MH4311 Cryptography

Lecture 17

Key Establishment

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
- Public key encryption
- Digital signature
- **Key generation, establishment and management**
  - Key generation
  - **Key establishment**
    - **Key establishment using symmetric key cryptography**
    - **Key establishment using public key cryptography**
      - **PKI, Certificate: TLS/SSL**
      - **SSH**
  - Secret sharing
- Introduction to other cryptographic topics

Recommended Reading

- CTP: Section 10.2, 10.5.3, 10.5.4

  Section 12.1,12.2,12.3,12.4

- Wikipedia

  Key establishment:

  - http://en.wikipedia.org/wiki/Kerberos_(protocol)
  - http://en.wikipedia.org/wiki/Public_key_certificate
  - http://en.wikipedia.org/wiki/Public_key_infrastructure
  - http://en.wikipedia.org/wiki/Transport_Layer_Security

Key Establishment

- Key establishment (also called key exchange) is to allow secret keys being exchanged between two parties, then the two parties can use keys for secure communication.
- Three approaches for key establishment between two users
  - Pre-share secret keys between any two users
    - Expensive
  - **Symmetric key cryptography (SKC) approach**
    - Each user shares a secret key with a trusted server
    - Then any two users establish secret keys by communicating with the trusted server
  - **Public key cryptography (PKC) approach**
    - Each user publishes its public key
    - Then others users can share secret keys with that user using public key cryptography

Key Establishment with Symmetric Key Cryptography

Key Establishment: SKC Approach

- Key establishment with symmetric key cryptography
  - Each user shares a secret key with a trusted server
  - Then any two users establish secret keys by communicating with the trusted server
- Schemes
  - *Kerberos
  - Bellare-Rogaway Scheme

Key Establishment: SKC: Kerberos*

- Kerberos (animal)
  - Three-headed guard dog of Hades (god of the underworld in Greek mythology)

Key Establishment: SKC: Kerberos*

- Kerberos (protocol/software)
  - A popular session key distribution scheme
    - Mainly for authentication so as to access a service over a network
    - DES/AES is used in the scheme
  - Developed by MIT in the late 1980s

Key Establishment: SKC: Kerberos*

- Simplified Kerberos V5

TA: trusted authority
Alice shares a secret key K_{Alice} with TA;
Bob shares a secret key K_{Bob} with TA;
Alice & Bob authenticate each other, and establish a secret key *K*

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$ and r_A to the TA.
2. The TA chooses a random session key K and a validity period (or *lifetime*), L. Then it computes a ticket to Bob,

$$t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice) \parallel L),$$

and

$$y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel L).$$

The TA sends t_{Bob} and y_1 to Alice.

3. Alice decrypts y_1 using her key K_{Alice}, obtaining K. Then Alice determines the current time, $time$, and she computes

$$y_2 = e_K(ID(Alice) \parallel time).$$

Finally, Alice sends t_{Bob} and y_2 to Bob.

4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. He also decrypts y_2 using the key K, obtaining $time$. Then, Bob computes

$$y_3 = e_K(time + 1).$$

Finally, Bob sends y_3 to Alice.

Key Establishment: SKC: Kerberos*

Simplified Kerberos V5 (cont.)

1. When Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1)$ has the form

$$d_{K_{Alice}}(y_1) = r_A \,||\, ID(Bob) \,||\, K \,||\, L,$$

for some K and L. If this condition does not hold, then Alice "rejects" and aborts the current session.

2. When Bob decrypts y_2 and t_{Bob}, he checks to see that the plaintext $d_K(y_2)$ has the form

$$d_K(y_2) = ID(Alice) \,||\, time$$

and the plaintext $d_{K_{Bob}}(t_{Bob})$ has the form

$$d_{K_{Bob}}(t_{Bob}) = K \,||\, ID(Alice) \,||\, L,$$

where $ID(Alice)$ is the same in both plaintexts and $time \leq L$. If these conditions hold, then Bob "accepts"; otherwise Bob "rejects."

3. When Alice decrypts y_3, she checks that $d_K(y_3) = time + 1$. If this condition holds, then Alice "accepts"; otherwise Alice "rejects."

Key Establishment: SKC: *Kerberos

- Kerberos is complicated
  - **Message Authentication Code is not used** in Kerberos, so the authentication in Kerberos is somehow complicated (achieved through encryption/decryption)

Key Establishment: SKC: Bellare-Rogaway Scheme

- Bellare-Rogaway key establishment scheme (1995)

1. Alice chooses a random number, r_A, and she sends $ID(Alice)$, $ID(Bob)$ and r_A to Bob.
2. Bob chooses a random number, r_B, and he sends $ID(Alice)$, $ID(Bob)$, r_A and r_B to the TA.
3. The TA chooses a random session key K. Then it computes

$$y_B = (e_{K_{Bob}}(K), MAC_{Bob}(ID(Alice) \,||\, ID(Bob) \,||\, r_B \,||\, e_{K_{Bob}}(K)))$$

and

$$y_A = (e_{K_{Alice}}(K), MAC_{Alice}(ID(Bob) \,||\, ID(Alice) \,||\, r_A \,||\, e_{K_{Alice}}(K))).$$

The TA sends y_B to Bob and y_A to Alice.

Key Establishment: SKC: Bellare-Rogaway Scheme

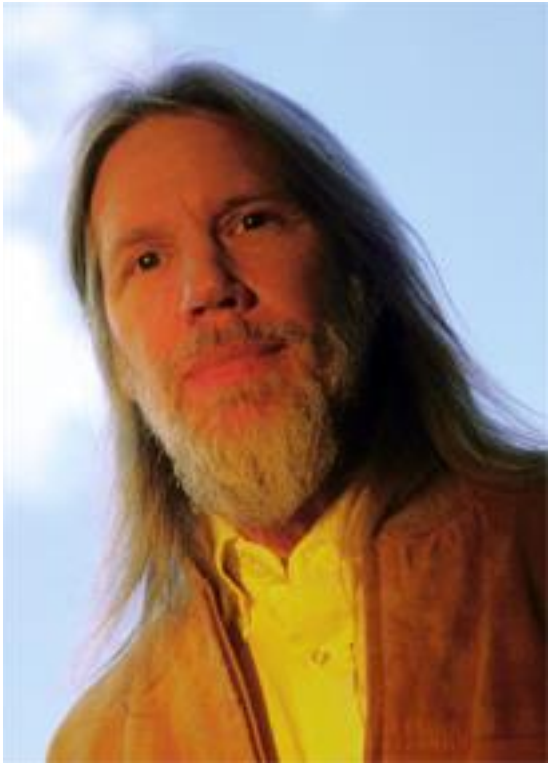
- Bellare-Rogaway key establishment scheme
  - Random numbers r_A and r_B are used in the scheme to ensure that a new session key K is established between Alice and Bob

Key Establishment with Public Key Cryptography

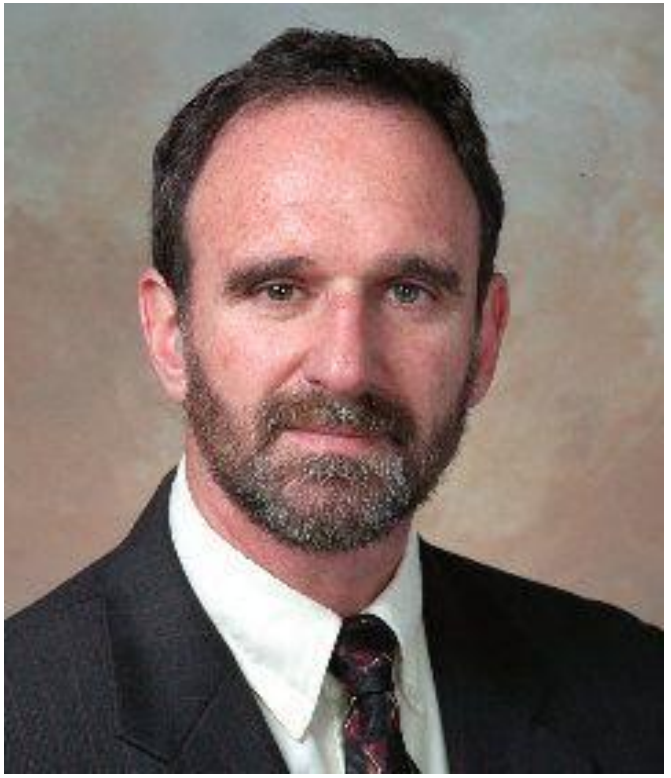
Key Establishment: PKC Approach

- Key Establishment using public key cryptography
  - Approach 1: Public key encryption schemes
    - RSA, ElGamal …
  - Approcah 2: Diffie-Hellman key exchange (DH)
    - The first public key cryptosystem (1976)
    - Based on the difficulty of discrete logarithm
    - Illustrated in the next two slides

Diffie-Hellman Key Exchange



Whitfield Diffie



Martin Hellman

Diffie-Hellman Key Exchange

Two system parameters that are used by all users:

1) a large prime ***p***

2) a generator ***g*** of the multiplicative cyclic group Z_p^*

	Alice	Bob
step 1:	generate random number r_a	generate random number r_b
step 2:	compute $Y_a = (g^{r_a}) \bmod p$	compute $Y_b = (g^{r_b}) \bmod p$
step 3:	send Y_a to Bob	send Y_b to Alice
step 4:	compute $K_a = (Y_b)^{r_a} \bmod p$	compute $K_b = (Y_a)^{r_b} \bmod p$

$$K_a = K_b$$

Key Establishment: PKC Approach

- An example of using public key encryption scheme to establish secret key:
  - SSH (Secure Shell)
    - Used for remote secure access to Linux/Unix system
    - Key establishment
      - Step 1. A user requests the public key of a Linux server
      - Step 2. The server sends its public key to the user
      - Step 3. **If the user trusts the received public key**, types "yes", then the public key of the server is used to establish secret key between the user's computer and the Linux/UNIX system
        - » the trusted public key is stored on the user's computer, and no confirmation is needed for future accesses
      - Step 4. After establishing a secure channel, the user enters password for remote login into the server, and start to access the server if password verification is successful.

Key Establishment: PKC Approach

- Sending public key over network
  - **Secure**
    - If the attacker does not have the capability to modify the network traffic
  - **Insecure**
    - If the attacker can modify the internet traffic, the attacker can launch the man-in-the-middle attack (Lecture 15)
    - How to ensure the authenticity of the received public key?
      - Public key infrastructure (PKI) is commonly used

Public Key Infrastructure (PKI)

PKI and Public Key Certificate

- A public key certificate binds a public key with its owner's identity
  - **Signed** by a certificate authority (CA) or other party
- Public key infrastructure (PKI)
  - Public key infrastructure is built on public key certificate
  - create, manage, distribute, use, store, and revoke public key certificates
  - With the use of PKI, we can ensure the authenticity of public keys over the internet

Public Key Certificate

- The most widely used public key certificate standard is **X.509**
  - Now version 3
- There are other certificates formats, but not widely accepted
  - PGP public key certificate (Web of Trust)
    - PGP stands for Pretty Good Privacy, an secure email system.
    - Each user self-sign his/her certificate, the trust on the certificate is based on other users' attestations
  - Simple Public Key Infrastructure (SPKI)

X.509

Contents of X.509 public key certificate

- Version
- Serial number
- Signature algorithm ID
- Issuer
- Validity period
  - Not Before
  - Not After
- Subject
- Subject public key info
  - Subject's public key algorithm
  - Subject's public key
- (optional information)
- Certificate signature

Version: v1, v2 or v3
Serial number: Used to uniquely identify the certificate.
Signature algorithm ID: the algorithm used to create the signature
Issuer: The name of the Certificate Authority
Valid-from: The date the certificate is first valid from.
Valid-to: The expiration date.
Subject: The person, or entity identified.
Subject public key info: the public key of the subject
Certificate signature: authenticate all the above contents, signed by the issuer (CA)

Example: X.509 public key certificate of NTULearn

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:a4:9c:37:b9:b7:d1:91:80:f6:8f:6d:85:b4:c2:b1
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
        Validity
            Not Before: Feb 22 00:00:00 2016 GMT
            Not After : May 21 12:00:00 2019 GMT
        Subject: C=SG, ST=Singapore, L=Singapore, O=Nanyang Technological University, CN=*.ntu.edu.sg
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ac:60:7c:e2:26:60:ca:d2:50:d2:ab:96:ef:78:
                    da:24:ca:5f:e0:78:d0:b9:40:a5:2f:40:6b:4c:89:
                    12:f5:46:70:e2:3e:fc:7d:d8:7e:a9:60:d5:98:aa:
                    cb:98:59:b4:30:28:0f:95:2b:78:5d:77:c1:ab:69:
                    cb:66:1c:70:86:87:84:4a:b1:cb:56:41:b8:72:cc:
                    c4:1d:fd:b6:36:f5:43:e3:fc:ca:d1:8a:ca:71:f4:
                    51:24:74:ce:16:b7:9d:91:15:22:65:de:6a:be:95:
                    6a:3d:5d:69:07:d3:52:26:28:b7:da:ad:33:a6:ef:
                    6c:54:c3:94:5f:40:c2:07:c6:fe:40:ac:00:1a:8b:
                    1d:f5:6a:6a:ae:a3:0e:ad:09:f2:29:1f:f8:bb:83:
                    30:ab:78:9c:94:7b:d1:1b:d3:6d:42:1f:21:b6:a7:
                    31:63:31:77:9b:75:27:11:42:3e:a2:48:93:39:47:
                    17:8e:34:12:ec:4f:fd:dd:e4:36:69:28:45:a9:04:
                    32:38:11:78:ec:96:b0:da:78:70:af:c1:20:88:9d:
                    63:8d:ab:c2:ef:86:e5:29:b7:35:d8:87:2d:b4:a4:
                    c5:50:ff:52:84:22:f8:bf:cc:46:70:b7:31:46:a0:
                    17:f2:0d:39:3e:9b:94:2d:c3:32:f1:15:87:2e:63:
                    9c:af
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

(…the extensions are omitted here)

```
Signature Algorithm: sha256WithRSAEncryption
     8e:8c:ac:b5:32:a9:6b:f9:93:e7:1e:96:00:63:14:d7:fc:83:
     08:c5:4e:7c:75:f4:5c:ed:e0:25:7c:ad:34:8b:59:88:d1:9f:
     ca:b1:57:8d:d5:90:d0:8f:85:76:37:a2:f6:d0:52:66:6a:ae:
     37:18:f6:68:93:a9:19:19:01:c2:51:74:0d:50:70:27:a9:e0:
     db:ad:60:7d:23:d6:3b:64:48:16:49:0f:0f:2b:35:76:c0:f9:
     6c:35:d8:30:71:10:0b:d9:3e:2d:82:03:ce:97:fb:1d:ea:29:
     06:02:bd:81:91:b8:f8:c4:af:3b:1a:16:fb:b3:ef:df:2b:42:
     c1:5a:03:7b:3d:b3:72:8d:f7:6f:37:40:7f:7e:23:79:ea:82:
     44:ab:bd:87:d2:7e:ad:8d:d5:ce:d8:5d:34:d8:6a:68:be:48:
     40:74:48:c2:8f:60:a2:a3:3f:f2:d3:6d:f8:1b:d5:c4:3b:2b:
     3a:b6:c2:3e:01:de:5d:9a:0f:c7:b2:e4:c3:ea:37:ce:1f:89:
     fc:d7:ac:c0:f4:59:72:de:d6:37:0e:64:9e:b9:88:78:e9:50:
     52:54:3f:7a:d2:18:3b:38:76:92:27:73:52:12:92:71:1b:0a:
     94:9d:89:b8:57:a3:29:e2:1f:28:02:49:b0:9b:99:c9:ad:2c:
     fa:3a:c7:7e
```

Signature

X.509 Certificate Generation

- A user (subject) first generates a key pair, keeping the private key secret
- The user sends the public key together with the identification information to the CA.
- The CA issues the certificate binding a public key together with a particular subject

X.509 Certificate Generation

- Normally a CA does not sign a certificate of a website directly
  - It is to protect the private key of the CA
  - This CA is called root CA. Its public key is stored in each web browser.
- CA gives a website a certificate chain which consists of at least three certificates
  - 1) Root certificate (Example: root CA DigiCert)
    - Issuer and subject fields are the same (the root CA)
    - The public key in the root certificate is that of the root CA
    - Self-signed using the private key of the root CA
    - Long validity period: more than 20 years

X.509 Certificate Generation

- CA gives a website a certificate chain which consists of at least three certificates (cont.)
  - 2) Intermediate certificate (Example: DigiCert SHA2 High Assurance Server CA is an intermediate CA of the root CA DigiCert)
    - Issuer is the root CA, subject is this intermediate CA
    - The public key in the certificate is that of the intermediate certificate
    - Signed using the private key of the root CA
    - Long validity period: more than 10 years

X.509 Certificate Generation

- CA gives a website a certificate chain which consists of at least three certificates (cont.)
  - 3) End-entity certificate
    (Example: NTULearn website is an end-entity)
    - Issuer is the intermediate CA, subject is the website address
    - The public key in the certificate is that of the website
    - Signed using the private key of the intermediate CA
    - Normally short validity period: one or two years
      (the price of a certificate is related to the validity period)

X.509 Certificate Revocation

- A certificate needs to be revoked
  - if the private key corresponding to the public key in the certificate has been compromised,
  - or if the subject of the certificate is not longer deemed trusted
- Each CA maintains a certificate revocation list which is a list of revoked certificates (by serial number). The certificate revocation list is signed by the CA.

X.509 Certificate Verification

- When a user accesses a secure website (https://), the website sends the certificate chain to the user
- Verification of the certificate chain of a website
  - We should ensure that all the certificates in the certificate chain are valid, then we can trust the public key of a website
  - Verification of the end-entity certificate
    - Is the certificate revoked?
    - Does the subject match the website being accessed?
    - Within validity period?
    - Verify the signature using the public key of intermediate CA

X.509 Certificate Verification

- Verification of the certificate chain of a website (cont.)
  - Verification of the intermediate certificate
    - Is the certificate revoked?
    - Within the validity period?
    - Verify the signature using the public key of root CA
  - Verification of the root certificate
    - Within the validity period
    - Verify the signature using the public key in the root certificate
    - **The public key in the certificate should match the CA's public key which was stored in the browser/computer**

X.509 Security

- Security problems with the X.509
  - Architectural weaknesses
    - The certificates being revoked may still be used
      - Most users trust certificates when the certificate revocation list is not available (the certificate revocation list may be unavailable due to the crash of the server of CA).
      - If an attacker can control the communication channel of a user to disable the access to the certificate revocation list, the attacker can use the certificates being revoked
    - Revocation of root certificates is not addressed
    - ….

X.509 Security

- Security problems with X.509
  - Problems with CAs
    - There are too many root CAs (driven by the profit of issuing certificates). Nowadays, each type of web browser supports at least 100 root CAs. If a hacker compromises **one** CA, the attacker can forge the certificate, thus can launch the man-in-the-middle attack successfully.
      - In 2011, Iran hackers forged the certificates of two CAs, Comodo and DigiNotar, then used the forged certificates to launch the man-in-the-middle attack against Iran's Gmail users.
    - An attacker may manage to get the certificate of a user by claiming to represent that user when the CA is not careful enough.
      - In 2001, CA VeriSign issued two certificates to a person claiming to represent Microsoft. The certificates have the name "Microsoft Corporation", so they could be used to spoof someone into believing that updates to Microsoft software came from Microsoft when they actually did not.

X.509 Security

- Security problems with X.509
  - Implementation issues
    - In some applications, certificate verification is not performed (to speed up the applications, or unaware of the security consequence)
      - In 2014, it was found that 73% of Android applications do not check certificates
    - Many implementations turn off revocation check
    - …..

TLS/SSL

TLS/SSL

- TLS/SSL protocol is widely used for secure web access:
  - All the https websites, such as webmail, facebook, online banking, online shopping ….
- TLS is the successor to the now-depreciated SSL
  - TLS: Transport Layer Security
  - SSL: Secure Socket Layer
  - TLS 1.2 is widely used today
  - TLS 1.3 is the coming version

TLS/SSL

- In a simple TLS/SSL protocol, a client's browser requests the server's certificate chain, then the client and the server use the server's public key to establish (also called exchange) secret key(s) using public key cryptosystem
- The exchanged secret key(s) are used to encrypt and authenticate the packets using symmetric key ciphers and message authentication code.

TLS/SSL Key Exchange

- There are two methods of key establishment using public key cryptosystems
  - Public key encryption
  - Diffie-Hellman key exchange
- Three public key cryptosystems are supported for TLS/SSL key exchange
  - RSA encryption
  - Diffie-Hellman key exchange (DH)
  - Elliptic curve Diffie-Hellman key exchange (ECDH)
    - The usage of ECDH is the same as that of DH

TLS/SSL Key Exchange

- Using RSA for key exchange is simple
  - The client verifies the server's public key in the certificate
  - The client generates a secret random number *S*, encrypts *S* using the RSA public key of the CA, sends the ciphertext to the server, the server recovers *S* (details are omitted here)

TLS/SSL Key Exchange

- In TLS, if Diffie-Hellman key exchange is needed, **Ephemeral Diffie-Hellman** key exchange (named EDH or DHE) is commonly used
  - DHE is used together with a digital signature algorithm
    - All those three algorithms in the Digital Signature Standard are supported in TLS
      - Digital Signature Algorithm (DSA)
      - RSA Digital Signature Algorithm (simply denoted as RSA in TLS)
      - Elliptic Curve Digital Signature Algorithm (ECDSA)
  - The server's public key in the certificate is the public key for digital signature

TLS/SSL Key Exchange

- In DHE, for each TLS session,
  - The server generates a new secret random number x, computes $\boldsymbol{t_s = g^x \bmod p}$, **signs (t_s, g, p)** using the server's private key, sends (t_s, g, p) together with the signature to the client
  - The client verifies the server's public key in the certificate.
  - The client uses the server's public key to **verify (t_s, g, p)**
  - The client generates a new secret random number y, computes $\boldsymbol{t_c = g^y \bmod p}$, sends t_c to the server
  - The server and client compute the secret $\boldsymbol{S = g^{xy} \bmod p}$

TLS/SSL Key Exchange

- DHE achieves **forward secrecy**
  - In case that the server's private key is compromised by attacker, the attacker can not recover the previous communication since there is no way to recover the previous secret keys being exchanged from the server's private key
  - But the attacker can use the server's private key to launch the man-in-the-middle attack to recover the data in the current TLS connections

TLS/SSL Symmetric Key Encryption and Message Authentication Code

- In TLS 1.0, 1.1, and 1.2, the encryption and authentication could be done separately
  - Commonly used encryption algorithm
    - AES, RC4
  - Commonly used message authentication code
    - HMAC (with SHA-1, SHA-256, or SHA-384)

TLS/SSL Symmetric Key Encryption and Message Authentication Code

- In the coming TLS 1.3, the encryption and authentication must be performed in a single algorithm (authenticated encryption algorithm)
  - For example, AES-GCM is an authenticated encryption algorithm
  - AES-CBC cannot be used in TLS 1.3 since it is not authenticated encryption algorithm

TLS/SSL Cipher Suites

- In TLS, the server and the client needs to agree on a cipher suite that specifies which algorithms would be used for key exchange, symmetric key encryption and message authentication code.

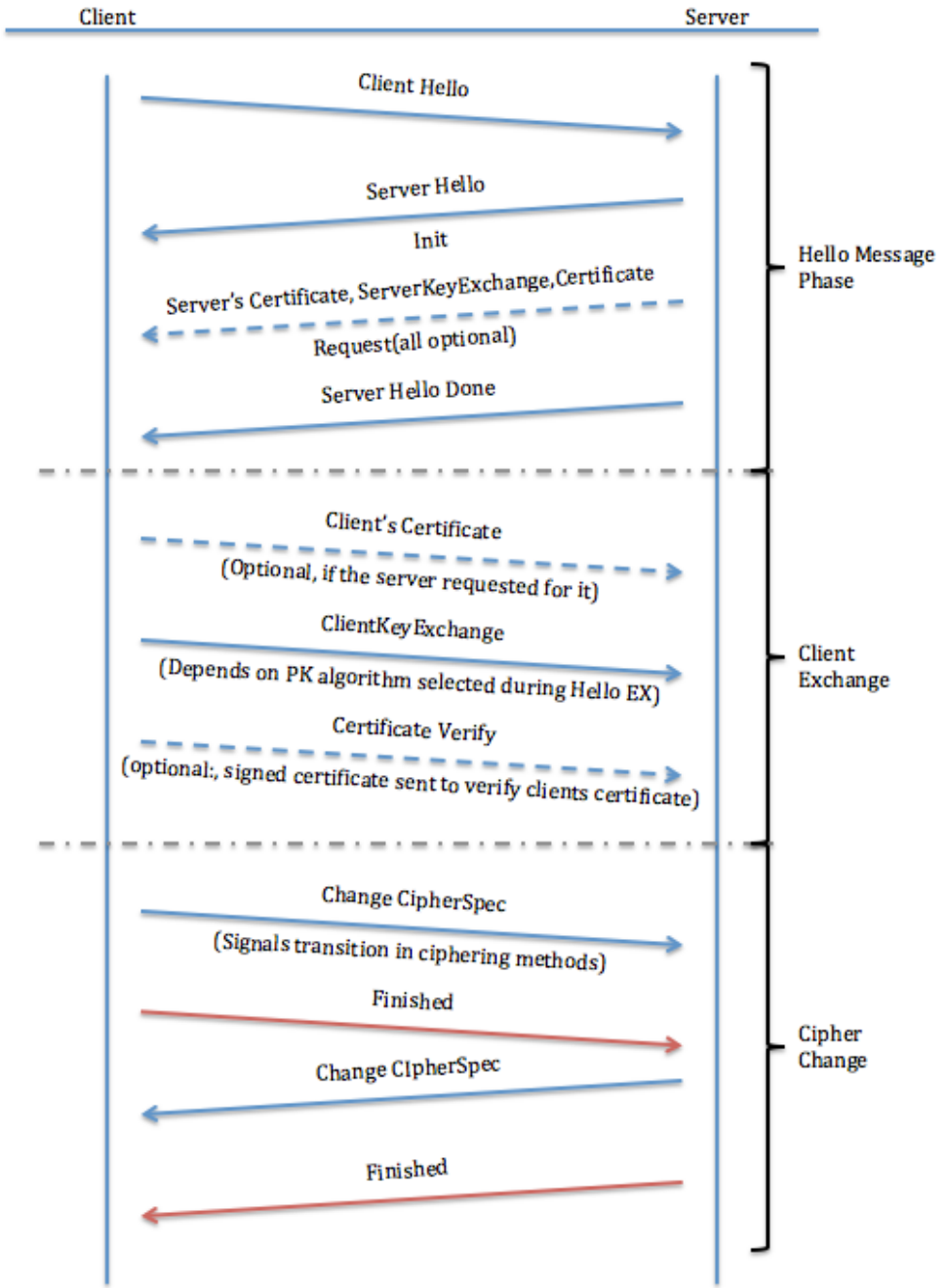
TLS/SSL Cipher Suites

- Some examples of cipher suites
  - TLS_RSA_WITH_AES_256_CBC_SHA
    - RSA encryption is used for key exchange
    - AES-256 in CBC mode is used for encryption
    - HMAC-SHA-1 is used for message authentication
  - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    - ECDHE together with RSA Digital Signature Algorithm is used for key exchange
    - AES-256 in CBC mode is used for encryption
    - HMAC-SHA-1 is used for message authentication
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - ECDHE-RSA: the same as above.
    - AES-128-GCM is used for message encryption and authentication
    - Here HMAC-SHA-256 is mainly **not** used for message authentication. It is mainly used for deriving keys here.

TLS/SSL Handshake Protocol

- TLS Handshake Protocol is responsible for the authentication and key exchange necessary to establish or resume secure sessions

TLS/SSL Handshake Protocol



TLS/SSL Handshake Protocol

- A simple TLS handshake protocol for establishing a secure session involves the following steps (Suppose that RSA encryption is used here for key exchange):
  - The client sends a "Client hello" message to the server, along with the highest version number understood by the client, client's random value and supported cipher suites.
  - The server responds by sending a "Server hello" message to the client, along with the protocol version being selected, the server's random value, a unique session ID, and the cipher suite being selected.
  - The server sends its certificate to the client.
  - The server sends the "Server hello done" message.

Continued on the next page….

TLS/SSL Handshake Protocol

- The client creates a random Pre-Master Secret and encrypts it using the verified public key from the server's certificate, sending the encrypted Pre-Master Secret to the server.
- The server receives the Pre-Master Secret. The server and client each generate the Master Secret and session keys based on the Pre-Master Secret.
- The client sends "Change cipher spec" notification to server to indicate that the client will start using the new session keys for authenticating and encrypting messages. Client also sends "Client finished" message.
- Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption and authentication using the session keys. Server sends "Server finished" message to the client.
- Client and server can now exchange application data over the secured channel they have established. All messages sent from client to server and from server to client are encrypted and authenticated using session key.

TLS/SSL Handshake Protocol

- The master secret is derived from (through hashing) the pre-master secret, the “Client Hello” random value, and the “Server Hello” random value
  - Master secret is always 48 bytes
- Normally four symmetric keys and two IVs are derived from the master secret in TLS 1.2

  client_write_MAC_key server_write_MAC_key
  client_write_enc_key server_write_enc_key
  client_write_IV server_write_IV

TLS/SSL Security

- There have been many security flaws in the TLS/SSL protocol and its implementations
- The coming version TLS 1.3 may hopefully contain much less security vulnerabilities

Summary

- Key establishment
  - Key establishment using symmetric key cryptography
    - Kerberos
    - Bellare-Rogaway key establishment scheme
  - Key establishment using public key cryptography
    - SSH
    - PKI, public key certificate: authenticate public keys
      - TLS/SSL