[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Raghav Agiwal (202151124)

Winter 2023-2024
Lecture (Week 11)

1 DLP (Discrete Logarithm Problem)

- **Definition**: The Discrete Logarithm Problem (DLP) states that given a cyclic group G of order n and a generator α of G, along with an element $\beta \in G$, the goal is to find an integer x such that $0 \leq x \leq (n-1)$ and $\alpha^x = \beta$.
- **Cyclic Group**: A cyclic group is a group that can be generated by a single element, called a generator. In this context, G is the cyclic group, α is the generator of G, and β is an element of G.
- **Exhaustive Search**: One way to solve the DLP is through exhaustive search. Given α, β, and n, you start with $x = 1$ and iterate up to $x = n$, checking each time if α^x equals β. Once you find a match, you have found the value of x. However, this approach has a complexity equal to the size of the group (n), which can be inefficient for large groups.
- **Baby-Step Giant-Step Algorithm**: This algorithm provides a more efficient solution to the DLP. It reduces the complexity from n to approximately $\sqrt{n}$. The algorithm works by precomputing a table of α^i for $i = 0$ to $\lfloor\sqrt{n}\rfloor$ (the "baby steps"), and then computing $\beta \cdot (\alpha^{-\sqrt{n}})^j$ for $j = 0$ to $\lfloor\sqrt{n}\rfloor$ (the "giant steps"). By comparing these values, the algorithm can find the discrete logarithm x.

1.1 Baby-Step Giant-Step Algorithm

The Baby-Step Giant-Step Algorithm is a method used to solve the Discrete Logarithm Problem (DLP) in a cyclic group. It provides a more efficient solution compared to the brute-force approach by reducing the time complexity from $O(n)$ to approximately $O(\sqrt{n})$, where n is the order of the cyclic group.

1.1.1 Algorithm Overview:

1. **Precomputation (Baby Steps):**
   - Generate a table of baby steps by computing and storing the values of α^i for $i = 0, 1, 2, ..., \lfloor\sqrt{n}\rfloor$, where α is the generator of the cyclic group and n is the order of the group.
2. **Computation (Giant Steps):**
   - For each $j = 0, 1, 2, ..., \lfloor\sqrt{n}\rfloor$, compute the giant step: $\beta \cdot (\alpha^{-\sqrt{n}})^j$, where β is the given element of the cyclic group for which we want to find the discrete logarithm.
3. **Search for Match:**

- Compare the values obtained from the giant steps with the precomputed values from the baby steps.
- If a match is found, i.e., if $\alpha^i = \beta \cdot (\alpha^{-\sqrt{n}})^j$, then the discrete logarithm x is given by $x = i + j \cdot \sqrt{n}$.

1.1.2 Detailed Explanation:

1. **Precomputation (Baby Steps):**
   - In this step, we compute a table of baby steps, which consists of powers of the generator α.
   - We compute and store α^i for $i = 0, 1, 2, ..., \lfloor\sqrt{n}\rfloor$ in a table.
   - This step requires $O(\sqrt{n})$ computations and memory to store the table.
2. **Computation (Giant Steps):**
   - For each $j = 0, 1, 2, ..., \lfloor\sqrt{n}\rfloor$, we compute the giant step.
   - The giant step is computed as $\beta \cdot (\alpha^{-\sqrt{n}})^j$.
   - This step requires $O(\sqrt{n})$ computations.
3. **Search for Match:**
   - We compare the values obtained from the giant steps with the precomputed values from the baby steps.
   - If a match is found, i.e., if $\alpha^i = \beta \cdot (\alpha^{-\sqrt{n}})^j$, then the discrete logarithm x is given by $x = i + j \cdot \sqrt{n}$.
   - If no match is found, we continue iterating through the giant steps until a match is found or we exhaust all possible values.

1.1.3 Complexity Analysis:

The precomputation step requires $O(\sqrt{n})$ computations and memory. The computation step also requires $O(\sqrt{n})$ computations. Therefore, the overall time complexity of the Baby-Step Giant-Step Algorithm is approximately $O(\sqrt{n})$, making it much more efficient than the brute-force approach for large cyclic groups.

1.1.4 Conclusion:

The Baby-Step Giant-Step Algorithm provides an efficient solution to the Discrete Logarithm Problem by reducing the time complexity from $O(n)$ to approximately $O(\sqrt{n})$. It is widely used in various cryptographic applications where efficient solutions to the DLP are required.

2 Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman key exchange protocol allows two parties to establish a shared secret key over an insecure communication channel without directly exchanging the key. It relies on the difficulty of the discrete logarithm problem for its security.

2.1 Algorithm Overview:

1. **Public Parameters Setup:**
   - Parties agree on public parameters:
     - A large prime number p.
     - A primitive root modulo p, denoted as g.
2. **Key Generation:**
   - Each party selects a private key: a for Alice and b for Bob. These are randomly chosen integers from the range $[1, p-1]$.
   - These private keys are kept secret.
3. **Public Key Exchange:**
   - Alice computes her public key $A = g^a \mod p$.
   - Bob computes his public key $B = g^b \mod p$.
   - Public keys A and B are exchanged over the insecure channel.
4. **Secret Key Derivation:**
   - Alice computes the shared secret key $K = B^a \mod p$.
   - Bob computes the shared secret key $K = A^b \mod p$.
   - Both parties derive the same shared secret key K.

2.2 Security:

- The security relies on the difficulty of the discrete logarithm problem, which is the challenge of finding x in the equation $g^x \mod p = y$.
- Without knowledge of a or b, an eavesdropper cannot compute the shared secret key K.

2.3 Application:

The Diffie-Hellman key exchange protocol is used in various cryptographic applications, including secure communication protocols like TLS/SSL, SSH, and VPNs.

2.4 Conclusion:

Diffie-Hellman key exchange enables secure key establishment between parties over an insecure channel. It offers confidentiality without requiring prior communication or shared secrets. The protocol's security is based on the computational complexity of solving the discrete logarithm problem in finite fields.

3 ElGamal Encryption

3.1 Definition:

ElGamal encryption is a public-key cryptosystem that allows two parties to communicate securely over an insecure channel. It's named after its inventor, Taher ElGamal, and it's based on the difficulty of solving the discrete logarithm problem in finite fields.

3.2 Formula:

3.2.1 Key Generation:

1. **Choose Parameters**:
   - Select a large prime number p.
   - Choose a generator g of a multiplicative group G modulo p.
2. **Generate Private Key**:
   - Select a random integer x from the set $\{1, 2, ..., p-2\}$.
3. **Compute Public Key**:
   - Calculate $y = g^x \mod p$.

3.2.2 Encryption:

1. **Message Encoding**:
   - Represent the message m as an element in G (often done by converting it into an integer).
2. **Choose Randomness**:
   - Select a random integer k from the set $\{1, 2, ..., p-2\}$.
3. **Compute Ciphertext Components**:
   - $c_1 = g^k \mod p$
   - $c_2 = m \cdot y^k \mod p$

3.2.3 Decryption:

1. **Compute Shared Secret**:
   - Calculate $s = c_1^x \mod p$.
2. **Decrypt**:
   - Compute $m = c_2 \cdot s^{-1} \mod p$, where s^{-1} is the modular multiplicative inverse of s modulo p.

3.3 Theory:

- **Discrete Logarithm Problem (DLP)**:
  - ElGamal encryption relies on the computational difficulty of solving the discrete logarithm problem.
- **Security**:
  - The security of ElGamal encryption depends on the size of the prime modulus p and the choice of generator g.

- **Applications**:
  - ElGamal encryption is used in various cryptographic protocols such as digital signatures, key exchange, and secure communication over insecure channels.

4 Kerberos

Kerberos stands as a computer network security protocol designed to validate service requests amid multiple trusted hosts over an untrusted network, such as the internet. Employing secret-key cryptography and a dependable third party, it ensures the authentication of client-server applications and verifies users' identities. The Kerberos framework engages various entities in its operations, fostering secure interactions within network environments.

- 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.

4.1 Kerberos Authentication Protocol Communication Flow:

Client Authentication Request to AS (Authentication Server):

The client initiates the authentication process by sending its identity (IDc), the identity of the Ticket Granting Server (TGS) (IDTGS), and a timestamp (TS1) to the AS. This request is encrypted using a shared secret key (SKc) between the client and AS. **Formula:**

$$C \rightarrow AS : \mathrm{E}(SKc, [IDc \mid\mid IDTGS \mid\mid TS1])$$

4.2 AS Response to Client (AS to C):

The AS receives the client's request and generates a session key (SKc,TGS) for further communication between the client and TGS. It then constructs a ticket granting ticket (TGT) containing relevant information such as the client's identity, the TGS's identity, a timestamp (TS2), and a validity period (Lifetime2). This information is encrypted using the shared secret key (SKc) and sent back to the client. **Formula:**

$$AS \rightarrow C : \mathrm{E}(SKc, [SKc, TGS \mid\mid IDTGS \mid\mid TS2 \mid\mid Lifetime2 \mid\mid TicketTGS])$$

Client Decrypts AS Response and Obtains TGT: The client decrypts the response received from the AS using its shared secret key (SKc) and extracts the session key (SKc,TGS) and the TGT. **Formula:**

$$[SKc, TGS,\ IDTGS,\ TS2,\ Lifetime2,\ TicketTGS]$$

4.2.1 Client Authentication Request to TGS (Ticket Granting Server):

The client requests access to a specific service by sending its identity (IDv), the TGT obtained from the AS, and an authenticator encrypted with the session key (SKc,TGS) to the TGS. **Formula:**

$$C \rightarrow TGS : \mathrm{E}(SKc, TGS, [IDv \mid\mid TicketTGS \mid\mid Authenticatorc])$$

4.2.2 TGS Response to Client (TGS to C):

The TGS receives the client's request, decrypts the TGT using its own secret key (SKTGS), and validates the client's identity. If the client is authenticated, the TGS issues a service ticket (Ticketv) containing relevant information such as the service's identity, a timestamp (TS4), and a validity period. This information is encrypted using the session key (SKc,v) shared between the client and the service. **Formula:**

$$TGS \rightarrow C : \mathrm{E}(SKc, TGS, [SKc, v \mid\mid IDv \mid\mid TS4 \mid\mid Ticketv])$$

Client Decrypts TGS Response and Obtains Service Ticket:

The client decrypts the response received from the TGS using the session key (SKc,TGS) and obtains the session key (SKc,v) and the service ticket (Ticketv). **Formula:**

$$[SKc, v,\ IDv,\ TS4,\ Ticketv]$$

Client Authentication Request to Verifier (V):

The client sends the service ticket (Ticketv) and a fresh authenticator encrypted with the session key (SKc,v) to the verifier (V). **Formula:**

$$C \rightarrow V : \mathrm{E}(SKc, v, [Ticketv \mid\mid Authenticatorc])$$

Verifier Response to Client (V to C):

The verifier decrypts the service ticket (Ticketv) using its own secret key (SKv) and verifies the client's identity using the decrypted authenticator. If the client is authenticated, the verifier sends a response encrypted with the session key (SKc,v) to the client, confirming the authentication. **Formula:**

$$V \rightarrow C : \mathrm{E}(SKc, v, [TS5 + 1])$$

4.3 Conclusion:

By integrating the provided communication flow with relevant formulas, we can better understand the encryption and decryption processes involved in the Kerberos authentication protocol, ensuring the confidentiality and integrity of communication between entities across an untrusted network.