

Computing the maximum of two sets of values using Yao's protocol

Federico Dittaro - 12345773

June 21, 2024

Contents

1	Introduction	3
1.1	Yao's protocol	3
1.2	My work	3
2	Implementation	4
2.1	Cryptography	4
2.2	Circuit description	5
2.3	Alice's communication pseudocode	5
2.4	Bob's communication pseudocode	5
2.5	Implementations of required functions	7
2.6	Other functions	8
3	Usage	9
3.1	Requirements	9
3.2	How to run the script	10

1 Introduction

The first subsection (1.1) contains a brief introduction to Yao’s protocol and how it works. In the second subsection (1.2) I will explain the assignment that I have implemented.

The second section (2) discusses the implementations, and the last section (3) covers the required libraries to run the script, and finally, how to do that.

1.1 Yao’s protocol

This project covers an example of *Secure Multi-Party Computation (MPC)* according to Yao’s protocol. This protocol allows two parties (Alice and Bob) with private inputs to compute a joint function of their inputs while ensuring that nothing but the output is learned. For example if Alice knows x and Bob knows y , they should only learn the result of $f(x,y)$.

MPC is based on the following ideas:

- **Privacy:** ensuring that nothing but the output is learned;
- **Correctness:** ensuring that the output is correctly computed.

Properties should be guaranteed even in the face of adversarial behavior:

- **Semi-honest:** adversary running the correct software cannot learn anything;
- **Malicious:** adversary running any software cannot learn anything (even if they know all the protocols, design and so on).

This goal is achieved by using a boolean circuit in which each gate is encrypted, and an Oblivious Transfer (OT) that is responsible for letting Bob know his encrypted input.

1.2 My work

Out of all the possible implementations, I decided to implement the maximum of two sets of values. In my case, the script can handle integers of at most 6 bits. In this implementation *privacy* is respected since neither Alice nor Bob can understand the set of the other. The same holds for *correctness*: assuming that both actors always behave honestly, the result obtained through Yao’s protocol is guaranteed to be the same as that obtained through a standard computation (this is verified by a program procedure).

Following the project ground rules, the implementation is based on the Oliver Roques library [1], uses Fernet as encryption scheme (which includes AES) and takes the input from two different files named “Alice.txt” and “Bob.txt”.

2 Implementation

In this section I will explain the communication protocol and the data flow between Alice and Bob.

2.1 Cryptography

The implementation of Yao's algorithm is based on Fernet's algorithm. Fernet is a part of the cryptography library in Python, which provides cryptographic recipes and primitives to Python developers. It is designed to be easy to use, provides secure encryption and decryption and guarantees that a message encrypted using it cannot be manipulated or read without the key.

It uses AES in CBC mode with a 128-bit key for encryption and HMAC with SHA256 for integrity checks. Encrypted data is base64 encoded, making it URL-safe and easy to store or transmit over HTTP. Each message has also a timestamp, allowing for optional expiration checks. Here's a breakdown of the encryption and decryption process:

Encryption

1. **Key generation:** generate a random key which is a 32-byte base64 encoded string;
2. **AES encryption:** the message is encrypted using AES in CBC mode with a randomly generated initialization vector (IV);
3. **HMAC:** an HMAC is calculated over the IV and the ciphertext using SHA256;
4. **Token Assembly:** the final token includes the version, the timestamp, the IV, the ciphertext, and the HMAC.

Decryption

1. **Token Parsing:** the token is parsed to extract the version, timestamp, IV, ciphertext, and HMAC;
2. **HMAC Verification:** the HMAC is recalculated and verified to ensure the integrity of the message;
3. **AES Decryption:** the ciphertext is decrypted using AES in CBC mode with the extracted IV;
4. **Message Retrieval:** the plaintext message is retrieved and returned.

2.2 Circuit description

The function implemented in the circuit calculates the maximum of two 6-bit numbers, as shown in figure 1. The Alice's input bits are labelled as (A5 A4 A3 A2 A1 A0) and Bob's input bits as (B5 B4 B3 B2 B1 B0). In both cases A5 and B5 are the most significant bits while A0 and B0 are the least significant bits. The idea is to compare bits from most significant to least significant. When a difference between two corresponding bits of the two numbers is detected, the bit set to one determines which number is greater. The output is the greater number.

2.3 Alice's communication pseudocode

Alice receive a circuit and an Oblivious Transfer element (OT). The output of the communication is the maximum value belonging to the set of Alice and Bob. The Alice's pseudocode can be seen in the algorithm box 1

Data: circuit, OT
Result: Max value between Alice and Bob
input \leftarrow takes numbers from the input file;
generate garbled circuit;
send garbled circuit over the socket;
max \leftarrow maximum of Alice set;
call `print_alice_to_bob` to print the intermediate result;
result \leftarrow send Alice's encrypted input and keys using `ot.get_result()`;
call `write_to_output_file` to print Alice maximum;
call `verify_output` to print the result and check if it is correct;

Algorithm 1: Alice's communication pseudocode

2.4 Bob's communication pseudocode

Bob evaluates the given circuit and send the result using the Oblivious Transfer. The behaviour of Bob is described in the algorithm box 2

Data: OT
Result: Max value between Alice and Bob
input \leftarrow takes numbers from the input file;
circuit \leftarrow retrieve circuit from socket;
max \leftarrow maximum of Bob set;
result \leftarrow compute the circuit output;
call `write_to_output_file` to print Bob maximum;
send the result to Alice using `ot.send_result()`;

Algorithm 2: Bob's communication pseudocode

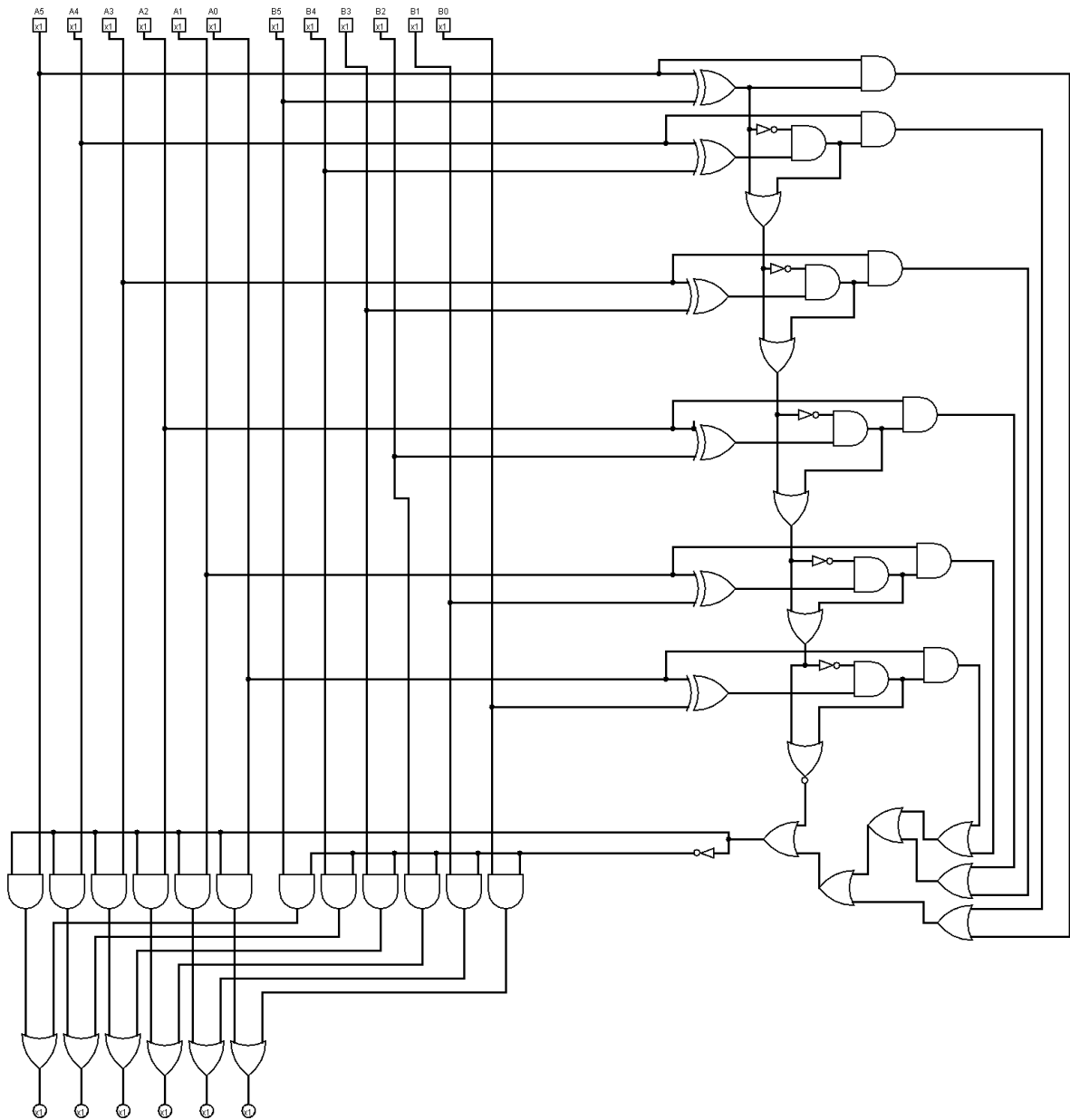


Figure 1: Implemented circuit for computing the max of two 6-bit numbers

2.5 Implementations of required functions

The implementation of the four required functions has been done as follows:

1. **print_alice_to_bob**: this function prints the necessary output from Alice that she wants to send to Bob. The parameters of this function are *a_inputs* that represents Alice's inputs, *a_wires* that represents Alice's wires, and *b_keys* that represents Bob's keys. These values are passed from the class Alice in the main.py file. The result of the computation is a file called **alice_to_bob.txt** in the output directory. It contains Alice's encrypted input described as a couple (key, encrypted bit) and Bob's keys represented as (key, encrypted bit). An example content of the output file is shown in Figure 2.

```
Alice encrypted inputs are:
Wire 1: Key: b'5Y0aL7Y9gFrna6062A5J4EKfpaUdUGNgXLurnwCr02Q=', Encrypted Bit: 0
Wire 2: Key: b'NX0mg4850b_Aw4TgnIH9JLtANaiseZ93E20NgxYEbrI=', Encrypted Bit: 0
Wire 3: Key: b'vsrHp-1Uy2pynVqy1SfXFM060Dkz39B-DkhK7MwmNks=', Encrypted Bit: 1
Wire 4: Key: b'04oqoLfXFt0bX77jpBKZmsvI9kxaqjyGCJ4AALBUdEE=', Encrypted Bit: 1
Wire 5: Key: b'bmKq3ML15XBx9950PlWuxlg0VuCqoBQXSH4QccN8iKdY=', Encrypted Bit: 1
Wire 6: Key: b'NrWKrywotNnGZnCCG1N-td2egcUBpexGqtq3esV6xxU=', Encrypted Bit: 1

Alice keys to Bob are:
Wire 7: (Key: b'Thh9XGpbnaAqPqp0U5rUYcderVQ3oHsiEhCH0MvPa8Q=', Encrypted bit: 0),
      (Key: b'AF6kcl0t6ae9T6xXPkd_zQLo7PKLEC_khcsIRE4KFPw=', Encrypted bit: 1)

Wire 8: (Key: b'b7E18hhymWeT4z1ucwB2gKixZs1NQBmkdb5ndtvqXcc=', Encrypted bit: 0),
      (Key: b'k8_Xh1A41a53u6J0HrcXuGb1WOGSw6HiTXL8G-quP9Q=', Encrypted bit: 1)

Wire 9: (Key: b'F81M4hKpgGwow6vUA4Z9BjxLB5P4W-6ToMBCuPIxwKU=', Encrypted bit: 1),
      (Key: b'a5fJG8jJ1NYq0KxXJ1IByaSpaoMZEIPcV8yeLVBWCY=', Encrypted bit: 0)

Wire 10: (Key: b'rHe3wJYy8sMQ66L0Ssap05kJUuaidUq5UF1IwdnCdaE=', Encrypted bit: 1),
      (Key: b'cMJtyO4c7iqj2Mbbb76Nbv8U5rp_B1-7yHiVm1yyfTQ=', Encrypted bit: 0)

Wire 11: (Key: b'k8ua_wm9mv5LiG7oXDEzg7efgzVCdMyO3CEw61tj0w8=', Encrypted bit: 0),
      (Key: b'qo1ex9rdg19f_Np6QjFcXajyLHWzNtbqISxd8KETj08=', Encrypted bit: 1)

Wire 12: (Key: b'LDw0sztUQzEPj-H28jwEY8XfcpW3LIP9p1YwrRd9Wuk=', Encrypted bit: 1),
      (Key: b'dvyaSf_ppf3nx0cFjaPx5_wuU8W6jbyTfbfJWazFGEQ=', Encrypted bit: 0)
```

Figure 2: alice_to_bob.txt

2. **alice_bob_OT**: this function prints in the console the OT between Alice and Bob that takes place in Yao's protocol. In order to print the important steps of the communication between the two parties, my program will print the following informations (this information are taken from the files *main.py*, *ot.py* and *yao.py*):

```

Bob starts listening to the socket waiting for a connection;
Alice connects and sends the circuit to Bob;
Bob ← retrieves circuit from socket;
Alice sends the encrypted input to Bob;
Bob ← retrieves the encrypted input from socket;
for each gate in the circuit do
    if gate_id in pbits_out then
        | Bob prints the result bit for the corresponding gate;
    else
        | Bob just prints the gate that he's evaluating;
    end
end
Bob sends circuit evaluation;
Alice ← retrieves the circuit evaluation from socket;

```

Algorithm 3: Bob and Alice communication pseudocode

3. **bob_mpc_compute and alice_mpc_compute:** this function prints as output the function that Bob wants to compute on the combined data. Similar to the previous function, this one is not directly implemented in the *format.py* file. Since the objective is to print the result of one of the three functions described in the project slide, the output could be considered the *result.txt* file in the output directory. This file is created after that Alice retrieves the result from Bob.
4. **verify_output:** this function verifies whether the output from *bob_mpc_compute* matches the output from a function computed in a non-multi-party way. This function it's called by Alice in the *main.py* file with the result of the computation as parameter. To confirm the correctness of the result, it is compared with the maximum value from the datasets of Alice and Bob, and everything is then written to the file **result.txt** in the output directory.

```

Bob's input max is 45
Alice's input max is 50
The max is correct and it is 50.

```

Figure 3: *result.txt*

2.6 Other functions

It is also possible to print the circuit by typing on the terminal (inside the *src* directory) *make table*. This function implemented in the class *LocalTest* (inside the *main.py* file) takes as parameters the JSON file containing the circuit and a print mode (by default

set to 'table'). The result is printed in the console and represents the circuit evaluation (look at image 4).

```
python main.py table
===== 6-bit find max =====
p-BITS: {1: 1, 2: 0, 3: 1, 4: 1, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 1, 15: 0, 16: 0, 17: 1, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 0, 24: 1, 25: 0, 26: 0, 27: 1, 28: 1, 29: 1, 30: 0, 31: 1, 32: 1, 33: 1, 34: 0, 35: 0, 36: 1, 37: 0, 38: 1, 39: 1, 40: 0, 41: 0, 42: 1, 43: 1, 44: 1, 45: 0, 46: 1, 47: 1, 48: 0, 49: 1, 50: 1, 51: 1, 52: 0, 53: 0, 54: 1, 55: 1, 56: 1, 57: 0, 58: 0, 59: 0, 60: 0, 61: 0, 62: 0, 63: 1, 64: 1}
1}
GATE: 13, TYPE: XOR
[0, 0]: [1, 1][7, 0]([13, 1], 1)
[0, 1]: [1, 1][7, 1]([13, 0], 0)
[1, 0]: [1, 0][7, 0]([13, 0], 0)
[1, 1]: [1, 0][7, 1]([13, 1], 1)
GATE: 14, TYPE: AND
[0, 0]: [1, 1][13, 0]([14, 0], 1)
[0, 1]: [1, 1][13, 1]([14, 1], 0)
[1, 0]: [1, 0][13, 0]([14, 0], 1)
[1, 1]: [1, 0][13, 1]([14, 0], 1)
GATE: 15, TYPE: NOT
[0]: [13, 0]([15, 1], 1)
[1]: [13, 1]([15, 0], 0)
GATE: 16, TYPE: XOR
[0, 0]: [2, 0][8, 1]([16, 1], 1)
[0, 1]: [2, 0][8, 0]([16, 0], 0)
[1, 0]: [2, 1][8, 1]([16, 0], 0)
[1, 1]: [2, 1][8, 0]([16, 1], 1)
GATE: 17, TYPE: AND
[0, 0]: [15, 0][16, 0]([17, 0], 1)
[0, 1]: [15, 0][16, 1]([17, 0], 1)
[1, 0]: [15, 1][16, 0]([17, 0], 1)
[1, 1]: [15, 1][16, 1]([17, 1], 0)
GATE: 18, TYPE: AND
[0, 0]: [2, 0][17, 1]([18, 0], 0)
[0, 1]: [2, 0][17, 0]([18, 0], 0)
[1, 0]: [2, 1][17, 1]([18, 1], 1)
[1, 1]: [2, 1][17, 0]([18, 0], 0)
```

Figure 4: make table output

3 Usage

This section contains the libraries needed to run the code on your machine, followed by the steps to launch the script.

3.1 Requirements

The code is working with at least a 3.8.10 Python version and needs also the following dependencies:

- **ZeroMQ** for communications;
- **Fernet** for encryption of garbled tables;
- **SymPy** for prime number manipulation.

To install all the dependencies is sufficient to run this command from a terminal (taken from [1]):

```
pip3 install --user pyzmq cryptography sympy
```

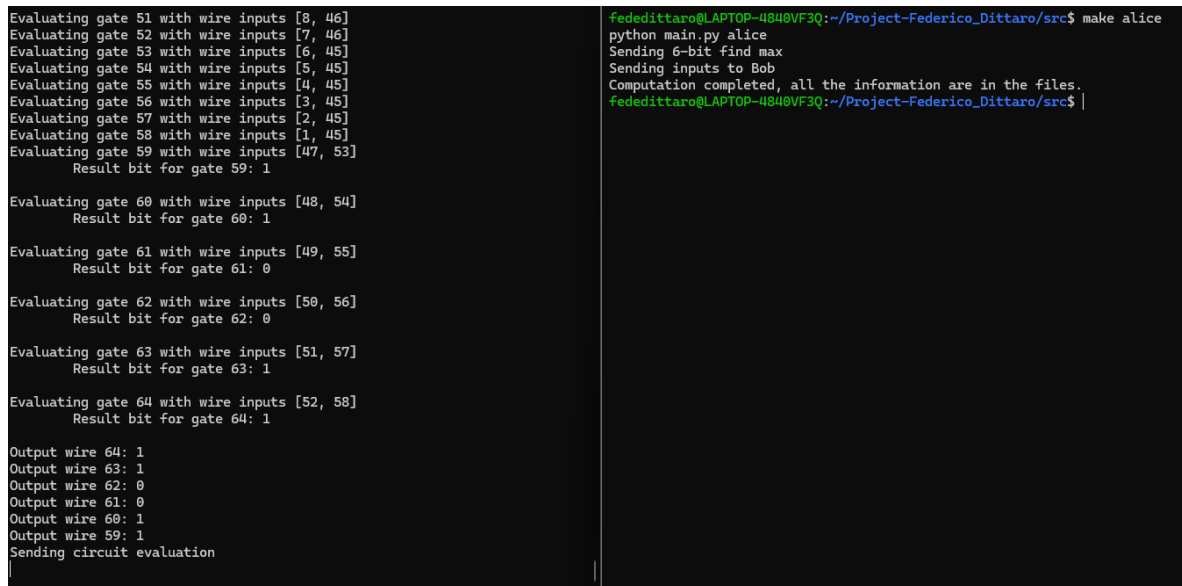
3.2 How to run the script

To run the script it is necessary to open two terminals and navigate with both of them inside the directory containing the `main.py` file (the file is contained in the directory `src`). At this point you need to create first the server (Bob) by typing `make bob` and then in the other terminal, run the client (Alice) by typing `make alice`.

If everything was done correctly, the two terminals should print something like the image 5 and all the other informations printed in the output files (inside the output directory). An example of what the output files will contain are the image 2 and 3.

In order to change the input parameters, it is sufficient to modify the corresponding input files (`Alice.txt` and `Bob.txt`).

If you want to print the circuit, it is sufficient to open just one terminal, navigate in the same directory as before and type `make table`. The output should be the same represented in the image 4.



```
Evaluating gate 51 with wire inputs [8, 46]
Evaluating gate 52 with wire inputs [7, 46]
Evaluating gate 53 with wire inputs [6, 45]
Evaluating gate 54 with wire inputs [5, 45]
Evaluating gate 55 with wire inputs [4, 45]
Evaluating gate 56 with wire inputs [3, 45]
Evaluating gate 57 with wire inputs [2, 45]
Evaluating gate 58 with wire inputs [1, 45]
Evaluating gate 59 with wire inputs [47, 53]
  Result bit for gate 59: 1

Evaluating gate 60 with wire inputs [48, 54]
  Result bit for gate 60: 1

Evaluating gate 61 with wire inputs [49, 55]
  Result bit for gate 61: 0

Evaluating gate 62 with wire inputs [50, 56]
  Result bit for gate 62: 0

Evaluating gate 63 with wire inputs [51, 57]
  Result bit for gate 63: 1

Evaluating gate 64 with wire inputs [52, 58]
  Result bit for gate 64: 1

Output wire 64: 1
Output wire 63: 1
Output wire 62: 0
Output wire 61: 0
Output wire 60: 1
Output wire 59: 1
Sending circuit evaluation

fededitaro@LAPTOP-4840VF3Q:~/Project-Federico_Dittaro/src$ make alice
python main.py alice
Sending 6-bit find max
Sending inputs to Bob
Computation completed, all the information are in the files.
fededitaro@LAPTOP-4840VF3Q:~/Project-Federico_Dittaro/src$
```

Figure 5: Output example

References

- [1] Oliver roques - garbled circuit. Available at <https://github.com/ojroques/garbled-circuit>.