ECE 443/518 Fall 2024 - Project 6

Garbled Circuits

Samman Chouhan

A20561414

III. Implement Bob the Evaluator

implement the `evaluateGarbledCircuit` function for Bob the evaluator.

• For each gate in the circuit:

- We get the two input signals `a` and `b` from the signals array using the gate's input wire IDs
- Extract selection bits from MSB (most significant bit) of each input signal
- Use these bits to calculate which row of the truth table to decrypt
- Create an AES cipher using the concatenated input signals as the 256-bit key
- Decrypt the selected row from the gate's truth table to get the output signal
- Store the output signal in the signals array at the gate's output wire position

• Key points about the implementation:

- The selection bits are extracted the same way Alice arranged them in `encryptGates`
- We use the same AES cipher setup as Alice for decryption
- The output signal is automatically the correct value (0 or 1) because Alice encrypted the truth table appropriately
- We print debug information to match Alice's format

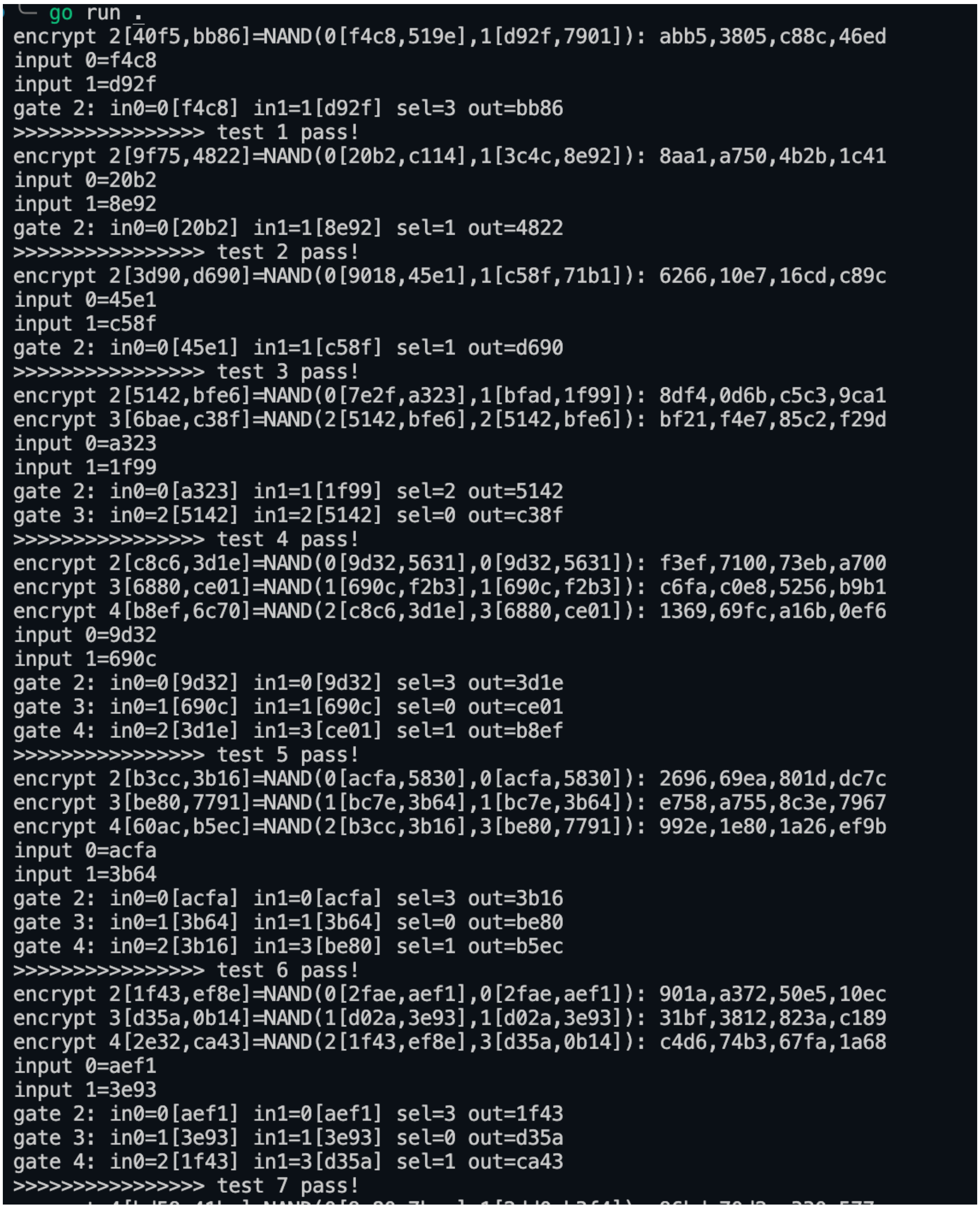
• Security properties:

- Bob never sees the actual 0/1 values for any wire
- Bob can only decrypt one row per gate using the signals they have
- The circuit structure is known but the computation remains private\

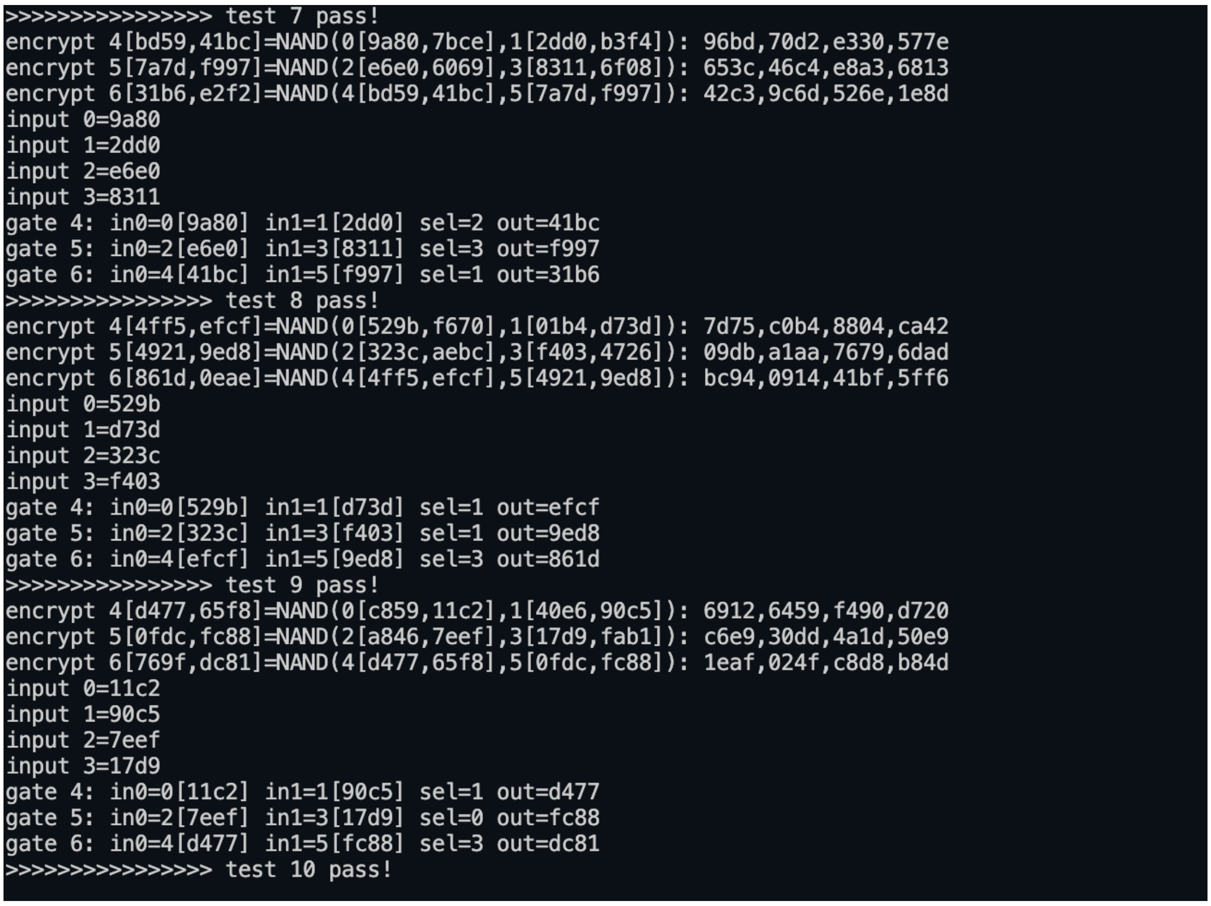
Code

```go
func evaluateGarbledCircuit(inputs [][]byte, gates []Gate) []byte {
    n := len(inputs) // number of inputs
    m := len(gates)  // number of gates

    // array of signals have a size of n+m
    signals := make([][]byte, m+n)

    // setup inputs signals
    for i := 0; i < n; i++ {
        signals[i] = inputs[i]
        fmt.Printf("input %d=%x\n", i, signals[i][:2])
    }

    // Evaluate each gate in order
    for _, gate := range gates {
        // Get input signals for this gate
        a := signals[gate.in0]
        b := signals[gate.in1]

        // Extract selection bits from input signals
        // The first bit (MSB) of each signal determines which row to select
        sa := (a[0] & 0x80) >> 7
        sb := (b[0] & 0x80) >> 7

        // Calculate table index using selection bits
        tableIndex := sa*2 + sb

        // Create AES cipher using concatenated input signals as key
        cipher, _ := aes.NewCipher(append(a, b...))

        // Decrypt the selected row from the truth table
        output := make([]byte, 16)
        cipher.Decrypt(output, gate.table[tableIndex])

        // Store the decrypted output signal
        signals[gate.out] = output

        fmt.Printf("gate %d: in0=%d[%x] in1=%d[%x] sel=%d out=%x\n",
            gate.out, gate.in0, a[:2], gate.in1, b[:2], tableIndex, output[:2])
    }
    // the last signal is the output

    return signals[n+m-1]
}
```

Test Cases Screenshot



```
╰─ go run .
encrypt 2[40f5,bb86]=NAND(0[f4c8,519e],1[d92f,7901]): abb5,3805,c88c,46ed
input 0=f4c8
input 1=d92f
gate 2: in0=0[f4c8] in1=1[d92f] sel=3 out=bb86
>>>>>>>>>>>>>>>>> test 1 pass!
encrypt 2[9f75,4822]=NAND(0[20b2,c114],1[3c4c,8e92]): 8aa1,a750,4b2b,1c41
input 0=20b2
input 1=8e92
gate 2: in0=0[20b2] in1=1[8e92] sel=1 out=4822
>>>>>>>>>>>>>>>>> test 2 pass!
encrypt 2[3d90,d690]=NAND(0[9018,45e1],1[c58f,71b1]): 6266,10e7,16cd,c89c
input 0=45e1
input 1=c58f
gate 2: in0=0[45e1] in1=1[c58f] sel=1 out=d690
>>>>>>>>>>>>>>>>> test 3 pass!
encrypt 2[5142,bfe6]=NAND(0[7e2f,a323],1[bfad,1f99]): 8df4,0d6b,c5c3,9ca1
encrypt 3[6bae,c38f]=NAND(2[5142,bfe6],2[5142,bfe6]): bf21,f4e7,85c2,f29d
input 0=a323
input 1=1f99
gate 2: in0=0[a323] in1=1[1f99] sel=2 out=5142
gate 3: in0=2[5142] in1=2[5142] sel=0 out=c38f
>>>>>>>>>>>>>>>>> test 4 pass!
encrypt 2[c8c6,3d1e]=NAND(0[9d32,5631],0[9d32,5631]): f3ef,7100,73eb,a700
encrypt 3[6880,ce01]=NAND(1[690c,f2b3],1[690c,f2b3]): c6fa,c0e8,5256,b9b1
encrypt 4[b8ef,6c70]=NAND(2[c8c6,3d1e],3[6880,ce01]): 1369,69fc,a16b,0ef6
input 0=9d32
input 1=690c
gate 2: in0=0[9d32] in1=0[9d32] sel=3 out=3d1e
gate 3: in0=1[690c] in1=1[690c] sel=0 out=ce01
gate 4: in0=2[3d1e] in1=3[ce01] sel=1 out=b8ef
>>>>>>>>>>>>>>>>> test 5 pass!
encrypt 2[b3cc,3b16]=NAND(0[acfa,5830],0[acfa,5830]): 2696,69ea,801d,dc7c
encrypt 3[be80,7791]=NAND(1[bc7e,3b64],1[bc7e,3b64]): e758,a755,8c3e,7967
encrypt 4[60ac,b5ec]=NAND(2[b3cc,3b16],3[be80,7791]): 992e,1e80,1a26,ef9b
input 0=acfa
input 1=3b64
gate 2: in0=0[acfa] in1=0[acfa] sel=3 out=3b16
gate 3: in0=1[3b64] in1=1[3b64] sel=0 out=be80
gate 4: in0=2[3b16] in1=3[be80] sel=1 out=b5ec
>>>>>>>>>>>>>>>>> test 6 pass!
encrypt 2[1f43,ef8e]=NAND(0[2fae,aef1],0[2fae,aef1]): 901a,a372,50e5,10ec
encrypt 3[d35a,0b14]=NAND(1[d02a,3e93],1[d02a,3e93]): 31bf,3812,823a,c189
encrypt 4[2e32,ca43]=NAND(2[1f43,ef8e],3[d35a,0b14]): c4d6,74b3,67fa,1a68
input 0=aef1
input 1=3e93
gate 2: in0=0[aef1] in1=0[aef1] sel=3 out=1f43
gate 3: in0=1[3e93] in1=1[3e93] sel=0 out=d35a
gate 4: in0=2[1f43] in1=3[d35a] sel=1 out=ca43
>>>>>>>>>>>>>>>>> test 7 pass!
[illegible]
```



```
>>>>>>>>>>>>>>>> test 7 pass!
encrypt 4[bd59,41bc]=NAND(0[9a80,7bce],1[2dd0,b3f4]): 96bd,70d2,e330,577e
encrypt 5[7a7d,f997]=NAND(2[e6e0,6069],3[8311,6f08]): 653c,46c4,e8a3,6813
encrypt 6[31b6,e2f2]=NAND(4[bd59,41bc],5[7a7d,f997]): 42c3,9c6d,526e,1e8d
input 0=9a80
input 1=2dd0
input 2=e6e0
input 3=8311
gate 4: in0=0[9a80] in1=1[2dd0] sel=2 out=41bc
gate 5: in0=2[e6e0] in1=3[8311] sel=3 out=f997
gate 6: in0=4[41bc] in1=5[f997] sel=1 out=31b6
>>>>>>>>>>>>>>>> test 8 pass!
encrypt 4[4ff5,efcf]=NAND(0[529b,f670],1[01b4,d73d]): 7d75,c0b4,8804,ca42
encrypt 5[4921,9ed8]=NAND(2[323c,aebc],3[f403,4726]): 09db,a1aa,7679,6dad
encrypt 6[861d,0eae]=NAND(4[4ff5,efcf],5[4921,9ed8]): bc94,0914,41bf,5ff6
input 0=529b
input 1=d73d
input 2=323c
input 3=f403
gate 4: in0=0[529b] in1=1[d73d] sel=1 out=efcf
gate 5: in0=2[323c] in1=3[f403] sel=1 out=9ed8
gate 6: in0=4[efcf] in1=5[9ed8] sel=3 out=861d
>>>>>>>>>>>>>>>> test 9 pass!
encrypt 4[d477,65f8]=NAND(0[c859,11c2],1[40e6,90c5]): 6912,6459,f490,d720
encrypt 5[0fdc,fc88]=NAND(2[a846,7eef],3[17d9,fab1]): c6e9,30dd,4a1d,50e9
encrypt 6[769f,dc81]=NAND(4[d477,65f8],5[0fdc,fc88]): 1eaf,024f,c8d8,b84d
input 0=11c2
input 1=90c5
input 2=7eef
input 3=17d9
gate 4: in0=0[11c2] in1=1[90c5] sel=1 out=d477
gate 5: in0=2[7eef] in1=3[17d9] sel=0 out=fc88
gate 6: in0=4[d477] in1=5[fc88] sel=3 out=dc81
>>>>>>>>>>>>>>>> test 10 pass!
```

IV. Support Additional Gate Types

code to add support for NOR and XOR gates, as they're useful for different types of computations. NOR, like NAND, is functionally complete, while XOR is particularly useful for tasks like comparison and arithmetic.

• Modified `encrypt Gates` in Alice. Go:

- Added support for NOR and XOR gates
- Each gate type has its own truth table encryption pattern:
    - NAND: 0,0→1; 0,1→1; 1,0→1; 1,1→0
    - NOR: 0,0→1; 0,1→0; 1,0→0; 1,1→0
    - XOR: 0,0→0; 0,1→1; 1,0→1; 1,1→0
- Updated error handling to include new gate types
- Modified debug output to show gate type

• Added new test functions to gc.go:

- `testNOR`: Tests all possible input combinations for NOR gate
- `testXOR`: Tests all possible input combinations for XOR gate
- `testMixed`: Tests a circuit using all three gate types
    - Implements (a XOR b) NOR (b NAND c)
    - Tests different input combinations to verify correct operation

• The test cases verify:

- Individual gate functionality
- Complex combinations of gates
- Edge cases and different input patterns

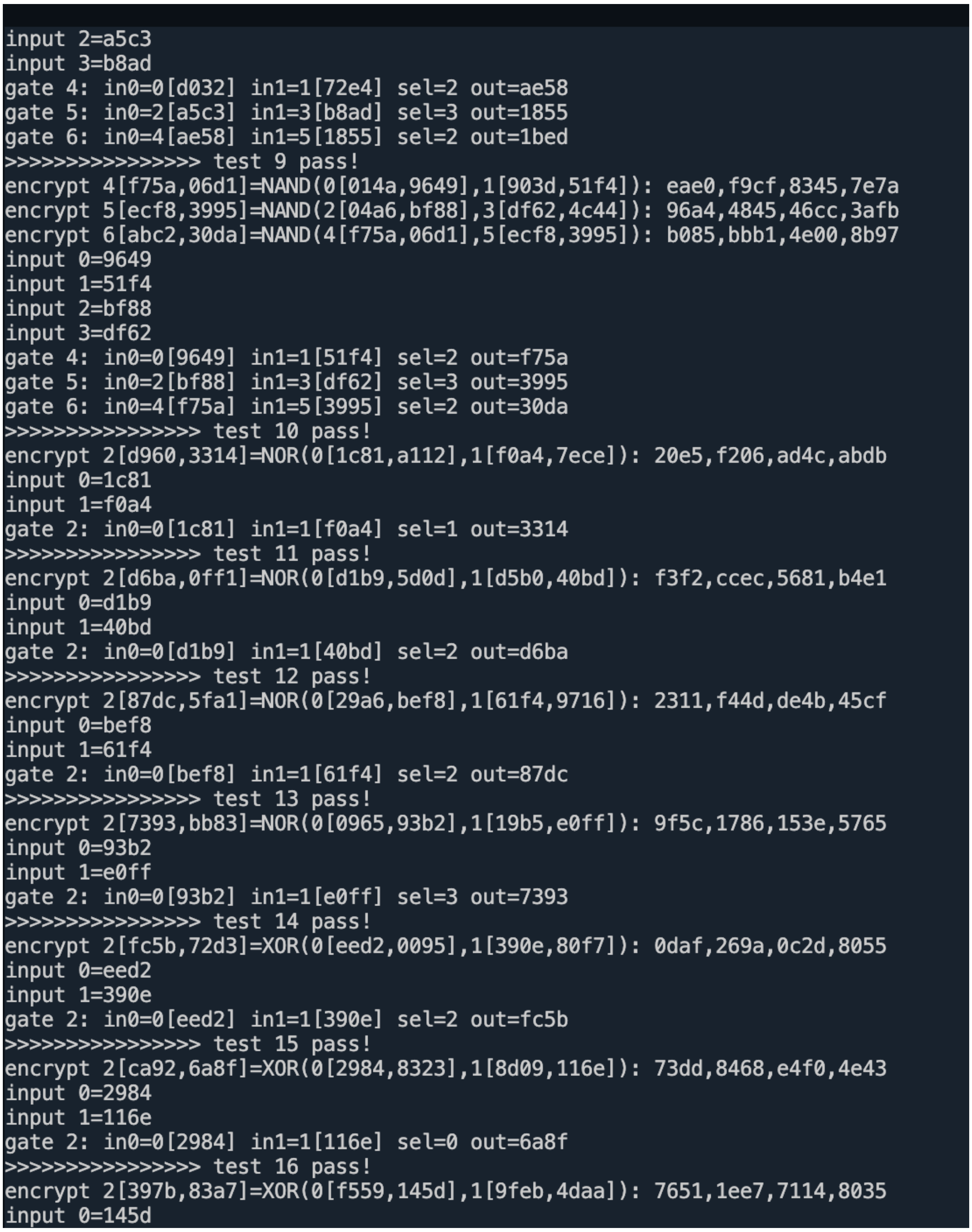
• Security considerations:

- The encryption mechanism remains secure for all gate types
- Each gate type maintains the property that only one row can be decrypted
- The circuit structure reveals the operations but not the intermediate values

Code

```go
       func encryptGates(gates []Gate, wires []Wire) {

    for i := range gates {
        // use a pointer so we can modify gate in the array
        gate := &gates[i]
        if gate.logic != "NAND" && gate.logic != "NOR" && gate.logic != "XOR" {
            panic("only NAND, NOR, and XOR gates are supported")
        }

        // input and output wires
        A := wires[gate.in0]
        B := wires[gate.in1]
        O := wires[gate.out]

        // selection bits are used to arrange the rows
        // in the garbled truth table
        sa := (A.v[0][0] & 0x80) >> 7
        sb := (B.v[0][0] & 0x80) >> 7

        // generate rows in the garbled truth table based on gate type
        switch gate.logic {
        case "NAND":
            gate.table[sa*2+sb] = encryptOneRow(A.v[0], B.v[0], O.v[1])
            gate.table[sa*2+1-sb] = encryptOneRow(A.v[0], B.v[1], O.v[1])
            gate.table[(1-sa)*2+sb] = encryptOneRow(A.v[1], B.v[0], O.v[1])
            gate.table[(1-sa)*2+1-sb] = encryptOneRow(A.v[1], B.v[1], O.v[0])
        case "NOR":
            gate.table[sa*2+sb] = encryptOneRow(A.v[0], B.v[0], O.v[1])
            gate.table[sa*2+1-sb] = encryptOneRow(A.v[0], B.v[1], O.v[0])
            gate.table[(1-sa)*2+sb] = encryptOneRow(A.v[1], B.v[0], O.v[0])
            gate.table[(1-sa)*2+1-sb] = encryptOneRow(A.v[1], B.v[1], O.v[0])
        case "XOR":
            gate.table[sa*2+sb] = encryptOneRow(A.v[0], B.v[0], O.v[0])
            gate.table[sa*2+1-sb] = encryptOneRow(A.v[0], B.v[1], O.v[1])
            gate.table[(1-sa)*2+sb] = encryptOneRow(A.v[1], B.v[0], O.v[1])
            gate.table[(1-sa)*2+1-sb] = encryptOneRow(A.v[1], B.v[1], O.v[0])
        }

        fmt.Printf("encrypt %d[%x,%x]=%s(%d[%x,%x],%d[%x,%x]): %x,%x,%x,%x\n",
            gate.out, O.v[0][:2], O.v[1][:2],
            gate.logic,
            gate.in0, A.v[0][:2], A.v[1][:2],
            gate.in1, B.v[0][:2], B.v[1][:2],
            gate.table[0][:2], gate.table[1][:2],
            gate.table[2][:2], gate.table[3][:2])
    }
```

Screenshot (Test Cases)



```
input 2=a5c3
input 3=b8ad
gate 4: in0=0[d032] in1=1[72e4] sel=2 out=ae58
gate 5: in0=2[a5c3] in1=3[b8ad] sel=3 out=1855
gate 6: in0=4[ae58] in1=5[1855] sel=2 out=1bed
>>>>>>>>>>>>>>>>> test 9 pass!
encrypt 4[f75a,06d1]=NAND(0[014a,9649],1[903d,51f4]): eae0,f9cf,8345,7e7a
encrypt 5[ecf8,3995]=NAND(2[04a6,bf88],3[df62,4c44]): 96a4,4845,46cc,3afb
encrypt 6[abc2,30da]=NAND(4[f75a,06d1],5[ecf8,3995]): b085,bbb1,4e00,8b97
input 0=9649
input 1=51f4
input 2=bf88
input 3=df62
gate 4: in0=0[9649] in1=1[51f4] sel=2 out=f75a
gate 5: in0=2[bf88] in1=3[df62] sel=3 out=3995
gate 6: in0=4[f75a] in1=5[3995] sel=2 out=30da
>>>>>>>>>>>>>>>>> test 10 pass!
encrypt 2[d960,3314]=NOR(0[1c81,a112],1[f0a4,7ece]): 20e5,f206,ad4c,abdb
input 0=1c81
input 1=f0a4
gate 2: in0=0[1c81] in1=1[f0a4] sel=1 out=3314
>>>>>>>>>>>>>>>>> test 11 pass!
encrypt 2[d6ba,0ff1]=NOR(0[d1b9,5d0d],1[d5b0,40bd]): f3f2,ccec,5681,b4e1
input 0=d1b9
input 1=40bd
gate 2: in0=0[d1b9] in1=1[40bd] sel=2 out=d6ba
>>>>>>>>>>>>>>>>> test 12 pass!
encrypt 2[87dc,5fa1]=NOR(0[29a6,bef8],1[61f4,9716]): 2311,f44d,de4b,45cf
input 0=bef8
input 1=61f4
gate 2: in0=0[bef8] in1=1[61f4] sel=2 out=87dc
>>>>>>>>>>>>>>>>> test 13 pass!
encrypt 2[7393,bb83]=NOR(0[0965,93b2],1[19b5,e0ff]): 9f5c,1786,153e,5765
input 0=93b2
input 1=e0ff
gate 2: in0=0[93b2] in1=1[e0ff] sel=3 out=7393
>>>>>>>>>>>>>>>>> test 14 pass!
encrypt 2[fc5b,72d3]=XOR(0[eed2,0095],1[390e,80f7]): 0daf,269a,0c2d,8055
input 0=eed2
input 1=390e
gate 2: in0=0[eed2] in1=1[390e] sel=2 out=fc5b
>>>>>>>>>>>>>>>>> test 15 pass!
encrypt 2[ca92,6a8f]=XOR(0[2984,8323],1[8d09,116e]): 73dd,8468,e4f0,4e43
input 0=2984
input 1=116e
gate 2: in0=0[2984] in1=1[116e] sel=0 out=6a8f
>>>>>>>>>>>>>>>>> test 16 pass!
encrypt 2[397b,83a7]=XOR(0[f559,145d],1[9feb,4daa]): 7651,1ee7,7114,8035
input 0=145d
```