COMP4337 – ASSIGNMENT

MID-TERM REPORT

z5116870 – Roark Menezes

VIDEO LINK: https://www.dropbox.com/s/0ju4kv34jy1ixne/Mid%20term%20DIMY.mp4?dl=0

ASSIGNMENT DIARY & PROGRESS SO FAR

Monday 29/03/21

Today, I began reading the DIMY (Did I Meet You) article, titled "DIMY: Enabling Privacy-preserving Contact Tracing. I started taking notes on the general idea of the DIMY protocol such as the techniques used including elliptic curves, Diffie-Helman key exchange, Shamir secret sharing (SSS), bloom filters and blockchain technology. I had to do extra research on SSS, bloom filters and blockchain technology since both of those concepts were relatively new to me.

The notes I took allow me to go into great detail and have given me a greater understanding of each concept, which is crucial during implementation, however a brief rundown of these are as follows: SSS is a mechanism which involves two parts, sharing and reconstruction. The secret is first split into parts and broadcast over many devices(sharing) and with enough parts the entire secret can be rebuilt (reconstruction). Bloom filters are probabilistic data structures taking the form of bit arrays, used to represent membership of a set, such as usernames. Queries to the set are run through a hash function which outputs an index of the bloom field. If the value of the bloom field at the index is 1 then the element MAY exist in the set, since bloom fields have a chance to output false positives. Blockchain technology focuses on decentralisation of transactions, eliminating the need for one entity controlling all transaction, such as a bank. In blockchain technology, each user maintains a ledger of all transactions enacted between each other, forming a public distributed ledger. Scripts known as smart contracts are used to handle requests such as execute payment, request payment etc., but in the context of DIMY, the blockchain will be used to maintain a system where users can upload their list of close contacts and other users can anonymously check for exposure to COVID. A diagram of this is shown below, taken from the DIMY article:

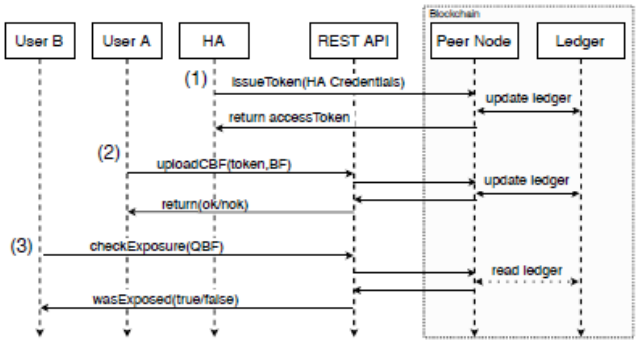

Fig. 6. Uploading to the blockchain.

I also took notes on how these concepts are applied specifically to the DIMY protocol. The DIMY protocol can be broken up into 5 stages: generating identifiers (elliptic curve Diffie-Hellman i.e. ECDH), advertising identifiers (SSS), storing encounter information (bloom filters), uploading encounter information (REST APIs) and the contact verification process (blockchain).

Tuesday 30/03/21

The plan for today was to begin Task 1, which was to generate the 16-byte (128-bit) Ephemeral ID as part of the generating identifiers component of the DIMY protocol. I chose to use python since I am most comfortable with this language and it was used for Lab 1. This ID generation needs to be done using ECDH, so I began researching ways in which this can be carried out in python, and I came across a few modules but the only one that managed to use a curve capable of outputting a 128-bit key was the "ecdsa" module. It contained an elliptic curve called "SECP128R1. Using the functions

provided on the GitHub repository for the module, I was able to generate a 128-bit key. The next part was I needed to get this ID generation procedure to repeat every 60 seconds, so I incorporated the use of threading, and put the procedure in its own function. Running the function now allows the ephemeral ID to update at 60 second intervals (changed to 2 seconds for testing as seen in the image below).

```
# 1. GENERATE EPHEMERAL ID
def generateEphID():
    global Eph_ID
    # Generate the 16 byte Ephemeral ID
    threading.Timer(2, generateEphID).start()
    sk = SigningKey.generate(curve = SECP128r1)
    vk = sk.verifying_key
    Eph_ID = sk.to_string()
    print(Eph_ID)
    print(len(Eph_ID))
    msg = b"message"
    signature = sk.sign(msg)
    assert vk.verify(signature, msg)
```

```
Command Prompt

D:\T1 2021\COMP4337\Assignment>python Dimy.py
b'\x17aX\xf9\xe0\xc1\xe1k\xa0D:\xcf\x01\xca\n='
16
EphID =  b'\x17aX\xf9\xe0\xc1\xe1k\xa0D:\xcf\x01\xca\n='
```

From the output above I confirmed that the output key was indeed 128-bit (16 bytes), by outputting in string form (using the to_string() function).

Wednesday 31/03/21

The next step after creating the ephemeral ID would be to broadcast/receive other ephemeral ID chunks, which were split using SSS. The “pycryptodome” module has inbuilt SSS functions that can split the key, given the parameters k and n, where k is the minimum number of chunks needed to recreate the secret and n is the total number of chunks. The values of k = 3 and n = 6 were used, as per the assignment specification. The output is shown below:

```
    # 2. SHAMIR SECRET SHARING
    # Split EphID into 6 parts, transmit one every 10 seconds
    print("EphID = ", Eph_ID)
    # Calculate hash of entire ID
    hash = hashlib.sha1(Eph_ID)
    print(hash)
    # Split ID
    shares = Shamir.split(3, 6, Eph_ID)
    # Print IDs
    for idx, share in shares:
        print("Index #%d: %s" % (idx, hexlify(share)))
```

```
Command Prompt

D:\T1 2021\COMP4337\Assignment>python Dimy.py
b'\xa1o\xa8\xc7\xa83\xa2\x8f\x9d\xd3"\xa3\xb1\xfc\xaaE'
16
EphID =  b'\xa1o\xa8\xc7\xa83\xa2\x8f\x9d\xd3"\xa3\xb1\xfc\xaaE'
<sha1 HASH object @ 0x0000017E38FEFD50>
Index #1: b'910ac8c88f655ce47a7c191cdeb6eb6b'
Index #2: b'1e573cf338cb89958d1d031db28731bd'
Index #3: b'2e325cfc1f9d77fe6ab238a2ddcd7093'
Index #4: b'a0d7d005f094ab8cc20e3adcc0b7fa2b'
Index #5: b'90b2b00ad7c255e725a10163affdbb05'
Index #6: b'1fef4431606c8096d2c01b62c3cc61d3'
```

These chunks were printed to confirm that n chunks were made. To test that the secret can be reconstructed, the opposite function to split() needs to be used which is called combine(). This takes in an index and the chunk, separately. It only needs 3 many shares to recreate the secret, however this is proposed to be completed at a later date.

OVERALL PLAN

DEADLINE	TASK	MAIN CONCEPTS
08/04/21	2	- SSS
11/04/21	3, 4, 5	- Combining chunks to get EphID - Combining EphIDs to get EncID
15/04/21	6, 7, 8	- Encoding bloom filter and deleting EncID - Make DBF for 10 min period (max 6 and delete any DBFs older than 60 mins) - Every 60 mins also make QBF by combining all DBFs (using bitwise OR)
18/04/21	9, 10	- Send QBF to API, check for close contact (display result) - Use UDP to implement send/receiving of messages between devices (e.g EphIDs)
20/04/21	EXTENSION PART 1	- Use TCP on port 55000 to run backend server, using my PC - Connect RasPI4 to PC using TCP connection - Test transfer of CBF/QBF data
21/04/21	EXTENSION PART 2	- Store all CBFs - Perform matching for QBFs received - Inform device uploading QBF the result of matching (if not CBF then return "not matched"
22/04/21	Report, Video and Code	- Finalize report - Finish video demonstration - Fix all code styling
23/04/21	**SUBMISSION**	

ISSUES FACED

So far, the only issues faced were when trying to generate the 128-bit ephemeral ID. It was difficult finding a curve that was capable of this as most ECDH curves would output 256-bit keys. After searching the internet and taking help from colleagues on Teams I found the "ecdsa" module which

had the right curve for the task. Additionally, when creating the thread for updating the ephemeral ID every 60 seconds, a global data type needed to be used however updating in a function needed it to be declared as a global variable both outside the function and inside. Previously I was only declaring it global outside the function, but now it seems to be working fine.

Another issue faced, and not yet solved, was trying to connect to the back-end REST API. The URL given seems to allow connection however the data type specified is a “JSON” object, for both the CBF and QBF bloom filters. I have not learnt enough about JSON objects to familiarize myself with communicating with REST APIs and JSON objects.