Secure multi-party computation

Assignment 4

Najah Kamal 325829133

Salam Qais 327876116

Sami Serhan 327876298

Semester 1

First of all, our code is written according to BeDoZa protocol and according to equation 3:

$$\text{For } d = 2 \text{ the function is } f_{\vec{a},4} \colon \{0,1,2,3\}^2 \to \{0,1\} \text{ defined by}$$

$$f_{\vec{a},4}(x_1, x_2) = \begin{cases} 1 & \text{if } a_1 x_1 + a_2 x_2 \geq 4 \\ 0 & \text{otherwise} \end{cases}$$

We used the code that we have written for assignment 3, but we have changed a lot in order to fit with the new assignment, and to enable using of MAC with BeDoZa.

In our code we have 5 classes:

- MAC class: it is simple as it is, has value, key and tag.
- Node class: each node has it value, key and tag, the kind of operation * or +, and the parents nodes, and a flag that decides if the second parent is a constant, and a array of sons, and a flag the determines wither this node is a leaf.
- Circuit class: it is a large class, that has many variables and methods, which we have explained about some of them in the code. This class can generate circuits and nodes according to the operations and the parents, and there are more conditions.
- Dealer class: the dealer as we learned makes MACs for Alice and Bob, and he does that randomly. Also we have the secret sharing.
- Alice class: it has many methods and variables. It can open the hidden variables by OpenTo function. It receives random MACs arrays from Bob and send another array back, in a way that we have learned in the class. Also, it can send and receives other things from and to Bob. In addition, Alice can do add wither it is with constant or not, the same for multiplication operation. And it can knows if she has an output. And verify the output.
- Bob class: as always - it is similar to Alice's class, but Alice's class is larger, so if you understand Alice's class you can understand Bob's one.

At the beginning

- Note: that the computation that we do have mod p in it, but we don't write it all the time.

What the subprotocol MAC is:

$MAC(x_A, x_B)$:

As we know, we want to generate keys and tags according to the values, and we do so like what we have learned in the class:

1. First, we sample $(\alpha_A, \beta_{A,1} \dots \beta_{A,m_1}) \leftarrow_R \mathbb{Z}_p$ – this is for Alice, and for Bob it is the same: $(\alpha_B, \beta_{B,1} \dots \beta_{B,m_2}) \leftarrow_R \mathbb{Z}_p$ – this is for Bob, and $m_1 \; and \; m_2$ are the lengths of $x_1 \; and \; x_2$
2. For every index $i_A \; on \; x_A \; and \; index \; i_B \; on \; x_B$, we create keys $k_{A,i_A} = (\alpha_A, \beta_{A,i_A}) \; and \; k_{B,i_B} = (\alpha_B, \beta_{B,i_B})$
3. And last thing, for every index $i_A \; on \; x_A \; and \; index \; i_B \; on \; x_B$, we create tags $t_{A,i_A} = \alpha_B * u_A + \beta_{B,i_B} \quad and \quad t_{B,i_B} = \alpha_A * u_B + \beta_{A,i_A}$

Now we will answer the exercise that we had in the class: to write the Add gate with a constant: $\boldsymbol{Add}([\boldsymbol{x}], \boldsymbol{c})$:

Alice output: $(z_A, k_{A,z}, t_{A,z})$ where:

- $z_A = x_A + c$
- $k_{A,z} = (\alpha_{A,x_A}, \beta_{A,x_A})$
- $t_{A,z} = t_{A,x_A}$

Bob output: $(z_B, k_{B,z}, t_{B,z})$ where:

- $z_B = x_B$
- $k_{B,z} = (\alpha_{B,x_B}, \beta_{B,x_B} - c * \alpha_{B,x_B})$
- $t_{B,z} = t_{B,x_B}$

And, now the MUL operation: $\boldsymbol{Mult}([\boldsymbol{x}], \boldsymbol{c})$:

Alice output : $(z_A, k_{A,z}, t_{A,z})$ where:

- $z_A = x_A * c \; mod \; p$
- $k_{A,z} = (\alpha_{A,x_A}, \beta_{A,x_A} * c \; mod \; p)$
- $t_{A,z} = t_{A,x_A} * c \; mod \; p$

Bob output: $(z_B, k_{B,z}, t_{B,z})$ where: (similar to Alice):

- $z_B = x_B * c$
- $k_{B,z} = (\alpha_{B,x_B}, \beta_{B,x_B} * c\ )$
- $t_{B,z} = t_{B,x_B} * c$

And now: $\boldsymbol{Mul}([\boldsymbol{x}], [\boldsymbol{y}])$:

- We first calculate: $[d] \leftarrow Add([x], [u])$ and $d \leftarrow Open([d])$
- Similar for e: $[e] \leftarrow Add([y], [v])$ and $e \leftarrow Open([e])$
- Then, Alice and Bob compute $[z]$ using this equation:
  $[z] = Add\left(Add\left([w], Add\big(Mult([v], d), Mult([u], e)\big)\right), e * d\right)$
- And the last thing: Alice outputs z_A and Bob outputs z_B

In addition, we do the verify function as we learned in the class, for example:

The verify for Alice is: $Ver(A, k_i, t, x)$:

- Alice outputs yes if $t = \alpha_i * x + \beta_i$, otherwise reject.

And, it is similar to Bob.

The OpenTo:

- For Alice: $OpenTo(A, [x])$:
  Bob sends $x_B\ and\ t_{B,x}\ to\ Alice$
  Alice outputs $x = x_A + x_B\ \ if\ Ver\big(k_{A,x}, t_{B,x}, x_B\big) = accept,\ \ (otherwise\ apport)$
- For Bob: $OpenTo(B, [x])$:
  Analogous
- $Open([x])$: Run both $OpenTo(B, [x])\ and\ OpenTo(A, [x])$
- Denote: $(x, \perp) \leftarrow OpenTo(A, [x])$

Now, Subprotocols Input wires sharing $\boldsymbol{Shr}(\boldsymbol{A}, \boldsymbol{x}), \boldsymbol{Shr}(\boldsymbol{B}, \boldsymbol{x})$:

To share each of Alice's (resp. Bob's) input wires:

1. The dealer D outputs a random authenticated secret sharing $[r]$
2. Alice and Bob run $(r, \perp) \leftarrow OpenRo(A, [r])$
3. Alice sends Bob $d = x - r$
4. Alice and Bob compute $[x] = [r] + d$

At the end, the offline and the online phase work like how we did in the class:

In the offline phase, we sampled a Beaver Triples and then we did secret sharing, and then we created the keys and the tags according to the values, and at the end, Alice and Bob sent each other their secret shares.

In the online phase, Alice and Bob share their input wires, and for every layer i, Alice and Bob compute the result pf the gates using the equation that we have written like Add and Mul (with constant or not), and at the end Alice and Bob verify he output, and they do Open and they return the output.

The privacy and the security of the protocols is known, because we have seen It in the lecture or we have proved it in the las assignment. It is because the major thing that we have done in this assignment is to enhance the BeDoZa protocol, that we have implied in assignment 3 and we have talked about in the class.