Security Protocols: Helping Alice and Bob to Share Secrets (COMP.SEC.220)

Tutorial 9: Building a Toy CL-PKC application

Antonis Michalas
`antonios.michalas@tuni.fi`

Mindaugas Budzys
`mindaugas.budzys@tuni.fi`

Hossein Abdinasibfar
`hossein.abdinasibfar@tuni.fi`

August 19, 2024

SUBMISSION DEADLINE: 11.11.2024 AT 23:00

Tutorial Description

Welcome to the next tutorial for the Security Protocols course. The main aim of this tutorial is to give students a basic understanding of how to utilize basic ECC curve operations using the Python programming language.

Alice wishes to securely communicate with Bob in the presence of a trusted Key Generation Center (KGC). Alice watched Lecture 9 and is now aware of Certificateless Public Key cryptography, as such she wants to use this scheme.

Lets help Alice out by implementing a dummy CL-PKC program that meets her set goals. Note that this implementation is *not* considered secure. However, completing this lab will be a valuable first step towards successfully completing the second coursework of the course. You can decide to implement all three entities as a single application or if you are feeling fancy, create three different applications and enable communication between them. Your choice!

EXERCISE 1 – KEY GENERATION PHASE

A CL-PKC scheme is heavily dependent on a set of public parameters published by the KGC. For the purposes of this lab, we will limit the set of public parameters to a $k-bit$ prime q, a point G which is the generator of the group G_q, the public key of the KGC P_{pub} and a hash algorithm H of your choice (SHA-256 should be okay).

1. You have been provided with a base implementation of the basic ECC operations such as Point addition and scalar multiplication. Build upon this script to accomplish your goals for this lab.

2. Compute a random KGC master secret key x and public key P_{pub}.

$$x \in \mathbb{Z}_q^*$$

```
x = randint(0, q)                                  #hint
```

$$P_{pub} = xG$$

```
P_pub = x * G                                      #hint
```

3. For each user (i.e., Alice and Bob), compute a secret value x_i and a partial public key P_i.

$$x_i \in \mathbb{Z}_q^*$$

$$P_i = x_i G$$

4. Compute the partial private key d_i and partial public key R_i for each user at the KGC.

$$r_I \in \mathbb{Z}_q^*$$

$$R_i = r_i G$$

$$d_i = r_i + xH(ID_i, R_i, P_i)$$

   where ID_i is a user identifier. For example, Alice could be represented by the string "ID_A"

5. For each user, compute the full private key sk_i and full public key PK_i

$$sk_i = (d_i, x_i)$$

$$PK_i = (R_i, P_i)$$

Hint: Be creative with how you perform the hashing and combine the keys. All approaches are encouraged as long as you don't break your code or the math.

Notes:

- All uppercase variables except for the user identifiers are ECC points while all lower case variables are integers.
- Scalar multiplication (multiplying a point by an integer) results in a point.

Exercise 2 – Encyption and Encapsulation

This is the next phase of our dummy application for Alice and Bob. Once Alice and Bob have computed their full private and public key pairs, the fun begins. Generate a sample message to be used as the plaintext. Lets consider Alice as A and Bob as B, and assume that A is initiating the secure communication channel.

1. Alice should generate 2 random values l_A and h_A and compute U and V

$$l_A \in \mathbb{Z}_q^*, \qquad h_A \in \mathbb{Z}_q^*$$

$$U = l_A G, \qquad V = h_A G$$

2. Compute the Y and T values

$$Y = R_B + H(ID_B, pk_B) * P_{pub} + P_B, \qquad T = h_A Y$$

3. Compute the session key K_{AB}

$$K_{AB} = H(Y, V, T, ID_B, P_B)$$

4. Use the K_{AB} with an encryption scheme of your choice to encrypt the plaintext message (AES should be an easy choice).

5. Encapsulate K_{AB} and your resulting ciphertext c_{AB} from the encryption in the previous step.

$$H = H(U, c_{AB}, T, ID_A, ID_B, P_A, P_B)$$

$$W = d_A + l_A * H + x_A * H$$

$$\varphi = (U, V, W)$$

EXERCISE 3 – DECAPSULATION AND DECRYPTION

This is the final phase of our dummy application. Upon receipt of the encapsulated variable and the ciphertext. Bob should be able to retrieve the original plaintext (Hopefully :)).

1. Compute the Y and T values and compare them to values generated in step 2 of Exercise 2. Any observations?
$$Y = (d_B + x_B)G, \qquad T = (d_B + x_B)V$$

2. Retrieve the session key K_{AB}. Is this the same key that was generated in Exercise 2?
$$H = H(U, c_{AB}, T, ID_A, ID_B, P_A, P_B)$$
$$K_{AB} = H(Y, V, T, ID_B, P_B)$$

3. Decrypt c_{AB} with the session key using the encryption scheme you chose in Exercise 2 and verify that this is the same plaintext that was encrypted.