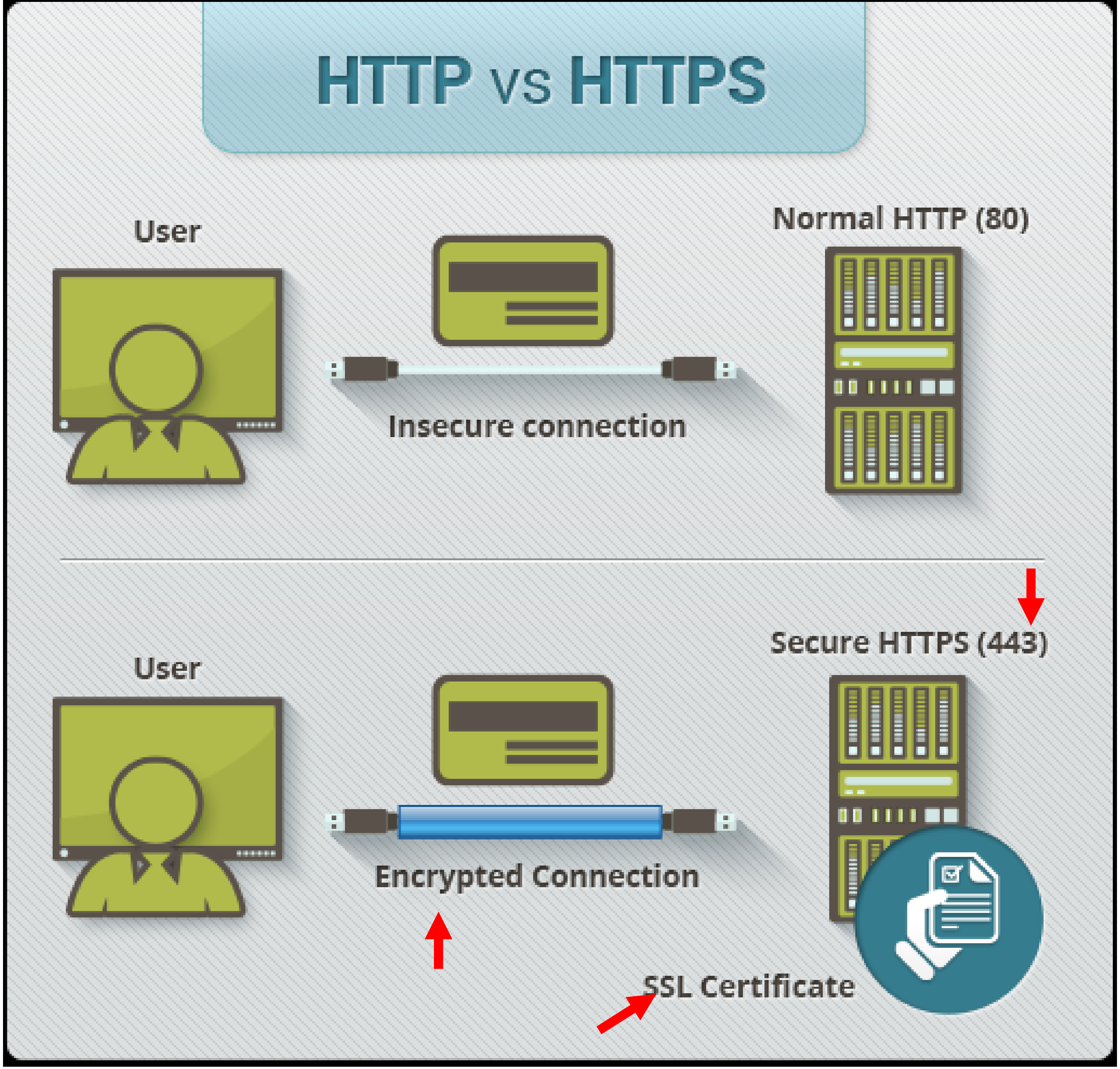
HTTPS (HTTP over SSL or HTTP Secure)

台大資管系 孫雅麗

Fall 2015

HTTPS (HTTP over SSL or HTTP Secure)

'Secure' means **all** communications between browser and website are **encrypted** (and **authenticated**).


HTTP vs HTTPS
User
Normal HTTP (80)
Insecure connection
User
Secure HTTPS (443)
Encrypted Connection
SSL Certificate

HTTPS

- **HTTPS pages** typically use one of two secure protocols to encrypt communications
  - SSL (Secure Sockets Layer) or
  - TLS (Transport Layer Security)

  as a sublayer under regular HTTP application layering.
- SSL is predecessor of TLS.
- Unless a different port is specified, HTTPS uses **port 443** instead of HTTP port 80 in its interactions.

TLS is based on SSL 3.0.

TRANSPORT LAYER SECURITY (TLS)

TLS: Design Goals

- Provide authentication, privacy and data integrity between two communicating applications.
- **Mutual** Server and Client authentications
- An encrypted connection
  - *Confidentiality* and *integrity*
- **Interoperability**
- **Extensibility**
  - *New* public key and encryption methods can be incorporated as necessary.

HTTPS: X.509 Certificates (2/4)

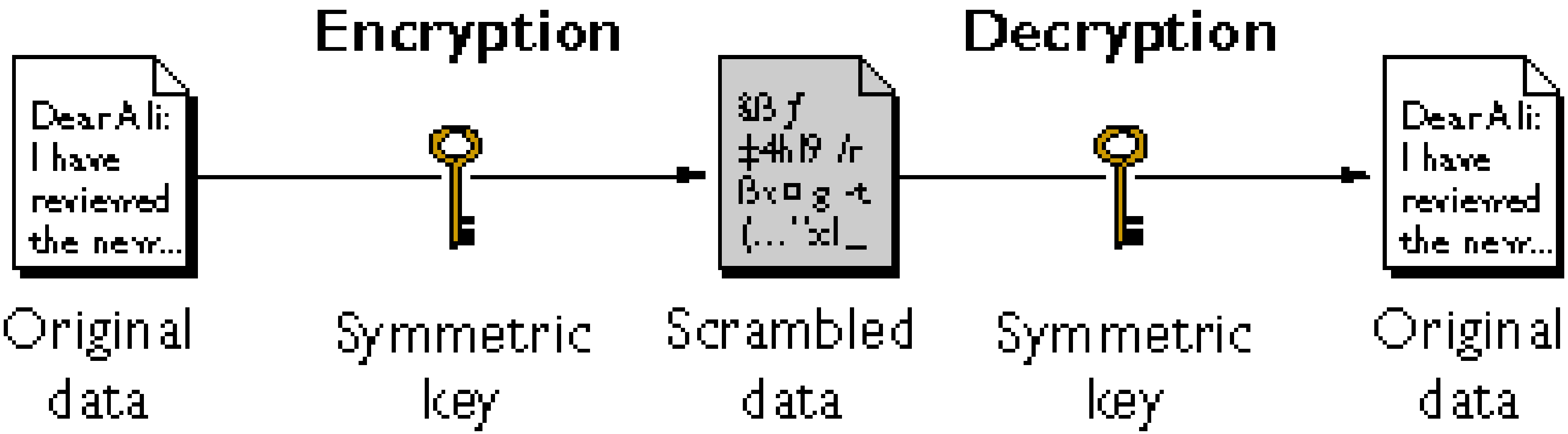
- HTTPS and TLS support the use of X.509 **digital certificates from server** ***for user to authenticate the server***, and to negotiate asymmetric session key for the secure session between them.
- Both the TLS and SSL protocols use an 'asymmetric' Public Key Infrastructure (PKI) system.

X.509

- **Certificate authorities** (CA) and a **public key infrastructure** (PKI) are necessary to verify the relation between a **certificate** and its **owner**, as well as to *generate*, *sign*, and *administer* the **validity** of certificates.
- **Verify the identities** via a web of trust, the 2013 **mass surveillance disclosures** indicated that certificate authorities are a weak point from a security standpoint, allowing man-in-the-middle attacks (MITM).

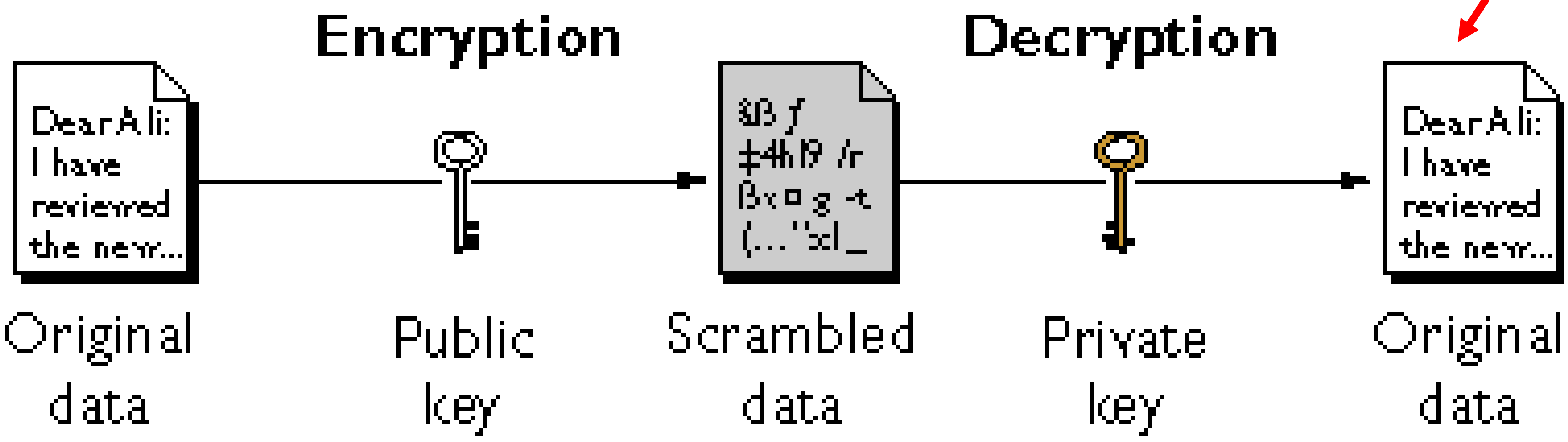
Symmetric Key Encryption

$y = f(x, key)$ $x = f'(y, key)$



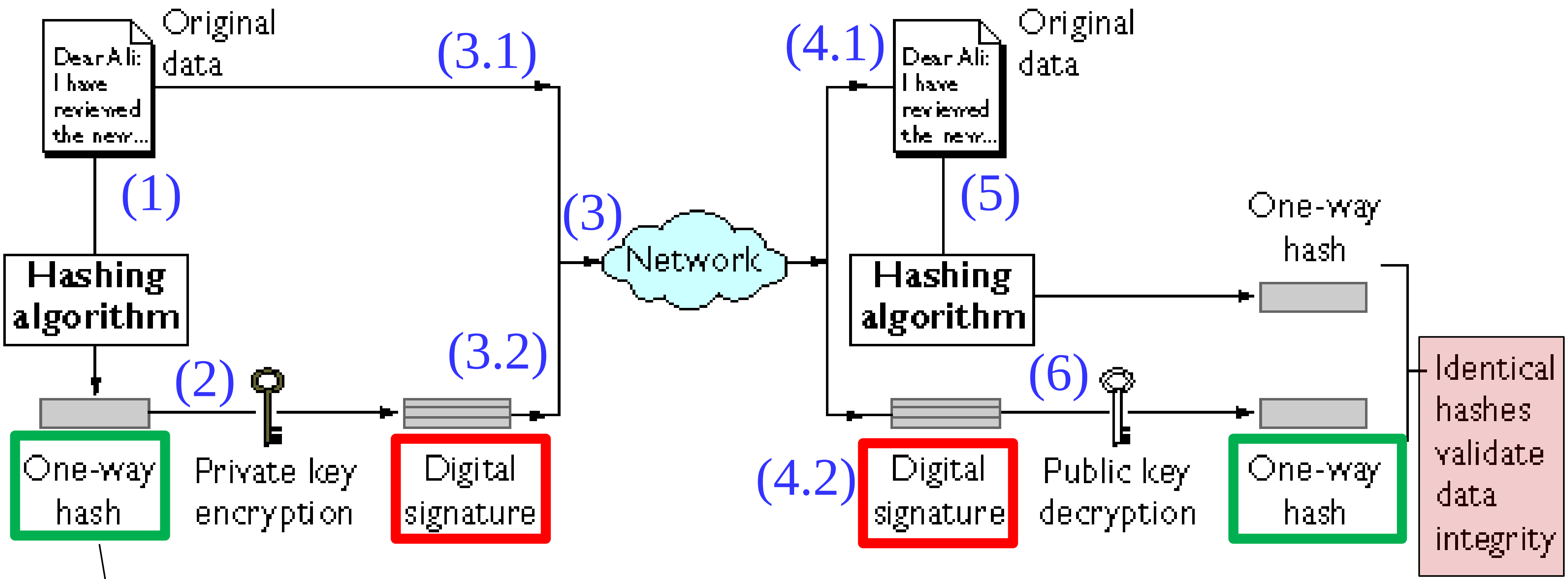
Public Key Encryption (Asymmetric Encryption)

$y = f(x, key_p)$ $x = f'(y, key_s)$



(key_s, key_p)

Digital Signature

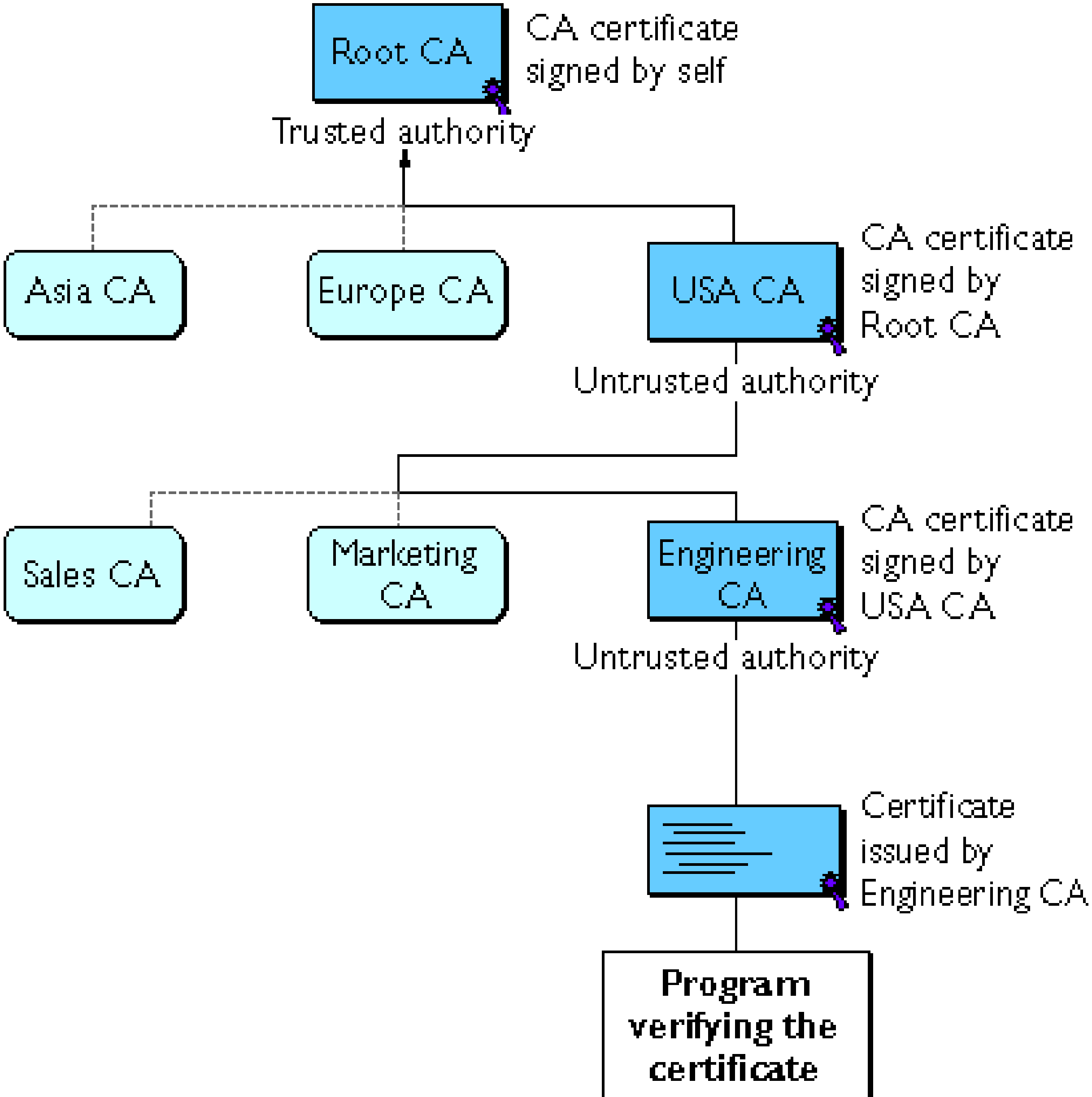


- The value of the hash is unique for the hashed data.
- The content of the hashed data cannot be deduced from the hash.

HTTPS: SSL Certificate (3/4)

- In the case of a website, **server** must first obtain a **SSL Certificate**
  - the **private key** remains *securely* ensconced (or shield) on the web **server**.
  - the **public key** is intended to be *distributed* to anybody and everybody that needs to be able to decrypt information that was encrypted with the private key.

Certificate Chain



HTTPS: Session Key (4/4)

- The **session key** is used to encrypt data flowing between the parties.
- This allows for data/message **confidentiality**, and *message authentication codes* for message **integrity** and as a by-product, **message authentication**.
- The use of HTTPS protects against *eavesdropping* and *man-in-the-middle attacks*.
- HTTPS was developed by Netscape.

Reminder

- HTTPS is *not* to be confused with S-HTTP, a security-enhanced version of HTTP developed and proposed as a standard by EIT.

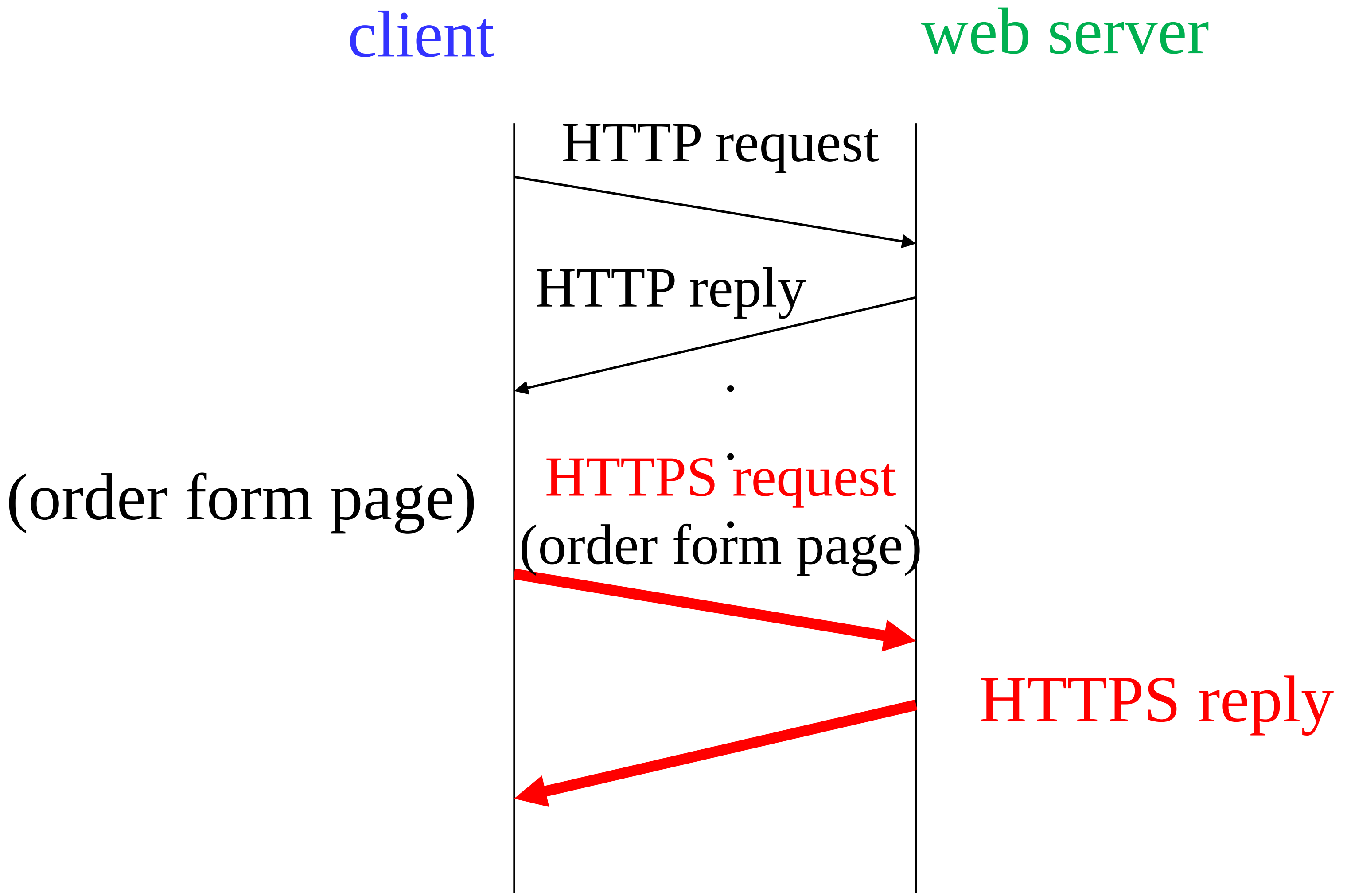
HTTPS: Secure Session Establishment

1. A **client** requests a HTTPS connection **to a webpage**.
2. The website **server** sends its SSL certificate back to client's browser.
   - The certificate contains **server's public key** needed to begin the **secure session**.

- The client's browser and the web server initiate the 'SSL handshake'.
- In the process, **shared secrets** **are generated** to establish a **uniquely** secure connection between the client and the server.


COMODO
SSL Certificate from Com
Comodo CA Ltd [GB] https://www.instantssl.com
SSL Certificate from Com
Comodo CA Ltd [GB] https://www.instantssl.com
SSL Certificate from Como...
Comodo CA Ltd (GB) https://www.instantssl.com
Search
https://www.instantssl.com/
Comodo CA Ltd [GB]
SSL Certificate from Como...
SSL Certificate from Comodo | Secure Socket Layer and More
https://www.instantssl.com/
Comodo CA Ltd
Google
Opera
SSL Certificate from Co...
Trusted
www.instantssl.com
Search with Google

Example Use Scenario (1/2)



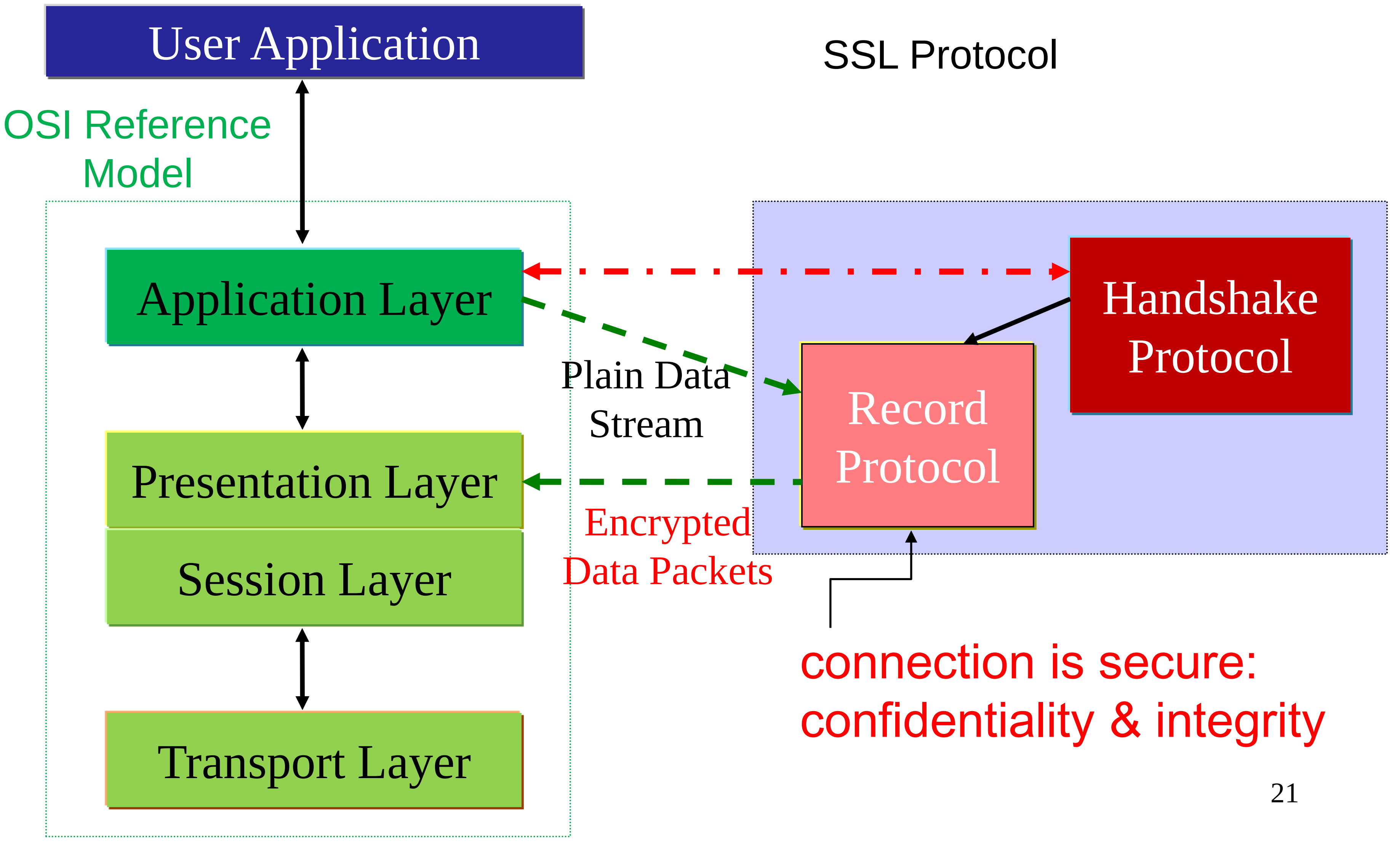
Example Use Scenario (2/2)

- Suppose a client visits a Web site to view their online catalog.
- When given a Web page order form with a Uniform Resource Locator (URL) that starts with https://.
- When client clicks "Send," to send the page back to server, client's browser's HTTPS layer will encrypt it.
- The acknowledgement client receives from the server will also travel in encrypted form, arrive with an https:// URL, and be decrypted by client's browser's HTTPS sublayer.

Transport Layer Security (TLS)

- RFC 5246, **The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.**
- RFC 6176, **Prohibiting Secure Sockets Layer (SSL) Version 2.0, March 2011.**

Transport Layer Security (TLS): Protocol Stack



Protocol Stack

- Handshake Protocol
  - performs ***mutual authentication*** **of server and client**, and
  - **negotiates cryptographic methods to be used**.
- Record Protocol
  - **packetizes data into** ***records***, and
  - performs the agreed encryption/decryption on records

The Record Protocol: two properties (1/2)

- Private (**confidentiality**) and reliable (**integrity**).
- The connection is private.
  - Symmetric cryptography is used for data encryption (e.g., AES [AES]).
  - A symmetric **encryption** key is generated uniquely for **each** connection.
  - The key is based on a secret negotiated by the TLS Handshake Protocol.

The Record Protocol: two properties (2/2)

- The connection is reliable.
  - Use a keyed Message Authentication Code (MAC) **to protect** message integrity.
  - Use **hash** functions (HMAC) (e.g., SHA-1) for MAC computations.

The Record Protocol

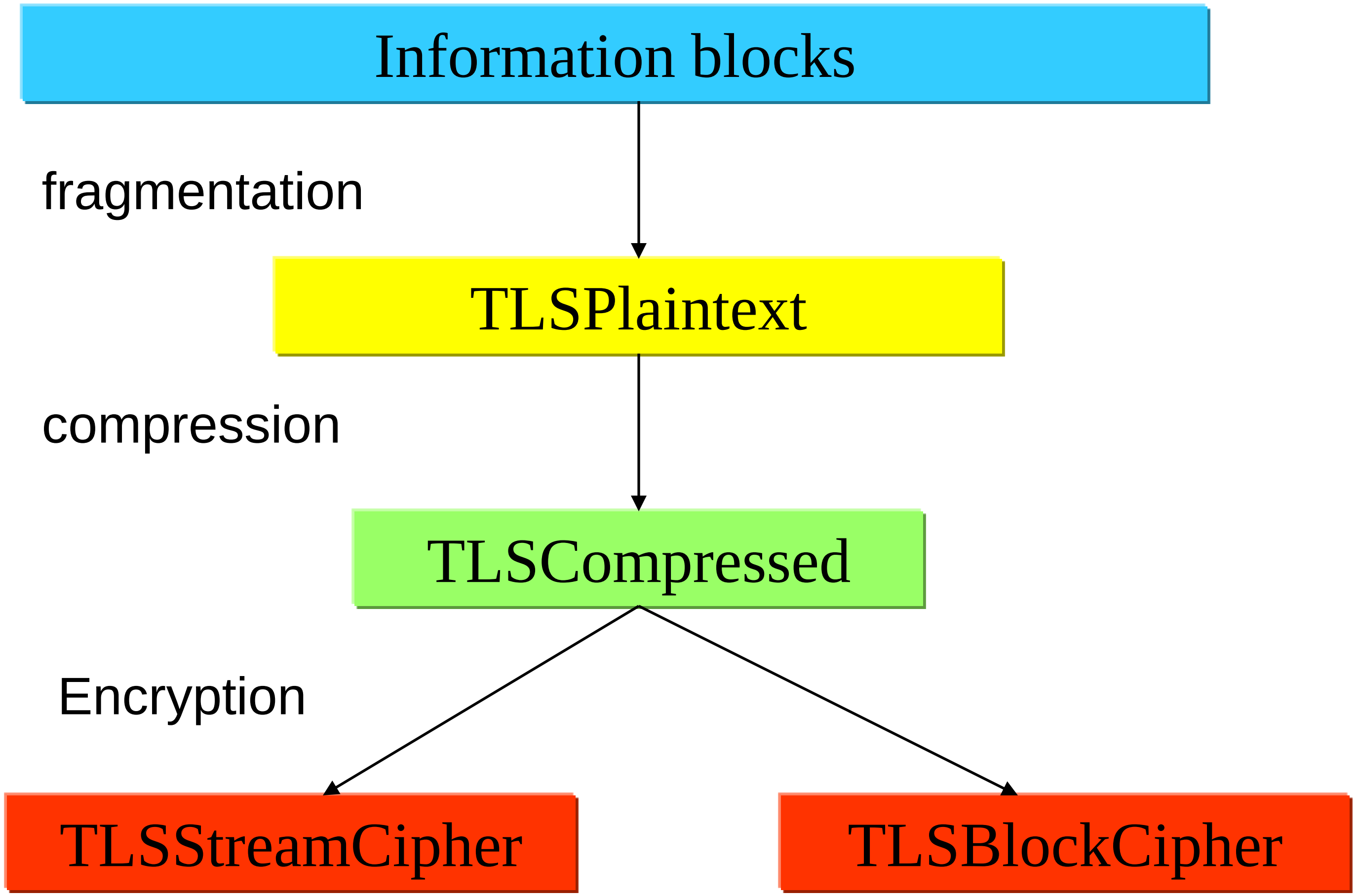
Sender

- Input: messages to be transmitted,
- Processing:
  - fragment the data into **blocks**
  - **compress** the data (option)
  - compute a **MAC** and apply
  - **encrypt**
  - transmit the result.

Receiver

- receive data, ***decrypt, verify, decompress, reassemble,*** and then deliver to higher-level clients

The Record Protocol



TLS Record Protocol: **Connection States**

- The **state** specifies **a compression algorithm, a MAC algorithm and an encryption algorithm**.
- The **parameters** for these algorithms are known: the MAC key and the bulk encryption keys for the connection in both the read and the write directions.
- Logically, there are always **four** connection states outstanding: the **current** read and write states, and the **pending** read and write states.
- All records are processed under the current read and write states.

Four Protocols Use the Record Protocol

- The TLS Record Protocol is used for **encapsulation** of various higher-level protocols.
- The handshake protocol
- The alert protocol
- The change cipher spec protocol
- The application data protocol

TLS Handshake Protocol

When a **TLS client** and **server** first **start communicating,** they

- agree on a **protocol version,**
- negotiate **cryptographic algorithms,**
- **authenticate each other**, and
- use public-key encryption techniques to **generate shared secrets**

before the application protocol transmits or receives its first byte of data.

The TLS Handshake Protocol: three properties

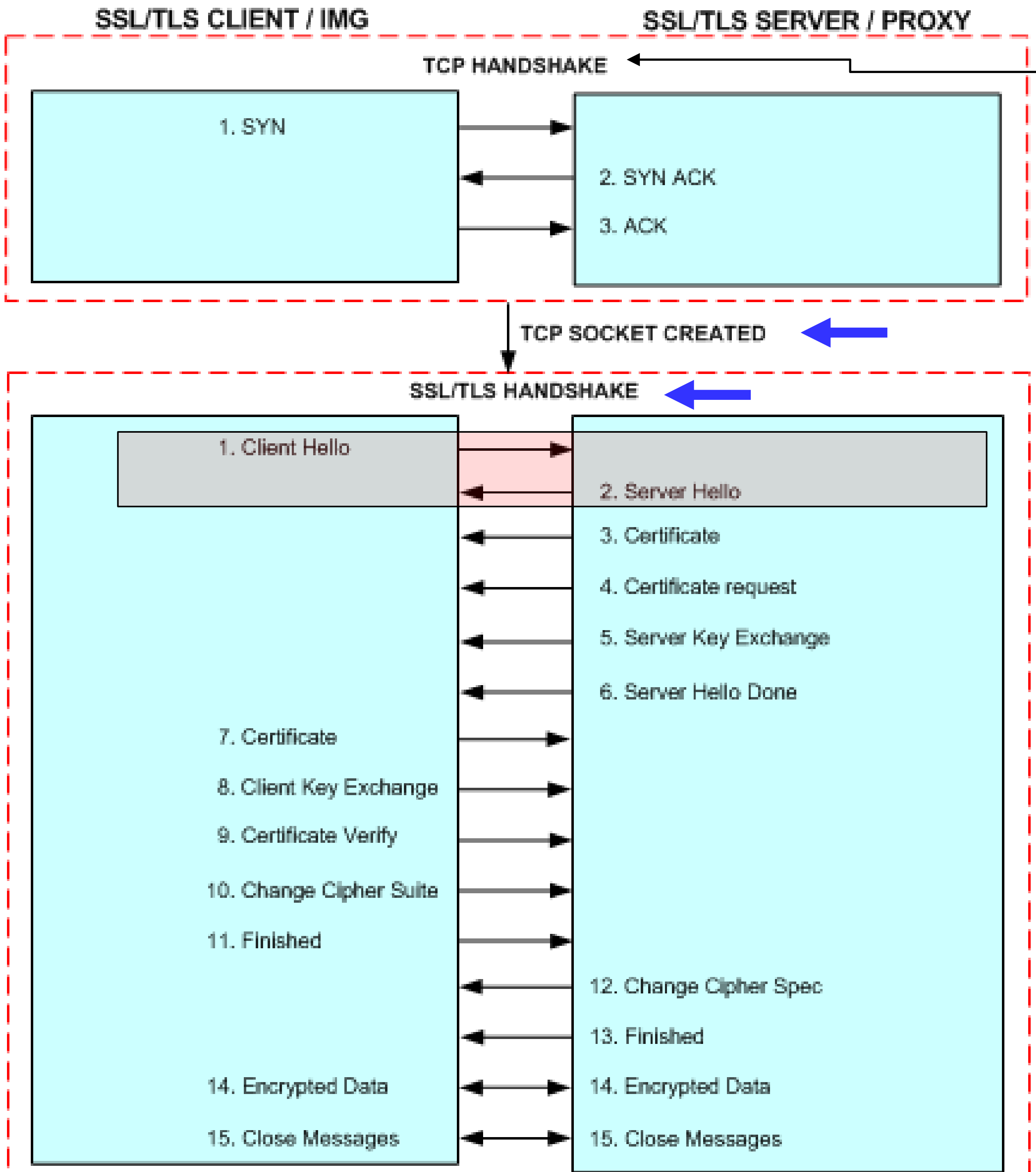
- The peer's identity can be authenticated using **asymmetric,** or public key, cryptography (e.g., RSA, etc.).
  - Optional; but generally required for ***at least one* of the peers**.
- The negotiation of a shared secret is secure.
- The negotiation is reliable. (integrity)
  - No attacker can modify the negotiation communication without being detected by the parties to the communication.

The Handshake Protocol

- It is responsible for **negotiating** a **session and has two sub-protocols.**
- Change Cipher Spec protocol
  - to notify the receiving party that **subsequent** records will be protected under the **newly negotiated CipherSpec and keys.**
- Alert Protocol
  - Alert messages convey the **severity of the message** and **a description of the alert**.
    - Closure alerts
    - Error alerts

#1: The Handshake Protocol

- Authenticate the peer's identity using asymmetric cryptography (e.g., PKI).
- Securely negotiate a shared secret (used by the Record Protocol).



TCP handshake

TLS handshake

Message Flow for A Full Handshake

Client

Server

(1) ClientHello

random$_C$, session_id, **cipher_suites**, compression_methods

(2) ServerHello

random$_S$, session_id, **cipher_suite**, compression_method

(3) ServerCertificate*

certificate_list

(4) ServerKey Exchange*

cryptographic_info (to allow the client to communicate the premaster secret)

(5) Certificate Request*

Certificate_types, certificate_authorities

(6) ServerHello Done

(to indicate the end of the ServerHello and associated messages)

*:optional
{}: encrypted

The Handshake Protocol: Step 1 & 2

1) Exchange **hello messages** to agree on algorithms, **exchange random values**, and check for session resumption.
2) Exchange the necessary **cryptographic parameters** to allow the client and server to agree on a *premaster secret*.
3) Exchange **certificates** and **cryptographic information** to allow the client and server to *authenticate themselves*.
4) Generate a *master secret* from the premaster secret and exchanged random values.
5) Provide security parameters to the record layer.
6) Allow the client and server to **verify** that their peer has calculated the *same* security parameters and that the handshake occurred without tampering by an attacker.

ClientHello and ServerHello

- The ClientHello and ServerHello are used to **establish security enhancement capabilities (attributes)** between client and server.
  - Protocol Version
  - Session ID
  - Cipher Suite
  - Compression Method
  - Two random values: ClientHello.random, ServerHello.random

Session ID (1/2)

- In ClientHello
  - If the session ID field is **empty**, it means the client wants to initialize a **new** session.
  - If not empty, the value identifies a session between the same client and server whose security parameters the client wishes to reuse (i.e., session resumption).

Session ID (2/2)

- In ServerHello
  - If the ClientHello.session_id was **non-empty**, the server will **look in its session cache** for a mach.
  - If a **match** is found, the server will respond with the **same** value as was supplied by the client.
  - Otherwise, this field will contain a different value identifying the **new** session.
  - The server may return an empty session_id to indicate that the session cannot be resumed.

CipherSuite(s)

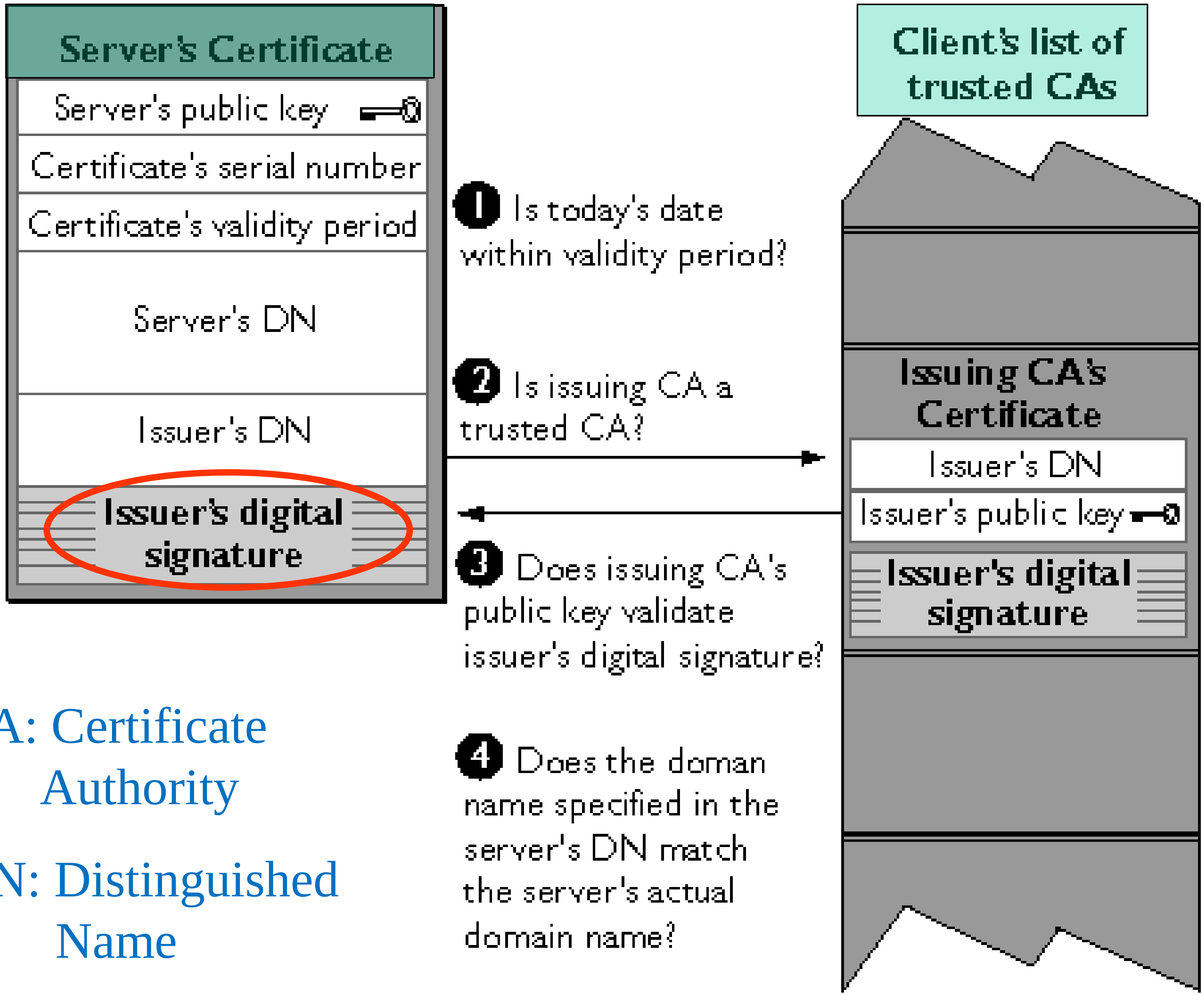
Include

- Key exchange algorithm
- Bulk encryption algorithm (including secret key length)
- MAC algorithm

The Handshake Protocol: Step 3

1) Exchange **hello messages** to agree on algorithms, exchange random values, and check for session resumption.
2) Exchange the necessary **cryptographic parameters** to allow the client and server to agree on a *premaster secret*.
3) Exchange **certificates** and **cryptographic information** to allow the client and server to ***authenticate themselves***.
4) Generate a *master secret* from the premaster secret and exchanged random values.
5) Provide security parameters to the record layer.
6) Allow the client and server to **verify** that their peer has calculated the *same* security parameters and that the handshake occurred without tampering by an attacker.

Server Certification



CA: Certificate Authority

DN: Distinguished Name

Root Certificate Authority (CA)

- Root CA is identified by a **root certificate** which is **an unsigned** or a **self-signed** **public key certificate**.
- A root certificate is part of a public key infrastructure scheme.
- The most common commercial variety is based on the **ITU-T X.509** standard.

臺灣憑證授權 (Certificate Authority, CA) 中心

目前有：

- 政府憑證管理中心 (www.pki.gov.tw)
- 內政部憑證管理中心 (moica.nat.gov.tw)
- 台灣網路認證中心 (www.taica.com.tw)
- 儲匯局電子證書認證中心 (ca.prsb.gov.tw)
- 網際威信公司 (www.hitrust.com.tw)

The Handshake Protocol: Step 4

1) Exchange **hello messages** to agree on algorithms, exchange random values, and check for session resumption.
2) Exchange the necessary **cryptographic parameters** to allow the client and server to agree on a ***premaster secret***.
3) Exchange **certificates** and **cryptographic information** to allow the client and server to *authenticate themselves*.
4) Generate a ***master secret*** from **the premaster secret and exchanged random values**.
5) Provide security parameters to the record layer.
6) Allow the client and server to **verify** that their peer has calculated the *same* security parameters and that the handshake occurred without tampering by an attacker.

Client Key Exchange

- With this message, the **premaster secret** is set, either through
  - transmission of the **RSA-encrypted secret (using the public key from the server's certificate)**, or
  - **Diffie-Hellman** parameters which will allow each side to agree upon the **same** premaster secret.

pre_master_secret

- The `pre_master_secret`
  - If from **key agreement,** then the `pre_master_secret` is the result of Diffie-Hellman key agreement.
  - If the `pre_master_secret` comes from a **key transport scheme**, then the **client encrypts a random value under the server's public key**.
    - In this scheme, only the client provides keying material.
    - When only one party provides the key, its called a key transport scheme.

Master Secret

- The `master_secret` is a common secret shared by the client and server. It is used to derive session specific keys.
- master_secret = PRF(pre_master_secret,
  "master secret",
  ClientHello.random,
  ServerHello.random)
  - PRF() : pseudo random function

In ClientKeyExchange

In ClientHello

In ServerHello

The pseudorandom function (PRF) based on HMAC

1. Take a **secret,** a **seed,** and an **identifying label** as input
2. Produce an output of arbitrary length.

- Use such as SHA-256 for a stronger standard hash function.
- The **SHA** (Secure Hash Algorithm) is one of a number of cryptographic hash functions.

Key Material

In ClientKeyExchange

- Key_material = PRF(master_secret,
  "key expansion",
  ClientHello.random,
  ServerHello.random)

In ClientHello

In ServerHello

Key Calculation

- There are 6 each secrets derived from the `master_secret`:
  - Client encryption key (Client_write_key)
  - Server encryption key (Server_write_key)
  - Client MAC key (Client_write_MAC_secret)
  - Server MAC key (Server_write_MAC_secret)
  - Client IV (Client_write_IV)
  - Server IV (Server_write_IV)

Key Calculation

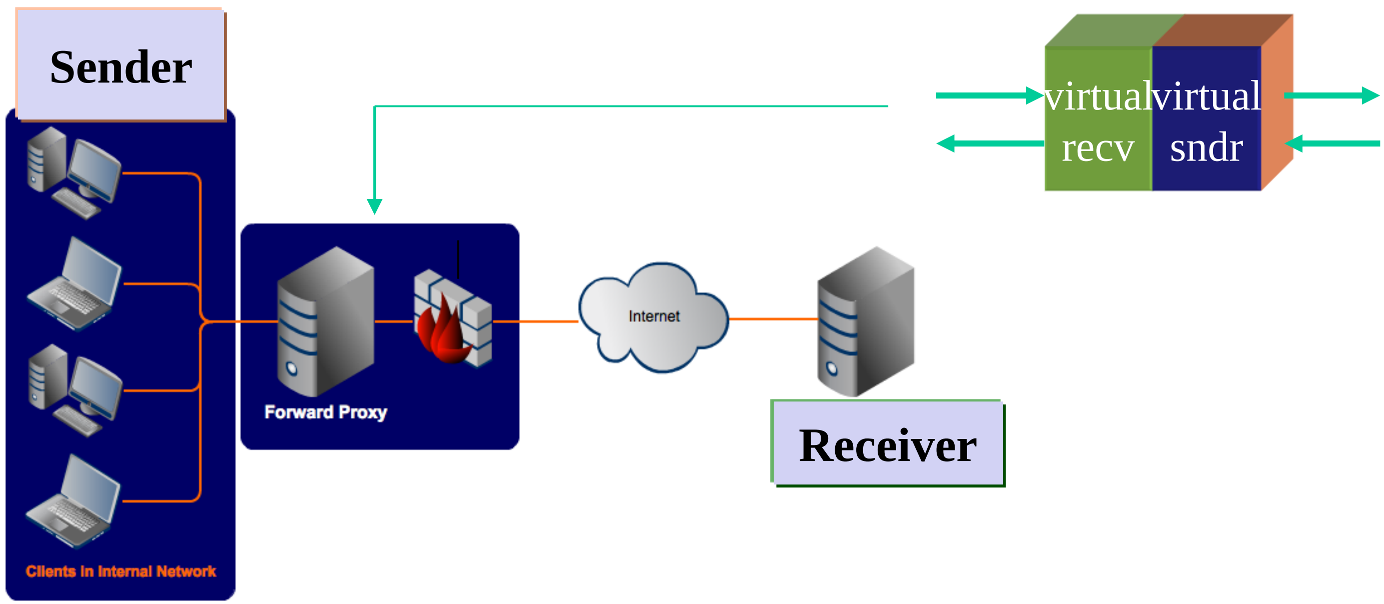
- Usage of different keys and secrets
  - **MAC_secret is used generate MAC of a packet.**
    - MAC is computed before encryption.
  - **Write_key is used for data encryption.**
    - The cipher encrypts the entire block, including the MAC.
  - Write_IV (initialization vector) is only generated for non-export block ciphers.

SSL: Potential Threats

- Today, SSL traffic accounts for 25% to 35% of all Internet traffic.
- Attackers can simply tunnel attacks in SSL traffic to circumvent defenses.

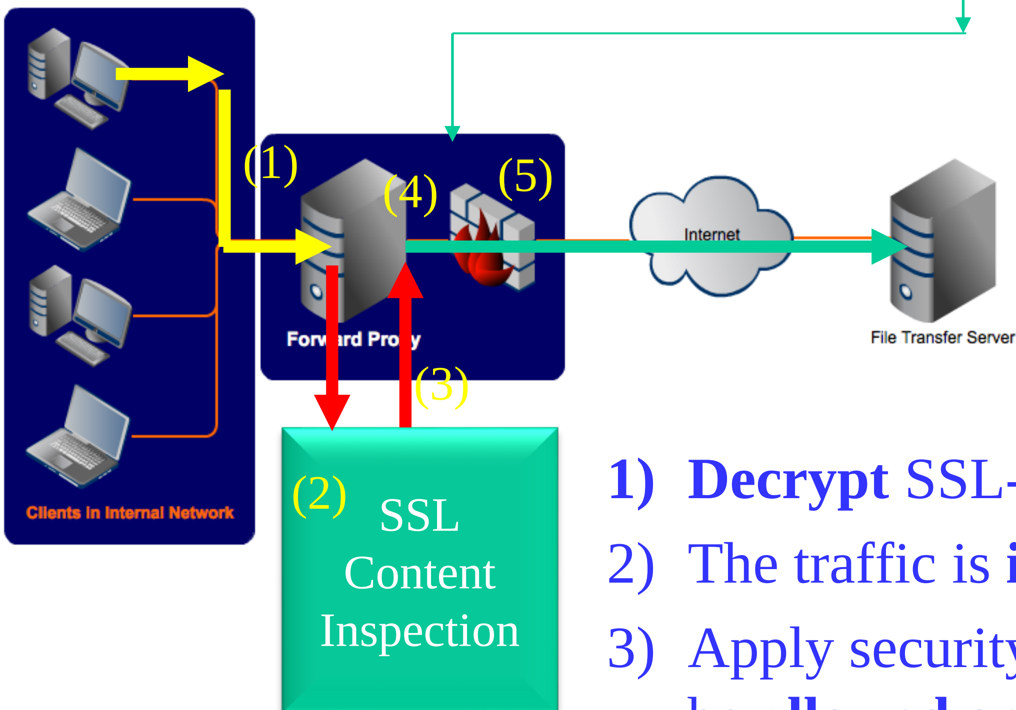
➢ Want to be able to **decrypt inbound and outbound SSL traffic** in firewall, IPS, UTM (unified threat management).

SSL Forward Proxy (1/3)



- A forward proxy is typically used **in tandem with a firewall** to enhance an internal network's security
- It **controls traffic originating from clients in the internal network to hosts on the Internet.**

SSL Forward Proxy (2/3)



- An SSL forward proxy consists of **two** SSL termination devices that have **separate secured sessions between server and client**.
- From the point of view of the web server, **it is the proxy server that issued the request**, not the client.
- Hence, the server **addresses its response to the proxy**.

1) **Decrypt** SSL-encrypted traffic;
2) The traffic is **inspected** and **analyzed**.
3) Apply security policy, an HTTP request can be **allowed** or *denied*.
4) The traffic, possibly scrubbed, is **encrypted** and forwarded to the intended destination.

The SSL Forward Proxy Server (3/3)

- NAT+application-level security control (e.g., A10 Thunder application delivery control SSL Insight)
- It can serve as a **single point of access and control**, making it easier for a corporate to enforce security policies.
- The proxy servers can **keep track of requests, responses**, and their **sources** and their **destinations**.

The end. ☺