Secure Multi-Party Computation

Assignment 3 - Passive BeDOZa Implementation

Or Dinar
Liad Ackerman

Late submission as approved by Dr. Akavia

04.02.2024

1 Passive BeDOZa - Theory

In this assignment we were given equation 3, and our task is to use the passive variant of BeDOZa protocol to securely compute the result.
We chose to implement the protocol using boolean circuits. Reminder:

Equation 3:

$$f_{\overrightarrow{a},4}(x_1, x_2) = \begin{cases} 1 & \text{if } a_1x_1 + a_2x_2 \geq 4 \\ 0 & \text{otherwise} \end{cases} \qquad for \quad \overrightarrow{a} \in \{0,1,2,3\}^2$$

1.1 What does it do?

The BeDOZA protocol uses the idea of *Secret Sharing Schemes* in order to securely compute an operation of a logical gate. In boolean circuits, It does this by picking a random bit r and computing $r \bigoplus x$ where x is an input bit.
In boolean circuits Secret Sharing Schemes have the following functions:

- Shr(x) - provided an x, pick a random bit (0 or 1) r and return (r, $r \bigoplus x$).
- Rec(S_1, S_2) - reconstructs the secret input buy applying $\bigoplus$ between the secret sharing themselves, then computing the XOR between the products.
- OpenTo(P,S_1) - returns $r \bigoplus r \bigoplus x$ to the party P, which, in turn, is the secret x.
- Open(S_1) - returns $r \bigoplus r \bigoplus x$ to all parties.

1.2 Implementation

In our implementation of the boolean circuit for Equation 3, we defined a function for each logical gate, as well as functions for the *Secret Sharing Scheme*'s operations. The functions in the python implementation are the following:

- shr(secret)
- recSelf(secret)
- rec(x, y)
- openTo(secret)
- XOR(x, y)
- XORconst(x, c)
- AND(x, y)
- ANDconst(x, c)
- OR(x, y)
- twoBitMulti(num1,num2)
- fourBitAdder(num1,num2)
- boolianCircuit(a1,a2,x1,x2)

1.3 Equation 3's Boolean Circuit

We'll provide an image of the boolean circuit for Equation 3 for reference.

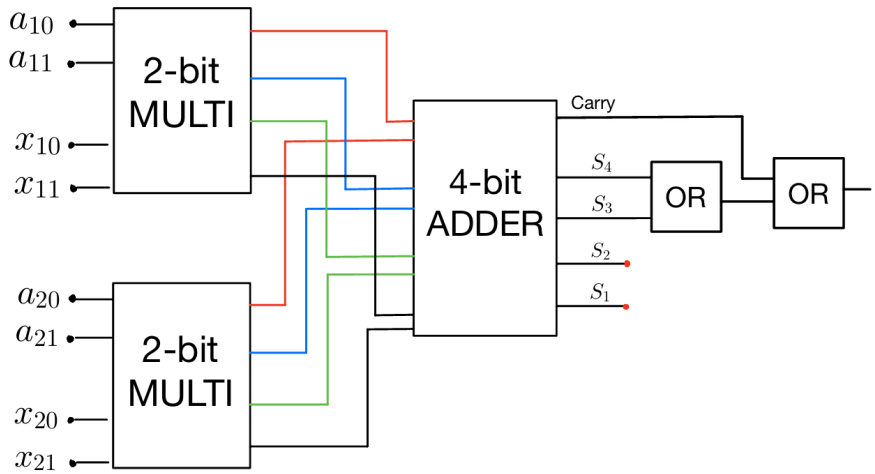

Figure 1: Boolean circuit that represents *Equation* 3.

We also recall that the implementation of 2-bit Multiplier and 4-bit Adder were as following:

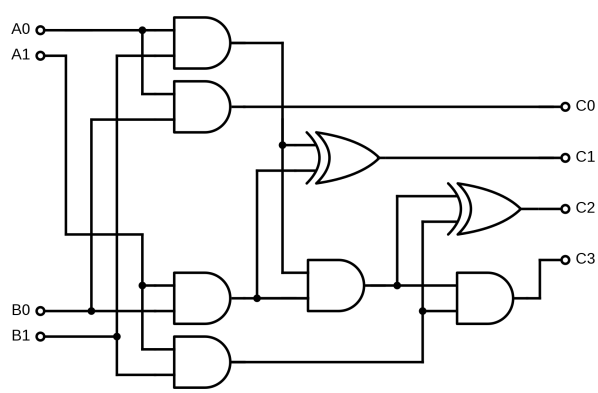

Figure 2: 2-bit Multiplier.

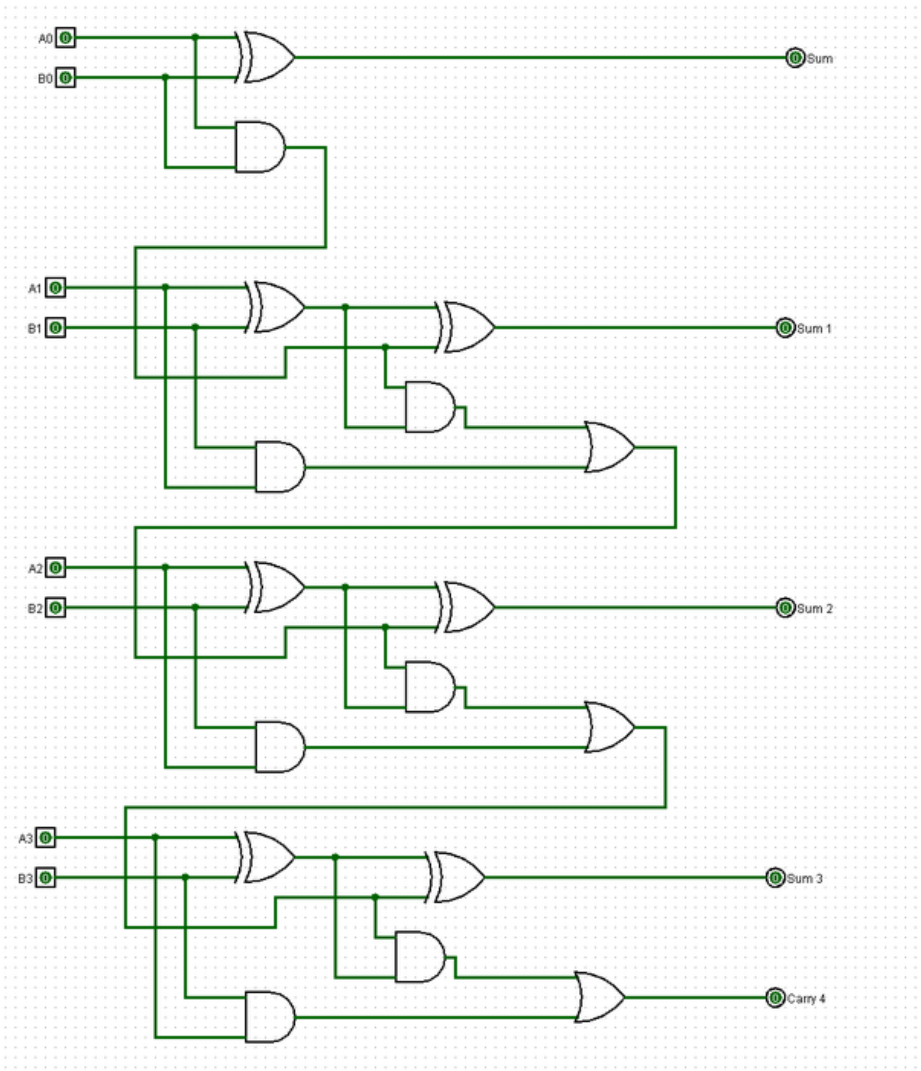

Figure 3: 4-bit Adder.

furthermore, since only AND and XOR gates are allowed, we also implemented the OR gate using AND and XOR gates, as shown in Figure 4 below:

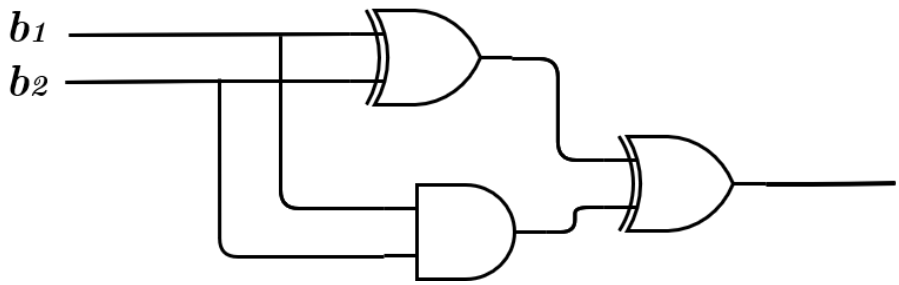

Figure 4: OR gate using XOR and AND gates

In the attached python file, we implemented each of the calculations above as functions, such that 2-bit Multiplier receives two 2-bit numbers - each of those bits is masked by *secret sharing*, and returns a secret shared 4-bit product $a_i x_i$. While 4-bit Adder receives two *secret shared* products from 2-bit Multiplier, and returns secret shared 5-bit sum $[Carry, bit4, bit3, bit2, bit1]$. We also recall that the sum is greater or equal to 4 (given threshold) if either one of $[Carry, bit4, bit3]$ is 1, hence the last two OR gates at the end.

All in all, we constructed a Boolean circuit for the function in Equation 3. It receives 4 Secret Shared bits for a and 4 Secret Shared bits for x and computes the equation.