Project Documentation

June 22, 2024

1. The version(s) of Python that your submission can be run under

---

Tested using : **Python 3.10.1**
MPC Function to compute : **Maximum** of two sets of values
Repository used and modified : https://github.com/ojroques/garbled-circuit

2. How to call the script, definition of any input parameters.

0.1) Install required dependencies :

pip install -r requirements.txt

or instead

pip3 install --user pyzmq cryptography sympy

If it doesn't work for you, **check** what libraries are used in the project in **the section 7** of this document

0.2) Setup **inputs** values in Alice.txt and Bob.txt

1) Open **2** different **terminal** window in the **project directory**

2) Run Bob side on one terminal

Use : "**python ./main.py bob**"

3) Run Alice side on the other terminal

Use : "**python ./main.py alice -c <circuit>**"

To do this you **must choose one** of the supplied <circuit> to do the max.
See the next page

Use : "python ./main.py alice -c circuits/max_4_num.json"

or use : "python ./main.py alice -c circuits/max_8_num.json"

or use : "python ./main.py alice -c circuits/max_16_num.json"

or use : "**python ./main.py alice -c circuits/max_32_num.json**"

Based on which one you choose, you can compute the maximum of :

max_4_num = 4 integer numbers in total (2 of Alice and 2 of Bob)
max_8_num = 8 integer numbers in total (4 of Alice and 4 of Bob)
max_16_num = 16 integer numbers in total (8 of Alice and 8 of Bob)
max_32_num = **32 integer numbers in total** (16 of Alice and 16 of Bob)

If you would like to use more integers, simply use the **createCircuitExtended.py**.
Look at the section that relates to it, you can find it in the last part of this document.

3. Differences from base repository and methods implemented.

Evaluation of the differences between the repository https://github.com/ojroques/garbled-circuit and my project programs implementation, highlighting the changes made.

UnchangedUpdatedNew

Main.py

Updated

- *main*(…) :

  Updated

  - added
    - explanation on how to launch the code
    - call to alice.print_alice_to_bob()
    - call to alice.write_alice_to_bob_to_json()

- Moved out other classes (that were in the library main.py) in respective files:
  - YaoGarbler(ABC) ---> yaoGarbler.py
  - Alice(YaoGarbler) ---> alice.py
  - Bob ---> bob.py

yaoGarbler.py

Unchanged

- Class moved in this new file

Yao.py

Unchanged

---

alice.py Updated

- *__init__*(…) Unchanged

- *start*(…) Updated
  - deleted the for cycle used to work with multiple circuits (in this project we work only with circuits/max_4_bit.json)

- *print*(…) Updated
  - read Alice inputs
  - for cycle to iterate along these
  - call to alice_mpc_compute(…)

- *alice_mpc_compute*(result, compared_numbers, a_wires, str_bits_a, b_wires, str_bits_b, outputs, str_result) New

  - write the result obtained from the MPC on alice_ot_side.txt

  - Args :

    result : dictionary with assotiations key : value
    It uses out gates ad key and the corrisponding bit computed as value
    a_wires : list of Alice's wires
    b_wires : list of Bob's wires
    outputs : list of output gates
    str_result : result of the computation

- *write_alice_to_bob_to_json*() New

  - write on “output_alice_bob.json” informations printed by print_alice_to_bob()
  - write also informations that Alice keeps for her (this was done to respect Rule 5 of the assignment)

- *print_alice_to_bob*() New

  - print on video the preliminar informations that Alice sends to Bob

    - circuit (in our project circuits/max_32_num.json)
    - garbled tables
    - parity bit of the output gates

- *get_ali_inputs*() New

- Read Alice's inputs from "Alice.txt"
- Convert the integer in binary (using as much bits for the encoding as much are Alice's wires for the circuit)
- In our project, we represent integers with 4-bit number
- Check and discard invalid values
- Add 0 as number to satisfy circuit input number request

bob.py

Updated

- __init__(…) Unchanged

- listen(…) Unchanged

- send_evaluation(...): Updated

  - read Bob input
  - for iterates now on these inputs
  - Bob output evaluation in decimal, output of send_result()
  - util.**verify_output**(…)

- get_bob_inputs(b_wires) New

  - read Bob input from "Bob.txt"
  - use as much bit for the binary encoding as Bob assigned wire b_wires (in our project 4 wires -> 4-bit binary number representation)
  - Check and discard invalid values
  - Add 0 number to satisfy circuit input nuber request
  - return a list with Bob input in binary encoding

util.py

Updated

- class Socket Unchanged

- class EvaluatorSocket(Socket) Unchanged

- class GarblerSocket(Socket) Unchanged

- verify_output(bob_numbers_binary, bob_max_found_decimal, a_wires) New

    - reads Alice and Bob input from “Alice.txt” and “Bob.tx”
    - compute not mpc the max betweenpair of elements of the two lists.
    - compare the max obtained with MPC and WITHOUT MPC
    - return a list of 0 (true) and 1 (false) based on the compare

    Args :
    - bob_numbers_binary : list of bob inputs in binary 4bit encoded form
    - bob_max_found_decimal : max MPC result
    - a_wires : list of Alice's input wires

ot.py Updated

- *__init__*(…) Unchanged

- *get_result*(…) Updated
    - added some print on video
    - call to self.**print_key_input_ot**(pair, gate)

- *send_result*(…) Updated
    - call to self.**bob_mpc_compute**()
    - obtain and prints Bob computation output

- *ot_garbler*(…) Updated
    - added some prints
    - call to self.**print_alice_ot**()
    - **write** on “**alice_ot_side.txt**” the OT from Alice’s prospective

- *ot_evaluator*(…) Updated
    - added some prints
    - call to self.**print_bob_ot**()
    - **write** on “**bob_ot_side.txt**” the OT from Bob’s prospective

- *print_key_input_ot*(pair, gate) New

    - prints on the file “alice_ot_side.txt” the keypair Alice sends to OT
    - pair is the keypair that Alice sends to the OT
    - gate is the gate ID, so the Bob wires

- *print_alice_ot*(data) New

- print on “alice_ot_side.txt” the OT protocol from Alice’s prospective
- data is a dictionary with key-value associations

- *print_bob_ot*(data) New

  - print on “bob_ot_side.txt” the OT protocol from Bob’s prospective
  - data is a dictionary with key-value associations

- *bob_mpc_compute*(result) New

  - prints on video and write on “bob_ot_side.txt” the output got from yao.evaluate(circuit, g_tables, pbits_out, a_inputs, b_inputs_encr)
  - convert Bob output in decimal encoding
  - return it

createCircuitExtended.py New

- createCircuitExtended.py

  - create truth table for 8 bit input (4 bit for side)
  - divide table and obtain logic expression for output columns
  - create a max circuit block
  - add more block to create a more sophisticated circuit
  - output <name_circuit.json> containing the Boolean circuit

4. Description of the circuit of the function.

The requested chosen function to be implemented is the **Maximum of two sets** of values, values must be taken from the set of integers.

To use the MPC to compute 2 numbers, leads to the fact that both parties will know the result. By comparing pair of values, one from Alice's set and one from Bob's one, **both will know the result of each comparison**.
To find the maximum value it is now possible for both parties to see the various result of the comparison of the two sets.
But even if we could get the right Maximum in this way, there is leakage of the result for each pair compared.

So, **compare** just **pair numbers** (one number for both of them) is **not sufficient**.

The **idea** is to create a **single block circuit** and then **concatenate** his result to further blocks.

Firstly, there is the need to compute the **Truth Table** for the case with **8 inputs**
Here **Alice** has the first 4 bit (a1 a2 a3 a4) and **Bob** has the last 4 bit (b1 b2 b3 b4)
In **output** we get (c1 c2 c3 c4) result of the max function Truth Table

```
{'Alice': '0 0 0 0', 'Bob': '0 0 0 0', 'Outputs': '0 0 0 0'}
{'Alice': '0 0 0 0', 'Bob': '0 0 0 1', 'Outputs': '0 0 0 1'}
{'Alice': '0 0 0 0', 'Bob': '0 0 1 0', 'Outputs': '0 0 1 0'}
{'Alice': '0 0 0 0', 'Bob': '0 0 1 1', 'Outputs': '0 0 1 1'}
{'Alice': '0 0 0 0', 'Bob': '0 1 0 0', 'Outputs': '0 1 0 0'}
{'Alice': '0 0 0 0', 'Bob': '0 1 0 1', 'Outputs': '0 1 0 1'}
{'Alice': '0 0 0 0', 'Bob': '0 1 1 0', 'Outputs': '0 1 1 0'}
{'Alice': '0 0 0 0', 'Bob': '0 1 1 1', 'Outputs': '0 1 1 1'}
{'Alice': '0 0 0 0', 'Bob': '1 0 0 0', 'Outputs': '1 0 0 0'}
{'Alice': '0 0 0 0', 'Bob': '1 0 0 1', 'Outputs': '1 0 0 1'}
{'Alice': '0 0 0 0', 'Bob': '1 0 1 0', 'Outputs': '1 0 1 0'}
{'Alice': '0 0 0 0', 'Bob': '1 0 1 1', 'Outputs': '1 0 1 1'}
{'Alice': '0 0 0 0', 'Bob': '1 1 0 0', 'Outputs': '1 1 0 0'}
{'Alice': '0 0 0 0', 'Bob': '1 1 0 1', 'Outputs': '1 1 0 1'}
{'Alice': '0 0 0 0', 'Bob': '1 1 1 0', 'Outputs': '1 1 1 0'}
```

Then we can divide this table to isolate rows that returns 1 for c1 c2 c3 c4
From them we can get the **SOP logic expression** for each oh c1 c2 c3 c4

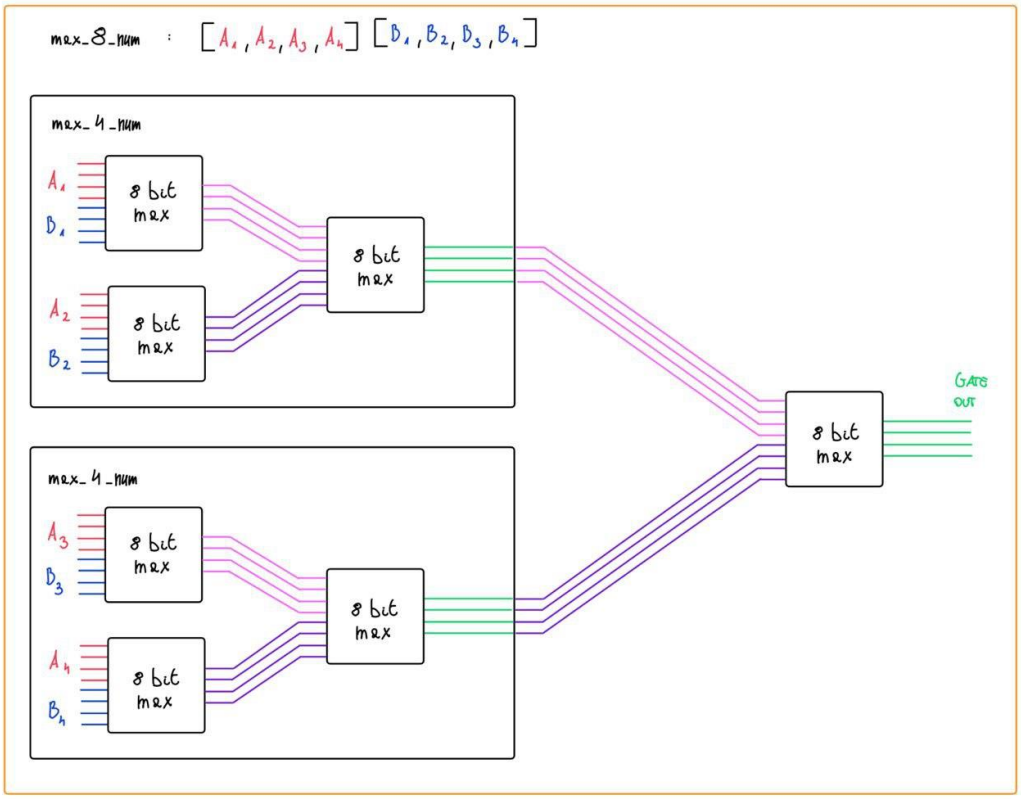
```
Espressione logica per c1: a1 | b1
Espressione logica per c2: (a1 & a2) | (b1 & b2) | (a2 & ~b1) | (b2 & ~a1)
Espressione logica per c3: (a1 & a2 & a3) | (b1 & b2 & b3) | (a1 & a3 & ~b1) | (a1 & a3 & ~b2) | (a2 & a3 & ~b1) | (b1 & b3 & ~a1) | (b1 & b3
& ~a2) | (b2 & b3 & ~a1) | (a3 & ~b1 & ~b2) | (b3 & ~a1 & ~a2)
Espressione logica per c4: (a1 & a4 & ~b1) | (b1 & b4 & ~a1) | (a1 & a2 & a3 & a4) | (b1 & b2 & b3 & b4) | (a1 & a2 & a4 & ~b2) | (a1 & a2 & a
4 & ~b3) | (a1 & a3 & a4 & ~b2) | (a2 & a3 & a4 & ~b1) | (b1 & b2 & b4 & ~a2) | (b1 & b2 & b4 & ~a3) | (b1 & b3 & b4 & ~a2) | (b2 & b3 & b4 &
~a1) | (a1 & a4 & ~b2 & ~b3) | (a2 & a4 & ~b1 & ~b2) | (a2 & a4 & ~b1 & ~b3) | (a3 & a4 & ~b1 & ~b2) | (b1 & b4 & ~a2 & ~a3) | (b2 & b4 & ~a1
& ~a2) | (b2 & b4 & ~a1 & ~a3) | (b3 & b4 & ~a1 & ~a2) | (a4 & ~b1 & ~b2 & ~b3) | (b4 & ~a1 & ~a2 & ~a3)
```

Using this logic expression, it is possible to generate a **Boolean Circuit** for the **single block**.

With this in mind, the code needs to firstly do the NOT gates, then computes all ANDs and lastly all the ORs.

Obtained this circuit, it is possible to create a block that is composed by 3 "**8-bit max**" circuits. This is the max_4_num.json circuit.

Replicating this block, it is possible to create the circuits that you can see inside "circuits" folder.



You can find :

- max_32_num.json : compare 32 numbers, 16 for side
- max_16_num.json : compare 16 numbers, 8 for side
- max_8_num.json : compare 8 numbers, 4 for side
- max_4_num.json : compare 4 numbers, 2 for side

5. Protocol flow between Alice and Bob as pseudocode

Pseudocode communication Alice to Bob

```
1 garbled_circuit ← create(circuit)
2 send(garbled_circuit, informations)
3 ali_inputs ← get()
4 ali_encrypted_input_keys ← encryption(inputs)
5 send(ali_encrypted_input_key)
5 bob_ot_gate ← receive()
6 send(ot_keys)
7 ot_garbler(ot_informations)
8 max_result_evaluation ← receive_mpc()
```

Pseudocode communication Bob to Alice

```
1 garbled_circuit ← listen(alice)
2 start_evaluation(circuit_informations)
3 binary_bob_input ← get_bob_input()
4 alice_encrypted_input ← receive()
5 bob_gate_keys ← send_ot(bob_gate_id, b)
6 max_result ← evaluate_mpc(garbled_circuit, alice_encrypted_input, bob_gate_keys)
7 send(max_result)
8 outcome ← verify(max_result, no_mpc_max_result)
```

6. Other information that is necessary to run your submission

To successfully **run the programs**, the user will first have to open **two** separate **terminals** in the **project directory**.

This way we will use one terminal to perform operations related to Alice, while the other will handle operations related to Bob.

Both executed scripts will take input for calculation from the "Alice.txt" and "Bob.txt" files.

These files contain the decimal representation of numbers that can be represented in binary using **4 bits** (so integers from 0 to 15).
Each element is separated from the next by a space.

You can **change** Alice or Bob's **input** by editing these files.
Es. Alice.txt contains : 3 9 22 25 367 6767676 1

- Decimal numbers > **15** get **discarded**
- If numbers are **less than** circuit **inputs requested**, the **list** get **completed** with **0**s
- If numbers are **more than** circuit **inputs requested**, **next numbers** will be **discarded**

First you are asked to **run Bob** side, so that he listens for new messages from the socket.

It's then asked to **run Alice's part**, providing the **circuit you wish** to use as a **parameter**. In the context of this project, we will use this command :

python ./main.py alice -c circuits/max_32_num.json

At this point it will be created in the project root directory a new file json named **output_alice_bob.json** that contains :

- Preliminar information **sent** by Alice to Bob :

  to_send =
  "circuit": circuit["circuit"] circuit taken from <circuit_name>.json

"garbled_tables": garbled_tables_base64 garbled table created
"pbits_out": circuit["pbits_out"], parity bit of the output gate

- Preliminar informations that Alice **keeps** for her :

rule_5 =
"keys": keys_base64, pair that she created for each wire
"pbits": circuit["pbits"], parity bit of each wire output

This additional information was **written on** the **json** to respect **Rule 5** of the assignment.
So the **circuit** that Alice produces is saved in “output_alice_bob.json” .

Then the computation proceeds, we will see two file “**alice_ot_side.txt**” and “**bob_ot_side.txt**” .

There is written the information about the OT protocol, from both the prospectives.

In **alice_ot_side.txt** we can see :

- The **OT** from the **Alice view** :
  - There is one OT for each Bob assigned to gate
- For each gate ID requested, you can also find the **key pair** that **Alice** sends to the OT.
- When done all necessary OT, Alice gets the **result** of the computation form the socket and print on file this output.

```
Bob request an ot for the gate 63
Alice sends this pair of key to the OT
(b'\x80\x04\x954\x00\x00\x00\x00\x00\x00\x00C,ZwrP7Xwwv1H6B8aZaWw8--pwrepQgvdjR1ojVUXQMj0=\x94K\x01\x86\x94.', b'\x80\x
-------------- Alice starts the OT protocol
Send to Bob the Prime Group
Send to Bob c: 4446413006492941983
Send to Bob
c1: 3997628982557841684
e0: b"n\xd6O\xc3\x95\x97S[-\xb4\xbc\xa2\x87\x1c\xee5o\x97_\x8eL\xd9^\xfc/\xbc\r\xad\x8b\x8e(\x9d\x03\xd1\xd5K\xb1\xb0P
e1: b'\x95+\xae\xd3\xac\x96\xbbv}\xdbI\x00\x9c\n \x04\x07\xf2>\xd8\x03\xca\xdf\xbc\xef\xf9\x13\xdaD\x08\xe7/\x9b\xa1\x
-------------- Alice ends OT protocol

Bob request an ot for the gate 64
Alice sends this pair of key to the OT
(b'\x80\x04\x954\x00\x00\x00\x00\x00\x00\x00C,TUXV0K5Zj6JSYHFl_T7v8Isgz3MC3BKi9CXuNM7Q87I=\x94K\x00\x86\x94.', b'\x80\x
-------------- Alice starts the OT protocol
Send to Bob the Prime Group
Send to Bob c: 3808708995358756787
Send to Bob
c1: 14430835744679511995
e0: b'\x94sQV\x0b\xb1\xe3\x15=\xad\xa7m\x19\x910.\xb3m\x84F:0\xd6Fe\xe0\x03\xfb\t\x1d\x01\xa7\xe5\xbf\xaav\xd2R\x83\xa
e1: b'\xdb\x15\n`\xcc\r4\x9f\x88/XW\xa0Z\xb4\xd5]\xb3,\x8d\xd1\xf0\x06\x9ed\xb2$\xce\x95O\xa7\tv\x82\xf7\x7f\xd5\x8a\x
-------------- Alice ends OT protocol
Ctrl+L to chat, Ctrl+K to generate
Alice[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3

Alice obtained from Bob the result : {1745: 1, 1754: 1, 1785: 1, 1864: 0}

################## Alice and Bob Pair Computation Ended ##################
```

In **bob_ot_side.txt** we can see :

- The **OT** from the **Bob view** :
  - There is **one OT for each Bob** assigned to **gate**
  - Bob sends a bit " **b** " to the **OT**, it is related to his inputs bit

- For each Bob input **gate**, bob request and obtain a relative **key mb**
- The **result** of the **MPC max function**

```
-------------- Bob starts OT
Bob sends to OT his choice b = 0
Bob got from Alice c = 4446413006492941983
Bob got from Alice :
c1 : 3997628982557841684
e0 : b"n\xd6O\xc3\x95\x97S[-\xb4\xbc\xa2\x87\x1c\xee5o\x97_\x8eL\xd9^\xfc/\xbc\r\xad\x8b\x8e(\x9
e1 : b'\x95+\xae\xd3\xac\x96\xbbv}\xdbI\x00\x9c\n \x04\x07\xf2>\xd8\x03\xca\xdf\xbc\xef\xf9\x13\
His key from the OT : mb = b'\x80\x04\x954\x00\x00\x00\x00\x00\x00\x00C,ZwrP7Xwwv1H6B8aZaWw8--pw
-------------- Bob Ends OT

-------------- Bob starts OT
Bob sends to OT his choice b = 0
Bob got from Alice c = 3808708995358756787
Bob got from Alice :
c1 : 14430835744679511995
e0 : b'\x94sQV\x0b\xb1\xe3\x15=\xad\xa7m\x19\x910.\xb3m\x84F:0\xd6Fe\xe0\x03\xfb\t\x1d\x01\xa7\x
e1 : b'\xdb\x15\n`\xcc\r4\x9f\x88/XW\xa0Z\xb4\xd5]\xb3,\x8d\xd1\xf0\x06\x9ed\xb2$\xce\x95O\xa7\t
His key from the OT : mb = b'\x80\x04\x954\x00\x00\x00\x00\x00\x00\x00C,TUXV0K5Zj6JSYHFl_T7v8Isg
-------------- Bob Ends OT

Bob obtained with the MPC his result : {1745: 1, 1754: 1, 1785: 1, 1864: 0}
In decimal : 14

################## Bob and Alice Pair Computation Ended ##################
```

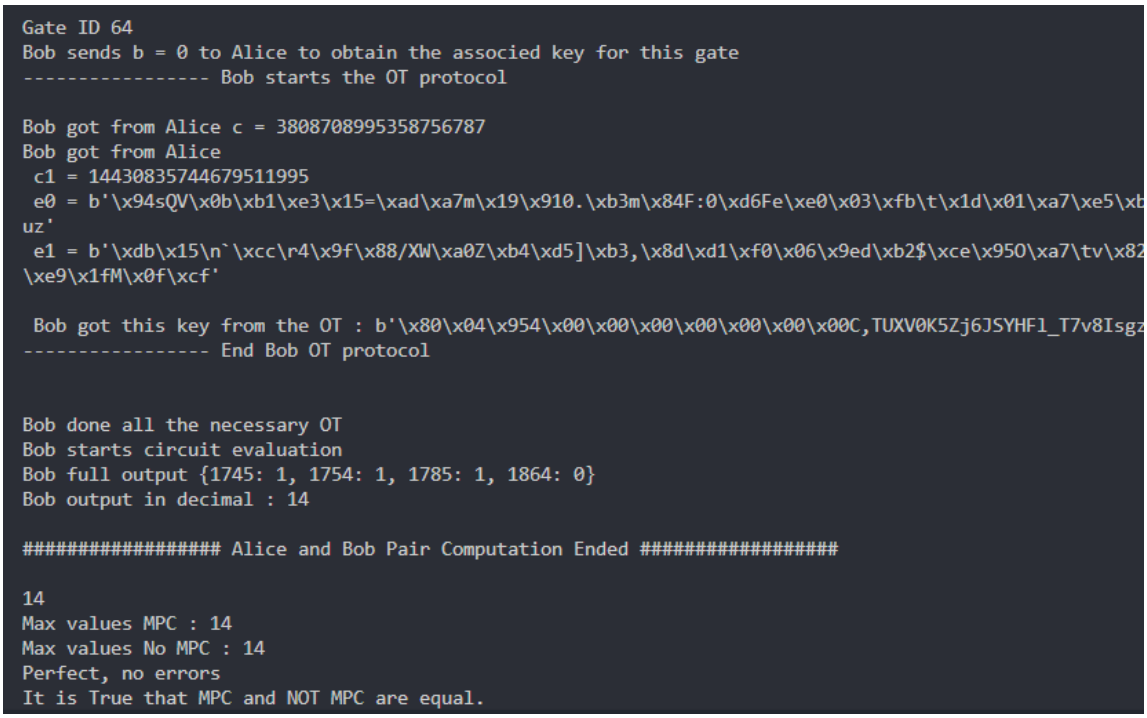
The txt files are generated using

ot.ObliviousTransfer.**print_alice_ot**()
ot.ObliviousTransfer.**print_bob_ot**()

Lastly, according to the **Rule 5** of the assignment, the **verification function**.

It is evaluated based on the fact that MPC and NOT MPC computations give back the same evaluation or not.

It doesn't get print in the file, but get **print** at the end in the **Bob terminal window**



Create a new max circuit using “createCircuitExtended.py”

Keep in mind that :

NUM_INPUT_TO_CONFRONT : set to 4, or 8, or 16, or 32 to obtain a max boolean circuit
NAME_OUTPUT_CIRCUIT : set here the name of the output json file containing the circuit

This program can be called using : **python createCircuitExtended.py**

This is the workflow:

- Generate the **truth table** for the **Max for 8 inputs**
- Generate **SOP logic expression** from the truth table
- Create a **circuit** form SOP logic expression using 8 input bits
- Replicate multiple times to manage more of that kind of unit
- Generate the **circuit** using NUM_INPUT_TO_CONFRONT (from it, one half if for Alice and other for Bob)

Edoardo Diana