Implementing Signal Protocol

Cryptography CS 411 & CS 507 Term Project for Fall 2021

E. Savaş
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

All coding development will be in the Python programming language. More information about the project is given in the subsequent sections.
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:93223115898197558905062012489877327981787036929201444813217704012422483432813
Y:89856292032257671854649200941983642557409873467439120718433039755876953376l9
```

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., 18007). 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': 18007, '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).)

2.1.2 Reseting 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': 18007, '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 Prekey 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 Reseting 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 ($\mathtt{K_{HMAC}}$)

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 ($\mathtt{K_{HMAC}}$) before the registration of OTKs. $\mathtt{K_{HMAC}}$ will be computed as follows:

- `T = SPK_A.Pri` $\cdot$ `SPK_S.Pub` (Diffie-Hellman with SPKs of the client and the server)
- $\mathtt{U} = \{\mathtt{T.x} \parallel \mathtt{T.y} \parallel \mathtt{b'NoNeedToRideAndHide'}\}$
- $K_{\mathtt{HMAC}} = \mathtt{SHA3\_256}(\mathrm{U})$

2.3.2 Registration of OTKs

Before communicating with other clients, you must generate 10 one-time public and private key pairs, namely $\mathtt{OTK}_{A_0}, \mathtt{OTK}_{A_1}, \ldots, \mathtt{OTK}_{A_9}$. 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 $\mathtt{OTK_{A_i}}$ using using the HMAC-SHA256 function and $\mathtt{K_{HMAC}}$ as the key. The HMAC value must be generated for concatenated form of the one-time public keys (e.g., ($\mathtt{OTK}_{A_i}\mathtt{.Pub.x} \parallel \mathtt{OTK}_{A_i}\mathtt{.Pub.y}$) for the i^{th} one-time pre-key). Finally, you must sent 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 Reseting 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 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.

Note that the signature generation and verification of this scheme are different from the one discussed in the lecture.

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

4 Phase III: Developing software to communicate with other clients

5 Appendix I: Timeline & Deliverables & Weight & Policies etc.

Project Phases	**Deliverables**	**Due Date**	**Weight**
Project announcement		10/12/2021	
First Phase	File: Client.py	17/12/2021	40%
Second Phase		24/12/2021	30%
Third Phase		31/12/2021	30%

5.1 Policies

- You may work in groups of two.
- Submit all deliverables in the zip file "cs411_507_tp1_yourname.zip".
- You may be asked to demonstrate a project phase to a TA or the instructor.
- In every phase, we will provide you with a validation software in Python language that can be used to check your implementation for correctness. 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.