CSA E0 312: Secure Computation	September 18, 2015
[Lecture 11-12]	
Instructor: Arpita Patra	*Submitted by: Dheeraj Ram*

In this lecture, we are going to see the multi-party extension of GMW protocol, and couple of optimizations on the same. We then explore Yao's protocol in detail.

1 n-Party Evaluation of GMW protocol

We have seen GMW protocol for 2 parties, in the previous lecture. We will be discussing how to extend that protocol to any arbitrary n players with (n, n) additive secret sharing.

Sharing inputs of all players

For sharing the inputs, we use (n, n) secret sharing on each player's inputs. So, besides the bearer of an input, no other party(s) will get to know the input of any other player, but they collectively hold all the inputs in the circuit.

1.1 Logic Gate Evaluation of GMW n-Party Protocol

XOR gates and NOT gates are easy to evaluate in the n-Party GMW Protocol. If each party XOR their shares of a and b, they will get the shares of $a \oplus b$. Similarly if just one party (say, always P_1) complements it's share of a, all parties hold shares of $\neg a$. We have seen how it works in the case with 2 parties. But it's not obvious how to implement AND Gate with (n, n) shares.

1.1.1 AND Gate Evaluation in GMW

Let x and y be two input/intermediate values in GMW logic circuit. We are about to see how to perform $x \cdot y$. The output is the (n, n) shares of $x \cdot y$, from the shares of x and y.

Let $x = (x_1 + x_2 + \cdots + x_n)$ and $y = (y_1 + y_2 + \cdots + y_n)$ where x_i and y_i are the secret shares of x and y respectively, held by player P_i. That means,

$$
\begin{aligned}
x \cdot y &= ((x_1 + x_2 + \cdots + x_n) \cdot (y_1 + y_2 + \cdots + y_n) \\
&= (x_1 \cdot y_1 + x_1 \cdot y_2 + \cdots + x_1 \cdot y_n) + (x_2 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_2 \cdot y_n) + \ldots \\
&\qquad + (x_n \cdot y_1 + x_n \cdot y_2 + \cdots + x_n \cdot y_n) \\
&= (x_1 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_n \cdot y_n) + \Sigma_{i \neq j}(x_i \cdot y_j) \\
&= \Sigma_i(x_i \cdot y_i) + \Sigma_{i \neq j}(x_i \cdot y_j)
\end{aligned}
$$

We can see that every party can compute one summand $(x_i \cdot y_i)$ by itself. It is also obvious that a summand of the form $(x_i \cdot y_j)$ should not be computed by a single party, as partial knowledge on the product compromise the secrecy of the other share [i]. The solution is to locally compute first n summands, and the remaining (n^2-n) summands of the form $(x_i \cdot y_j)$ should be $(2,2)$ shared between the players P_i and P_j. That way, if adversary corrupts just one party among P_i and P_j, he can't gain any information about the secret share of the other party from the secret sharing property, and if adversary corrupts both, he gains no new information other than the shares of the parties he corrupts.

The $(2,2)$ secret sharing of $(x_i \cdot y_j)$ can be done with OT, as in the figure 1:

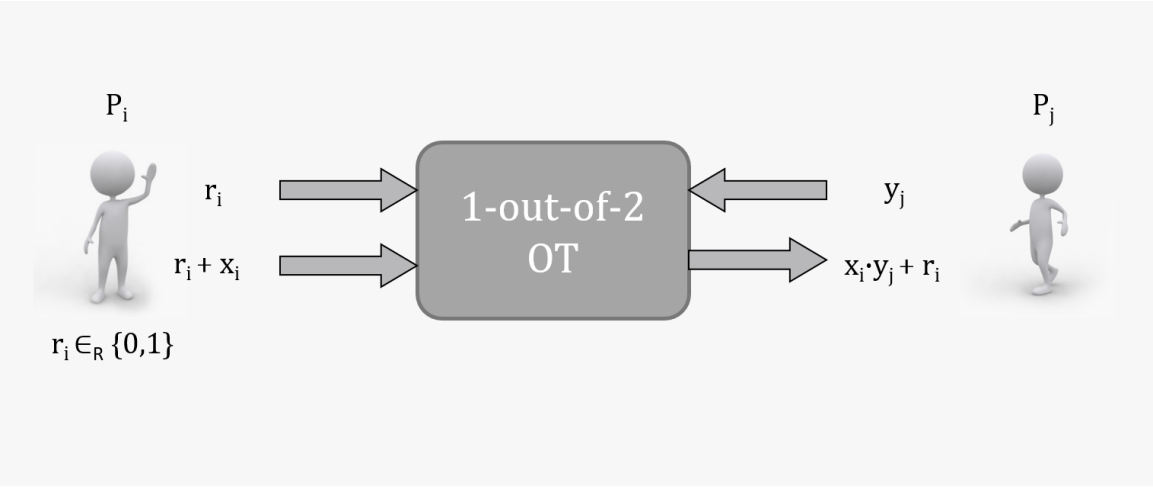

Figure 1: $(2,2)$ sharing of $x_i \cdot y_j$

The shares of $(x_i \cdot y_j)$ will be r_i from P_i and $r_i + x_i \cdot y_j$ from P_j. The correctness is trivial as the addition of the two shares will produce $x_i \cdot y_j$. Along with that, P_j gains no information about x_i from the OT execution[ii]. It is obvious that for any two pair of players (P_i,P_j), there will be two cross-summands such as $(x_i \cdot y_j)$ and $(x_j \cdot y_i)$. So we need to execute $2(n^2-n)$ OTs in total for a single AND operation in GMW with n parties. In effect, the share of the product $x \cdot y$, from a party P_i is the sum of following values:

- $x_i \cdot y_i$
- r_i from all OTs where P_i acted as a sender
- output of the OTs where P_i acted as a receiver

[i] For example, if $x_i \cdot y_j$ is computed by player P_i, then he will see both x_i and $x_i \cdot y_j$. If x_i happened to be one, then the summand will be same as y_j, which is a security breach.

[ii] It follows directly from the security of Sender of OT

This setting will provide security for honest party(s), even if there are $n-1$ corrupted parties present in the protocol (i.e. threshold $t = n-1$). In figure 2 we can see that the OT outputs are masked with independent random bits each time, as the last player is honest. That means the summand value(s) is always one-time padded. From the security of OT, we can argue that no other information (especially the share(s) of honest party) except the output of OT, will be revealed from OT execution.

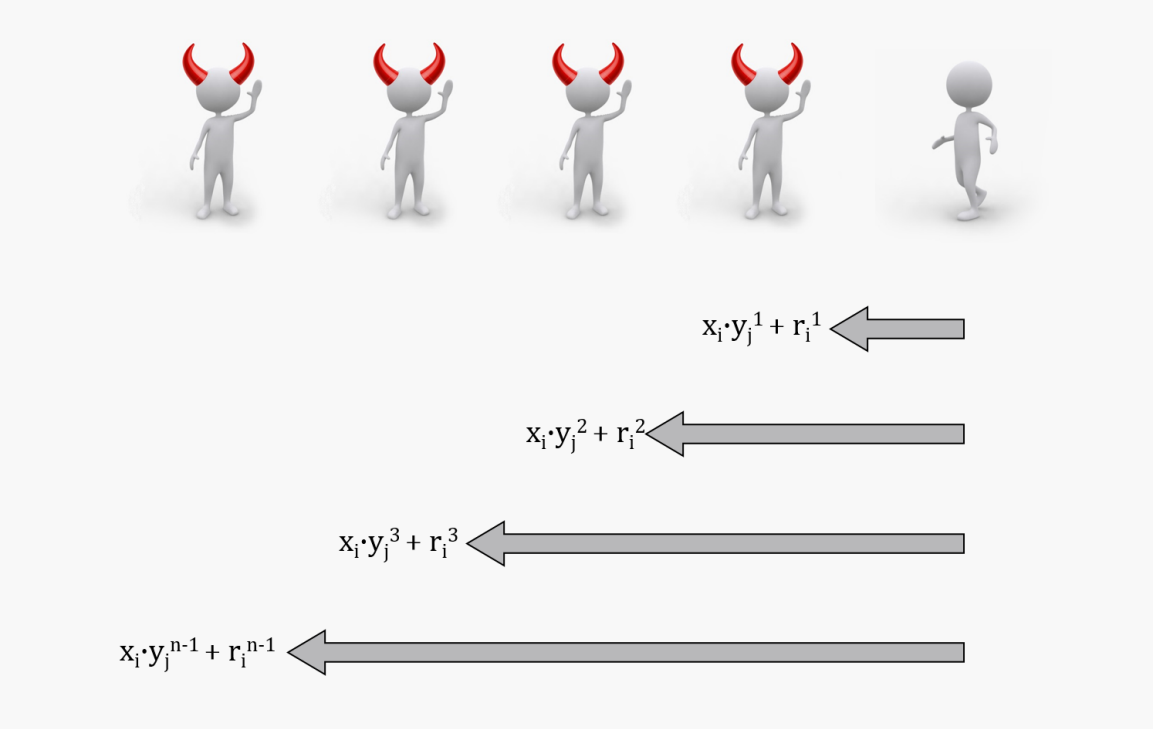

Figure 2: Honest party sharing x_i

1.2 Security Analysis of GMW n-Party Protocol

Security of the honest player lies in the security of OT. In the figure 1, no information about y_j will be leaked to P_i because of the security of Receiver. Similarly, no information about x_i is revealed to P_j because of the security of Sender. In short, the security of GMW Protocol lies in the security of OT. Note that the security of GMW is information theoretic if we can realize an ideal OT. In unconventional settings like noisy channel, OT realization in i.t. setting is possible.

1.3 Efficiency of GMW and Improvements

If multiplicative depth of the circuit is d, this protocol demands a round complexity of $O(d)$. It's because, AND operations in two different chains can be parallelized, but those in the same chain can't be.

Let $\mathbf{C}_{AND}$ be the number of AND operations in the circuit. We need $O(n^2)$ OT execution for each AND gate, and OT implementations require PKE operations. Hence we need a total of $O(n^2 \cdot \mathbf{C}_{AND})$ PKE operations. We can improve the performance by shifting the OT to the Offline phase and by using OT Extensions. Now we discuss these improvements.

1.3.1 Improvement using Offline-Online Paradigm

We can perform OT execution overhead in the Offline phase and we need to perform only bit XOR operations in Online phase. For this purpose we use Random OT, as given in the figure 3.

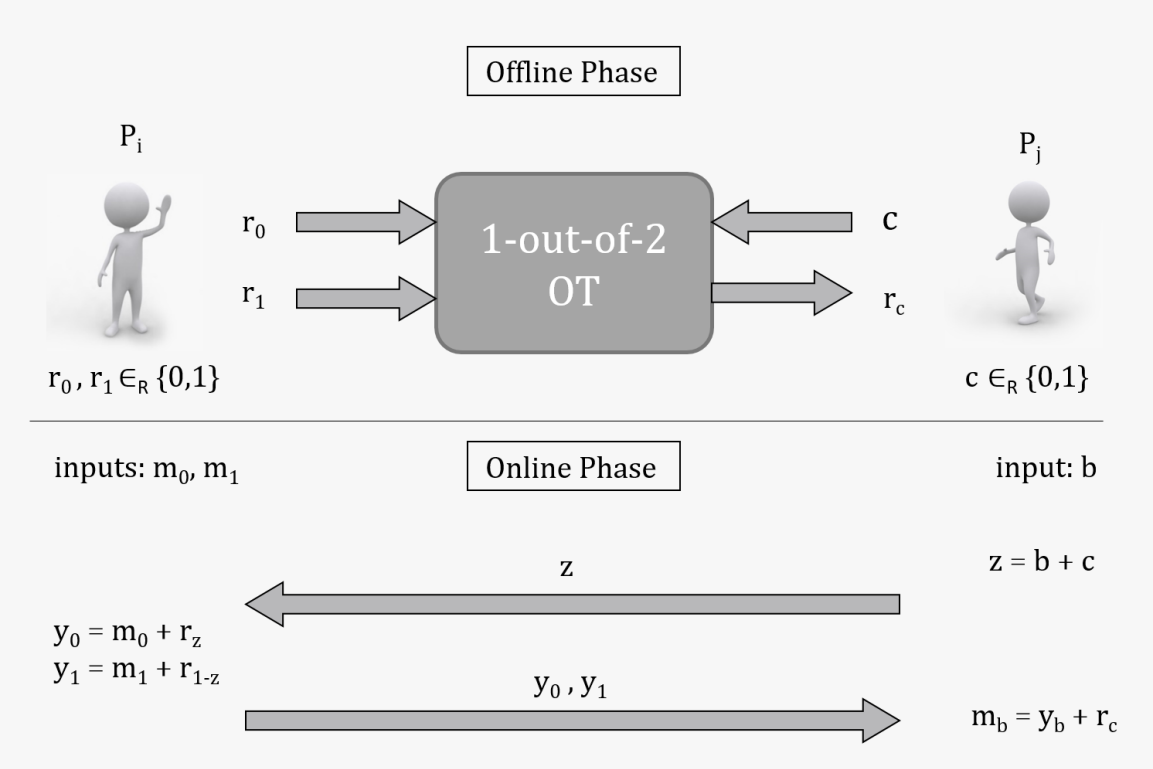

Figure 3: OT execution using Random OT from Offline phase

Correctness can be proven as follows:

$$\begin{aligned} m'_b &= y_b + r_c \\ &= y_b + r_{z+b} \\ &= \begin{cases} y_0 + r_z & \text{if b} = 0 \\ y_1 + r_{1-z} & \text{otherwise} \end{cases} \\ &= m_b \end{aligned}$$

The decrypted value m'_b is indeed m_b. We are left with how to prove the security of both the players. From the security for Receiver in OT, P_i will learn only z, which is a one-time

pad of the actual choice bit b with random bit c. Also from the security for Sender in OT, P_j will not know anything other than r_c, y_0, y_1 from the entire protocol. Now from y_b, P_j can decrypt m_b like we analyzed in correctness part. From y_{1-b} , P_j can't decrypt m_{1-b} as it is one-time masked with r_{1-c}, which is unknown to P_j from the security of Sender in Random OT execution. Note that one random OT in offline phase can be used only once in online phase, otherwise compromise extra information about b and/or m_{1-b} as they will not be one-time padded anymore.

Analysis of the Offline-Online setting of GMW. With Offline-Online setting, GMW protocol has $O(n^2 \cdot C_{AND})$ PKE operations (OT) in offline phase, and $O(n^2 \cdot C_{AND})$ XOR operations in Online phase. Now we will see how to improve GMW protocol to $O(n^2 \cdot C_{AND})$ SKE operations and k OT executions in Offline phase, and $O(n^2 \cdot C_{AND})$ XOR operations in Online phase.

1.3.2 Improvement using OT Extention (IKNP03)[1]

In this section, we discuss about several optimization primitives that are compatible to use together.

- **Domain Extension using PRG**: When we need to perform a random OT on large messages, instead perform OT using a small message and take the output of PRG with the message as the seed. The output of OT will be pseudo-random.
- **OT Extension**: It is possible that for a circuit, we need millions of OTs to perform. Even if we shift it to the Offline phase, it is still an overhead to perform a large number of OTs. The solution to this problem is OT Extension from k OTs to $poly(k)$ OTs at the cost of SKE operations.

We will explore more about these two primitives, in this lecture. Note that it can provide only computational security.

Domain Extension of OT using PRG. It is possible to reduce the size of communication if we switch from Random OT to Pseudo-Random OT. Parties can perform Random OT on pure random seeds, and both masking and unmasking is done with the output of PRG using the random seeds, as in figure 4. Using a PRG:$\{0,1\}^k \to \{0,1\}^m$ we can extend the domain from k to m where $k = |r_i|$, $m = |m_i| = |y_i|$, and $m = poly(k)$.

OT Extension. Here we *transform* k number of OTs with input size m into m OTs, each with input size l, where l is the size of output from the Hash function we choose in this section. We will be using the notation A_i represents i^{th} row of matrix A, and A^j representing j^{th} column of matrix A.

We would like to perform m OTs of message size l, using k base OTs. Let the sender be denoted as P_0 and receiver as P_1, in the context of OT. Let the inputs of P_0 be $(r_{(1,0)}, r_{(1,1)}), (r_{(2,0)}, r_{(2,1)}), \cdots (r_{(i,0)}, r_{(i,1)}), \cdots (r_{(m,0)}, r_{(m,1)})$ for m OTs, and let the inputs of P_1 be $\vec{B} = (b_1, b_2, \cdots b_m)$. We want P_1 to have the output $r_{(1,b_1)}, r_{(2,b_2)}, \cdots r_{(i,b_i)}, \cdots r_{(m,b_m)}$ at the end of the execution of OT Extension Protocol, using k base OTs.

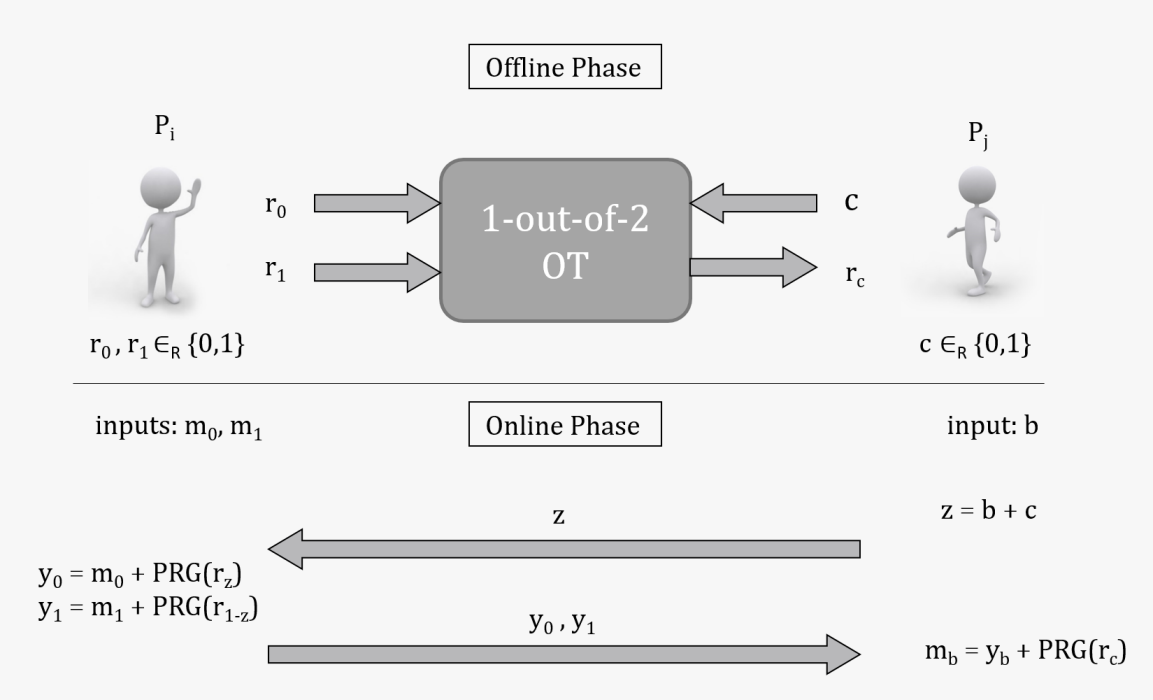

Figure 4: Extending domain from k to m where $m = poly(k)$

Claim 1 *Using k base OTs, it is possible to create two matrices $T = [t_{i,j}]_{m \times k}$ and $Q = [q_{i,j}]_{m \times k}$ known to (only) P_1 and P_0 respectively, with the following property.*

1. *T_i are chosen at random.*
2. *$\vec{S}$ is a k-bit vector, and known to P_0 only.*
3. *if $b_i = 0$ then $Q_i = T_i$, otherwise $Q_i = T_i \oplus \vec{S}$*

Assuming Claim 1 is true, we need to perform m OTs using Q, T, $\vec{S}$ and the inputs of both parties. The protocol is as follows:

1: P_0 computes $y_{(i,0)}$ and $y_{(i,1)}$, $\forall 1 \leq i \leq m$ as follows.
   - $y_{(i,0)} := Q_i \oplus r_{(i,0)}$
   - $y_{(i,1)} := Q_i \oplus \vec{S} \oplus r_{(i,1)}$
2: P_0 sends $(y_{(i,0)}, y_{(i,1)})$ to P_1, $\forall 1 \leq i \leq m$
3: P_1 calculates $r_{(i,b_i)} = T_i \oplus y_{(i,b_i)}$

But there is a trivial attack on the algorithm above. If the P_j is corrupted, he can simply perform $y_{(1,0)} \oplus y_{(1,1)} \oplus y_{(2,0)} \oplus y_{(2,1)}$ and that results in $r_{(1,0)} \oplus r_{(1,0)} \oplus r_{(1,0)} \oplus r_{(1,0)}$ along with $r_{(1,b_1)}$ and $r_{(2,b_2)}$. He may repeat this over many combinations and in the end, P_j obtain information about the inputs of P_i, which he is not supposed to know. So it is essential to

mask the messages with something else, but depends on Q_i. Consider the following algorithm:

1: P_0 computes $y_{(i,0)}$ and $y_{(i,1)}$, $\forall 1 \leq i \leq m$ as follows.
   - $y_{(i,0)} := H(i, Q_i) \oplus r_{(i,0)}$
   - $y_{(i,1)} := H(i, Q_i \oplus \vec{S}) \oplus r_{(i,1)}$
2: P_0 sends $(y_{(i,0)}, y_{(i,1)})$ to P_1, $\forall 1 \leq i \leq m$
3: P_1 calculates $r_{(i,b_i)} = H(i, T_i) \oplus y_{(i,b_i)}$

The correctness of this protocol immediately follows from the fact that $Q_i = T_i \oplus (b_i \cdot \vec{S})$. if $b_i = 0$, then $Q_i = T_i$, so the $r_{(i,0)}$ being computed correctly. Also, if $b_i = 1$, then $Q_i = T_i \oplus \vec{S}$ and $y_{(i,1)} = H(i, Q_i \oplus \vec{S}) = H(i, T_i)$. So the decryption of $r_{(i,1)}$ will also be correct, when $b_i = 1$.

Security of the Receiver is trivial, as he is not sending anything to Sender (at least for now). Security of the Sender lies in the security of Hash function we use. We can see that, the same $\vec{S}$ is being used to XOR with each Q_i, and that relation should not influence the output of the Hash function. So it demands a *Correlation-Robust* hash function (we can use Random Oracle (RO) as it is Correlation-Robust). It is interesting to note that, in practical implementations we replace RO with some hash functions. The proof of security, unlike it seems, is not useless in practical implementations. If a protocol in practice using this OT extension fails, then it boils down to the security offered by the hash function we used. The protocol can fail only because of the hash function in use could not emulate RO.

Proof of Claim 1: P_0 and P_1 perform k base OTs where P_1 act as sender and P_0 will act as the receiver. The inputs to the base OTs, and construction of T, Q and $\vec{S}$ is generated as follows:

1: P_1 chooses a random $m \times k$ matrix T.
2: P_1 set the inputs (m_0, m_1) as $(T^i, T^i \oplus \vec{B})$, for all i^{th} OT execution, $1 \leq i \leq k$
3: P_0 set a random choice bit b and save as $\vec{S}[i]$, for all i^{th} OT execution, $1 \leq i \leq k$
4: P_0 saves the output q^i from the i^{th} OT, into a matrix Q, column-wise.

The property (1) from Claim 1 is proven trivially. From the security of Receiver in OT, property (2) is also followed. Now, if b_1 is 0, then for all i^{th} OTs, the first bit of q^i is same as that of T^i, regardless of $\vec{S}$ value because, both T^i and $T^i \oplus \vec{B}$ is same at the first position. As the first bit of all the columns of Q follows this property, we can argue $q_1 = t_1$ if $b_1 = 0$. On the other hand, if $b_1 = 1$ then $Q_1[j] = t_1[j]$ if $\vec{S}[j] = 0$ and $Q_1[j] = \neg T_1[j]$ if $\vec{S}[j] = 1$. That means, $Q_1 = T_1 \oplus \vec{S}$ if $b_1 = 1$. We can show for any row j, with similar arguments, that it holds property (3). Along with that, the security for both sender and receiver gives protection to $\vec{S}$ and $\vec{B}$ from one another.

■

The OT Extension can be visualized as given in the figure 5

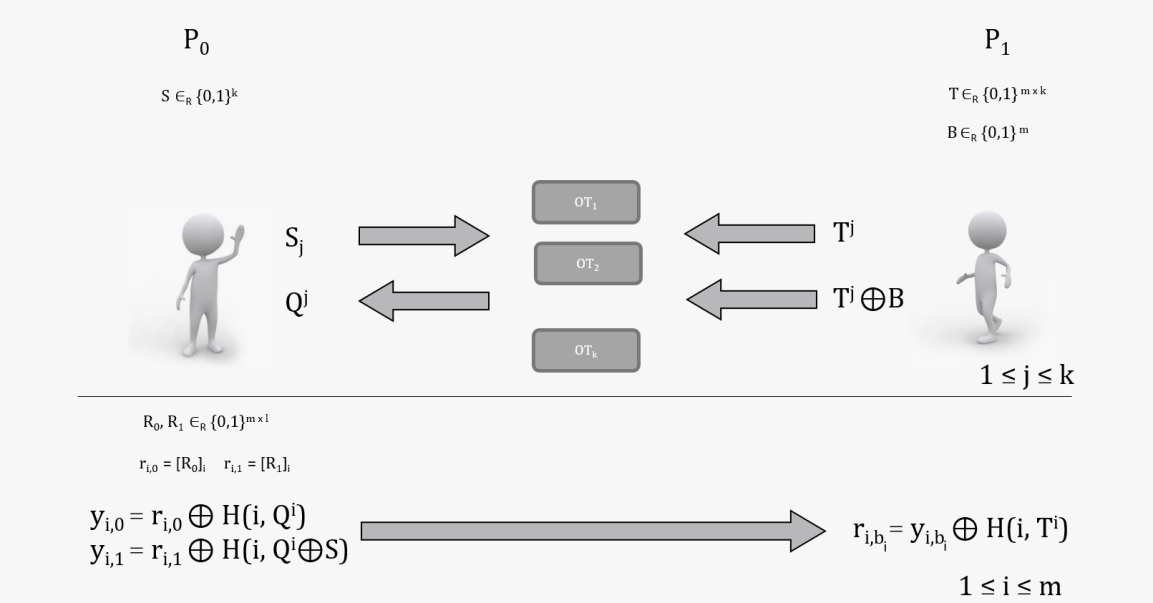

Figure 5: OT Extension (IKNP03)

2 Yao's Protocol

We have seen GMW have a round complexity of $O(d)$, k OTs in the Offline phase, and $O(n^2 \cdot C_{AND})$ SKE operations in Online phase. Yao's circuit, which we are about to discuss, gives us a constant round protocol. It demands k Offline OTs and $O(c)$ SKE operations where c is the total number of gates in the logic circuit. But it is only for 2 parties[iii]. The idea of a garbled circuit is due to A. Yao, who described the technique in oral presentations [2, p. 27] about SFE [4, 5]. The first written account of the method is by Goldreich, Micali, and Wigderson [3]. The protocol they describe, crediting Yao [4], involves associating two tokens to each wire of a boolean circuit, these having hidden semantics of 0 and 1.

2.1 Yao's Circuit

Among the two parties, one will act as a *garbled circuit-constructor* and the other one will act as a *garbled circuit-evaluator*. We denote the first party as P_0 and the latter as P_1.

2.1.1 Garbled Circuit Construction

We assume P_0 has the logic circuit corresponding to the function which P_0 and P_1 collectively want to compute. The logic circuit consists of a number of (a)wires and (b)logic gates. First, P_0 will assign two keys k_0 and k_1 for each wire in the circuit, where the keys are chosen randomly from a key-space. These must be identically looking keys, for all wires. k_0

[iii] BMR90 protocol is an extension of Yao's protocol to n parties. (D. Beaver, S. Micali, and P. Rogaway, 1990)

denotes the bit 0 and k_1 denotes 1 in the wire. Now he will take each logic gate (generally 2 inputs and hence 4 combinations of input bits) and the next level keys according to the gate. Let's explore the construction of logic gates in detail.

Let's denote the input wires as a and b for the gate g. k_0^a, k_1^a be the keys of wire a, and similar notation for wires b and c. We have 4 combinations of key pair $< k_x^a, k_y^b >$. We will encrypt 4 messages using these 4 key pairs. The message which is encrypted each time will be one of the keys of wire c (k_0^c or k_1^c). Which one to encrypt depends on the key pair. i.e, with key pair $< k_x^a, k_y^b >$, we will be encrypting the key k_z^c where $z = g(x, y)$ and $x, y, z \in \{0, 1\}$. We construct the gate in a way that, the evaluator will know exactly one key out of 2 keys of each wire, and he is unable to distinguish the meaning of the key. We can visualize these cipher-texts as boxes which needs 2 keys to open.

It is obvious to see how to encrypt for NOT gate; we will possess only two cipher-texts which are encryption of k_0^c using k_1^a and encryption of k_1^c using k_0^a. It is also important that to randomly permute these 4 cipher-boxes to lose the meaning of mapping from key-pair to their values. This figure 6 gives an idea about how the garbled AND gate looks like.

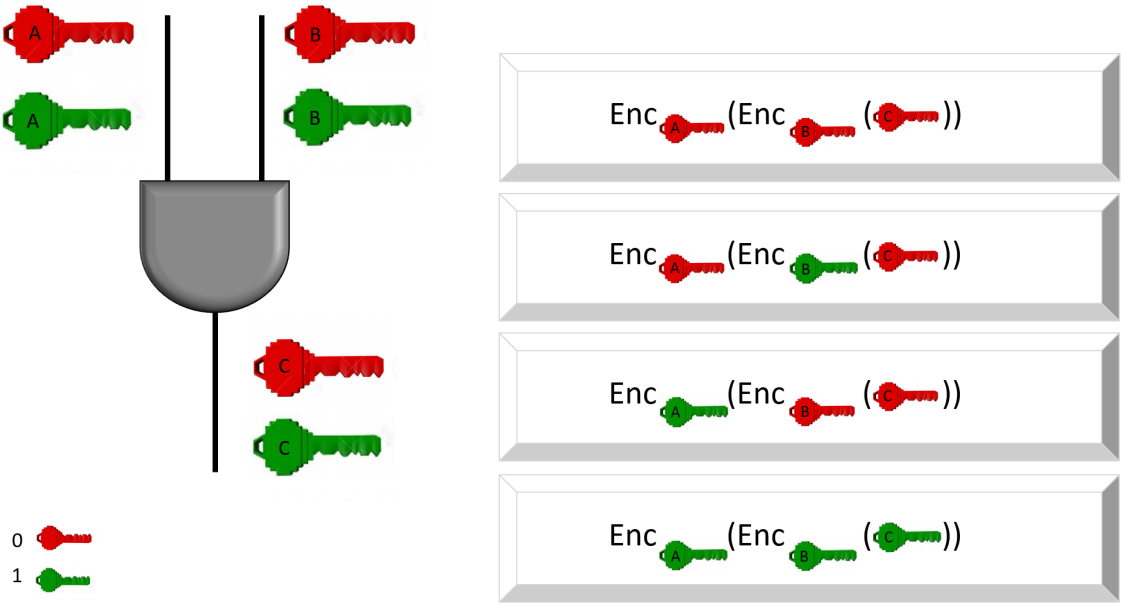

Figure 6: Garbled AND gate in Yao's circuit

Once the keys for all wires are chosen and the gates are garbled, P_0 provides P_1 the following to evaluate the circuit:

1. All garbled gates (4 or 2 cipher-boxes per gate)
2. Exactly 1 key corresponding to the input bit, in all input wires of P_0
3. The meaning of all output wire keys (mapping: $k_0^z \to 0, k_1^z \to 1$ for all output wire z)

4. A mechanism to know the key corresponding to input bit of P_1 in each of his input wires.

It is not possible to provide both the keys for each of P_1's input wires; that leads to multiple evaluations of same circuit with same input from P_0, that may leads to gain some information about P_0's input[iv]. This problem of exchanging exactly 1 key out of 2 keys, with the choice of receiver can be done using OT. If there are k number of input wires for P_1, we need to run k OTs with P_1 providing input bit as choice bit, and P_0 providing keys for 0 and 1 as it's messages, as in figure 7.

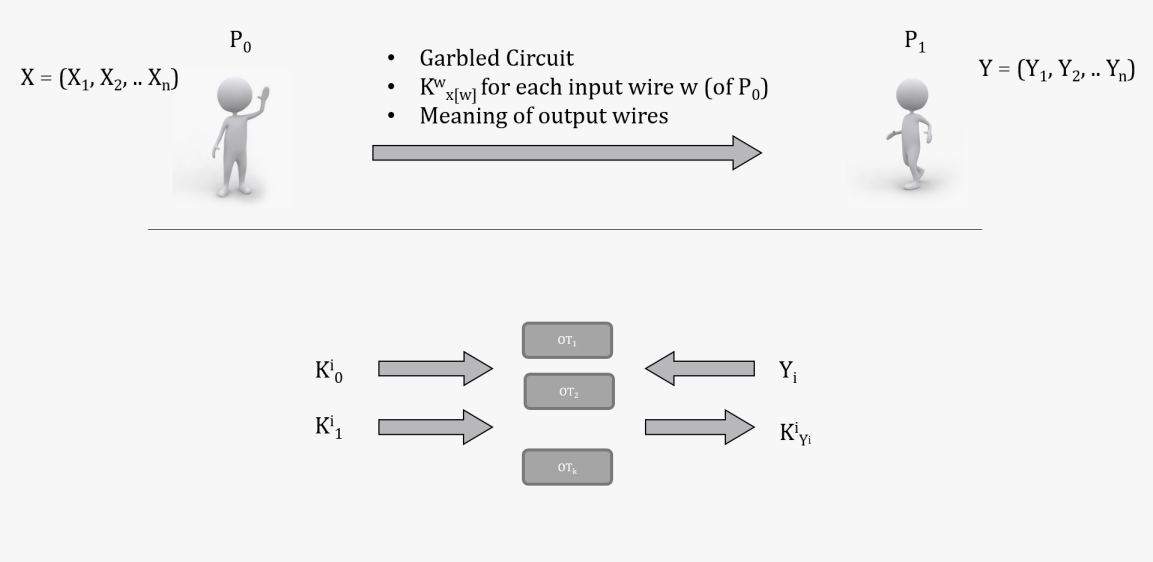

Figure 7: Yao's protocol

2.1.2 Circuit Evaluation

Upon receiving P_0's circuit with all the input keys, the circuit-evaluation is just local computation for P_1. After the circuit evaluation, P_1 will be left with the keys in the output wires. P_1 decrypts the output bits with the decryption table provided for output wires. It is optional (rather, depending on the specification of the problem), P_1 sends the output to P_0. The security lies in the security of OT as well as the indistinguishability of keys.

We leave the instantiation details of Yao's protocol to the next lecture.

[iv] Eg: Consider a functionality to find the maximum bidder where P_0 bids X and P_1 bids Y without revealing X or Y to each other. If the evaluator can run circuit many times with same X but different Y, P_1 can run a binary search to find out what is X.

References

[1] [IKNP03]Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. *Extending oblivious transfers efficiently. In Advances in Cryptology - CRYPTO'03, volume 2729 of LNCS, pages 145-161. Springer*, 2003.

[2] O. Goldreich. Cryptography and cryptographic protocols. Manuscript, June 9 2001.

[3] O. Goldreich, S. Micali, and A. Wigderson. *How to play any mental game, or a completeness theorem for protocols with honest majority. In A. Aho, editor, 19th ACM STOC, pages 218229.* ACM Press, May 1987.

[4] A. Yao. *How to generate and exchange secrets. In Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162167. IEEE, 1986.

[5] A. Yao. Protocols for secure computations. In 23rd FOCS, pages 160164. IEEE Computer Society Press, Nov. 1982

[6] Arpita Patra. *http://drona.csa.iisc.ernet.in/ arpita/SecureComputation15.html* . Course Materials.