[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Chitranshi Srivastava (202051055)

Winter 2022-2023
Lecture 22 (Week 12)

1 Elliptic Curve Digital Signature Algorithm(ECDSA)

Let us first recall that:
RSA-Signature:

- RSA Encryption/Decryption:
  Encryption : c = x^e mod n
  Decryption : x = c^d mod n

- Signature :
  Signature : s = x^d mod n
  Verification : x = s^e mod n

In ECDSA, we have (E, P) as public keys.

Secret Key : a
Public Key : aP

Here, we need

- elliptic curve EC
- a base point-G on the curve. G is such that there exists a large prime number n, such that

$$n \cdot G = 0$$
$$\implies (n-1)G \boxed{+} G = nG$$

Secret Key : d_A
Public Key Q_A : $d_A \cdot G$

Now let us see a scenarios between Alice and Bob:

(E, G, n) is known to everyone

Alice	Bob
secret : d_A public : $Q_A = d_A G$	
messange : m	

1.1 Process of Signature

Let us now see the **Process of Signature**:

1. e = Hash(m)
2. Z→ L_n leftmost bits of e when L_n is the bit length of n
3. K → randomly from [1, n-1]
4. (x_1, y_1) = K·G
5. r = x_1 mod n
   if r = 0 then go to step 3
6. $s = K^{-1}$ [Z+ $r \cdot d_A$] mod n
   if s = 0, then go to step 3
7. Signature (r, s) on message m

1.2 Verification of ECDSA

Let us now see **Verification of ECDSA** performed by Bob:

1. Q_A is not equal to 0
2. Q_A lies on the curve EC or not
3. $n \times Q_A = n \cdot (d_A \cdot G) = d_A \cdot (n \cdot G) = 0$
4. verify r, s ∈ [1, n-1]
5. e = Hash(m)
6. Z → L_n leftmost bits of e
7. u_1 = Z·s^{-1} mod n
   $u_2 = r \cdot s^{-1}$ mod n
8. $(x_2, y_2) = u_1G + u_2Q_A$
   if $(x_2, y_2) = 0$, then signature is invalid. Here addition is addition on the curve.
9. If r ≡ x_2 mod n, then signature is valid, otherwise invalid.

Let us see the proof now:

$$\begin{gathered} \mathrm{c} = u_1G + u_2Q_A \\ \mathrm{c} = u_1G + u_2d_AG \\ \mathrm{c} = (u_1 + u_2d_A)\mathrm{G} \\ \mathrm{c} = (\mathrm{Z}\cdot s^{-1} + rs^{-1}d_A)\mathrm{G} \\ \mathrm{c} = (\mathrm{Z} + rd_A)s^{-1}\mathrm{G} \\ \text{Substituting } s^{-1} \end{gathered}$$

$$\mathrm{c} = (\mathrm{Z} + r \cdot d_A)(K^{-1}(Z \ + \ r \cdot d_A))^{-1}\mathrm{G}$$
$$\mathrm{c} = (\mathrm{Z} + r \cdot d_A)(Z \ + \ r \cdot d_A)^{-1}\mathrm{KG}$$
$$\mathrm{c} = \mathrm{K}{\cdot}\mathrm{G}$$

Hence, proved.

It is not the general form of the Elliptic Curve. In fact, we will not be using this curve for practical purposes.

2 ElGamal Public Key Cryptosystem

It is a public key encryption algorithm like RSA. Unlike RSA, the security of ElGamal Encryption is based on the Discrete Log Problem. The steps below presents a detailed explanation of the encryption and decryption process.

1. Select a prime p.
2. Consider the group $(Z_p^*, *_p)$.

$$Z_p^* = \{1, 2, 3, \ldots, (p-1)\}$$
$$x *_p y = x * y (mod\ p)$$

   If $x \in Z_p^*$, then gcd(x, p) = 1 and hence multiplicative inverse of x under modulo p will exist.

3. Select a primitive element $\alpha \in Z_p^*$, also known as generator of Z_p^*.

$$Z_p^* \text{ is a cyclic group}$$
$$Z_p^* = \langle \alpha \rangle$$

4. The plaintext space is Z_p^* and the key space if $\{(p, \alpha, a, \beta), \beta = \alpha^a \ mod\ p\}$
5. The public key in the algorithm is $\{P, \alpha, \beta\}$ and the secret key is $\{a\}$.
6. Select a random number $x \in Z_{p-1}$. This x is also kept secret.
7. **Encrpytion:** The encryption algorithm produces a tuple as the ciphertext.

$$e_K(m, x) = (y_1, y_2)$$
$$y_1 = \alpha^x \ mod\ p$$
$$y_2 = m \cdot \beta^x \ (mod\ p)$$

8. **Decryption:** The decryption is performed as follows,

$$d_K(y_1, y_2) = y_2 \cdot (y_1^a)^{-1} \ mod\ p = m$$
$$y_1^a = (\alpha^x)^a \ mod\ p = \beta^x \ mod\ p$$
$$y_2 \cdot (y_1^a)^{-1} = m \cdot \beta^x \cdot \beta^{x-1} \ mod\ p = m \ mod\ p$$

   The randomness in the ciphertext is because of the randomly chosen x.

The public is $\{\beta, \alpha, p\}$. Finding a is difficult from β and α (the discrete log problem). But, we know the ciphertext and,

$$y_1 = \alpha^x$$

If we can compute α^{ax} from $\beta = \alpha^a$ and α^x, we are done and we don not need to find a or x and we can find message m. We will be able to break the ElGamal Encryption but however this will not ensure solving the Discrete Log Problem. It is similar to the Diffie Hellman Problem. If you are able to compute g^{ab} from g^a and g^b, then you will break the Diffie Hellman Key Exchange Algorithm, but the Discrete Log Problem will still not be solved.

3 Discrete Logarithm Problem

The Discrete Logarithm Problem states that given a cyclic group G of order n and the generator of G, α and an element $\beta \in G$, find integer x such that $0 \leq x \leq (n-1)$ such that $\alpha^x = \beta$.

If you are asked to compute a, given g and g^a, you can use the exhaustive search by running a loop from i = 1 to i = n, the size of the group (n). Compute g^i, if $g^i = g^a$, we get the result. The complexity of this search will be the size of the group.

There is an algorithm known as Baby-Step Giant-Step Algorithm. It solves the Discrete Log Problem in $\sqrt{n}$ complexity.

3.1 Baby-Step Giant-Step Algorithm

Firstly, we compute $m = \sqrt{n}$. Since, n is order of the group and α is generator of the group, hence $\alpha^n = 1$. Now, let's say $\beta = \alpha^x$, then using Division Algorithm we can write x as,

$$\begin{aligned} x &= i \cdot m + j, \text{ where } 0 \leq i < j \\ \therefore \alpha^x &= \alpha^{i \cdot m} \cdot \alpha^j \\ \implies \beta &= \alpha^{i \cdot m} \cdot \alpha^j \end{aligned}$$

Taking $\alpha^{i \cdot m}$ to the right side,

$$\begin{aligned} \alpha^j &= \beta(\alpha^{im})^{-1} \\ \alpha^j &= \beta(\alpha^{-m})^i \end{aligned}$$

Now, instead of finding x, we have to find i and j. The size of i and j is equal to $m = \sqrt{n}$. Now, we need to find i and j in such a way that the complexity should not multiply.

Let us now define the algorithm formally. The input to the algorithm is the generator α of the cyclic group G, order of group G (n), and $\beta \in G$. The output is the discrete log $x = \log_\alpha \beta$. The steps are written below.

1. Set $m \leftarrow \lceil \sqrt{n} \rceil$
2. Prepare a table T with entries $j, \alpha^j \, 0 \leq j < m$. Sort the table T by α^j values.
3. Compute α^{-m} and set $\gamma \leftarrow \beta$
4. for i = 0 to i = (m-1), do:
    - Check if γ is second component of some entry in T.

- If $\gamma = \alpha^j$, then compute $x = i \cdot m + j$
- Set $\gamma = \gamma \cdot \alpha^{-m}$

The table can be prepared offline and requires a $O(\sqrt{n})$ space. The number of multiplications involved during the runtime of algorithm are $O(\sqrt{n})$. The time taken to sort the table is $O(\sqrt{n} \cdot log(\sqrt{n})) \implies O(\sqrt{n} \cdot log(n))$

4 Kerberos (Version 4)

Kerberos is a computer network security protocol that authenticates service requests between two or more trusted hosts across an untrusted network, like the internet. It uses secret-key cryptography and a trusted third party for authenticating client-server applications and verifying users' identities.

The third parties involved in Kerberos are:

- Ticket Generating Server (TGS)
- Authentication Server (AS)
- Verifier (V)

Using these third parties, the client C is verified. The course of communication that takes place in authentication is described below.

1. Client will start the communication when he/she will log into the server. The client will send the following details to the Authentication Server.

$$C \rightarrow AS : ID_c \mid\mid ID_{TGS} \mid\mid TS_1$$
$$ID_c \rightarrow \text{Identity of Client}$$
$$ID_{TGS} \rightarrow \text{Identity of TGS}$$
$$TS_1 \rightarrow \text{Timestamp}$$

2. The AS will receive the information and send an encrypted message back to the client with the following information. AS performs symmetric key encryption using the key SK_c, which is shared between the AS and the client.

$$AS \rightarrow C : E(SK_c, [SK_{c,TGS} \mid\mid ID_{TGS} \mid\mid TS_2 \mid\mid Lifetime_2 \mid\mid Ticket_{TGS}])$$
$$SK_{c,TGS} \rightarrow \text{key used for communication between Client and TGS}$$
$$Lifetime_2 \rightarrow \text{for how long this ticket/data will be valid}$$

ID_{TGS} and TS_2 are ID of TGS and timestamp as earlier. The ticket is generated by AS and contains the following information.

$$Ticket_{TGS} = E(SK_{TGS}, [SK_{c,TGS} \mid\mid ID_c \mid\mid AD_c \mid\mid ID_{TGS} \mid\mid Lifetime_2])$$
$$AD_c \rightarrow \text{Address of client}$$

As it can be seen the $Ticket_{TGS}$ is encrypted using the key SK_{TGS}. It is the key used for communication between AS and TGS and it is not known to any other party.
The client will receive the response from the AS and will be able to decrypt it as it was encrypted using SK_c, which is shared between client and AS. The client will recieve the following information:

$$[SK_{c,TGS},\ ID_{TGS},\ TS_2,\ Lifetime_2,\ Ticket_{TGS}]$$

The client will receive the ticket but will not be able to decrypt it as client doesn't have SK_{TGS}. The client will also receive the key $SK_{c,TGS}$ which is the shared secret key between client and TGS. This secret key is also called as session key. When you establish a session in server, you get a session key for encrypted communication.

3. Now, since the client has session key, it will initiate communication with the TGS.

$$C \rightarrow TGS : ID_v \ ||\ Ticket_{TGS} \ ||\ Authenticator_c$$
$$Authenticator_c = E(SK_{c,TGS}, [ID_c \ ||\ AD_c \ ||\ TS_3])$$

The TGS will receive the ticket and decrypt it (as TGS has SK_{TGS}) to get the session key $SK_{c,TGS}$. From the ticket TGS will also receive the ID_c and AD_c (sent by AS, as ticket was generated by AS). Using the session key $SK_{c,TGS}$, TGS will decrypt the $Authenticator_c$ to get ID_c and AD_c (sent by Client). If the identity and address sent by AS and client are matching, then the client is verified to TGS.

4. The TGS now sends information back to the client.

$$TGS \rightarrow C : E(SK_{c,TGS}, [SK_{c,v} \ ||\ ID_v \ ||\ TS_4 \ ||\ Ticket_v])$$

The key $SK_{c,v}$ is session key between client and verifier. The $Ticket_v$ contains the following information.

$$Ticket_v = E(SK_v, [SK_{c,v} \ ||\ ID_c \ ||\ AD_c \ ||\ ID_v \ ||\ TS_4 \ ||\ Lifetime_4])$$

The key SK_v is used for communication between TGS and verifier and it is not known to any third party. Hence, $Ticket_v$ can not be decrypted at client side. The client receive the information from the TGS and decrypts it using the $SK_{c,TGS}$. The client gets the following data:

$$[SK_{c,v},\ ID_v,\ TS_4,\ Ticket_v]$$

5. The client sends the following information to the Verifier.

$$C \rightarrow V :\ Ticket_v \ ||\ Authenticator_c$$

The client will send a fresh autenticator to the verifier which has the following data:

$$Authenticator_c = E(SK_{c,v}, [ID_c \ ||\ AD_c \ ||\ TS_5])$$

Verifier will decrypt the data sent by the client and then it will decrypt the $Ticket_v$ to get $SK_{c,v}, ID_c, AD_c, ID_v, TS_4)$ and $Lifetime_4$. Using $SK_{c,v}$, verifier will decrypt the $Authenticator_c$ and will be able to check if it is coming from correct client.

6. The verifier will return the following tho the client.

$$V \rightarrow C :\ E(SK_{c,v}, [TS_5 + 1])$$

We have written here $TS_5 + 1$ but it can be any function of TS_5. The client will be able to verify the timestamp after decrypting the received information using $SK_{c,v}$. He/She will get $TS_5 + 1$. The client is authenticated and verified.