CS670 Assignment-2

Chayan Kumawat - 220309

Program:	B.Tech
Branch:	CSE
Batch:	2026
Date of Sub:	29.03.2034 (DD-MM-YYYY)

Question 1 : Secure Multiparty Computation Using Cards

Problem Statement:

The protocol should only reveal the final outcome of the computation without disclosing any additional information. Specifically, this means that if Bob does not like Alice, he cannot learn whether Alice likes him. Similarly, Alice cannot distinguish if Bob likes her. By definition, if they both like each other, then they naturally learn that the other party's input is LIKE.

- **Bob cannot distinguish row 1 and row 3**.
- **Alice cannot distinguish row 1 and row 2**.

Row	A	B	Result of the Protocol
1	NO-LIKE	NO-LIKE	NO-LIKE
2	NO-LIKE	LIKE	NO-LIKE
3	LIKE	NO-LIKE	NO-LIKE
4	LIKE	LIKE	LIKE

The Completed Protocol:

Alice and Bob are provided with two different cards with their backside looking exactly the same for all the cards used in the protocol. Each of them has a King and an Ace card, and there is also a public Ace card.

Instructions:

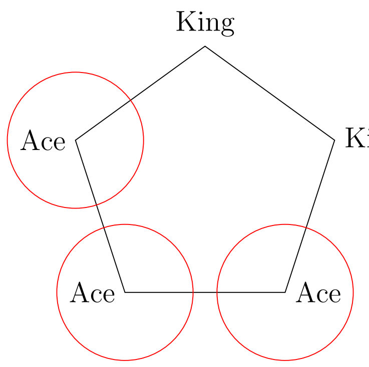
- If Alice likes Bob she will place the cards face down in order: "King", "Ace".
- If Alice does not like Bob, she will place the cards face down in the order: "King", "Ace".
- If Bob does not like Alice, he will place the cards face down in the order: "Ace", "King".
- If Bob likes Alice, he will place the cards face down in the order: "King", "Ace".

Additionally, between their cards, they place an additional facedown Ace card.

LIKE	A	B
0	Ace — King	King — Ace
1	King — Ace	Ace — King

- Five cards are placed face down on the table.
- Alice and Bob take turns **privately cycle-shifting** the cards without looking at them, and **without changing the ordering of the cards**.
- They can cycle the cards as often and in any direction they like, **ensuring the other player doesn't know the amount or directions of the cyclic permutations**.
- Finally, all cards are placed in a **circle and revealed**.
- If there are **three Aces consecutively (i.e., Ace, Ace, Ace) in the final cyclic permutation**, it is considered a LIKE; otherwise, it isn't.

0	0	Ace—King—Ace—King—Ace
0	1	Ace— King—Ace—Ace—King
1	0	King—Ace—Ace—King—Ace
1	1	King—Ace—Ace—Ace—King

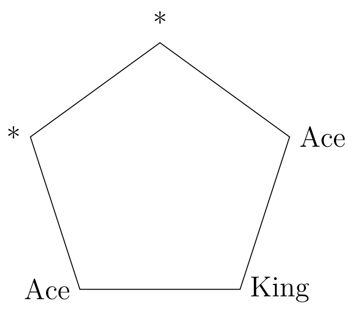


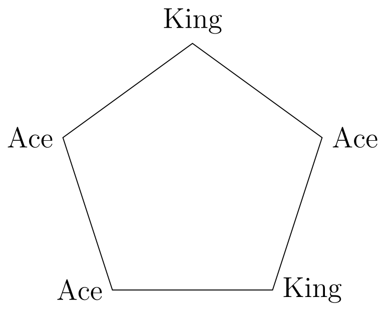
It can be seen that no matter how often we cycle the rows, **three consecutive Ace are only possible if both Alice and Bob encoded 1**. Note that in the table above, you need to keep in mind that each cell in the last column ”wraps around”, i.e. is conceptually arranged in a circular fashion. Instead of, say, King, Ace,Ace,Ace,King, it could also be any cyclic rotation like, say, Ace, King, King, Ace, Ace, and there would still be three hearts in a (cyclic) row.

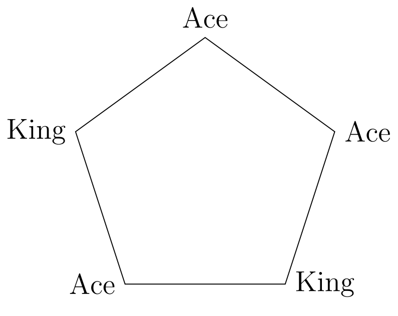
Question 1 : Why this Works

Without loss of generality, I will demonstrate a scenario where **Alice does not like Bob**, and **Alice is unable to distinguish between Bob liking her and Bob not liking her**. Similarly, we can consider the converse scenario where Bob does not like Alice, and Bob cannot distinguish between Alice liking him and Alice not liking him.
Case 1: Alice NO-LIKE Bob and Bob LIKE/NO-LIKE Alice

- The initial sequence of cards is Ace-King-Ace-*-* (here * represents the possible choices for Bob, which could be Ace-King or King-Ace).
- Since they are unaware of the number of turns the other person has made, **they cannot determine their own cards when revealed in a circular fashion**. This ambiguity arises from multiple sequences of King-Ace, making it **indistinguishable to predict the other person's input**, given the circular arrangement and lack of knowledge about the opponent's turns.
- For instance, both possible configurations at the end of the protocol could be:







The above 2 permutations are indistinguishable, showing that Alice cannot discern between row 1 and row 2.

Row	A	B	Result of the Protocol
1	NO-LIKE	NO-LIKE	NO-LIKE
2	NO-LIKE	LIKE	NO-LIKE
3	LIKE	NO-LIKE	NO-LIKE
4	LIKE	LIKE	LIKE

Table 1: Truth Table for Case 1

Similarly, we can apply the same reasoning for the scenario where Bob NO-LIKE Alice and Alice LIKE/NO-LIKE Bob. In this case, we can again find possible final results of the protocol and demonstrate that these cases are indistinguishable. Thus, Bob cannot distinguish between row 1 and row 3.

Question 2

Secure Multiparty Computation for Dot Products

Part 1 :

- Let us consider two parallel non-conflicting chains of messaging.
- Chain 1: P2 to P1, then P1 to P0
  - P2 sends $(X1, Y1, X1 \cdot Y0 - \alpha)$ to P1 in $x_{2,1}$ time
  - P1 sends $(x1 + X1)$ and $(y1 + Y1)$ to P0 in $x_{1,0}$ time
- Chain 2: P2 to P0, then P0 to P1
  - P2 sends $(X0, Y0, X0 \cdot Y1 + \alpha)$ to P0 in $x_{2,0}$ time
  - P0 sends $(x0 + X0)$ and $(y0 + Y0)$ to P1 in $x_{0,1}$ time

  Thus, the final answer will be the maximum of the two times: $(x_{2,1} + x_{1,0}, x_{2,0} + x_{0,1})$

Part 2 :

- Assume that parties P_0 and P hold $(\mathbf{x}_0, \mathbf{y}_0)$ and $(\mathbf{x}_1, \mathbf{y}_1)$ respectively, such that $\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_1$ and $\mathbf{y} = \mathbf{y}_0 + \mathbf{y}_1$.
- Their goal is to obtain shares of $\langle \mathbf{x}, \mathbf{y} \rangle$.
- P2 sends $(\mathbf{X}_0, \mathbf{Y}_0, \langle \mathbf{X}_0, \mathbf{Y}_1 \rangle + \mathbf{T})$ to P0 and P2 sends $(\mathbf{X}_1, \mathbf{Y}_1, \langle \mathbf{X}_1, \mathbf{Y}_0 \rangle - \mathbf{T})$ to P1.
- P0 sends $(\mathbf{x}_0 + \mathbf{X}_0)$ and $(\mathbf{y}_0 + \mathbf{Y}_0))$ to P1. Similarly, P1 sends $(\mathbf{x}_1 + \mathbf{X}_1)$ and $(\mathbf{y}_1 + \mathbf{Y}_1)$ to P0
- P0 computes $\mathbf{Z}_0 = \langle \mathbf{x}_0, (\mathbf{y}_0 + (\mathbf{y}_1 + \mathbf{Y}_1)) \rangle - \langle \mathbf{Y}_0, (\mathbf{x}_1 + \mathbf{X}_1) \rangle + \mathbf{Y}_0 + \langle \mathbf{X}_0, \mathbf{Y}_1 \rangle + \mathbf{T}$
- Similarly, P1 computes $\mathbf{Z}_1 = \langle \mathbf{x}_1, (\mathbf{y}_1 + (\mathbf{y}_0 + \mathbf{Y}_0)) \rangle - \langle \mathbf{Y}_1, (\mathbf{x}_0 + \mathbf{X}_0) \rangle + \mathbf{Y}_1 + \langle \mathbf{X}_1, \mathbf{Y}_0 \rangle - \mathbf{T}$
- Now Observe that $\mathbf{z}_0 + \mathbf{z}_1 = \langle (\mathbf{x}_0 + \mathbf{x}_1), (\mathbf{y}_0 + \mathbf{y}_1) \rangle$ This was the MPC protocol to calculate the shares of $\langle \mathbf{x}, \mathbf{y} \rangle$.

Question 2

Implementing an xorif functionality

Problem statement

- We have two parties P0 and P1 having the hold (x_0, y_0, b_0) and (x_1, y_1, b_1) respectively. They have the property that, $x = x_0 \oplus x_1$,$y = y_0 \oplus y_1$, and $b = b_0 \oplus b_1$
- Their goal is compute $z0$ and $z1$ respectively such that, both the following conditions hold true:
  - $z_0 \oplus z_1 = x$ if $(b_0 \oplus b_1 = 0)$
  - $z_0 \oplus z_1 = x \oplus y$ if $(b_0 \oplus b_1 = 1)$
  - i.e we can say that $z_0 \oplus z_1 = x \oplus (y \wedge (b_0 \oplus b_1))$

The Protocol:

- Let's consider the expression that we aim to compute through Multi-Party Computation (MPC):

$$z_0 \oplus z_1 = x \oplus (y \wedge (b_0 \oplus b_1))$$

$$\Rightarrow \text{we can rewrite } x \oplus (y \wedge (b_0 \oplus b_1)) \text{ as}$$

$$\Rightarrow (x_0 \oplus x_1) \oplus ((y_0 \oplus y_1) \wedge (b_0 \oplus b_1))$$

$$\Rightarrow (x_0 \oplus x_1) \oplus (((y_0 \wedge (b_0 \oplus b_1)) \oplus ((y_1 \wedge (b_0 \oplus b_1)))$$

$$\Rightarrow (x_0 \oplus x_1) \oplus (((y_0 \wedge b_0) \oplus (y_0 \wedge b_1)) \oplus ((y_1 \wedge b_0) \oplus (y_1 \wedge b_1)))$$

$$\Rightarrow (x_0 \oplus x_1) \oplus ((y_0 \wedge b_0) \oplus (y_0 \wedge b_1) \oplus (y_1 \wedge b_0) \oplus (y_1 \wedge b_1))$$

$$\Rightarrow (x_0 \oplus x_1) \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1) \oplus (y_1 \wedge b_0) \oplus (y_1 \wedge b_1)$$

$$\Rightarrow x_0 \oplus x_1 \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1) \oplus (y_1 \wedge b_0) \oplus (y_1 \wedge b_1)$$

$$\Rightarrow x_0 \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1) \oplus x_1 \oplus (y_1 \wedge b_0) \oplus (y_1 \wedge b_1)$$

$$\Rightarrow (x_0 \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1)) \oplus (x_1 \oplus (y_1 \wedge b_1) \oplus (y_1 \wedge b_0))$$

- The shares $(x_0 \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1))$ and $(x_1 \oplus (y_1 \wedge b_1) \oplus (y_1 \wedge b_0))$ represent $z_0 \oplus z_1$. Calculating $x_0 \oplus (y_0 \wedge b_0)$ is straightforward for P_0, and similarly for $x_1 \oplus (y_1 \wedge b_1)$ for P_1. We need P_0 to obtain the value of $y_0 \wedge b_1$ using an MPC protocol, and similarly, P_1 to obtain the value of $y_1 \wedge b_0$.
- We will employ the Du-Atallah protocol so that P_0 can compute $y_0 \wedge b_1$ and P_1 can compute $y_1 \wedge b_0$.
- Now, P_0 can compute $(x_0 \oplus (y_0 \wedge b_0) \oplus (y_0 \wedge b_1))$ and P_1 can compute $(x_1 \oplus (y_1 \wedge b_1) \oplus (y_1 \wedge b_0))$.
- We will now employ the Free-XOR protocol for the bitwise computation of $(x_0^i \oplus (y_0^i \wedge b_0^i) \oplus (y_0^i \wedge b_1^i)) \oplus (x_1^i \oplus (y_1^i \wedge b_1^i) \oplus (y_1^i \wedge b_0^i))$.
- Thus, we have successfully computed the desired $z_0 \oplus z_1$.

Question 3

Point-and-Permute Optimization

For the **Point-and-Permute Optimization**, typically, four ciphertexts are sent for each gate. However, there exist **specific gates**, although rare, for which only two ciphertexts suffice. This optimization is applicable solely to **single-gate circuits**; in **multi-gate circuits**, sending all four ciphertexts is necessary, as the **Garbler** lacks knowledge about the **colors of intermediate wires**.

The rationale behind this lies in the fact that the **Garbler** knows their input label and thus the color associated with it. Consequently, they can transmit the two ciphertexts corresponding to that color. For example, if the **Garbler's input** were labeled $A1$, they could transmit:

1. $\text{ENC}(A1, \text{ENC}(B0, \text{ENC}(C0)))$ with colors $(\text{Blue}, \text{Yellow})$
2. $\text{ENC}(A1, \text{ENC}(B1, \text{ENC}(C1)))$ with colors $(\text{Blue}, \text{Blue})$

This approach effectively reduces the number of ciphertexts required for certain gates, **optimizing the overall efficiency** of the encryption process.

Consider the **Point-and-Permute Optimization**:

A	B	Ciphertext	Colors
$A0$	$B0$	$\text{ENC}(A0, \text{ENC}(B0, \text{ENC}(C0)))$	$(\text{Yellow}, \text{Blue})$
$A1$	$B0$	$\text{ENC}(A1, \text{ENC}(B0, \text{ENC}(C0)))$	$(\text{Blue}, \text{Yellow})$
$A0$	$B1$	$\text{ENC}(A0, \text{ENC}(B1, \text{ENC}(C0)))$	$(\text{Yellow}, \text{Yellow})$
$A1$	$B1$	$\text{ENC}(A1, \text{ENC}(B1, \text{ENC}(C1)))$	$(\text{Blue}, \text{Blue})$

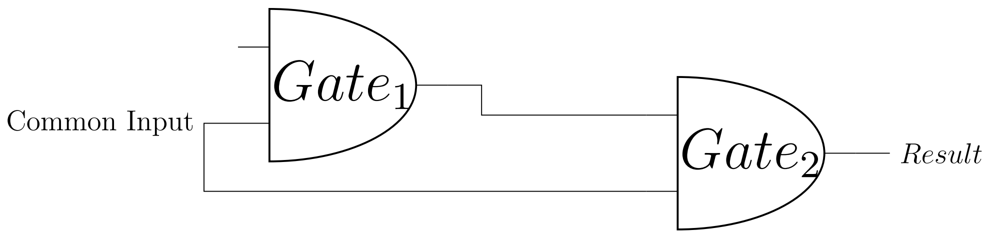
(The above example is taken from class)

Question 3

Choice of the encryption algorithm

These are the following reason because of which we cannot use One Time pad encrpytion algorithm for plain garbled circuit but can use for point and permute optimization.

- **One-time pads are only secure when we do not use them with the same key multiple times for encryption**. In the case of garbled circuits, **there can be a possibility that we have to encrypt again with the same key**, which could lead to trouble. Let's say you have a circuit with two gates, where the first gate's one input is common with the input of the second gate. After performing the Multiparty Computation (MPC) for gate 1, while performing the protocol for gate 2, the same key will be used for encryption. Since we used one-time pads, this will not be secure as it can be decrypted with the previous key.



- With **point-and-permute** optimization, we add a color bit at the end of the key. This **will create a new key** (which will be completely random and independent of the previous key). By doing this, even if we have the same complex example as before with the same input, a different key will be generated each time it is used as a gate input. **This will maintain the security of the system**.

- When the evaluator evaluates the encrypted output of the garbled gate one by one, among all the possible encrypted outputs, decryption of the one-time pad is done using XOR operation with the key of the one-time pad. The problem with this encryption algorithm used in garbled circuits is that the evaluator may obtain some output (either correct or some garbage) by decrypting all the encrypted outputs. In this case, **the evaluator will not know which output is the correct one, and this is supposed to be determined**.

- In the case of **Point and Permute** optimization, the **evaluator knows which encrypted output has to be decrypted** to obtain the correct result (key of the output) that is supposed to be calculated.