Integer sum using MPC via Yao's protocol
Project documentation

Voltan Gabriele
gavoltan@edu.aau.at

May 2023

1 Introduction

This project implements secure evaluation of a function between two parties, using Yao's garbled circuit protocol. The aim of the project was to implement a MultiParty Computation (MPC), in order to study its functioning and grasp its potential.

Specifically, my project implements the calculation of the *sum* of Alice and Bob's elements in a secure way, that is, without one participant knowing, or being able to know in the future, the other participant's elements. In summary, the model works as follows:

1. Alice and Bob enter their own set of numbers
2. Alice creates the circuit (as she is the garbler) and sends it to Bob, together with her encrypted inputs
3. Bob evaluates the circuit and sends the result to Alice

The used circuit supports the addition of *8 bit* numbers. This implies that the sum of the numbers of each participant must not exceed 255 (the maximum number that can be represented in binary with 8 bits). This is a hard constraint: if it is violated, the model will stop, reporting the error.

Acknowledgments: this project has been implemented based on the code proposed in the GitHub repository reported in brackets (`https://github.com/ojroques/garbled-circuit`).

2 Dependency installation

The Python version used to implement the project is 3.9. However, the code should work with all versions of Python 3.

The dependencies needed for this to work are:

- **SymPy** used to manipulate prime numbers
- **Cryptography** to encrypt garbled tables with AES in CBC mode
- **ZeroMQ** to implement communication sockets

Many other libraries are used, but all of these are already present in Python 3. To install libraries not present in Python 3 you can run the following command: `$ pip3 install --user pyzmq cryptography sympy`

3 Architecture

The project consists of 4 files located inside the */src* folder. All four files are required for the model to function properly. The python files that you present are:

- **main.py**: implements the correct functioning of Alice and Bob. It also implements the Checker, which takes care of verifying the correctness of the result
- **yao.py**: implements
  - Encryption and decryption functions
  - Evaluation function used by Bob to get the results of a yao circuit
  - GarbledCircuit class which generates the keys, p-bits and garbled gates of the circuit
  - GarbledGate class which generates the garbled table of a gate
- **ot.py**: implement oblivious transfer
- **util.py**: implements many functions related to network communications and asymmetric key generation.[1]

4 Usage

As mentioned in the section 3, all files are contained within the */src* directory. Therefore, all commands executed from the terminal will have to refer to that folder. For simplicity, I recommend moving inside the appropriate directory.

[1] https://github.com/ojroques/garbled-circuit/blob/master/README.md

4.1 Operations

All tests are done on the local network. You can edit the network information in util.py.

1. Run the server (Bob): `$ python3 main.py bob`
2. In another terminal, run the client (Alice): `$ python3 main.py <--mode=> alice`

On both terminals, you will be prompted to enter the corresponding participant inputs. After entering them, Alice and Bob will print the results.

4.2 Alice's usage

Alice has two printing modes: one allows to print its inputs and the outputs of the circuit; the other allows you to print a representation of the garbled tables of the circuit. By default, the first mode is chosen.

- To print Alice's inputs and circuit results: `$ python3 main.py alice -m circuit` or just `$ python3 main.py alice`
- To print the garbled tables of the circuit: `$ python3 main.py alice -m table`

5 JSON Circuit

A function is represented as a boolean circuit using some of the available gates from the original project:

- AND
- OR
- XOR

A few assumptions are made (from the original documentation):

- 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.

An example of the circuit representation can be found within the original documentation.[2]

[2]https://github.com/ojroques/garbled-circuit/blob/master/README.md

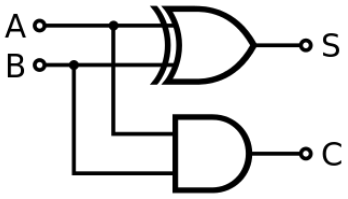

Figure 1: Half-adder circuit

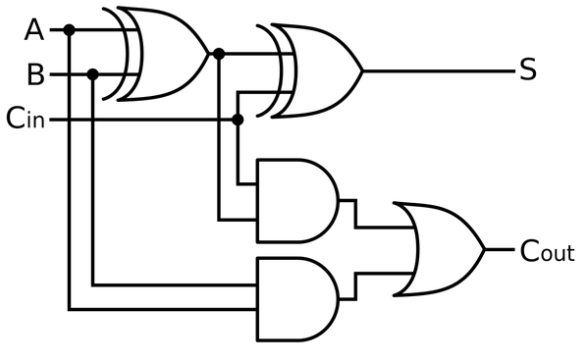

Figure 2: Full-adder circuit

6 Implementation

6.1 Function and circuit

In my model I decided to implement 8 bit sum. Since adding two 8-bit numbers can result in a 9-bit number, the output of the circuit will have 9 bits. Alice and Bob will enter their respective sets of numbers and then calculate the sum of them. By doing so, they will not reveal their numbers (important security requirement). At this point, through a *Half-adder* and seven *Full-adder*, the sum of the two numbers will be calculated.

As we can see from the circuits shown, the difference between the two circuits lies in the inputs: the half adder 1 [3], having to calculate the first bit of the result, does not need to receive the carry as input, as it does not exist. In output however, both return the result bit and the respective carry, which will be input to the next full-adder 2 [4] (except for the last one).

6.2 Yao's encryption

As for the encryption system used by these Yao's implementation, I chose AES mode CBC and the IV will be prepended to the ciphertext from the encryption function.

7 Pseudocode

In order to better understand the behavior of Alice and Bob, the pseudocodes of the two participants are now shown.

[3]https://www.techopedia.com/definition/7509/half-adder
[4]https://www.eeweb.com/full-adder-nand-equivalent/

Data: set_alice
Result: sum of Alice and Bob's numbers
create_circuit();
print(Checker.expected_output);
send(circuit);
send_inputs_with_ot();
receive_results_from_Bob();
Checker.get_correctness();
send_all_results_to_Bob();

Algorithm 1: Alice's step - Garbler

Data: set_bob
Result: sum of Alice and Bob's numbers
receive_circuit();
receive_Alice_inputs();
choose_inputs();
compute_results_using_ot();
send_evalutation_to_Alice();
receive_results_from_Alice();

Algorithm 2: Bob's step - Evaluator

As can be seen from the pseudocode, Alice, after receiving the evaluation from Bob, calculates the correctness of the result using the Checker and finally sends all the results to Bob. This step was not present in the original implementation, but was added as required by the project specifications.

8 System outputs

In order to understand the correct interpretation of the system outputs, examples of executions are shown.

```
gabrielevoltan@MBP-di-Gabriele src % python3 main.py alice
[INPUT] What is the number of elements in the Alice's set?
Enter it now: 3
[INPUT] Now you need to enter the elements of the Alice's set!
The input format is: num1, num2, num3 ...
Enter them now: 9, 20, 40

======== 8-bit full adder ========

Alice[31, 32, 33, 34, 35, 36, 37, 38] = 1 0 1 0 0 0 1 0
Outputs[112, 122, 132, 142, 152, 162, 172, 182, 185] = 0 0 1 0 1 1 1 0 0

[EXPECTED OUTPUT] The sum of the elements from Alice and Bob should be: 116
[OUTPUT] The sum of the elements is: 116
[CORRECT] The yao's output is equal to the one computed in normal way.
```

Figure 3: Alice's terminal

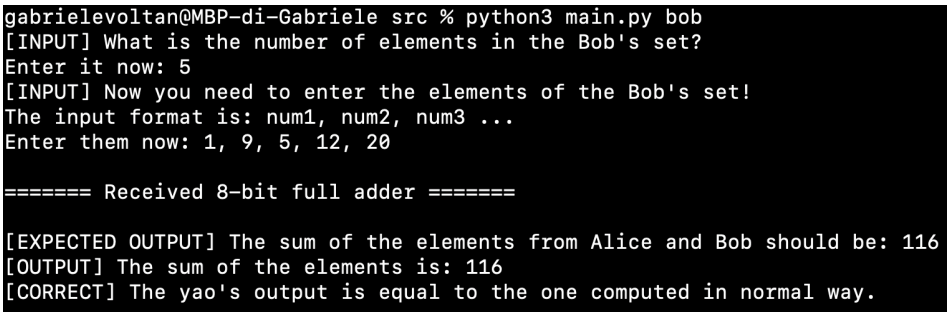

Figure 4: Bob's terminal

The output shown in the figure is that of the classic execution, in *circuits* mode. As we can see, the output of Alice 3 and that of Bob 4 are very similar. Alice shows her inputs and the results of the circuit, both showing the expected result, the calculated result and the correctness.

The output shown in the 5 figure instead shows an execution of Alice in *table* mode. In this mode, Alice prints all the garbled tables for each logic gate in the circuit. Note that the image does not show all of the output as it is very long.

```
gabrielevoltan@MBP-di-Gabriele src % python3 main.py alice -m table
[INPUT] What is the number of elements in the Alice's set?
Enter it now: 3
[INPUT] Now you need to enter the elements of the Alice's set!
The input format is: num1, num2, num3 ...
Enter them now: 1, 10, 100
======== 8-bit full adder ========
P-BITS: {131: 1, 132: 0, 133: 0, 134: 0, 135: 1, 141: 0, 142: 1, 143: 0, 144: 0, 145: 0, 151: 1, 152: 1, 153: 1, 154: 0, 155: 1, 31:
1, 32: 0, 33: 0, 34: 1, 35: 0, 161: 0, 36: 1, 162: 0, 163: 1, 164: 0, 165: 0, 37: 0, 171: 1, 172: 1, 173: 1, 174: 1, 175: 0, 181: 1,
182: 0, 183: 1, 184: 0, 185: 0, 61: 1, 62: 1, 63: 0, 64: 0, 65: 0, 66: 0, 67: 1, 68: 0, 38: 1, 111: 1, 112: 1, 121: 1, 122: 0, 123: 0
, 124: 0, 125: 1}
GATE: 111, TYPE: AND
[0, 0]: [31, 1][61, 1]([111, 1], 0)
[0, 1]: [31, 1][61, 0]([111, 0], 1)
[1, 0]: [31, 0][61, 1]([111, 0], 1)
[1, 1]: [31, 0][61, 0]([111, 0], 1)
GATE: 112, TYPE: XOR
[0, 0]: [31, 1][61, 1]([112, 0], 1)
[0, 1]: [31, 1][61, 0]([112, 1], 0)
[1, 0]: [31, 0][61, 1]([112, 1], 0)
[1, 1]: [31, 0][61, 0]([112, 0], 1)
GATE: 121, TYPE: XOR
[0, 0]: [32, 0][62, 1]([121, 1], 0)
[0, 1]: [32, 0][62, 0]([121, 0], 1)
[1, 0]: [32, 1][62, 1]([121, 0], 1)
[1, 1]: [32, 1][62, 0]([121, 1], 0)
GATE: 122, TYPE: XOR
[0, 0]: [111, 1][121, 1]([122, 0], 0)
[0, 1]: [111, 1][121, 0]([122, 1], 1)
[1, 0]: [111, 0][121, 1]([122, 1], 1)
[1, 1]: [111, 0][121, 0]([122, 0], 0)
GATE: 123, TYPE: AND
[0, 0]: [32, 0][62, 1]([123, 0], 0)
[0, 1]: [32, 0][62, 0]([123, 0], 0)
[1, 0]: [32, 1][62, 1]([123, 1], 1)
[1, 1]: [32, 1][62, 0]([123, 0], 0)
GATE: 124, TYPE: AND
[0, 0]: [111, 1][121, 1]([124, 1], 1)
[0, 1]: [111, 1][121, 0]([124, 0], 0)
[1, 0]: [111, 0][121, 1]([124, 0], 0)
[1, 1]: [111, 0][121, 0]([124, 0], 0)
GATE: 125, TYPE: OR
[0, 0]: [123, 0][124, 0]([125, 0], 1)
[0, 1]: [123, 0][124, 1]([125, 1], 0)
[1, 0]: [123, 1][124, 0]([125, 1], 0)
[1, 1]: [123, 1][124, 1]([125, 1], 0)
GATE: 131, TYPE: XOR
[0, 0]: [33, 0][63, 0]([131, 0], 1)
[0, 1]: [33, 0][63, 1]([131, 1], 0)
[1, 0]: [33, 1][63, 0]([131, 1], 0)
[1, 1]: [33, 1][63, 1]([131, 0], 1)
GATE: 132, TYPE: XOR
[0, 0]: [125, 1][131, 1]([132, 0], 0)
[0, 1]: [125, 1][131, 0]([132, 1], 1)
[1, 0]: [125, 0][131, 1]([132, 1], 1)
[1, 1]: [125, 0][131, 0]([132, 0], 0)
```

Figure 5: Alice's terminal - table mode