CS 203.4860	Secure Multi-Party Computation	Spring 2023

Homework 3: Report

Lecturer: Dr. Adi Akavia *Student(s): Nir Segal, Hallel Weinberg, Michael Rodel*

Abstract

In this homework we implemented the passively secure BeDoZa protocol for computing the function. We wrote code in Python that uses this protocol to compute the following function:

$$f_{\vec{a},4}(x) = \begin{cases} 1 & a_1x_1 + a_2x_2 \geq 4 \\ 0 & otherwise \end{cases}$$

Our notebook colab notebook is here:
`https://colab.research.google.com/drive/1hLscIzPuZ48IzVM1LelaTVKcdZszYTYj?usp=sharing`.

1 Introduction

Motivation. When 2 parties want to exchange information, they have to use a secure protocol to ensure that the information remains confidential and protected from any malicious actors who may try to intercept it. The BeDoZa protocol is a passively secure protocol that provides a way for two parties to securely exchange information.

Secure Computation Technique. The BeDoZa protocol achieves passive security by using a Boolean circuit, which is a circuit that describes the output of a function for all possible inputs. The two parties compute the output of the function by using Secret Sharing to not reveal their inputs. This ensures that even if an attacker is listening in, they cannot determine the inputs of the parties and therefore cannot compute the output.
The BeDoZa protocol achieves security against passive unbounded adversary by using a trusted dealer and circuit evaluation. The complexity of the protocol is $O(circuit - size)$.[Wik]
The protocol uses pre-processing which done by the trusted dealer.
Alice's input is written on an n-size wire $(x_1, ..., x_n)$, and Bob's input also written on an n-size wire $(x_n + 1, ..., x_{2n})$. The output wire is a L-size write. We represent the calculation in a tree (the leaves are the inputs and the root is the output. The nodes which are not the root or the leaves represent XOR or AND gates (both types are with constant or two wires). This tree contains d layers, s.t. inputs to gates at layer $i \in 1, .., d$ are from layers $< i$.
We secretly share wires' values between Alice and Bob by performing secret share input wires, that propagates secret sharing layer by layer, and once obtained a secret sharing of the output wire, open (=reconstruct).
The offline phase is done by the trusted dealer, with input $t \in N$, and repeats t times, as

follows:
1. Sample a "Beaver triples": $u, v \leftarrow_R 0, 1$ and $w = u \cdot v \ mod \ 2$.
2. Secret share:
$[u] := (u_A, u_B) \leftarrow Shr(u)$
$[v] := (v_A, v_B) \leftarrow Shr(v)$
$[w] := (w_A, wB) \leftarrow Shr(w)$
3. Send $(u_A, v_A, w_A) toAlice$ and (u_B, v_b, w_b) to Bob.
This is a secure operation because the dealer is a trusted, and a party can not change the triple she got from the dealer since it is not active (but passive). So, as a result, the offline phase is secure.

By using the passively secure BeDoZa protocol, two parties can exchange information securely from passive unbounded attackers. This is especially useful in scenarios where there is a risk of passive attacks. Then, we perform the online phase. Here, Alice A with input x and the shares (u_A, v_A, w_A), Bob B with input y and the shares (u_B, v_B, w_B).
The steps are:
1) Alice and Bob share their input wires:
$[xi] = (x_{iA}, x_{iB}) \leftarrow Share(A, x_i) for i = 1, .., n$
$[xi] = (x_{iA}, x_{iB}) \leftarrow Share(B, x_i) for i = n + 1, .., 2n$
2) For each circuit layer i = 1,. . . ,d:
Alice and Bob securely evaluate all gates in layer i using XOR and AND sub-protocols.
3) Alice and Bob reconstruct the output wire value x^L :
$(z, \perp) \leftarrow OpenTo(A, [x^L])$
Now, to show that that the online phase is secure, we will show that each sub-protocol in it is secure, and conclude that the whole online phase is secure.
First, step 1) consists only $Share(A, x_i)$ and $Share(B, x_i)$ sub-protocols. $Share(A, x_i)$ works as follows: Alice computes $(x_{iA}, x_{iB}) \leftarrow Shr(x_i)$ for $i = 1, .., n$ and sends x_{iB} to Bob. $Share(B, xi)$ is analogous for $i = n + 1, .., 2n$. Here, nothing can be assumed accept what we want to be assumed, since we do XOR with x_{iA} (and x_{iB}) with a uniform distributed value before sending it to the other party, so the other party can not tell what the value of the sending party was.
$OpenTo(A, [x])$ means that Bob sends x_b to Alice, and then she calculates the XOR of this x_b and her private data x_a, to obtain x. $OpenTo(B, [x])$ is analogous.$OpenTo([x])$ means we run both $OpenTo(A, [x])$ and $OpenTo(B, [x])$. Also here, nothing can be assumed accept what we want to be assumed, since we already open "everything", and the data can not be changed, since we assume that our adversary is passive (so he can not manipulate the data).
Now, we consider the privacy of evaluating XOR gates. Here, Alice outputs $z_B = x_A \oplus c$ (while c is written in the Boolean circuit, s.t. the dealer initializes it, but it also saved in Alice's and Bob's data), and Bob outputs $z_B = x_B$. If the XOR operation is done with a wire from previous layer, then Alice outputs $z_A = x_a \oplus y_A$ and Bob outputs $z_B = x_B \oplus y_B$. In both cases the privacy maintained, because the outputs reveal the whole x only if some party does XOR the received data with the party's private data, but it can't do so because we assume a passive adversary. So the XOR operation is privacy-preserving.
Now, we consider if evaluating AND gates is a privacy preserving operation. One version

of AND is $AND([x], c)$. Here Alice outputs $z_A = c \cdot x_A$, and Bob outputs $z_B = c \cdot x_B$. Like in XOR, this operation preserves privacy, since the outputs reveal the whole x only if some party does XOR the received data with the party's private data, but it can't do so because we assume a passive adversary. The Second version of AND gate is $AND([x], [y](.$ Here we output $[d]$, which we get from the output f $XOR([x], [u])$ and then open it using $Open([d])$. Same operation is done with $[e]$: we output $[e]$, which we get from the output f $XOR([t], [v])$ and then open it using $Open([e])$. As we have already showed, XOR and Open operations are privacy preserving operations, so as a result - the AND operation is also a privacy preserving operation.

Application. In our application we defined 3 classes: the dealer, Alice and Bob, with the first responsible for the offline phase and the last two for the online phase. We first randomly generate $\vec{a}, \vec{x}$ to be used as input to the protocol and then we communicate between the classes using a few lines of code:

1. We initialize the dealer and perform the offline phase.
2. We perform the online phase: we initialize Alice and Bob with their inputs $\vec{a}, \vec{x}$ and the outputs of the offline phase $(u_A, v_A, w_A), (u_B, v_B, w_B)$.
3. Alice and Bob share their inputs.
4. For all layer in the circuit:
5. Alice computes $z_A, ANDS$ according to the gates in the layer and sends $ANDS$ to Bob.
6. Bob computes z_B (he uses $ANDS$ only if the gate is $AND([x], [y])$) and sends to Alice.
7. Alice updates her z_A for all the $AND([x], [y])$ gates in this layer by using what Bob computed.

Finally, Alice reconstructs $z = z_A \oplus z_B$ and outputs z.
Note that actually Given $\vec{a}$ and $\vec{x}$, z is equal to the result of the function $f_{\vec{a},4}(x_1, x_2)$ for $\vec{a}, \vec{x}$ above.

Empirical Evaluation. We conducted experiments as follows: for each possible input $\vec{a} = (a_1, a_2), \vec{x} = (x_1, x_2)$ we computed using BeDoZa protocol an output z.
In short, the results we received correspond to the expected output of the function $f_{\vec{a},4}(x_1, x_2)$ for $\vec{a}, \vec{x}$ the aforementioned.

2 Preliminaries

In this homework we used the Boolean circuit we built in the first homework (drawings in figures 2, 3, 4, 5).

2.1 Against Passive Attacker

The definition of passive attacker is someone who can see all the web traffic of Alice and Bob, and can understand more than the things have been sent. Our goal is to unable his understandings beyond the traffic he sees.

2.2 Secret Sharing

To be able execute the protocol we need every party to have a secret. And if we reconstruct all these secrets we will get a valuable information. In our case if we will reconstruct all the secrets we will get all the initial inputs. In our case we have a dealer who spreads all these secrets to all the parties in a safe way, it means we can't infer from the traffic what is the combinations of the secrets.

3 Protocols

3.1 Secure Computation Technique: BeDoZa Protocol

Parties: Alice A, Bob B, Trusted dealer D.
Functionality: $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\} \times \perp$, $(x, y) \to (f(x, y), \perp)$.
Circuit: A Boolean circuit $C : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$.

L wires $x_1, \ldots, x_L$: $x_1, \ldots, x_n$- Alice's input. $x_{n+1}, ..., x_{2n}$- Bob's input. x_L- output wire.

d layers s.t. inputs to gates at layer $i \in \{1, .., d\}$ are from layers $< i$.

Gates: XOR with constant or of two wires. AND with constant or of two wires.

Wires' values are secret shared between Alice and Bob:

1. Secret share input wires.
2. Propagates secret sharing layer by layer.
3. Once obtained a secret sharing of the output wire, open (=reconstruct).

Notation: $[x]$ denotes a secret sharing of $x \in \{0,1\}$,

where Alice holds $x_A \in \{0,1\}$ and Bob holds $x_B \in \{0,1\}$
and where: (x_A, x_B) is uniform random in $\{0,1\}^2$ subject to: $x_A \oplus x_B = x$.

3.1.1 The Algorithm

Parties: Dealer D with input $t \in N$.

Algorithm 1 The Offline Phase | secret share beaver triples

1: **Repeat t times:**
2: Sample a "Beaver triples": $u, v \leftarrow_R \{0,1\}$ and $w = u \cdot v$.
3: Secret share:

$$[u] := (u_A, u_B) \leftarrow Shr(u)$$
$$[v] := (v_A, v_B) \leftarrow Shr(v)$$
$$[w] := (w_A, w_B) \leftarrow Shr(w)$$

4: Send (u_A, v_A, w_A) to Alice and (u_B, v_B, w_B) to Bob.

Parties: Alice A with input x and the shares (u_A, v_A, w_A), Bob B with input y and the shares (u_B, v_B, w_B).

Algorithm 2 The Online Phase | Securely evaluate a circuit C with $\#AND \leq t$

5: Alice and Bob share their input wires:

$$[x_i] = (x_{iA}, x_{iB}) \leftarrow Share(A, x_i) \text{ for } i = 1, .., n$$
$$[x_i] = (x_{iA}, x_{iB}) \leftarrow Share(B, x_i) \text{ for } i = n+1, .., 2n$$

6: For each circuit layer $i = 1, \ldots, d$, Alice and Bob securely evaluate all gates in layer i using XOR and AND sub-protocols (algorithm 5 and algorithm 6 correspondence).
7: Alice and Bob reconstruct the output wire value x^L: $(z, \perp) \leftarrow OpenTo(A, [x^L])$

3.1.2 Sub-Protocols

Algorithm 3 Sub-protocol | Sharing input wires

1: $Share(A, x_i)$**:** Alice computes $(x_{iA}, x_{iB}) \leftarrow Shr(x_i)$ and sends x_{iB} to Bob.
2: $Share(B, x_i)$**:** Bob computes $(x_{iA}, x_{iB}) \leftarrow Shr(x_i)$ and sends x_{iA} to Alice.

Algorithm 4 Sub-protocol | Opening secret shared values

1: $OpenTo(A, [x])$**:**
2: Bob sends x_B to Alice.
3: Alice outputs $x = x_A \oplus x_B$.
4: $OpenTo(B, [x])$**:**
5: Alice sends x_A to Bob.
6: Bob outputs $x = x_A \oplus x_B$.
7: $Open([x])$**:** Run both $OpenTo(B, [x])$ and $OpenTo(A, [x])$.

Algorithm 5 Sub-protocol | Evaluating XOR gates

1: $XOR([x], c)$**:**
2: Alice outputs $z_A = x_A \oplus c$.
3: Bob outputs $z_B = x_B$.
4: $XOR([x], [y])$**:**
5: Alice outputs $z_A = x_A \oplus y_A$.
6: Bob outputs $z_B = x_B \oplus y_B$.

Algorithm 6 Sub-protocol | Evaluating AND gates

1: $AND([x], c)$**:**
2: Alice outputs $z_A = c \cdot x_A$.
3: Bob outputs $z_B = c \cdot x_B$.
4: $AND([x], [y])$**:**
5: $[d] \leftarrow XOR([x], [u])$ and $d \leftarrow Open([d])$.
6: $[e] \leftarrow XOR([y], [v])$ and $e \leftarrow Open([e])$.
7: Compute $[z] = [w] \oplus (e \cdot [x]) \oplus (d \cdot [y]) \oplus (e \cdot d)$.

3.2 Application: BeDoZa Protocol

Algorithm 7 BeDoZa Protocol

1: 1. $n \leftarrow 4$. 2. $\#total\_layers \leftarrow 14$. 3. $\#AND\_gates \leftarrow 21$.
2: Generate two global arrays e, d of size $\#AND\_gates$.
3: Generate two random binary vectors $\vec{a} = (a_1, a_2), \vec{x} = (x_1, x_2)$ where: $a_1, a_2, x_1, x_2 \in \{0, 1, 2, 3\}$ and mark the binary digits of a_1, a_2, x_1, x_2 like this:

$$a_1 = a_{11}a_{10} \text{ and } a_2 = a_{21}a_{20}$$

$$x_1 = x_{11}x_{10} \text{ and } x_2 = x_{21}x_{20}$$

Algorithm 8 The Offline Phase | The dealer

4: **for all** $\#total\_layers$ **do**
5: Generate two arrays u, v (with $\#AND\_gates$ cells) of random values from $\{0, 1\}$.
6: Generate an array w such that $w = u \cdot v$.
7: Generate 3 arrays u_A, v_A, w_A (with $\#AND\_gates$ cells) of random values from $\{0, 1\}$.
8: Generate 3 arrays u_B, v_B, w_B such that $u = u_A \oplus u_B$, $v = v_A \oplus v_B$ and $w = w_A \oplus w_B$.
9: Generate a Boolean circuit (drawings in figures 2, 3, 4, 5) that represent the function:

$$f_{\vec{a},4}(x) = \begin{cases} 1 & a_1x_1 + a_2x_2 \geq 4 \\ 0 & otherwise \end{cases}$$

The circuit is represented by an array as follows:

each row i represents the ith layer in the circuit. Each member j in row i represents the j logical gate (from left to right) in layer i. define:

Each logic gate is marked as follows: (g, i, wire1, wire2) when:

- **g** is 0 if the gate is an xor and 1 if the gate is an and gate.

- **i** is 1 if wire2 is a fixed number (c) and 0 otherwise.

- **wire1** is the index of the first wire (for the array of gate results from the previous layers) that enters the gate.

- **wire2** is the second wire of the gate: can be a fixed number (0 or 1) or the *index* of the second wire, depending on the value of i.

10: Send (u_A, v_A, w_A) to Alice and (u_B, v_B, w_B) to Bob.
11: **end for**

Algorithm 9 The Online Phase | Alice and Bob

Alice and Bob share their input wires:

1. Alice computes (x_{iA}, x_{iB}) for $i = 1, .., n$ and sends x_{iB} to Bob.

   Randomly generate x_{iA} and compute $x_{iB} = x \oplus x_{iA}$ for $i = 1, .., n$.

2. Bob computes (x_{iA}, x_{iB}) for $i = n+1, .., 2n$ and sends x_{iA} to Alice.

   Randomly generate x_{iB} and compute $x_{iA} = y \oplus x_{iB}$ for $i = n+1, .., 2n$.

NOTE In short, Alice holds an array A of size $2n$ and Bob holds an array B of size $2n$ so that $A \oplus B = [x, y]$ i.e. $A \oplus B$ is an array of $x \circ y$.

$number\_of\_layer \leftarrow 0$.
$number\_of\_curr\_AND \leftarrow 0$.
while $number\_of\_layer < \#total\_layers$ **do**

Algorithm 10 Alice computes two arrays $z_A, ANDS$ and sends $ANDS$ to Bob

for i between 0 and number of gates in the $number\_of\_layer$-th layer **do**
  if the i-th gate is $XOR([x], c)$ **then**
    $z_A[i] = x_A \oplus c$
  end if
  if the i-th gate is $XOR([x], [y])$ **then**
    $z_A[i] = x_A \oplus y_A$
  end if
  if the i-th gate is $AND([x], c)$ **then**
    $z_A[i] = x_A \cdot c$
  end if
  if the i-th gate is $AND([x], [y])$ **then**
    $q \leftarrow number\_of\_curr\_AND$
    In order to compute e, d: $ANDS[i] = [x_A \oplus u_A[q], y_A \oplus v_A[q], q]$.
    $number\_of\_curr\_AND+ = 1$.
  end if
end for

Algorithm 11 Bob computes and sends z_B to Alice

for i between 0 and number of gates in the $number\_of\_layer$-th layer **do**
  if the i-th gate is $XOR([x], c)$ **then**
    $z_B[i] = x_B$
  end if
  if the i-th gate is $XOR([x], [y])$ **then**
    $z_B[i] = x_B \oplus y_B$
  end if
  if the i-th gate is $AND([x], c)$ **then**
    $z_B[i] = x_B \cdot c$
  end if
  if the i-th gate is $AND([x], [y])$ **then**
    $q \leftarrow ANDS[i][2]$.
    $d[i] = ANDS[i][0] \oplus (x_B \oplus u_B[q])$. ▷ $d[i] = (x_A \oplus u_A[q]) \oplus (x_B \oplus u_B[q])$
    $e[i] = ANDS[i][1] \oplus (y_B \oplus v_B[q])$. ▷ $e[i] = (y_A \oplus v_A[q]) \oplus (y_B \oplus v_B[q])$
    $z_B[i] = w_B \oplus (e[i] \cdot x_B) \oplus (d[i] \cdot y_B) \oplus (e[i] \cdot d[i])$.
  end if
end for

Algorithm 12 Alice receives z_B from Bob

for i between 0 and number of gates in the $number\_of\_layer$-th layer **do**
  if the i-th gate is $AND([x], [y])$ **then**
    $q \leftarrow ANDS[i][2]$
    $z_A[i] = w_A \oplus (e[i] \cdot x_A) \oplus (d[i] \cdot y_A)$.
  end if
end for
$number\_of\_layer+ = 1$.
end while

Algorithm 13 End Of The Protocol

Alice outputs $z = z_A \oplus z_B$.

4 Implementation

The code is written in Python. Numpy library is the only one required to run the code. Our code generates 2 random inputs $\vec{a} = (a_1, a_2)$ and $\vec{x} = (x_1, x_2)$ and returns an output z, the result of $f_{\vec{a},4}(x_1, x_2)$.
The code is here:
`https://colab.research.google.com/drive/1hLscIzPuZ48IzVM1LelaTVKcdZszYTYj?usp=sharing`.

5 Empirical Evaluation

Each row in the table is an experiment: we conducted 256 experiments on all possible inputs of a_1, a_2 and x_1, x_2 for which we got output z:

Table 1: Our experiments.

Begin of Table				
x_1	x_2	a_1	a_2	z
0	0	0	0	0
0	0	0	1	0
0	0	0	2	0
0	0	1	0	0
0	0	2	0	0
0	0	0	3	0
0	0	1	1	0
0	0	2	1	0
0	0	1	2	0
0	0	2	2	0
0	0	3	0	0
0	0	1	3	0
0	0	2	3	0
0	0	3	1	0
0	0	3	2	0
0	0	3	3	0
0	1	0	0	0
0	1	0	1	0
0	1	0	2	0
0	1	1	0	0
0	1	2	0	0
0	1	0	3	0
0	1	1	1	0
0	1	2	1	0
0	1	1	2	0
0	1	2	2	0

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
0	1	3	0	0
0	1	1	3	0
0	1	2	3	0
0	1	3	1	0
0	1	3	2	0
0	1	3	3	0
0	2	0	0	0
0	2	0	1	0
0	2	0	2	1
0	2	1	0	0
0	2	2	0	0
0	2	0	3	1
0	2	1	1	0
0	2	2	1	0
0	2	1	2	1
0	2	2	2	1
0	2	3	0	0
0	2	1	3	1
0	2	2	3	1
0	2	3	1	0
0	2	3	2	1
0	2	3	3	1
1	0	0	0	0
1	0	0	1	0
1	0	0	2	0
1	0	1	0	0
1	0	2	0	0
1	0	0	3	0
1	0	1	1	0
1	0	2	1	0
1	0	1	2	0
1	0	2	2	0
1	0	3	0	0
1	0	1	3	0
1	0	2	3	0
1	0	3	1	0
1	0	3	2	0
1	0	3	3	0
2	0	0	0	0
2	0	0	1	0
2	0	0	2	0

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
2	0	1	0	0
2	0	2	0	1
2	0	0	3	0
2	0	1	1	0
2	0	2	1	1
2	0	1	2	0
2	0	2	2	1
2	0	3	0	1
2	0	1	3	0
2	0	2	3	1
2	0	3	1	1
2	0	3	2	1
2	0	3	3	1
0	3	0	0	0
0	3	0	1	0
0	3	0	2	1
0	3	1	0	0
0	3	2	0	0
0	3	0	3	1
0	3	1	1	0
0	3	2	1	0
0	3	1	2	1
0	3	2	2	1
0	3	3	0	0
0	3	1	3	1
0	3	2	3	1
0	3	3	1	0
0	3	3	2	1
0	3	3	3	1
1	1	0	0	0
1	1	0	1	0
1	1	0	2	0
1	1	1	0	0
1	1	2	0	0
1	1	0	3	0
1	1	1	1	0
1	1	2	1	0
1	1	1	2	0
1	1	2	2	1
1	1	3	0	0
1	1	1	3	1

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
1	1	2	3	1
1	1	3	1	1
1	1	3	2	1
1	1	3	3	1
2	1	0	0	0
2	1	0	1	0
2	1	0	2	0
2	1	1	0	0
2	1	2	0	1
2	1	0	3	0
2	1	1	1	0
2	1	2	1	1
2	1	1	2	1
2	1	2	2	1
2	1	3	0	1
2	1	1	3	1
2	1	2	3	1
2	1	3	1	1
2	1	3	2	1
2	1	3	3	1
1	2	0	0	0
1	2	0	1	0
1	2	0	2	1
1	2	1	0	0
1	2	2	0	0
1	2	0	3	1
1	2	1	1	0
1	2	2	1	1
1	2	1	2	1
1	2	2	2	1
1	2	3	0	0
1	2	1	3	1
1	2	2	3	1
1	2	3	1	1
1	2	3	2	1
1	2	3	3	1
2	2	0	0	0
2	2	0	1	0
2	2	0	2	1
2	2	1	0	0
2	2	2	0	1

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
2	2	0	3	1
2	2	1	1	1
2	2	2	1	1
2	2	1	2	1
2	2	2	2	1
2	2	3	0	1
2	2	1	3	1
2	2	2	3	1
2	2	3	1	1
2	2	3	2	1
2	2	3	3	1
3	0	0	0	0
3	0	0	1	0
3	0	0	2	0
3	0	1	0	0
3	0	2	0	1
3	0	0	3	0
3	0	1	1	0
3	0	2	1	1
3	0	1	2	0
3	0	2	2	1
3	0	3	0	1
3	0	1	3	0
3	0	2	3	1
3	0	3	1	1
3	0	3	2	1
3	0	3	3	1
1	3	0	0	0
1	3	0	1	0
1	3	0	2	1
1	3	1	0	0
1	3	2	0	0
1	3	0	3	1
1	3	1	1	1
1	3	2	1	1
1	3	1	2	1
1	3	2	2	1
1	3	3	0	0
1	3	1	3	1
1	3	2	3	1
1	3	3	1	1

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
1	3	3	2	1
1	3	3	3	1
2	3	0	0	0
2	3	0	1	0
2	3	0	2	1
2	3	1	0	0
2	3	2	0	1
2	3	0	3	1
2	3	1	1	1
2	3	2	1	1
2	3	1	2	1
2	3	2	2	1
2	3	3	0	1
2	3	1	3	1
2	3	2	3	1
2	3	3	1	1
2	3	3	2	1
2	3	3	3	1
3	1	0	0	0
3	1	0	1	0
3	1	0	2	0
3	1	1	0	0
3	1	2	0	1
3	1	0	3	0
3	1	1	1	1
3	1	2	1	1
3	1	1	2	1
3	1	2	2	1
3	1	3	0	1
3	1	1	3	1
3	1	2	3	1
3	1	3	1	1
3	1	3	2	1
3	1	3	3	1
3	2	0	0	0
3	2	0	1	0
3	2	0	2	1
3	2	1	0	0
3	2	2	0	1
3	2	0	3	1
3	2	1	1	1

Continuation of Table 1				
x_1	x_2	a_1	a_2	z
3	2	2	1	1
3	2	1	2	1
3	2	2	2	1
3	2	3	0	1
3	2	1	3	1
3	2	2	3	1
3	2	3	1	1
3	2	3	2	1
3	2	3	3	1
3	3	0	0	0
3	3	0	1	0
3	3	0	2	1
3	3	1	0	0
3	3	2	0	1
3	3	0	3	1
3	3	1	1	1
3	3	2	1	1
3	3	1	2	1
3	3	2	2	1
3	3	3	0	1
3	3	1	3	1
3	3	2	3	1
3	3	3	1	1
3	3	3	2	1
3	3	3	3	1
End of Table				

Note that for each a_1, a_2, x_1, x_2 we got z which corresponds to the result of the function $f_{\vec{a},4}(x_1, x_2)$ for these a_1, a_2, x_1, x_2:

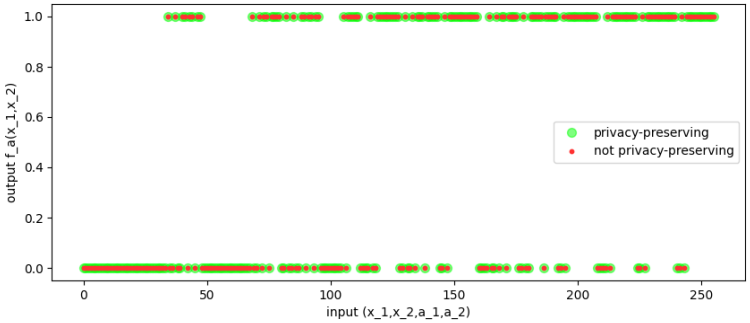

Figure 1: Comparison between privacy-preserving computation and not privacy-preserving computation

Note: You can see our tests file here: `https://colab.research.google.com/drive/1HWgjpiOMu5YAHWxTD9sM7uYfV5SdF6I6?usp=sharing` and the Benchmark tests file here: `https://colab.research.google.com/drive/1zq_n3P6MT9c-BJIWHfyoJmNs5ly9oy7l?usp=sharing`.

6 Conclusions

As shown in the results in the previous seciton, We see that the proposed approach yields correct results for $f_{\vec{a},4}(x_1, x_2)$. Therefore, the output correctness of the proposed approach is not compromised by considering privacy.
By using BeDoZa protocol, we were able to maintain participants' (Alice and Bob) privacy because the private data didn't need to be disclosed for computations.

References

[Wik] Wikipedia. Adversary.

Appendices

A Drawings of the Boolean circuit from homework 1

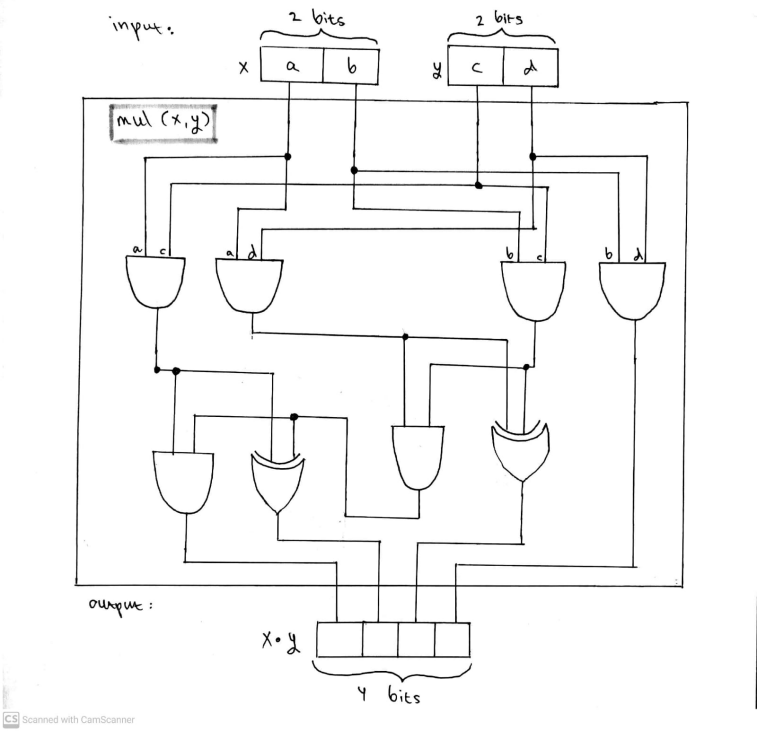

Figure 2: A black box of multiplication

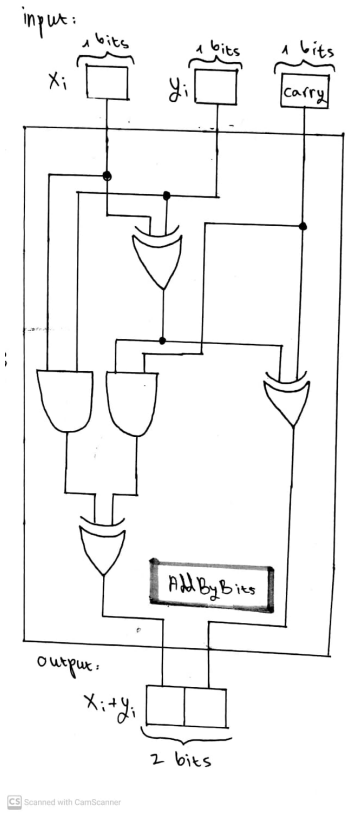

Figure 3: A black box of addition of 2 bits with consideration for carry

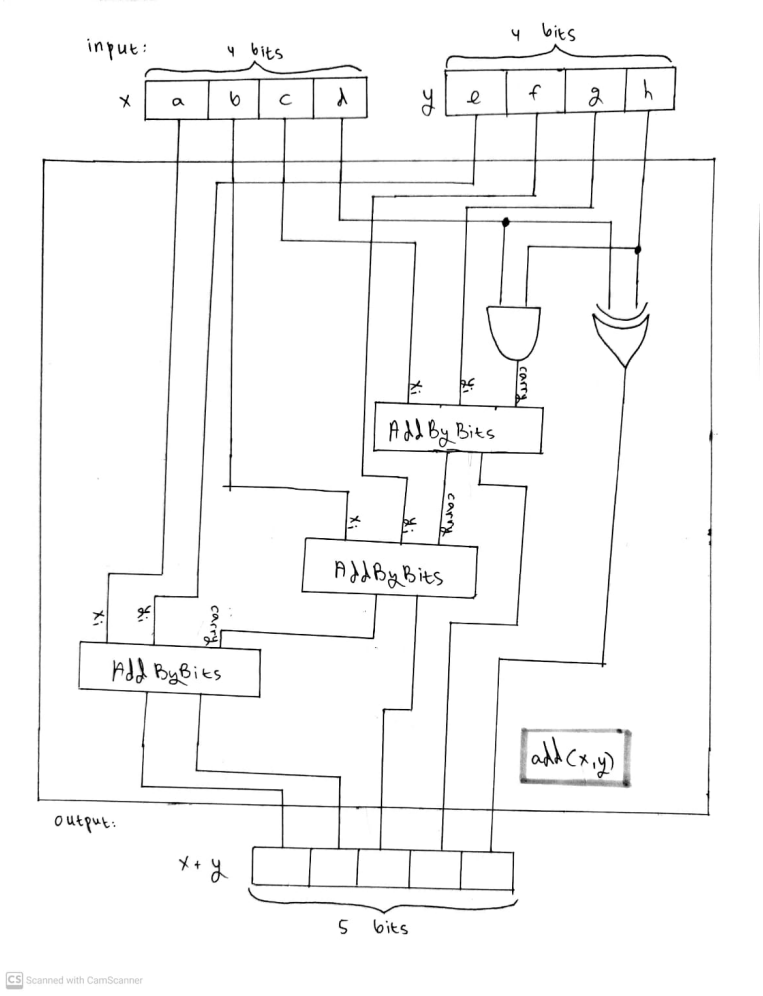

Figure 4: A black box of addition of 2 numbers

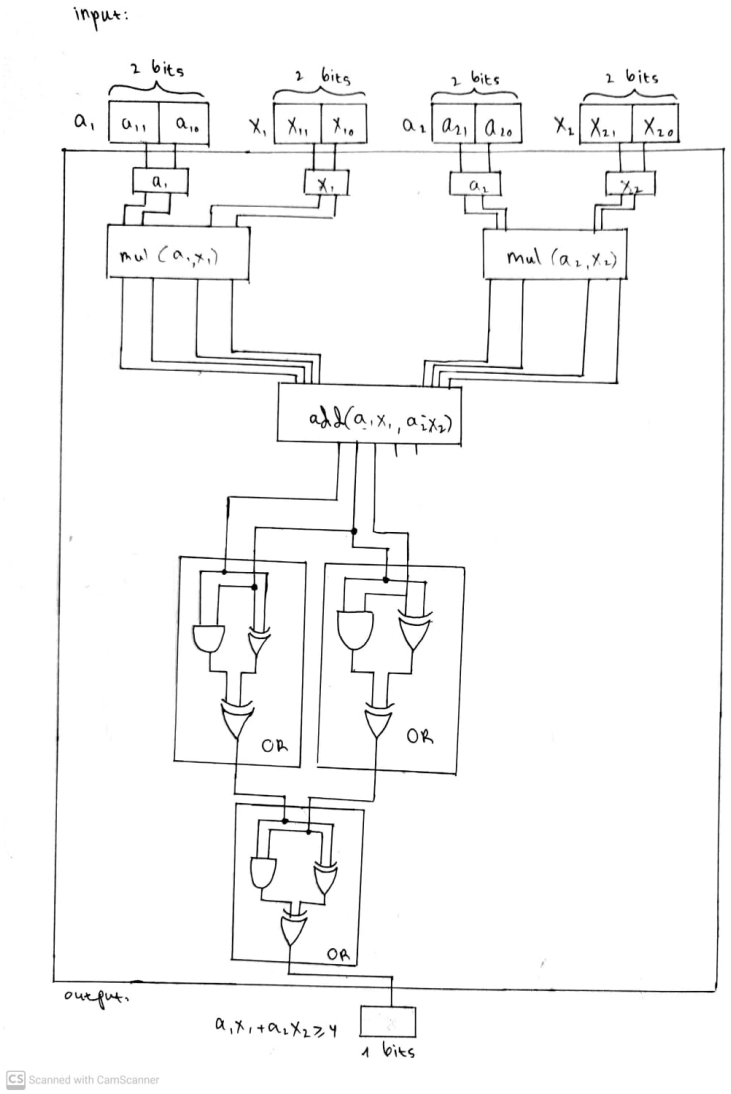

Figure 5: The entire Boolean circuit