[CS304] Introduction to Cryptography and Network Security

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

Scribed by: Sanidhya Kumar (202151138) — Lecture (Week 12)

1 Introduction to the Signal Protocol

The Signal Protocol stands as a cryptographic framework ensuring end-to-end encryption for confidential communication. It serves as the foundational structure for various widely-used messaging applications such as Signal, WhatsApp, and Facebook Messenger. Below is an elucidation of its operational mechanism:

1. **Key Establishment:** The Signal Protocol initiates with a key establishment phase to facilitate secure communication between two entities, typically denoted as Alice and Bob. It employs a variant of the Diffie-Hellman key exchange known as the Double Ratchet algorithm.

2. **Double Ratchet Algorithm:** This algorithm generates a sequence of ephemeral keys to ensure both forward secrecy and future secrecy. Forward secrecy guarantees that compromising one key does not jeopardize the security of prior or subsequent keys, while future secrecy ensures the continued security of past messages even if long-term keys are compromised.

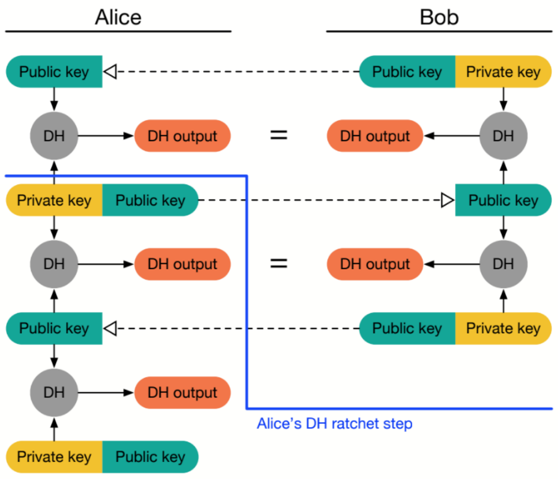

Figure 1: Double Ratchet Algorithm - Alice Side

3. **Session Initialization:** Alice and Bob exchange their long-term identity keys along with one-time prekeys to establish a secure session.

4. **Chain Key:** Each participant maintains a chain key, responsible for deriving fresh keys for every message. This key is updated following the transmission or reception of each message.

5. **Root Key:** The root key, a durable secret shared between Alice and Bob, serves to derive new chain keys and authenticate messages.

6. **Message Encryption:** When Alice intends to transmit a message to Bob, she generates a new message key for encrypting the message. Furthermore, she generates a fresh chain key based on her existing one.

7. **Message Authentication:** A Message Authentication Code (MAC) is utilized to authenticate the message, ensuring its integrity during transit.

8. **Forward and Future Secrecy:** Due to the generation of new keys for each message and the maintenance of a limited key set, compromise of a single key does not compromise the security of past or future messages.

9. **Message Reception:** Upon receiving the encrypted message, Bob decrypts it using his session keys and updates his chain key accordingly.

10. **Asynchronous Communication:** The Signal Protocol accommodates asynchronous communication, allowing message transmission and reception even if one party is offline. Upon reconnection, missed messages can be securely retrieved.

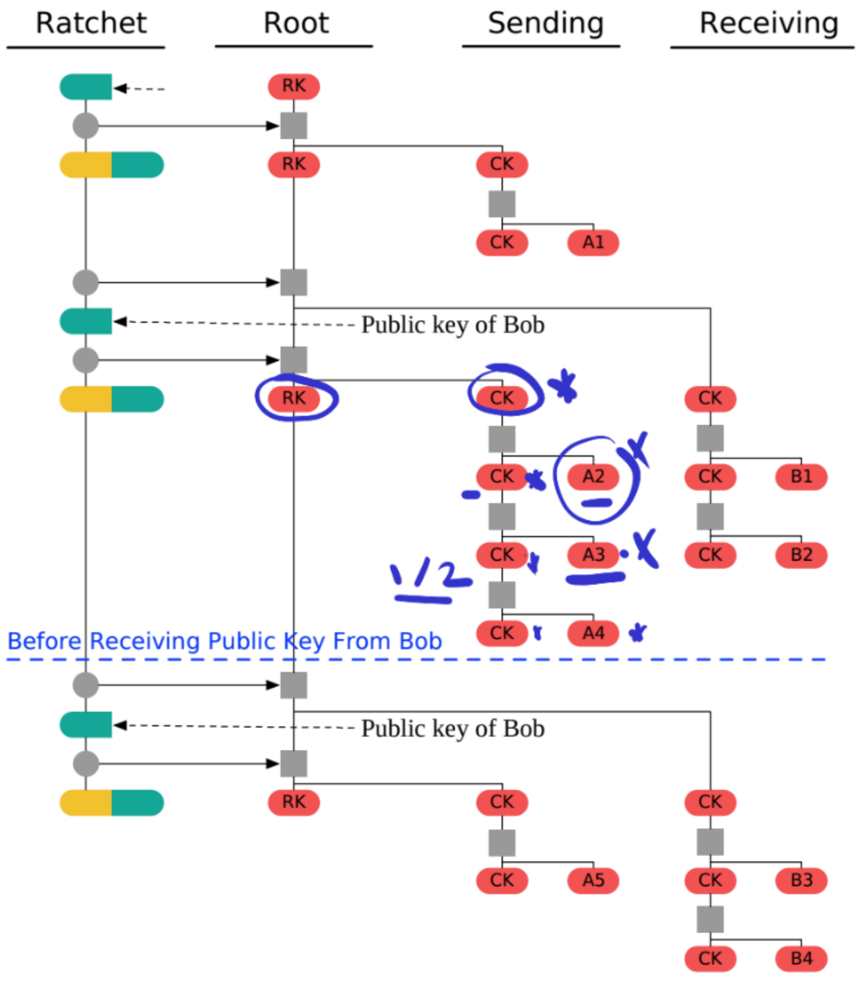

Figure 2: Communication between Alice and Bob

2 Zero Knowledge Proof using Discrete Log Problem

In a Zero Knowledge Proof (ZKP), a cryptographic technique is utilized wherein one party, the prover, can demonstrate to another party, the verifier, the validity of a statement without revealing any additional information except its truthfulness. When employing the Discrete Logarithm Problem, a ZKP allows a party to confirm their familiarity with a discrete logarithm without explicitly disclosing the logarithm itself.

The Discrete Logarithm Problem (DLP) is a fundamental issue in mathematics and cryptography. Given a group G with a generator g and an element h in the group, finding the integer x such that $g^x = h$ is computationally challenging, especially when the group is large and properly selected.

2.1 Zero Knowledge Proof (ZKP)

A ZKP for the DLP involves the following steps:

- **Setup:** The prover and verifier agree on a prime number p, a generator g of cyclic group G of order q (where q is prime), and an element h in G.
- **Commitment:** The prover selects a random integer r and computes $y = g^r \mod p$. Prover sends y to the verifier.
- **Challenge:** The verifier selects a random challenge c (bit).
- **Response:** If $c = 0$, the prover sends r to the verifier. Otherwise, the prover sends $r + c \times x$ to the verifier.
- **Verification:** The verifier checks whether $g^{r+x\times c} = h \times y^c$ holds. If yes, it indicates that the prover knows the discrete logarithm x of h with respect to g, thus validating the ZKP.

2.2 Explanation

- **Soundness:** The ZKP is sound because if the prover does not know the discrete logarithm x, it is computationally infeasible for them to generate a convincing response to the verifier.
- **Completeness:** If the prover indeed knows the discrete logarithm x, they can generate a convincing response that satisfies the verifier with overwhelming probability.
- **Zero-Knowledge Property:** The ZKP does not disclose any information about the discrete logarithm x to the verifier other than the fact that the prover possesses it. This property ensures that the protocol reveals no additional information about the prover's secret.

2.3 Application

- ZKPs find various applications in cryptography and beyond, including authentication protocols, secure multi-party computation, and anonymous cryptocurrencies like Zcash.
- By employing ZKPs for the DLP, parties can prove possession of secret keys or knowledge of specific values without revealing those secrets, thereby enhancing privacy and security in numerous cryptographic protocols.

3 RC4 Stream Cipher

RC4, devised by Ron Rivest in 1987, stands as a symmetric stream cipher renowned for its straightforwardness and rapidity, rendering it prevalent across diverse applications. The acronym "RC" denotes "Rivest Cipher," while it is alternatively recognized as "ARC4" or "ARCFOUR" (reportedly owing to its purported utilization in the initial Netscape SSL protocol, though this claim lacks official substantiation).

3.1 Algorithm

The RC4 algorithm functions through the following steps:

1. **Key Initialization**:
    - RC4 employs a variable-length key, typically ranging from 40 to 2048 bits, to initialize its internal state.
    - Utilizing a key-scheduling algorithm (KSA), the key is expanded into an initial permutation of the internal state array (S-box).
2. **Pseudo-Random Bit Generation**:
    - After completing the key setup, RC4 generates a stream of pseudo-random bits or bytes.
    - This generation process involves iteratively swapping elements of the state array based on the current state and outputting a byte from the array.
3. **XOR with Plaintext/Ciphertext**:
    - The generated pseudo-random stream is XORed with the plaintext to produce ciphertext (or with ciphertext to retrieve plaintext).
    - This XOR operation ensures reversibility of the encryption and decryption processes with the same key.

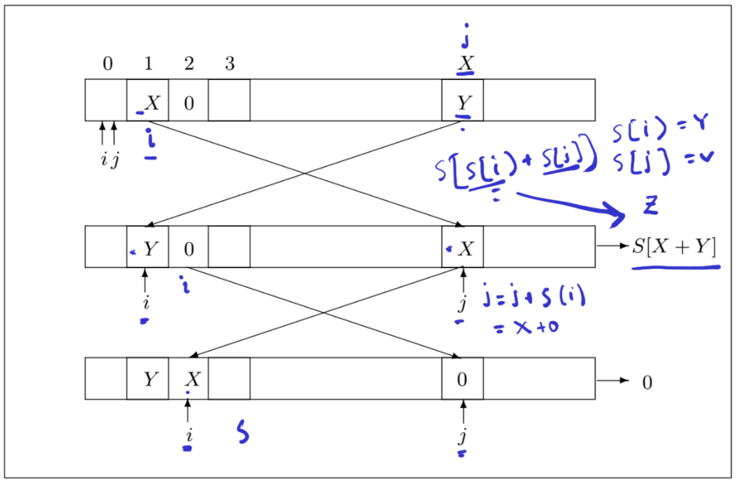

Figure 3: The Biased Second Output of RC4

3.2 Drawbacks

While RC4 enjoyed widespread adoption previously, it has since fallen out of favor for most cryptographic applications due to numerous vulnerabilities uncovered over time. These vulnerabilities comprise biases in the initial stream bytes and weaknesses within the key scheduling algorithm.

Despite its security shortcomings, RC4 persists in application within certain legacy systems and protocols. However, contemporary cryptographic standards, such as AES (Advanced Encryption Standard), are strongly advocated for new implementations owing to their superior security assurances.

4 Secured Sockets Layer (SSL)

Many cryptographic protocols rely on three fundamental cryptographic primitives:

- Symmetric Key Encryption Algorithms: These are favored because computations in Public Key Cryptography are typically more computationally intensive than those in symmetric key cryptography.
- Key Exchange Protocols: These facilitate the establishment of a common shared key between communicating parties. A signature mechanism is often employed to verify the authenticity of the source and the public keys to mitigate attacks such as Man-in-the-Middle attacks.
- Message Authentication Codes (MACs): These are used to authenticate encrypted data, providing assurance that the message originated from the expected party and has not been tampered with.

The SSL protocol is widely implemented, especially in HTTPS. It ensures secure communication between two parties by incorporating key exchange, symmetric key encryption, signature-based authentication of public keys, and Message Authentication Codes to authenticate messages.

The process begins with a connection. A **connection** refers to a transport mechanism that furnishes a suitable type of service. These connections are transient, with each connection being linked to a single session. A **session**, for instance, is established when accessing a HTTPS website, wherein a temporal duration is defined for exchanging data securely. Throughout this period, all data exchanges remain fully encrypted. Sessions are initiated by the Handshake Protocol and define a collection of cryptographic security parameters that can be utilized across multiple connections. They serve to circumvent the resource-intensive renegotiation of new security parameters for each connection.

The state of a session is determined by the following parameters:

1. **Session identifier:** An arbitrary byte sequence designated by the server to identify an active or resumable session state.
2. **Peer certificate:** An X509.v3 certificate of the peer, which may be null. Typically, it represents a signed public key. Acquiring an SSL certificate entails having your public key authenticated by a certified authority using their secret key, thus enabling verification by others that the authority has endorsed it. X509.v3 is an example of such a certified authority.

3. **Compression method:** The algorithm employed to compress data before encryption, accompanied by a corresponding decompression algorithm.

4. **Cipher spec:** This specification outlines the bulk data encryption algorithm (e.g., null, AES), a hash algorithm (e.g., MD5 or SHA-1) utilized for MAC calculation, and the mechanism for key exchange (e.g., Diffie-Hellman, ECDH). Additionally, it defines cryptographic attributes such as the hash size.

5. **Master secret:** A 48-byte secret mutually shared between the client and server. It serves to generate certain keys, including those used for data encryption and MAC generation.

6. **Is resumable:** A flag denoting whether the session is capable of initiating new connections.

The state of a connection is characterized by the following parameters:

1. **Server and client random:** Byte sequences individually chosen by the server and client for each connection. These numbers are utilized in specific computations to thwart certain attacks.

2. **Server write MAC secret:** The secret key employed in MAC operations on data transmitted by the server, shared between the client and server.

3. **Client write MAC secret:** The symmetric key utilized in MAC operations on data transmitted by the client.

4. **Server write key:** The symmetric encryption key utilized for encrypting data by the server and decrypting it by the client.

5. **Client write key:** The symmetric encryption key employed for encrypting data by the client and decrypting it by the server.

6. **Initialization vectors:** In the case of using a block cipher in CBC mode, an initialization vector (IV) is maintained for each key. This field is initially set during the SSL Handshake Protocol. Subsequently, the final ciphertext block from each record is retained for utilization as the IV with the succeeding record.

7. **Sequence numbers:** Each party manages separate sequence numbers for transmitted and received messages within each connection. Upon sending or receiving a "change cipher spec message," the relevant sequence number is reset to zero. Sequence numbers are restricted to a maximum value of $2^{64} - 1$.

4.1 SSL Record Protocol

Within the SSL framework, the initial protocol in operation is the SSL Record Protocol. This protocol furnishes two vital services for SSL connections:

1. **Confidentiality:** As stipulated by the Handshake Protocol, a shared secret key is established for the conventional encryption of SSL payloads.

2. **Message Integrity:** Additionally, the Handshake Protocol delineates a shared secret key employed to construct a message authentication code (MAC), thereby ensuring message integrity.

Given a packet, it is first broken into multiple blocks of fixed size, with the block length depending on the protocol version. Each block undergoes compression, compressing the fragmented data to a fixed length size. The MAC is then generated using the MAC Secret Key on this compressed data. The MAC is concatenated with the compressed data. Subsequently, this concatenated data is encrypted using an encryption algorithm. Finally, a header containing public information (not to be encrypted) is added before the encrypted data. This constitutes the complete packet to be transferred either from client to server or vice versa.

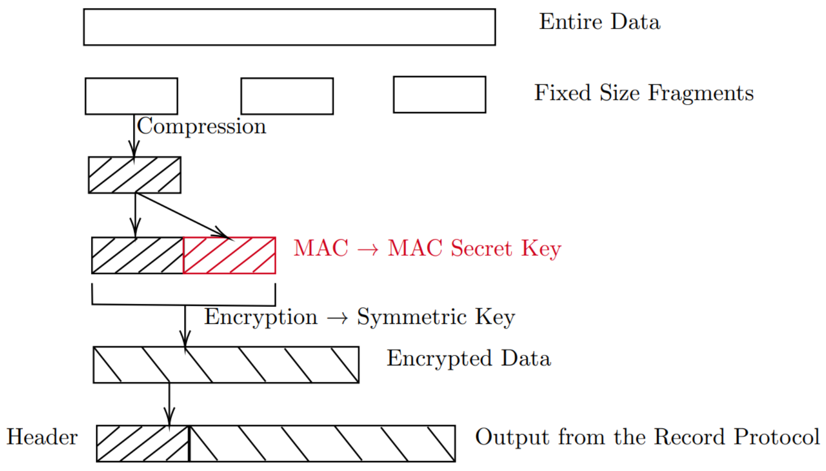

Figure 4: Flowchart for SSL

Suppose we have received data from the server. The length of the header part is known, so it is discarded from the packet, and any necessary information required is extracted from it. Then, decryption is performed, yielding the compressed data concatenated with the MAC. Since the length of the MAC is known, that part is removed, leaving only the compressed data. From this compressed data, a new MAC is generated. If the generated MAC matches the received MAC, the data is authenticated. If the MACs match, decompression is performed to retrieve the original data contained in the packet.

The two keys required are the MAC Secret Key and the encryption key, which are the same at both the client and server sides. The MAC is generated using the formula:

$$\begin{aligned}\text{MAC} :&\text{hash}(\text{MAC\_write\_secret}||\text{pad2}||\\ &\text{hash}(\text{MAC\_write\_secret}||\text{pad1}||\text{seq\_num}||\\ &\text{SSLCompressed.type}||\text{SSLCompressed.length}||\text{SSLCompressed.fragment}))\end{aligned}$$

The list of encryption algorithms supported by SSL is as follows:

AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128

During negotiation between the client and server, they agree upon one encryption algorithm. The header contains the following information, which is public and not required to be encrypted:

- Content Type (8 bits): The type of data.
- Major Version (8 bits): For example, if using version 3 of SSL, the major version will be 3.
- Minor Version (8 bits): For example, if using version 3 of SSL, the minor version will be 0.
- Compressed Length (16 bits): Length of the compressed data.

4.2 Change Cipher Spec Protocol

Another protocol utilized in SSL is the Change Cipher Spec Protocol. Its primary function is to update the cipher suite to be utilized on the current connection. For instance, suppose RC4 is being used for encrypting the data. After a certain period, the client and server may desire to alter the encryption mechanism and transition to AES. The Change Cipher Spec Protocol serves this purpose in such scenarios.

4.3 Alert Protocol

The Alert Protocol serves the primary function of issuing alerts in case of anomalies or errors. Below is a partial list of alerts, along with descriptions of when they are triggered by the Alert Protocol.

- **unexpected_message:** Indicates the receipt of an inappropriate message.
- **bad_record_mac:** Indicates the reception of an incorrect MAC.
- **decompression_failure:** Indicates a failure in the decompression function, such as the inability to decompress or decompressing beyond the maximum allowable length.
- **handshake_failure:** Indicates the inability of the sender to negotiate an acceptable set of security parameters from the available options.
- **illegal_parameter:** Indicates a field in a handshake message that is out of range or inconsistent with other fields.
- **close_notify:** Notifies the recipient that the sender will cease sending any further messages on this connection. Both parties are required to send a close_notify alert before closing the write side of the connection.
- **bad_certificate:** Indicates the receipt of a corrupt certificate, such as one containing a signature that does not verify.
- **unsupported_certificate:** Indicates that the type of the received certificate is not supported.
- **certificate_revoked:** Indicates that a certificate has been revoked by its signer.
- **certificate_expired:** Indicates that a certificate has expired.
- **certificate_unknown:** Indicates that some unspecified issue arose during the processing of the certificate, rendering it unacceptable.

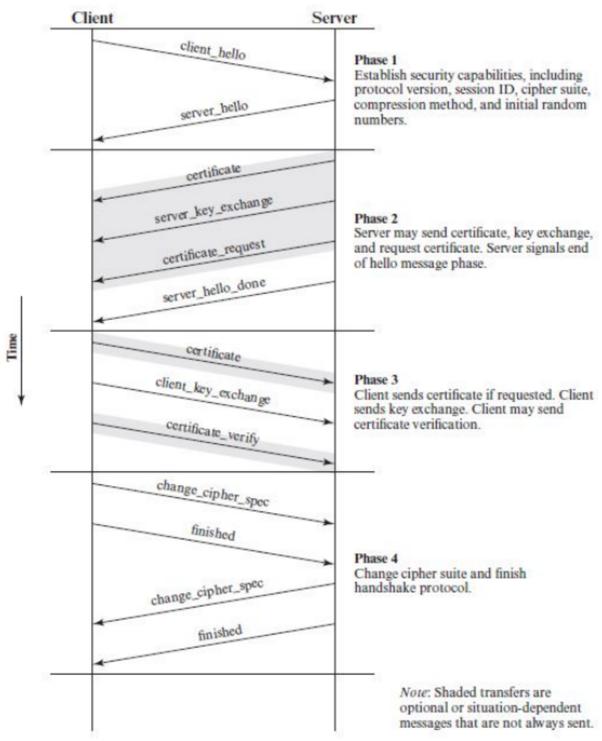
Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Figure 5: Handshake Protocol Flowchart

4.4 Handshake Protocol

Step 1: Initiation and Exchange The Handshake Protocol begins with the client initiating communication by sending a `client_hello` message, followed by the server responding with a `server_hello` message.

Client:

- Sends `client_hello` message containing:
  - Version
  - Random structure
  - Session ID
  - Cipher Suite
  - Compression Method

Server:

- Responds with `server_hello` message mirroring client's parameters.

Step 2: Certificate Exchange and Key Initiation The server sends its certificate (`server_key_exchange`) and may request the client's certificate (`certificate_request`). It then concludes with a hello done message.

Server:

- Sends certificate and initiates key exchange.
- Optionally requests client's certificate.
- Concludes with a hello done message.

Step 3: Client Response and Verification The client responds by sending its certificate, key exchange data (`client_key_exchange`), and, if requested, a certificate verification (`certificate_verify`).

Client:

- Responds with its certificate and key exchange data.
- Optionally sends certificate verification.

Step 4: Cipher Specification and Finish Both parties signal cipher specification compatibility (`change_cipher_spec`) and send finish messages (`finished`).

Client:

- Signals cipher specification compatibility.
- Sends finish message.

Server:

- Signals cipher specification compatibility.
- Sends finish message.

Step 5: Master Secret Key Generation The master secret key is generated using a pre-master secret and random values.

Formula:

$$\text{master\_secret} = \text{MD5}(\text{pre\_master\_secret}||\text{SHA}(A||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$
$$||\text{MD5}(\text{pre\_master\_secret}||\text{SHA}(BB||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$
$$||\text{MD5}(\text{pre\_master\_secret}||\text{SHA}(CCC||\text{pre\_master\_secret}||\text{ClientHello.random}||\text{ServerHello.random}))$$

Step 6: Key Block Generation The master secret is used to generate a key block for encryption, MAC, and initialization vectors.

Formula:

$$\text{key\_block} = \text{MD5}(\text{master\_secret}||\text{SHA}(A||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$
$$||\text{MD5}(\text{master\_secret}||\text{SHA}(BB||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$
$$||\text{MD5}(\text{master\_secret}||\text{SHA}(CCC||\text{master\_secret}||\text{ServerHello.random}||\text{ClientHello.random}))$$

This detailed breakdown outlines the handshake protocol's steps, including key generation and exchange, ensuring secure communication between parties.