**Jigyas Sharma**

**Dr. Han Wang**

**EECS 700**

**15 November 2023**

**Homework 3: Report**

**Task 1:**

1. To produce a random nonnegative matrix for Alice and Bob for their respective sizes, we use the numpy library.

# Alice generates a 5x8 matrix

A = np.random.randint(low=0, high=100, size=(5, 8))

# Bob generates an 8x4 matrix

B = np.random.randint(low=0, high=100, size=(8, 4))

1. The design of the cryptographic protocol that can be used to securely compute the product of AxB is as follows:

**Step 1:** Alice produces a key pair using paillier crypto system.

**Step 2:** Alice then sends her public key to Bob.

**Step 3:** Bob multiplies his matrix with some random number(in our case 4) before encrypting his matrix. So, even if Alice can decrypt this matrix, the information is redacted.

**Step 4:** Alice receives the encrypted matrix. Since, there is paillier encryption used, Alice can use the homomorphic properties to multiply the encrypted matrix with her plain text value. Alice, then sends this multiplied matrix back to Bob.

**Step 5:** Bob receives the multiplied matrix which he then divides by his random value(4) to reveal the actual answer of matrix multiplication.

During these steps Bob never finds out the value of Alice’s matrix and Alice never finds out the value of Bob’s matrix.

1. *Code for Alice and Bob’s Computer is provided in the zip file as* ***Alice.py*** *and* ***Bob.py***
2. The screenshots below post the value. The first set of screenshots are with 512 key size and the second set is with 1024 key size. The print statement only reveals an object, however, when using 512 key size the computation takes less time and resources when compared to 1024 key size.

*A screenshot of a computer

Description automatically generated*

Figure 1: Alice's Computer 512 key size

A screenshot of a computer

Description automatically generated

Figure 2: Bob's Computer 512 key size

A screenshot of a computer

Description automatically generated

Figure 3: Alice's Computer 1024 key size

A screenshot of a computer

Description automatically generated

Figure 4: Bob's Computer 1024 key size

1. Scenario 1: When a plaintext value is multiplied by an encrypted value it gives out the correct matrix value. This can be done by calculating the matrix multiplication value from Alice and Bob’s original matrix. Since, in the above situation Bob’s matrix was encrypted however, Alice was able to multiply the values with her plaintext. We can also verify this by storing the normal matrix multiplication and comparing it to the decrypted matrix multiplication. (Refer to figure 5)

A screenshot of a computer program

Description automatically generated

Figure 5: Cross checking encrypted and normal matrix multiplication

Scenario 2: When trying to multiply two encrypted values, for example both the matrices are encrypted, the value will not hold. Consider, this simple program below which tries to multiply two encrypted values at which point the compiler throws an error. (Refer figure 6)

A screenshot of a computer code

Description automatically generated

Figure 6: Two encrypted values multiplication throws an error

**Task 2:**

**Input boolean vectors:**

**Alice :** 0 0 0 0 1 1 1 0 0 0

**Bob:** 1 1 1 1 0 0 0 0 1 0

**SFDL PROGRAM:**

/\*

\* Calculate the scalar product of two Boolean vectors

\*/

program ScalarProduct {

const N = 10;

type Bool = Int<1>; // Assuming Bool as 1-bit integer

type Vector = Bool[N];

type AliceInput = Vector;

type BobInput = Vector;

type ScalarProductResult = Int<32>; // 32-bit integer to hold the scalar product

type Input = struct {AliceInput alice, BobInput bob};

type Output = struct {ScalarProductResult result};

function Output output(Input input) {

int<32> sum = 0;

int<8> i;

for (i = 0 to N-1) {

sum = sum + (input.alice[i] \* input.bob[i]); // Self-referencing assignment

}

Output result;

result.result = sum;

return result;

}

}