Lecture Note 5: Network Security in Practice

Dongbin BAI 23038946R
Yuqin WANG 21068692R

April 4, 2024

In this lecture, practical algorithms that are employed in real-world network security enhancing scenarios are introduced. Before diving into the detailed theories about the algorithms, we would first summarize the Elliptic Curve-based encryption and signature. Then we are going to describe SSL/TLS and HTTPS in details in the following sections in the lecture note.

1 Elliptical Curve Cryptography (ECC)

In previous lectures we saw many applications of the discrete log, CDH, and DDH assumptions in a finite cyclic group $\mathbb{G}$. A primary example for the group $\mathbb{G}$ is the multiplicative group (or subgroup) of integers modulo a sufficiently large prime p. This group is problematic for a number of reasons, most notably because the discrete log problem in this group is not sufficiently difficult. The best known algorithm, called the general number field sieve (GNFS) can run in time $exp(\widetilde{O}(log\ p)^{1/3})$. This algorithm is the reason why, in practice, we must use a prime p whose size is at least 2048 bits. Arithmetic modulo such large primes is slow and greatly increases the cost of deploying cryptosystems that use this group.

Several other families of finite cyclic groups with an apparent hard discrete log have been proposed. Of all these proposals, the group of points of an elliptic curve over a prime finite field is the most suitable for practice, and is widely used on the Internet today. The best known discrete log algorithm in an elliptic curve group of size q runs in time $O(\sqrt{q})$. The group operation could use a small number of arithmetic operations modulo a 256-bit prime, which is considerably faster than arithmetic modulo a 2048-bit, while keeping the same insurance on security.

1.1 Elliptic Curves

Let K be any field. The projective plane $\mathbb{P}^2(K)$ over K is defined as the set of triples

$$(X, Y, Z)$$

where $X, Y, Z \in K$ are not all simultaneously zero. On these triples is defined an equivalence relation

$$(X, Y, Z) \equiv (X^{'}, Y^{'}, Z^{'})$$

if there exists a $\lambda \in K$ such that

$$X = \lambda X', Y = \lambda Y' \text{ and } Z = \lambda Z'.$$

An *elliptic curve* over K will be defined as the set of solutions in the projective plane $\mathbb{P}^2(K)$ of a homogeneous Weierstrass equation of the form

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

with $a_1, a_2, a_3, a_4, a_6 \in K$. This equation is also referred to as the long Weierstrass form. Such a curve should be non-singular in the sense that, if the equation is written in the form $F(X, Y, Z) = 0$, then the partial derivatives of the curve equation should not vanish simultaneously at any point on the curve.

For convenience, we will most often use the affine version of the Weierstrass equation, given by

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{1}$$

where $a_i \in K$. Assume, for the moment, that char $K \neq 2, 3$, and consider the change of variables given by

$$b_2 = a_1^2 + 4a_2$$

$$X = X' - \frac{b_2}{12}$$

$$Y = Y' - \frac{a_1}{2}(X' - \frac{b_2}{12}) - \frac{a_3}{2}$$

This change of variables transforms the long Weierstrass form given in Equation 1 to the equation of an isomorphic curve given in short Weierstrass form,

$$E : Y^2 = X^3 + aX + b$$

for some $a, b \in K$. One can then define a group law on an elliptic curve using the chord-tangent process.

The chord process is defined as follows, see Fig. 1.1 for a diagrammatic description. Let P and Q be two distinct points on E. The straight line joining P and Q must intersect the curve at one further point, say R, since we are intersecting a line with a cubic curve. The point R will also be defined over the same field of definition as the curve and the two points P and Q. If we then reflect R in the x-axis we obtain another point over the same field which we shall call $P + Q$. One can show that the chord-tangent process turns E into an abelian group with the point at infinity $\mathcal{O}$ being the zero.

1.2 Elliptic Curves over Finite Fields

For cryptographic applications we are mostly interested in elliptic curves over finite fields. For simplicity, we only consider elliptic curves denoted over a finite field $\mathbb{F}_p$ where $p > 3$ is a prime.

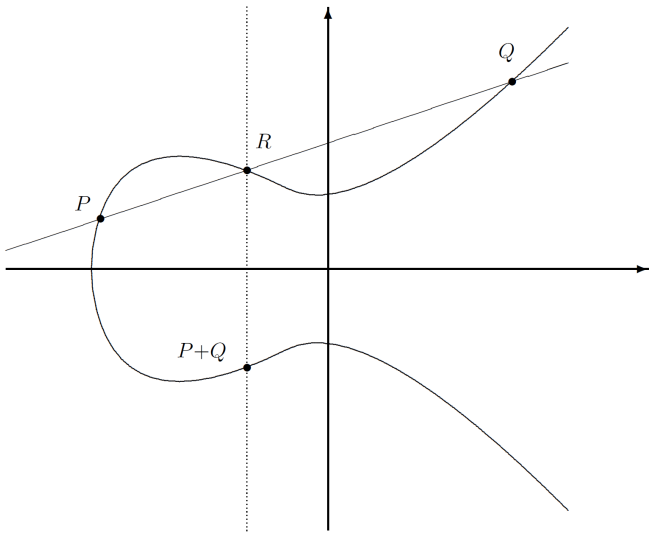

Figure 1: Adding two points on an elliptic curve [S+03]

Definition 1. *Let $p > 3$ be a prime. An **elliptic curve** E defined over $\mathbb{F}_p$ is an equation*

$$y^2 = x^3 + ax + b \tag{2}$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$. We write $E/\mathbb{F}_p$ to denote the fact that E is defined over $\mathbb{F}_p$. [BS23]

The condition $4a^3 + 27b^2 \neq 0$ ensures that the equation $x^3 + ax + b = 0$ does not have a double root. This is needed to avoid certain degeneracies.

As we discussed in the previous section, there is a natural group law denoted on the points of an elliptic curve. The group operation is written additively using the symbol "$\boxplus$" to denote point addition. We define the point at infinity $\mathcal{O}$ to be the identity element: for all $P \in E(\mathbb{F}_{p^e})$ we define $P \boxplus \mathcal{O} = \mathcal{O} \boxplus P = P$.

This addition law makes the set $E(\mathbb{F}_{p^e})$ into a group. The identity element is the point at infinity. Every point $\mathcal{O} \neq P = (x_1, y_1) \in E(\mathbb{F}_{p^e})$ has an additive inverse, namely $-P = (x_1, -y_1)$. Finally, it can be shown that this addition law is associative. The group law is clearly commutative, $P \boxplus Q = Q \boxplus P$ for all $P, Q \in E(\mathbb{F}_{p^e})$, making this an abelian group.

1.3 Elliptic Curves in Practice

1.3.1 Curve P256

Two widely used elliptic curves, called **secp256r1** and **secp256k1**, are specified in a standard called SEC2, where SEC is an acronym for "standards for efficient cryptography." Both curves are defined over a 256-bit prime field, hence the "256" in their names. The 'r' in secp256r1 signifies that the curve is a random curve, meaning that it was generated by a

certain sampling procedure. The curve secp256r1 is widely used in Internet protocols, while secp256k1 is widely used in blockchain systems.

The curve secp256r1. This curve was approved by the U.S. National Institute of Standards (NIST) for federal government use in a standard published in 1999. The NIST standard refers to this curve as **Curve P256**. All implementations of TLS 1.3 are required to support this curve for Diffie-Hellman key exchange. It is the only mandatory curve in the TLS 1.3 standard. The curve secp256r1 is defined as follows:

- The curve is defined over the prime $p_r := 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. The special structure of this prime is meant to improve the performance of arithmetic modulo p_r.
- The curve has the Weierstrass form $y^2 = x^3 - 3x + b$ where $b \in \mathbb{F}_{p_r}$ written as a 255-bit number in hexadecimal is:

  $b :=$ `5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`.

- The number of points on this curve is a prime number. Recall that it must be close to p_r.
- The standard also specifies a point G_r that generates the entire group $E(\mathbb{F}_{p_r})$, where E is the curve secp256r1.

How was the odd looking parameter b selected? The reality is that we do not really know. The standard lists an unexplained constant called a seed S. This seed was provided as input to a public deterministic algorithm, that generated the parameter b. This process was designed to select a random curve that resists the known discrete log attacks. The problem is that we do not know for sure how the seed S was selected. An organization that wants to use secp256r1 might worry that S was chosen adversarially so that discrete log on the resulting curve is easy. Currently we do not know how to select such a seed even if we wanted to, so this concern is just an intriguing speculation. As far as we can tell, secp256r1 is a fine curve to use. It is widely used in Internet protocols.

Security of discrete log on secp256r1. Because the prime p_r is close to 2^{256}, the number of points on the curve is also close to 2^{256}. Therefore, computing discrete log on the curve using a generic discrete log algorithm takes approximately 2^{128} group operations. We assume that no algorithm can compute discrete log much faster than that. The intent is that discrete log on the curve (as well as CDH and DDH on both curves) should be at least as hard as breaking AES-128. Consequently, if one is aiming for the level of security provided by AES-128, then secp256r1 curve can be used for Diffie-Hellman key exchange, public-key encryption, and digital signatures.

1.3.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

Because the Schnorr system was protected by a patent, NIST opted for a more ad-hoc signature scheme based on a prime-order subgroup of $\mathbb{Z}_p^*$ that eventually became known as the Digital Signature Algorithm or DSA. The standard was later updated to support elliptic curve groups defined over a finite field.

The Elliptic Curve Digital Signature Algorithm (ECDSA) signature scheme (G, S, V) uses the group of points $\mathbb{G}$ of an elliptic curve over a finite field $\mathbb{F}_p$. Let g be a generator of $\mathbb{G}$ and let q be the order of the group $\mathbb{G}$, which we assume is prime. We will use multiplicative notation for the group operation. We will also need a hash function H defined over $(\mathcal{M}, \mathbb{Z}_q^*)$.

The scheme works as follows:

- $G()$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_q^*$ and set $u \leftarrow g^\alpha \in \mathbb{G}$. Output $sk := \alpha$ and $pk := u$.
- $S(sk, m)$: To sign a message $m \in \mathcal{M}$ with secret key $sk = \alpha$ do:

repeat:

$\alpha_t \xleftarrow{R} \mathbb{Z}_q^*$, $u_t \leftarrow g^{\alpha_t}$

let $u_t = (x, y) \in \mathbb{G}$ where $x, y \in \mathbb{F}_p$

treat x as an integer in $[0, p)$ and set $r \leftarrow [x]_q \in \mathbb{Z}_q$

$s \leftarrow (H(m) + r\alpha)/\alpha_t \in \mathbb{Z}_q$

until $r \neq 0$ and $s \neq 0$
output $(r, s) \in \mathbb{Z}_q^2$

- $V(pk, m, \sigma)$: To verify a signature $\sigma = (r, s) \in \mathbb{Z}_q^2$ on $m \in \mathcal{M}$ with $pk = u \in \mathbb{G}$ do:

if $r = 0$ or $s = 0$ then output `reject` and stop
$\alpha \leftarrow H(m)/s \in \mathbb{Z}_q$, $b \leftarrow r/s \in \mathbb{Z}_q$
$\hat{u}_t \leftarrow g^a u^b \in \mathbb{G}$
if $\hat{u}_t$ is the point at infinity in $\mathbb{G}$ then output `reject` and stop
let $\hat{u}_t = (\hat{x}, \hat{y}) \in \mathbb{G}$ where $\hat{x}, \hat{y} \in \mathbb{F}_p$
treat $\hat{x}$ as an integer in $[0, p)$ and set $\hat{r} \leftarrow [\hat{x}]_q \in \mathbb{Z}_q$
if $r = \hat{r}$ output `accept`; else output `reject`

When using the elliptic curve P256, both p and q are 256-bit primes. An ECDSA signature $\sigma = (r, s)$ is then 512 bits long.

A straightforward calculation shows that the scheme is correct: for every key pair (pk, sk) output by G, and every message $m \in \mathcal{M}$, if $\sigma \xleftarrow{R} S(sk, m)$ then $V(pk, m, \sigma)$ outputs `accept`. The reason is that $\hat{u}_t$ computed by V is the same as u_t computed by S.

For security, it is important that the random value α_t generated during signing be a fresh uniform value in $\mathbb{Z}_q^*$. Otherwise the scheme can become insecure in a strong sense: an attacker can learn the secret signing key α.

ECDSA is not strongly secure. While the Schnorr signature scheme is strongly secure, the ECDSA scheme is not. Given an ECDSA signature $\sigma = (r, s)$ on a message m, anyone can generate more signatures on m. For example, $\sigma' := (r, -s) \in (\mathbb{Z}_q^*)^2$ is another valid signature on m. This σ' is valid because the x-coordinate of the elliptic curve point $u_t \in \mathbb{G}$ is the same as the x-coordinate of the point $1/u_t \in \mathbb{G}$.

2 Transmission Control Protocol & Internet Protocol

Transmission Control Protocol (TCP) and Internet Protocol (IP) are the collections of protocols that allows the computers to connect with the network and exchange data with other computers over the Network. The TCP/IP model is the foundation of the Internet that was introduced by the U.S. Department of Defence (DoD). TCP/IP protocols are based and implemented on the reference of the OSI Model (Open System Interconnection). OSI Model is the theoretical model that defines some set of protocols that need to be followed by every network for the successful transmission of data over the network. Most networking systems that we encounter today follow the TCP/IP model. TCP/IP Model much simplified practical model that is practically used in the real world. [Int21]

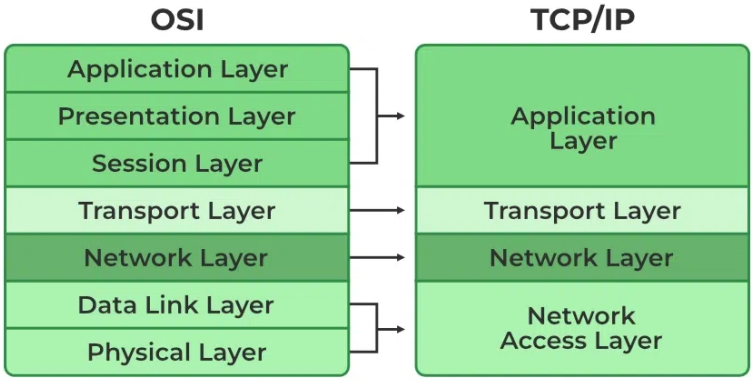

Figure 2: OSI model and TCP/IP model [Gee21]

TCP/IP Model is a practically implemented version of the OSI Model. Although layers are different in TCP/IP Model in comparison with the OSI Model as shown in Fig. 2. The layers are grouped according to the task performed by each layer. It has commonly 4 layers: Application Layer, Transport Layer, Network Layer and Network Interface Layer (or Network Access Layer).

- **Application Layer**: The application layer encompasses interactions between networked applications. When applications necessitate communication with the network, they adhere to the protocols specified within this layer. Numerous protocols are employed by individual applications for transmitting data to or retrieving data from the network.
- **Transport Layer**: The TCP/IP transport layer protocols exchange data receipt acknowledgments and retransmit missing packets to ensure that packets arrive in order and without error. End-to-end communication is referred to as such. Transmission Control Protocol (TCP) and User Datagram Protocol are transport layer protocols at this level (UDP).
  - **TCP**: Applications can interact with one another using TCP as though they were physically connected by a circuit. TCP transmits data in a way that resem-

bles character-by-character transmission rather than separate packets. A starting point that establishes the connection, the whole transmission in byte order, and an ending point that closes the connection make up this transmission.

  - **UDP**: The datagram delivery service is provided by UDP, the other transport layer protocol. Connections between receiving and sending hosts are not verified by UDP. Applications that transport little amounts of data use UDP rather than TCP because it eliminates the processes of establishing and validating connections.

- **Network Layer**: The Network Layer is a layer in the Internet Protocol (IP) suite, which is the set of protocols that define the Internet. The Network Layer is responsible for routing packets of data from one device to another across a network. It does this by assigning each device a unique IP address, which is used to identify the device and determine the route that packets should take to reach it.

- **Network Access Layer**: The Network Access Layer consists of two sublayers called Data Link Layer and Physical Layer. The packet's network protocol type, in this case, TCP/IP, is identified by the data-link layer. Error prevention and "framing" are also provided by the data-link layer. While the Physical Layer is responsible for generating the data and requesting connections. It acts on behalf of the sender and the Network Access layer on behalf of the receiver.

3 Security Socket Layer & Transport Layer Security

In this section, we embark on a comprehensive exploration of Secure Sockets Layer (SSL) and its follow-on Internet standard, Transport Layer Security (TLS), which are foundational protocols that establish a fortified channel for secure data transmission over computer networks. Positioned just above TCP in the networking stack (as depicted in Figure 3), these protocols can be universally implemented within the underlying protocol suite, affording transparency to applications and ensuring widespread adoption. This approach not only enables end-to-end security between clients and servers but also offers the flexibility to be integrated into specific software packages. A testament to its ubiquity and importance, TLS functionality is embedded within the vast majority of web browsers and is an integral component of modern web servers. The evolution from SSL to TLS has marked a pivotal progression in internet security standards, and understanding their mechanisms, applications, and impact is crucial for navigating the digital landscape. Through this paper, we intend to dissect the architecture, discuss the implementation choices, and evaluate the security measures that SSL and TLS provide, illuminating their role in the current era of internet communication.

3.1 Historical Overview of SSL and TLS

The digital landscape of the 1990s witnessed a growing concern for security as the internet increasingly facilitated commercial and private communications. To address these security

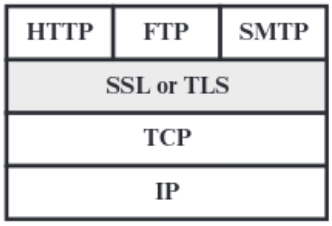

Figure 3: Relative location of security facilities of TLS [Sta]

concerns, SSL (Secure Sockets Layer) was developed by Netscape Communications Corporation, marking the beginning of a series of protocols designed to secure data transmission over the internet.

3.1.1 The Birth and Evolution of SSL

SSL's journey began with the unreleased SSL 1.0 with serious security flaws, quickly followed by SSL 2.0 in 1995. While SSL 2.0 was the first protocol version released to the public, it also had significant security flaws, which led to the development of SSL 3.0 in 1996. SSL 3.0 was developed by Paul Kocher in conjunction with Netscape engineers Phil Karlton and Alan Freier. The reference implementation was carried out by Christopher Allen and Tim Dierks from Consensus Development. It introduced major improvements over its predecessor, such as a complete redesign of the handshake process and enhanced encryption features. Despite these advancements, it was not free from vulnerabilities, as demonstrated by subsequent exploits like the POODLE attack.

3.1.2 Transitioning to TLS

As the internet matured, so did the need for more sophisticated security measures. This led to the Internet Engineering Task Force (IETF) stepping in to standardize a successor to SSL, resulting in the introduction of TLS 1.0 in 1999. TLS 1.0 was first defined in RFC 2246, an attempt to shore up the security of SSL 3.0, yet it maintained a high degree of backward compatibility.

In 2006, TLS 1.1 was released in RFC 4346, addressing some of the cryptographic weaknesses inherent in TLS 1.0, such as issues related to initialization vectors. The evolution continued with TLS 1.2 defined in RFC 5246 in 2008, which introduced a suite of stronger cryptographic algorithms and was widely adopted as the de facto standard for secure web communications.

3.1.3 The Modern Era with TLS 1.3

The most significant leap came with TLS 1.3 (defined in RFC 8446) in 2018. This iteration of the protocol, defined in RFC 8446, represented a major security advancement. It streamlined the handshake process, significantly reducing the time required to establish a secure connection, and enforced the use of forward secrecy, ensuring that the compromise of a single key would not compromise past communications.

3.2 Security Socket Layer

3.2.1 Architecture

The Secure Sockets Layer (SSL) architecture, integral to internet security, operates atop the foundational internet protocols: IP for routing and addressing, and TCP for reliable data transmission. Central to SSL are the SSL Record Protocol, which encrypts and authenticates data exchange to maintain confidentiality and integrity, and the SSL Handshake Protocol, which establishes a secure connection through mutual authentication and key exchange for encryption, both functioning above TCP at the session layer.

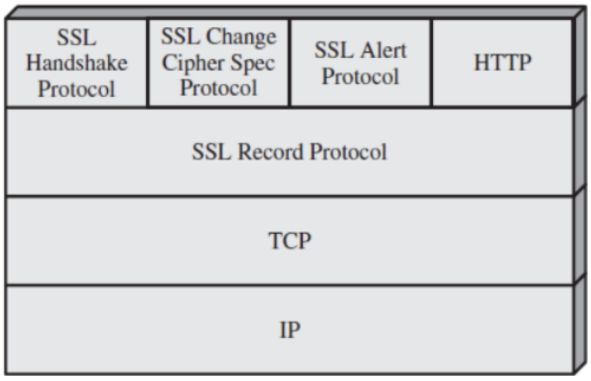

Figure 4: SSL architecture on page 24, lecture slides 5

In practice, the SSL process unfolds in four stages:

- **TCP connection setup**: This begins with the setup of a TCP connection, where the client sends a synchronization signal (SYN) to the server, and the server responds with an acknowledgment (ACK), effectively establishing the TCP connection.
- **Handshake**: Once the TCP connection is active, the SSL handshake begins. During the handshake, both the client and server negotiate to select the specific encryption algorithms and methods they will use for the session. Then, the server, and optionally the client, will be authenticated, to confirms the identity of the parties involved in the communication. Following authentication, keys are established for use in the SSL session.
- **Data transfer**: With the secure channel established, data is transferred between the client and server. This data is encrypted, ensuring that the information exchanged is protected from eavesdropping or tampering.
- **TCP connection closure**: After the secure data transfer is complete, the TCP connection is closed. This is done by sending a finish signal (FIN) followed by an acknowledgment (ACK), signaling the end of the session.

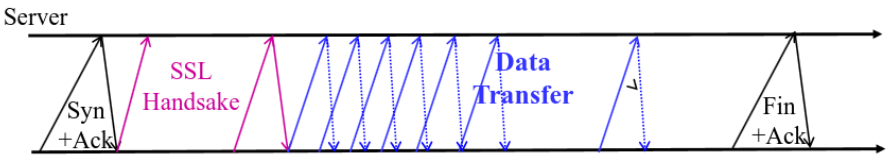

Figure 5: SSL process on page 25, lecture slides 5

3.2.2 Record Protocol

The SSL/TLS Record Layer operates atop a reliable transport protocol, typically TCP, leveraging session keys established during the Handshake phase to secure communication. The Record Layer's duties include fragmenting, compressing, authenticating, and encrypting application data as depicted in Figure 2. The process is as follows:

Application data is initially fragmented into smaller, manageable blocks. Each block then undergoes a sequence of security enhancements. First, the data fragment is compressed to reduce its size. Subsequently, a Message Authentication Code (MAC), derived from algorithms such as HMAC-MD5 or HMAC-SHA256, is computed for the compressed data to ensure its integrity and authenticity. The data is then encrypted using a symmetric encryption algorithm like AES or DES to ensure confidentiality. Finally, an SSL Record header, containing the necessary control information, is prefixed to the encrypted message forming the complete SSL Record.

The integration of MAC and encryption can be implemented in three distinct methods:

- **MAC-then-Encrypt**: This method involves computing the MAC on the plaintext data, then encrypting both the plaintext and the MAC. Although this method bundles authentication and encryption, it has been criticized as it does not protect the integrity of the ciphertext itself. An attacker might alter the encrypted data, which would not be detectable until after decryption.

- **MAC-and-Encrypt**: With this approach, the MAC is computed on the plaintext and then the data is encrypted separately. The MAC and the ciphertext are transmitted together. However, this method also fails to protect the integrity of the encrypted message since the MAC is computed on the plaintext.

- **Encrypt-then-MAC**: This technique, which is now widely recommended, involves encrypting the plaintext first, followed by the computation of the MAC on the resulting ciphertext. The MAC is then appended to the ciphertext. This approach ensures the integrity of the encrypted data as any alteration to the ciphertext would result in a MAC verification failure upon decryption.

The SSL/TLS Record Layer has been the subject of various security vulnerabilities over time, which have prompted the evolution of the protocols to enhance security. Here are some notable vulnerabilities associated with the Record Layer:

- **Attacks on RC4**: RC4 is a stream cipher once popular in SSL/TLS encryption. However, over time, vulnerabilities were discovered, including biases in the RC4 keystream

that could be exploited to gradually reveal plain text from repeated ciphertexts. Due to these vulnerabilities, the use of RC4 in SSL/TLS has been deprecated, and its use is strongly discouraged.

- **CBC IV Reuse (BEAST)**: The Block Encryption Algorithm Secure Transport (BEAST) attack takes advantage of a vulnerability in the Cipher Block Chaining (CBC) mode of operation used with block ciphers. In earlier versions of TLS (1.0 and before), the Initialization Vector (IV) for CBC encryption could be predictable, allowing an attacker to derive information about the plaintext of two consecutive blocks. To mitigate this, later versions of TLS use an explicit per-record IV, improving the security of CBC mode.

- ***MAC-then-Encrypt* and Padding Attacks**: When using *MAC-then-Encrypt*, the MAC is computed on the plaintext and then both the plaintext and MAC are encrypted together. This process involves padding the plaintext to align with the block cipher's block size. Vulnerabilities such as the Padding Oracle On Downgraded Legacy Encryption (POODLE) exploit the predictability and malleability of the padding structure. This can allow an attacker to decrypt messages by manipulating the ciphertext and analyzing the server's responses. To protect against such attacks, the 'Encrypt-then-MAC' method is preferred, and SSL/TLS implementations should use AEAD (Authenticated Encryption with Associated Data) ciphers, which combine encryption and authentication in a single operation and do not require separate padding.

3.2.3 Handshake Protocol

The SSL/TLS handshake involves several steps, which are divided into three phases:

Phase 1: Security Capabilities Negotiation

- **Client to Server**: The client sends a `client_hello` message which includes the client's SSL/TLS version, supported cipher suites, compression methods, and a client-generated random number.

- **Server to Client**: The server responds with a `server_hello` message, selecting the SSL/TLS version, cipher suite, compression method, and providing a server-generated random number for the client to verify the server's identity. This step confirms the server's readiness to begin secure communication.

Phase 2: Authentication and Key Exchange

- **Server to Client**:
  - The server sends its `certificate` for for the client to authenticate the server's identity.
  - If the key exchange algorithm requires additional data (e.g., Diffie-Hellman parameters), the server sends a `server_key_exchange` message.
  - The server requests the client's certificate for mutual authentication using a `certificate_request` message.

  - The server ends this phase with a `server_hello_done` message.

- **Client to Server**:
  - The client verifies the server's certificate.
  - The client sends its own certificate to the server.
  - The client sends a `client_key_exchange` message, containing the pre-master secret encrypted with the server's public key, or the necessary key exchange information.
  - The client sends back a `certificate_verify` message for authentication.

Phase 3: Finalizing the Handshake

- **Both parties** compute the master secret from the pre-master secret and exchanged random numbers.
- **Client to Server**: The client sends a `change_cipher_spec` message to signal that subsequent messages will be encrypted. The client also sends a `finished` message, encrypted with the agreed cipher suite, to verify the key exchange and authentication process.
- **Server to Client**: The server sends a `change_cipher_spec` message, followed by an encrypted `Finished` message, to confirm the handshake completion on its end.
- After both the client and server have sent and verified the `finished` messages, the secure communication channel is established and ready for data transfer.

3.3 Transport Layer Security

3.3.1 TLS 1.2

With the introduction of Transport Layer Security (TLS) 1.2, specified in RFC 5246, a suite of security enhancements was implemented to address the evolving landscape of cyber threats. The key improvements that define TLS 1.2 over its predecessors are as follows:

- **Transition from MD5/SHA-1 to SHA-256**: TLS 1.2 deprecated the use of the MD5 and SHA-1 hashing algorithms, which were susceptible to collision attacks, in favor of the more secure SHA-256. This algorithm is part of the SHA-2 family and provides a 256-bit hash, significantly increasing security for data integrity.
- **Support for Authenticated Encryption**: Cipher suites that incorporate authenticated encryption were introduced, combining encryption and authentication in one step. This not only provides confidentiality but also verifies the integrity and authenticity of the encrypted data.

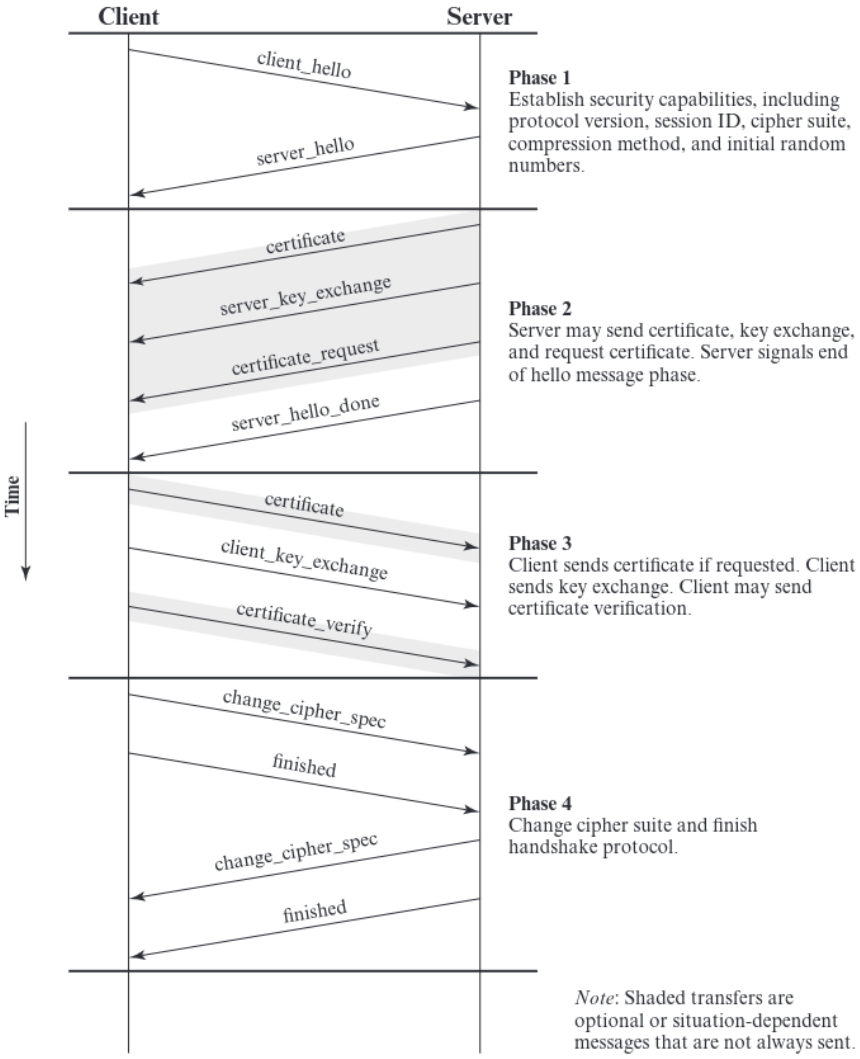

Figure 6: SSL handshake protocol action [Sta]

- **Authenticated Encryption with Additional Data (AEAD)**: The inclusion of AEAD cipher suites like AES-GCM and ChaCha20-Poly1305 allows for the encryption of plaintext and the authentication of both the ciphertext and any associated data. This dual functionality ensures a high standard of security and data integrity.
- **Added HMAC-SHA256 Cipher Suites**: TLS 1.2 extended its cryptographic capabilities by including cipher suites that use HMAC-SHA256 for message authentication. This is a significant improvement over the HMAC implementations using MD5 or SHA-1.
- **Removal of IDEA and DES Cipher Suites**: Outdated cipher suites based on the International Data Encryption Algorithm (IDEA) and Data Encryption Standard (DES) were removed from TLS 1.2 due to their vulnerabilities and limited key sizes. This change fosters a stronger security posture for communications protected by TLS.

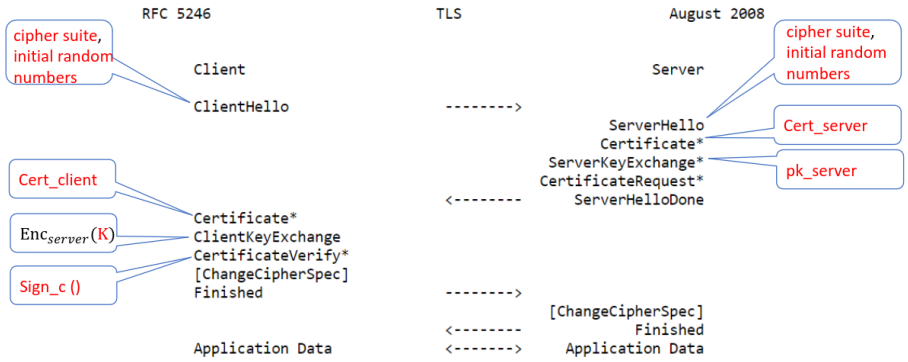

Figure 7: message flow for a full handshake of TLS 1.2 on page 37, lecture slides 5

3.3.2 TLS 1.3

The introduction of Transport Layer Security (TLS) 1.3, as defined in RFC 8446, marked a significant advancement in the TLS protocol with the aim of improving security, efficiency, and speed of encrypted communications. TLS 1.3 simplifies and streamlines the protocol, reducing the number of round trips required to establish a secure connection and introducing several new features to enhance security. Below we summarize the key features of TLS 1.3:

- **Authenticated Encryption with Associated Data (AEAD)**:
  - TLS 1.3 mandates the use of AEAD cipher suites, which combine encryption, decryption, authentication, and validation into a single operation, thereby providing stronger security guarantees.
- **Removal of Static RSA and Diffie-Hellman Cipher Suites**:
  - The protocol removes support for static RSA and DH key exchange methods, enforcing the use of ephemeral key exchange techniques that provide forward secrecy.
- **Encrypted Handshake Messages**:
  - All handshake messages after the initial `ClientHello` and `ServerHello` are encrypted, protecting the integrity of the key exchange and authentication processes.
- **Key Derivation Function**:
  - TLS 1.3 uses the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) for generating keys, which is seen as more secure and efficient.
- **Support for "Zero Round-Trip Time" (0-RTT)**:
  - This feature allows clients to send encrypted data to the server during the initial handshake, without waiting for the handshake to complete, thus enabling faster connection setups. This is particularly useful for reducing latency in subsequent connections.

- **Session Resumption with Pre-Shared Keys (PSK)**:
  - TLS 1.3 allows for session resumption using pre-shared keys, which can significantly reduce the number of round trips needed for subsequent handshakes and speed up the secure connection establishment.

4 HTTPS

HTTPS, denoting HTTP secured by SSL or TLS, is a protocol designed to encrypt data between a web browser and a server. While modern browsers inherently support HTTPS, its effectiveness depends on server compatibility. Notably, some search engines are yet to adopt HTTPS support.

The user-visible change with HTTPS is the 'https://' prefix in the URL instead of 'http://', indicating that the communication is secured and takes place over port 443, in contrast to the standard port 80 for HTTP.

4.1 Key Aspects of HTTPS Encryption

The following components are encrypted under HTTPS:

- The URL of the requested document.
- The document's content.
- Data in browser forms.
- Cookies in both directions.
- The contents of the HTTP header.

HTTPS is formalized in RFC 2818, "HTTP Over TLS," and maintains the core functionality of HTTP while providing an encrypted transport layer.

4.2 Establishing a Secure Connection

In an HTTPS context, the HTTP client also functions as the TLS client. The process includes:

- Connection initiation to the server on the required port.
- Beginning the TLS handshake with a TLS ClientHello message.
- Sending HTTP data as TLS application data after the handshake.
- Adhering to standard HTTP behaviors, like persistent connections.

HTTPS operates across three connection awareness levels, incorporating HTTP request handling, TLS session management, and TCP connectivity.

4.3 Terminating a Secure Connection

To close a connection, the following steps are taken:

- Indication of closure through the HTTP header: `Connection: close`.
- Proper closure at the TLS level involves exchanging 'close_notify' alerts before terminating the connection.

HTTP clients must also be prepared for unexpected TCP disconnections without a 'close_notify' alert. Such events may necessitate a security warning due to their potential as indicators of an attack.

References

[BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

[Gee21] GeeksforGeeks. Tcp/ip model, 2021.

[Int21] InterviewBit. Introduction to the tcp/ip model, 2021.

[S+03] Nigel Paul Smart et al. *Cryptography: an introduction*, volume 3. McGraw-Hill New York, 2003.

[Sta] William Stallings. Cryptography and network security: Principles and practice.