Problem 3 Implementing Secure Multiparty Computation (MPC) using SFDL

Alice holds a private Boolean vector A with 10 Boolean entries ($\{0, 1\}^{10}$) while Bob holds another private Boolean vector B with another 10 Boolean entries ($\{0, 1\}^{10}$). Design and implement a protocol using the Fairplay to securely compute the scalar product A · B without sharing their inputs with each other. For example, if B = [0, 1, 0, 0, 1, 1, 0, 1, 1, 1] and A = [1, 1, 1, 1, 0, 1, 1, 1, 1, 1], the scalar product A · B = 5.

- **The scalar product computation should be converted to garbled circuits using SFDL.**
- **Fairplay secure function evaluation: https://www.cs.huji.ac.il/project/Fairplay/.**
- **Readme file for running Fairplay SFE: https://www.cs.huji.ac.il/project/Fairplay/Fairplay/Readme.txt**

Tasks:

1. Alice generates Boolean entries for A and Bob generates Boolean entries for B.
2. Write the SFDL program for Alice and Bob.
3. Compile it for Alice and Bob, and run the protocol (communication is integrated in Fairplay).
4. Report the input Boolean vectors, the SFDL program, SHDL circuit, and output results A · B (for two parties).

Submission requirement:

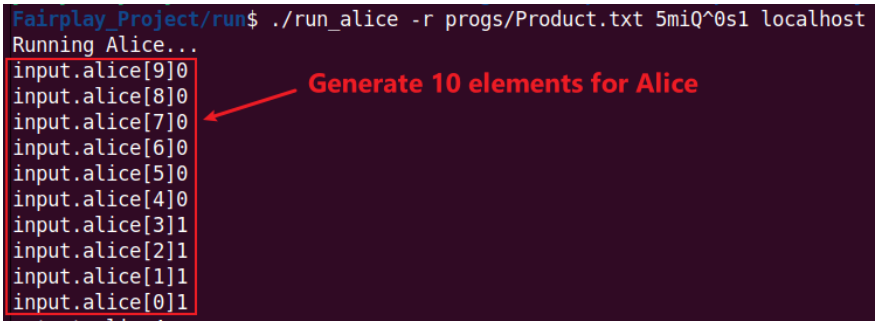
(1) a report including the protocol design, implementation details, input matrices and output results,
(2) screenshots of the major steps of each task and your answers (you can explain your findings with tables or figures), and
(3) source code files – all named with the prefix "hw3-3-" (e.g., hw3-3-report.pdf, and hw3-3-matrix.txt).

My solution:

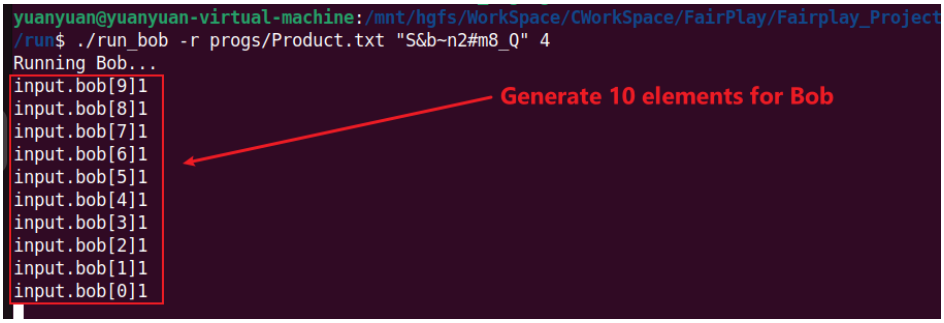
Task 1. Alice generates Boolean entries for A and Bob generates Boolean entries for B.

Based on the explanation of the program that is implemented based on Secure Function Definition Language (SFDL) in the paper, we need to input values that are generated values on the GUI of this SFDL program.

For Alice, the input values in my experiment is a vector A = [0,0,0,0,0,0,1,1,1,1]



For Bob, the input values in my experiment is a vector B = [1,1,1,1,1,1,1,1,1,1]



Note: A*B=4 here. We can verify this scalar product value via the SFDL program in the following tasks and steps.

Task 2. Write the SFDL program for Alice and Bob.

The scalar product SFDL program is designed based on the below program structure:

```
program <program-name> {
   <type declarations>
   <function declarations>
}
```

Here is the type I defined:

```
/*
 * Compute the Product of two sorted arrays
 */
program Product {

// Constants
// Define the size based on the size of vector A,B = 10
const inp_size = 10;

// Type Definitions

type Elem  =  Int<16>;
type AliceInput = Elem[inp_size];
type AliceOutput = Int<16>;
type BobInput = Elem[inp_size];
type BobOutput = Int<16>;

type Input = struct {AliceInput alice, BobInput bob};
type Output = struct {AliceOutput alice, BobOutput bob};
```

Here is the function I defined:

```
// Function Definitions

// This is the main function
function Output output(Input input) {
    var Int<8> Outputproduct;
    var Int<8> i;
    Outputproduct = 0;

    for (i = 0 to inp_size - 1) {
       Outputproduct = Outputproduct + (input.alice[i] & input.bob[i]);
    }

    output.alice = Outputproduct;
    output.bob = Outputproduct;
}
```

Task 3. Compile it for Alice and Bob, and run the protocol (communication is integrated in Fairplay).

Step 1. Compile the product SFDL program:

Compile for the party who is called Bob:

```
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/Wo
/run$ ./run_bob -c progs/Product.txt
Program compiled.
Performing multi-to-single-bit transformation.
Transformation finished.
Unique vars transformations.
Unique vars transformations finished.
Program Optimization: Phase I.
Program Optimization: Phase II.
Optimization finished.
Writing to circuit file.
Completed.
Writing to format file.
Completed.
```

Compile for the party who is called Alice:

```
yuanyuan@yuanyuan-virtual-machine: /mnt/hgfs/WorkSpace/C...
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/WorkSpace/CW
Fairplay_Project/run$  ./run_alice -c progs/Product.txt
Program compiled.
Performing multi-to-single-bit transformation.
Transformation finished.
Unique vars transformations.
Unique vars transformations finished.
Program Optimization: Phase I.
Program Optimization: Phase II.
Optimization finished.
Writing to circuit file.
Completed.
Writing to format file.
Completed.
```

After compiling, we see both commands produce the same two output files:
1. progs/Product.txt.Opt.circuit (an SHDL circuit)
2. progs/Product.txt.Opt.fmt

```
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/WorkSpace/CWorkSpace/FairPlay/Fairplay_P
/run/progs$ ll
total 45
drwxrwxrwx 1 root root  4096 Mar 15 17:56 ./
drwxrwxrwx 1 root root  4096 Mar 15 17:56 ../
-rwxrwxrwx 1 root root   462 May 14  2004 And.txt*
-rwxrwxrwx 1 root root   531 May 14  2004 Billionaires.txt*
-rwxrwxrwx 1 root root   586 May 14  2004 KDS.txt*
-rwxrwxrwx 1 root root   898 May 14  2004 Median.txt*
-rwxrwxrwx 1 root root   529 May 14  2004 Millionaires.txt*
-rwxrwxrwx 1 root root   790 Mar 15 17:41 Product.txt*
-rwxrwxrwx 1 root root 29574 Mar 15 17:52 Product.txt.Opt.circuit*
-rwxrwxrwx 1 root root  2155 Mar 15 17:52 Product.txt.Opt.fmt*
```

The two output files

Note: the SHDL circuit is Product.txt.Opt.circuit

Step 2. Run the protocol

Communication is integrated in Fairplay. According to theory and explanation in the paper (Fairplay — A Secure Two-Party Computation System), it's considered several flavors of oblivious transfer(OT) algorithms in this Fairplay tool. Since 4 is the best one among oblivious transfer(OT) algorithms, I will use OT type = 4 to run the protocol.

Run Bob (should be first):
./run_bob -r progs/Product.txt "S&b~n2#m8_Q" 4

```
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/WorkSpace/CWorkSpace/FairPlay/Fairplay_Project
/run$ ./run_bob -r progs/Product.txt "S&b~n2#m8_Q" 4
```

Note: For Bob, the input values in my experiment is a vector B = [1,1,1,1,1,1,1,1,1,1]
After using the Fairplay to securely compute the scalar product A · B without sharing their inputs with each other, we get is 4.

```
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/WorkSpace/CWorkSpace/FairPlay/Fairplay_Projec
/run$ ./run_bob -r progs/Product.txt "S&b~n2#m8_Q" 4
Running Bob...
input.bob[9]1
input.bob[8]1
input.bob[7]1
input.bob[6]1
input.bob[5]1
input.bob[4]1
input.bob[3]1
input.bob[2]1
input.bob[1]1
input.bob[0]1
output.bob4
```

The scalar product we get after running the Fari play to securely compute

Run Alice (should be second):

./run_alice -r progs/Product.txt 5miQ^0s1 localhost

```
Fairplay_Project/run$ ./run_alice -r progs/Product.txt 5miQ^0s1 localhost
Running Alice...
```

Note: For Alice, the input values in my experiment is a vector A = [0,0,0,0,0,0,1,1,1,1]

After using the Fairplay to securely compute the scalar product A · B without sharing their inputs with each other, we get is 4.

```
yuanyuan@yuanyuan-virtual-machine:/mnt/hgfs/WorkSpace/CWorkSpace/FairPlay/
Fairplay_Project/run$ ./run_alice -r progs/Product.txt 5miQ^0s1 localhost
Running Alice...
input.alice[9]0
input.alice[8]0
input.alice[7]0
input.alice[6]0
input.alice[5]0
input.alice[4]0
input.alice[3]1
input.alice[2]1
input.alice[1]1
input.alice[0]1
output.alice4
```

The scalar product we get after using the Fairplay to securely compute

Hence, the SFDL program verifies A*B = 4.

Task 4. Report

(1) The input Boolean vectors in Task 1 are:

A = [0,0,0,0,0,0,1,1,1,1]

B = [1,1,1,1,1,1,1,1,1,1]

(2) The SFDL program for Task 2 is hw3-3-Product (SFDL program).txt. The key code is shown in Task 2.

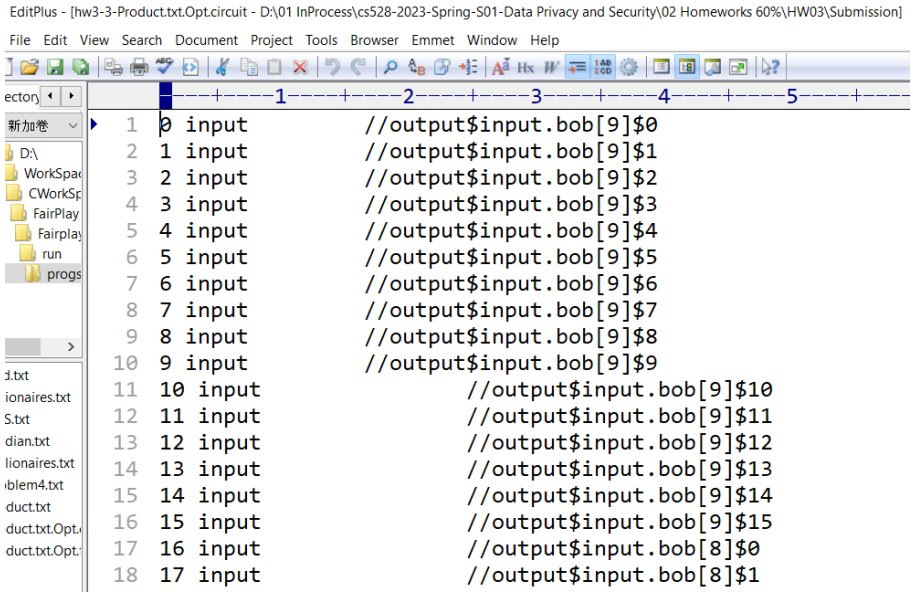
When you run this script, make sure:

a) You will run on a Linux system with JVM has been installed.

b) Run the script based on the readme file which is hw3-3-Readme-Product.txt. You might need to edit the name of the scripts. Because we are required to upload all documents in certain Naming Conventions, I have to follow the requirements and add the prefix 'hw3-3'.

(3) SHDL circuit is hw3-3-Product.txt.Opt.circuit

For a detailed circuit, please see the document: hw3-3-Product.txt.Opt.circuit. Here is the partial display of the circuit.



(4) The output results A · B for two parties are 4 after using the Fairplay to securely compute the scalar product A · B without sharing their inputs with each other.