CS 203.4860 **Secure Multi-Party Computation** Spring 2023

Homework 2: Report

Lecturer: Dr. Adi Akavia *Student(s): Yara Shamali, Samih Warwar, Maias Omar*

Abstract

In this article we will implement the passively secure BeDOZa protocol for computing a specific function .

1 Introduction

BeDOZa is an encryption scheme used for securing data stored in databases. It uses random noise to ensure the confidentiality of the data. BeDOZa is passively secure, meaning attackers who gain access to the encrypted data cannot learn any information about the plaintext.

Motivation. The motivation behind this technique is to enable secure communication between parties by ensuring that both parties have a shared understanding of the cryptographic algorithm or function being used for encryption or decryption without uncovering private data .

Secure Computation Technique. here we will explore and implement a secure two-party protocol for computing the function specified in the euqation below using the passively secure BeDOZa protocol.

Application. we implemented BeDOZa to solve the equation down below , using key techniques that include : an offline phase which is the process of generating Beaver triples the u v and w for a specific cryptographic function or algorithm before any communication or encryption takes place.
online phase : which is the process of using the precomputed values to encrypt or decrypt the data in real-time using the subprotocols AND, XOR and Open To.

Empirical Evaluation. The BeDOZa protocol has some type of limitation such as its vulnerability to certain types of attacks, particularly those that target the random noise added during the encryption process.Nonetheless, overall, BeDOZa is a promising approach to database encryption that offers passively secure protection for sensitive data.

$$f_{\vec{a},4}(x_1,x_2) = \begin{cases} 1 & \text{if } a_1x_1 + a_2x_2 \geq 4 \\ 0 & \text{otherwise} \end{cases}$$

2 Preliminaries

The circuit we already built in the first assignment and the Secret sharing.

3 Protocols

We will implement the passively secure BeDOZa that it has two phases the offline phase and the online phase using three sub protocols AND, XOR and OpenTo.Specifically in this assignment we will suppose that we have three parties Alice, Bob and the Dealer.

3.1 Offline phase

let T be the number of the AND gates in our circuit.
Repeat this steps T times:
1.Creating the Beaver triples

$$u,v \leftarrow_R \{0,1\} \text{ and } w = u \cdot v \mod 2$$

2.Secret share

$$\begin{aligned} [u] &:= (u_A, u_B) && \leftarrow Shr(u) \\ [v] &:= (v_A, v_B) && \leftarrow Shr(v) \\ [w] &:= (w_A, w_B) && \leftarrow Shr(w) \end{aligned}$$

3.Send
Sending (Ua,Va,Wa) to Alice and (Ub,Vb,Wb) to Bob

3.2 Online phase

We will propagate the secret sharing layer by layer
First of all Alice and Bob will share there input wires:

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

now for each circuit layer i=1,...,d Alice and Bob securely evaluate all gates in layer i using XOR and AND subprotocols.

And finally Alice and Bob reconstruct the output wire value xL:

$$(z, \perp) \leftarrow \text{OpenTo}(\text{ A }, [x^L])$$

Now we will define the sub protocols AND XOR and openTo
1.OpenTo:

OpenTo(A, [x]):	Bob sends x_B to Alice Alice outputs $x = x_A \oplus x_B$.
OpenTo(B, [x]):	Analogous.
Open([x]):	Run both OpenTo(B, [x]) and OpenTo(A, [x]).

2.XOR:

XOR([x],c): Alice outputs $z_A = x_A \oplus \mathbf{c}$
Bob outputs $z_B = x_B$

XOR([x],[y]): Alice outputs $z_A = x_A \oplus y_A$
Bob outputs $z_B = x_B \oplus y_B$

3.AND:

AND([x],c): Alice outputs $z_A = c \cdot x_A$
Bob outputs $z_B = c \cdot x_B$

AND([x],[y]):

[d] $\leftarrow$ XOR([x],[u])
d $\leftarrow$ **Open**([d])

[e] $\leftarrow$ XOR([y],[v])
e $\leftarrow$ **Open**([e])

Compute: $[z] = [w] \oplus (e \cdot [x]) \oplus (d \cdot [y]) \oplus (e \cdot d)$
(using subprotocols XOR([·],[·]), AND([·],·))

4 Implementation

In our code u can choose the four values a1,a2,x1,x2 E0,1,2,3 and then youll see the offline phase where the Dealer generate the Beaver triples for Alice and Bob and then the Online phase where we run the circuit layer by layer using the subprotocols.

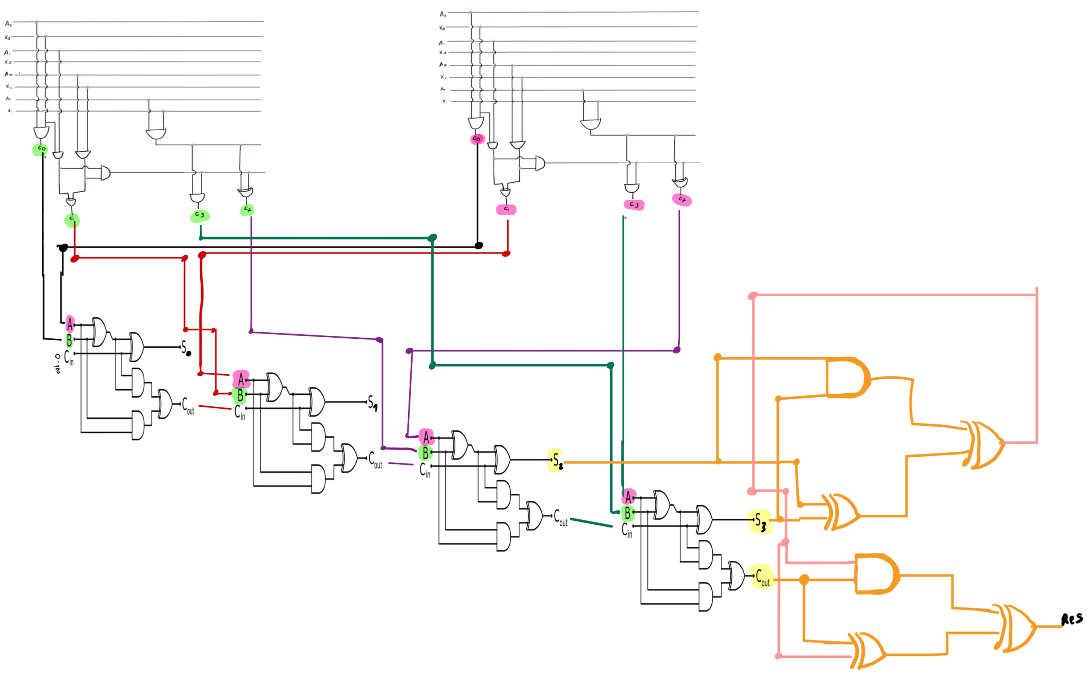
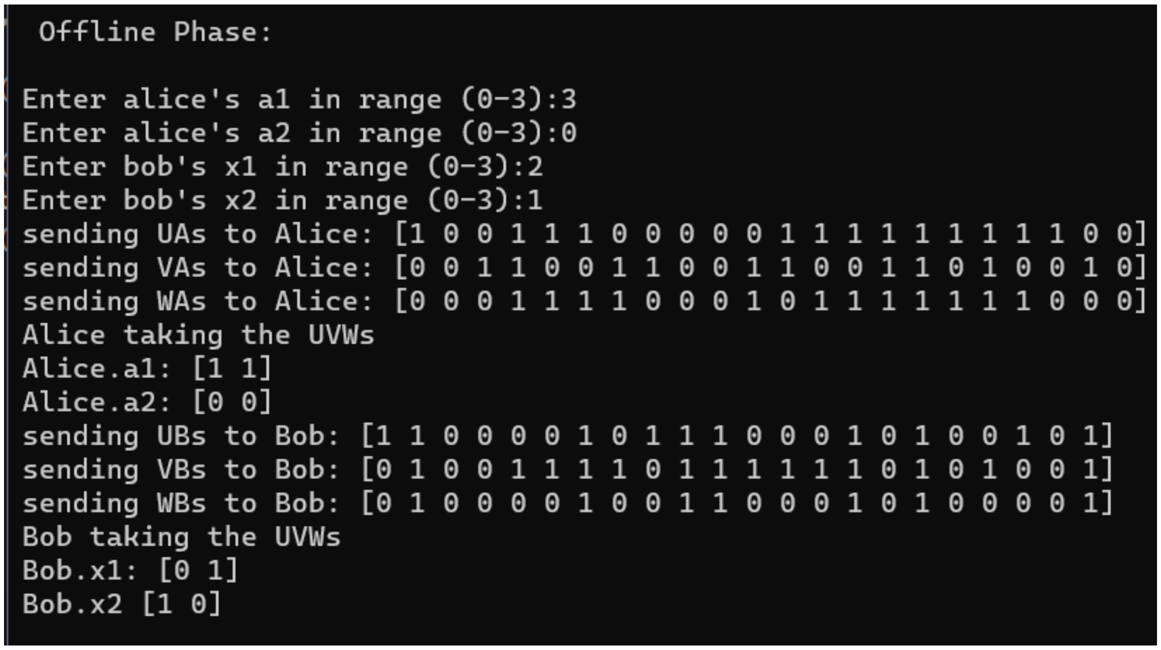
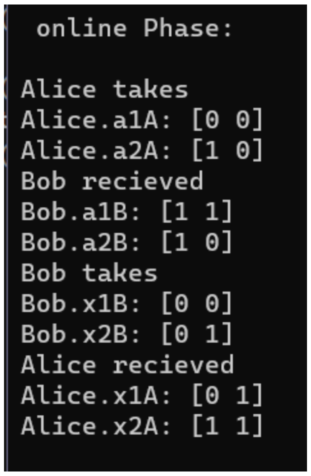


Figure 1: we will use this circuit from assignment 1

5 Empirical Evaluation

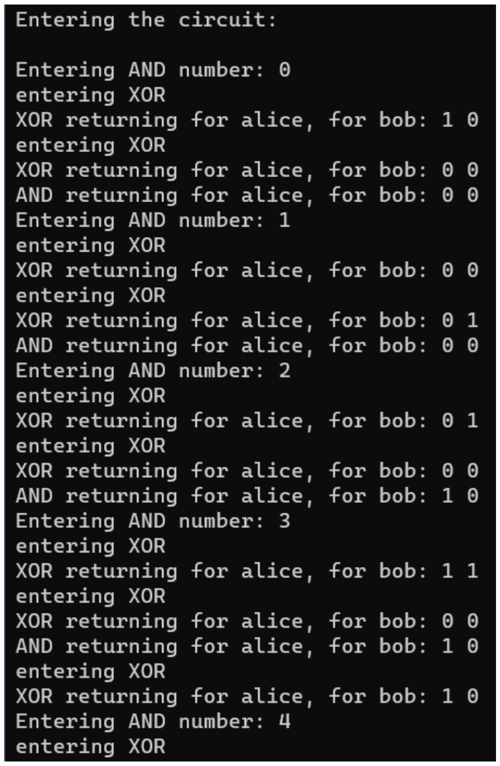


```
 Offline Phase:

Enter alice's a1 in range (0-3):3
Enter alice's a2 in range (0-3):0
Enter bob's x1 in range (0-3):2
Enter bob's x2 in range (0-3):1
sending UAs to Alice: [1 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0]
sending VAs to Alice: [0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 0]
sending WAs to Alice: [0 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 0]
Alice taking the UVWs
Alice.a1: [1 1]
Alice.a2: [0 0]
sending UBs to Bob: [1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 1 0 1]
sending VBs to Bob: [0 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1]
sending WBs to Bob: [0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1]
Bob taking the UVWs
Bob.x1: [0 1]
Bob.x2 [1 0]
```



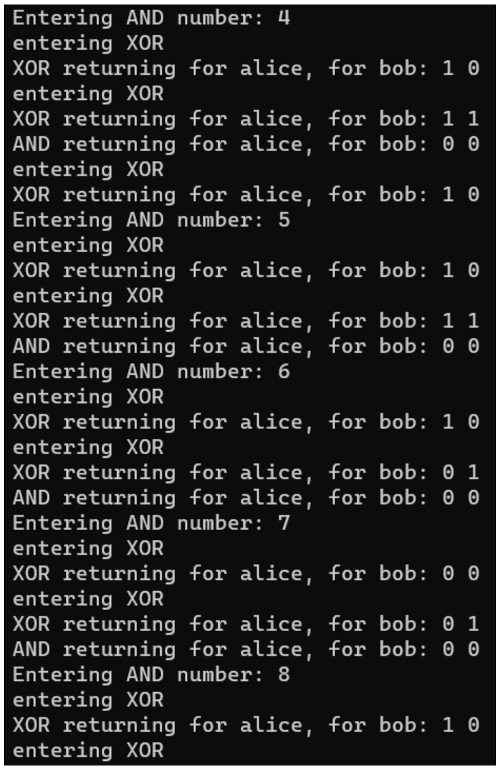
```
 online Phase:

Alice takes
Alice.a1A: [0 0]
Alice.a2A: [1 0]
Bob recieved
Bob.a1B: [1 1]
Bob.a2B: [1 0]
Bob takes
Bob.x1B: [0 0]
Bob.x2B: [0 1]
Alice recieved
Alice.x1A: [0 1]
Alice.x2A: [1 1]
```

(a)



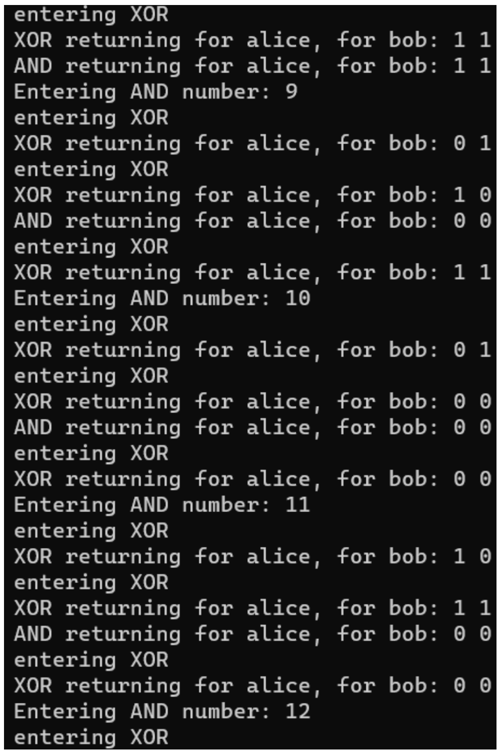
```
Entering the circuit:

Entering AND number: 0
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 0 0
Entering AND number: 1
entering XOR
XOR returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 0 1
AND returning for alice, for bob: 0 0
Entering AND number: 2
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 1 0
Entering AND number: 3
entering XOR
XOR returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0
Entering AND number: 4
entering XOR
```

(b)



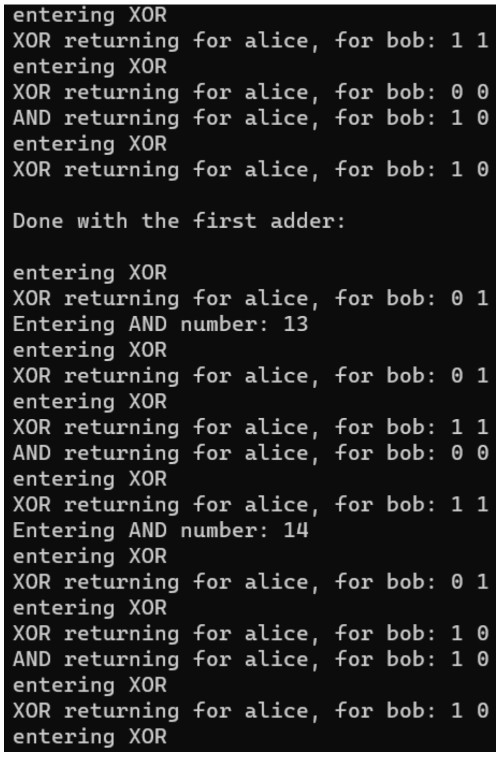
```
Entering AND number: 4
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 1 0
Entering AND number: 5
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 0 0
Entering AND number: 6
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 0 1
AND returning for alice, for bob: 0 0
Entering AND number: 7
entering XOR
XOR returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 0 1
AND returning for alice, for bob: 0 0
Entering AND number: 8
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
```

(a)



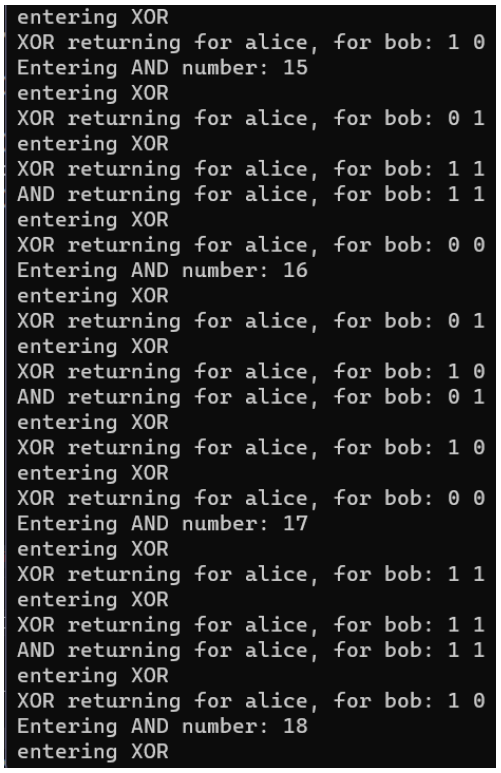
```
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 1 1
Entering AND number: 9
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 1 1
Entering AND number: 10
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 11
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 12
entering XOR
```

(b)



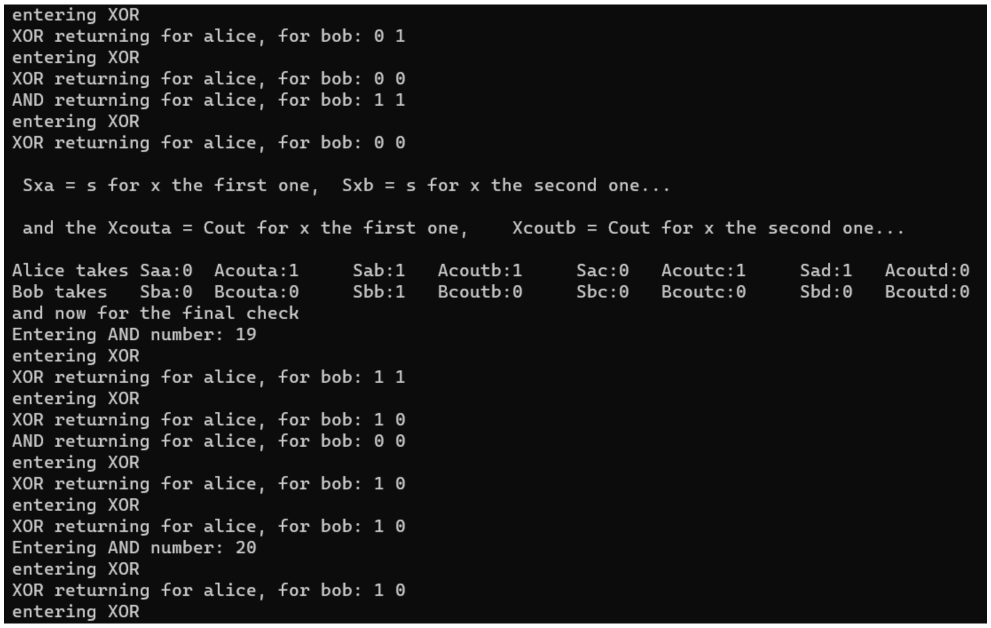
```
entering XOR
XOR returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0

Done with the first adder:

entering XOR
XOR returning for alice, for bob: 0 1
Entering AND number: 13
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 1 1
Entering AND number: 14
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
```

(a)



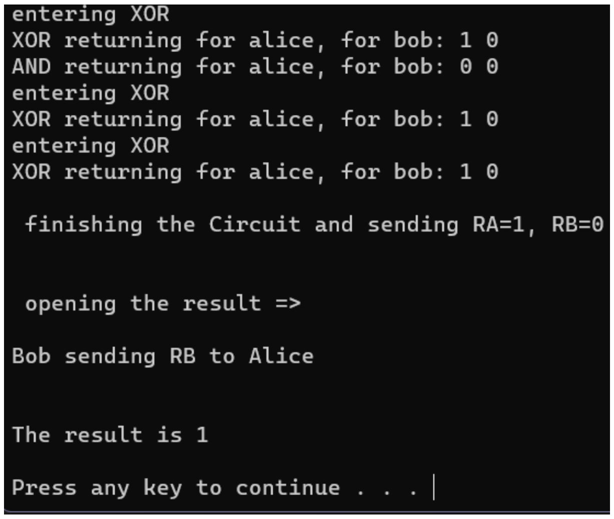
```
entering XOR
XOR returning for alice, for bob: 1 0
Entering AND number: 15
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 16
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 17
entering XOR
XOR returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 1 1
AND returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 1 0
Entering AND number: 18
entering XOR
```

(b)



```
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 0 0
AND returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 0 0

 Sxa = s for x the first one,  Sxb = s for x the second one...

 and the Xcouta = Cout for x the first one,    Xcoutb = Cout for x the second one...

Alice takes Saa:0  Acouta:1     Sab:1   Acoutb:1     Sac:0   Acoutc:1     Sad:1   Acoutd:0
Bob takes   Sba:0  Bcouta:0     Sbb:1   Bcoutb:0     Sbc:0   Bcoutc:0     Sbd:0   Bcoutd:0
and now for the final check
Entering AND number: 19
entering XOR
XOR returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0
Entering AND number: 20
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
```

(a)



```
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 0 0
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0

 finishing the Circuit and sending RA=1, RB=0


 opening the result =>

Bob sending RB to Alice


The result is 1

Press any key to continue . . .
```

(b)

6 Conclusions

BeDOZa is Perfectly secure against passive unbounded adversary and we only need the Dealer (that trusted party) in the offline phase.In addition it has Optimal computational complexity but when it comes to communication and Storage complexity it will go up to the number of the AND gates and Round complexity (number of the layers in the circuit.