LAB – 4(Bonus)

Name: Neetha Adhmoolam(ID: 11552464), Venkata Sai Susanth Gunda(11602605)

Task 1:

A cryptographic technique called Yao's Garbled Circuits was developed by Andrew Yao in the 1980s and is used to carry out safe two-party computation. This protocol aims to enable two parties to compute a function together over their private inputs while keeping those inputs hidden from one another. To put it briefly, the way the protocol operates is that one party (referred to as the "garbler") encrypts the input and the function so as to hide the true values. After receiving this encoded data, the other party (referred to as the "evaluator") uses a sequence of processes to compute the result without discovering anything about the inputs save the final output.

For every gate in the circuit, an encoding entails generating a "garbled table" that represents every conceivable pairing of inputs and matching encrypted outputs. These tables are sent to the evaluator by the garbler, who utilizes them to assess the circuit gate by gate. Without learning about the individual inputs, the evaluator only receives the end outcome. Yao's Garbled Circuits are a valuable tool for safe multiparty computation in situations where privacy is important because, in essence, they allow two parties to jointly calculate a function while maintaining the privacy of their inputs.

Referece:

- https://en.wikipedia.org/wiki/Garbled_circuit
- https://medium.com/zkpass/yaos-classical-garbled-circuits-4cbdbadf288e#:~:text=The%20garbled%20circuit%20is%20a,in%20the%20garbled%20circuit%20protocol.

Task 2:

```
student@na0557:~$ python --version
Python 3.10.12
```

- Checking for the current version of python.

```
student@na0557:~$ git clone https://github.com/kalyancheerla/SSS_Lab4_Garbled_Circuit_Protocol.git
Cloning into 'SSS_Lab4_Garbled_Circuit_Protocol'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 11 (delta 2), reused 7 (delta 2), pack-reused 0
Receiving objects: 100% (11/11), 5.35 KiB | 322.00 KiB/s, done.
Resolving deltas: 100% (2/2), done.
student@na0557:~$ cd SSS_Lab4_Garbled_Circuit_Protocol/
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ pip install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Collecting pycryptodome
  Using cached pycryptodome-3.19.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.19.0
```

- Clone the repository and installing the dependencies required.

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 0 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 0
Kindly check your arguments, ['XOR', '0']
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 0 1
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 1 0
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 1 1
Output is 0
```

- The script to produce an output saying that 0 is the outcome of XORing 0 and 0.
- The script ought to produce a message stating that XORing 0 and 1 yields 1.
- The script ought to produce an output specifying that 1 is the outcome of XORing 1 and 0.
- The script should produce an output saying that XORing 1 and 1 yields 0.

Task 3:

- To print the encoded values of A and B, uncomment the lines in the main function.

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 1 1
Encoded A (16bytes): b'\x1d\x17i\xe4\x94\xbb}\x0f\xcf\xbd\xa3\xb0^\x8e&\xf8'
Encoded B (16bytes): b'qE\xb7D\xe7\xf4j\xf9\t,\xa5\x19n\xde\r\xe6'
Encoded O (16bytes): b'\xa1\x14\xd4\xc3"\xf2-T\xac\xd1\x17t\xef2\x11\x14'
Output is 0
```

- The encoded values of A , B and O are printed.

Task4:

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XOR 1 1
Garbled Table:
                16bytes - b'\x13\x8ct\xef#\xf37\x80\x9f\xd3\x18w\x0eQ\xd0\x02'
                16bytes - b'\xef\xe0\x18p\xbd\xef!\xd4\xb1\xbe\xfe\xdb&\x1bd&'
                16bytes - b'\xf3\xc5\x9c\xef\xa6}\x8ar@\xfd6\xb3\xf9]\xa4\x96'
                16bytes - b'\xf1\xb8\xb4{\xf5\xdc\\\n\x0c\xd9\xfeIil\xb6\x07'
Output is 0
```

- The Garbled Table, a mapping of all feasible input combinations to matching output labels, is shown in this section. An input combination and the corresponding output label are displayed on each line. The labels for the inputs (0, 0), (0, 1), (1, 0), and (1, 1) are displayed in this example. In order to preserve privacy while calculating the intended function, the Garbled Circuit protocol processes and encrypts the inputs, as shown by the output.

Task 5:

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py AND 0 0
Output Labels: [b'\x960\x03%0\xd2@\x9c\xe6DE\xcc\xca@\t\xc5', b'0\xcb\xc8Q\xd1\xa4\n\xaaO\x82\xf9u\xbf\\\xc6U']
Garbled Table:
               16bytes - b'I#\xf2\xe0S\xff\xfa\xa2\x10Z)\xbdn\xc0,\x99'
               16bytes - b'\xe3GA\xa0BT\xb0\x97y3\xf0\xa1\xb2\x80\xda\n'
               16bytes - b'\x0c\xf8\x0bS\x7fG\x13\xa8\xa0\x08\xb5\xbc\xd9\xa7\tL'
               16bytes - b'\xafN\xc5}\x13\x0c\xd9f\xdf\x00\xb6*\xf7\x94\xac\x0b'
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py AND 0 1
Output Labels: [b'\xd8\x80\x10\xf6\x9b\xc7\x1d\x1a\xbce\xff\x13sn\x91\x8f', b'\xa5\x9b\xbb`x\xa4:\x1d|\xcc\t\x0f+\x87\x99+']
Garbled Table:
               16bytes - b'3\x98@}v\x06\x19L\xbfp\x19))\xc3\t\xc8'
               16bytes - b'\xb7\x17\xfc\xa5\xec\x11\xa9\x06\xa5\xda\xac&\xac\xd3\xb4\x8f'
               16bytes - b'w\xf4\x92N[\xc7\xc0\xe8 \xb0\xbfU\x1f2e,'
               16bytes - b'+\xb6\xfb\xfeT\xf0\x1e\xc3\x02\xee\x99^}\xe7\x14A'
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py AND 1 0
Output Labels: [b'<\xe2\xfa\xf5\xa89-\x0f\xb9U\xfd:\xfe}F!', b"?\x81-\xd6\x16\xa3\xce\x1aR\xcf\xe7'\xc3Kp\xec"]
Garbled Table:
               16bytes - b'\x95\xfc8\xc1\x99L\xb7v&i\x88\xe8tN\x9b\xaa'
               16bytes - b'\xc5<B\xedh[\x90\xe4\xce\xa2fm\xd7\xd7\xf8\x1b'
               16bytes - b'\xe7E\x95\xcf\x8bG\xa4\x948\xc4\xad\x98Y}\xb3\xdf'
               16bytes - b'z\x87\xafGk \x80\xbf\xc7\x1a\x0fA\xa22\x92\xd6'
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py AND 1 1
Output Labels: [b'A\x99\xbd\xa7Wx@\x14U\x91\xb4\xd3\xd3h\x05(', b'\x95V\xc8\x7f\xdcZ\x075\xa8\xca\x04\x0cl\xfe\xb8Q']
Garbled Table:
               16bytes - b'\xba+\xf7\xfc\xe8\xcao\xc5\xf1\x11$+O:A6'
               16bytes - b'\xaad\xe8\xa8/!\x96D\xa3\xc4,\xe6\xf9\xf0\x91$'
               16bytes - b'\x8e\x9a\x96N\xbcR\x84h\xb6\x1cE@\xde\xf8\xee\x84'
               16bytes - b'\xa5\xf6h[\xd2s]\xe8\x03\x1db\x0bg~\x19\xfa'
Output is 1
```

- As like the OR operation I have printed all the AND operation as the output and the Garbled Circuit protocol processes and encrypts the inputs, as shown by the output.

Task6:

```
##############################################################################
# Logic Gates
##############################################################################
def AND_gate(a, b): return a & b
def XOR_gate(a, b): return a ^ b
def OR_gate(a, b):  return a | b
def XNOR_gate(a, b): return ~(a^b) & 1
##############################################################################
```

- Defind a funtion calleed XNOR as : ~(a^b)&1

```
def main(argv):
    gates = {
                'AND': AND_gate,
                'XOR': XOR_gate,
                'OR' : OR_gate,
                'XNOR': XNOR_gate,
            }
```

- In the main function named the XNOR_gate as XNOR.

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ nano onegate_gc.py
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 1
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ nano onegate_gc.py
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 1
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ nano onegate_gc.py
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 0
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  0 1
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  1 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python onegate_gc.py XNOR  1 1
Output is 1
```

- I Printed out all the output for the XNOR gate.

Task 7:

In a Garbled Circuit protocol, the Garbler Logic entails encoding inputs, creating a Garbled Table with corresponding labels, and safely sending this data to an assessor. The evaluator uses the garbled data to do calculations, and the Garbler decodes the outcome to protect privacy all along the way.

Task 8:

Receiving garbled data, searching through entries in the Garbled Table, safely assessing the gate action, producing an encoded output, and returning it to the garbler are all part of the Evaluator Logic of a Garbled Circuit protocol. The protocol's privacy-preserving feature guarantees that the evaluator can calculate the outcome without learning the inputs' true values.

Task 9:

The Advanced Encryption Standard (AES) is the encryption cipher that is being utilized for double encryption. The Cipher, Crypto.Both cipher1 and cipher2 in the double_encrypt and double_decrypt functions employ the AES module from the Crypto library.In terms of switching to a different symmetric cipher, it is theoretically feasible to use one instead of AES. But it's crucial to make sure the cipher you use is safe and appropriate for the particular cryptographic needs. The system's overall security may be impacted by the encryption selected.

Should you alter the double encryption's order in this manner:

- cipher1.encrypt(cipher2.encrypt(data)) is returned.

Next, you should decrypt in the opposite order than when you started:

- cipher2.decrypt(cipher1.decrypt(data)) is returned.

To keep the encryption and decryption processes running in the proper order, this modification is required. In response to your inquiry concerning numerous parties and Multi-Party computing (MPC), it appears that the code you provided is intended primarily for a two-party computing scenario. In

multiparty computation (MPC), parties work together to compute a function over their private inputs. Significant modifications must be made to the MPC code, and extra cryptographic protocols—such as safe multi-party computation protocols like Yao's Garbled Circuits or more recent ones like the GMW protocol—are typically needed.

The existing code is a condensed example that can be used to learn the fundamentals of secure function evaluation between two parties and for educational purposes. It is not possible to modify the code to make it work for a more broad MPC scenario; instead, a more sophisticated cryptographic protocol would be needed. MPC is usually implemented in real-world applications using professional cryptographic libraries that support it, such as PySyft for PyTorch.

Task 10 :

(A OR B) AND C (TRUTH TABLE):

A	B	C	Result
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Modifies the main code:

```
# Check for the inputs
if not ((len(argv) == 4) and (argv[0] in gates.keys()) and \
   (argv[1] in ['0', '1']) and (argv[2] in ['0', '1']) and (argv[3] in ['0', '1'])):
    print(f'Kindly check your arguments, {argv}')
    exit(-1)

# Generate Input labels and Garbled table
A_labels, B_labels, C_labels = generate_labels(), generate_labels(), generate_labels()

# Garble the OR gate between A and B
AB_labels, OR_table = garble_gate(gates['OR'], A_labels, B_labels)

# Garble the AND gate between (A OR B) and C
ABC_labels, AND_table = garble_gate(gates['AND'], AB_labels, C_labels)

# Evaluation starts here
A, B, C = int(argv[1]), int(argv[2]), int(argv[3])
encoded_A = A_labels[A]
encoded_B = B_labels[B]
encoded_C = C_labels[C]

# Evaluate the OR gate between A and B
encoded_AB = evaluate_garbled_gate(OR_table, encoded_A, encoded_B,AB_labels)

# Evaluate the AND gate between (A OR B) and C
encoded_ABC = evaluate_garbled_gate(AND_table, encoded_AB, encoded_C,ABC_labels)

# Convert the result back to an integer
```

```
    # Convert the result back to an integer
    ABC = ABC_labels.index(encoded_ABC)

    print(f'Output is {ABC}')

if __name__ == '__main__':
    main(sys.argv[1:])
```

Generated output:

```
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 0 0 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 0 0 1
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 0 1 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 0 1 1
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 1 0 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 1 0 1
Output is 1
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 1 1 0
Output is 0
student@na0557:~/SSS_Lab4_Garbled_Circuit_Protocol$ python twogate_gc.py OR 1 1 1
Output is 1
```