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 but the main purpose is to eliminate the need for the dealer using 1 out of 2 Oblivious Transfer.

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 in top of that in this assignment we will get rid of the dealer.

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 with the help of the 1 out of 4 OT using the 1 out of 2 OT .

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 and now we'll use our OT to eliminate the need for the dealer in the protocol that we implemented in the previous assignment.
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 ,Secret sharing, Beaver Triples and DDH assumption.

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 in addition we will use the 1 out of 2 OT in the offline phase.Specifically in this assignment we will suppose that we have two parties Alice and Bob.

3.1 Offline phase

let T be the number of the AND gates in our circuit.
we will prepare al the UVWs for both Bob and Alice Repeat these steps T times:
1. Alice samples Ua,Va. And Bob samples Ub,Vb and Wb when:

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

2.Bob and Alice engage in a 1 out of 4 OT protocol, where:

Alice (Receiver) has selection value: $i = 2u_A + v_A$ //$i \in \{0,1,2,3\}$

Bob (Sender) has 4 messages:

$m_0 = (0 \oplus u_B) \cdot (0 \oplus v_B) \oplus w_B$
$m_1 = (0 \oplus u_B) \cdot (1 \oplus v_B) \oplus w_B$
$m_2 = (1 \oplus u_B) \cdot (0 \oplus v_B) \oplus w_B$
$m_3 = (1 \oplus u_B) \cdot (1 \oplus v_B) \oplus w_B$

Upon termination: Alice has $m_i = (u_A \oplus u_B) \cdot (v_A \oplus v_B) \oplus w_B$

Bob learned nothing on u_A, v_A

Alice sets $\mathbf{w_A} := \mathbf{m_i}$

To implement the 1 out of 4 OT we will use the 1 out of 2 OT 4 times when Alice the receiver with input i and Bob is the Sender with input m0 m1 m2 m3:

1. Run 1-out-of-2 OT on	R's input 1 if i=0 (0 o/w)	S's input (r_0 , m_0)
2. Run 1-out-of-2 OT on	R's input 1 if i=1 (0 o/w)	S's input $(r_1 , m_1 \oplus r_0)$
3. Run 1-out-of-2 OT on	R's input 1 if i=2 (0 o/w)	S's input $(r_2 , m_2 \oplus r_0 \oplus r_1)$
4. Run 1-out-of-2 OT on	R's input 1 if i=3 (0 o/w)	S's input $(r_3 , m_3 \oplus r_0 \oplus r_1 \oplus r_2)$

and at the end Alice will do XOR between every result she got from 1 until the time when i=1 (including the edges)

*1 out of 2 OT using ELGamal:

Receiver: Send (pk_0 , pk_1) to Sender, where

$\mathbf{pk_b} := pk$ for $(pk,sk) \leftarrow Gen(1^\kappa)$ and

$\mathbf{pk_{1-b}} := pk'$ for $pk' \leftarrow OGen(r)$

Sender: Send (c_0, c_1) to Receiver, where $\mathbf{c_0} \leftarrow Enc_{pk0}(m_0; r_0)$ and $\mathbf{c_1} \leftarrow Enc_{pk1}(m_1; r_1)$

Receiver: outputs $\mathbf{m_b} = Dec_{sk}(\mathbf{c_b})$

when:

$Gen(1^\kappa)$: $sk = \alpha \leftarrow_R \{0,..,q-1\}$
$pk = (g,h)$ for $h = g^\alpha$
output (pk,sk)

$Enc_{pk}(m)$: on input a message $m \in G$, and $pk = (g,h)$
sample $r \leftarrow_R \{0,..,q-1\}$
output $C = (g^r , m \cdot h^r)$.

$Dec_{sk}(C)$: on input $C = (c_1,c_2)$
output $m = c_2 \cdot c_1^{-\alpha}$

G of order q where DDH believed to hold:

- Let p be a prime number s.t. $p = 2q + 1$.
- Let $G \subseteq Z_p^*$ be the order q subgroup consisting of all quadratic residues mod p.

OGen(r): Use random coins r to sample $s \leftarrow_R \{0,..,p-1\}$
Output $h = s^2 \bmod p$

Note that in this assignment we chose that q=131 therefore p=263 and g=2 and these parameters define the subgroup G: 1, 2, 4, 8, 16, 32, 64, 128, 256, 123, 246, 125, 250, 133, 266, 135, 270, 139, 278, 141, 282, 143, 286, 147, 294, 151, 302, 155, 310, 157, 314, 161, 322, 163, 326, 167, 334, 169, 338, 171, 342, 175, 350, 177, 354, 179, 358, 181, 362, 185, 370, 187, 374, 191, 382, 193, 386, 197, 394, 199, 398, 203, 406, 205, 410, 207, 414, 211, 422, 215, 430, 217, 434, 219, 438, 223, 446, 227, 454, 229, 458, 233, 466, 235, 470, 239, 478, 241, 482, 245, 490, 247, 494, 251, 502, 253, 506, 257, 514, 259, 518, 263

We defined this group by going through every number y (1,p-1) and compute x = power(y,2) (mod p)

3.2 Online phase

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

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

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:

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

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 $z_A = x_A \oplus c$
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] ← XOR([x],[u])
d ← **Open**([d])

[e] ← XOR([y],[v])
e ← **Open**([e])

Compute: [z] = [w] ⊕ (e · [x]) ⊕ (d · [y]) ⊕ (e ·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 you'll 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.

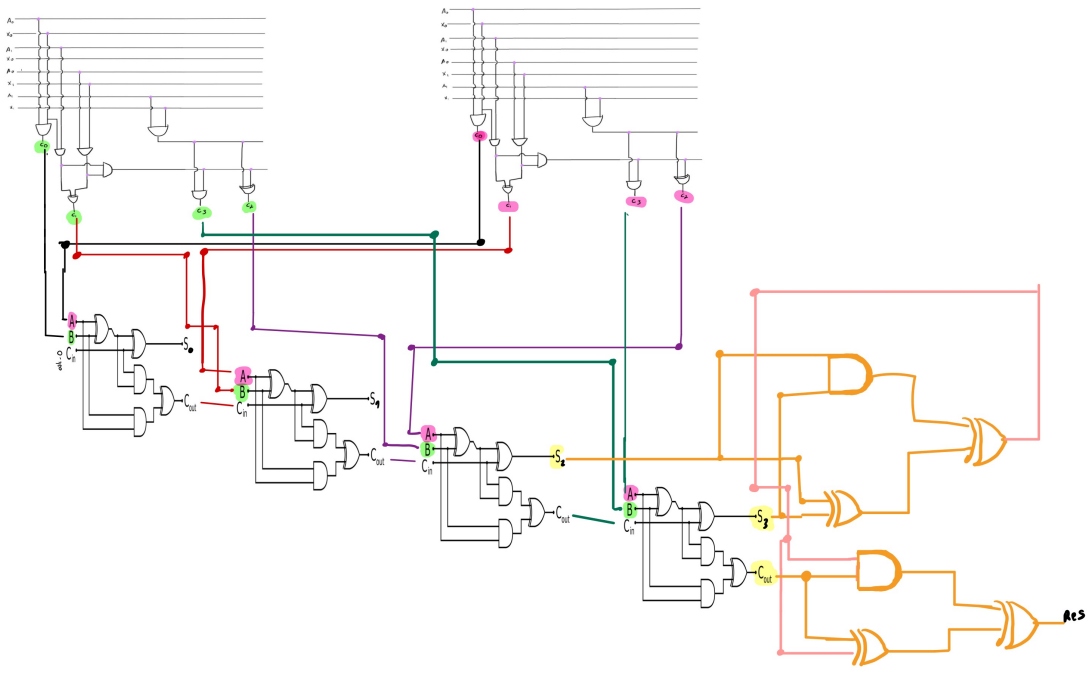


Figure 1: we will use this circuit from assignment 1

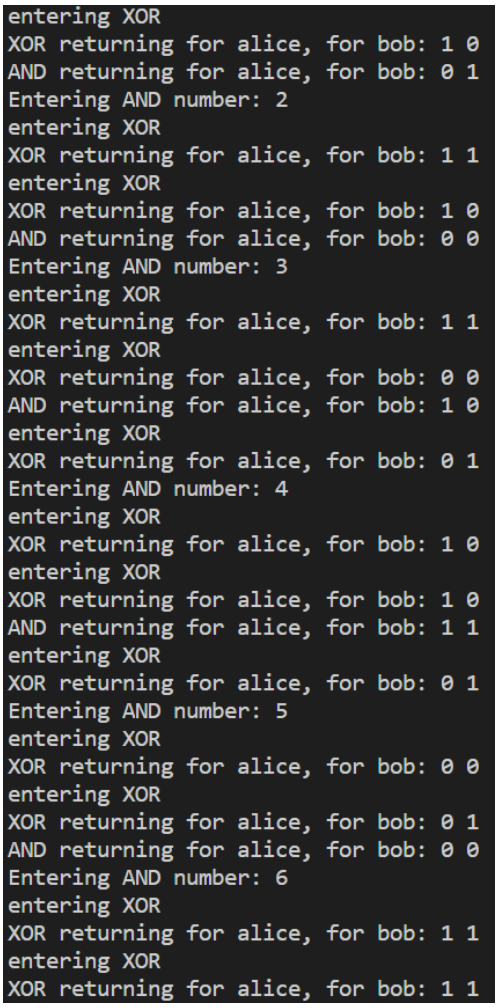
5 Empirical Evaluation

```
 Offline Phase:

Enter alice's a1 in range (0-3):2
Enter alice's a2 in range (0-3):3
Enter bob's x1 in range (0-3):3
Enter bob's x2 in range (0-3):2
Alice making the UVWs
Alice.a1: [0 1]
Alice.a2: [1 1]
Bob making the UVWs
Bob.x1: [1 1]
Bob.x2 [0 1]
Alice filling WAs[0]=1
Alice filling WAs[1]=1
Alice filling WAs[2]=0
Alice filling WAs[3]=1
Alice filling WAs[4]=0
Alice filling WAs[5]=1
Alice filling WAs[6]=0
Alice filling WAs[7]=0
Alice filling WAs[8]=0
Alice filling WAs[9]=0
Alice filling WAs[10]=0
Alice filling WAs[11]=0
Alice filling WAs[12]=0
Alice filling WAs[13]=1
Alice filling WAs[14]=0
Alice filling WAs[15]=1
Alice filling WAs[16]=1
Alice filling WAs[17]=1
Alice filling WAs[18]=1
Alice filling WAs[19]=1
Alice filling WAs[20]=0
Alice filling WAs[21]=0
```

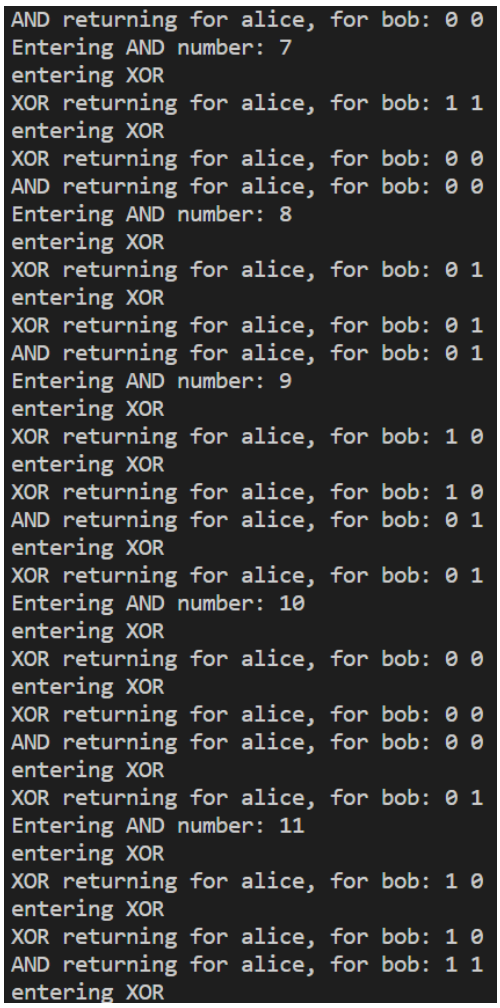
(a)

```
now Alice have:
UAs: [0 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 0]
VAs: [1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1]
WAs: [1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0]

 and Bob have:
UBs: [1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0]
VBs: [1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0]
WBs: [1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0]

 online Phase:

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

Entering the circuit:

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

(b)


entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 0 1
Entering AND number: 2
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 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: 0 1
Entering AND number: 4
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 0 1
Entering AND number: 5
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: 6
entering XOR
XOR returning for alice, for bob: 1 1
entering XOR
XOR returning for alice, for bob: 1 1

(a)


AND returning for alice, for bob: 0 0
Entering AND number: 7
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: 0 0
Entering AND number: 8
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 0 1
AND returning for alice, for bob: 0 1
Entering AND number: 9
entering XOR
XOR returning for alice, for bob: 1 0
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: 0 1
Entering AND number: 10
entering XOR
XOR returning for alice, for bob: 0 0
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 1
Entering AND number: 11
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
XOR returning for alice, for bob: 1 0
AND returning for alice, for bob: 1 1
entering XOR

(b)

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

Done with the first adder:

entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 13
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 1
entering XOR
XOR returning for alice, for bob: 1 1
Entering AND number: 14
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: 0 0
entering XOR
XOR returning for alice, for bob: 0 1
entering XOR
XOR returning for alice, for bob: 0 0
Entering AND number: 15
entering XOR
XOR returning for alice, for bob: 1 0
entering XOR
```

(a)

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

(b)

```
 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:0     Sac:0   Acoutc:1     Sad:0   Acoutd:1
Bob takes   Sba:0  Bcouta:1     Sbb:1   Bcoutb:1     Sbc:1   Bcoutc:0     Sbd:1   Bcoutd:1
and now for the final check
Entering AND number: 19
entering XOR

 opening the result =>

Bob sending RB to Alice


The result is 1
```

6 Conclusions

BeDOZa is Perfectly secure against passive unbounded adversary and with the help of the ElGamal and the OT protocols we got rid of the dealer 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).