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

Bowen Cheng, Zhifeng Gu, Haoyu Zhao

April 29, 2024

In this lecture, we first introduce the concepts and definitions for secure computation. Specifically, we introduce semi-honest and malicious security adversary models. We then discuss how to construct protocols for secure MPC. Specifically, we make a detailed introduction to Yao's Protocol and GMW Protocol. Finally, we introduce custom protocols, with a special case of Private Set Intersection.

1 Secure Computation: Concepts & definitions

Secure multi-party computation, also referred to as secure computation, multi-party computation (MPC), or privacy-preserving computation, is a branch of cryptography that aims to develop techniques for multiple parties to collaboratively perform computations on their inputs while maintaining the confidentiality of those inputs. In contrast to conventional cryptography tasks, where cryptography ensures the security and integrity of communication or storage, and the adversary is outside the system and participants, MPC safeguards participant's privacy from each other.

Specifically, in an MPC scenario, a given number of parties $P_1, P_2, ..., P_n$ who do not trust each other, nor any other third-party, have their own private data $x_1, x_2, ..., x_n$, wish to compute a public function $(y_1, y_2, ..., y_n) = F(x_1, x_2, ..., x_n)$ without revealing their privacy. That is to say, each of the participant P_i should know nothing other than its private input x_i and the corresponding output of the function y_i.

1.1 Security Definition

Intuitively, the security definition in MPC should cover:

- Privacy: Each party should not learn anything other than their output
- Independence: Each party's input should be independent from each others' inputs
- Correctness: The output should be computed correctly
- Fairness: Each and every party should be delivered their output

Traditionally, security is established by providing a list of potential adversary actions that are considered breaching it. However, this can be challenging as it is difficult to ensure that the list covers all scenarios. Therefore, here we choose a general definition that first provides an ideal world and then proves that the real world is equivalent in terms of security to the ideal world [GM84], namely the *read-ideal paradigm*.

In **the Ideal World**, there is a trusted incorruptible third-party T who receives inputs from all participants and computes the function F. The inputs and outputs of all parties are transmitted in perfect confidence. Adversaries in the ideal world could still take control of any parties $P_i, P_j, ...$ other than T and make the attack, but it could be recognized since their choice inputs are independent of honest parties' inputs. In contrast, in **the Real World**, there does not exist any trustworthy party. Instead, parties must communicate through a prescribed protocol π, which defines each party P_i a function π_i to encode their next message. Messages are exchanged in a peer-to-peer (P2P) form as there is no third-party to relay messages. Adversaries in the real world could also corrupt parties.

The computation security in MPC is therefore defined as, for every real-world adversary A, there exists an ideal adversary A', s.t. joint distribution $(HonestOutput, AdvOutput)$ is indistinguishable. If the application proves to be secure in the ideal case, it can be inferred that it is also secure when a real protocol is implemented instead, as long as the protocol security is proven. To prove the security of a protocol under such definition, we only need to prove that the functionality of the real world is indistinguishable from the ideal world, so that the protocol is at least as safe as the ideal functionality.

1.2 Adversary Modeling

Unlike traditional cryptography like digital signature where we believe adversaries are outside the participants, in MPC scenarios, we must assume some of the participants are corrupted. These corrupted parties as adversaries may try to breach the security of the protocol. The solutions are different when we have the assumption that the number of corrupted parties t is smaller than half the number of all parties $n/2$, or $t \geq n/2$ may happen. The latter cases includes that one party in 2PC is adversary, or unlimited participants could be corrupted and collude with each other to attack honest parties.

Adversaries can be categorized by how much they hurt to the security of the protocol. There are essentially two types of adversaries in terms of behavior, and each can be modeled by a form of security, namely semi-honest and malicious adversary, to be detailed in the following subsections.

Security against different adversaries can also be categorized on various other assumptions. It could be against polynomial-time adversary computational power (i.e. based on computationally hard problem like factorization), or unbounded information-theoretic security (like relying on physical unavailability on channels).

1.2.1 Semi-honest Security

In scenarios described by semi-honest security, adversaries merely try to learn more about honest parties by inspecting communications, but still execute the protocol specification honesty. Multiple adversaries could also share their information eavesdropped from honest parties, and make common decisions on their inputs. Semi-honest security is commonly referred to as passive security, as adversaries are assumed not to make any attacks on other than eavesdropping.

In semi-honest models, honest parties who have their own input and message, and adversaries who view all parties they controlled execute the protocol honestly. Semi-honest security can be proven by (1) define the ideal functionality in the ideal world and (2) construct a simulator, and prove that the view of adversaries in the simulator is indistinguishable with the real world.

It is worth noting that semi-honest is rather a naive assumption, resulting in weak security in the real world. However, protocols secure against semi-honest adversaries can be useful in preventing unintended information leakage, and are often used as the first step towards higher security.

1.2.2 Malicious Security

In the malicious scenario, adversaries may arbitrarily deviate from the protocol, launching attack actively to cheat other parties. Other than eavesdropping messages in the semi-honest setting, a malicious adversary could take actions during the protocol execution in addition to analyzing the protocol. This includes the ability to control, manipulate and inject malicious message arbitrarily to other parties.

There are two important additions to consider in protocol design: (1) When the adversary deviate from the protocol, it may have the possibility of affecting the outputs delivery to honest parties, including manipulation or prevention on the result; (2) The input of adversary could be not well-defined.

Since the deviation of the protocol by malicious adversaries could affect the output of honest parties, protocol security under these circumstances cannot guarantee the correctness of the message honest parties receive. In this case, the protocol should be able to detect the manipulation of the message and force the communication to abort. The secure protocol against malicious attacks should also provide that as long as honest parties do obtain the message, it is proven to be correct. In both cases, honest parties' privacy is protected

To define a protocol dealing with the second addition, the simulator in the ideal world should select inputs for the corrupt parties, and the goal is to ensure that the effects achieved by these inputs in the ideal world are equivalent to those in the real world. The process is named **Extraction** from the real-world adversary input to construct ideal world input.

Protocols that achieve security against malicious adversaries provide a very high guarantee for MPC security. Generally, such security against active adversaries will lead to a reduction in efficiency. Compromising in the strict definition could achieve a balance in efficiency. Covert security [AL10] is designed to fit the setting that adversaries could cheat actively but only if they are not caught.

2 Yao's Protocol for Secure MPC

2.1 Oblivious Transfer (OT)

Before we start introducing Yao's protocol for secure MPC, we introduce a basic tool called oblivious transfer, which ensures that the receiver can securely select one thing from the sender.

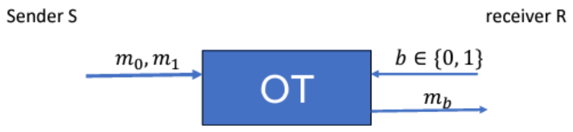


Figure 1: Oblivious Transfer

As shown in Figure 1, the standard definition of 1-out-of-2 OT involves two parties, a Sender S holding two secrets m_0, m_1, and a receiver R holding a choice bit $b \in 0, 1$. OT is a protocol allowing R to obtain m_b while learning nothing about the "other" secret m_{1-b}. At the same time, S does not learn anything at all. In other words, S doesn't know which m is choosen by R.

OT is quite useful in MPC, as OT is theoretically equivalent to MPC as shown in [Kil88]. Given OT, one can build MPC without any additional assumptions. And one can directly obtain OT from MPC vice versa.

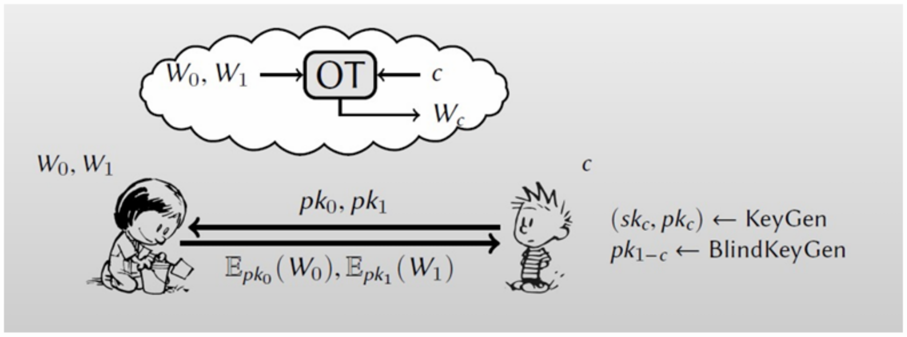

W_0, W_1
OT
c
W_c
W_0, W_1
c
pk_0, pk_1
$\mathbb{E}_{pk_0}(W_0), \mathbb{E}_{pk_1}(W_1)$
$(sk_c, pk_c) \leftarrow$ KeyGen
$pk_{1-c} \leftarrow$ BlindKeyGen

Figure 2: OT construction

So the problem is how to construct OT in practice. As shown in Figure 2, receiver R wants to select W_c from W_0, W_1. The OT step can be as follows:

- The receiver generates valid secret public key pair (sk_c, pk_c) and blind public key pk_{1-c} and sends pk_0 and pk_1 to the sender.
- Sender encrypts messages W_0 and W_1 with pk_0 and pk_1 to get $\mathbb{E}_{pk_0}(W_0)$ and $\mathbb{E}_{pk_1}(W_1)$ and sends the encryption to the receiver.
- Receiver receives $\mathbb{E}_{pk_0}(W_0)$ and $\mathbb{E}_{pk_1}(W_1)$ and tries to decrypt the ciphertext using private key sk_c. As the receiver only has sk_c, he will only get W_c but have no access to W_{1-c}.

The process needs a public-key encryption that supports blind key generation, *i.e.* the ability to sample a public key without knowledge of the secret key, like ElGamal algorithm.

A 1-out-of-2 OT is a cryptographic protocol securely implementing the function F^{OT} defined below:

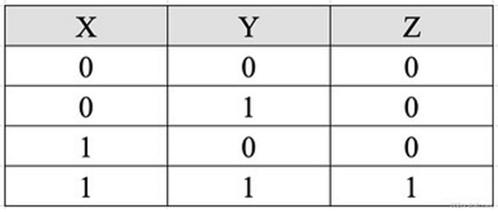
- Parameters: Two parties: Sender S and Receiver R. S has input secrets m_0, m_1 and R has a selection bit $b \in 0, 1$.
- Functionality F^{OT}: S sends m_0, m_1 to F^{OT}, and R sends b to F^{OT}. R receives m_b, and S receives nothing.

2.2 History of MPC

The idea of secure computation was introduced by Andrew Yao in the early 1980s [Yao82]. Secure computation was primarily of only theoretical interest for the next twenty years. In the early 2000s, algorithmic improvements and computing costs make it more realistic to build practical systems, *e.g.* Fairplay [MNPS04]. Since then the speed of MPC protocols has improved by more than five orders of magnitude.

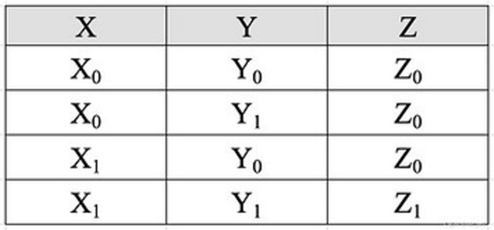
2.3 Yao's Garble Circuit

First, we discuss two-party computation. Every computation of function could be transferred to computing a Boolean circuit. Yao's protocol is a semi-honest secure (2-party) computation for Boolean circuits. Before we start, we focus on the semi-honest case. [GMW87] shows that if we can a construct semi-honest secure MPC for any circuit, we can construct a malicious secure MPC for any circuit. Here is a Yao's garble circuit for the two-party, boolean case. And we take the AND gate for example.



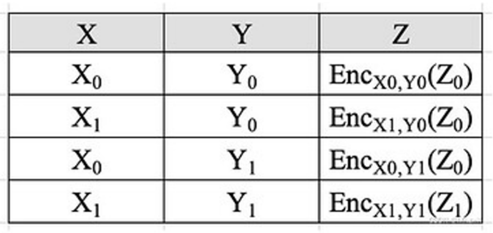
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3: AND gate truth table



X	Y	Z
X_0	Y_0	Z_0
X_0	Y_1	Z_0
X_1	Y_0	Z_0
X_1	Y_1	Z_1

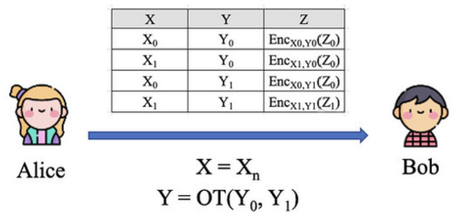
Figure 4: Random substitution



X	Y	Z
X_0	Y_0	$\mathrm{Enc}_{X0,Y0}(Z_0)$
X_1	Y_0	$\mathrm{Enc}_{X1,Y0}(Z_0)$
X_0	Y_1	$\mathrm{Enc}_{X0,Y1}(Z_0)$
X_1	Y_1	$\mathrm{Enc}_{X1,Y1}(Z_1)$

Figure 5: Encryption twice and shuffle

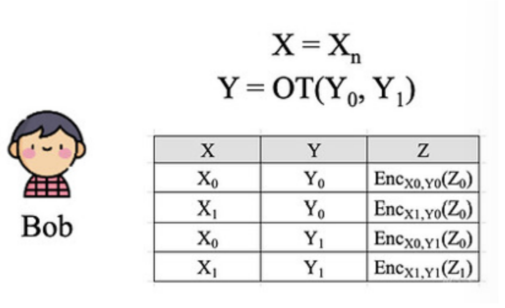
As shown in Figure 3, this is the truth table for a normal AND gate. Then the sender (Alice) substitutes each item with a random number as in Figure 4. Next, the sender encrypts Z twice, using X and y as secret keys as illustrated in Figure 5. And the sender shuffles the encrypted table and sends the last encrypted column to the receiver (Bob).



X	Y	Z
X_0	Y_0	$\mathrm{Enc}_{X0,Y0}(Z_0)$
X_1	Y_0	$\mathrm{Enc}_{X1,Y0}(Z_0)$
X_0	Y_1	$\mathrm{Enc}_{X0,Y1}(Z_0)$
X_1	Y_1	$\mathrm{Enc}_{X1,Y1}(Z_1)$

Figure 6: Communications between two parties

Then, as illustrated in Figure 6, Alice sends a random number X_0/X_1 according to her input $X = 0/1$, Bob gets random number Y_0/Y_1 according to his input $Y = 0/1$ via oblivious transfer.



X	Y	Z
X_0	Y_0	$\mathrm{Enc}_{X0,Y0}(Z_0)$
X_1	Y_0	$\mathrm{Enc}_{X1,Y0}(Z_0)$
X_0	Y_1	$\mathrm{Enc}_{X0,Y1}(Z_0)$
X_1	Y_1	$\mathrm{Enc}_{X1,Y1}(Z_1)$

Figure 7: Bob calculates the garbled circuit

As shown in Figure 7, we assume $X = 0, Y = -1$, then Bob will have X_0 and Y_1. Bob tries to decrypt Z in every row, only the 3rd row can be decrypted successfully, and Bob gets Z_0. But Bob doesn't know whether Z_0 represents 0 or 1.

At last, as shown in Figure 8, Alice and Bob share the computation result. As Alice knows the true value of Z_0, she recovers the true value and shares it with Bob.

The simple AND gate garbled circuit can be generalized to garbled general circuit framework to solve much more complex computation problems as Figure 9. The process of garbling a circuit:

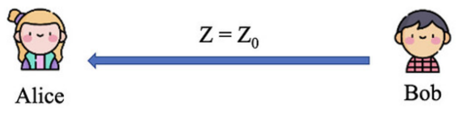


Figure 8: Alice and Bob share computation result

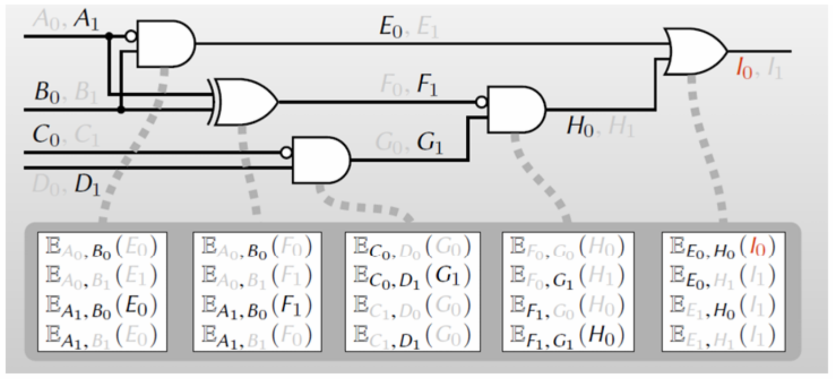

A0, A1
B0, B1
C0, C1
D0, D1
E0, E1
F0, F1
G0, G1
H0, H1
I0, I1
$\mathbb{E}_{A_0,B_0}(E_0)$
$\mathbb{E}_{A_0,B_1}(E_1)$
$\mathbb{E}_{A_1,B_0}(E_0)$
$\mathbb{E}_{A_1,B_1}(E_0)$
$\mathbb{E}_{A_0,B_0}(F_0)$
$\mathbb{E}_{A_0,B_1}(F_1)$
$\mathbb{E}_{A_1,B_0}(F_1)$
$\mathbb{E}_{A_1,B_1}(F_0)$
$\mathbb{E}_{C_0,D_0}(G_0)$
$\mathbb{E}_{C_0,D_1}(G_1)$
$\mathbb{E}_{C_1,D_0}(G_0)$
$\mathbb{E}_{C_1,D_1}(G_0)$
$\mathbb{E}_{F_0,G_0}(H_0)$
$\mathbb{E}_{F_0,G_1}(H_1)$
$\mathbb{E}_{F_1,G_0}(H_0)$
$\mathbb{E}_{F_1,G_1}(H_0)$
$\mathbb{E}_{E_0,H_0}(I_0)$
$\mathbb{E}_{E_0,H_1}(I_1)$
$\mathbb{E}_{E_1,H_0}(I_1)$
$\mathbb{E}_{E_1,H_1}(I_1)$

Figure 9: Garbled general circuit framework

- Pick random labels W_0; W_1 on each wire
- "Encrypt" truth table of each gate
- Garbled circuit all encrypted gated
- Garbled encoding one label per wire

The process of garbled evaluation:

- Only one ciphertext per gate is decrypted
- Result of decryption is the value on the outgoing wire

3 Goldreich-Micali-Wigderson (GMW) Protocol

Unlike Yao's protocol, the GMW protocol [GMW87] can extend to multi-party (more than two parties) scenarios. The GMW protocol employs a boolean-circuit representation to compute functions and offers security against semi-honest adversaries, even when multiple parties are corrupted. The fundamental semi-honest protocol utilizes Oblivious Transfer (OT) to execute boolean gates effectively. Furthermore, this protocol can also be extended to accommodate arithmetic circuits. The term GMW is often used for a protocol that combines the two techniques into an n-party multi-party computation protocol which is actively secure.

3.1 Extending from OT_1^2 to OT_1^n

Consider Alice holding $x_1, \dots, x_n \in \{0,1\}^\ell$ and Bob holding $i \in \{1, \dots, n\}$. Through OT_1^n, Bob obtains x_i while Alice knows nothing about i. OT_1^n can be constructed through the following steps.

1. Alice generates $k_0 = 0^\ell$ and randomly generates $k_j \in \{0,1\}^\ell, j = 1, \dots, n$.
2. Alice and Bob perform OT_1^2 operation n times. At jth operation, Alice provides $k_0 \oplus \cdots \oplus k_{j-1} \oplus x_j$ and k_j. If $j = i$, Bob chooses the former. If $j \neq i$, Bob chooses the latter.
3. Bob obtains $\{k_0, \dots, k_{i-1}\}$ from the 1st to $(i-1)$th OT_1^2 and $k_0 \oplus \cdots \oplus k_{i-1} \oplus x_i$ from the ith OT_1^2 to decrypt x_i.

Let us analyze the security of the above OT_1^n. For Bob, his security can be directly inherited from OT_1^2, namely OT_1^2 ensures that Alice does not know whether Bob chooses the former or the latter in each round, so Alice does not know the true value of i. For Alice, we consider that Bob chose the former in ith round. If $j < i$, Bob chooses k_j from OT_1^2, not obtaining any information of x_j. If $j > i$, Bob at most can obtain $x_j \oplus k_i \oplus$ others. Not knowing Alice randomly generating k_i, Bob does not know x_j.

3.2 Securely Computing Any Constant Size Function using OT_1^n

OT_1^n can address a very common problem: securely computing any constant size function. Constant size function refers to functions whose input values are limited.
Consider Alice and Bob owning private input $x \in S_A, y \in S_B$ respectively. They want to compute $f(x, y)$.

1. Alice computes x with all possible values of y to obtain $A = \{f(x, y_1), f(x, y_2), \dots, f(x, y_n)\}$.
2. Alice performs OT_1^n with Bob. Alice provides A and Bob provides $y = y_i$ according to i.
3. Bob obtains $f(x, y_i) = f(x, y)$ and shares the result to Alice if necessary.

OT_1^n ensures that Bob only obtains the final result $f(x, y_i)$ and get nothing more about x. At the same time, OT_1^n ensures that Alice cannot know i from Bob.

3.3 The GMW Protocol

Let's consider the basic case. Alice and Bob own one bit of input $x, y \in \{0,1\}$ and they want to compute $w = G(x, y)$, where G is a logic gate circuit. The GMW protocol contains the following steps.

1. Alice randomly generates $x_a \in \{0,1\}$ and sends $x_b = x \oplus x_a$ to Bob. Due to the randomness of x_a, Bob cannot decrypt x.

2. Bob randomly generates $y_b \in \{0,1\}$ and sends $y_a = y \oplus y_b$ to Alice. Due to the randomness of y_b, Alice cannot decrypt y.

3. Alice randomly generates $z_a \in \{0,1\}$ and enumerates all the possible values of $f(x_b, y_b) = z_a \oplus G(x_a \oplus x_b, y_a \oplus y_b)$. Due to $x_b, y_b \in \{0,1\}$, $f(x_b, y_b)$ has totally four possible values, namely, $\{f(0,0), f(0,1), f(1,0), f(1,1)\}$.

4. Alice and Bob perform OT_1^4. Alice provides $\{f(0,0), f(0,1), f(1,0), f(1,1)\}$ and Bob provides i corresponding to (x_b, y_b). OT_1^4 ensures that Bob only obtains the final result $f(x_b, y_b)$ and Alice does not know i so that she knows nothing about $x_b, y_b, f(x_b, y_b)$.

5. Bob gets $z_b = f(x_b, y_b)$.

6. Alice and Bob can reveal z_a, z_b to get the true value of z. $z_a \oplus z_b = z_a \oplus z_a \oplus G(x_a \oplus x_b, y_a \oplus y_b) = G(x_a \oplus x_b, y_a \oplus y_b) = G(x, y) = z$

For more complex logic circuits, Alice and Bob will calculate each logic gate sequentially using the above method to obtain the shared value of w_a, w_b of each line w. The GMW protocol guarantees that $w = w_a \oplus w_b$. At last, Alice and Bob only reveal the shared value of the output line of the circuit to obtain the result of the logic circuit. Alice and Bob do not know the true value of the intermediate circuit because each side only has half of the shared value.

Please refer to [GMW87] for more details.

4 Custom Protocols

Custom protocols refer to protocols that are specifically designed and developed to address specific requirements or solve particular problems. These protocols are tailored to meet the unique needs of a specific application, system, or network environment, rather than relying on existing generic protocols.

When designing a custom protocol, the protocol designer has the flexibility to define the communication patterns, message formats, cryptographic techniques, and security assumptions that best suit the intended use case. This allows for a more fine-grained control over the protocol's behavior and performance characteristics.

Custom protocols can be developed for various purposes, such as secure communication, data exchange, authentication, privacy preservation, or any other specific requirement. They may utilize a combination of cryptographic primitives, algorithms, and techniques to achieve their intended goals.

4.1 Private Set Intersection (PSI)

Private Set Intersection (PSI) is a cryptographic protocol that allows two or more parties to compute the intersection of their respective private sets without revealing any information about the elements in their sets to each other.

To illustrate Private Set Intersection (PSI), we consider a simplified scenario as depicted in Figure 10. In this scenario, Bob wishes to determine whether Alice possesses a specific

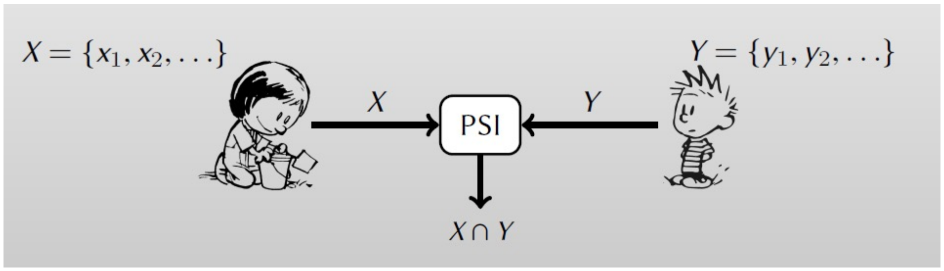


Figure 10: Example of Private Set Intersection (PSI)

number *yi* within a set X. To accomplish this, they employ PSI by exchanging holding sets and computing the intersection set, ultimately solving Bob's query. While one possible method involves utilizing a hash function to calculate key values and transmitting them to PSI, this approach may be deemed insecure, particularly when dealing with numbers with limited entropy, such as phone numbers. Alternatively, a promising approach involves employing the Diffie-Hellman algorithm. However, this method may introduce some time overhead due to its computational requirements.

Please refer to [PSZ18] for more details.

References

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 1987.

[Kil88] J Kilian. Founding cryptography on oblivious transfer. *Acm Stoc*, 1988.

[MNPS04] D. Malkhi, N. Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. 2004.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

[Yao82] Andrew C. Yao. Protocols for secure computation. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, 1982.