CAN304 Lab 4

Advanced Cryptographic Schemes

In last lab, you learnt the ECDH key agreement protocol and two ECDH-based AKA protocols. In this lab, you will learn a lightweight design which improves the efficiency of the ECDH-based AKA protocol.

Dependencies

- Python 3
- 2 VM (clone one VM from your current VM)

1. The YS-ECDH AKA protocol

The YS-ECDH AKA is the elliptic curve version of a variation of the DH key agreement which enables authentication of the parties [1].

In YS-ECDH AKA, initially, A and B share the same ECC parameters (E, G) where E is an elliptic curve and G is the base point. In addition, A and B stores their own long-term ECC public-private key pairs (PK_A, SK_A) and (PK_B, SK_B) and the public key PK_B, PK_A of each other. The protocol has the following steps:

- A generates a random value R_A and computes $U_A = SK_A + R_A$
- B generates a random value R_B and computes $U_B = SK_B + R_B$
- A and B exchange U_A and U_B through the insecure channel
- A calculates $sharedKey = (U_B \cdot G - PK_B) \cdot R_A$
- B calculates $sharedKey = (U_A \cdot G - PK_A) \cdot R_B$
- Now both A and B have the same $sharedKey = R_A \cdot R_B \cdot G$

We provide a prototype for the YS-ECDH AKA protocol. Now, run the prototype between two VMs with the same setting. Pay attention to the time consumed in both parties.

Practice:

(a) Change the ip address in YS-A.py and YS-B.py

(b) Run YS-B.py

(c) Run YS-A.py

You should see the identical shared key is computed on both sides. The time consumed on both sides should be closed.

2. The lightweight version

In ECDH-based AKA, the most time-consuming operation is the scalar multiplication between an integer and a point. The YS-ECDH AKA involves two scalar multiplications on each side in every protocol run. The idea of lightweight design is to transfer one scalar multiplication from the weak side to the more powerful side. Assume A is a limited IoT device and B is a more powerful device, the lightweight version of protocol has the following steps:

- A generates a random value R_A and computes $U_A = SK_A + R_A$
- B generates a random value R_B and computes $U_B = SK_B + R_B, T_B = U_B \cdot G$
- A and B exchange U_A and T_B through the insecure channel
- A calculates $sharedKey = (T_B - PK_B) \cdot R_A$
- B calculates $sharedKey = (U_A \cdot G - PK_A) \cdot R_B$
- Now both A and B have the same $sharedKey = R_A \cdot R_B \cdot G$

In above steps, A executes only one scalar multiplication while B executes three. When B is much powerful than A, the protocol efficiency will be significantly improved as more computations are carried out by a powerful party.

We provide a prototype for the lightweight AKA protocol. Now, run the prototype between two VMs with the same setting. Pay attention to the time consumed in both parties.

Practice:

(a) Change the ip address in LW-A.py and LW-B.py

(b) Run LW-B.py

(c) Run LW-A.py

You should see the identical shared key is computed on both sides. The time consumed on A should be much less than that on B.

Homework:

Can you find any vulnerability of the YS-ECDH AKA and its lightweight version protocol?

Hint: Assume attackers can acquire agreed secrets of old sessions.

Reference

[1] Yacobi, Y. and Shmuely, Z., 1989, August. On key distribution systems. In Conference on the Theory and Application of Cryptology (pp. 344-355). Springer, New York, NY.