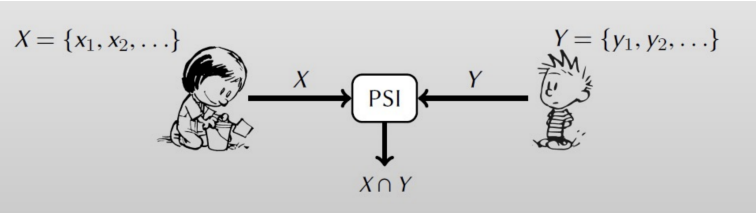

Figure 5: A motivating example of PSI

We will now present the current state-of-the-art two-party private set intersection (PSI) protocol, as outlined in [KKRT16]. This protocol is built upon the protocol proposed in [PSSZ15], which heavily relies on Oblivious PRF (OPRF) as a subroutine. OPRF is a type of secure multi-party computation (MPC) protocol that allows two players to compute a pseudorandom function (PRF) F in such a way that the player who holds the PRF key k never learns the input x held by the other player, who in turn obtains $F_k(x)$. In what follows, we will first describe how PSI can be obtained from OPRF and then provide a brief overview of the OPRF construction. The main improvement in [KKRT16] is a faster OPRF.

Obtaining PSI from OPRF. We will describe the Pinkas-Schneider-Segev-Zohner (PSSZ) construction [PSSZ15], which builds PSI from an OPRF. To be specific, we will describe the parameters used in PSSZ when the two parties have roughly the same number n of items.

The PSSZ protocol relies on Cuckoo hashing [PR04], which uses three hash functions. To assign n items into b bins using Cuckoo hashing, we first select three random hash functions h_1, h_2, and h_3 from $0, 1^*$ to $[b]$, and initialize empty bins $\mathcal{B}[1, \cdots, b]$. To hash an item x, we check if any of the bins $\mathcal{B}[h_1(x)]$, $\mathcal{B}[h_2(x)]$, or $\mathcal{B}[h_3(x)]$ are empty. If an empty bin is found, we place x in that bin and terminate. Otherwise, we choose a random $i \in 1, 2, 3$, evict the item currently in $\mathcal{B}[h_i(x)]$, and replace it with x. We then recursively try to insert the evicted item. If this process does not terminate after a certain number of iterations, then the final evicted element is placed in a special bin called the stash.

PSSZ uses Cuckoo hashing to implement PSI. Suppose party P_1 has input set X and party P_2 has input set Y, where $|X| = |Y| = n$. Party P_2 maps its items into $1.2n$ bins using Cuckoo hashing, along with a stash of size s. At this point, party P_2 has at most one item per bin and at most s items in its stash. Party P_2 pads its input with dummy items so that each bin contains exactly one item and the stash contains exactly s items.

To obtain private set intersection, the two parties in the protocol of [KKRT16] execute $1.2n + s$ instances of an Oblivious PRF (OPRF). The protocol uses the Pinkas-Schneider-Segev-Zohner(PSSZ) construction [PSSZ15], which builds PSI from an OPRF. In the i-th OPRF instance, P_2 acts as the receiver and inputs each of its $1.2n + s$ items to the OPRF. The PRF evaluated in this instance is denoted by $F(k_i, \cdot)$. If P_2 has mapped item y to bin i using Cuckoo hashing, then P_2 gains knowledge of $F(k_i, y)$; if y is mapped to position j in the stash, then P_2 learns $F(k_{1.2n+j}, y)$.

On the other hand, P_1 can compute $F(k_i, \cdot)$ for any i. So, P_1 computes sets of candidate PRF outputs:

$$H = \{F(k_{h_{i(x)}}, x) | x \in X \; and \; i \in \{1, 2, 3\}\}$$
$$S = \{F(k_{1.2n+j}, x) | x \in X \; and \; j \in \{1, \cdots, s\}\}$$

To identify the intersection of X and Y, P_1 randomly permutes the elements of H and S and sends them to P_2. P_2 can then check for intersections as follows: if an item y is mapped to the stash, P_2 verifies whether the associated output of the Oblivious Pseudorandom Function (OPRF) is present in S. On the other hand, if an item y is mapped to a hashing bin, P_2 verifies whether the associated OPRF output is present in H.

The security of the protocol against a semi-honest P_2 is based on the pseudorandomness of the PRF outputs. For an item x in X but not in Y, the corresponding PRF outputs $F(k_i, y)$ are pseudorandom. The PRF outputs should also remain pseudorandom under related keys to ensure the safety of instantiating the PRF instances with related keys.

The protocol is considered correct as long as the PRF does not introduce further collisions, meaning that for $x \neq x'$, $F(k_i, x) \neq F(k_{i'}, x')$. Proper parameter setting is crucial to prevent such collisions.

A more efficient OPRF construction for the Private Set Intersection (PSI) protocol was introduced in [KKRT16]. The construction is based on the observation that the code C does not necessarily need to have the full set of properties of error-correcting codes. The resulting pseudorandom codes enable a 1-out-of-∞ Oblivious Transfer (OT) protocol that can be used to produce an efficient PSI.

In particular,

- The method does not rely on decoding, hence the code does not need to be efficiently decodable.
- The requirement is that for all possibilities r and r_0, the Hamming weight of $C(r) \oplus C(r_0)$ should be at least the computational security parameter $\mathbf{k}$. Although it suffices for this Hamming distance guarantee to hold with overwhelming probability over the choice of C, there are some subtle nuances to be aware of, which we will discuss in greater detail.

Let us assume that C is a random oracle with suitably long output for the sake of convenience. It is generally challenging to find a near-collision when C is long enough. This means it is hard to find values r and r_0 such that $C(r) \oplus C(r_0)$ has a low (less than a computational security parameter $\mathbf{k}$) Hamming weight. A pseudorandom code (PRC) with an output length of $k = 4\mathbf{k}$ is sufficient to make near-collisions negligible.

A pseudorandom code (PRC) can be defined as a function C (or family of functions, in our standard model instantiation) that holds coding-theoretic properties, namely, minimum distance, in a cryptographic sense.

By relaxing the requirement on C from an error-correcting code to a pseudorandom code, we eliminate the a-priori bound on the size of the receiver's choice string. This means that the receiver can use any string as its choice string, and the sender can associate a secret value $H(\mathbf{q}_j \oplus [C(r') \cdot s])$ with any string r'. As discussed above, the receiver can only compute $H(\mathbf{t}_j) = H(\mathbf{q}_j \oplus [C(r) \cdot s])$, the secret corresponding to its choice string r. The property of the PRC is such that, with overwhelming probability, all other values of $\mathbf{q}_j \oplus [C(r') \cdot s]$ (that a polytime player may ever ask) differ from $\mathbf{t}_j$ in a way that would require the receiver to guess at least $\mathbf{k}$ bits of s.

The above 1-out-of-∞ OT can be viewed as a kind of Oblivious Pseudo-Random Function (OPRF). Intuitively, $r \mapsto H(\mathbf{q} \oplus [C(r) \cdot s])$ is a function that the sender can evaluate on any input, whose outputs are pseudorandom, and which the receiver can evaluate only on its chosen input r.

The use of 1-out-of-∞ oblivious transfer (OT) as an oblivious pseudorandom function (OPRF) involves two main subtleties:

- The receiver obtains more information than just the output of the "PRF." Specifically, the receiver learns $t = q \oplus [C(r) \cdot s]$, rather than just $H(t)$.
- The protocol is designed to compute many instances of the "PRF" with related keys. Specifically, s and C are shared among all instances.

According to [KKRT16], this construction can be securely employed in place of the OPRF in the PSSZ protocol. This technique can also scale to support private intersections of sets, regardless of the number of elements, over a wide area network in less than 7 seconds with $n = 2^{20}$.

Although pairwise intersections can be computed iteratively to achieve set intersection of multiple sets, extending the above 2PC PSI protocol to the multi-party setting is not straightforward. Several challenges must be addressed, such as protecting the information on set intersection that one player gains during 2PC computation. [KMP+17] proposed an efficient extension of the above PSI protocol to the multi-party setting.

References

[BS20] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1257–1272, 2017.

[MGW87] Silvio Micali, Oded Goldreich, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM New York, NY, USA, 1987.

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

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 515–530, 2015.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SS84] Goldwasser Shafi and Micali Silvio. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[WOG88] Avi Wigderson, MB Or, and S Goldwasser. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, pages 1–10, 1988.

[Yak17] Sophia Yakoubov. A gentle introduction to yao's garbled circuits. *preprint on webpage at https://web. mit. edu/sonka89/www/papers/2017ygc. pdf*, 2017.