Homework 6

COM402 — Information Security and Privacy 2019

- Submission instructions are on Moodle. Submissions sent after the deadline **WILL NOT** be graded.
- In the event that you find vulnerabilities, you are welcome to disclose them to us (can even have a bonus !)
- Do not forget to submit your source files to Moodle along with your tokens.

Exercise 1: NOP sled galore!

It's your good friend NOP sled from the buffer overflow exercise! You don't need to do anything for the first part of this homework. Isn't that awesome? Go ahead and skip to the exercise 2!

Exercise 2: Blockchain (50p)

Everybody has a blockchain — *so why not you too?* We set up a simple blockchain at `com402.epfl.ch` that waits for new blocks. You don't have to include any broadcast protocol like in Bitcoin, but we ask you to perform some mining and send the block to the com402 server.

Installation

In this exercise, you'll have to program in Go. **Please install a Go version of at least 1.11.** You will have **`template.go`** and **`go.mod`** files. Please put them in the same folder. You have to complete template.go. Once your code is complete, please run:

```
go build template.go
```

This creates a binary, **template**. To run the code, use **`./template`**.

Communicating with the blockchain

To communicate with the blockchain, you have two methods:

```
# returns a list of all blocks that are stored in the skipchain
# identified by the genesis id.
# Input:
# - roster, representing the set of nodes maintaining the blockchain
-- conodes -- which is created from the server URL (already in
template)
# - sid, which is the genesis-id (already in template)
```

```
# Output:
# - all blocks since the genesis-block
# - an error message

cl := skipchain.NewClient() (already in template)
cl.GetUpdateChain(roster, sid)

# Asks the blockchain to add a block
# Input:
# - first_block, the first block from the list of blocks returned by
# the GetUpdateChain() call
# - nil -- this field is not required, so please pass nil
# - data -- data you want to add to the latest block
# Output:
# - previous block of the blockchain
# - latest block of the blockchain which should be yours
# - an error message (if something went wrong)

cl.StoreSkipBlock(first_block, nil, data)
```

The URL for the blockchain is:
`tls://com402.epfl.ch:7002 (already in template)`

And the genesis-id (which corresponds to the skipchain-id):
`6a6ea0e9f701862ddd746ad80d331fc4834a5f358443572eadc138a93cfdbd3a (already in template)`

Mining

To mine a block, you need to create binary data that conforms to some SHA256 value. The data format is:

```
nonce = data[0:32]
hash_of_last_block = data[32:64]
full_email = data[64:]
```

You find the hash of the last block in the return value of **`GetUpdateChain()`**. Only the last block is important. The libraries **crypto/rand** and **crypto/sha256** will be helpful.
For the data to be accepted by the server, the first three bytes of the SHA256 hash of the data has to be 0! This is what the nonce is for. You either use it as a counter or take random data (both should give the same result) and iterate, creating new SHA256 hashes, **until you find one hash that gives 0x000000 in the beginning**. With our simple sample-script it takes about 4 minutes, so don't worry if you don't immediately find a valid hash! For testing reasons you can start with 0x0000 to see if your method works. Even though it will be rejected by the blockchain…

If you find a better way than to try out different combinations, be sure to write a paper about it ;)

Adding the block

Once you found a correct data for your block, submit it to the blockchain with `storeBlock()`. Be sure to check the return message. In case the block is not accepted, you will get an error message.

You can also check on http://com402.epfl.ch/static/conode.html for error messages from the nodes.

Finished

You need to add at least 3 blocks to the blockchain to get your token from the website.

Exercise 3: NOP sled galore!

Miss me? It's your good friend NOP sled from the buffer overflow exercise once again! Go ahead and skip to the exercise 4!

Exercise 4: Private Information Retrieval (50p)

In this exercise you will implement a magic protocol called information-theoretic PIR (Private Information Retrieval, see lectures): as a client, query a set of servers for a piece of information without revealing to them what you are asking for.

We will implement a communication system that uses PIR to privately retrieve chat messages. The system's protocol consists of 2 phases: the setup phase where clients and servers exchange cryptographic keys, and the communication phase where clients upload and download their chat messages from the servers.

In more detail, we have the following steps in the protocol:

- **Setup phase**:
  - Servers start listening for UDP packets on a given IP and port
  - A Client, let's call it *Com402Client*
- **Communication phase — Upload**:
  - *Com402Client* client generates *n* chat messages
  - *Com402Client* then sends *n* messages to the *primary* server
  - Servers then store the messages:
    - Primary server stores them and sends to the next server in chain
    - Next server does the same, until eventually the last server in the chain obtains the plaintext message.

    - At the end of this phase all the servers have *n* plaintext chat messages. All the messages have an index, so the servers have them in the same order as well.
- **Communication phase — Download**:
    - In this phase a client, let's call it *StudentClient*, wants to download a message with a specific index, but doesn't want servers to learn which index is that. For this, *StudentClient* uses PIR.
    - *StudentClient* generates *m* bitmasks (one for each server) of length *n* (total number of messages), in such way that when XORed, the result will be the target index (index of a message which *StudentClient* wants to retrieve).
    - *StudentClient* sends bitmasks to the servers and asks each server to select messages according to the bitmask, XOR the selected messages and send the result back
    - *StudentClient* can the recover the desired message by combining all the responses from the servers.

We provide you with two skeleton scripts, one for the servers and one for the clients. You can download them on the com402 website. In the scripts you will find detailed instructions on what and how you should implement. We will test and grade two things:

1. Your server code (35p): once you finish your implementation, upload your script to com402
2. Your client code (15p): we will grade only the client which performs the PIR

When submitting your solutions, please submit your server and client separately (i.e. submit your server and get a token for it, then submit your client and get a second token for it)

You are provided with enough code to make the whole system work locally on your machine, so please test thoroughly before submitting the solution.

Good luck!