ECE 518 Project 4

Alan Palayil Due Date: 12/04/2022

Section I Overview

In this project, we work on the Garbled Circuit which was discussed in Lecture 18. We implement the circuit that enables two parties to compute a function. There were modifications done for the simplicity of the garbled circuit within Golang. We implemented garbled circuit mechanism including the garbler, input/output, and few tests to evaluate the evaluator.

Section II Alice the Garbler

In the implementation of the garbler is in alice.go file in prj04-go. In alice.go, the wires are labelled from 0 to n+m-1 with each wire object containing two random 128-bit values, one for 0 or 1. Alice generates these values in a function and is not revealed to Bob. Only NAND logical gate function for Alice to generate the garbled truth table fir each Gate object in the function encryptGates. In troubleshooting, only the first two bytes were printed in the log messages for the 128-bit signals.

The file is not modified as the tests are focused on the evaluator in bob.go file.

Section III Implement Bob the Evaluator

In the implementation for the evaluator within the garbled circuit we are to work on the function evaluateGarbledCircuit in bob.go file. We heavily refer to Lecture 18 to work over the function. We setup the gates array in a way such that it is guaranteed that when you process a gate following its order in the gates array, both of its input signals are available already. Using the hint within the for loop:

```
a := signals[gate.in0]

b := signals[gate.in1]
```

We need to implement S_a and S_b using the logic S_a = a[0]/128 and S_b = b[0]/128 and find the location index (S_aS_b) of the decrypted key 'g' which basically equates to:

```
sasb := sa*2 + sb
```

We add a new AES cipher for 'g' similar to the cipher in alice.go and decrypt the table using the location index and assign the signal to the decrypted the cipher.

Section IV Testing the Implementation

We test our code by running the go package using "go run" There are 10 test testcases in gc.go which are executed from the main function. I have added the Debug Console to display the passing of each testcase during the execution.

```
Starting: C:\Users\alanp\go\bin\dlv.exe dap --check-go-version=false --
listen=127.0.0.1:53141 from a:\Downloads\prj04-go

DAP server listening at: 127.0.0.1:53141

Type 'dlv help' for list of commands.

encrypt 2[623e,a3b8]=NAND(0[e9bb,1713],1[585a,e440]): 2c30,d10c,7005,d0b7

input 0=e9bb

input 1=585a

>>>>>>>>>>>>>>>> test 1 pass!

encrypt 2[4c64,d47d]=NAND(0[c2c0,2e83],1[bab6,1301]): f316,12ba,244d,dfa8

input 0=c2c0

input 1=1301

>>>>>>>>>>>>>>>> test 2 pass!

encrypt 2[4fd9,f988]=NAND(0[2ca9,9370],1[4ca9,c877]): d853,d5b6,d601,e9de

input 0=9370

input 1=4ca9

>>>>>>>>>>>>>>>> test 3 pass!

encrypt 2[2636,dc09]=NAND(0[c1cc,280b],1[78c2,8745]): 2e9c,0ac2,e50c,a035

encrypt 3[110e,89cb]=NAND(2[2636,dc09],2[2636,dc09]): e678,19d9,da36,2d14

input 0=280b
```

```
input 1=8745

>>>>>>>>>>>>>>>> test 4 pass!

encrypt 2[ac31,0423]=NAND(0[12ca,a16d],0[12ca,a16d]): 85ed,6a76,0945,6d0b

encrypt 3[a55b,60d4]=NAND(1[b127,52ed],1[b127,52ed]): fa65,ba3c,ebea,8c2e

encrypt 4[70dd,cbf1]=NAND(2[ac31,0423],3[a55b,60d4]): 1b57,d002,1f4e,0abb

input 0=12ca

input 1=b127

>>>>>>>>>>>>>>>> test 5 pass!

encrypt 2[a9e6,2548]=NAND(0[640d,ddcc],0[640d,ddcc]): c8fa,d817,dc96,cd66

encrypt 3[cd9e,3c19]=NAND(1[125c,ea3f],1[125c,ea3f]): 3e64,98dc,c722,c07b

encrypt 4[4655,aba3]=NAND(2[a9e6,2548],3[cd9e,3c19]): c1ec,10fc,43da,9115

input 0=640d

input 1=ea3f

>>>>>>>>>>>>>>>> test 6 pass!

encrypt 2[ba5f,70d3]=NAND(0[6ba3,863a],0[6ba3,863a]): 45bb,636c,3371,3528

encrypt 3[5437,b678]=NAND(1[8db4,520a],1[8db4,520a]): d510,9386,20b3,9837

encrypt 4[0222,a145]=NAND(2[ba5f,70d3],3[5437,b678]): 42c5,3035,6b44,76e8

input 0=863a

input 1=520a

>>>>>>>>>>>>>>>> test 7 pass!

encrypt 4[ad89,4097]=NAND(0[d49b,23f2],1[0bf0,b748]): 4c51,e4b4,615d,ef10

encrypt 5[a644,2894]=NAND(2[6ced,a5ed],3[1bce,c2f1]): 7b7e,edd1,9411,b256

encrypt 6[a9a2,4e24]=NAND(4[ad89,4097],5[a644,2894]): d619,8838,dc18,40cf

input 0=d49b

input 1=0bf0
```

```
input 2=6ced

input 3=1bce

>>>>>>>>>>>>>>>> test 8 pass!

encrypt 4[d2fd,2708]=NAND(0[0b7d,e27e],1[ee07,14b3]): 7afa,6d8c,dd9c,c6d1

encrypt 5[27ef,82c7]=NAND(2[4542,ae06],3[183f,d234]): 2dc1,0e47,2b78,d717

encrypt 6[1fa4,fc55]=NAND(4[d2fd,2708],5[27ef,82c7]): 79e0,71bf,3273,b9e1

input 0=0b7d

input 1=14b3

input 2=4542

input 3=183f

>>>>>>>>>>>>>>>> test 9 pass!

encrypt 4[e524,34c5]=NAND(0[c295,36d2],1[f020,6920]): 3cec,594c,6df1,3542

encrypt 5[202f,a867]=NAND(2[3a88,f219],3[6d7a,ac84]): 0a5a,8e8a,b548,ba4a

encrypt 6[e38b,4152]=NAND(4[e524,34c5],5[202f,a867]): 978e,7b4b,d2f4,b03a

input 0=36d2

input 1=6920

input 2=f219

input 3=6d7a

>>>>>>>>>>>>>>>> test 10 pass!

Process 13376 has exited with status 0

Detaching

dlv dap (18012) exited with code: 0
```

Section V Appendix

alice.go

```go
package main

// Alice the garbler

import (
    "crypto/aes"
    "crypto/rand"
    "fmt"
)

type Wire struct {
    // For each wire, we need to prepare two random
    // 128-bit values, v[0] for 0 and v[1] for 1.
    v [2][]byte
}

type Gate struct {
    // NAND, NOR, etc.
    logic string

    // id of input and output wires
    in0, in1, out int

    // garbled truth table
    table [4][]byte
```

```go
}


// helper function to make gate creation easy
func makeGate(logic string, in0, in1, out int) Gate {
    return Gate{logic: logic, in0: in0, in1: in1, out: out}
}


func encryptWires(n, m int) []Wire {
    // We need n+m wires for n input bits and m gates.
    wires := make([]Wire, m+n)

    for i := range wires {
        // use a pointer so we can modify wire in the array
        wire := &wires[i]

        // For each wire, we need to prepare two random
        // 128-bit values, v[0] for 0 and v[1] for 1.
        wire.v[0] = make([]byte, 16)
        wire.v[1] = make([]byte, 16)
        rand.Read(wire.v[0])
        rand.Read(wire.v[1])

        // The first bit is the selection bit. It need
        // to be different for v[0] and v[1] - correct
        // v[1] by inverting its first bit if not.
```

```go
        if wire.v[0][0]&0x80 == wire.v[1][0]&0x80 {
            wire.v[1][0] = wire.v[1][0] ^ 0x80
        }
    }

    return wires
}

// encrypt a row in the truth table
func encryptOneRow(a, b, o []byte) []byte {
    if len(a) != 16 || len(b) != 16 || len(o) != 16 {
        panic("only 128-bit wires are supported")
    }

    // setup AES block ciphers with 256-bit keys
    c, _ := aes.NewCipher(append(a, b...))

    // encrypt the output
    garbled := make([]byte, 16)
    c.Encrypt(garbled, o)

    return garbled
}

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" {
        panic("only NAND 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
    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])

    fmt.Printf("encrypt %d[%x,%x]=NAND(%d[%x,%x],%d[%x,%x]): %x,%x,%x,%x\n",
        gate.out, O.v[0][:2], O.v[1][:2],
```

```go
            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])

    }

}


func garbleCircuit(gates []Gate) []Wire {

    m := len(gates)

    n := gates[len(gates)-1].out + 1 - m

    wires := encryptWires(n, m)

    encryptGates(gates, wires)

    return wires

}
```

bob.go

```go
package main


import (

    "crypto/aes"

    "fmt"

)


// Bob the evaluator


func evaluateGarbledCircuit(inputs [][]byte, gates []Gate) []byte {
```

```go
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])

}


// add code below to evaluate the gates

for _, gate := range gates {

    a := signals[gate.in0]

    b := signals[gate.in1]

    sa := (a[0] & 0x80) >> 7

    sb := (b[0] & 0x80) >> 7

    sasb := sa*2 + sb

    g, _ := aes.NewCipher(append(a, b...))

    ug := make([]byte, 16)

    g.Decrypt(ug, gate.table[sasb])

    signals[gate.out] = ug

}
```

```
    // the last signal is the output
    return signals[n+m-1]
}
```

gc.go

```
package main


import (
    "bytes"
    "fmt"
)


func encryptInputs(inputs []int, wires []Wire) [][]byte {
    signals := make([][]byte, len(inputs))
    for i, input := range inputs {
        signals[i] = wires[i].v[input]
    }
    return signals
}


func decryptOutput(signal []byte, last Wire) int {
    if bytes.Compare(signal, last.v[0]) == 0 {
        return 0
    } else if bytes.Compare(signal, last.v[1]) == 0 {
        return 1
    }
```

```
    panic("invalid output signal")
}

func doTest(t string, gates []Gate, inputs []int, expected int) {
    wires := garbleCircuit(gates)
    signals := encryptInputs(inputs, wires)
    signal := evaluateGarbledCircuit(signals, gates)
    output := decryptOutput(signal, wires[len(wires)-1])
    if output != expected {
        panic("incorrect output for test " + t)
    } else {
        fmt.Printf(">>>>>>>>>>>>>>>> test %s pass!\n", t)
    }
}

func test123() {
    gates := []Gate{
        makeGate("NAND", 0, 1, 2),
    }
    doTest("1", gates, []int{0, 0}, 1)
    doTest("2", gates, []int{0, 1}, 1)
    doTest("3", gates, []int{1, 0}, 1)
}

func test4() {
```

```go
    gates := []Gate{
        makeGate("NAND", 0, 1, 2),
        makeGate("NAND", 2, 2, 3),
    }
    doTest("4", gates, []int{1, 1}, 1)
}

func test567() {
    gates := []Gate{
        makeGate("NAND", 0, 0, 2),
        makeGate("NAND", 1, 1, 3),
        makeGate("NAND", 2, 3, 4),
    }
    doTest("5", gates, []int{0, 0}, 0)
    doTest("6", gates, []int{0, 1}, 1)
    doTest("7", gates, []int{1, 1}, 1)
}

func test890() {
    gates := []Gate{
        makeGate("NAND", 0, 1, 4),
        makeGate("NAND", 2, 3, 5),
        makeGate("NAND", 4, 5, 6),
    }
    doTest("8", gates, []int{0, 0, 0, 0}, 0)
```

```go
    doTest("9", gates, []int{0, 1, 0, 0}, 0)

    doTest("10", gates, []int{1, 1, 1, 0}, 1)

}



func main() {

    test123()

    test4()

    test567()

    test890()

}
```

go.mod

```
module prj04



go 1.19
```