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

Homework 7: Report

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

Abstract

In this homework we implemented Yao's protocol for the Boolean circuit of a specific function.
We wrote code in Python that uses this protocol 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/14mqxprFbJD92OPiR5jR1sFScD_U66UaA?usp=sharing`.

1 Introduction

Motivation. In recent years, there has been increasing interest in privacy-preserving computation and secure two-party computation [LP04]. These techniques aim to enable parties to perform computations on their private data without revealing sensitive information to each other or any potential adversaries [Wik].
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. Yao's protocol enables two parties to securely exchange information using a garbled circuit, ensuring confidentiality and privacy.

Secure Computation Technique. Yao's protocol is a significant technique in secure multi-party computation (MPC) created by Andrew Yao in 1982. It enables two parties, Alice and Bob, to jointly compute a function's result without revealing their private inputs to each other[LP04]. The protocol operates on a Boolean circuit, ensuring that parties securely evaluate the circuit and obtain correct outputs without compromising privacy.
The protocol progresses as follows:

1. **Input and Circuit Representation:** Alice and Bob each hold private inputs, represented as binary strings. The function they want to compute is expressed as a Boolean circuit. The circuit computes the desired function based on Alice and Bob's inputs.

2. **Garbled Circuit Generation (Pre-processing):** Alice constructs the garbled circuit based on the Boolean circuit representation. She generates cryptographic labels (encryption of the gate's output) for all possible input combinations. The encrypted

labels hide the gate's functionality and inputs. Alice then sends the garbled circuit to Bob.

3. **Input Oblivious Transfer (OT):** Oblivious Transfer allows Bob to securely obtain labels corresponding to his input wires in the garbled circuit. Bob's input choices (0 or 1) are sent to Alice via OT, ensuring Alice remains oblivious to Bob's actual input. Alice performs the OT protocol and sends labels back to Bob.

4. **Garbled Circuit Evaluation:** With the garbled circuit and labels corresponding to his input wires, Bob locally evaluates the garbled circuit without revealing his inputs. He uses the labels to determine encrypted outputs of each gate and performs decryption operations.

5. **Result Retrieval:** Bob obtains the encrypted output of the entire circuit and sends it back to Alice.

6. **Decryption and Output Extraction:** Alice, holding the private key, decrypts the output received from Bob. She obtains the final result of the computation, the output of the jointly computed function.

Through this process, Alice and Bob maintain privacy and confidentiality.

Application. In our application we defined 2 classes: Alice and Bob. We first randomly generate $\vec{a}, \vec{x}$ as inputs to the protocol and then communicate between the classes using a few lines of code:

1. Alice generates the boolean circuit.

2. Alice assigns labels of $128 - bits$ to each wire and gate in the boolean circuit.

3. Alice builds the garbled circuit.

4. Alice sends to Bob the garbled circuit and the labels that fit Alice's inputs.

5. Alice and Bob compute the labels of the Bob's input wires using OT protocol.

6. Bob compute the output label using Alice's labels and sends it to Alice.

7. Finally, Alice converts the output label to the real output z 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. The empirical evaluation of Yao's protocol involved conducting a series of experiments with different inputs $\vec{a} = (a_1, a_2)$ and $\vec{x} = (x_1, x_2)$. The results obtained were compared to the expected output of the function $f_{\vec{a},4}(x_1, x_2)$ for the given inputs.
We verified the correctness and privacy preservation of the protocol based on the theoretical foundations of secure two-party computation and the adversary model [Wik, GMW91].

2 Preliminaries

In this paper we used the boolean circuit we built in homework 1 (in figures 3, 4, 5 and 6).

2.1 Garbled Circuit

A Garbled Circuit is a cryptographic technique used in secure multi-party computation (MPC) to protect the privacy of inputs during computation. In this approach, a circuit representing the desired function is "garbled" by encrypting its intermediate results, using cryptographic labels. These labels conceal the actual data and gate functionality. Two parties, Alice and Bob, can then securely evaluate the garbled circuit together without revealing their private inputs. Alice provides the garbled circuit to Bob, who uses oblivious transfer to obtain the labels corresponding to his input wires. Bob locally evaluates the garbled circuit, obtaining the encrypted output, which is sent back to Alice for decryption. The final result remains concealed from each party, ensuring privacy and confidentiality. Garbled Circuit plays a crucial role in enabling secure computation scenarios where data privacy is a top priority.

2.2 Oblivious Transfer

Oblivious Transfer is a protocol with 2 participants (Receiver and Sender). The Receiver want one of the messages the Sender holds. The Receiver sends a choice (an index of which of the messages he wants) and the Sender sends him back the message according to his choice.

2.2.1 1-out-of-2 OT Functionality

The Receiver has an index $i \in \{0, 1\}$ and the Sender has two messages m_0, m_1. In the end of the protocol, the Receiver gets m_i and the Sender gets $None$.

2.3 ElGamal Cryptosystem

ElGamal is an implementation of 1-out-of-2 Oblivious Transfer when the protocol is passive. ElGamel cryptosystem consists of 4 algorithms $(Gen, Enc, Dec, Ogen)$ that together make a secure protocol for OT.

Gen: Generate secret and public keys.

Enc: Use the secret key to encrypt the message.

Dec: Use the public key to decrypt the encrypted message.

OGen: Generate a dummy public key.

3 Protocols

3.1 Secure Computation Technique: Yao's Protocol

Parties: Alice A and Bob B.
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.

3.1.1 The Protocol

Parties: Alice A with input x and circuit c, Bob B with input y.

Algorithm 1 Yao's Protocol

```
1: Alice assigns labels of $128 - bits$ to each wire and gate in the boolean circuit.
   She assigns two labels for each wire $x$:
   - The first label for $x = 0$
   - The second label for $x = 1$.
2: for each layer in the real circuit do            ▷ Alice builds the garbled circuit
3:     for each gate in the layer do
4:         Alice generates a garbled table for this gate that encode the gate's truth table.
           For each combination of two input labels $A, B$ and output label $C$, the encrypted output
           label is $\mathbb{E}_{A,B}(C)$ (more in algorithm 2).
5:     end for
6: end for
7: Alice sends to Bob the garbled circuit and the labels that fit Alice's inputs.
8: Alice and Bob use $1 - out - of - 2\ OT$ to compute the labels of the Bob's input wires:
   for each bit $j$ in Bob's input $y$:
       Bob is the receiver with choice input $i = y_j$.
       Alice is the Sender with messages $m_0^j$ (this is the label for Bob's input $j$ whose value
         is 0) and $m_1^j$ (this is the label for Bob's input $j$ whose value is 1).
   NOTE The sub-protocol is described in algorithm 4.
9: for each layer in the garbled circuit do            ▷ Bob compute the output
10:    for each gate in the layer (with input labels $A, B$ and four output labels $C$) do
11:        Bob decrypts the output using $\mathbb{D}_{A,B}(C)$ (more in algorithm 2).
12:    end for
13: end for
14: Bob sends the output label to Alice.
15: Alice converts the output label to the real output.
```

3.1.2 Sub-Protocols

Algorithm 2 Sub-protocol | Encryption And Decryption

```
1: $\mathbb{E}_{A,B}(C)$:
    1) Use $SHA-256$ to encode the input labels $A, B$. mark the $256-bits$ result as $X$.
    2) Add to the output label $C$ a tail of 128 zeros.
    3) Return $XOR(X, C)$.
2: $\mathbb{D}_{A,B}(C)$:
    1) Use $SHA-256$ to encode the input labels $A, B$. mark the $256-bits$ result as $X$.
    2) Compute $decrypted\_label = XOR(X, C)$.
    3) If $decrypted\_label$ has a tail of 128 zeros: return the first $128-bits$ of $decrypted\_label$.
    4) Else: return $None$.
```

Algorithm 3 Sub-protocol | ElGamal Cryptosystem

```
1: $Gen(1^k)$:
    1) Sample $sk \leftarrow_R \{0, ..., q-1\}$.
    2) Generate $pk = (g, h)$ such that $h = g^{sk}\ mod\ p$.
    3) Output $(sk, pk)$.
2: $Enc_{pk}(m)$:
    1) Sample $r \leftarrow_R \{0, ..., q-1\}$.
    2) Generate $C = (c_1, c_2)$ such that $c_1 = g^r\ mod\ p$ and $c_2 = m \cdot h^r\ mod\ p$.
    3) Output $C$.
3: $Dec_{sk}(C)$:
    1) Generate $m = c_2 \cdot c_1^{-sk}$ i.e. $m = m' \cdot h^r \cdot (g^r)^{-sk}\ mod\ p = m' \cdot (g^{sk})^r \cdot (g^r)^{-sk} = m'$.
    2) Output $(sk, pk)$.
4: $OGen(r)$:
    1) Sample $s \leftarrow_R \{0, ..., p-1\}$ and generate $h = s^2\ mod\ p$.
    2) Output $pk = (g, h)$.
```

Algorithm 4 Sub-protocol | Passive 1-Out-Of-2 Oblivious Transfer from ElGamal

```
1: $OT_2(p, q, g)$:                ▷ use ElGamal subprotocol found in algorithm 3
    Receiver:
    Receiver has a choice bit- 0 or 1.
    1) $pk_0, sk = Gen()$.
    2) $pk_1 = OGen(random())$.
    3) $receiver\_choice = choice$.
    Sender:
    Sender has 2 messages- $m_0$ and $m_1$ and $(pk0, pk1)$.
    1) $c_0 = Enc_{pk_0}(m_0)$ and $c_1 = Enc_{pk_1}(m_1)$.
    2) Send $(c_0, c_1)$ to receiver.
    Receiver:
    Receiver has $receiver\_choice$, $(c_0, c_1)$ and $sk$.
    1) If $receiver\_choice == 1$ do $m' = c_1$, else $m' = c_0$.
    2) $dec'_m = Dec_{sk}(m')$
    3) Output $dec'_m$.
```

3.2 Application: Yao's Protocol

Algorithm 5 Yao's Protocol

```
1: Alice generates the circuit.
2: Alice assigns labels of $128 - bits$ to each wire and gate in the boolean circuit.
3: Alice builds the garbled circuit and generates labels for all the constants in the circuit.
4: Alice sends to Bob the garbled circuit, the labels that fit Alice's inputs and the labels
   for the constants.
5: Alice (sender) and Bob (receiver) use $1 - out - of - 2$ $OT$ to compute the labels of the
   Bob's input wires.
6: Bob compute the output label using Alice's labels and constant's labels and sends it to
   Alice.
7: Alice converts the output label to the real output $z$.
8: Alice outputs $z$.
```

4 Implementation

The implementation of the algorithm is done using Python programming language. In order to run the code, you will need to have Numpy, random, and hashlib libraries installed.
Our code generates 2 random inputs $\vec{a} = (a_1, a_2)$ and $\vec{x} = (x_1, x_2)$ and returns the output z, which corresponds to the result of $f_{\vec{a},4}(x_1, x_2)$.
The code is here:
`https://colab.research.google.com/drive/14mqxprFbJD92OPiR5jR1sFScD_U66UaA?usp=sharing`.

5 Empirical Evaluation

Each row in the table represents an experiment, and we conducted a total of 256 experiments by considering all possible inputs of a_1, a_2, and x_1, x_2 for which we obtained an 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

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

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

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

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

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

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

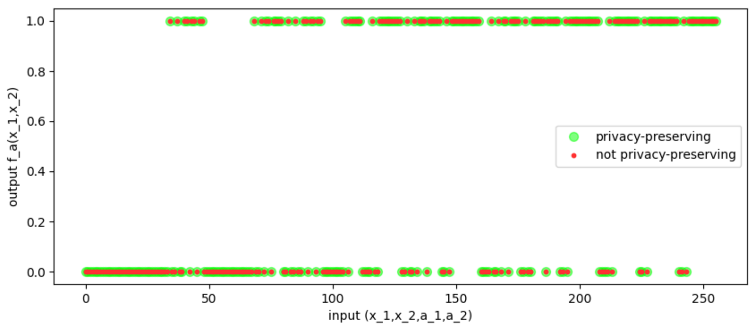

Figure 1: Comparison between privacy-preserving computation and not privacy-preserving computation

Note: You can see our tests file here: `https://colab.research.google.com/drive/19-HjA2zUSsp2gk8oDMTsmTFyQ9yFkTI3?usp=sharing` and the Benchmark tests file here: `https://colab.research.google.com/drive/1N1CgFOL9nwuS8u1Ubnwp2wiMsExpjubF?usp=sharing`.

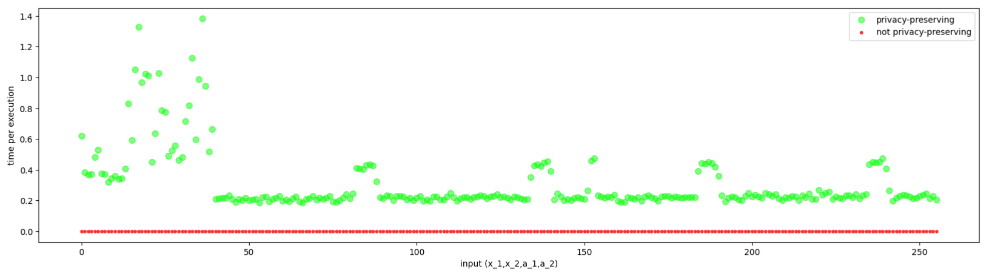

Figure 2: Running time comparison between privacy-preserving computation and not privacy-preserving computation

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 Yao's protocol, we were able to maintain participants' (Alice and Bob) privacy because the private data didn't need to be disclosed for computations.

References

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. On the play-off between computational and statistical zero-knowledge. *Journal of Cryptology*, 4(2):101–139, 1991.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Paper 2004/175, 2004. `https://eprint.iacr.org/2004/175`.

[Wik] Wikipedia. Adversary. Retrieved July 23, 2023.

Appendices

A Schematic Diagrams of The Boolean Circuit From Homework 1

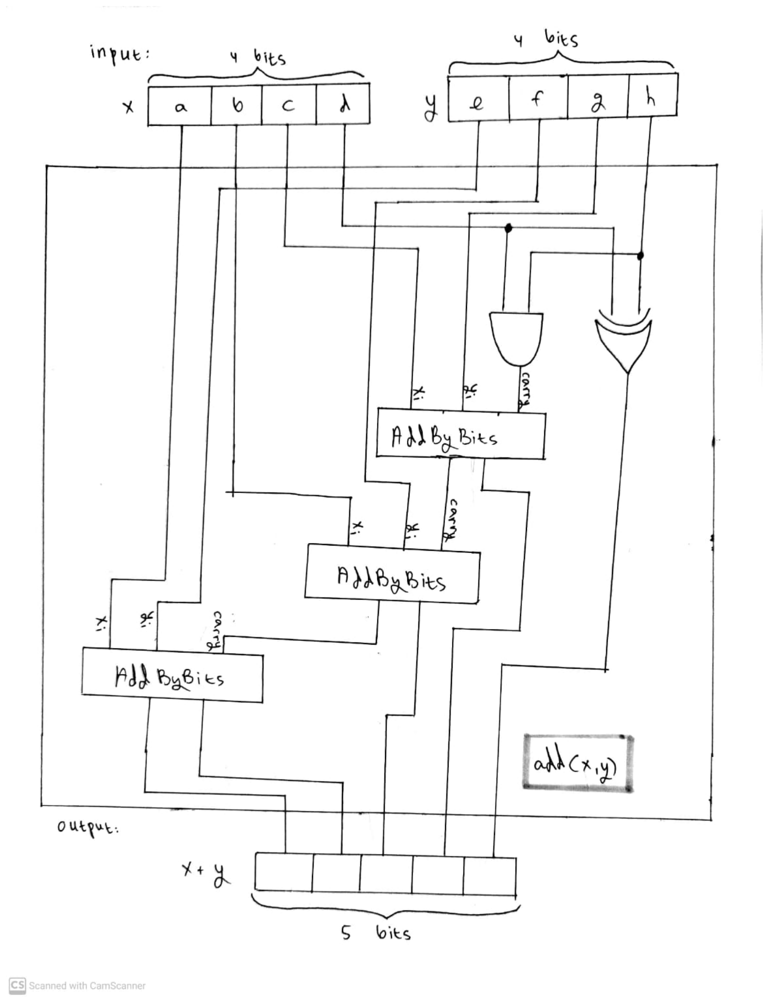

Figure 3: A black box of addition

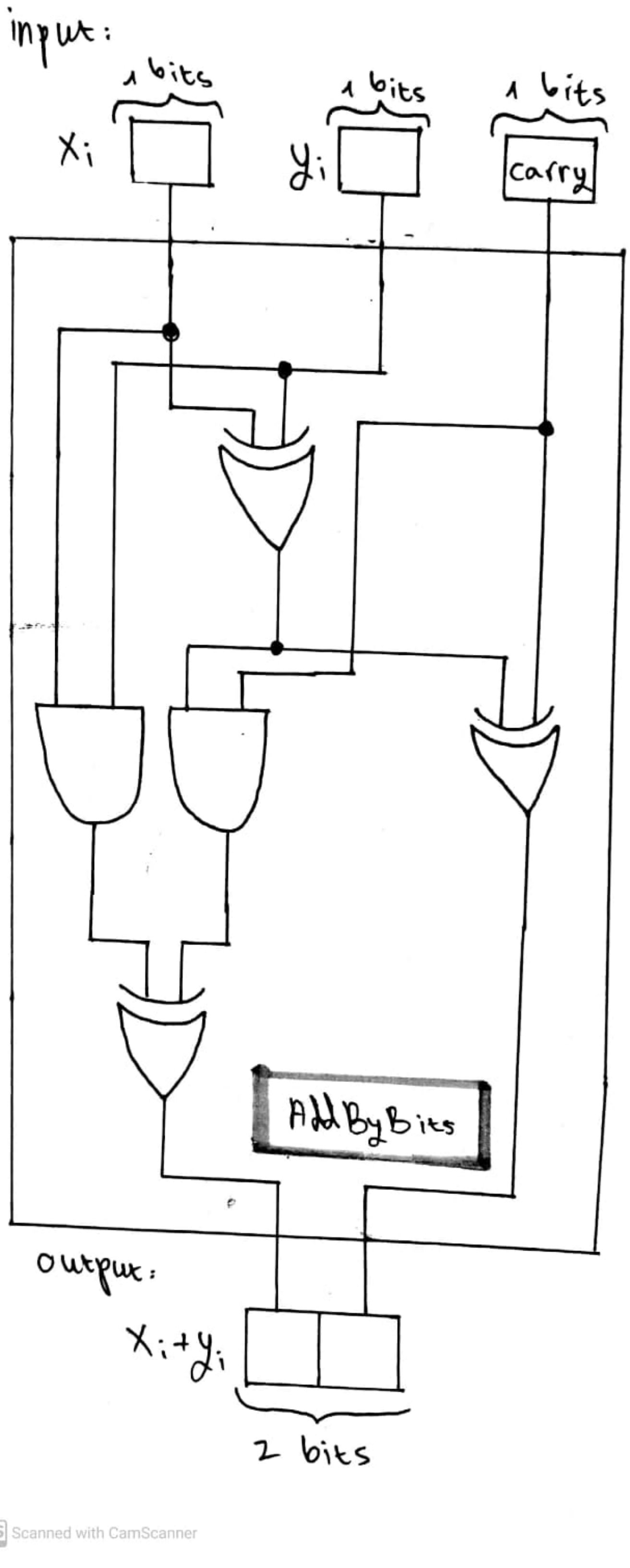

Figure 4: A black box of bits addition

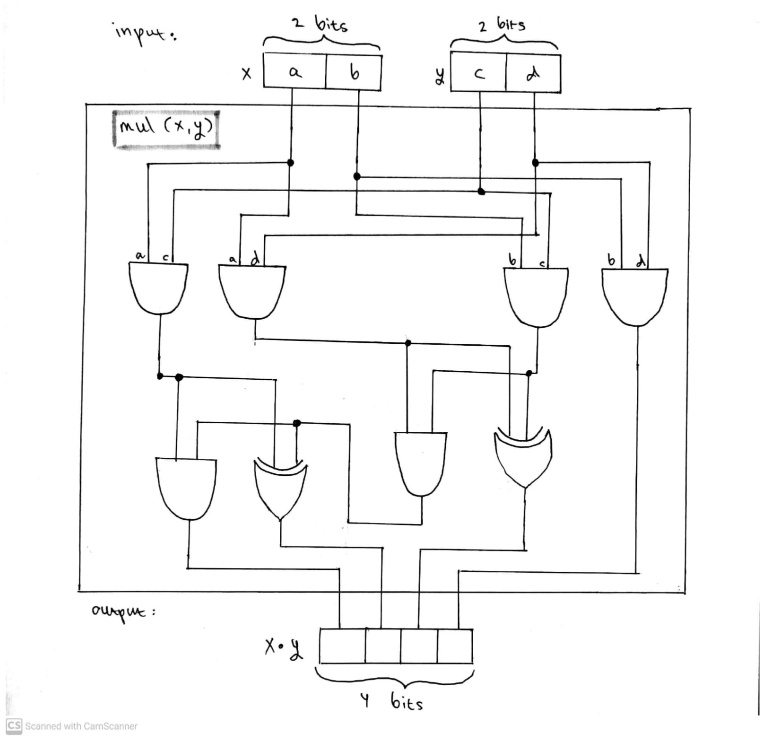

Figure 5: A black box of multiplication

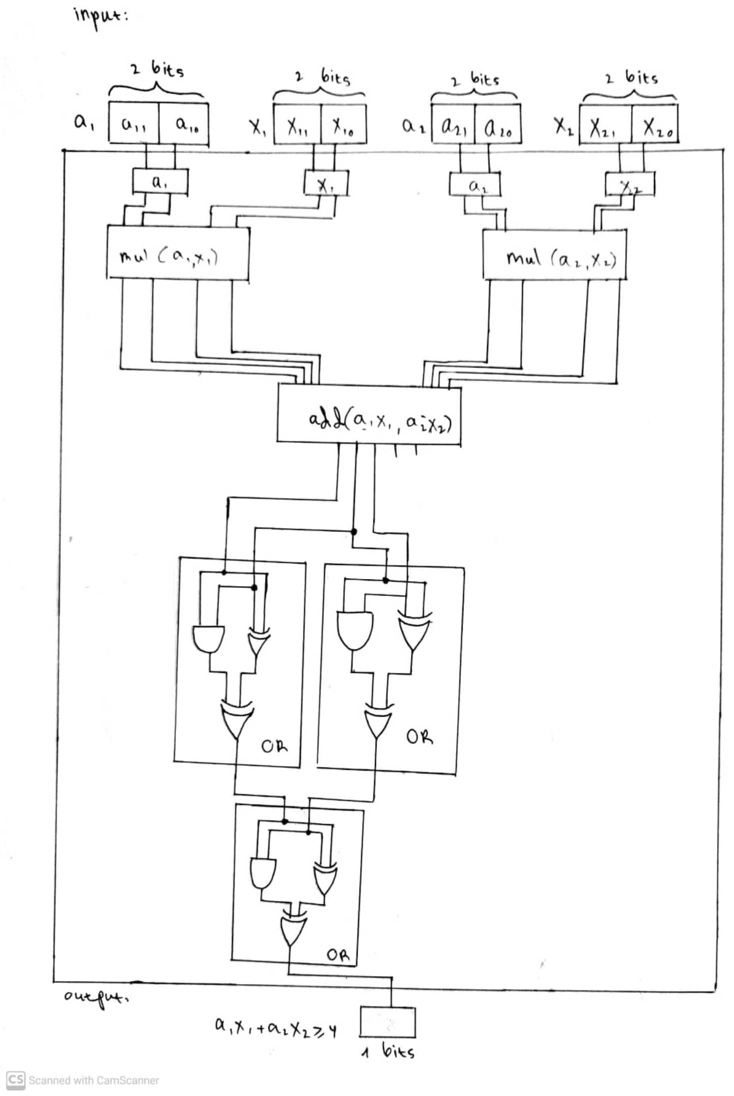

Figure 6: The entire Boolean circuit