WEEK 5 Lecture Note

Peng Zizhao, Tong Jerry, Wang Fangxiao

Abstract

In this lecture, we explore the fundamentals of TCP/IP, the implementation of SSL/TLS, and the role of HTTPS in secure communication. We begin by delving into the TCP/IP model, its layered structure, and the primary protocols involved. Next, we discuss the SSL/TLS protocol, which provides secure, encrypted communication between clients and servers over the internet. We examine the handshake mechanisms, cipher suites, and certificate-based authentication employed in SSL/TLS. Finally, we introduce HTTPS, a secure version of HTTP that employs SSL/TLS to protect sensitive information transmitted between users and websites, while also highlighting the results and potential risks associated with CA Misissuance.

1 TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is a suite of communication protocols that form the foundation of the internet and facilitate data transmission and communication between computers and networks. Introduced in the mid-1970s, TCP/IP was initially developed by the United States Department of Defense for its ARPANET project and has since become the standard protocol for computer networking.

The TCP/IP model consists of four layers, which are responsible for different aspects of data communication. These layers are:

- Application Layer: The Application Layer is the topmost layer in the TCP/IP model and is responsible for providing the interface between the user applications and the underlying network. This layer includes protocols such as HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), and SMTP (Simple Mail Transfer Protocol) that enable users to access, share, and exchange data over the internet.
- Transport Layer: The Transport Layer is responsible for providing reliable, end-to-end communication between the source and destination devices. The two primary protocols used in this layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP provides a connection-oriented, reliable communication service, while UDP offers a connectionless, faster, but less reliable service.
- Network Layer: The Network Layer is responsible for routing and forwarding data packets between networks. This layer uses the Internet Protocol (IP) to assign unique IP addresses to devices and determine the best path for data transmission. The Network Layer ensures that data packets are delivered to the correct destination, even if the source and destination devices are on different networks.
- Link Layer: The Link Layer, also known as the Data Link Layer or the Network Interface Layer, is responsible for the physical transmission of data over the network. This layer includes protocols such as Ethernet and Wi-Fi, which handle the process of converting data into electrical signals or radio waves and transmitting them over the network medium, such as cables or wireless channels. The Link Layer adds a header and footer to the data packet, creating a frame that encapsulates the information from the higher layers.

2 Secure Socket Layer

In this section, we first give a overall introduction about the architecture of secure socket layer (SSL), and introduce the two important parts of it, i.e., the record layer protocol and the handshake layer, respectively.

2.1 Architecture

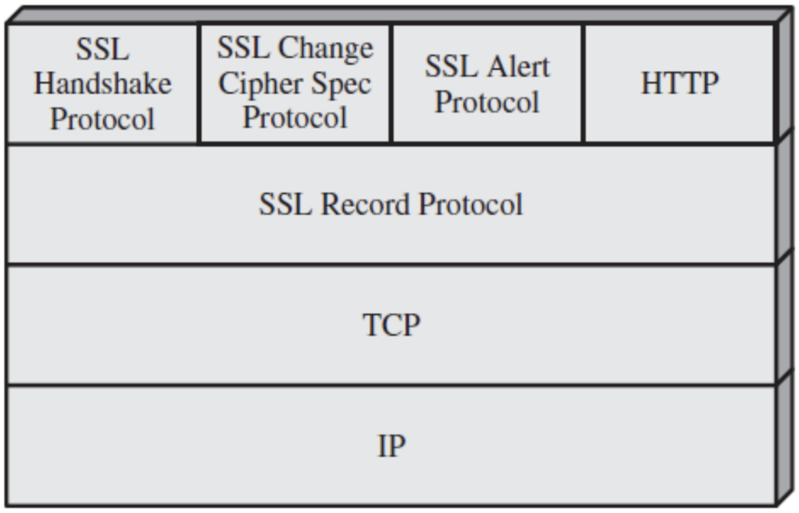


Figure 1: SSL architecture on page 17, lecture slides 5.

The architecture of SSL is shown in Fig. 1. The SSL architecture is built on top of the existing internet communication protocols, i.e., the Internet Protocol (IP) and the Transmission Control Protocol (TCP). IP provides the basic routing and addressing functions necessary for data transmission over the internet, while TCP provides reliable data transfer and flow control mechanisms. The SSL record protocol is responsible for encrypting and authenticating the data that is transmitted between a client and a server. The SSL handshake protocol is responsible for establishing a secure connection between a client and a server. This protocol operates at the session layer, just above the SSL record protocol.

The procedure of SSL including four parts. Firstly, the client establishes a TCP connection with the server. Then, the authenticate server and the client establish keys through negotiate (agree on) algorithms. After that, the SSL can be used to ensure the security and privacy of data being transferred from one system or device to another. After the data being transferred, the TCP connection is closed.

2.2 Record-layer Protocol

The record-layer protocol is built on the reliable communication through TCP and the session key established by the handshake process. There are four stages in record layer, including fragment, compress, authenticate and encrypt, as shown in Fig. 2. The application data, firstly, is fragmented into multiple pieces where each fragment is processed by the following steps respectively. The fragment is compressed and the massage authenticated code (MAC) generated by the MAC algorithm (e.g., HMAC MD5 and SHA256) is attached. Then, the data is encrypted by the encryption algorithm (e.g., AES or DES). Finally, the SSL record header is appended to the start of the message.

There are three types of method of assembling encryption and MAC:

• **MAC-then-Encrypt**. MAC-then-Encrypt is a method used to secure data in which a MAC is generated for the plaintext message first, and then the plaintext message and MAC are together encrypted to produce a ciphertext. This method does not provide any integrity on the ciphertext since we know nothing about the information about the data is authentic or spoofed.

• **Encrypt-and-MAC**. In Encrypt-and-MAC method, MAC is generated based on the plaintext message, and the message is encrypted without the MAC. The MAC and the encrypted message are

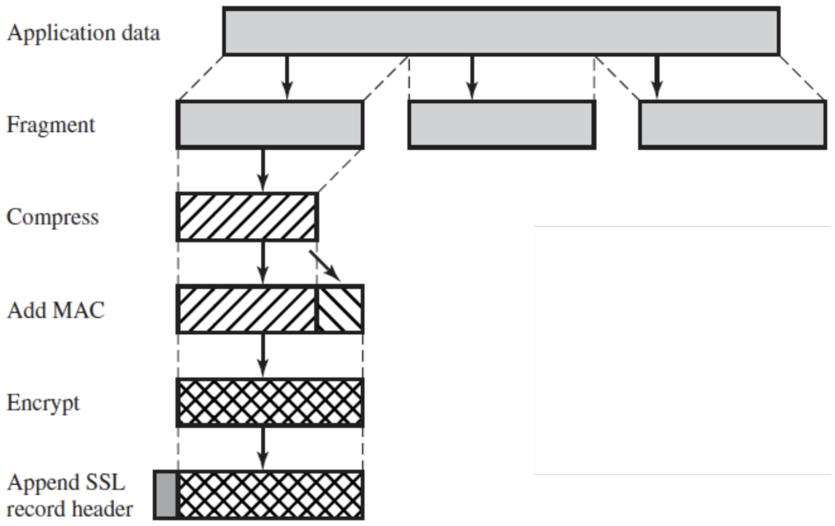


Figure 2: Architecture of record protocol on page 21, lecture slides 5.

sent together. This method also does not provide integrity on the message again, since the MAC is taken against the plaintext.

• **Encrypt-then-MAC**. Encrypt-then-MAC is also used to secure data where the plaintext message is first encrypted, and a MAC is generated for the encrypted message. The resulting MAC is then appended to the ciphertext. This method provides both confidentiality and data integrity and is considered to be more secure than the former two methods, as the MAC is computed over the encrypted message, making it impossible for an attacker to modify the ciphertext without invalidating the MAC.

2.3 Handshake Layer

The handshake layer is the most complex part in SSL. It allows the server and client to authenticate each other. The server and client negotiate encryption, MAC algorithm and cryptographic keys before transmitting the application data.

AKE1 is a simple protocol making use of a Certificate Authority (CA), which bind the identities to public keys through certificates.

$Cert_P$	. denotes P's certificate, binding his identity id_P to his public keys for encryption and signing.
$Enc_P(m)$	. denotes an encryption of the message m under P's public encryption key.
$Sig_P(m)$	. denotes a signature on the message m under P's public verification key.
K	. denotes the set of session keys.
R	. denotes a large set, which will be used to generate random nonces.

The client and server firstly choose random keys from the two sets K and R. Each user will verify the certificate it receives, and the client should also verify the signature and the message.

A more complex handshake method is introduced in SSLv2. The main difference between the SSLv2 handshake and the AKE1 handshake is that the master key of SSLv2 is constructed by the two keys, i.e., the client key K_C and the server key K_S.

3 The development of SSL and TLS

Transport Layer Security (TLS) and **secure Socket Layer(SSL)** are the protocols designed to provide communications security over a computer network.The SSL (Secure Sockets Layer) protocol has three versions, namely SSLv1, SSLv2, and SSLv3. TLS (Transport Layer Security) is a product of standardizing SSLv3, and serves the same purpose of providing secure communication between web browsers and web servers.

Remark 3.1. *A variety of high-level application protocols, such as HTTP, FTP, TELNET, can be transparently established on top of the SSL/TLS protocol layer.*

TLS/SSL consists of two parts:

- a handshake protocol that performs authenticated key exchange to establish the shared keys.
- a record-layer protocol that uses those shared keys to encrypt/authenticate the parties' communication.

3.1 SSL/TLS History

In 1994, Netscape Communications Corporation introduced the first web browser, Netscape Navigator, and also introduced the HTTPS protocol, which used SSL encryption. This was the origin of SSL.

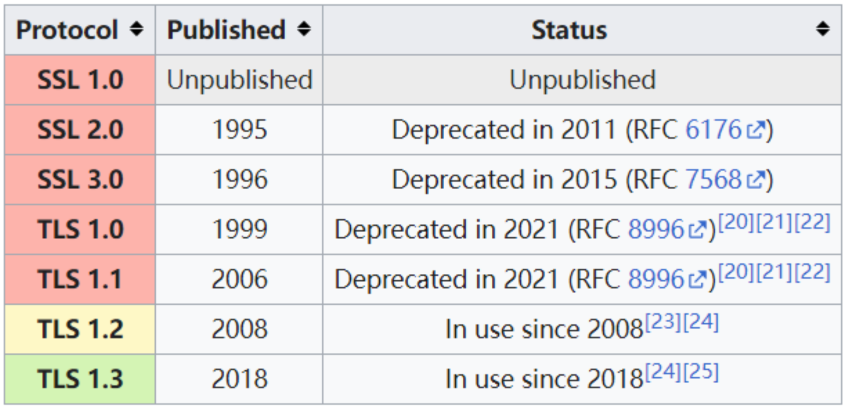
SSL version 1.0 was never publicly released because of serious security flaws in the protocol.

Version 2.0, after being released in February 1995 was quickly discovered to contain a number of security and usability flaws.

These flaws necessitated the complete redesign of the protocol to SSL version 3.0. Released in 1996, it was produced by Paul Kocher working with Netscape engineers Phil Karlton and Alan Freier, with a reference implementation by Christopher Allen and Tim Dierks of Consensus Development. Newer versions of SSL/TLS are based on SSL 3.0. The 1996 draft of SSL 3.0 was published by IETF as a historical document in RFC 6101[FKK11].

TLS 1.0 was first defined in RFC 2246 [DA99] in January 1999 as an upgrade of SSL Version 3.0, and written by Christopher Allen and Tim Dierks of Consensus Development.

TLS 1.1 was defined in RFC 4346 in April 2006. TLS 1.2 was defined in RFC 5246 [DR08] in August 2008. TLS 1.3 was defined in RFC 8446 [Res18] in August 2018.



Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 (RFC 6176)
SSL 3.0	1996	Deprecated in 2015 (RFC 7568)
TLS 1.0	1999	Deprecated in 2021 (RFC 8996)[20][21][22]
TLS 1.1	2006	Deprecated in 2021 (RFC 8996)[20][21][22]
TLS 1.2	2008	In use since 2008[23][24]
TLS 1.3	2018	In use since 2018[24][25]

Figure 3: SSL and TLS protocols on wikipedia.

3.2 SSL 2.0

SSL 2.0 used the same cryptographic keys for message authentication and encryption. It had a weak MAC construction that used the MD5 hash function with a secret prefix, making it vulnerable to length extension attacks. And it provided no protection for either the opening handshake or an explicit message close, both of which meant man-in-the-middle attacks could go undetected. Moreover, SSL

2.0 assumed a single service and a fixed domain certificate, conflicting with the widely used feature of virtual hosting in Web servers, so most websites were effectively impaired from using SSL.

Compared to SSLv1, SSLv2 has the following improvements:

- Improved handshake protocol: SSLv2 uses a more flexible handshake protocol, including negotiation between the client and server to agree on a set of common encryption algorithms and authentication methods.
- Expanded cryptographic algorithms: SSLv2 supports multiple encryption algorithms, such as RC2, RC4, and DES.
- Improved error handling: SSLv2 adds error checking and handling mechanisms to the handshake protocol, making communication more reliable.
- Compression mechanism added: SSLv2 adds data compression mechanisms to reduce data transmission overhead.

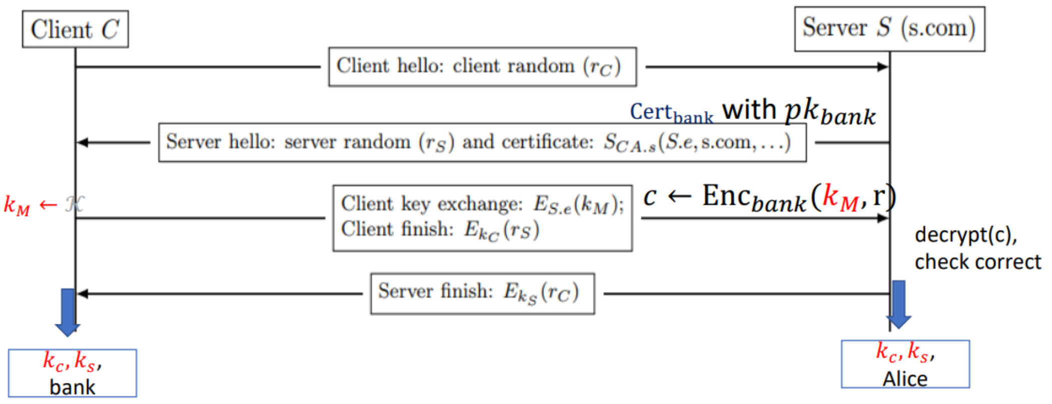
For the improved handshake protocol, the detailed process in Figure 4 is shown below:

PHASE 1: First, the client sends a Client Hello message to the server, which mainly includes the cipher suites, a client random number r_C. Note that this is transmitted in plaintext.

PHASE 2: Upon receiving the message, the server responds with the cipher suites, its digital certificate $S_{CA,s}(S.e, s.com, ...)$, and a server-generated random number r_S. Note that this is transmitted in plaintext.

PHASE 3: The client then begins to verify the digital certificate and may continue to trace it up to a trusted CA (Certificate Authority), CA's CA, CA's CA's CA, etc. After verifying the certificate, the client generates a new **pre-master key** k_M and uses the public key $S.e$ in the certificate to encrypt it. Then, the encrypted pre-master key $E_{S.e(k_M)}$ is sent to the server. Note that this transmission is asymmetrically encrypted. And client use the pre-master key k_M, two random numbers r_C and r_S, **a symmetric encryption key** k_c is generated using the chosen algorithm: Key Derivation function $KDF(k_M)$. This key is then used for normal data communication. k_C can protect traffic from the client to server. At this point, the client's handshake process is complete, as it has achieved the final goal of the handshake: verifying the server's legitimacy and obtaining the symmetric encryption key. So the sign of client finish $E_{k_C}(r_S)$ is also sent to server.

PHASE 4: When the server receives the encrypted pre-master key, it decrypts it using its private key and then uses it together with the previously mentioned random number r_C and r_S to generate **the symmetric encryption key** r_S using the same algorithm: Key Derivation function $KDF(k_M)$. k_S can protect traffic from the server to client. At this point, the server's handshake process is also complete. The sign of server finish $E_{k_S}(r_C)$ is sent to client.



Client C
Server S (s.com)
Client hello: client random (r_C)
Cert$_{bank}$ with pk_{bank}
Server hello: server random (r_S) and certificate: $S_{CA.s}(S.e, \text{s.com}, \ldots)$
$k_M \leftarrow \mathcal{K}$
Client key exchange: $E_{S.e}(k_M)$;
Client finish: $E_{k_C}(r_S)$
$c \leftarrow \text{Enc}_{bank}(k_M, r)$
decrypt(c),
check correct
Server finish: $E_{k_S}(r_C)$
k_c, k_s,
bank
k_c, k_s,
Alice

Figure 4: Handshake on page 26, lecture slides 5.

In the SSL/TLS handshake process, a Key Derivation Function (KDF) is used to generate the symmetric encryption keys from the shared secret k_M obtained during the handshake.

Question 3.1. *Why we need a Key Derivation function here?*

Answer. The reason for using a KDF is to ensure that the resulting keys are strong and unpredictable, even if an attacker knows some of the input values (such as the random numbers and the shared secret). The KDF uses a one-way function, meaning that it is computationally infeasible to derive the input values from the output keys.

Furthermore, the KDF is designed to ensure that the same inputs always generate the same output keys, which is important for the symmetric encryption and integrity protection of subsequent data communication between the client and server. □

Question 3.2. *DH over Z_p^* when $K \in Z_p^*$?*

Answer. To encrypt a message Z_p^* by $K \cdot Mmodp$.

To encrypt a message using AES, the key should be bits? $K_c = Hash(K)$ etc.

It is not secure to utilize K from Z_p^* as a bit string. NOT EVERY bits is random □

The versions of the client hello and server hello messages introduced above are simplified versions. We provided a more detailed handshake process here.

Phase 1 Establish security capabilities, including session ID, cipher suite, compression method, and initial random numbers.

Phase 2 Server may send certificate, key exchange, and request certificate.

Phase 3 Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

At this point, the entire TLS handshake process is complete, and symmetrically encrypted communication can begin.

It should be noted that SSLv2 is considered insecure in modern internet environments, with many security vulnerabilities, and has been deprecated.

3.3 TLS 1.2

The message flow of TLS 1.2 is shown in Figure 5. TLS 1.2 has made important improvements and enhancements over TLS 1.1, its features including:

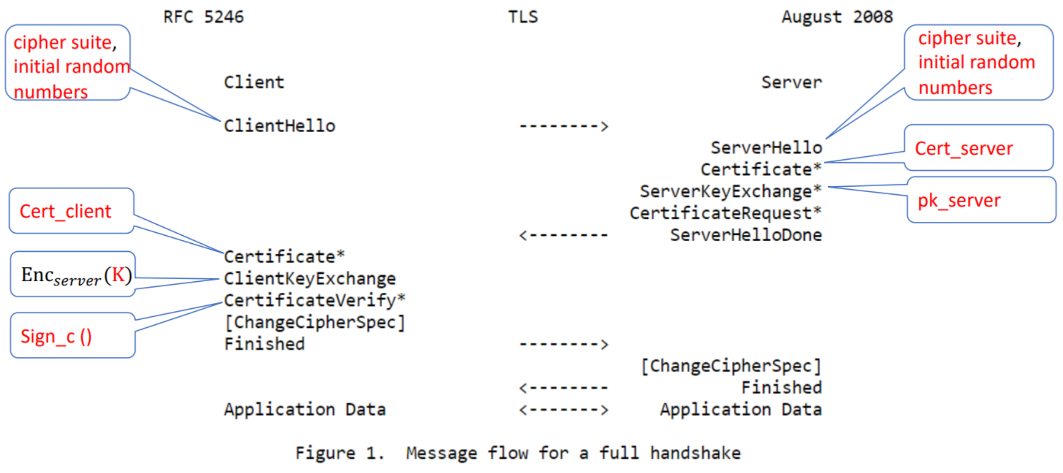


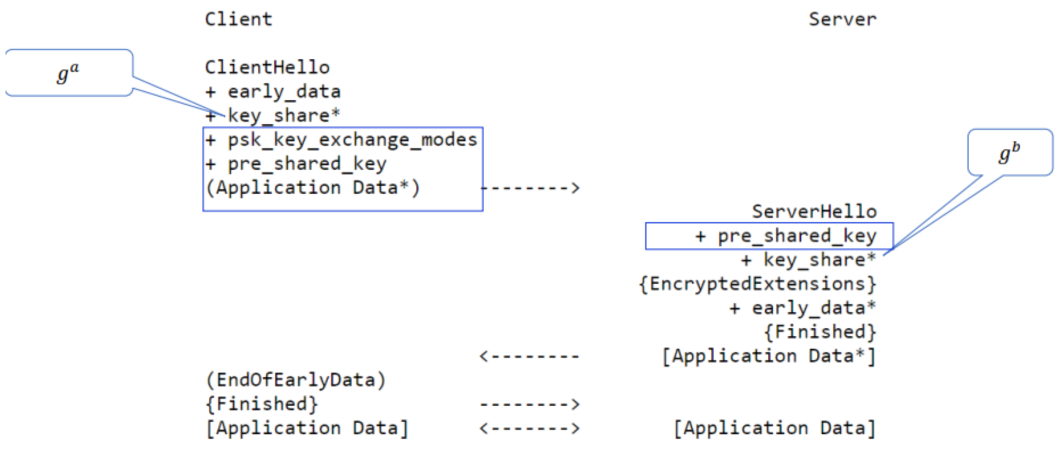
Figure 5: Message flow of TLS 1.2-RFC 5246 on page 31 of lecture 5.

- Support for more Hash algorithms. TLS 1.2 adds support for new Hash algorithms such as **SHA-256**, SHA-384, and SHA-512. TLS 1.2 still allows the use of SHA-1 and MD5 as encryption algorithms, but these algorithms are considered insecure and therefore not recommended for use.

- TLS 1.2 added support for Authenticated Encryption (AE). TLS 1.2 supports two AEAD (Authenticated Encryption with Associated Data) modes, Galois/Counter Mode (GCM) and Counter with CBC-MAC (CCM), to ensure data confidentiality and integrity. These modes can simultaneously achieve encryption and identity verification, providing higher security.
- TLS 1.2 removed support for the IDEA and DES cipher suites. This is because the security of these algorithms has been compromised or is considered insufficiently secure for use in TLS. Instead, TLS 1.2 added support for more secure encryption algorithms, such as HMAC-SHA256 cipher suites.
- TLS 1.2 still supports encryption algorithms such as RSA, CBC, RC4, and SHA1. However, RSA requires a public key, whereas Diffie-Hellman Key Exchange is considered better and provides forward security. CBC is vulnerable to attacks such as BEAST and LUCKY 13. Similarly, the RC4 and SHA1 algorithms are also considered insecure. These algorithms have been completely removed in TLS 1.3.

3.4 TLS 1.3

TLS 1.3 was defined in RFC 8446 in August 2018. It is based on the earlier TLS 1.2 specification. The message flow is shown in Figure 6.



```
       Client                                               Server

       ClientHello
       + early_data
       + key_share*
       + psk_key_exchange_modes
       + pre_shared_key
       (Application Data*)     -------->
                                                       ServerHello
                                                  + pre_shared_key
                                                      + key_share*
                                             {EncryptedExtensions}
                                                     + early_data*
                                                        {Finished}
                               <--------       [Application Data*]
       (EndOfEarlyData)
       {Finished}              -------->
       [Application Data]      <------->        [Application Data]
```

Figure 6: Message flow of TLS 1.3-RFC 8446 on page 38 of lecture 5.

The features of TLS 1.3 include: support Authenticated Encryption with Associated Data (AEAD); static RSA and Diffie-Hellman (Enc) cipher suites have been removed; all handshake messages is encrypted/after key is established; key derivation function is HMAC.

One feature of TLS 1.3 is using one side-use Diffie-Hellman instead of PKE. One example is shown in Figure 7.

Assuming Alice and Bank want to negotiate a symmetric key for communication in an insecure network environment, where g and p are prime numbers. The DH algorithm is as follows:

Phase 1: Firstly, Alice saves her private key a (random number), and Bank also saves his private key b (random number).

Phase 2: Alice calculates$A = g^a \bmod p$ and sends g,p, and A to Bank.

Phase 3: Bank calculates $B = g^b \bmod p$ and sends B to Alice.

Phase 4: Alice obtains the symmetric key$K = B^a \bmod p = (g^b \bmod p)^a \bmod p = g^a b \bmod p$.

Phase 5: Similarly, Bank obtains the symmetric key $K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^a b \bmod p$.

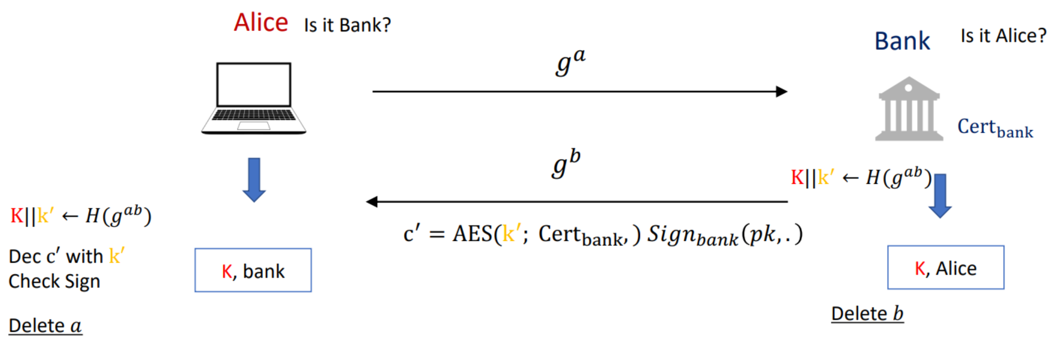


Figure 7: Protocol of TLS 1.3 on page 34 of lecture 5.

Another feature of TLS 1.3 is the supporting of “zero round-trip time” (0-RTT). If there is a pre-shared keys (PSK), then may be used to establish a new connection ("session resumption" or "resuming" with a PSK).

4 HTTPS

4.1 HTTPS

HTTPS represents the integration of HTTP and SSL protocols to facilitate secure communication between a client and a server over the internet. This combination ensures data privacy, integrity, and authentication for users when interacting with websites and online services.

The primary distinction observed by users when interacting with a website using HTTPS is that the URL addresses begin with "https://" instead of "http://". This "s" in "https://" stands for "secure" and indicates that the connection is encrypted, safeguarding the transmitted data from eavesdropping or tampering by malicious actors.

For standard HTTP connections, port 80 is utilized for communication between the client and the server. This type of connection does not provide encryption or authentication, leaving the data transmitted potentially exposed to security risks.

On the other hand, when HTTPS is employed, port 443 is used for communication. This port activation triggers the TLS/SSL handshake process, which establishes a secure connection between the client and the server. The handshake process involves the exchange of cryptographic keys and the verification of the server’s digital certificate, issued by a trusted CA. This ensures that the data transmitted over the connection is encrypted and that the server’s identity is authenticated, preventing possible man-in-the-middle (MITM) attacks.

The communication with HTTPS is shown in Fig 8. First, the client initiates the process by typing in the bank’s domain name. Then the domain name will be sent to a DNS query to resolve the bank’s domain name to an IP address. The Domain Name System translates the human-readable domain name into a numerical IP address that identifies the bank’s server on the internet.

After obtaining the IP address, the client initiates a TCP connection with the bank’s server. This starts with a two-way handshake:

- The client sends a TCP SYN (synchronize) packet to the server, requesting a connection.
- The server responds with an SYN-ACK (synchronize-acknowledge) packet, indicating it is open to establishing a connection.

Once the TCP connection is established, the client initiates a TLS session to secure the communication. The client sends a "ClientHello" message to the server, which includes supported TLS versions, cipher suites, and a random number (ClientRandom). Then The server responds with a "ServerHello" message, which includes the chosen TLS version, cipher suite, and another random

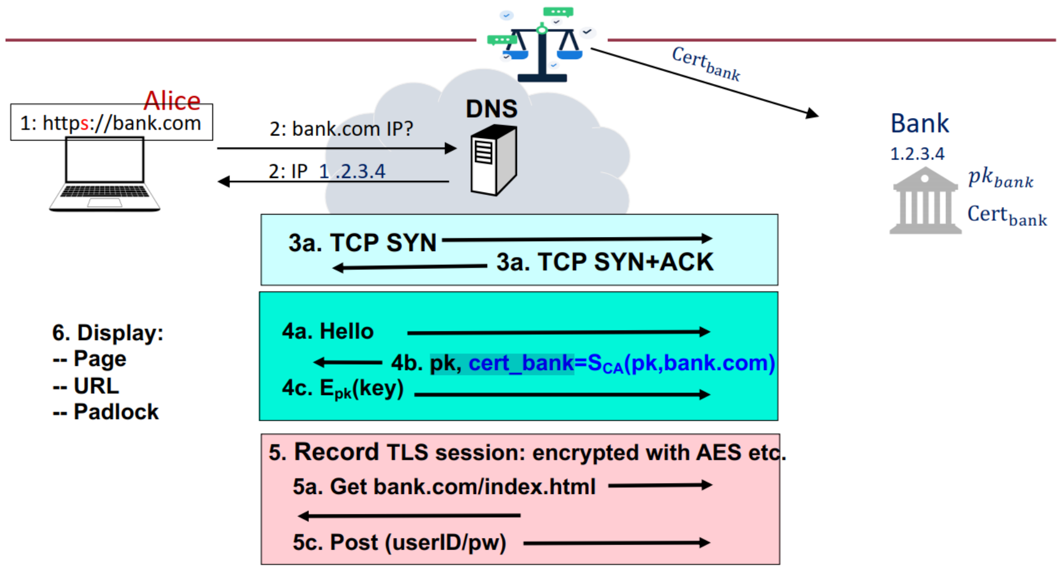


Figure 8: Communication with HTTPS

number (ServerRandom). The server then sends its digital certificate to the client, which contains the server's public key and is signed by a trusted Certificate Authority. The server may also request the client's certificate for mutual authentication.

The client verifies the server's certificate by checking its validity period, the CA's signature, and ensuring the domain name matches the bank's domain. If the certificate is valid and trusted, the client proceeds with the TLS handshake.

The client generates a pre-master secret using the server's public key (pk) from the cert bank and sends it to the server. Both the client and server use the pre-master secret, ClientRandom, and ServerRandom to derive the same master secret, which is then used to create symmetric encryption keys for secure communication.

The client and server exchange "Finished" messages, which are encrypted using the newly established symmetric keys. This step confirms that the handshake was successful and the secure connection is established.

With the secure TLS connection in place, the client and server can now exchange sensitive data, such as login credentials and financial transactions, with the assurance that the communication is encrypted and protected from eavesdropping or tampering. The client can view the display of the Page, URL, and Padlock from the back.

4.2 Certificate Misissuance

The security and integrity of HTTPS communications are fundamentally reliant on the premise that Certificate Authorities (CAs) issue valid and trustworthy digital certificates. These certificates serve as a confirmation of the identity of the entities engaged in online transactions and communications, acting as the backbone of the HTTPS ecosystem. However, the system may be jeopardized when a CA is compromised or mistakenly issues certificates, leading to potentially severe security breaches and undermining users' confidence in the internet's security infrastructure.

Notable incidents involving the compromise or mismanagement of CAs include:

- In 2011, Certificate Authorities Comodo and DigiNotar were breached by attackers, resulting in the unauthorized issuance of fraudulent certificates for prominent web services such as Gmail and Yahoo! Mail. These incidents exposed the vulnerabilities in the CA infrastructure, raising concerns about the potential for malevolent actors to intercept or manipulate sensitive user data, and ultimately leading to the revocation and dissolution of DigiNotar.

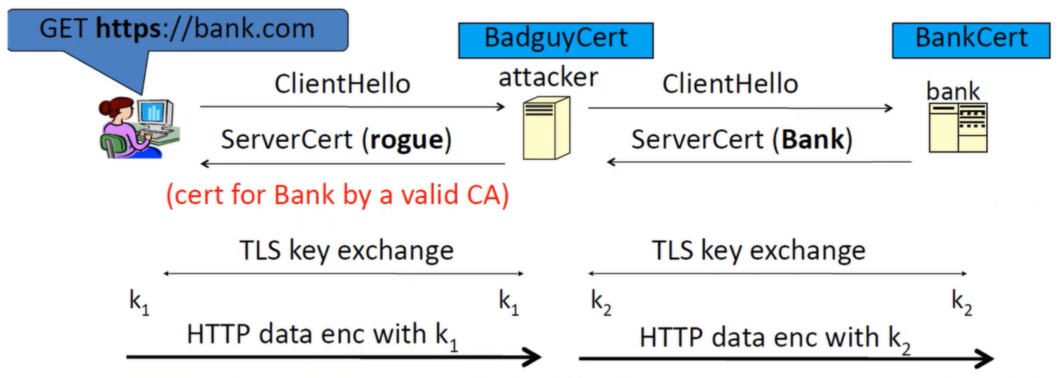


Figure 9: Insecure CA

- In 2013, TurkTrust inadvertently issued a certificate for the gmail.com domain, which could have enabled attackers to impersonate Google's email service and intercept or tamper with users' communications. The certificate was subsequently revoked, and the incident highlighted the need for enhanced oversight and controls in the certificate issuance process.
- In 2016, Chinese CA WoSign issued a certificate for the GitHub domain, among other infractions, which could have put the users of the popular code-sharing platform at risk of man-in-the-middle attacks. As a result of these transgressions, WoSign's certificates were no longer trusted by major internet browsers such as Chrome, Firefox, and Apple, highlighting the severe consequences that can arise from the mismanagement of digital certificates.

The Fig. 9 shows the consequences of compromised certificate authorities. The attacker knows data between the user and the bank and sees all traffic and can modify data at will.

In conclusion, the security of HTTPS communications is contingent upon the proper functioning and management of Certificate Authorities. When CAs are compromised or erroneously issue certificates, the entire HTTPS ecosystem can be jeopardized, leading to potential security breaches and a loss of trust in the underlying infrastructure. To maintain the credibility and effectiveness of the HTTPS protocol, it is imperative to establish robust security measures and oversight mechanisms to prevent future incidents of certificate misissuance and CA compromise.

References

[DA99] Tim Dierks and Christopher Allen. The tls protocol version 1.0. Technical report, 1999.

[DR08] Tim Dierks and Eric Rescorla. Rfc 5246: The transport layer security (tls) protocol version 1.2, 2008.

[FKK11] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. Technical report, 2011.

[Res18] Eric Rescorla. The transport layer security (tls) protocol version 1.3. Technical report, 2018.