Yao’s Protocol

Aakash Rathee

June 20th, 2024

The goal of the project was to implement a multi-party computation (MPC) scheme for two parties using Yao’s protocol to determine the maximum value between two sets of 4-bit integers provided by the parties. My implementation extends the work of Olivier Roques and Emmanuelle Risson[1] on Yao’s protocol in the following ways:

- Encryption using AES in CTR mode
- Logging of messages sent between the two parties
- Logging of evaluation results for each computation of the circuit
- Logging of oblivious transfer
- Verification of results
- Input from a text file
- Evaluation circuit which find the maximum value between two sets of 4-bit unsigned integers

1. Circuit

The circuit has 8 inputs, with 4 inputs allocated to each party, and it produces 4 outputs. It uses 4 full subtractor circuits in series to determine which input has a larger value, and 4 multiplexer circuits in parallel to select the appropriate input for output based on the results of the subtractor circuits. Figure 1 shows the overview of the full circuit, due to the modular nature of the circuit, it can easily be extended for *n*-bits by adding more *n* subtractor and multiplexer circuits. In addition the circuit can be modified to find the minimum input value by switching the position of the A_i and B_i in the multiplexer circuit.

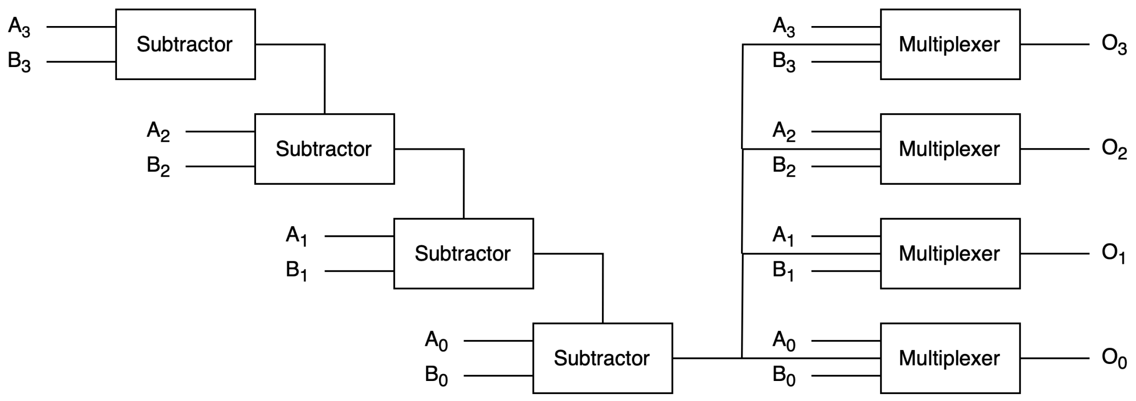

Figure 1. Overview of 4 bit max circuit

1.1. Subtractor Circuit

Figure 2 shows the subtractor circuit, it consists of 3 input A_i, B_i and C_{i-1}, a single output C_i and 5 logic gates. Where A_i and B_i are the input bits provided by two parties, C_{i-1} is the carry bit from the previous subtractor circuit, and C_i is output of the current subtractor circuit and is called

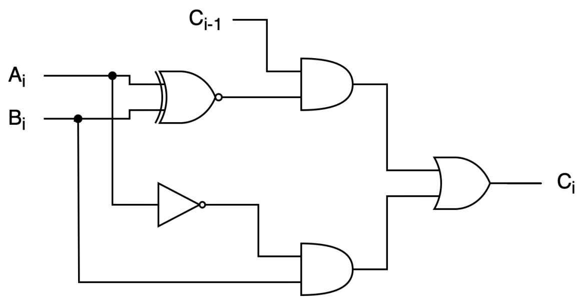

Figure 2. Subtractor circuit

the carry bit[2]. The difference bit was removed from the output of the subtractor circuit as it does not hold any importance when the circuit is used to find the the max of the input numbers.

Equation (1) shows the subtractor circuit represented as boolean equation.

$$C_i = (\neg A_i \wedge B_i) \vee (C_{i-1} \wedge \neg(A_i \oplus B_i)) \tag{1}$$

Table 1 shows the truth table for the subtractor circuit

A_i	B_i	C_{i-1}	C_i
F	F	F	F
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	F
T	T	F	T
T	T	T	T

Table 1. Truth table of subtractor circuit

Since the first subtractor won't have a carry bit, the circuit was modified to incorporate the absents of the carry bit, which can be seen if Figure 3.

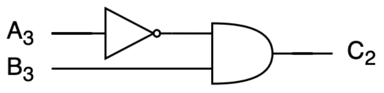

Figure 3. First subtractor circuit

Equation (2) show the first subtractor circuit represented as a boolean equation.

$$C_2 = \neg A_3 \wedge B_3 \tag{2}$$

Table 2 show the truth table for the first subtractor circuit.

A_3	B_3	C_2
F	F	F
F	T	T
T	F	F
T	T	F

Table 2. Truth table of the first subtractor circuit

1.2. Multiplexer Circuit

Figure 4 shows the multiplexer circuit, it consist of 3 inputs A_i, B_i and C and a single output O_i. Where A_i and B_i are the input bits provided by two parties, C is the control bit and O_i is the output bit. This type of multiplexer is called 2x1 multiplexer and can be thought out as an if else statement - if C then A_i otherwise B_i.[3]

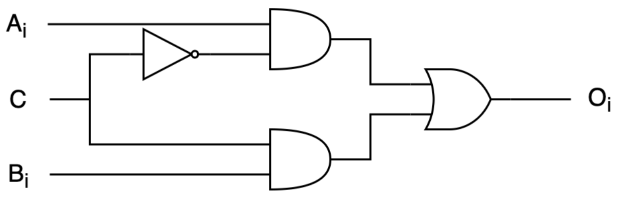

Figure 4. Multiplexer circuit

Equation (3) shows the multiplexer circuit represented as a boolean equation.

$$O_i = (A_i \wedge \neg C) \vee (B_i \wedge C) \tag{3}$$

Table 3 shows the truth table for the multiplex circuit.

A_i	B_i	C	O_i
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	F
T	F	T	T
T	T	F	T
T	T	T	T

Table 3. Truth table of multiplexer circuit

1.3. Circuit.json

The circuit used for the evaluation is represented using a JSON file. Their are rules and limitations to follow which are laid out by the Github Repo.

- Only the following gates allowed to be used in the circuit.
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - NXOR
- Bob knows the boolean representation of the function. Thus the principle of “No security through obscurity” is respected.
- All gates have one or two inputs and only one output.
- The outputs of lower numbered gates will always be wired to higher numbered gates and/or be defined as circuit outputs.
- The gate id is the id of the gate’s output.

Using the rules listed above the 4 bit max circuit was created shown in Figure 5. For Alice the input wires are 1, 2, 3, 4 and for Bob the inputs wires 11, 12, 13, 14. The output wires of the circuit are 101, 102, 103, 104. The circuit can also be found in the log files for Alice and Bob as it is transferred during the execution of the program.

```
{
  "id": "4 BIT MAX",
  "alice": [1, 2, 3, 4],
  "bob": [11, 12, 13, 14],
  "out": [101, 102, 103, 104],
  "gates": [
    {"id": 20, "type": "NOT", "in": [4]},
    {"id": 21, "type": "AND", "in": [20, 14]},
    {"id": 22, "type": "XNOR", "in": [3, 13]},
    {"id": 23, "type": "AND", "in": [21, 22]},
    {"id": 24, "type": "NOT", "in": [3]},
    {"id": 25, "type": "AND", "in": [24, 13]},
    {"id": 26, "type": "OR", "in": [23, 25]},
    {"id": 27, "type": "XNOR", "in": [2, 12]},
    {"id": 28, "type": "AND", "in": [26, 27]},
    {"id": 29, "type": "NOT", "in": [2]},
    {"id": 30, "type": "AND", "in": [29, 12]},
    {"id": 31, "type": "OR", "in": [30, 28]},
    {"id": 32, "type": "XNOR", "in": [1, 11]},
    {"id": 33, "type": "AND", "in": [31, 32]},
    {"id": 34, "type": "NOT", "in": [1]},
    {"id": 35, "type": "AND", "in": [34, 11]},
    {"id": 40, "type": "OR", "in": [35, 33]},
    {"id": 41, "type": "NOT", "in": [40]},
    {"id": 48, "type": "AND", "in": [41, 4]},
    {"id": 49, "type": "AND", "in": [40, 14]},
    {"id": 101, "type": "OR", "in": [48, 49]},
    {"id": 46, "type": "AND", "in": [41, 3]},
    {"id": 47, "type": "AND", "in": [40, 13]},
    {"id": 102, "type": "OR", "in": [46, 47]},
    {"id": 44, "type": "AND", "in": [41, 2]},
    {"id": 45, "type": "AND", "in": [40, 12]},
    {"id": 103, "type": "OR", "in": [44, 45]},
    {"id": 42, "type": "AND", "in": [41, 1]},
    {"id": 43, "type": "AND", "in": [40, 11]},
    {"id": 104, "type": "OR", "in": [42, 43]}
  ]
}
```

Figure 5. Circuit as JSON

2. Algorithm

The whole program can be broken down into 2 parts - initialisation and Yao's protocol. The initialisation part is responsible for reading the inputs from the txt file and the Yao's protocol part is responsible for finding the maximum 4 bit integer value from the two sets of inputs using Yao's protocol and the evaluation circuit mentioned above. The two parties will be referred as Alice and Bob where Alice is garbler and Bob is the evaluation.

2.1. Initialisation

Algorithm 1 and algorithm 2 show the initialisation of Alice and Bob respectively, the initialisation of both the parties is nearly identical. Both the parties take *oblivious_transfer*, *bit_size*, *inputs_file* and *logs_file* as input, the *oblivious_transfer* and *bit_size* must be of the same value for both the parties. Where *oblivious_transfer* is used to decide where oblivious transfer is to be used during the evaluation of the circuit. *bit_size* is the number of input bits per party in the evaluation circuit, this defaults to 4. *inputs_file* and *logs_file* are the path to input integer set txt file and log file respectively.

```
Data:
    string circuits,
    bool oblivious_transfer,
    int bit_size,
    string inputs_file,
    string logs_file

Result: initialisation of Alice

circuits ← parse_json(circuits)
socket ← GarblerSocket(logs_file)
ot ← ObliviousTransfer(socket, oblivious_transfer)
inputs ← parse_input_file(inputs_file, bit_size)
global_max ← -1
```

Algorithm 1. Initialisation of Alice

```
Data:
    bool oblivious_transfer,
    int bit_size,
    string inputs_file,
    string logs_file

Result: initialisation of  Bob

socket ← GarblerSocket(logs_file)
ot ← ObliviousTransfer(socket, oblivious_transfer)
inputs ← parse_input_file(inputs_file, bit_size)
global_max ← -1
```

Algorithm 2. Initialisation of Bob

Algorithm 3 shows how the inputs integers are read from the txt file, which is used in both algorithm 1 and algorithm 2 as *parse_input_file* function. The algorithm has two inputs *bit_size* and *inputs_file* which are the same from algorithm 1 and algorithm 2. The integer values in the input txt file are all in the first file separated by a single space. The function is responsible for checking if each input value is in the acceptable range and converting them into binary.

```
Data:
  int bit_size,
  string inputs_file

Result: array of string representing the input set of integers in binary

max_int ← 2 ** bit_size - 1
file ← open(input_file)
contents ← file.readlines().split(" ")

items ← []

for int_str in contents
 int_val ← int(int_str)

 if int_val < 0 or int_val > max_int then
  raise error("Out of range value")
 else
  items.push(bin(int_val, bit_size))
 end

return items
```

Algorithm 3. Parsing of input txt file

2.2. Yao's Protocol

Algorithm 4 and algorithm 5 show the overview of how the max integer is found using Yao's protocol for Alice and Bob respectively. Alice initiates the conversation with Bob by sending over the information about the circuit. Once Bob receives the information it acknowledges it. Now both the parties can start with the evaluation of the circuit with their inputs. Once the evaluation is complete Alice sends over a message to Bob to end the communication between them.

```
Data:
 circuits
Result: None

for circuit in circuits
 socket.send_wait(circuit)
 evaluate(circuit)
end

socket.send_wait("end")
```

Algorithm 4. Overview of Alice

```
Data: None
Result: None

for message in socket.poll()
 if message.type == "circuit"
  socket.send(true)
  evaluate(message)
 elif message.type == "end"
  socket.send(true)
  break
 end
end
```

Algorithm 5. Overview of Bob

Algorithm 6 and algorithm 7 show how the evaluation of the circuit takes place for Alice and Bob respectively. For both the parties the algorithm is very similar. Firstly, the function extracts information from the message about the circuit. Then both the parties check if they have used all the inputs at least once for evaluation with the circuit, and then this information is exchanged between the parties. After this a input is selected by both the parties and the individual bits are extracted from the string and converted into integer and which are mapped to input wires of the party. Now the evaluation of the circuit can start, for the evaluation of the circuit the function from the GitHub repo is utilised. After the evaluation both the parties have the result, which is the maximum of the two inputs. Using the result both the parties update their global max locally.

The algorithm leaks a lot of information about the inputs of the other party, which defeats the purpose of MPC. Ideal to find the global max the circuit should be evaluated only once with the max values of the individual party. It will ensure no other input value is leaked to the other party. This method was not used due to the restriction in the project description.

```
Data:
  message
  list[str] inputs
Result: None

circuit ← message.circuit
pbits ← message.pbits
keys ← message.keys

globalMax ← 0
hasAliceExhausted ← False
hasBobExhausted ← False
counter ← 0

while ! hasAliceExhausted && ! hasBobExhausted

  if counter == inputs.length then
    counter ← 0
    hasAliceExhausted ← True
  end

  hasBobExhausted ← socket.sendwait(hasAliceExhausted)

  bits ← []
  for val in inputs[counter]
    bits.push(int(val))
  end

  counter ← counter + 1

  results ← ot.getResult(bitsToWire(circuit.AliceWire, bits), bobKeys)

  if result > globalMax then
    globalMax ← result
  end
end
```

Algorithm 6. Evaluation function of Alice

```
Data:
  message
  list[str] inputs
Result: None

circuit ← message.circuit
pbitsOut ← message.pbitsOut
garbledTables ← message.garbledTables

globalMax ← 0
hasAliceExhausted ← False
hasBobExhausted ← False
counter ← 0

while ! hasAliceExhausted && ! hasBobExhausted

  if counter == inputs.length then
    counter ← 0
    hasBobExhausted ← True
  end

  hasAliceExhausted ← socket.recv()
  socket.send(hasBobExhausted)

  bits ← []
  for val in inputs[counter]
    bits.push(int(val))
  end

  counter ← counter + 1

  results ← ot.sendResult(circuit, garbledTables, pbitsOut, bitsToWire(circuit.bobWire, bits))

  if result > globalMax
    globalMax ← result
  end
end
```

Algorithm 7. Evaluation function of Bob

3. Extensions

3.1. Encryption using AES

The *encrypt* and *decrypt* functions in *yao.py* of the Github repo was modified so that they use AES in CTR mode for encryption and decryption.

3.2. Logging

The *Socket* class in *util.py* of the Github repo was modified to store the messages that were sent and received between the two parties and a new function was created which created a JSON file of these messages. In addition to this the result of each circuit evaluation are also stored in the *Socket* class and when the oblivious transfer has started and ended.

3.3. Verification

The verification function is used to check if both the parties have found the correct global max. The function runs after the evaluation is complete, i.e after Bob acknowledges the end message from Alice. The function acts as a third party which gets the global max from both the parties along with their input file paths. Firstly the function checks if both the parties have the same global max, this ensures that the parties have the same value. After it the function computes the global max using the input files, and compares this value to the parties global max. It ensures that both the parties have the true global max. If this condition is also met the function logs 1 in the verification output file, otherwise 0.

4. Execution

In order to run the program, follow the following steps. The program was written and tested using python version 3.11.

- The following external libraries were used, which can be install using the following command. Make sure the correct version is install for each of the library.

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

  - pyzmq (26.0.3) - for socket communication
  - cryptography (42.0.7) - for encryption of garbled tables with AES in CTR mode
  - sympy (1.12) - for prime number manipulation

- The project folder consist 5 python files and a folder necessary to run the program.
  - 4bit_max.json: circuit file which is described in the section above
  - alice.py: implementation of Alice
  - bob.py: implementation of Bob
  - utils.py: helper functions for reading input from input file and converting the circuit's output to a integer value.
  - main.py: runs Alice and Bob in two separate threads and implementation of verification functionality
  - src: modified code from the GitHub repo.

In addition to the files listed above, there are files for Alice's and Bob's input, which will allow you to run the program without creating those files.

In order to run the program move inside the root directory of the project. The program can be ran in two different mode - with verification and without verification.

4.1. With Verification

To the program with the verification functionality run the following command in the root directory.

```
python main.py
```

The command has some optionals flags listed in Table 4 below.

Flag	Default Value	Description
-c	4bit_max.json	Path to the circuit file
-b	4	Number of input wires to the circuit for a single party
-ia	inputs_alice.txt	Path to input file for Alice, the file must present at the location
-ib	inputs_bob.txt	Path to input file for Bob, the file must present at the location
-la	logs_alice.json	Path for log file for Alice, the file is created by the program at that location
-lb	logs_bob.json	Path for log file for Bob, the file is created by the program at that location

Flag	Default Value	Description
-v	verification.txt	Path for verification file, the file is created by the program at that location
--disable-ot	False	Weather the parties should use oblivious transfer or not, using this flag disables oblivious transfer

Table 4. Optional flags for running the program with verification

Figure 6 shows the system output of the program after a successful execution (given that the flags were provided with the correct information). The first line shows the global max from Bob, the second line shows the global max from Alice and the third line shows the weather both the parties have the true global max.

```
Bob global max: 15
Alice global max: 15
Verification: 1
```

Figure 6. System output when running the program with verification

4.2. Without Verification

To run program without the verification functionality, you will need to have two separate terminal windows, navigate to the root directory of the project, and run the following commands (one in each window).

```
python alice.py

python bob.py
```

Both of these commands also have some optional flags, shown in Table 5 and Table 6.

Flag	Default Value	Description
-c	4bit_max.json	Path to the circuit file
-b	4	Number of input wires to the circuit for a single party, must be same for both the parties
-i	inputs_alice.txt	Path to input file for Alice, the file must present at the location
-l	logs_alice.json	Path for log file for Alice, the file is created by the program at that location
--disable-ot	False	Weather the parties should use oblivious transfer or not, using this flag disables oblivious transfer, must be same for both the parties

Table 4. Optional flags for `python alice.py`

Flag	Default Value	Description
-b	4	Number of input wires to the circuit for a single party, must be same for both the parties
-i	inputs_bob.txt	Path to input file for Bob, the file must present at the location
-l	logs_bob.json	Path for log file for Bob, the file is created by the program at that location

Flag	Default Value	Description
--disable-ot	False	Weather the parties should use oblivious transfer or not, using this flag disables oblivious transfer, must be same for both the parties

Table 5. Optional flags for `python bob.py`

Figure 7 and Figure 8 shows the the system output of the program after a successful execution (given that the flags were provided with the correct information). Both the system output show the computed global max by the respectively party.

```
Computed global max: 15
```

Figure 7. System output for `python alice.py`

```
Computed global max: 15
```

Figure 8. System output for `python bob.py`

4.3. Additional Information

The circuit is built for 4 bit unsigned integers, meaning the accept values are between the range of 0 and 15, any value outside this range would cause an error in the program.

The integers values in the input file (.txt) are separated by a single space. In addition the file must contain only a single line with all the values. Example files are present in the submission for reference.

References

[1] Olivier Roques, Emmanuelle Risson, “garbled-circuit”. Available: https://github.com/ojroques/garbled-circuit

[2] javatpoint, “Full Subtractor”. Avaiable: https://www.javatpoint.com/full-subtractor-in-digital-electronics

[3] javapoint, “Multiplexer”. Available: https://www.javatpoint.com/multiplexer-digital-electronics