[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024

Scribed by: Dipean Dasgupta(202151188) — Lecture 19,20(Week 11)

1 Elliptic Curve Digital Signature Algorithm

ECDSA consists of Elliptic Curve (EC), point at infinity P =Θ, and Base Point G. The Base location is a location on the EC that has a big prime integer n such that $n \cdot G = \Theta$.

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

This implies that $(n-1) \cdot G$ and G have same x co-ordinate.

Assume Alice needs to perform a digital signature, and Bob needs to confirm Alice's signature. Alice will select an integer d_A and calculate $Q_A = d_A \cdot G$. Alice's public key will be the point Q_A, while her secret key will be the integer d_A. Alice must perform a digital signature on message m.

Steps to Perform Digital Signature using ECDSA

- Compute e = Hash(m)
- Compute Z, where Z is the L_n leftmost bits of e, where L_n is the bit length of n.
- Choose a random integer K such that $1 \leq K \leq (n-1)$
- Compute $(x_1, y_1) = K \cdot G$
- Compute $r = x_1 \; mod \; n$. If r = 0, go to step 3.
- Compute $S = K^{-1} \cdot [Z + r \cdot d_A] \; mod \; n$. If S = 0, go to step 3.
- (r, S) is signature on message m.

Steps to perform Verification of the Signature

- Check if Q_A is not equal to Θ. If $Q_A = d_A \dot{G} = \Theta$, then by the definition of G, $d_A = n$.
- Check if Q_A lies on the EC
- Check if $n \cdot Q_A = \Theta$ or not. It should be equal to Θ because,

$$n \cdot Q_A = n \cdot (d_A \cdot G) = d_A \cdot (n \cdot G) = d_A \cdot \Theta = \Theta$$

- Check if $1 \leq r, \; s \leq (n-1)$
- Compute e = Hash(m)
- Compute Z, where Z is the L_n leftmost bits of e, where L_n is the bit length of n.
- Compute $u_1 = Z \cdot S^{-1} \; mod \; n$ and $u_2 = r \cdot S^{-1} \; mod \; n$
- Compute $(x_2, y_2) = (u_1 \cdot G + u_2 \cdot Q_A)$. If $(x_2, y_2) = \Theta$, then signature is invalid.

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

ElGamal is a public-key encryption technique similar to RSA. Unlike RSA, ElGamal Encryption relies on the Discrete Log Problem to ensure its security. The following steps provide a full explanation of the encryption and decryption process.

- Select a prime p.
- 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.

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

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

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

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

- **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. We don't need to locate a or x and can find message m. We will be able to crack the ElGamal Encryption, however this will not solve the Discrete Log Problem. It is analogous to the Diffie-Hellman Problem. If you can compute g^{ab} from g^a and g^b, you will break the Diffie Hellman Key Exchange Algorithm. However, the Discrete Log Problem will remain unsolved.

3 Discrete Logarithm Problem

The Discrete Logarithm Problem asserts 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 $0 \leq x \leq (n-1)$ so that $\alpha^x = \beta$.

To compute a given g and g^a, utilize the exhaustive search by looping from i = 1 to i = n, where n is the group size. Calculate g^i; if $g^i = g^a$, we obtain the result. This search's difficulty will be determined by the size of the group.

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

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

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

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

Instead of finding x, we must discover i and j. The dimensions of i and j are equivalent to $m = \sqrt{n}$. Now we need to discover i and j so that the complexity does 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 takes up $O(\sqrt{n})$ space. The algorithm performs $O(\sqrt{n})$ multiplications during its runtime. Sorting the table takes $O(\sqrt{n} \cdot log(\sqrt{n}))$, which implies $O(\sqrt{n} \cdot log(n))$.

4 Kerberos (Version 4)

Kerberos is a computer network security protocol for authenticating service requests between two or more trusted hosts over an untrusted network, such as the internet. It employs secret-key cryptography and a trusted third party to authenticate client-server applications and validate user identities.

The third parties involved in Kerberos are:

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

The client C is validated through these third parties. The communication flow that occurs during authentication is explained below:

- The communication will begin when the client logs onto the server. The client will submit the following information to the authentication server.

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

- The AS will receive the information and return an encrypted message to the client containing the following information. The AS encrypts symmetric keys with the shared key SK_c.

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

  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 shown, the $Ticket_{TGS}$ is encrypted with the key SK_{TGS}. It is the communication key between AS and TGS, and no other party knows it.
  The client will receive the response from the AS and be able to decrypt it because it was encrypted using SK_c, which is shared between client and AS. The client will receive the following information:

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

  The client will receive the ticket but will be unable to decrypt it since they do not have SK_{TGS}. The client will also get the key $SK_{c,TGS}$, which is the shared secret key between the client and TGS. This secret key is also known as a session key. When you start a session on the server, you receive a session key for encrypted communication.

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

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

  The TGS will receive the ticket and decrypt it (because it possesses SK_{TGS}) to obtain the session key $SK_{c,TGS}. TGS will receive theID_c$ and AD_c (sent by AS, as the ticket was generated by AS). Using the session key $SK_{c,TGS}$, TGS decrypts the $Authenticator_c$ to obtain ID_c and AD_c (provided by Client). If the identities and addresses given by AS and client match, the client is authenticated using TGS.

- The TGS now sends information back to the client.

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

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

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

  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:

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

- The client sends the following information to the Verifier.

$$C \to V :\ Ticket_v \mid\mid Authenticator_c$$

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

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

  The Verifier will decrypt the data given by the client and then decrypt the $Ticket_v$ to obtain $SK_{c,v}, ID_c, AD_c, ID_v, TS_4$) and $Lifetime_4$. Using $SK_{c,v}$, the verifier will decrypt the $Authenticator_c$ and determine if it comes from the proper client.

- The verifier will return the following tho the client.

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

  The expression $TS_5 + 1$ can represent any function of TS_5. The client can validate the timestamp after decrypting the received information with $SK_{c,v}$. He/she will receive TS_5+1. The client is authenticated and validated.

5 Signal Protocol

Signal Protocol is used for end-to-end encryption. WhatsApp was the first application to use the Signal protocol. End-to-end encryption indicates that you want to communicate data in such a way that even if you transfer it to a public server, the server will have no knowledge of the data. The name implies that there must be an encryption algorithm involved, which is utilized in such a way that the server is unable to decrypt the data. That is, you can consider utilizing a symmetric key encryption algorithm to encrypt large amounts of data. The key is established between the two participants, and the server knows nothing about it. Key establishment entails key exchange (ECDH in this case) between the two parties. Every data is routed exclusively through the server.

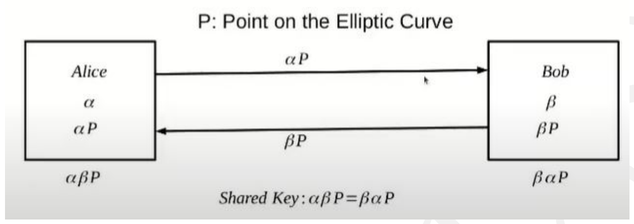


Figure 1: Classical Diffie-Hellman on Elliptic Curve

5.1 Authenticated Encryption with Associated Data (AD)

- Associated Data(AD) is authenticated but not encrypted
- Schemes are nonce-based(and deterministic)

Sender : C = Enc(K, N, AD, M)

K = key
N = Nonce(random number to be only used once)
AD = Associated Data
M = Message

$$\textbf{Sender} \xrightarrow{\text{N, AD, C}} \textbf{Receiver}$$

Receiver : M = Dec(K, N, AD, C)

We are sending AD along with the message over the server but AD is not encrypted. If the AD received by the receiver is correct, only then my decryption is valid. So, we can say that AD is authenticating the source of the message.

5.2 Details about Signal Protocol

- Generation of shared secret key
- Ensuring End-to-End Encryption

- Secure Transfer of data from one user to another via Server
- Ensuring Forward Secrecy(For example: Even if the secrecy of 10th message is compromised, the previous communication will still be secure). In signal protocol, suppose secrecy of mth message is compromised, still 1 to m-1 messages are secure. Also, messages from m+1 are also secure.

This protocol is used in several apps like whatsapp, signal, facebook messenger, skype.
It is an open source protocol: https://signal.org/

5.3 Main Cryptographic modules of Signal Protocol

- X3DH(Extended Triple Diffie-Hellman)
- ECDSA
- Double Ratchet

X3DH is used to generate the shared keys. ECDSA is used to sign the public keys so that no one else can produce valid public key on behalf of me. Finally Double Ratchet performs the encryption and different keys are generated for every message.
Note: Curve X25519 or X448 is used in ECDH and ECDSA.

5.3.1 Registration

Alice registers himself to the Server. He has to provide the id to register. Like, for whatsapp, phone number is the id and OTP is used to verify it.

$$\begin{tikzcd}\textbf{Alice} \arrow[r, "\text{Registration}"] & \textbf{Server}\end{tikzcd}$$

During this process, a packet of keys are generated. Let us say that Alice generates the $Packet_A$ which consists of the following:

- **Identity key :** $IK_A \rightarrow (n_{IA}, n_{IA}P)$
  Updated once to the server - Long term
- **Signed prekey :** $SPK_A \rightarrow (n_{SPA}, n_{SPA}P)$
  Updated weekly or once in a month
- **Prekey Signature : Sig(**IK_A, $Encode(SPK_A)$**)**
  Updated weekly or once in a month
  Note: $n_{SPA}P$ is signed using secret key component of IK_A
- **A set of one-time prekeys :** $OPK^1{}_A$, $OPK^2{}_A$, $OPK^3{}_A$
  (Optional) Uploaded when the server requests for it

This packet is then uploaded to the server.
Alice then sends $Packet_A$ to the Server.

$$\begin{tikzcd}\textbf{Alice} \arrow[r, "Packet_A"] & \textbf{Server}\end{tikzcd}$$

Similarly, Bob will do the same thing.

$$\begin{tikzcd}\textbf{Bob} \arrow[r, "\text{Registration}"] & \textbf{Server}\end{tikzcd}$$

Bob generates the $Packet_B$ which consists of the following:

- **Identity key :** $IK_B \rightarrow \ (n_{IB}, n_{IB}P)$
  Updated once to the server - Long term
- **Signed prekey :** $SPK_B \rightarrow \ (n_{SPB}, n_{SPB}P)$
  Updated weekly or once in a month
- **Prekey Signature : Sig**$(IK_B,\ Encode(SPK_B))$
  Updated weekly or once in a month
  Note: $n_{SPB}P$ is signed using secret key component of IK_B
- **A set of one-time prekeys :** $OPK^1{}_B,\ OPK^2{}_B,\ OPK^3{}_B$
  (Optional) Uploaded when the server requests for it

This packet is then uploaded to the server.

$$\begin{tikzcd}\textbf{Bob} \arrow[r, "Packet_B"] & \textbf{Server}\end{tikzcd}$$

Server stores both the packets.

Alice	**Server**	**Bob**
PacketA	PacketA, PacketB	PacketB

$$\begin{tikzcd}\textbf{Alice} & \textbf{Server} \arrow[l, "\text{PacketB}"']\end{tikzcd}$$

$$\textbf{Alice} \overset{\text{Verify?}}{=} \text{Signature of PacketB}$$

Alice	**Server**	**Bob**
PacketA, PacketB	PacketA, PacketB	PacketB

Alice verifies the signature using the public key of Bob($n_{IB}P$. Now Alice has all public components of Bob's packet($IK_B,\ SPK_B$) and her packet($IK_A,\ SPK_A$). Now she performs the Diffie-Hellman key exchange.

1. Alice generates an ephemeral key $EK_A = n_{EA},\ n_{EA}P$

2. $DH_1 = \mathrm{DH}(IK_A,\ SPK_B)$
   DH is performed using $n_{IA},\ n_{SPB}P$

3. $DH_2 = \mathrm{DH}(EK_A,\ IK_B)$
   DH is performed using $n_{EA},\ n_{IB}P$

4. $DH_3 = \mathrm{DH}(EK_A,\ SPK_B)$
   DH is performed using $n_{EA},\ n_{SPB}P$

5. If key bundle contains one-time prekey,

   - $DH_4 = \mathrm{DH}(EK_A,\ OPK_B)$
   - $SK_A = \mathrm{KDF}(DH_1 \mid\mid DH_2 \mid\mid DH_3)$

6. Else $SK_A = \mathrm{KDF}(DH_1 \mid\mid DH_2 \mid\mid DH_3)$

Here, KDF = SHA-256. Now we are done with step 1, that is, X3DH. Let us move forward.

1. Alice computes associated data
   AD = Encode(IK_A) || Encode(IK_B)
   (Only public parts are used here)

2. Alice encrypts an initial message using AD, SK_A and generates the corresponding initial ciphertext.

$$\begin{tikzcd}
\textbf{Alice} \arrow[r, "\text{initial ciphertext, } IK_A,\ EK_A\text{, Bob's Prekeys}"] & \textbf{Server} \\
\textbf{Server} \arrow[r, "\text{initial ciphertext, } IK_A,\ EK_A\text{, Bob's Prekeys}"] & \textbf{Bob}
\end{tikzcd}$$

Now, let us see what happens on Bob's side.

1. Bob recovers IK_A, EK_A first.

2. Bob performs DH(same as Alice) and KDF by using his secret keys.
   $SK_B = \mathrm{KDF}(DH_1 \mid\mid DH_2 \mid\mid DH_3)$

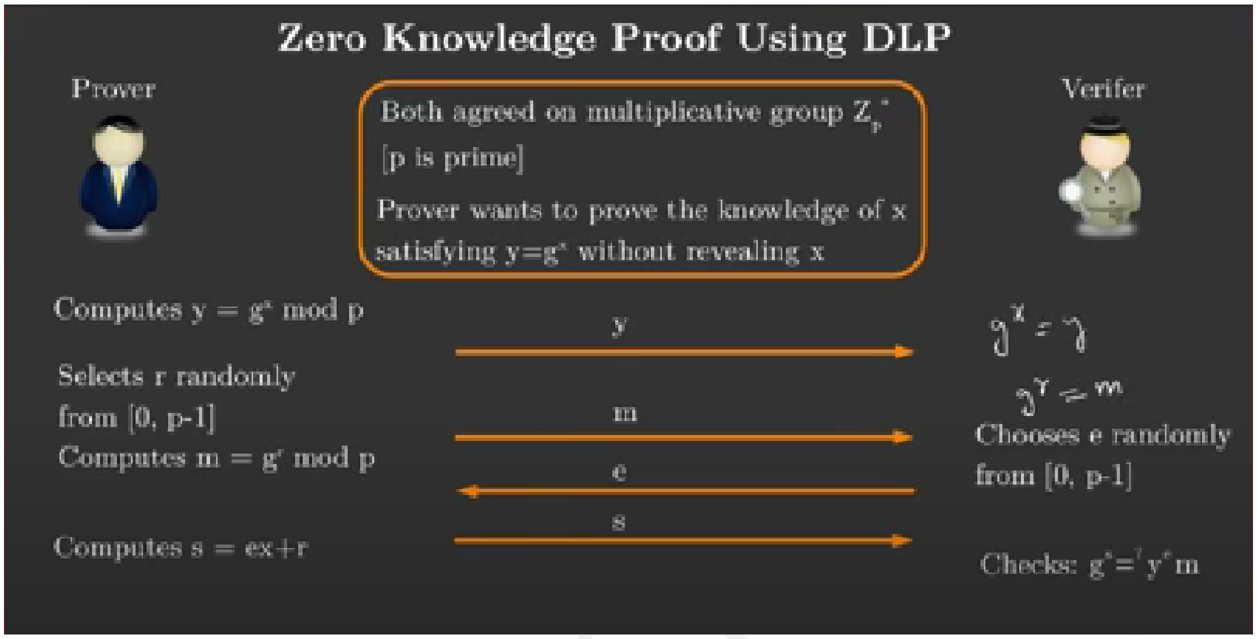
3. Due to properties of DH and KDF, $SK_B = SK_A$.

4. Constructs AD = Encode(IK_A) || Encode(IK_B) (public parts)

5. Decrypts the initial ciphertext using SK_B), AD and recovers the original message.

6 Zero Knowledge Proof using DLP

Suppose I have certain document and I want to prove that I am the owner of it without sharing the document.



Zero Knowledge Proof Using DLP

Prover

Verifer

Both agreed on multiplicative group Z_p^* [p is prime]

Prover wants to prove the knowledge of x satisfying $y=g^x$ without revealing x

Computes $y = g^x \bmod p$ $\xrightarrow{y}$ $g^x = y$

Selects r randomly from [0, p-1]

Computes $m = g^r \bmod p$ $\xrightarrow{m}$ $g^r = m$

$\xleftarrow{e}$ Chooses e randomly from [0, p-1]

Computes $s = ex+r$ $\xrightarrow{s}$ Checks: $g^s \overset{?}{=} y^e m$

$$g^s = g^{ex+r} = g^{ex} \cdot g^r = y^e m$$