Implementing Signal Protocol

Cryptography CS 411 & CS 507 Term Project for Fall 2022

Atıl Utku Ay
Computer Science & Engineering
Sabancı University
İstanbul

Abstract

You are required to develop a simplified version of the Signal Protocol, which provides forward secrecy and deniability. Working in the project will provide you with insight for a practical cryptographic protocol, a variant of which is used in different applications such as WhatsApp.

1 Introduction

The project has three phases:

- **Phase I** Developing software for the Public Key Registration
- **Phase II** Developing software for receiving messages from other clients
- **Phase III** Developing software to communicate with other clients

1. All coding development will be in the Python programming language. More information about the project is given in the subsequent sections.
2. You should connect to the university's VPN to be able to connect to the server **if you are connecting from outside of the campus**. Otherwise, you won't be able to send your requests.
   Check the IT's website (`https://mysu.sabanciuniv.edu/it/en/openvpn-access`)

2 Phase I: Developing software for the Public Key Registration

In this phase of the project, you are required to upload one file: "Client.py". You will be provided with "Client_basics.py", which includes all required communication codes.

In this protocol, Elliptic Curve Cryptography (ECC) is used in key exchange protocol and digital signature algorithm with NIST-256 curve. You should select "`secp256k1`" for the elliptic curve in your python code. There are three types of public keys; Identity Key (`IK`), Signed Pre-key (`SPK`) and One-time Pre-key (`OTK`).

2.1 Identity Key (IK)

Identity Key (IK) consists of a long-term public-private key pair, which each party generates once and uses to sign his/her SPK as shown in Section 2.2 .

2.1.1 Registration of Identity Keys

The identity public key of the server $\texttt{IK}_S\texttt{.Pub}$ is given below.

```
X:0xce1a69ecc226f9e667856ce37a44e50dbea3d58e3558078baee8fe5e017a556d
Y:0x13ddaf97158206b1d80258d7f6a6880e7aaf13180e060bb1e94174e419a4a093
```

Firstly, you are required to generate an identity private and public key pair $\texttt{IK}_A\texttt{.Pri}$ and $\texttt{IK}_A\texttt{.Pub}$ for yourself. The key generation is described in the "Key Generation" algorithm in Section 2.4. Then, you are required to register your public key with the server. The identity key registration operation consists of four steps:

1. After you generate your identity key pair, you should sign your ID (e.g., 26045). The details of the signature scheme is given in the "Signature Generation" algorithm in Section 2.4. Then, you will send a message, which contains your student ID, the signature tuple and $\texttt{IK}_A\texttt{.Pub}$, to the server. The message format is

   {'ID': stuID, 'H': h, 'S': s, 'IKPUB.X': ikpub.x, 'IKPUB.Y': ikpub.y}

   where stuID is your student ID, h and s are the signature tuple and ikpub.x and ikpub.y are the x and y and coordinates of $\texttt{IK}_A\texttt{.Pub}$, respectively. A sample message is given in 'samples.txt'.

2. If your message is verified by the server successfully, you will receive an e-mail, which includes your ID and a 6 digit verification code: code.

3. If your public key is correct in the verification e-mail, you will send another message to the server to authenticate yourself. The message format is "{'ID': stuID, 'CODE': code}", where code is the verification code which, you have received in the previous step. A sample message is given below.

   {'ID': 26045, 'CODE': 209682}

4. If you send the correct verification code, you will receive an acknowledgement message via e-mail, which states that you are registered with the server successfully and contains a code to reset your identity key if you need. (You must save the reset code to delete your IK from server, in case you need (e.g., your identity key is lost or compromised).)
   Note: Once you receive an acknowledgment email, your Identity key will be saved locally in your computer.

2.1.2 Resetting your IK

If you lose your private identity key $\texttt{IK}_A\texttt{.Pri}$, and need to reset your identity key pair, you should send a message to the server to delete your public identity key $\texttt{IK}_A\texttt{.Pub}$ from the server. The message format is "{'ID': stuID, 'RCODE': rcode}", where rcode is reset code which was provided in the acknowledgement e-mail. A sample message is as follows

{'ID': 26045, 'RCODE': 209682}

If you lose the reset code, you should send an e-mail to cs411tpserver@sabanciuniv.edu. Your IK will be deleted from the server after 8 hours.

2.2 Signed Pre-key (SPK)

Signed Pre-key is another long-term key pair, which all parties generate once and is used in the registration of one-time pre-keys (OTK).

2.2.1 Registration of SPK

After you have registered your identity key, you are required to generate one pair of signed pre-key; $\texttt{SPK}_A\texttt{.Pub}$ and $\texttt{SPK}_A\texttt{.Pri}$. Then, you must sign the public key part of the signed pre-key, $\texttt{SPK}_A\texttt{.Pub}$ using your identity key $\texttt{IK}_A$. The signature, for which a scheme is given in Section 2.4, must be generated for the concatenated form of the public signed pre-key: $(\texttt{SPK}_A\texttt{.Pub.x} \parallel \texttt{SPK}_A\texttt{.Pub.y})$. Finally, you must send your signed pre-key to the server in the form of

{'ID': stuID , 'SPKPUB.X': spkpub.x, 'SPKPUB.Y': spkpub.y, 'H': h, 'S': s},

where h and s denote the signature tuple. If your signed pre-key is registered successfully, the server will return its signed pre-key $\texttt{SPK}_S\texttt{.Pub}$ in the same format. After you check the validity of the signature of $\texttt{SPK}_S\texttt{.Pub}$, you may use it.

2.2.2 Resetting your SPK

If you lose your private signed pre-key $\texttt{SPK}_A\texttt{.Pri}$, and need to reset your signed pre-key pair, you should sign your stuID using your identity key $\texttt{IK}_A$ and send a message to the server to delete your public identity key $\texttt{SPK}_A\texttt{.Pub}$ from the server. The message format is

{'ID': stuID, 'H': h, 'S': s}

2.3 One-time Pre-key (OTK)

One-time Pre-keys are the keys which are used to generate symmetric session keys in communication with other clients. Therefore, each client in the system must register his/her OTKs to the server before the communication.

2.3.1 Generating HMAC Key ($\texttt{K}_{\texttt{HMAC}}$)

In the registration of OTKs, a hash-based MAC (HMAC) function will be used for authentication to provide deniability (instead of digital signature). Therefore, you should generate a symmetric HMAC Key ($\texttt{K}_{\texttt{HMAC}}$) before the registration of OTKs. $\texttt{K}_{\texttt{HMAC}}$ will be computed as follows:

- T = SPK_A.Pri · SPK_S.Pub (Diffie-Hellman with SPKs of the client and the server)
- U = {b'CuriosityIsTheHMACKeyToCreativity' ‖ T.y ‖ T.x}
- $K_{\texttt{HMAC}}$ = SHA3_256(U)

2.3.2 Registration of OTKs

Before communicating with other clients, you must generate 10 one-time public and private key pairs, namely $\texttt{OTK}_{A_0}, \texttt{OTK}_{A_1}, \ldots, \texttt{OTK}_{A_9}$. The key generation is described in the "Key Generation" algorithm in Section 2.4.

Then, you must compute an HMAC value for each of your public one-time pre-key $\texttt{OTK}_{\texttt{A}_i}$ using the HMAC-SHA256 function and $\texttt{K}_{\texttt{HMAC}}$ as the key. The HMAC value must be generated for concatenated form of the one-time public keys (e.g., ($\texttt{OTK}_{A_i}\texttt{.Pub.x} \,\|\, \texttt{OTK}_{A_i}\texttt{.Pub.y}$) for the i^{th} one-time pre-key). Finally, you must send your one-time public keys to the server in the form of

{'ID': stuID, 'KEYID':i , 'OTKI.X': OTKi.x, 'OTKI.Y': OTKi.y, 'HMACI': hmaci},

where i is the ID of your one-time pre-key. You must start generating your one-time pre-keys with IDs from 0 and follow the order.

2.3.3 Resetting your OTKs

If you lose the private part of your one-time pre-keys, and need to reset your one-time pre-key pairs, you should sign your stuID using your identity key IK and send a message to the server to delete your registered one-time pre-keys. The message format is

{'ID': stuID, 'H': h, 'S': s}

2.4 The Digital Signature Scheme

Here, you will develop a Python code that includes functions for signing given any message and verifying the signature. For this digital signature scheme, you will use an algorithm, which consists of three functions as follows:

- **Key Generation:** A user picks a random secret key $0 < s_A < n - 1$ and computes the public key $Q_A = s_A P$.
- **Signature Generation:** Let m be an arbitrary length message. The signature is computed as follows:
  1. $k \leftarrow \mathbb{Z}_n$, (i.e., k is a random integer in $[1, n-2]$).

2. $R = k \cdot P$
3. $r = R.x \pmod{n}$, where $R.x$ is the x coordinate of R
4. $h = \text{SHA3\_256}(r||m) \pmod{n}$
5. $s = (k + s_A \cdot h) \pmod{n}$
6. The signature for m is the tuple (h, s).

- **Signature Verification:** Let m be a message and the tuple (s, h) is a signature for m. The verification proceeds as follows:

1. $V = sP - hQ_A$
2. $v = V.x \pmod{n}$, where $V.x$ is x coordinate of V
3. $h' = \text{SHA3\_256}(v||m) \pmod{n}$
4. Accept the signature only if $h = h'$
5. Reject it otherwise.

IMPORTANT:

- This step is about generating the necessary keys for our communication protocol. Therefore, make sure to run each step alone. For instance, if you already generated your `Identity Key`, then you moved to the part where you generate the `Signed Pre-Key`, you make sure you do not run the previous code of generating `IK` again.
- **WHAT CAN GO WRONG?** The reason behind the previous note is that you may end up asking the server to send you the `rcode` multiple times (of course after resetting your IK). After a while, you may not see the message (falling within the SPAM box). Anyhow, you can test all the functions at once if you have completed the respective phase and want to see if everything is working flawlessly.

3 Phase II: Developing software for receiving messages from other clients

In this phase of the project, you are required to upload one file: "Client_phase2.py". You will be provided with "Client_basic_phase2.py", which includes all required communication codes. Moreover, you will find sample outputs for this phase in 'sample_vector.txt', which is also provided on SUCourse.

You are required to develop a software for downloading five (5) messages from the server, which were uploaded to the server originally by a pseudo-client, which is implemented by us, in this phase[1]. The details are given below.

In Phase I, you have already implemented the registration protocol. The details are explained in Section 2. If you have not implemented yet, you must implement the protocol before Phase II and register your long-term public key with the server.

[1]This is indeed an asynchronous messaging application, whereby other users can send you a message even if you are not online. Yes, exactly like WhatsApp application.

[2]In this protocol, a session key, K_S, is generated (the details will be explained in Section 3.1.1) for each message block[2] using ECDH. Then, one encryption and one HMAC key are generated from K_S for each message in the message block using a key derivation function KDF chain (the details will be explained in Section 3.1.2). The messages are encrypted using AES256-CTR and authentication is provided using HMAC-SHA256[3].

3.1 End-to-End Secure Communication Scheme

The protocol, which you implement in this project, provides end-to-end security in communication. In other words, no other party (including the server) can read and/or modify original messages. Moreover, forward secrecy is another feature of the protocol. Therefore, long-term keys should not be used for encryption. In order to achieve the end-to-end secure communication with forward secrecy, the communicating parties must use one-time keys to compute a session key (K_S). Then, they must generate an encryption and an HMAC key for each message of the corresponding message block using a key derivation function (KDF) chain. Finally, they encrypt the messages using AES256-CTR for confidentiality and compute the HMAC-SHA256 value of the ciphertext for integrity.

3.1.1 Session Key (K_S)

As mentioned before, both parties should establish a session key (K_S) for a message block to generate encryption and HMAC keys at first. K_S is computed using DH key exchange protocol. Therefore, when a user wants to send a message block to another user, s/he requests an OTK of the receiver from the server. Then, s/he generates a DH key pair which is called as "Ephemeral Key (EK)" and it is used in the DH key exchange along with the receiver's OTK. The computation of (K_S) in the receiver side is given below:

- T = OTK_A.Pri · EK_B.Pub[4]
- U = {T.x ‖ T.y ‖ b'ToBeOrNotToBe'} [5]
- K_S = SHA3_256(U)

where OTK_A.Pri denotes the private OTK of the receiver and EK_B.Pub is public ephemeral key of the sender.

3.1.2 Key Derivation Function (KDF) Chain

The Key Derivation Function (KDF) is a cryptographic function that takes a secret KDF key (K_{KDF}) and returns an encryption key, an HMAC key and KDF key for the next iteration. If there are more than one message in a message block, KDF is used more than once to generate different encryption and HMAC keys for each message. The outputs of KDF are computed as follows:

[2]Message block refer to a group of messages, which are sent by a client consecutively without receiving any message in between from the other party. You can think of Whatsapp as an example. You may send 3 messages consecutively to your friend. Then, s/he may send 2 messages. Finally, you may send 4 messages. And so on. Each of them is called a message block.

[3]Note that, SHA3_256 is used for key generation. On the other hand, SHA2_256 is used for MAC computation.

[4]The sender will compute T = EK_B.Pri·OTK_A.Pub. If you are confused, check how Diffie-Hellman Key Exchange Protocol is working. The rest of the computation is the same for the sender.

[5]https://youtu.be/pGZxOhCP6eo?t=307

- K_{ENC} = SHA3_256(K_{KDF} || b'YouTalkingToMe')[6]
- K_{HMAC} = SHA3_256(K_{KDF} || K_{ENC} || b'YouCannotHandleTheTruth')
- $K_{KDF.Next}$ = SHA3_256(K_{ENC} || K_{HMAC} || b'MayTheForceBeWithYou')

After K_S is computed, it will be used as the KDF key for the first message. If there is only one message in the message block, K_{ENC} and K_{HMAC} will be used for encryption and HMAC values, respectively. However, $K_{KDF.Next}$ will not be used (only if there is one message in the block). Otherwise, $K_{KDF.Next}$ is used as the KDF key to generate K_{ENC} and K_{HMAC} for the next message of the message block. Figure 1 shows how the KDF chain is working for a message block which contains 3 messages, m_1, m_2 and m_3 namely[7].

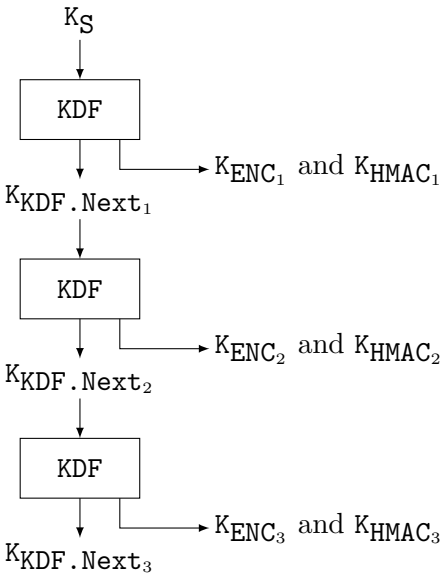

Figure 1: KDF Chain for a message block with 3 messages

In this example, K_S is computed by both parties before the communication and used as K_{KDF} to generate K_{ENC_1} and K_{HMAC_1}, where K_{ENC_1} and K_{HMAC_1} are the encryption and HMAC keys of m_1, respectively. K_{KDF_1} is used to generate K_{ENC_2} and K_{HMAC_2} for m_2. And so on...

3.2 Downloading Messages from the server

As mentioned above, you will download 5 messages from the server. In order to download one message from the server, you must sign your ID using your Identity Key and send a message to the server in the form of

```
{'ID': stuID, 'S': s, 'H': h }
```

The message will be downloaded in the form of

[6] https://youtu.be/-QWL-FwX4t4

[7] For convenience in implementation, message index starts from 1 instead of 0.

{‘IDB’: stuIDB, ‘OTKID’: otkID, ‘MSGID’: msgID, ‘MSG’: msg , ‘EK.X’: EK.x ,‘EK.Y’: EK.y },

where stuIDB is the ID of the sender, otkID is the ID of your one-time prekey, which is used to generate session key, msgID indicates the order number of the message in the corresponding message block, msg contains the nonce, the ciphertext and MAC value of the ciphertext[8], and EK.x and EK.y are x and y coordinates of the ephemeral public key (E_K.Pub) of the sender, respectively. (**Note:** there is only one message format. Therefore, all messages in the same message block contain the same otkID and EK_B.Pub information.)

3.3 Decrypting the Messages From the Pseudo-Client

In this phase, you will request and receive **five** messages from a pseudo-client, which simulates your correspondence sending messages to you. Therefore, you need to request to generate 5 messages from pseudo-client (via the server) at first. The message format is

{‘ID’: stuID, ‘H’: h, ‘S’: s}

When you request five messages, the messages will be ready on the server and you may download them (explained in Section 3.2). After you download the messages, you need to compute K_S and KDF chain at first. Then, you should check the MAC values of the messages and decrypt them (4 messages will have valid MAC values, but one of them will not.). Finally you should send a message for each received message, which you downloaded, in the form of:

{‘IDA’: stuIDA, ‘IDB’: stuIDB, ‘MSGID’: msgID, ‘DECMSG’: decmsg },

where stuIDA and stuIDB are the ID of you and the pseudo-client, respectively. msgID is the ID of the message (The order of the sent messages will be the same as the received messaged. In other words, you should put the same message ID with the received message.) and decmsg is the decrypted form of the message if the MAC value is valid. Otherwise, you should set decmsg field to "INVALIDHMAC".

3.4 Displaying the Final Message Block

After receiving the messages from the pseudo-client and **decrypting them successfully**, you are asked to ask the server about whether the pseudo-client has deleted some messages from the received block or not. This case is similar to the case in WhatsApp messages, a sender may delete some messages after sending them. You won't implement the exact scenario. However, you will display the messages of the received block as given in the sample run. Here are the steps:

1. Send the server a request that includes your stuID and its signature using your IK in the form:
   {‘ID’: stuID, ‘H’: h, ‘S’: s}

[8] msg = nonce‖ciphertext‖MAC
Where the nonce is the AES-CTR nonce of 8 bytes. You can conclude the size of the MAC value from the used hash function algorithm.

It is the same request as in Step 3.3 but uses a different service (given to you in the `Client_basics.py`).

2. After the server verifies your signature, then it will send (as a result to your request) a list that has the message ids (`MSGID`) of the ones that got deleted.

3. Finally, you will display the messages in the same manner given in the sample run.

4. **Note:** Bear in mind that there is an invalid message among the block. You should not print it as you could not verify its signature. However, the pseudo-client can delete it, and you may receive its `MSGID` as well.

4 Phase III: Developing software to communicate with other clients

5 Appendix I: Timeline, Deliverables, Weight & Policies

Project Phases	Deliverables	Due Date	Weight
Project announcement		18/12/2022	
First Phase	File: Client.py	25/12/2022	40%
Second Phase	File: Client.py	01/01/2023	30%
Third Phase	File: -	-	-

5.1 Policies

- You may work in groups of two.
- Submit all deliverables in the zip file "cs411_507_tp2_surname1_surname2.zip".
- Only one submission from one student is enough. However, the 2^{nd} policy point should be respected.
- For codes, only .py scripts should be uploaded.
- **Points will be deducted if you don't respect the naming and file format criteria.**
- You may be asked to demonstrate a project phase to a TA or the instructor.
- Since you will interact with a remote server to register your keys, send and receive messages. This server script itself is a validation method to check whether your codes are working correctly or not. We will also use it to check your implementation. If your implementation in a project phase fails to pass the validation, you will get no credit for that phase.
- Your codes will be checked for their similarity to other students' codes; and if the similarity score exceeds a certain threshold you will not get any credit for your work.