CSA E0 235: Cryptography	16 Sept 2019
Scribe for Lecture 9	
Instructor: Arpita Patra	*Submitted by: Nikita Yadav*

In this lecture, we studied multi-party computation (MPC) problem, it's applications and a fundamental approach to solve MPC. We studied in detail the Yao's protocol for 2-party computation (2PC).

1 Yao's Millionaires' Problem

Yao's Millionaires' Problem is a **2-party computation** problem which was introduced in 1982 by computer scientist and computational theorist Andrew Yao. The problem discusses two millionaires, A and B, who are interested in finding out who is richer among them without disclosing their individual assets. Formally, A has wealth a, B has wealth b, and they wish to compute a $\geq$ b without revealing the values a or b. This is an important problem in Cryptography, the solution of which is used in many commercial applications.

2 Multi-party Computation

2.1 Definition

In MPC, there are mutually distrusting parties P_1, P_2, ..., P_n and each has private data, respectively, d_1, d_2, ..., d_n. Some of the parties may be corrupted. They want to compute the value of a common function on their private data: $F(d_1, d_2, ..., d_n)$ but without disclosing their own data to each other. One way to solve this problem is to introduce a trusted third party (TTP), which will take the input from all the parties and compute the function F and then returns only the output $F(d_1, d_2, ..., d_n)$ to the parties. But this solution does not work in real world because of non-existence of TTP. The goal of MPC is to design a protocol, where, by exchanging messages only among themselves the parties can securely compute function F on their inputs. The most basic properties that a MPC protocol aims to ensure are -

1. **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. Either the honest parties are guaranteed to compute the correct output or they abort if they find an error.

2. **Privacy:** No information beyond the function output is leaked by the protocol.

In this class, we consider the problem of achieving security in the presence of semi-honest (or passive) parties who follow the protocol specifications, but attempt to learn additional information by analysing the transcript of messages received during the execution.

2.2 Applications of secure MPC

Secure MPC allows cryptographically secure data analysis over sensitive data. Bringing in significant social benefit in contexts where data sharing is constrained or prevented by legal, ethical, or privacy restrictions. Following are few examples of such applications.

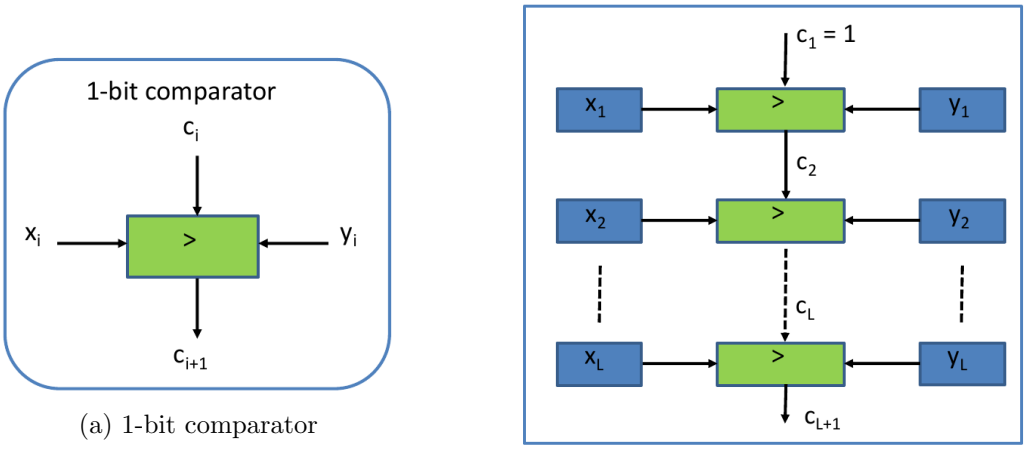
1. **Secure data analysis** over sensitive salary data of 10 million people in Boston to calculate pay disparity.
2. **Train model on private medical data** held by several sources to offer the best treatment for diseases.
3. **Compute collision probability of two satellites** in the space owned by competing countries who do not wish to disclose the orbit details of their satellites.
4. **Implement secure auction** to find a fair price for suger-beet in Denmark.
5. **Implement online sexual assault reporting platform** (allegation escrow) that can detect repeat perpetrators and create pathways to support the victims.
6. **Secure intersection** of flyer's list of a fligt and the (inter)national register of no-flyer's list (homeland security).
7. **Detect hate speech** (personal text classification).

3 A Fundamental Approach to MPC

Let f be a polynomial-time function. Every efficiently computable function can be represented by a boolean/airthmetic circuit C whose input wires will take the inputs of the function f and output wires give the output of funcion f. Such a boolean circuit can be constructed from any functional complete set of logic gates such as AND, OR, NOT. MPC is then done via secure circuit evaluation. Computation complexity depends on $|C|$.

3.1 Circuit Abstraction Example: $\geq$

Consider the function f which takes two L-bit non-negative integers X and Y as input and outputs 1 if X is greater than equal to Y else 0. We will construct a circuit for computing f. We can construct a 1-bit comparator (see figure 1 (a)).
Given two bits x_i, y_i and carry bit c_i, the green colored box compute which bit is greater and output c_{i+1}, where $c_{i+1} = 1 \iff (x_i > y_i)$ **or** $([x_i = y_i]\ AND\ [c_i = 1])$. c_{i+1} can be expressed in terms of digital logic gates as $c_{i+1} = x_i \oplus [(x_i \oplus c_i)(y_i \oplus c_i)]$. Using this 1-bit comparator we can construct a L-bit comparator (see figure 1(b)). c_{L+1} is the output of the L-bit comparator and $c_{L+1} = 1 \iff X \geq Y$.



(a) 1-bit comparator

(b) L-bit comparator

Figure 1: Circuit abstraction example: $\geq$

4 Yao's Protocol for 2-Party Computation

Yao presented a constant-round protocol for securely computing any two-party functionality in the presence of semi-honest adversaries. Let f be a polynomial-time function and let x and y are the parties' respective inputs. The first step is to view the function f as a boolean circuit C. The circuit $C(x,y)$ is computed gate-by-gate, from the input wires to the output wires. There are four types of wires in a circuit: circuit-input wires (that receive the input values x and y), circuit-output wires (that carry the values $C(x,y)$, gate-input wires (that enter some gate g) and gate output-wires (that leave some gate g). The values that are allocated to all the wires that are not circuit-output should not be learnt by any party (these values may reveal information about the other party's input that could not be otherwise learned from the output).
The basic idea behind Yao's protocol is to provide a method of computing a circuit so that values obtained on all wires other than circuit-output wires are never revealed. The following subsections describe the building blocks for Yao's 2PC protocol.

4.1 Circuit Garbling

Garbling is the process by which boolean circuit is encrypted. As shown in Figure 2, the initial boolean circuit representing function f is passed to a Garbler Gb which produces three functions - encoding function En and encoding information e, decoding function De and decoding information d, garbled circuit evaluation function Ev and garbled circuit C. A Garbling scheme is represented as a tuple (Gb, En, Ev, De).
As shown in Figure 3, in Yao's protocol, one of the parties, henceforth the sender, constructs a garbled circuit C and sends it to the other party, henceforth the receiver. The sender and receiver then interact so that the receiver obtains the encoded inputs $En(x,e)$ and $En(y,e)$. Given these inputs, the receiver then computes the circuit, obtains the output, use decoder De and decoding information d to get actual output and concludes the protocol. The

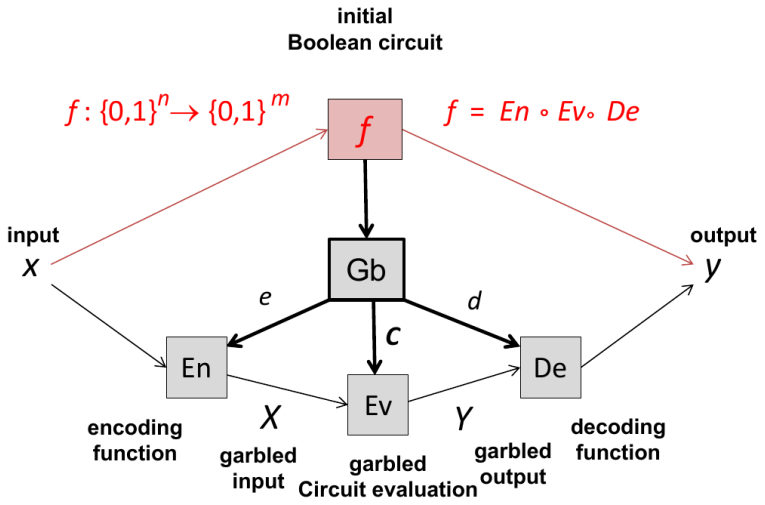


Figure 2: Circuit garbling

receiver's output can include the sender's output in encrypted form (where only the sender know the decryption key). Then, the receiver can forward the sender its output at the end of the computation. The receiver learns nothing more than its own output.

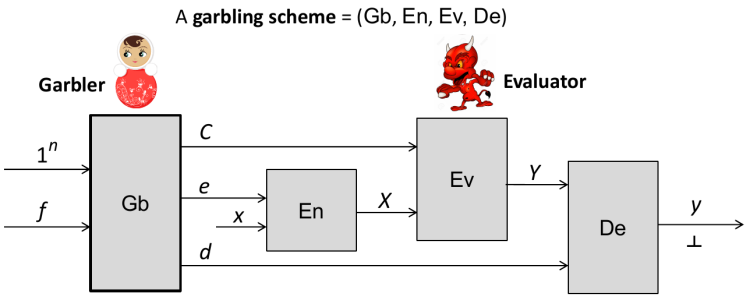


Figure 3: Yao's protocol overview

The protocol ensure following properties:

1. **Privacy:** It ensures that original input remains private to the parties and only encoded input is revealed which does not leak any information about original input.

2. **Obliviousness:** The receiver cannot infer anything from the output of the Evaluation unless the sender send the decoding information d. It gives output privacy when d is withheld.

3. **Authenticity:** It provides unforgeability of the output of Ev.

4.2 Construction and Evaluation of Garbled Circuit

As shown in figure 4, for every wire in the circuit, two random keys are specified such that one key represents 0 and the other represents 1. For example, let a be the label of some wire. Then, two keys k_a^0 and k_a^1 are chosen, where k_a^i represent the bit i. An important observation here is that even if one the parties know the key k_a^i obtained by wire a, this does not help it to determine if i = 0 or i = 1 (because both k_a^0 and k_a^1 are identically distributed).

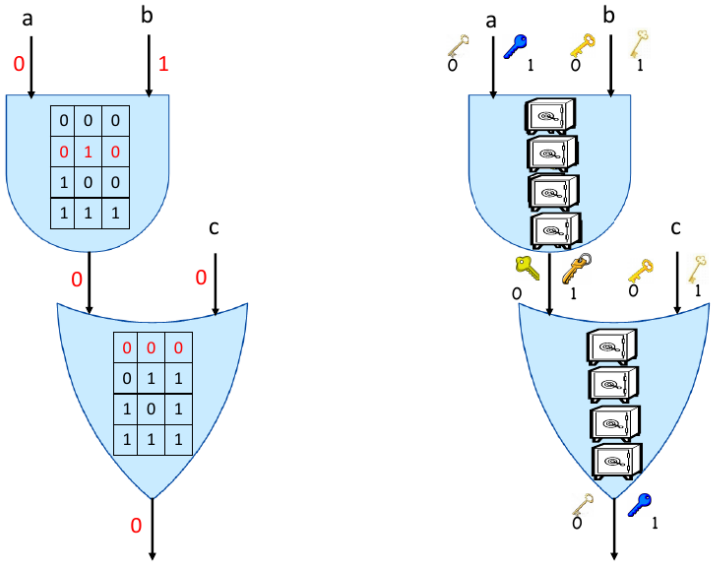


Figure 4: Construction of garbled circuit C

Let us understand how to construct a single garbled gate. Let g be a OR gate with incoming wires w_1 and w_2, and output wire w_3. Then, given two random values k_1^i and k_2^j, we compute the value of output wire (also a random key K_3^0 or k_3^1). For this computation, we need to use garbled truth table for gate g that maps random input values to random output values. Figure 5 shows garbled OR gate truth table. This mapping should have the property that given two input keys, it is only possible to learn the output key that corresponds to the output of the gate (the other output key must be kept secret).

input wire w_1	**input wire w_2**	**output wire w_3**	**garbled computation table**
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	k_3^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

Figure 5: Garbled OR gate

For each logic gate we have 4 doubly-locked boxes (see Figure 4), where each box is associated with a row in the truth table. The four boxes are locked with the input keys and

contains the output key. For example, in case of OR gate g, box 1 is locked with keys k_1^0 and k_2^0 and contains k_3^0. Given a set of keys corresponding to the input wires, it is possible to open only the box corresponding to the actual input values and only one box can be opened with a pair of keys. Also, this does not reveal any information about the input and output since the keys are not labelled. The boxes must be randomly ordered so that a box's position does not reveal anything about the value it is associated with. Figure 6 shows an example of garbled circuit evaluation.

1. k_a^0, k_b^1 and k_c^0 are input to the garbled circuit on the respective wires a.b and c.
2. In AND gate, k_a^0 and k_b^1 are input on wires a and b, and only one of the locked boxes can be opened with this pair of keys which corresponds to original input (0,1). The output key obtained from unlocked box, say k_w^0 corresponds to value 0. This is one of the inputs to the next OR gate.
3. k_c^0 is other input for OR gate on wire c.
4. Again OR gate has four boxes and only one of the boxes can be opened with k_w^0 and k_c^0 keys. The unlocked box gives key, say k_d^0 which corresponds to value 0.
5. The output key k_d^0 is compared with the two possible keys on that output wire to learn the semantic of the output.

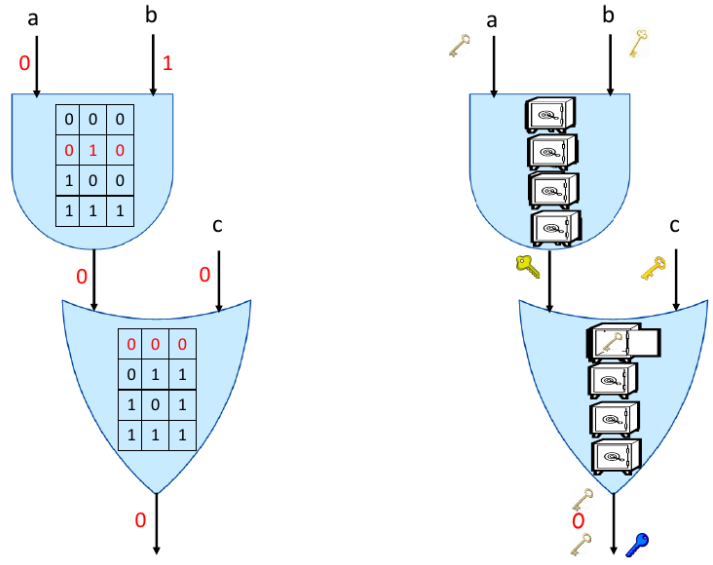


Figure 6: Evaluation of garbled circuit

A garbled circuit consists of garbled gates along with the output decryption tables. These tables map the random values on the circuit-output wires back to their corresponding real values. That is, for a circuit-output wire, the pair (0, k_w^0) and (1, k_w^1) are provided. Given the keys associated with input x and y, the entire circuit is computed gate-by-gate. Then, after obtaining result on circuit-output wire, it is decrypted by using output

decryption tables to get the result *C(x,y)*. In actual garbled circuit construction, double encryption replaces doubly-locked boxes and decryption keys replace physical keys. This gives us 4 ciphertexts per logic gate as shown in Figure 5.

4.3 SKE with Special Correctness

As discussed in previous section, in every gate, the receiver is given two random keys that enable it to decrypt and obtain the random key for the gate-output wire (refer Figure 5). A problem that arises here is how can the receiver know which value is the intended decryption. Notice that it may be the case that all ciphertexts can be decrypted. To solve this problem we use SKE with special correctness. A SKE (G,E,D) has special correctness if for two distinct keys (k_1, k_2), encryption under k_1 will result in $\perp$ (error) when decrypted under k_2 with high probability. Mathematically,

$$Pr[D_{k_2}(E_{k_1}) \neq \perp] \leq \varepsilon(n) \ \forall m \tag{1}$$

where n is security parameter and m is message.

Let us construct an encryption scheme π = *(Gen, Enc, Dec)* defined over *(*$K = \{0,1\}^n$, $M = \{0,1\}^n$, $C = \{0,1\}^{3n}$*)* with above property. Let $F = f_k$ be a family of pseudorandom functions, where f_k: $\{0,1\}^n \rightarrow \{0,1\}^{2n}$ for $k \in \{0,1\}^n$.

1. **Gen:** It takes as input security parameter n and outputs a random value $k \in \{0,1\}^n$. This is the secret key used to choose f_k.

2. **Enc:** It takes 2 inputs - message m and key k. It chooses a random number r in $\{0,1\}^n$ and concatenates the message m with a string 0^n to obtain $m0^n$. It then outputs ciphertext C = (r, $m0^n \oplus f_k(r)$).

3. **Dec:** It takes ciphertext $C = (c_1, c_2)$ and k as inputs and computes m = $c_2 \oplus f_k(c_1)$. If the last n bits of $f_k(c_1)$ are equal to the last n bits of c_2 then it outputs the message else it outputs $\perp$.

Reading Assignment: Prove that scheme π is CPA-secure.

4.4 What security from SKE is needed?

Our scheme must be able to tolerate a bad evaluator. A bad evaluator should be able to open only one box with a given pair of keys and should not have any information about the three unopened boxes. This is important and let us consider an example to understand why. Consider an AND gate, if it can guess that the unopened messages hidden in the three ciphertext are same then it knows that the key it decrypted corresponds to 1. This is enough to break the security of garbling scheme. Thus even one bit of information should not be leaked. To prove the security of the entire protocol, we have to prove the security of the garbling scheme. To secure our garbling scheme, we have to come up with a secure SKE which we will discuss in next section.

4.4.1 Chosen Double Encryption (CDE) security

To define the security of the scheme we bring in the game as shown in Figure 7. Four keys are involved in encryption, 2 of these belongs to the Adversary and 2 are not known to Adversary. We have to incorporate this in our game. We allow adversary to pick 2 keys - k_0 and k_1. The challenger picks 2 keys - k'_0 and k'_1. Now, like a CPA game, the game starts with a *training phase* in which the adversary is given *oracle access* to the double encryption scheme. The adversary can choose the message and its key on which it want encryption, this is shown in Figure 7 with ** symbol. In *challenge phase*, adversary sends 2 messages- (x_0, y_0, z_0), (x_1, y_1, z_1) which corresponds to the three unopened boxes. The challenger pick random bit b and encrypt (x_b, y_b, z_b) in following manner - $c_0 \leftarrow Enc_{k0}(Enc_{k'1}(x_b))$, $c_1 \leftarrow Enc_{k'0}(Enc_{k1}(y_b))$, $c_2 \leftarrow Enc_{k'0}(Enc_{k'1}(z_b))$. In all the ciphertexts at least one key is not known to the adversary.Then it gives c_0, c_1, c_2 to adversary. The adversary can ask for *post-challenge oracle training*. This scheme is called ***chosen double encryption (CDE)*** scheme. 'Double' because there are 2 layers of encryption. 'Chosen' because we give adversary pre-challenge and post-challenge oracle training. The adversary has to guess which of the triples is encrypted and respond with bit b'. The output of the game is 1 if the adversary correctly guesses the bit i.e. b'=b. The scheme is CDE secure if for every probabilistic polynomial time adversary A, there is a negligible function negl, such that:

$$Pr(Priv^{cde}_{A,\pi}(n) = 1) \leq 1/2 + negl \tag{2}$$

Reading Assignment: If a scheme π is CPA-secure then it is also CDE-secure.

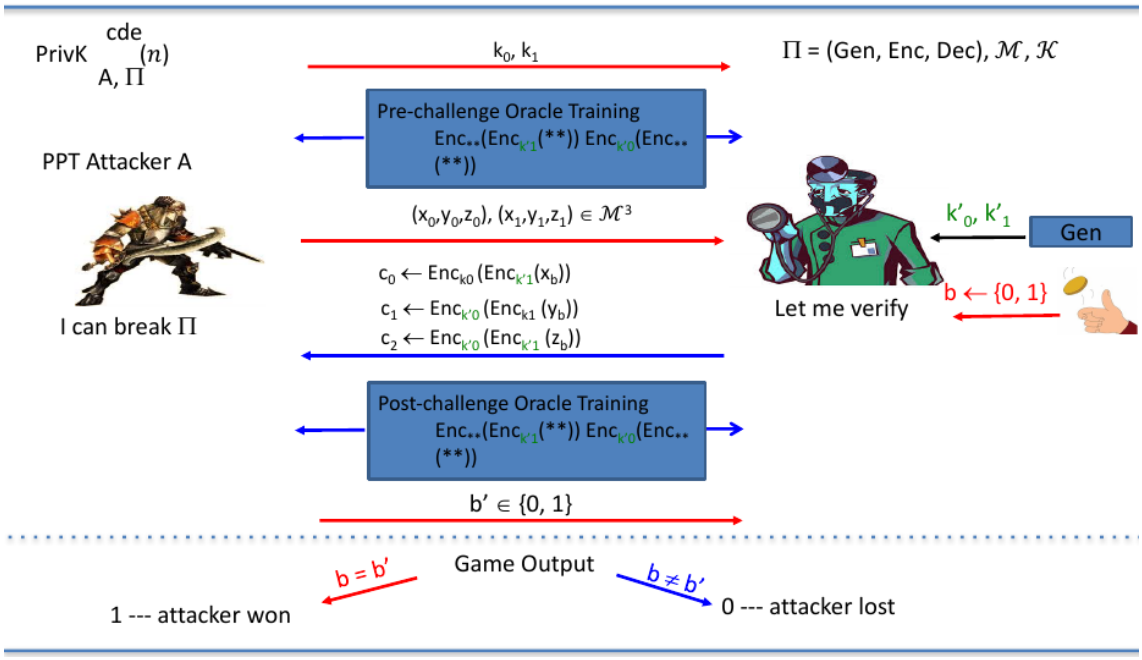


Figure 7: Chosen double encryption (CDE) indistinguishability experiment

4.5 Oblivious Transfer

Oblivious Transfer is a protocol in which sender transfers one of the potentially many pieces of information to the receiver, but remains oblivious as to what piece was transferred and the receiver receives only one of the pieces and does not get any information about other pieces. Figure 8 shows 1-out-of-2 oblivious tranfer. The sender S has two messages m_1 and m_2, the receiver R has a bit r and wishes to receive m_r. R does not want S to learn the bit r and S does not want R to learn m_{1-r}. They uses 1-2 OT, which takes input m_1 and m_2 from sender S and bit r from receiver R, then it gives m_r to R.

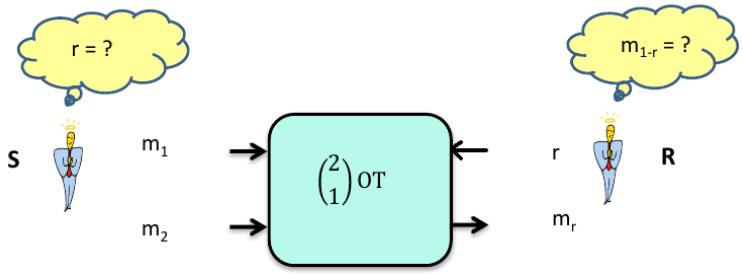


Figure 8: Oblivious transfer protocol between 2 parties S and R

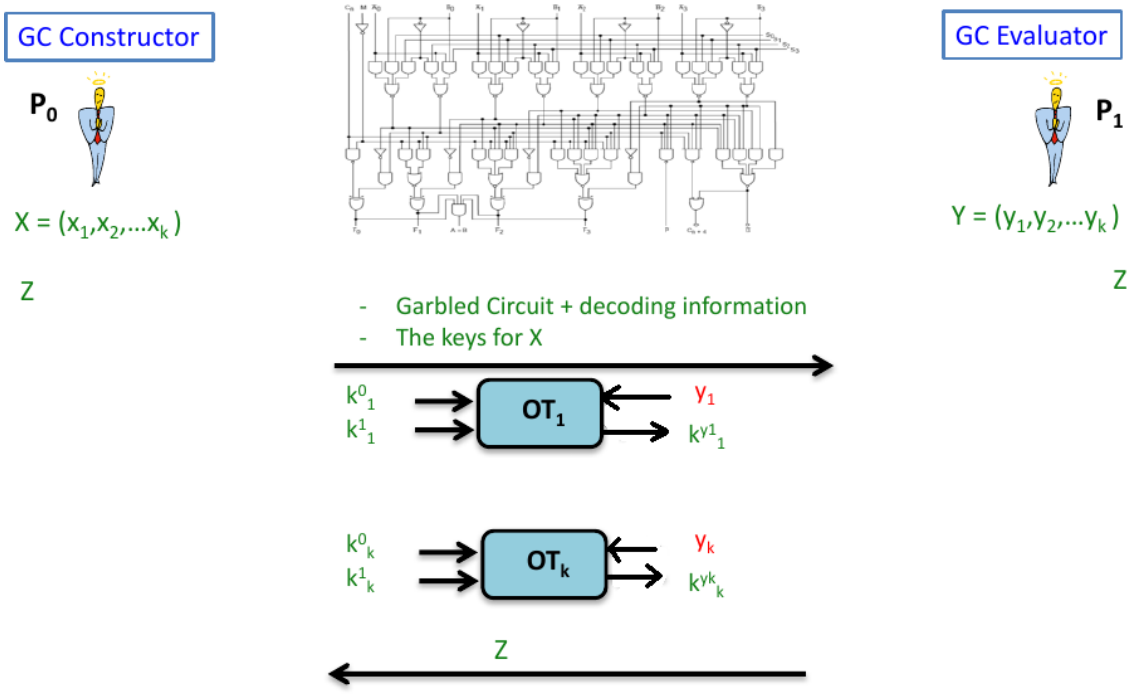


Figure 9: Yao's 2-party computation protocol

4.6 Yao's 2PC protocol: Putting is all together

Refer to figure 9 for the complete picture of Yao's 2PC protocol. We learnt in sections 4.1 and 4.2 how a garbled circuit is constructed and evaluated. The GC constructor, here P_0 sends Garbled circuit and deconding information (for output) to GC evaluator, here P_1. Garbler gives its encoded input X (in form of keys) to the evaluator. We have not explained until now that how does the evaluator obtains the keys for its own input. Garbler cannot send all of the keys pertaining to evaluator's input (i.e. both 0 and 1 keys on the evaluator's input wires), because this would enable the evaluator to compute C(x,y') for all y'. This is much more information than a single C(x,y). Also, the evaluator cannot openly tell the garbler which keys to send it, this will leak the evaluator's input. The solution to this is 1-out-of-2 oblivious transfer protocol as shown in figure 9. The 1-2 OT is used k times to get keys for every bit of y = (y_1, y_2, .., y_k). The evaluator then evaluates the garbled circuit on the garbled inputs and uses output decoding information to get output. The output is sent to the garbler.

5 References

[1] Arpita Patra, `https://www.csa.iisc.ac.in/~cris/e0_235.html`
[2] A Proof of Security of Yaos Protocol for Two-Party Computation, by Yehuda Lindell and Benny Pinkas, 2006.