Eloy Ramirez Hernanz
CWID: A20433696

Part I

1. **Alice generates random nonnegative integer entries for A while Bob generates random nonnegative integer entries for B.**

   As it can be seen in the code of both scripts, both Alice and Bob generate its own matrix, with random numbers from 0 to 10 to avoid complex calculations. The matrix of Alice is A and the matrix of Bob is B. In the following picture it is possible to see the code for generating each matrix in each script, and the values of them.

   Images of the code used for generating the random matrix (by changing the 10, you can select the range of values of the numbers of the matrix):

```
#Generation of random matrix A, using numbers between 0 and 10.
A = np.random.randint(10, size=(5, 8))

#Generation of random matrix B, using numbers between 0 and 10.
B = np.random.randint(10, size=(8, 4))
```

   Matrix A and B used for keys of 512 bits:

```
                                [[7 0 0 3]
                                 [9 6 1 7]
                                 [5 3 6 6]
[[8 6 8 8 2 5 5 2]               [1 3 6 0]
 [9 5 0 4 7 7 7 9]               [2 2 5 8]
 [1 5 6 2 4 8 4 0]               [0 7 9 4]
 [1 6 1 8 7 2 5 0]               [5 1 0 7]
 [8 2 9 9 8 8 8 7]]              [6 7 8 7]]
```

   Matrix A and B used for keys of 1024 bits:

```
                                [[7 5 6 0]
                                 [4 7 2 3]
                                 [0 1 4 2]
[[4 0 9 6 7 0 4 1]               [4 7 1 9]
 [4 8 7 4 6 3 7 7]               [8 9 5 9]
 [7 1 2 4 6 3 2 4]               [7 9 1 0]
 [7 1 3 6 2 3 2 4]               [0 8 7 7]
 [5 7 9 0 7 5 8 0]]              [9 6 2 8]]
```

2. Design the protocol between Alice and Bob to perform secure computation.

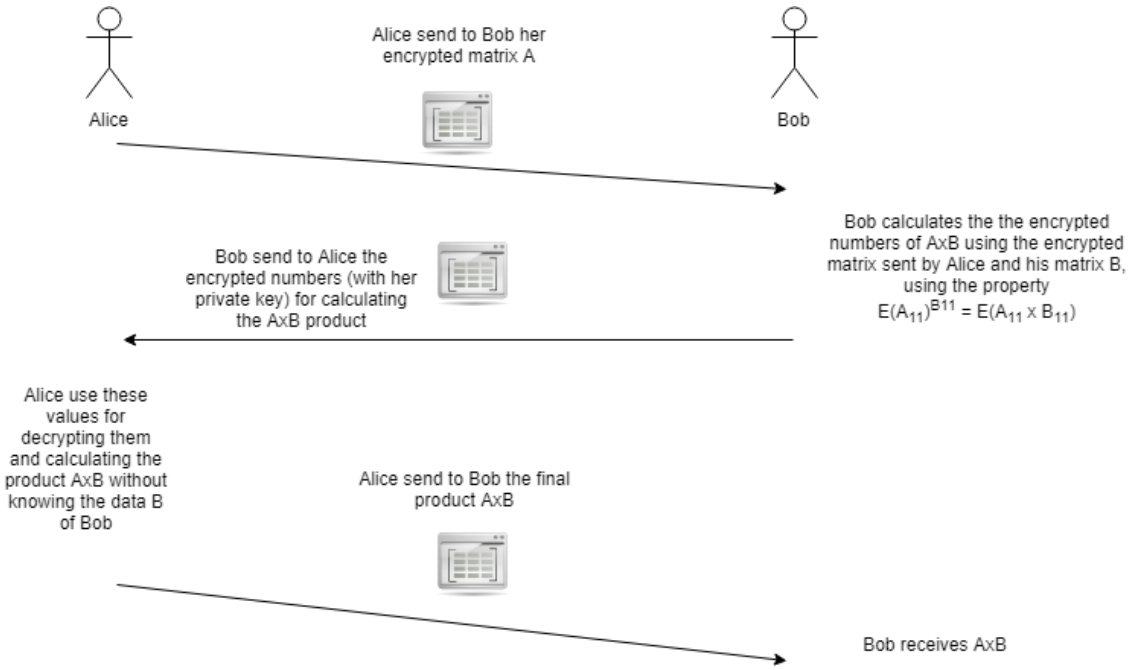
For the secure computation, the protocol consists of creating a private key and public key for encrypt and decrypt the values of the matrix. Then, send to the other party the matrix encrypted with the public key using socket communication, and the receiver could use HE for obtaining the result matrix AxB encrypted. For obtain each value of this matrix, which is a sum of the product of numbers of both matrix, I have used the following property in each of the products which have to be done for obtaining the AxB encrypted:

$$E(A_{11} \times B_{11}) = E(A_{11})^{B11}$$

This case is the same as I did on the script, where Alice send to Bob the encrypted values of the matrix A ($E(A_{11})$ in the example of above) and Bob power these values with the values of his matrix B without encrypt them, obtaining the encrypted result of each element that has to be summed to obtain each element of the AxB matrix ($E(A_{11} \times B_{11})$ in the above example).

After using a loop for obtaining the encrypted AxB in the receiver, then this matrix is sent to the sender using sockets again. When it is received, the sender can decrypt all the values of the matrix with the private key and sum them for obtaining the matrix AxB required.

Following this protocol, neither of both parties of the communication shows their information (matrix A and matrix B), but can obtain an operation where have been used both information (the product of AxB in this problem).



3. **Write the programs for Alice and Bob: communication should be established to exchange encrypted messages, e.g., using Socket programming.**

   The code of the programs can be seen in the files Alice.py and Bob.py in the submitted folder, which is explained with comments.

4. **Report the input matrices, the last ciphertext (right before the decryption) and the decrypted product AxB using two different key sizes 512-bit and 1024 bit.**
   Here I will show you the execution of the protocol between Alice and Bob, one using 512 bit key and another 1024 key bit.

   Results with 512 bit key

   Input matrices:

```
                          [[7 0 0 3]
                           [9 6 1 7]
                           [5 3 6 6]
[[8 6 8 8 2 5 5 2]         [1 3 6 0]
 [9 5 0 4 7 7 7 9]         [2 2 5 8]
 [1 5 6 2 4 8 4 0]         [0 7 9 4]
 [1 6 1 8 7 2 5 0]         [5 1 0 7]
 [8 2 9 9 8 8 8 7]]        [6 7 8 7]]
```

   Last ciphertext (the one which includes the values for obtaining the AxB product, decrypted and used by Alice for obtaining AxB):

0	list	32	[263244557457883864017157952185073338079718166060665836607754324685032 ...
1	list	32	[458527689395929765828556830608932680159175134951166370780442195508919 ...
2	list	32	[469490753227240800412657789380069063641639035807416054484388961204127 ...
3	list	32	[258313817137804884714160948216153228698877499468479582380968696220971 ...
4	list	32	[204698803207033902568890689724084952476664873323451801804772762034368 ...

   As it can be seen, there are 32 values per row, because each row of AxB has 4 elements, each composed of the sum of 8 values of these 32 (32/4 = 8 values to sum per element of row). These 32 values are the ones of kind $E(A_{11} \times B_{11}) = E(A_{11})^{B11}$ (This example would be the first value to sum of the first element of the first row of AxB)
   I will put the value in the txt file, but it is so long.

Decrypted matrix AxB

Índice	Tipo	Tamaño	
0	list	4	[199, 142, 173, 199]
1	list	4	[215, 175, 199, 258]
2	list	4	[112, 122, 145, 166]
3	list	4	[113, 96, 113, 150]
4	list	4	[226, 195, 278, 293]

In the last picture it can be seen the rows of the matrix AxB, which are 5 and have 4 elements each one (4 columns).

Results with 1024 bit key

Input matrices:

```
                    [[7 5 6 0]
                     [4 7 2 3]
                     [0 1 4 2]
[[4 0 9 6 7 0 4 1]   [4 7 1 9]
 [4 8 7 4 6 3 7 7]   [8 9 5 9]
 [7 1 2 4 6 3 2 4]   [7 9 1 0]
 [7 1 3 6 2 3 2 4]   [0 8 7 7]
 [5 7 9 0 7 5 8 0]] [9 6 2 8]]
```

Last ciphertext (the one which includes the values for obtaining the AxB product, decrypted and used by Alice for obtaining AxB):

Índice	Tipo	Tamaño	Valor
0	list	32	[172812847765145751397323927841925695912372315034268746219048614698725 ...
1	list	32	[914810524168481305831051072235066925712708380451219818814857823235760 ...
2	list	32	[729872972318600043362058724766509019789721548061984840505808616714636 ...
3	list	32	[239806509589653717085376936247168919556278323139040525489077857116492 ...
4	list	32	[271092062395147446025996917706898171304254743199642334399643585647506 ...

Same explanation as the one of 512.
I will put the value in the txt file, but it is so long.

Decrypted matrix AxB

Índice	Tipo	Tamaño	
0	list	4	[117, 172, 131, 171]
1	list	4	[208, 290, 168, 233]
2	list	4	[174, 193, 111, 143]
3	list	4	[150, 172, 97, 127]
4	list	4	[154, 255, 176, 158]

Eloy Ramirez Hernanz
CWID: A20433696

Part II

1. **Alice generates random Boolean entries for A while Bob generates random Boolean entries for B.**
   This is done by introducing in the terminals the values of the vectors, as it can be seen in the following pictures of the terminals of Alice and Bob, where the inputs of each terminal are the values of the vector A (in case of Alice) and vector B (in case of Bob):

```
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_bob -r progs/hw2-II-ScalarProduct.txt asdkfjs 4
Running Bob...
input.bob[9]0
input.bob[8]1
input.bob[7]0
input.bob[6]1
input.bob[5]0
input.bob[4]1
input.bob[3]0
input.bob[2]1
input.bob[1]0
input.bob[0]1
output.bob5
```

```
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_alice -r progs/hw2-II-ScalarProduct.txt asdkfjs 127.0.0.1
Running Alice...
input.alice[9]0
input.alice[8]1
input.alice[7]0
input.alice[6]1
input.alice[5]0
input.alice[4]1
input.alice[3]0
input.alice[2]1
input.alice[1]0
input.alice[0]1
output.alice5
```

2. **Write the SFDL program for Alice and Bob, compile it for Alice and Bob, and run the protocols (communication is integrated in Fairplay).**

   The following pictures shows the SFDL program, the picture of compilations and the running of the protocol.

   SFDL program for Alice and Bob:

```
/*
 * Eloy Ramirez Hernanz
 * CWID: A20433696
 *
 * Compute the scalar product of two binary arrays of size 10
 */
program ScalarProduct {

// Constants

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};

// Function Definitions

// This is the main function
function Output output(Input input) {

   var Int<8> i;
   var Int<8> result;

   result=0;

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

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

Compilation for Alice and Bob:

```
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_bob -c progs/hw2-II-ScalarProduct.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.
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_alice -c progs/hw2-II-ScalarProduct.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.
```

Running for Alice and Bob:

```
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_bob -r progs/hw2-II-ScalarProduct.txt asdkfjs 4
Running Bob...
input.bob[9]0
input.bob[8]1
input.bob[7]0
input.bob[6]1
input.bob[5]0
input.bob[4]1
input.bob[3]0
input.bob[2]1
input.bob[1]0
input.bob[0]1
output.bob5
```

```
eloy@DESKTOP-4FG54I1:/mnt/c/Users/elora/Downloads/Fairplay_Project/run$ ./run_alice -r progs/hw2-II-ScalarProduct.txt asdkfjs 127.0.0.1
Running Alice...
input.alice[9]0
input.alice[8]1
input.alice[7]0
input.alice[6]1
input.alice[5]0
input.alice[4]1
input.alice[3]0
input.alice[2]1
input.alice[1]0
input.alice[0]1
output.alice5
```

3. **Report the input Boolean vectors, the SFDL program, SHDL circuit and output result A·B.**

The input Boolean vectors I use in the example are the ones used in the pictures of the first question of part II and the last 2 pictures of the question 2, but they can be changed by the user for the ones they wanted. That's why they are random.
In the pictures I used the following vectors:
A = [1,0,1,0,1,0,1,0,1,0]
B = [1,0,1,0,1,0,1,0,1,0]

The SFDL program can be seen in the picture of question 2 of this part II of the assignment.

The SHDL circuit is attached in the submission of the assignment.

The output result with the vectors selected is 5, as it can be seen in the pictures attached both in the first question of this part II of the assignment and in the last 2 pictures of the question 2.