Assignment 2

Jiyanshu Dhaka

1. Secure Multiparty Computation Using Cards

Alice and Bob use playing cards to decide if they should go on a date. They want to reveal their choices only if both of them like each other. The complete protocol is:

1. Alice and Bob each have a set of two cards: a "King" and an "Ace".
2. If Alice likes Bob, she arranges her two cards face down in the order: "King", "Ace".
3. If Alice does not like Bob, she arranges her two cards face down in the order: "Ace", "King".
4. If Bob likes Alice, he arranges his two cards face down in the order: "Ace", "King".
5. If Bob does not like Alice, he arranges his two cards face down in the order: "King", "Ace".
6. An additional "Ace" card is placed face down between their cards.
7. After both Alice and Bob have placed their cards in the order of Alice's cards, the additional "Ace" card, and then Bob's cards.
8. They perform cyclic shuffling any number of time keeping the number of shuffles to be private and then check the cards. (ex. cyclic shuffle of AKAAK can be any of KAAKA,AAKAK,AKAKA and KAKAA).
9. They keep doing this until two consecutive "King" cards appear or all possible cyclic shuffling is done.
10. We check during cyclic shuffle, if at any point, two consecutive "King" cards (or 3 consecutive "Ace" cards) are revealed, then both Alice and Bob agree that they go on a date.
11. If no consecutive "King" cards are revealed, they do not proceed with the date.

Explanation

We should be able to prove that the following three cases are equivalent for the algorithm:

1. Only Alice likes Bob
2. Only Bob likes Alice
3. Both don't like each other

Case 1: Only Alice likes Bob

In this case, the arrangement of cards will be "KAAKA". After all possible cyclic shuffling(AAKAK,AKAKA, KAKAA and AKAAK), the arrangement ensures that no consecutive "King" cards appear together. When they reveal the cards, the presence of an "Ace" between the "King" cards ensures that no consecutive "King" cards are revealed. So, they do not proceed with the date.

Case 2: Only Bob likes Alice

Similarly, in this case, the arrangement of cards will be "AKAAK". After all possible cyclic shuffling(KAAKA,AAKAK,AKAKA and KAKAA) the arrangement ensures that no consecutive "King" cards appear together. When they reveal the cards, the presence of an "Ace" between the "King" cards ensures that no consecutive "King" cards are revealed. So, they do not proceed with the date.

Case 3: Both don't like each other

If neither Alice nor Bob likes each other, then the arrangement will be "AKAKA". Again, during the revelation(KAKAA,AKAAK,KAAKA and AAKAK), no consecutive "King" cards appear, allowing them to not proceed with the date.

Conclusion

Alice and Bob can use this protocol to decide if they want to go on a date, without revealing their personal preferences unless they both like each other. The protocol uses playing cards and a cyclic shuffling process to securely compute the decision. This ensures privacy by preventing either party from deducing the other's preferences. It also guarantees fairness by allowing both Alice and Bob to agree on the outcome based on the revealed cards.

2. Secure Multiparty Computation on Secret Shares

2.1.1 Secure Multiparty Computation for Dot Products

In the initial secure multiplication protocol, parties $P2$, $P0$, and $P1$ engage in exchanging values and performing computations. Each transmission between

parties consumes some time, referred to as $x_{i,j}$.

Original Protocol Communication Time Analysis:

1. **Dealer ($P2$) Sampling**:
    - This step doesn't require communication; $P2$ simply selects random values.
2. **Sending Values to $P0$ and $P1$**:
    - $P2$ sends $(X0, Y0, X0 \cdot Y1 + \alpha)$ to $P0$ and $(X1, Y1, X1 \cdot Y0 - \alpha)$ to $P1$.
    - It takes $x_{2,0}$ time for $P2$ to send to $P0$ and $x_{2,1}$ time for $P2$ to send to $P1$.
3. **Sending Adjusted Values between $P0$ and $P1$**:
    - $P0$ sends $(x_0 + X0)$ and $(y_0 + Y0)$ to $P1$.
    - $P1$ sends $(x_1 + X1)$ and $(y_1 + Y1)$ to $P0$.
    - This step takes $x_{0,1}$ time for $P0$ to $P1$ and $x_{1,0}$ time for $P1$ to $P0$.
4. **Computations by $P0$ and $P1$**:
    - Once they have the necessary data, $P0$ and $P1$ perform their computations independently.

Total Communication Time in Original Protocol:

The total communication time equals $x_{2,0} + x_{2,1} + x_{0,1} + x_{1,0}$.

Optimized Protocol:

The goal of the optimized protocol is to improve efficiency by refining how values are shared and exchanged among parties.

1. **Dealer ($P2$) Sampling**:
    - $P2$ continues to choose random values as before.
2. **Parallel Sharing of Values from $P2$**:
    - Now, $P2$ sends $(X0, Y0, X0 \cdot Y1 + \alpha)$ to $P0$ and $(X1, Y1, X1 \cdot Y0 - \alpha)$ to $P1$ simultaneously to optimise the algorithm.
3. **Sharing Adjusted Values between $P0$ and $P1$**:

- After getting tuples from $P2$, $P0$ and $P1$ shares the tuples with each other according to step 2 and 3.
- This exchange takes $x_{0,1}$ time for $P0$ to $P1$ and $x_{1,0}$ time for $P1$ to $P0$.

4. **Computations by $P0$ and $P1$:**

   - After receiving the adjusted values, $P0$ and $P1$ proceed with their calculations independently.

Total Communication Time in Optimized Protocol:

- The total communication for $P0$ is $\mathrm{x}_{2,0} + x_{0,1}$.
- The total communication for $P1$ is $\mathrm{x}_{2,1} + x_{1,0}$.
- As both computations can be done independently so the communications can be done parallely. so total time will be $\max(\mathrm{x}_{2,0} + x_{0,1}, x_{2,1} + x_{1,0})$.

Comparison:

- **Original Protocol**: Involves a linear exchange of messages between parties.
- **Optimized Protocol**: Allows for simultaneous sharing of values, reducing overall communication time.

Conclusion:

The optimized protocol streamlines communication by enabling parties to share values concurrently, enhancing efficiency. This adjustment not only saves time but also makes secure multiparty computation more feasible for practical applications.

2.1.2 MPC protocol to compute the shares of z

Random Vector Generation by P2:

- P2 generates four random vectors: $\vec{X}_0, \vec{Y}_0, \vec{X}_1, \vec{Y}_1$, each of some finite dimension n.
- P2 also generates a random scalar value α.

Computing Adjusted Values:

- P2 will compute $\langle \vec{X}_0, \vec{Y}_1 \rangle + \alpha$ for P0.
- P2 will compute $\langle \vec{X}_1, \vec{Y}_0 \rangle - \alpha$ for P1.

Sharing Adjusted Vectors:

- P2 sends $(\vec{X}_0, \vec{Y}_0, \langle \vec{X}_0, \vec{Y}_1 \rangle + \alpha)$ to P0.
- P2 sends $(\vec{X}_1, \vec{Y}_1, \langle \vec{X}_1, \vec{Y}_0 \rangle - \alpha)$ to P1.

Sharing in-between P0 and P1:

- P0 sends $(\vec{x}_0 + \vec{X}_0)$ and $(\vec{y}_0 + \vec{Y}_0)$ to P1.
- P1 sends $(\vec{x}_1 + \vec{X}_1)$ and $(\vec{y}_1 + \vec{Y}_1)$ to P0.

Computations by P0 and P1:

- Upon receiving modified vectors, each party computes the inner product of the received vector and their local share of the original vectors.
- P0 computes: $z_0 = \langle \vec{x}_0 + \vec{X}_0, \vec{y}_0 + (\vec{y}_1 + \vec{Y}_1) \rangle - \langle \vec{Y}_0, \vec{x}_1 + \vec{X}_1 \rangle + (\langle \vec{X}_0, \vec{Y}_1 \rangle + \alpha)$
- P1 computes: $z_1 = \langle \vec{x}_1 + \vec{X}_1, \vec{y}_1 + (\vec{y}_0 + \vec{Y}_0) \rangle - \langle \vec{Y}_1, \vec{x}_0 + \vec{X}_0 \rangle + (\langle \vec{X}_1, \vec{Y}_0 \rangle - \alpha)$
- Finally, the computed values z_0 and z_1 together represent the inner product of the shared vectors $\vec{x}$ and $\vec{y}$.

Changes in the notation and operators:

- The protocol remains consistent with the previously outlined structure, it is just that vectors are there instead of scalar values. In this adjusted context, the operation of multiplication changes to the dot product between vectors, while addition denotes either scalar or vector addition, contingent upon the nature of the elements being combined.

2.2 Implementing xor_if Functionality

Implementation using MPC:

1. **Secure AND Operation**: Use an MPC protocol to compute $(y \& a)$ securely.
2. **Conditional XOR**: Based on the shares obtained from the AND operation and shares of x, perform XOR operation conditionally to obtain shares of the result z.
3. **Combine Shares**: Combine shares from both parties to obtain the final result z.

Bit Extension

We extend the input bit b to a 64-bit number γ where, when b is 1, all 64 bits of γ are set to 1, and when b is 0, all 64 bits of γ are set to 0. This extension is crucial for secure bitwise AND operations later in the computation.

Each party starts with their own share of b, denoted as b_0 and b_1 for Party 0 (P0) and Party 1 (P1) respectively. To compute γ, each party simply XORs their share of b resulting in $\gamma = \gamma_0 \oplus \gamma_1 = b_0 + b_1$.

Calculation of $(y\&\gamma)$ using MPC

Now, To calculate $(y\&a)$ securely, we use an MPC protocol similar to the Du-Atallah protocol, where addition is replaced with XOR operation and multiplication with AND operation. Each party computes their shares of $(y\&a)$ using the MPC protocol. So, with γ computed, we can proceed to compute $y\&\gamma$, where y is the input y and & represents the bitwise AND operation.

Both parties execute the MPC protocol to securely compute $y\&\gamma$. Each party's share of y (y_0 for P0 and y_1 for P1) is used along with their corresponding share of γ (γ_0 for P0 and γ_1 for P1) to compute their share of Z, denoted as Z_0 and Z_1 respectively. Note that both parties have shares of b but that does not reveal information about b and MPC protocol keeps the bit secret.

After the protocol execution, both parties obtain their respective shares of Z, and it holds that $Z = Z_0 \oplus Z_1$.

XOR with Conditional Result

Once each party has their shares of $(y\&a)$, they XOR it with their shares of x (denoted as r_0 and r_1), based on the condition determined by b. If b is true, parties XOR their shares of $(y\&a)$ with their shares of x, obtaining shares of $x \oplus y$. If b is false, parties XOR their shares of $(y\&a)$ with their shares of x, which effectively leaves the value unchanged. Finally, parties combine their shares to obtain the final result z, where $z_0 \oplus z_1 = x$ if b is false, and $z_0 \oplus z_1 = x \oplus y$ if b is true.

Compute the Final Result

In this final step, both parties compute their shares of the final result z based on their shares of x and Z.

Party 0 computes $z_0 = x_0 \oplus Z_0$, and Party 1 computes $z_1 = x_1 \oplus Z_1$. Because if The final output z satisfies $z = x \oplus (y\&\gamma)$.

Conclusion

The `xor_if` functionality can be securely implemented in an MPC setting using protocols that ensure privacy-preserving computations. By leveraging secure operations such as XOR and AND within MPC protocols, parties can collaboratively compute the desired result without revealing sensitive information. This approach enables the fulfillment of privacy requirements while achieving the desired computation securely.

3 Optimizations in Garbled Circuits

3.1 Point-and-Permute Optimization

The two types of gates that require only two cipher texts are:

Optimizations in Garbled Circuits

Point-and-Permute Optimization

While doing the point-and-permute optimization, for each gate we need to send four ciphertexts. However, there are some (albeit very few) gates for which you can get away with sending just two ciphertexts.

1. **When the evaluator knows the truth value in one of the input wires in AND gate:**
   - If inputs are A and B with output C, and the evaluator knows the truth value of B, they can infer the truth value of C without needing to transmit additional information.
   - For example, if B is known to be false, then C must also be false regardless of the value of A .

- Similarly, if the evaluator knows that $B \oplus \delta$ is true, then they can deduce the truth value of C based on A, because knowing $A \oplus C$ would suffice to infer H(B $\oplus\delta)\oplus A \oplus C$.

2. **Row reduction of Garbled Tables (Equivalent to XOR gates):**
   - In certain cases, particularly in optimizations like Free XOR, which allows for XOR gates to be evaluated without any additional communication overhead, the row reduction technique can significantly reduce the number of ciphertexts transmitted.
   - In this technique, instead of transmitting both rows of the garbled truth table, the evaluator can choose to XOR one row with another. This operation can effectively reduce the number of ciphertexts needed to evaluate XOR gates.

3. **XOR Gates:**
   - XOR gates are special in that they only require two ciphertexts to be evaluated securely.
   - This property arises from the nature of XOR operations. Given the truth values of the inputs, the output of an XOR gate can be determined without revealing any additional information.
   - The XOR operation exhibits commutative and associative properties. By assigning garbled values to input wires in a manner where one value signifies 0 and the other signifies 1, the resulting garbled output directly reflects the XOR outcome of the input garbled values. Consequently, only two cipher texts are necessary for XOR operations.
     In summary, these optimizations leverage specific properties of the gates involved in the circuit to reduce the number of ciphertexts transmitted during evaluation, thereby improving the efficiency of garbled circuit protocols.

3.2 Choice of the encryption algorithm

1) In the context of regular non-streamlined Encrypted Circuits, the utilization of One-Time Pads (OTPs) as the encryption method poses impracticality due to the necessity of reusing the same key multiple times during the concealment of the truth table.
2) The security guarantees offered by the One-Time Pad mechanism are contingent upon its singular employment, as connoted by its nomenclature.
3) Nevertheless, with the introduction of Point-and-Shuffle optimization methodologies, a fundamental shift occurs within the paradigm.
4) Under these circumstances, the entity tasked with evaluating the circuit decrypts only one specific value per iteration.
5) Consequently, the secure employment of One-Time Pads becomes a feasible endeavor.
6) Within the foundational framework of non-optimized Encrypted Circuits, the application of One-Time Pads (OTPs) as the encryption mechanism is precluded by the following rationales:
   a) Firstly, there is the issue of uniformity. OTPs necessitate identical keys for both encryption and decryption, whereas in the context of garbled circuits, each gate corresponds to a distinct key pair. Consequently, the adoption of OTPs would necessitate the imposition of unique key requirements

for every gate, thereby engendering excessive complexity in both the generation and evaluation phases of garbled circuits.

b) Additionally, OTP keys exhibit a reliance on the data being encrypted. This implies that for identical input data, distinct keys would be required for different gates, resulting in inefficiencies in circuit generation and evaluation processes.

7) However, the introduction of Point-and-Shuffle Optimization strategies alleviates the necessity for strict uniformity, owing to the substantial reduction in the number of ciphertexts involved. This reduction permits the utilization of XOR-based ciphers.

8) Within this context, XOR operations emerge as computationally lightweight, and the requirement for unique keys for encrypting the input wires per gate diminishes, rendering the integration of OTP-like ciphers more feasible.