Lecture 24 - Zero Knowledge Identification System

Reading

Paper: https://arrow.dit.ie/cgi/viewcontent.cgi?article=1031&context=itbj

Look at page 38, section 7.9. to 7.12 on page 43 (this is the page numbers in the PDF - it is the 28th page OF the PDF).

Walk through of algorithm with the example from the paper.

Also see: https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/

Also: the reading from last time - has a nice section on this method for identification of users.

First a tiny detour - how to authenticate a QR or RFID tag.

QR codes encode some sort of text. Some RFID tags encode just a number. Others like NFC encode a chunk of text between 84 and 1084 characters long. Usually these chunks of text are URLs to some website or set of information. People would like to use the tags as proof-of-authenticity. The problem is how to get them to be secure. NFC tags can have computation built int - but it is really a small amount - the power for the NFC is coming from radio waves being transmitted from the source. So you take you Android and its tiny transmitter and send the NFC tag a tiny bit of power - that then looses lots of power because it has to be picked up by a tiny antenna in the "chip" and then used to do a small, small amount of computation and then using that same power send back an answer. So building any kind of "authentication" that is meaningful into the chip is difficult. A QR code is a static image - so it will not do any computation at all.

So if you can't do the authentication on the chip or device. What about just adding Two Factor Authentication after the device is scanned. Basically have the QR or RFID send you to a page where you have to authenticate using a device (iPhone, Android) and provide proof of your authenticity at that point.

Zero knowledge based chains - ZCache

Monero claims to hide who spent the money by "mixing" it into a set of 6 other transfers.

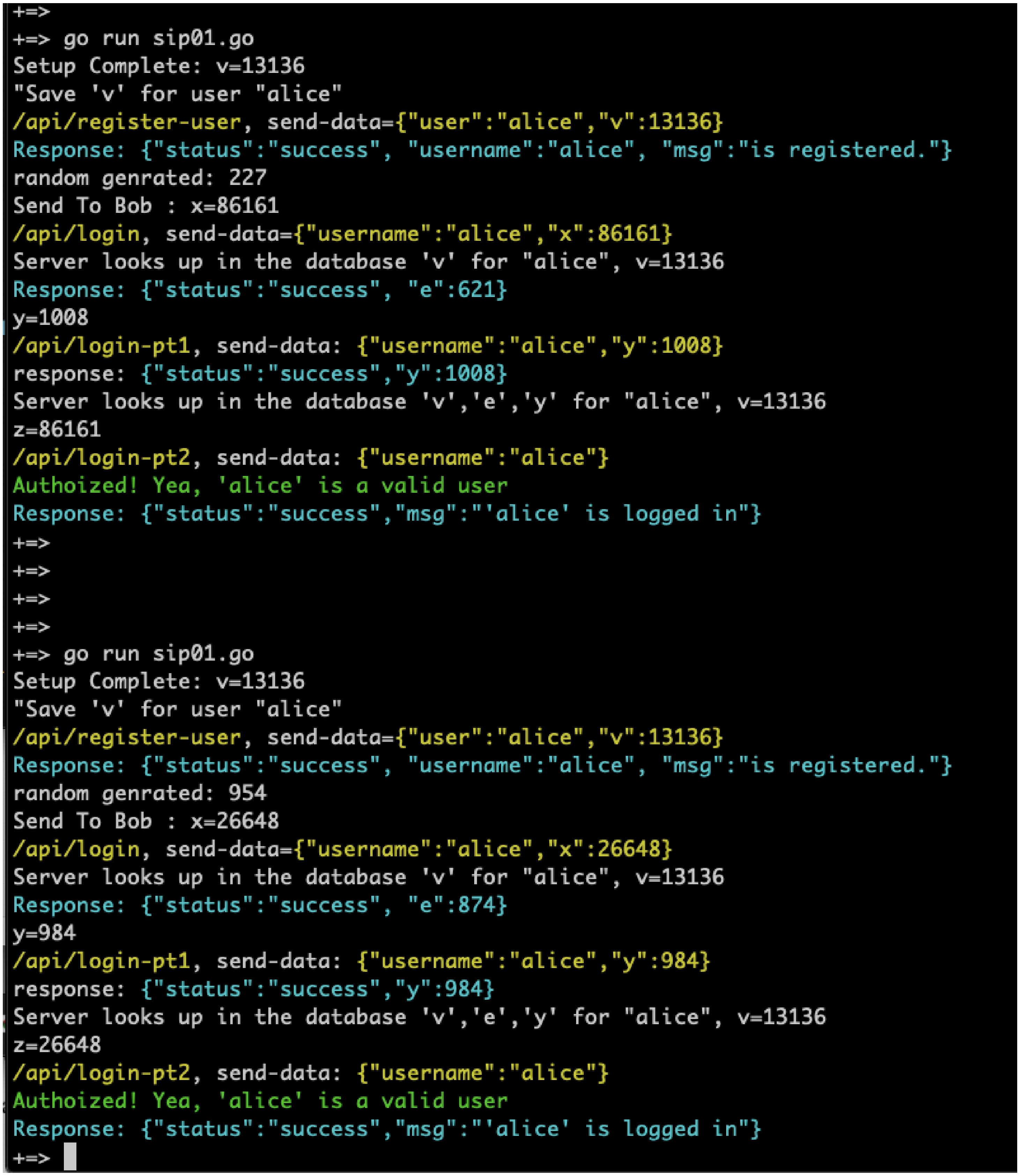
ZCache uses cryptography. Specifically RFC-8235 "Schnorr Non-interactive Zero-Knowledge Proof"

Zero knowledge proof for use as ID

```
package main

import (
	"fmt"
	"math/big"
	"math/rand"
	"time"

	"github.com/pschlump/MiscLib"
)

// From: https://arrow.dit.ie/cgi/viewcontent.cgi?article=1031&context=itbj
// IdProtocalsInCrypto.pdf
```

```go

type DBRecord struct {
    v *big.Int
    e *big.Int
    y *big.Int
    x *big.Int
}

var database map[string]*DBRecord

func init() {
    database = make(map[string]*DBRecord)
}

func main() {

    rand.Seed(time.Now().UnixNano())

    // ----------------------------------------------------------
    // Registration and Setup
    // ----------------------------------------------------------

    // From p40
    p := big.NewInt(88667) // password or hash of password to convert pw to number

    q := big.NewInt(1031)      // value 1 = 'q', Pre Chosen : large prime
    alpha := big.NewInt(70322) // value 2 == 'alpha', devisor of (p-1)
    a := big.NewInt(755)       // value 3 == 'a', alpha = (beta**((p-1)/q))mod p
    {

        // v = (alpha ^ ( q-a )) % p
        t1 := big.NewInt(0)
        t1.Sub(q, a)
        v := big.NewInt(0)
        v.Exp(alpha, t1, p) // note the 'p' is the "mod"

        fmt.Printf("Setup Complete: v=%s\n", v)
        fmt.Printf(`"Save 'v' for user "alice"` + "\n")

        fmt.Printf(`%s/api/register-user%s, send-data=%s{"user":"alice","v":%d}%s`+"\n",
            MiscLib.ColorYellow, MiscLib.ColorReset, MiscLib.ColorYellow, v, MiscLib.ColorReset)
        // Save the validation value 'v' for "alice" in the database.
        database["alice"] = &DBRecord{v: v}
        fmt.Printf(`%sResponse: {"status":"success", "username":"alice", "msg":"is registered."}%s`+"\n",
            MiscLib.ColorCyan, MiscLib.ColorReset)
    }

    // Alice is the Client:

    // ----------------------------------------------------------
    // Message 1 - Client to Server
    // ----------------------------------------------------------

    // Alice Chooses, and send to Bob
    // r := big.NewInt(543) // Should be random, but for this example
    randNum := genRan(999)
    fmt.Printf("random genrated: %d\n", randNum)
    r := big.NewInt(randNum)
    x := big.NewInt(0)
    x.Exp(alpha, r, p) // x=(alpha^r) % p

    fmt.Printf("Send To Bob : x=%s\n", x)
    fmt.Printf(`%s/api/login%s, send-data=%s{"username":"alice","x":%d}%s`+"\n",
        MiscLib.ColorYellow, MiscLib.ColorReset, MiscLib.ColorYellow, x, MiscLib.ColorReset)
    dbr := database["alice"]

    //
    //

    v := dbr.v
    fmt.Printf(`Server looks up in the database 'v' for "alice", v=%d`+"\n", v)

    // ------------------------------------------------------------------------
    // Response to Message 1, Server back to client
    // ------------------------------------------------------------------------
    {
        dbr := database["alice"]
        dbr.x = x
        database["alice"] = dbr
    }

    y := big.NewInt(0)
    // Bob is the Server:
    // Bob sends the challenge 'e' back to Alice e to do the computation
    // e := big.NewInt(1000) // how chose (random?)
    randNum = genRan(999)
    e := big.NewInt(randNum)
    {
        dbr := database["alice"]
        dbr.e = e
        database["alice"] = dbr
    }
    fmt.Printf(`%sResponse: {"status":"success", "e":%d}%s`+"\n", MiscLib.ColorCyan, e, MiscLib.ColorReset)
    {

        // Alice(client) now computes: y = a*e % q
        t2 := big.NewInt(0)
        t2.Mul(a, e)
        t2.Mod(t2, q) // 45664
        t2.Add(t2, r) // 851 is correct

        y = t2
        fmt.Printf("y=%s\n", y) // Prints 851
        fmt.Printf(`%s/api/login-pt1%s, send-data: %s{"username":%q,"y":%d}%s`+"\n",
            MiscLib.ColorYellow, MiscLib.ColorReset, MiscLib.ColorYellow, "alice", y, MiscLib.ColorReset)
        fmt.Printf(`response: %s{"status":"success","y":%d}%s`+"\n",
            MiscLib.ColorCyan, y, MiscLib.ColorReset)
    }
    {
        dbr := database["alice"]
        dbr.y = y
        database["alice"] = dbr
    }

    // At this point.
    // Alice has 'y' - by calculating it from 'e'
    // Bob has 'y' saved in database.

    // ------------------------------------------------------------------------
    // Message 2 - Client (Alice) with response to challenge.
    // ------------------------------------------------------------------------

    z := big.NewInt(0)
    {
        // Bob (server) verifies: x == z == (a^y) * (v^e) % p
        // or a Better version of the same calulation
        // Bob (server) verifies: x == z == ((a^y)%p)*((v^e)%p) % p

        dbr := database["alice"]
        v := dbr.v // Validation value
        e := dbr.e // random saved from earlier
        y := dbr.y // calculated on server and saved.
        fmt.Printf(`Server looks up in the database 'v','e','y' for "alice", v=%d`+"\n", v)
        //

        t3 := big.NewInt(0)
        t3.Exp(alpha, y, p)
        t4 := big.NewInt(0)
        t4.Exp(v, e, p)
        t5 := big.NewInt(0)
        t5.Mul(t3, t4)
        t5.Mod(t5, p)
        z = t5
    }

    fmt.Printf("z=%s\n", z)
    fmt.Printf(`%s/api/login-pt2%s, send-data: %s{"username":"alice"}%s`+"\n",
        MiscLib.ColorYellow, MiscLib.ColorReset, MiscLib.ColorYellow, MiscLib.ColorReset)

    // ------------------------------------------------------------------------
    // Response 2 - Success/Fail message from server back to client
    // ------------------------------------------------------------------------

    {
        // fetch 'x' from earlier
        dbr := database["alice"]
        x = dbr.x

        if x.Cmp(z) == 0 {
            fmt.Printf("%sAuthoized! Yea, 'alice' is a valid user%s\n", MiscLib.ColorGreen, MiscLib.ColorReset)
            fmt.Printf(`%sResponse: {"status":"success","msg":"'alice' is logged in"}%s`+"\n",
                MiscLib.ColorCyan, MiscLib.ColorReset)
        } else {
            fmt.Printf("%sNope nope nope%s\n", MiscLib.ColorRed, MiscLib.ColorReset)
            fmt.Printf(`%sResponse: {"status":"error","msg":"'alice' is not a valid user"}%s`+"\n",
                MiscLib.ColorRed, MiscLib.ColorReset)
        }
    }

}

// Not! Not a cryptographic random number generation!
func genRan(m int) int64 {
    return int64(rand.Intn(m))
}
```

Reunsts of 2 runs:



```
+=>
+=> go run sip01.go
Setup Complete: v=13136
"Save 'v' for user "alice"
/api/register-user, send-data={"user":"alice","v":13136}
Response: {"status":"success", "username":"alice", "msg":"is registered."}
random genrated: 227
Send To Bob : x=86161
/api/login, send-data={"username":"alice","x":86161}
Server looks up in the database 'v' for "alice", v=13136
Response: {"status":"success", "e":621}
y=1008
/api/login-pt1, send-data: {"username":"alice","y":1008}
response: {"status":"success","y":1008}
Server looks up in the database 'v','e','y' for "alice", v=13136
z=86161
/api/login-pt2, send-data: {"username":"alice"}
Authoized! Yea, 'alice' is a valid user
Response: {"status":"success","msg":"'alice' is logged in"}
+=>
+=>
+=>
+=>
+=> go run sip01.go
Setup Complete: v=13136
"Save 'v' for user "alice"
/api/register-user, send-data={"user":"alice","v":13136}
Response: {"status":"success", "username":"alice", "msg":"is registered."}
random genrated: 954
Send To Bob : x=26648
/api/login, send-data={"username":"alice","x":26648}
Server looks up in the database 'v' for "alice", v=13136
Response: {"status":"success", "e":874}
y=984
/api/login-pt1, send-data: {"username":"alice","y":984}
response: {"status":"success","y":984}
Server looks up in the database 'v','e','y' for "alice", v=13136
z=26648
/api/login-pt2, send-data: {"username":"alice"}
Authoized! Yea, 'alice' is a valid user
Response: {"status":"success","msg":"'alice' is logged in"}
+=>
```