NYU TANDON SCHOOL OF ENGINEERING

Network Security

CS 6823 – Lecture 6
Message Integrity, PKI, SSL/TLS

Phillip Mak
pmak@nyu.edu

Lesson Objectives

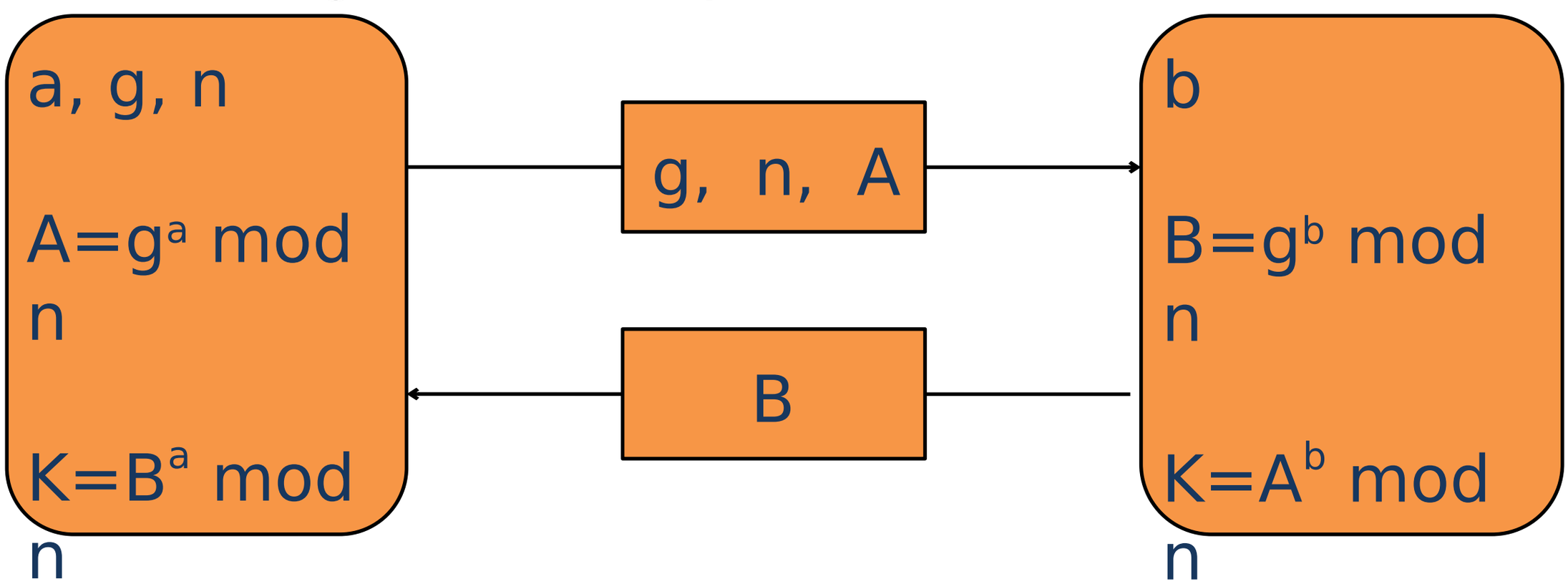
- Understand how nonces improves authentication
- Understand how RSA, DH, AES are the backbone of Internet communications
- Apply modern crypto concepts to solve scenarios
- Understand how PKI certificates are issues and used
- Explain Perfect Forward Security
- Understand protocol level details of a TLS connection
- Identify vulnerabilities and issues in PKI/TLS systems

NYU TANDON SCHOOL OF ENGINEERING

Protecting Diffie-Hellman Exchanges

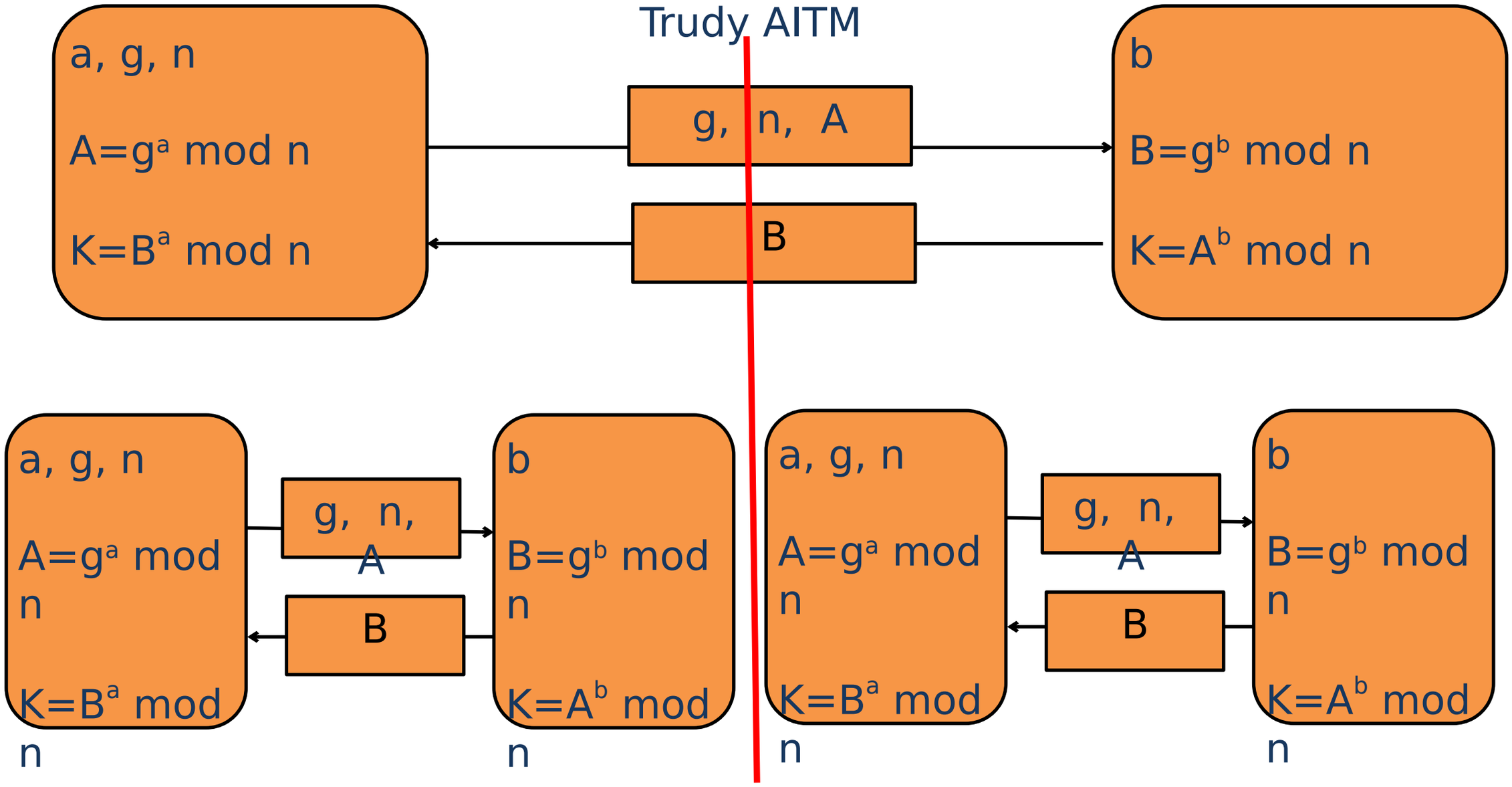
Diffie-Hellman

- Allows two entities to agree on shared key.
  - But does not provide encryption
- n is a large prime; g is a number less than n.
  - n and g are made public



DH Needs to be Protected

- The public values (g, n, A, B) are modified, then the key K can be modified



NYU TANDON SCHOOL OF ENGINEERING

Message Integrity and Digital Signatures

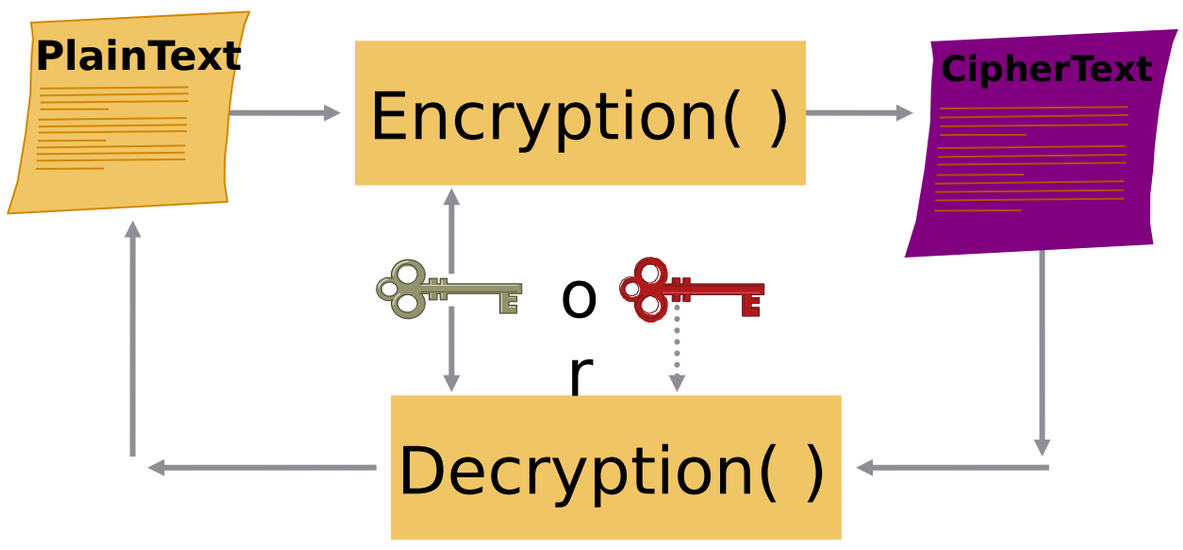
Message Integrity

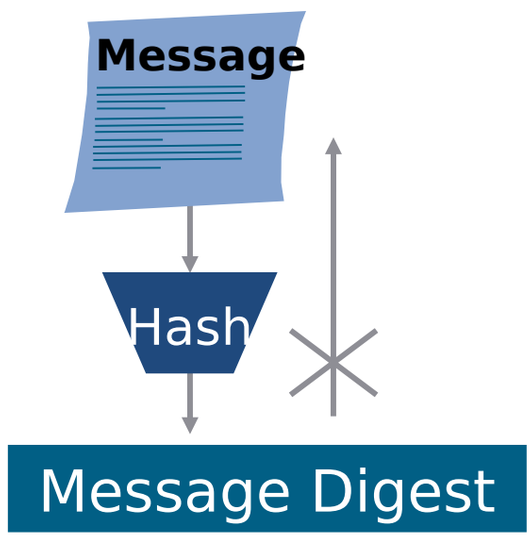
- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been artificially delayed (playback attack)
  - Sequence of messages is maintained

- Let's first talk about message digests

NYU TANDON SCHOOL OF ENGINEERING

Encryption vs. Hashing



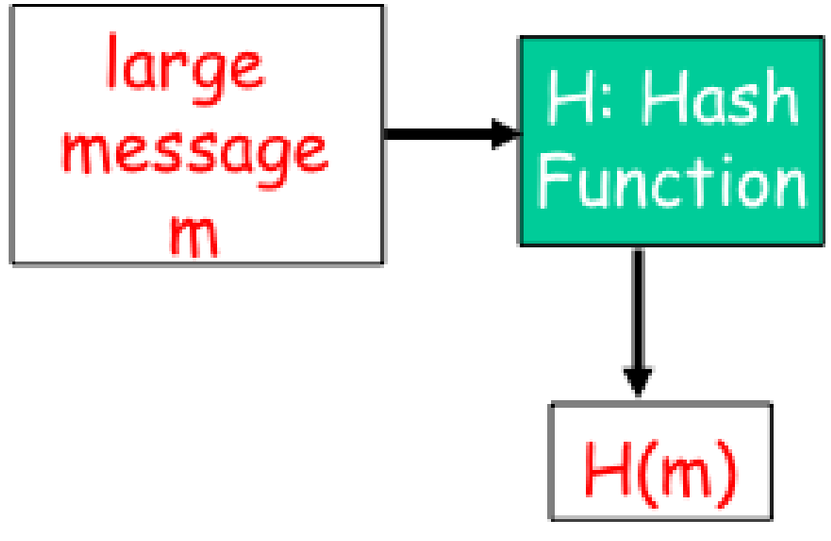


- Encryption keeps communications private
- Encryption and decryption can
  use same or different keys
- Achieved by various algorithms,
  e.g. DES, CAST
- Need key management

- Hash transforms message into fixed-size string
- One-way hash function
- Strongly collision-free hash
- Message digest can be viewed as “digital fingerprint”
- Used for message integrity check and digital certificates
- Hash is generally faster than encryption

Message Digests

- Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: "message signature"
- Note that H( ) is a many-to-1 function
- H( ) is often called a "hash function"



- Desirable properties:
  - Easy to calculate
  - Irreversibility
  - Collision resistance: Computationally difficult to produce m and m' such that H(m) = H(m')
  - Seemingly random output

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - –computes 128-bit message digest in 4-step process.
  - –Usually represented as 32 HEX digits
- SHA-1 is also used (sometimes just called SHA)
  - –US standard [NIST, FIPS PUB 180-1]
  - –160-bit message digest

```
kobrien-laptop:~ kobrien$ echo "test" | md5sum
d8e8fca2dc0f896fd7cb4cb0031ba249  -

kobrien-laptop:~ kobrien$ echo "test" | md5sum
d8e8fca2dc0f896fd7cb4cb0031ba249  -

kobrien-laptop:~ kobrien$ echo "test1" | md5sum
3e7705498e8be60520841409ebc69bc1  -

kobrien-laptop:~ kobrien$ echo "test1" | md5sum
3e7705498e8be60520841409ebc69bc1  -
```

Commonly Used Hash Functions (MD5 and SHA)

- Both MD5 and SHA are derived based on MD4

- MD5 provides 128-bit output, SHA provide 160-bit output, (only first 96 bits used in IPSec)

- Both of MD5 and SHA are considered one-way strongly collision-free hash functions

- SHA is computationally slower than MD5, but more secure

- ***MD5, SHA1 not collision resistant***

  Relevance to non-repudiation, commitment

So What Does This Mean?

- SHA1 is still much safer than MD5
  - Best known attack has effort > 2^64
- HMAC SHA1 (keyed SHA1) believed to be unaffected by current attacks
- Industry making a move towards SHA256 (SHA2) and other secure crypto methods
- Actual transition will take place within standard groups first
  - IETF and NIST among others addressing this issue

Security Level of Crypto Algorithms

Security Level	Work Factor	Algorithms
Weak	$O(2^{40})$	DES, MD5
Legacy	$O(2^{64})$	RC4, SHA1
Minimum	$O(2^{80})$	3DES, SEAL, SKIPJACK, RSA-1024, DH-1024
Standard	$O(2^{128})$	AES-128, SHA-256, RSA-2048, DH-2048
High	$O(2^{192})$	AES-192*, SHA-384
Ultra	$O(2^{256})$	AES-256, SHA-512 RSA-4096, DH-4096

*AES-192 is not used in practice

Key Size requirements according to NIST SP800-57r5

Table 2: Comparable security strengths of symmetric block cipher and asymmetric-key algorithms

Security Strength	Symmetric Key Algorithms	FFC (DSA, DH, MQV)	IFC* (RSA)	ECC* (ECDSA, EdDSA, DH, MQV)
≤ 80	2TDEA	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA[68]	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Before 2030

After 2030

Table 3: Maximum security strengths for hash and hash-based functions

Security Strength	Digital Signatures and Other Applications Requiring Collision Resistance	HMAC,[70] KMAC,[71] Key Derivation Functions,[72] Random Bit Generation[73]
≤ 80	SHA-1[74]	
112	SHA-224, SHA-512/224, SHA3-224	
128	SHA-256, SHA-512/256, SHA3-256	SHA-1, KMAC128
192	SHA-384, SHA3-384	SHA-224, SHA-512/224, SHA3-224
≥ 256	SHA-512, SHA3-512	SHA-256, SHA-512/256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512, KMAC256

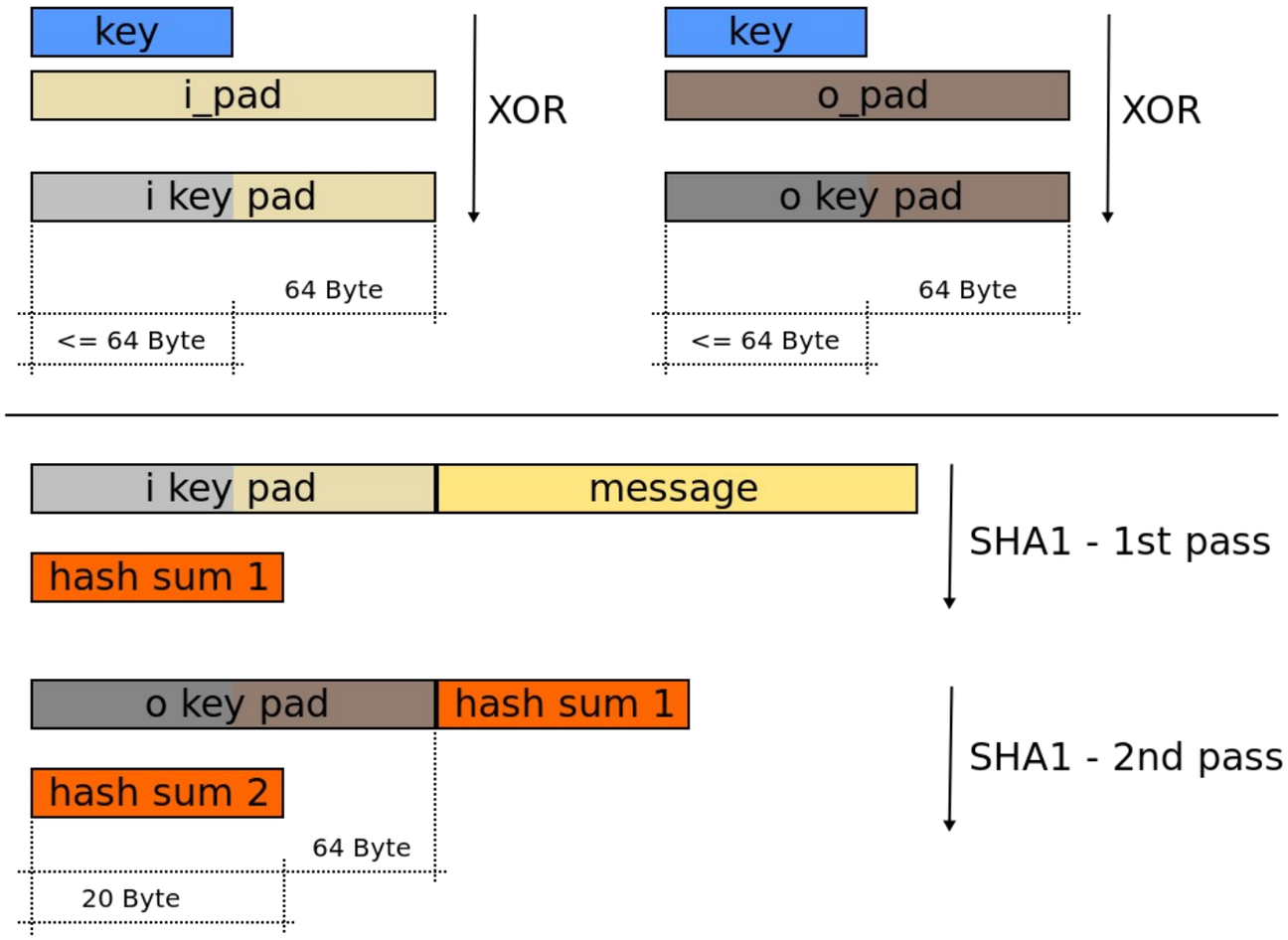
Post-Quantum Cryptography (PQC)

- NIST quantum-resistant cryptographic algorithms

Public-key Encryption and Key-establishment Algorithms
ML-KEM (CRYSTALS-KYBER)
HQC

Digital Signature Algorithms
ML-DSA (CRYSTALS-DILITHIUM)
SLH-DSA (SPHINCS+)

Hash-Based Message Authentication Code (HMAC)



$$HMAC(K, m) = H\Big((K \oplus opad) \;||\; H\big((K \oplus ipad) \;||\; m\big)\Big)$$

- *opad* and *ipad* are constants
- ***Authenticates sender***
- ***Verifies message integrity***
- No encryption!
- Also called “keyed hash”

You do not need to memorize HMAC algorithm

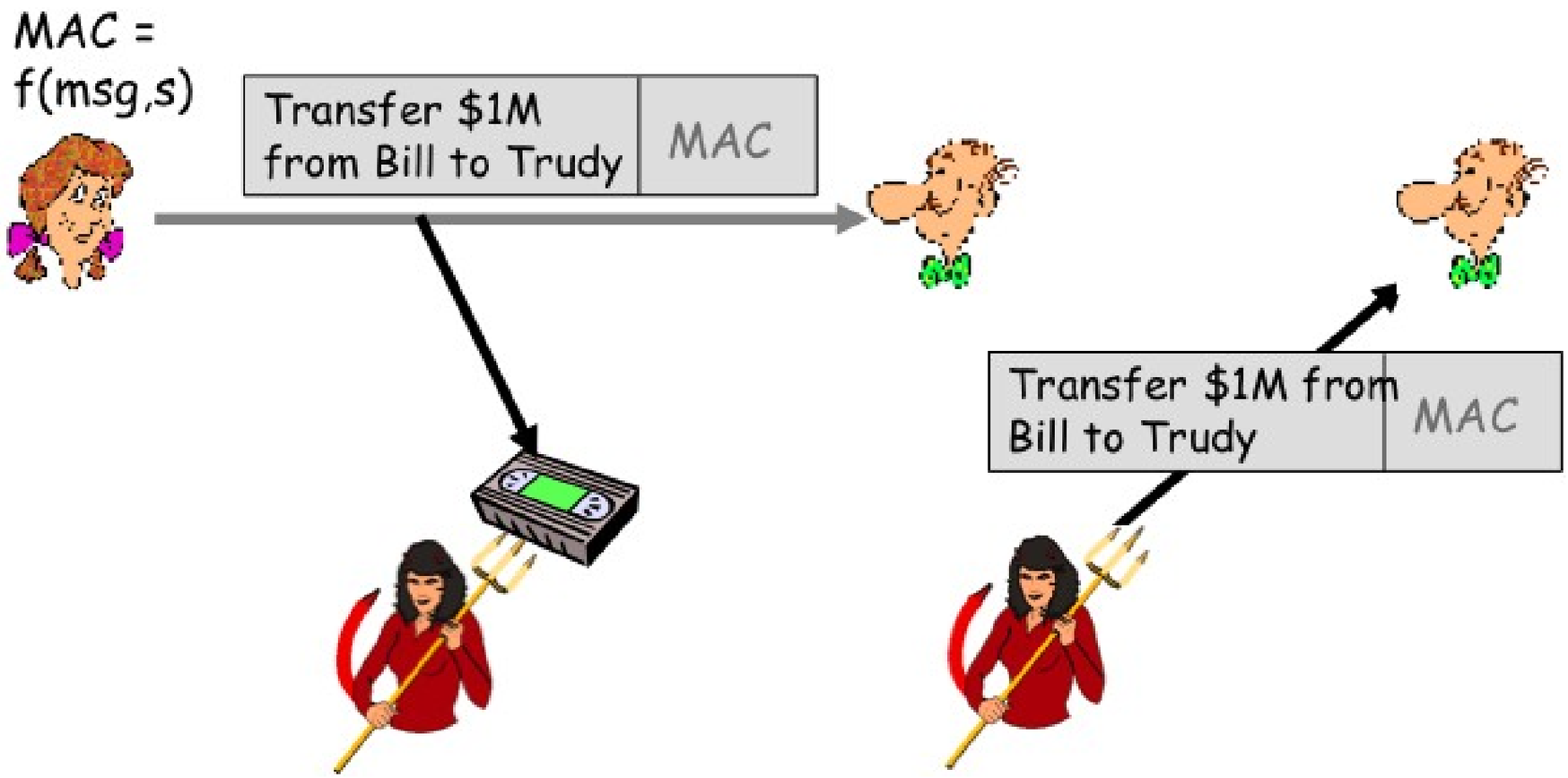
Source: Wikipedia

End Point Authentication

- Want to be sure of the originator of the message – *end-point authentication*.

- Assuming Alice and Bob have a shared secret, will MAC provide message authentication.
  - We do know that Alice created the message.
  - But did she send it?

Playback Attack



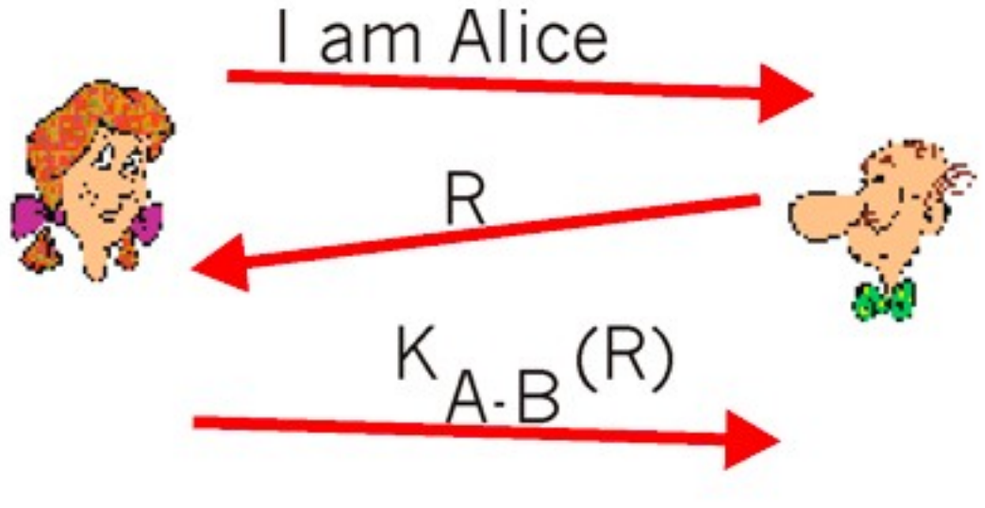
Bob cannot distinguish between the original communication and the later playback

Problem is that the shared secret is used over and over

Defending Against Playback Attack: Nonce

1) Alice sends the message, "I am Alice," to Bob

2) Bob chooses a nonce, *R*, and sends it to Alice



3) Alice encrypts the nonce using Alice and Bob's symmetric secret key, K_{A-B}, and sends the encrypted nonce, $K_{A-B}(R)$ back to Bob.

A nonce is a number that a protocol will only ever use once-in-a-lifetime

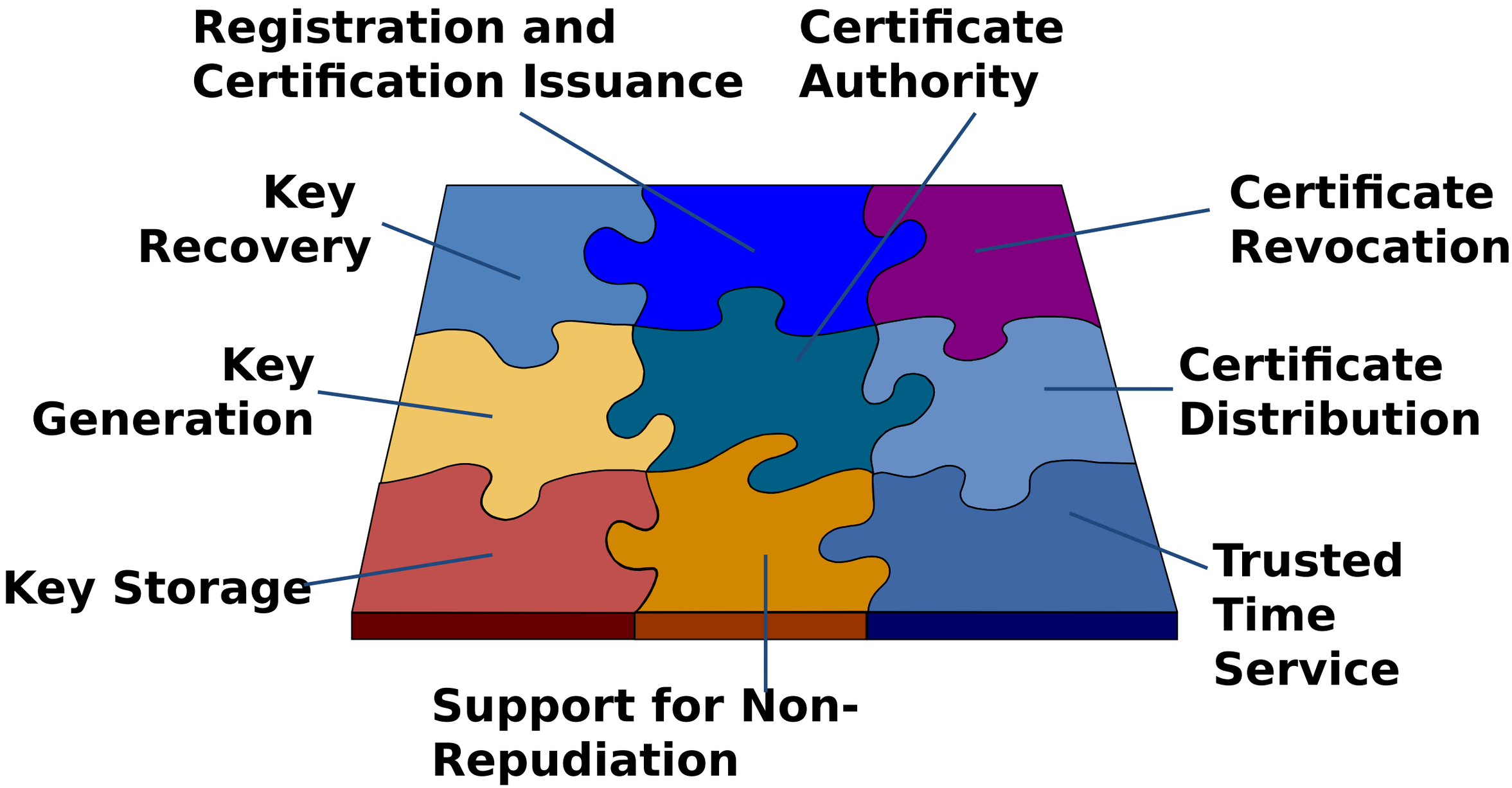
Nonce (con't)

- *It is the fact that Alice knows K_{A-B} and uses it to encrypt a value that lets Bob know that the message he receives was generated by Alice.*

- *The nonce is used to ensure that Alice is "live." Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then Alice is authenticated.*

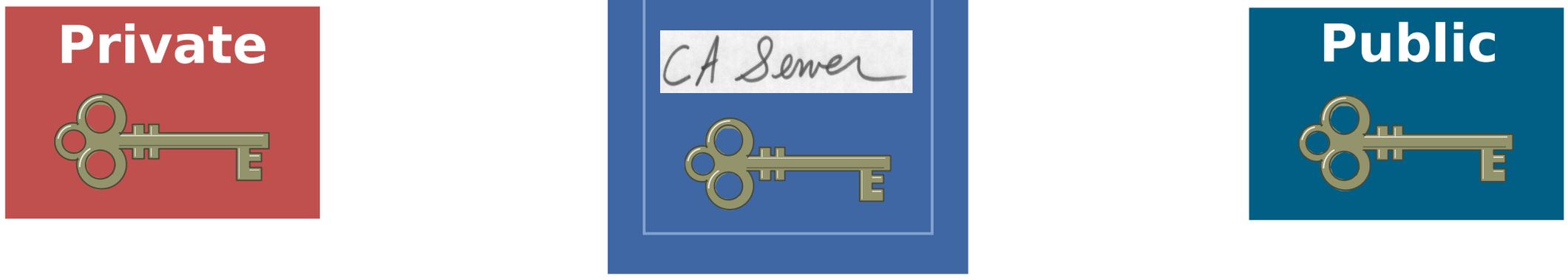
Public Key Infrastructure (PKI)

PKI: IKE Authentication Architecture



PKI is an infrastructure for numerous services that work together to enable interoperability of digital

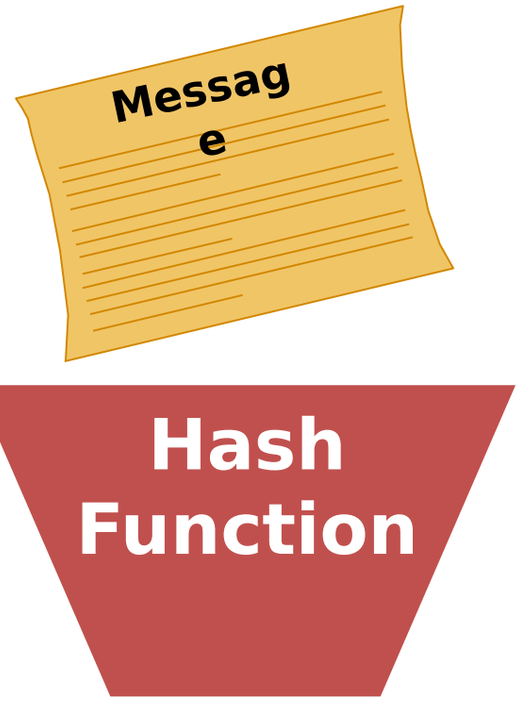
Digital Signatures



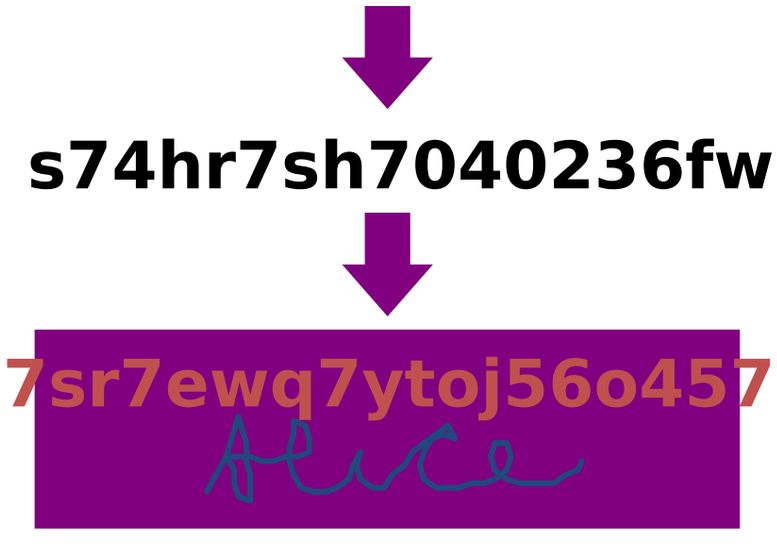
- Entity authentication
- Data origin authentication
- Integrity
- Non-repudiation

Digital Signatures

One-Way Function; Easy to Produce Hash from Message, “Impossible” to Produce Message from Hash



Hash of Message



Sign Hash with Private Key

Signature = “Encrypted” Hash of Message

NYU TANDON SCHOOL OF ENGINEERING

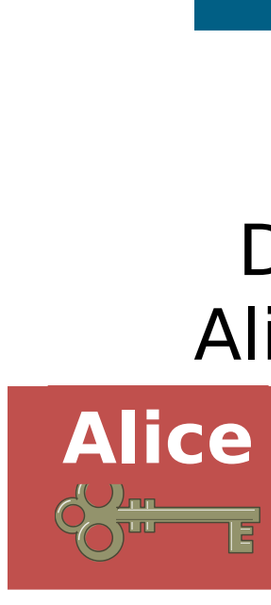
Signature Verification

Decrypt the Received Signature

Signature

Decrypt Using Alice's Public Key

Alice



Hash of Message

Signature

Message with Appended Signature

Re-Hash the Received Message

Hash Function

Hash Message

If Hashes Are Equal, Signature Is Authentic

Digital Signatures (more)

Alice thus verifies that:

- Bob signed m.
- No one else signed m.
- Bob signed m and not m’.

Non-repudiation:

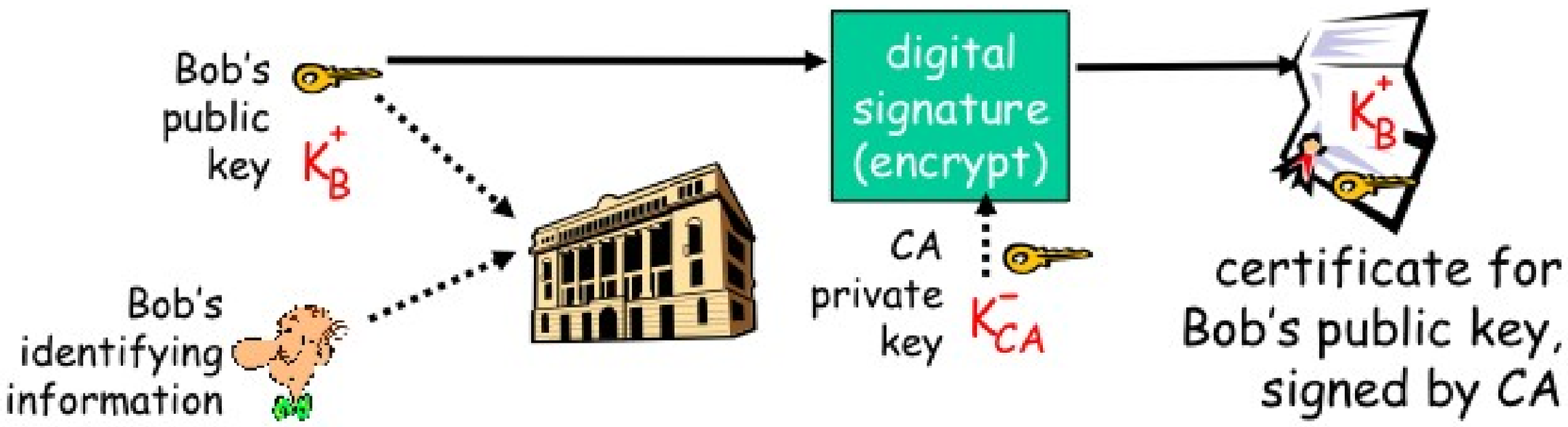
- Alice can take m, and signature $K_B(m)$ to court and prove that Bob signed m.
  - Not really to court; current laws don’t allow that
- Doesn’t stop Trudy from replaying message m

Public Key Certifcation

- Motivation: Trudy plays pizza prank on Bob
  - –Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - –Trudy signs order with her private key
  - –Trudy sends order to Pizza Store
  - –Trudy sends to Pizza Store her public key, but says it's Bob's public key.
  - –Pizza Store verifies signature; then delivers four pizzas to Bob.
  - –Bob doesn't even like Pepperoni

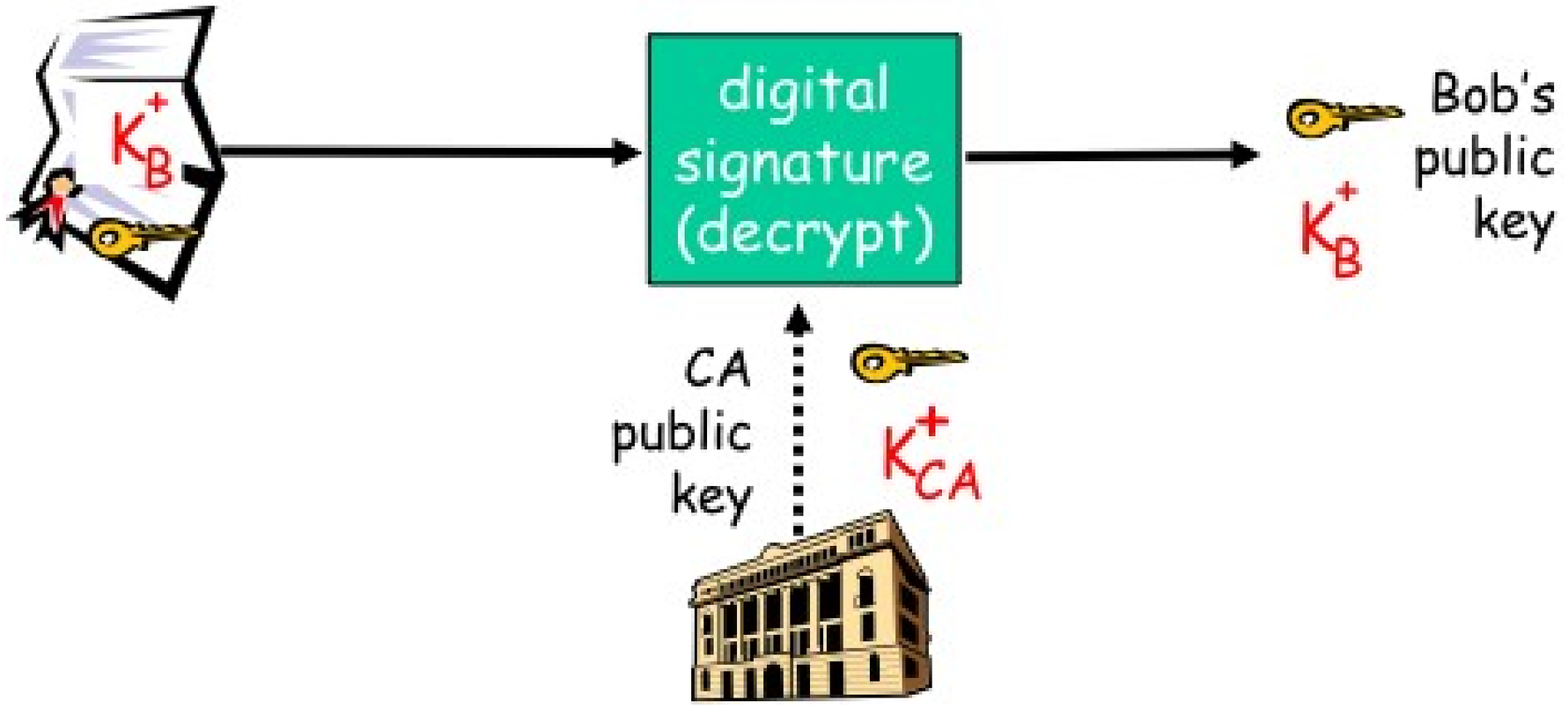
Certificate Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



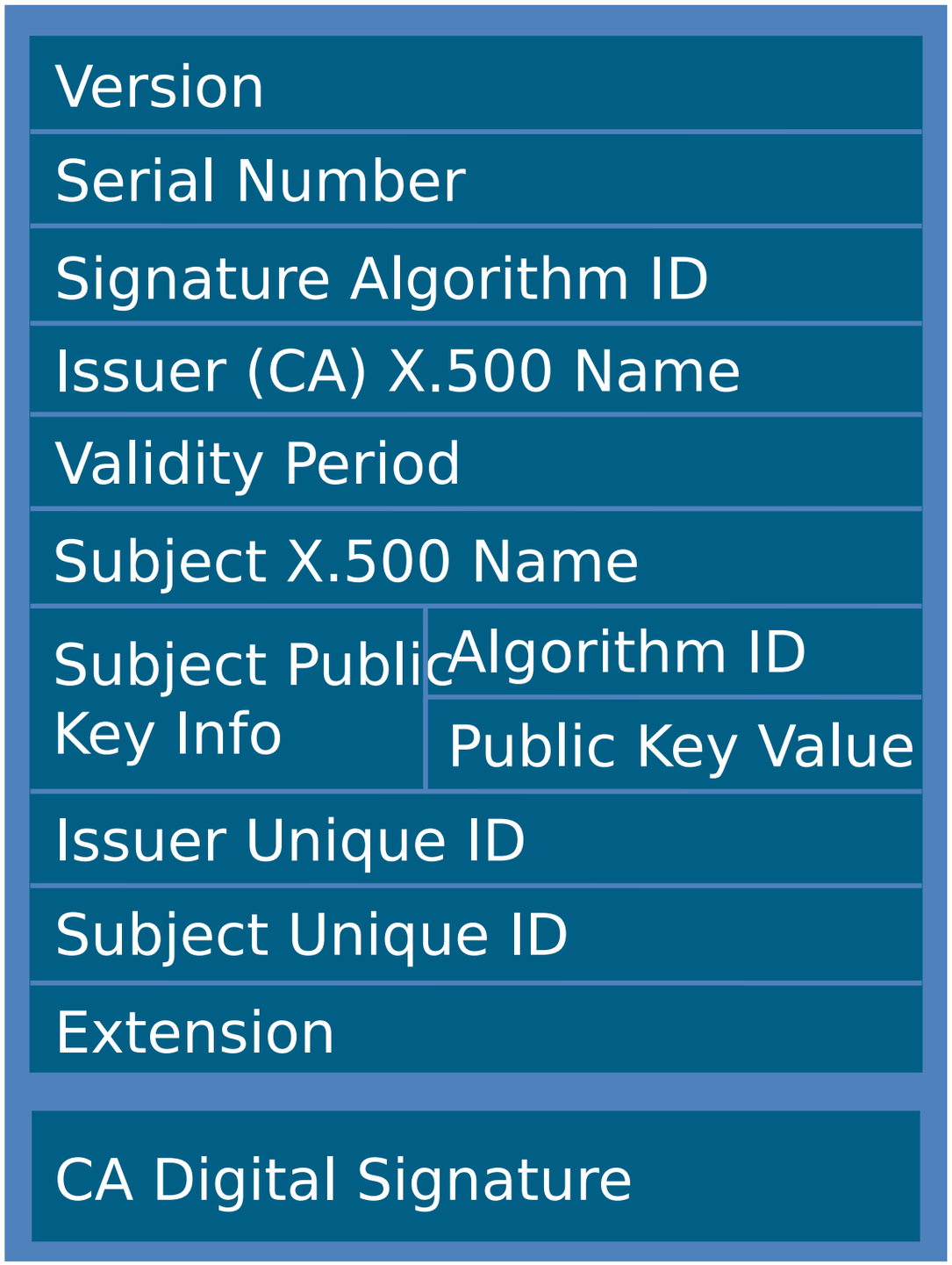
Certificate Authorities

- When Alice wants Bob's public key:
  - –gets Bob's certificate (Bob or elsewhere).
  - –apply CA's public key to Bob's certificate, get Bob's public key



NYU TANDON SCHOOL OF ENGINEERING

X.509 v3 Certificate



Signing Algorithm, e.g. SHA1withRSA

CA's Identity

Lifetime of this Cert

User's Identity, e.g. cn, ou, o

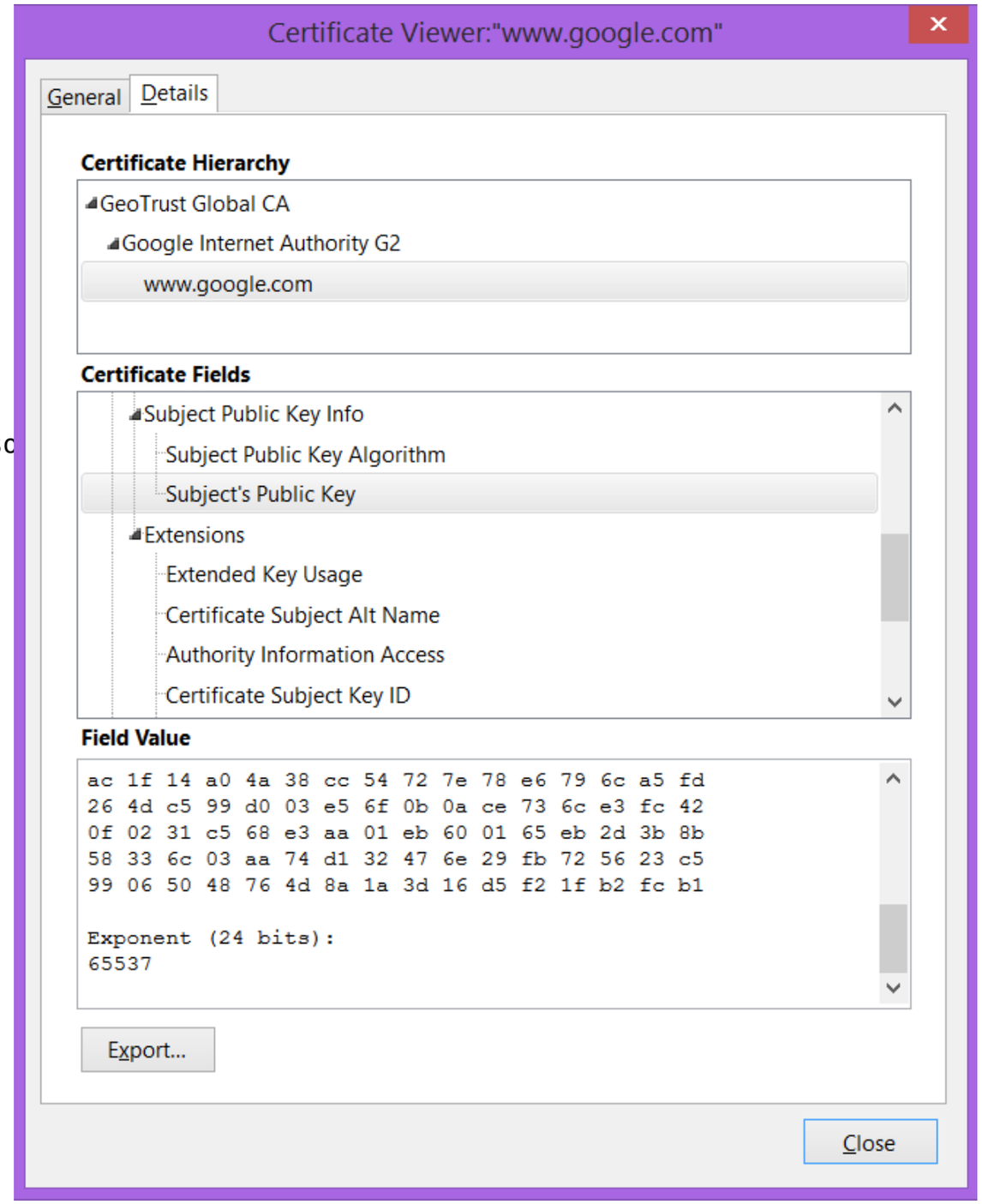
User's Public Key (Bound to User's Subject Name)

Other User Info, e.g. subAltName, CDP

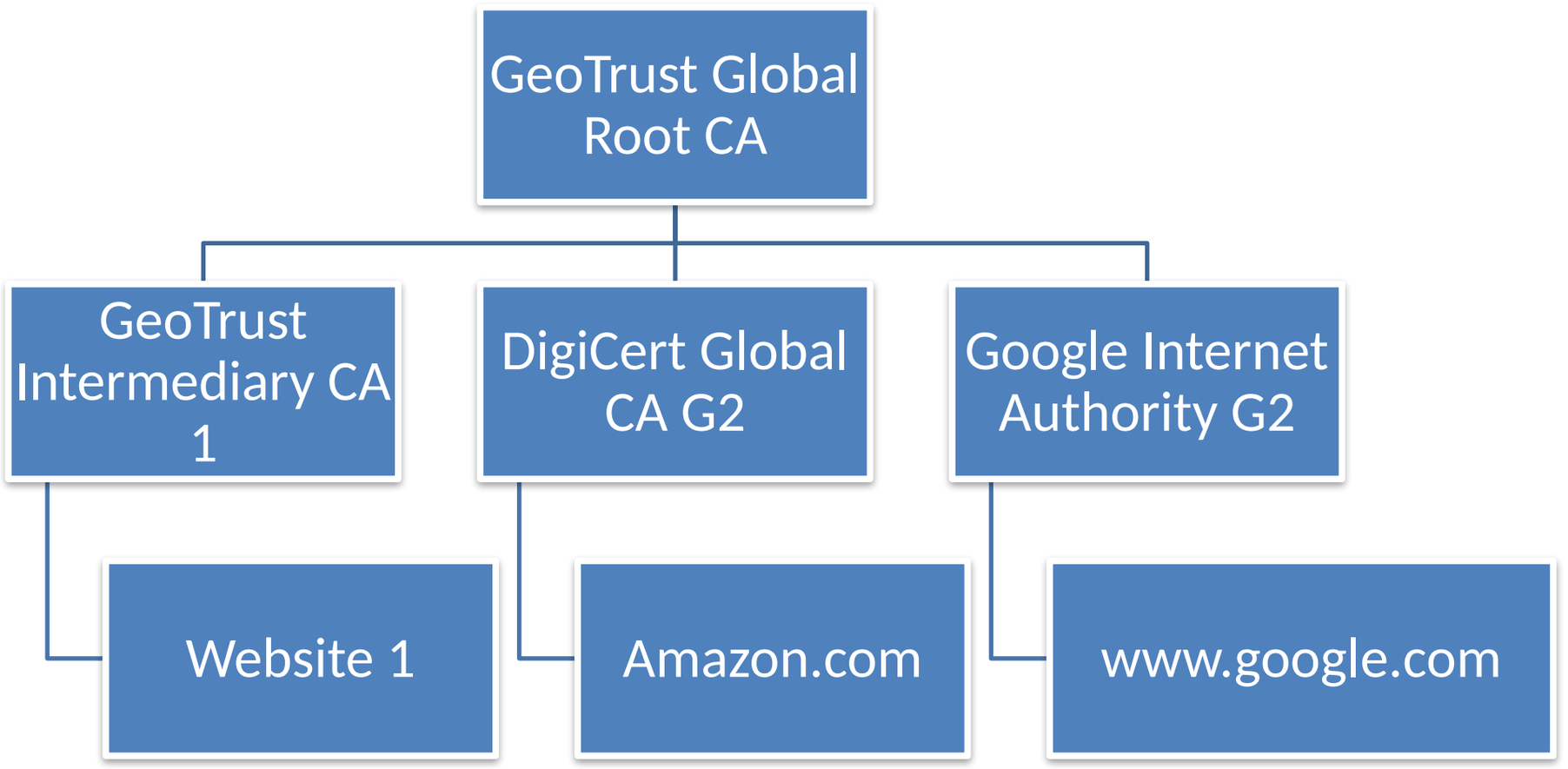
The certificate is signed by the parent CA's Private Key

Example X.509 Certificate

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesc
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```



PKI Certificate Authority Example



Root CA

Typically loaded into the browser

Intermediary CA

Typically on the webserver and cached by browser

SSL Certificate

Sent with the website when SSL/TLS begins

PKI Certificate Authority Example

- Your browser trusts *GeoTrust Root CA* Implicitly
- *GeoTrust Root CA* trusts *Google Internet Authority G2*
- *Google Internet Authority G2* trusts *www.google.com*

  ∴ therefore you trust *www.google.com*

- Root CA certificates are pre-installed with browsers
- A web browser will not trust a certificate if it's not installed and trusted by a Root CA
- However, the browser will trust anything the Root CA trusts
- Browser will check the following:
  - Check the Common Name (CN) is www.google.com
  - Check the Certificate Revocation List (CRL) for each CA (not actually done)
  - Check the Validity dates "not before" and "not after"

Certificate Issuance

- User goes to Certificate Authority to request a certificate
  - Verisign, GeoTrust, et al. for SSL/TLS and email
  - Apple, Microsoft for code signing
- Authority performs business and technical checks on the requester (usually pretty minimal)
- CA generates and issues certificate (usually through an Intermediary CA)
- There is no standard on who becomes a CA
  - Trust of a root CA depends on browsers/OS.
  - Firefox, Chrome, Microsoft, and Apple have different trusted root CAs
  - Companies routinely run their own CAs for encrypted email
- Multiple certificates can be issued for the same Subject
  - E.g., google and Microsoft have had CA issue unauthorized certs, allowing man-in-the-middle attacks

Certificate Validation

When a browser visits a TLS site, the certificate needs to be validated

- How is a certificate validated?
  - Browser downloads server's cert
    - The cert is usually sent by server
    - Browser checks if the certificate is valid
      - Subject Alternative Name (SAN) – for website certificates
        - » For TLS certificates, checks if the website is listed in the SAN
      - Subject Common Name (CN) – for other certificates
        - » For other certificates, checks if the entity is correct
          - E.g., pmak@nyu.edu
      - Date is between "Valid from" and "Valid to"
      - Check if the certificate has been revoked (various methods)
      - If the cert is or is not for a CA
        - » Basic Constraints, Subject Type=CA
        - » Only CAs are allowed to issue certs, that is, the www.google.com cert is not supposed to be issuing certificates
      - Check that the signature is valid
      - Many others
    - Repeat until the browser gets to the trusted Root CA in the certificate store
- There were numerous exploitable bugs with certificate validation due to the number of implementation errors

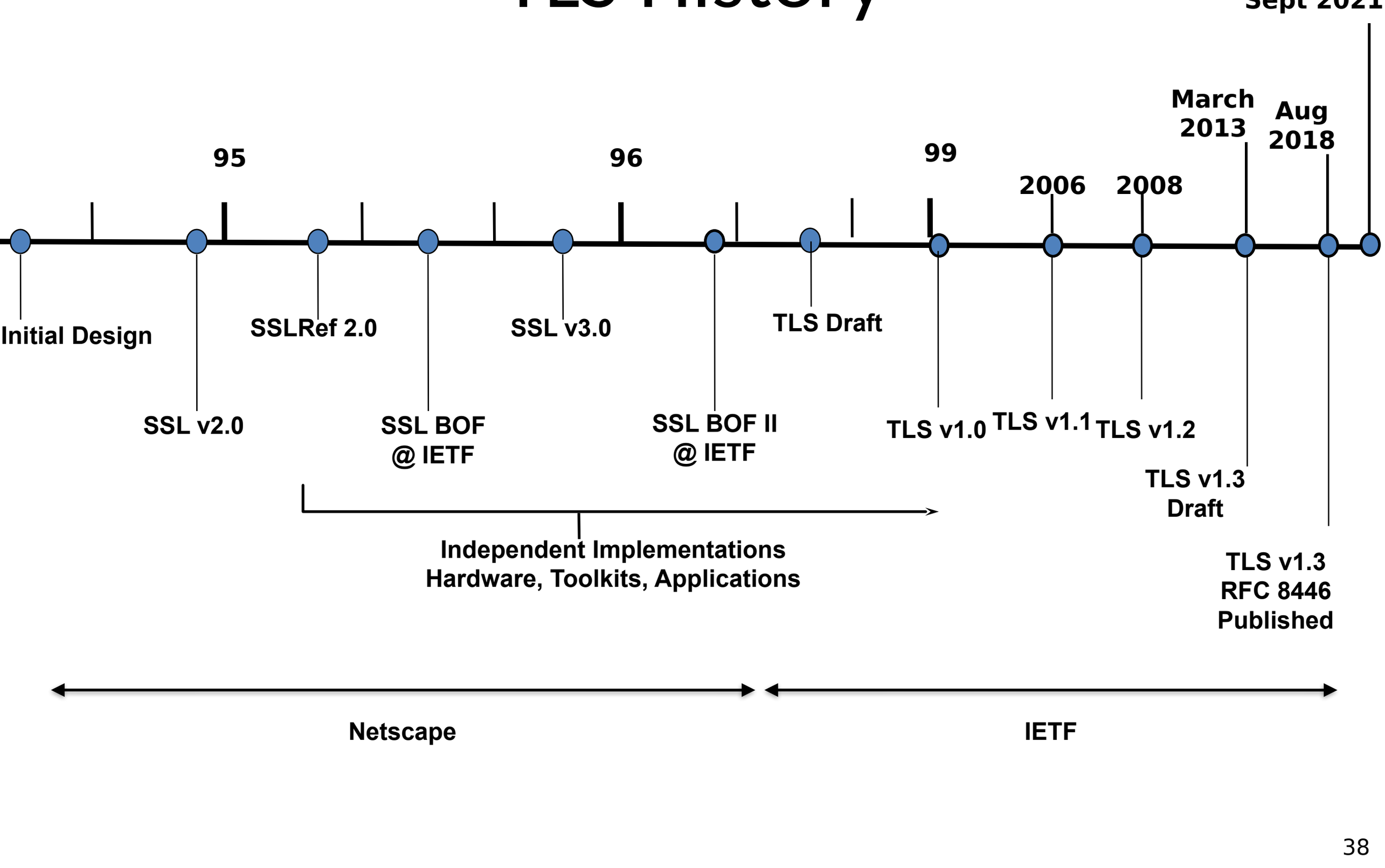
SSL/TLS

TLS: Transport Layer Security

- New name for SSL: Secure Sockets Layer
- Works on top of TCP
- Widely deployed security protocol
  - Supported by all browsers and web servers
  - Enables Internet commence
- Originally designed by Netscape in 1993
- Number of variations:
  - TLS: transport layer security, RFC 2246
- Goal is create a tunnel over TCP that can carry any TCP data
- Available to all TCP applications
  - Secure socket interface
- Provides
  - Confidentiality, Integrity, Authentication, Non-Repudiation

NYU TANDON SCHOOL OF ENGINEERING

TLS History



TLS Cipher Strength – Key Exchange and Authentication Protocol

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
RSA	Yes	Yes	Yes	Yes	Yes	No
DH-RSA	No	Yes	Yes	Yes	Yes	No
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes
ECDH-RSA	No	No	Yes	Yes	Yes	No
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes
DH-DSS	No	Yes	Yes	Yes	Yes	No
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[79]
DHE-ECDSA (forward secrecy)	No	No	No	No	No	Yes
ECDH-ECDSA	No	No	Yes	Yes	Yes	No
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes
DHE-EdDSA (forward secrecy)	No	No	No	No	No	Yes
ECDH-EdDSA	No	No	Yes	Yes	Yes	No
ECDHE-EdDSA (forward secrecy) [80]	No	No	Yes	Yes	Yes	Yes
PSK	No	No	Yes	Yes	Yes	Yes
RSA-PSK	No	No	Yes	Yes	Yes	No
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes
SRP	No	No	Yes	Yes	Yes	No
SRP-DSS	No	No	Yes	Yes	Yes	No
SRP-RSA	No	No	Yes	Yes	Yes	No
Kerberos	No	No	Yes	Yes	Yes	?
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes	No
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes	No
GOST R 34.10-2012[81]	No	No	No	No	Yes	Yes

Wikipedia Mar 2024. There are a lot of caveats to each protocol/cipher

NYU TANDON SCHOOL OF ENGINEERING

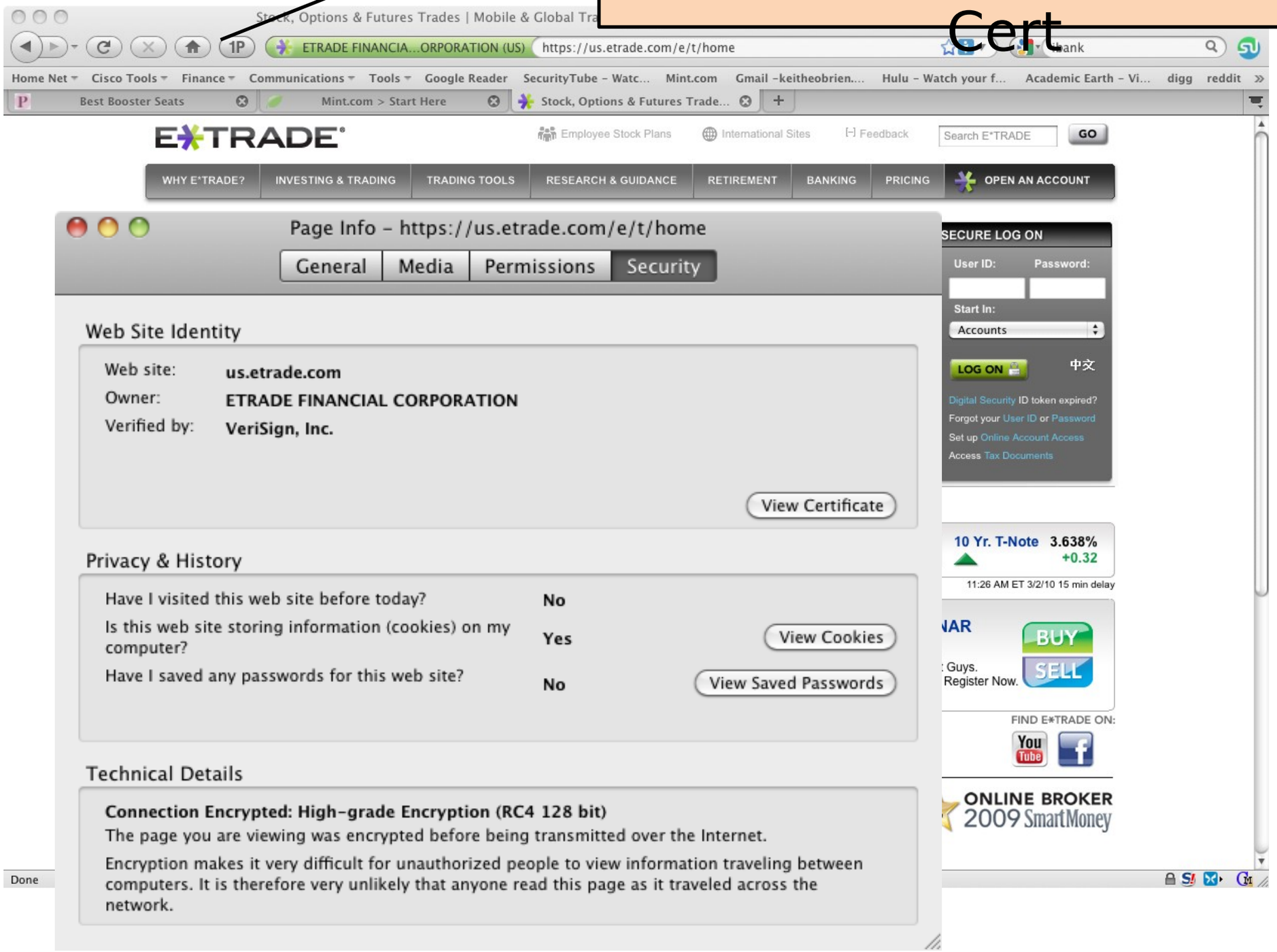
TLS Cipher Strength – Bulk Encryption Algorithm

From Wikipedia Mar 2024.

Cipher			Protocol version					
Type	Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0[n 1][n 2][n 3][n 4]	TLS 1.0[n 1][n 3]	TLS 1.1[n 1]	TLS 1.2[n 1]	TLS 1.3
Block cipher with mode of operation	AES GCM[82][n 5]	256, 128	—	—	—	—	Secure	Secure
	AES CCM[83][n 5]		—	—	—	—	Secure	Secure
	AES CBC[n 6]		—	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	—
	Camellia GCM[84][n 5]	256, 128	—	—	—	—	Secure	—
	Camellia CBC[85][n 6]		—	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	—
	ARIA GCM[86][n 5]	256, 128	—	—	—	—	Secure	—
	ARIA CBC[86][n 6]		—	—	Depends on mitigations	Depends on mitigations	Depends on mitigations	—
	SEED CBC[87][n 6]	128	—	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	—
	3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecure	Insecure	—
	GOST R 34.12-2015 Magma CTR[81][n 7]	256	—	—	Insecure	Insecure	Insecure	—
	GOST R 34.12-2015 Kuznyechik CTR[81]	256	—	—	—	—	Secure	—
	GOST R 34.12-2015 Magma MGM[81][n 5][n 7]	256	—	—	—	—	—	Insecure
	GOST R 34.12-2015 Kuznyechik MGM[81][n 5]	256	—	—	—	—	—	Secure
	IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecure	—	—
	DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecure	—	—
		40[n 10]	Insecure	Insecure	Insecure	—	—	—
	RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	—	—	—
Stream cipher	ChaCha20-Poly1305[92][n 5]	256	—	—	—	—	Secure	Secure
	RC4[n 11]	128	Insecure	Insecure	Insecure	Insecure	Insecure	—
		40[n 10]	Insecure	Insecure	Insecure	—	—	—
None	Null[n 12]	–	Insecure	Insecure	Insecure	Insecure	Insecure	—

NYU TANDON SCHOOL OF ENGINEERING

SSL Extended Validation Cert



EV SSL Certs are technically just as secure as regular certs, but for EV, the CA has performed additional

NYU TANDON SCHOOL OF ENGINEERING

Types of TLS Cert

1. DV – Domain Validation – only the domain name is validated
2. OV – Organization Validaton – the company is validated
   - Check the duns records for the company name
3. EV – Extended Validation
   - “Extended” checks on the company

- OV & EV certificates provide limited value to the user because users do not understand the difference from a DV
- DV certs can be obtained for free – from “Lets Encrypt”

SSL Overview

1. Establish a TCP connection
2. Client specifies to server specifications of SSL that it can support
3. Server picks the specifications to use
4. Server sends to client its certificate for the client verifies server certificate
   - Client verifies that server is indeed amazon.com
5. Compute shared symmetric key
6. Begin secure communications

Secure Sockets Layer Protocol

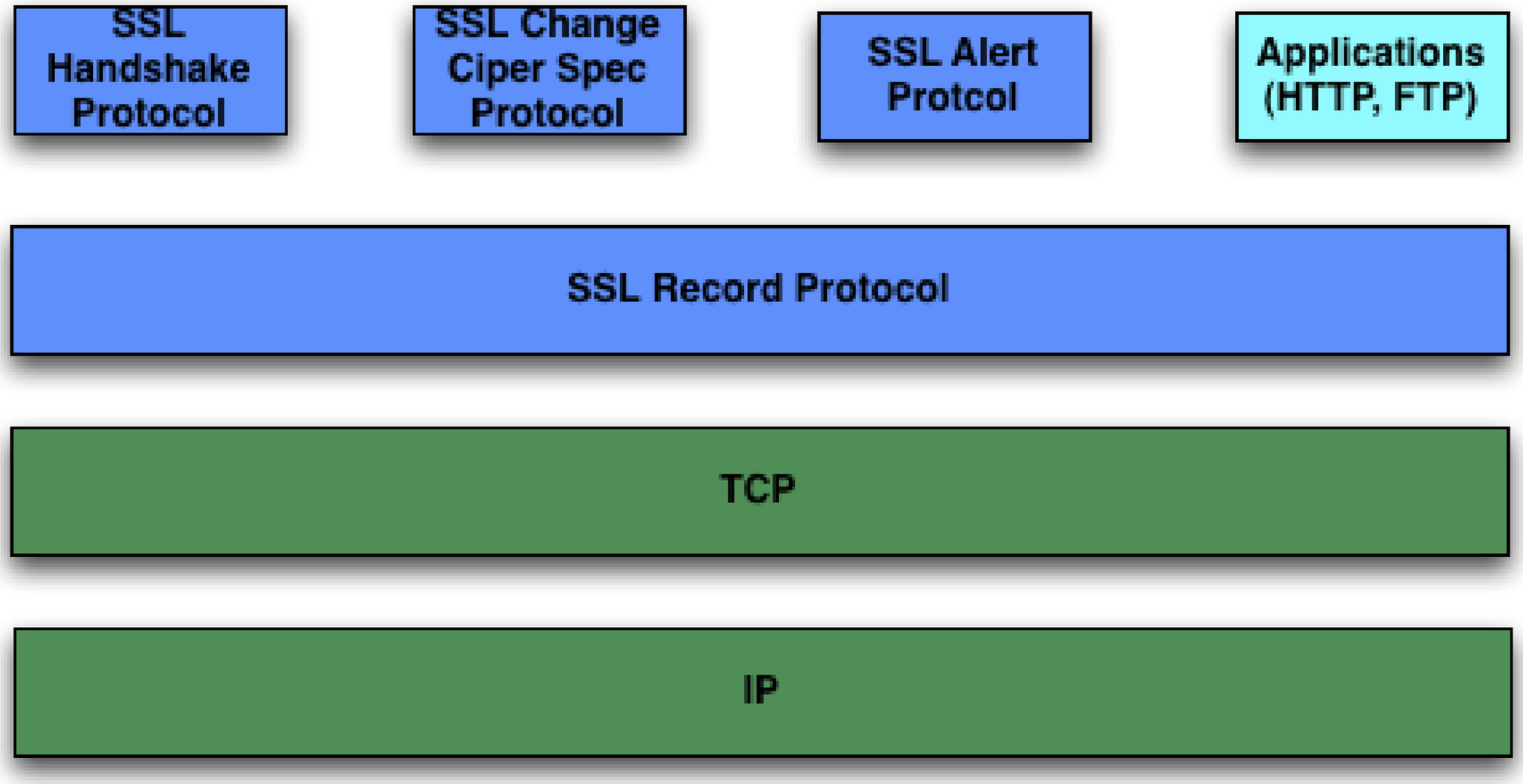
- SSL is designed to operate in a number of different modes, depending on the requirement of the network connection
  - No authentication, no encryption
  - Authentication without encryption
  - Encrypted communication only
  - Encryption and authentication of the server **(most common)**
  - Encryption and authentication of client and server

TLS Benefits

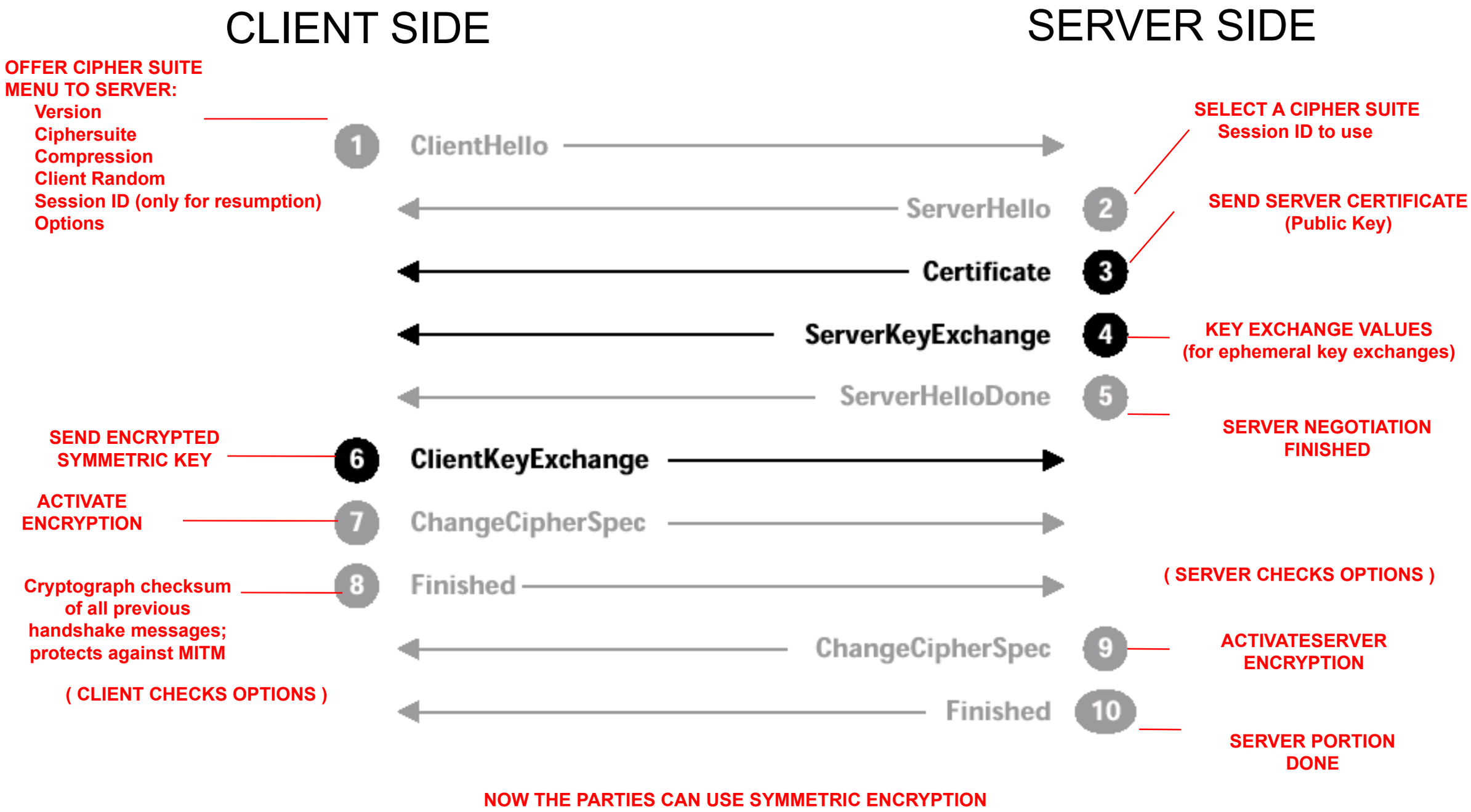
- Create a tunnel over TCP for any data
- Ensure an attacker can't change data arbitrary
- Ensure an attacker can't see some of the data
  - Attacker can still see what sites and traffic frequency
  - Attacker can see parts of the page not encrypted
- Attacker can break the TCP connection by sending TCP Resets, but it can be detected

SSL Architecture

- SSL Data is split into “records”



TLS Full Handshake (TLS 1.2 & below)



Session Resumption

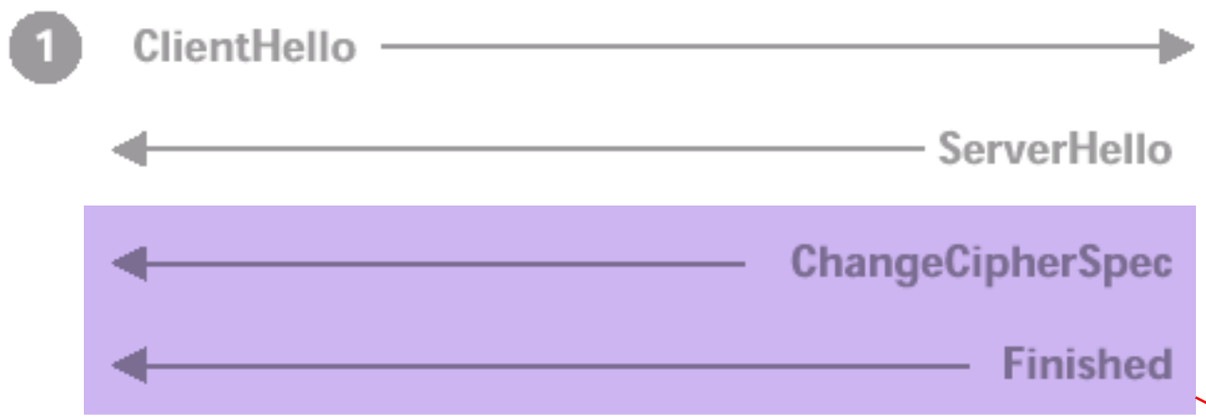
- Client/Server simply send a new Hello message
- Using the Master key already established, new per connections keys can be created using the new ClientRandom and ServerRandom
- Server and Client send Finished messages do complete the resumption
- Session resumption is used to avoid expensive initial handshake process
  - Just by transferring new random values server and client established new set of keys using the existing master key
- Also see, Version Rollback attack

TLS Abbreviated Handshake

CLIENT SIDE

SERVER SIDE

OFFER CIPHER SUITE MENU TO SERVER:
- Version
- Ciphersuite
- Compression
- Client Random
- Session ID (only for resumption)
- Options



SERVER CONFIRMS SESSION ID FOR RESUMPTION

ACTIVATESERVER ENCRYPTION

SERVER NEGOTIATION FINISHED

ACTIVATE ENCRYPTION

Cryptograph checksum of all previous handshake messages; protects against MITM

(SERVER CHECKS OPTIONS)

NOW THE PARTIES CAN USE SYMMETRIC ENCRYPTION

SSL Cipher Suite

- For public-key, symmetric encryption and certificate verification we need
  - Key exchange algorithm
  - Symmetric encryption algorithm
  - Message digest (hash) algorithm

- This collection is called a cipher suite
- SSL supports a variety of cipher suites
- Client and server must agree upon a common suite
- The client offers a choice; the server picks one.

Perfect Forward Secrecy

- If a key is compromised, only the specific session it protected will be revealed to an attacker. The security of previous or future encrypted sessions is not affected.
- Keys are securely deleted after use. Without these keys, there is no way captured cipher text can be decrypted.
- This is called perfect forward secrecy.
- Bit of a misnomer – Actually computational forward secrecy.

Common Types of Keys (Examples)

- Key Exchange and Authentication Protocols
  - RSA
    - Client encrypts random secret key with server's RSA key in certificate
  - DH_RSA (fixed Diffie-Hellman)
    - Uses DH keys A, g, n included in the server's certificate
    - Server uses the same "A" each time (specified in the certificate)
    - Signed with server's RSA public key
  - DHE_RSA (Ephemeral Diffie-Hellman)
    - Diffie-Hellman Ephemeral signed by the server RSA key in cert
    - Ephemeral means a, b is generated new each time
      - This is the Diffie-Hellman we learned in class
    - Signed with server's RSA public key
  - ECDHE_RSA
    - Variant DHE_RSA except uses the Ecliptic Curve concept of DH
- Symmetric Protocols
  - RC4, AES, DES
- Hashing Protocols
  - MD5, SHA

Full list at https://www.openssl.org/docs/apps/ciphers.html

Example Cipher Suites

KEY-EXCHANGE & AUTHENTICATION ALGORITHM

SYMMETRIC ALGORITHM

INITIAL (NULL) CIPHER SUITE

CIPHER SUITE CODES USED IN SSL MESSAGES

HASH ALGORITHM

SSL_NULL_WITH_NULL_NULL = { 0, 0 }
SSL_RSA_WITH_NULL_MD5 = { 0, 1 }
SSL_RSA_WITH_NULL_SHA = { 0, 2 }
SSL_RSA_EXPORT_WITH_RC4_40_MD5 = { 0, 3 }
SSL_RSA_WITH_RC4_128_MD5 = { 0, 4 }
SSL_RSA_WITH_RC4_128_SHA = { 0, 5 }
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = { 0, 6 }
SSL_RSA_WITH_IDEA_CBC_SHA = { 0, 7 }
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0, 8 }
SSL_RSA_WITH_DES_CBC_SHA = { 0, 9 }
SSL_RSA_WITH_3DES_EDE_CBC_SHA = { 0, 10 }
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 = { 192 , 46 };

Example:

TLS_DHE_RSA_WITH_AES_128_CBC_SHA

- Diffie-Hellman Ephemeral key exchange
- Key exchange signed with RSA private keys for purpose of authentication
- Symmetric (bulk) encryption is AES 128-bits using CBC mode
- Hashing algorithm is SHA1

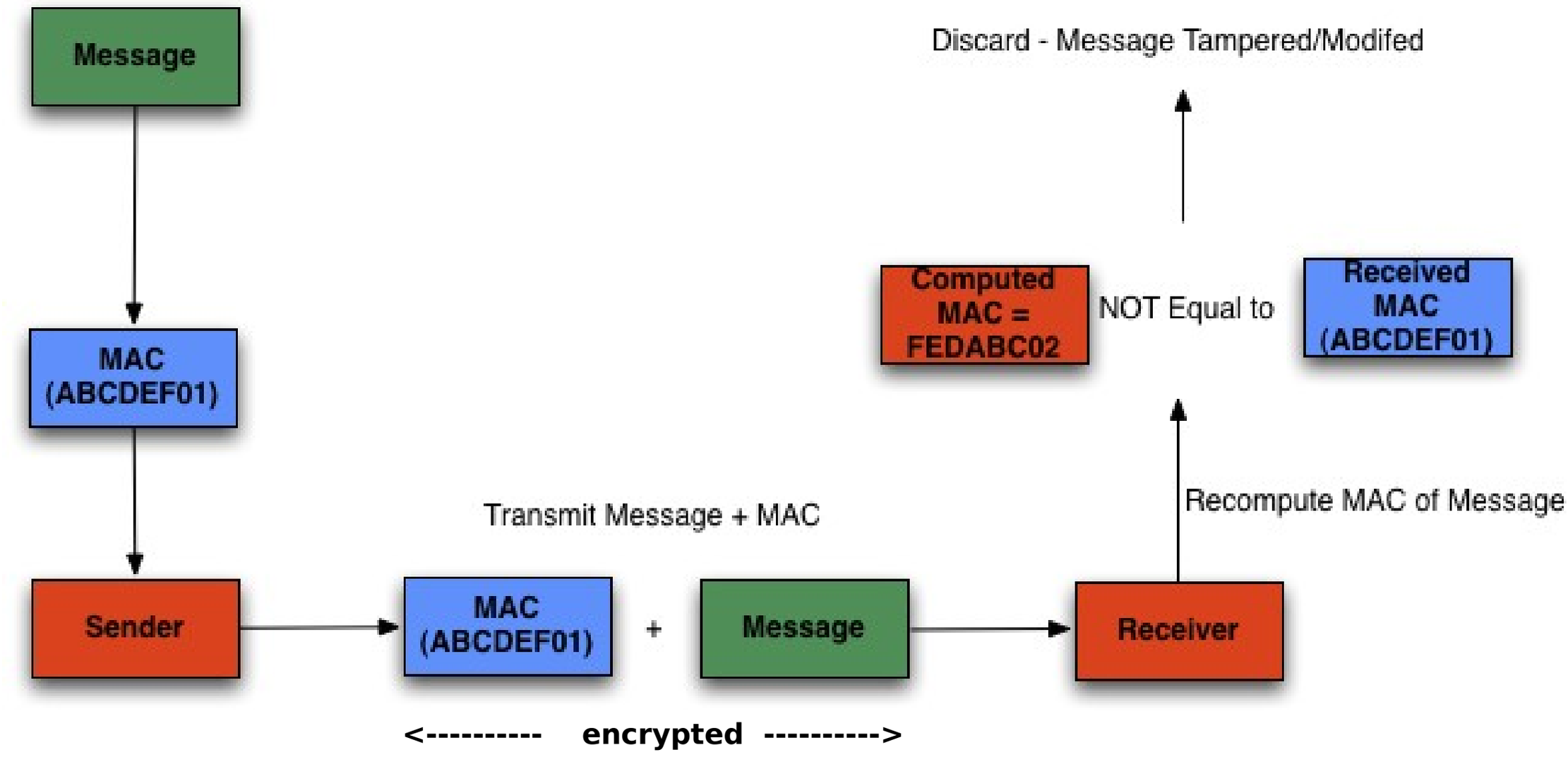
Encrypt then HMAC or HMAC then Encrypt?

- HMAC is keyed-hash
- Encrypt then HMAC (best – what TLS1.3 uses)
  - Encrypt the message first, then append the MAC of the ciphertext
  - MAC provides integrity on the cipertext
  - Host first checks MAC before continuing processing
  - Bad data is dropped before reaching the decryption engine
- HMAC then Encrypt (What TLS < 1.3 uses)
  - MAC the plaintext, then encrypt everything
  - MAC provides integrity on the plaintext
  - Hosts decrypts ciphertext first, then checks integrity
    - Decryption engine is exposed to attacker can be fed false data
- Encrypt and HMAC
  - Append the MAC of the plaintext to the ciphertext
  - No integrity on the ciphertext
  - Has known choosen-plaintext attacks
  - Host does not check MAC before decrypting

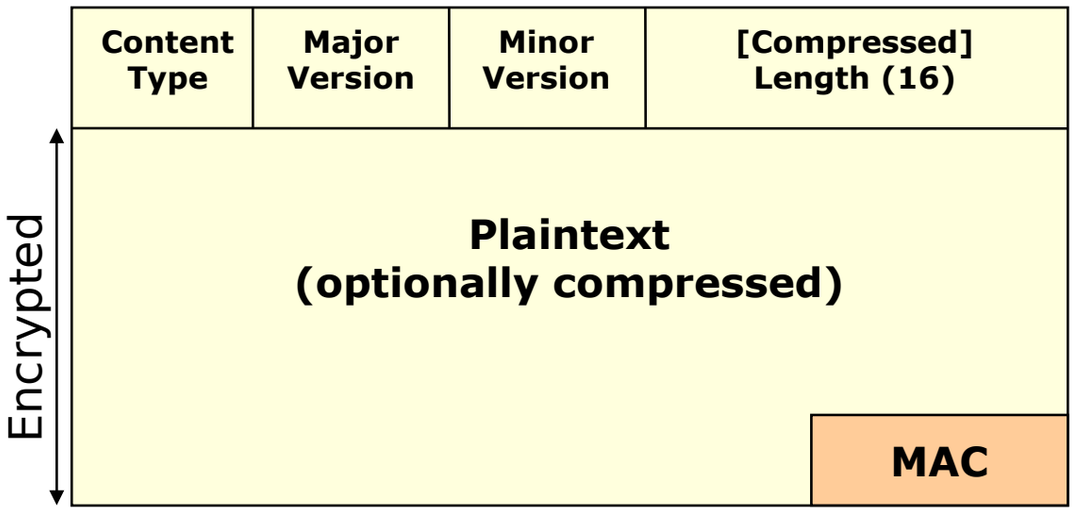
SSL - Integrity

- Compute fixed length Message Authentication Code (HMAC)
  - Includes hash of message
  - Shared secret
  - Sequence number
  - Transmit this HMAC with the message
    - TLS uses MD5, SHA-1
    - TLS 1.2 only uses SHA256+

SSL: Integrity

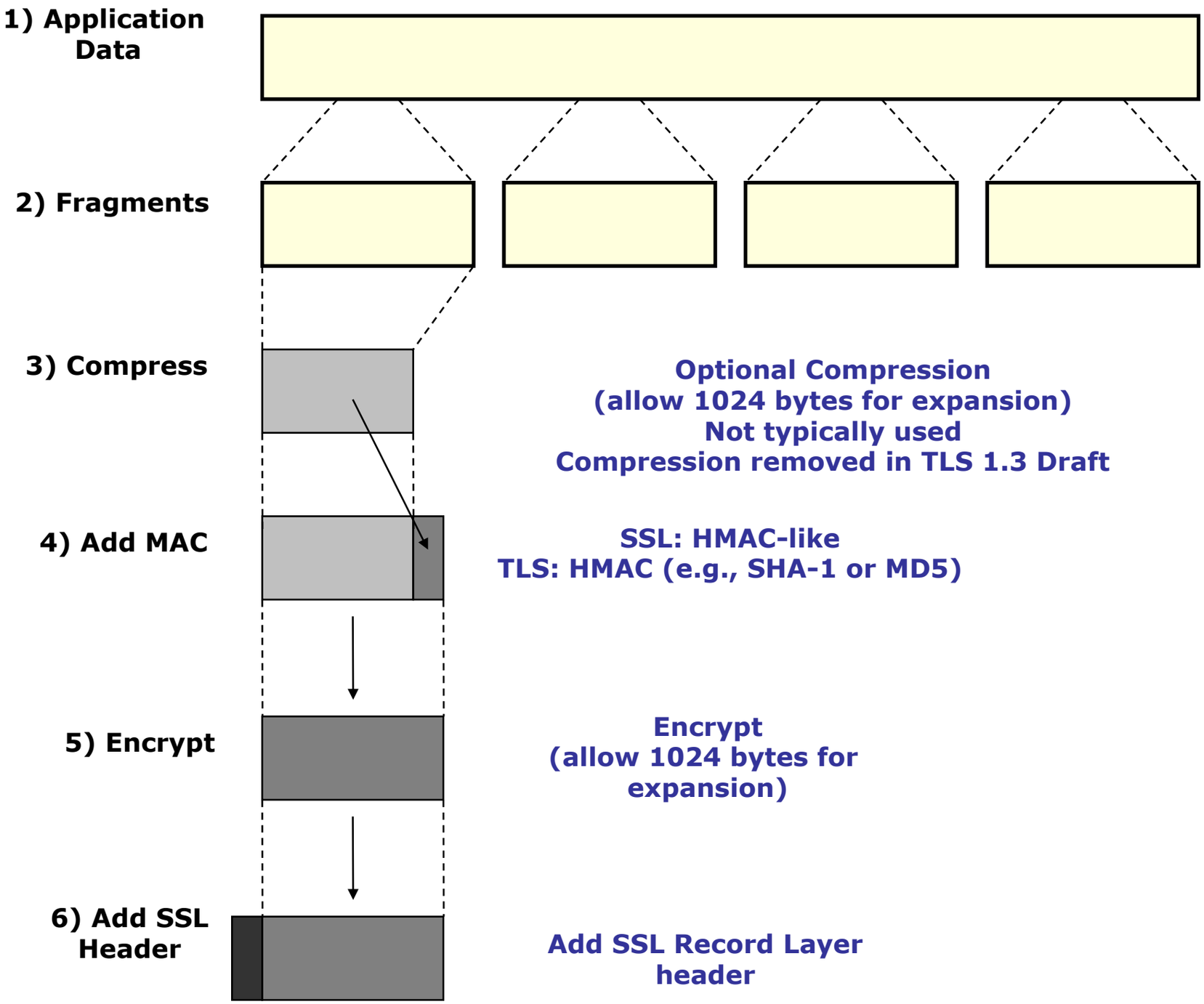


Record Layer Layer Format



- Header (5bytes)
  - Content Type (1byte):
    - change_cipher_spec (20)
    - alert (21)
    - handshake (22)
    - application_data (23)
  - Major Version (1byte):
    - Major version of SSL
    - All versions are (3)
  - Minor Version: (1byte)
    - Minor version of SSL
    - SSL v3 (0)
    - TLS 1.0 (1)
    - TLS 1.1 (2)
    - TLS 1.2 (3)
  - Data Length (2bytes)
    - Max data length: 18kb

Record Layer Protocol Operations



SSL Handshake Protocol

Type	Length	Content
1 byte	**3 bytes**	**>= 0 bytes**

- The beginning of a SSL connection
- Protocol within the record protocol
  - A record can contain several handshake messages
- Allows server and client to authenticate one another and establish security parameters
- Consists of following phases:
  - Establish security capabilities
  - Server authentication and key exchange
  - Client authentication and key exchange
  - Finish
- Type (1byte)
  - ClientHello
  - ServerHello
  - Certificate
  - ServerKeyExchange
  - CertificateRequest
  - ServerHelloDone
  - CertificateVerify
  - ClientKeyExchange
  - Finished

Handshake Protocol

The handshake protocol is used before any application data is transmitted.

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_hello_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Client Hello Parameters

- Session ID: An arbitrary byte to identify an active or resumable session
  - Could be created by using Diffie-Hellman.
  - Or by the hash of the shared secret and salt.
- Version: Identifies highest version number of SSL client can support.
- Compression method: The algorithms supported.
- Cipher spec: Specifies the bulk data encryption algorithms (such as null, DES, etc.) and a hash algorithms (such as MD5 or SHA-1) supported.
- Random Number: 32 byte random number used to seed cryptographic calculations. First four bytes should be date and time.

Server Hello Parameters

- Session ID: An arbitrary byte to identify an active or resumable session
- Version: Identifies SSL protocol version to be used.
- Compression method: The algorithm to be used.
- Cipher spec: Specifies the bulk data encryption algorithm and a hash algorithm to be used. Also key size and hash size.
- Random Number: 32 byte random number used to seed cryptographic calculations and chosen by server.

  Note that whereas client hello components are proposals, server hello components are selections.

Server KeyExchange

- Contains key information. Exact contents depend on the specific algorithm being used.
- For RSA it would be modulus and public key.
- For Diffie-Helman contains server side key message.

ServerHelloDone

- Indicates to client server has finished initial negotiation messages.
- Message itself contains no other information.

Client KeyExchange

- Establishes symmetric encryption key information.
- For example, if RSA being used then client generated session key and uses servers public key to encrypt the session key.
- Only real server can decrypt to obtain session key.

Finished Message

- Allows both sides to verify that negotiation has been successful and security has not been compromised.
- Is encrypted and authenticated by the cipher suite just established.
- Contains cryptographic hash of important information of just finished negotiation:
  - Key information
  - Contents of all previous hash messages exchanged by the systems
  - Special value indicating client or server
  - No hash of ChangeCipherState message (see Dropped ChangeCipherState attack)

SSL Change Cipher Spec Protocol

- Causes *pending state* to be copied into *current state*.

- Why separate protocol?
  - Change Cipher Spec protocol notifies the Record Layer protocol to change cipher specifications (keys etc.)
  - Forces a the next handshake message to use a new record with the new encryption spec

SSL Alert Protocol

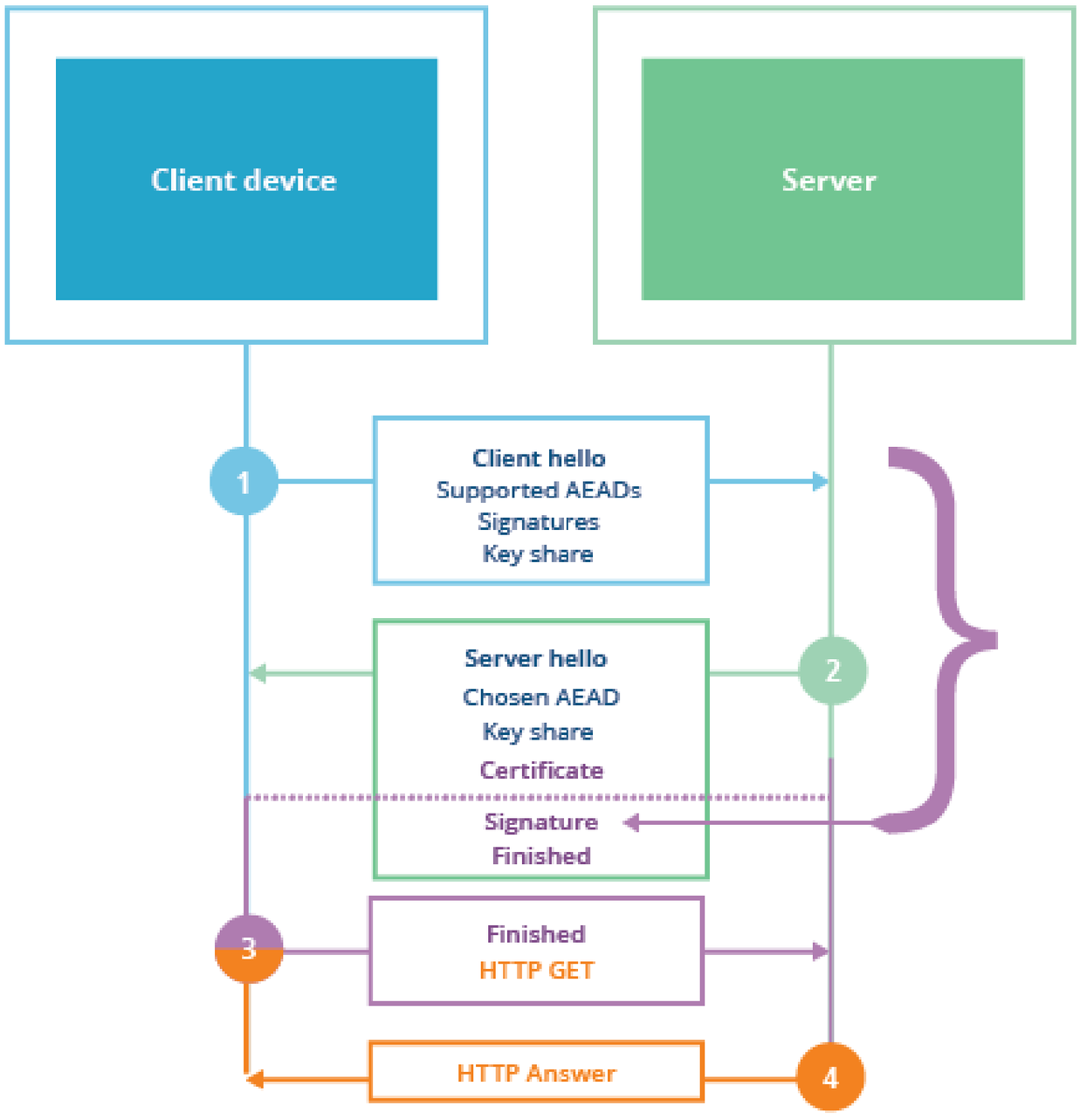
- Alert Protocol – Used for SSL related alert and error messages
  Two levels of alerts:
  - Warnings: Actions not specified in the spec.
  - Fatal:
    - Terminate the connection responsible for the alert
    - Invalidate the session that contained the terminated connection
    - No new connections allowed using that session, existing connections unchanged
    - Clients, servers should forget all session information

- Closure Alerts:
  - Client, server should notify each other of connection terminations
  - Uses close_notify alert
  - Otherwise, Truncation Attacks are possible
    - "*Defuse the bomb by removing the red wire, after removing the green wire*" what if "after removing the green wire" is never received!!!
    - Security is compromised by terminating the connection prematurely

SSL Encryption

- Pre-master Secret
  - Created by client; used to “seed” calculation of encryption parameters
  - Very simple: 2 bytes of SSL version + 46 random bytes
  - Sent encrypted to server using server’s public key
- Master Secret
  - Generated by both parties from premaster secret and random values generated by both client and server
- Key Material
  - A bit stream generated from the master secret and shared random values
  - Length of the stream may vary depending on negotiated key sizes
- Encryption Keys
  - Extracted from the key material

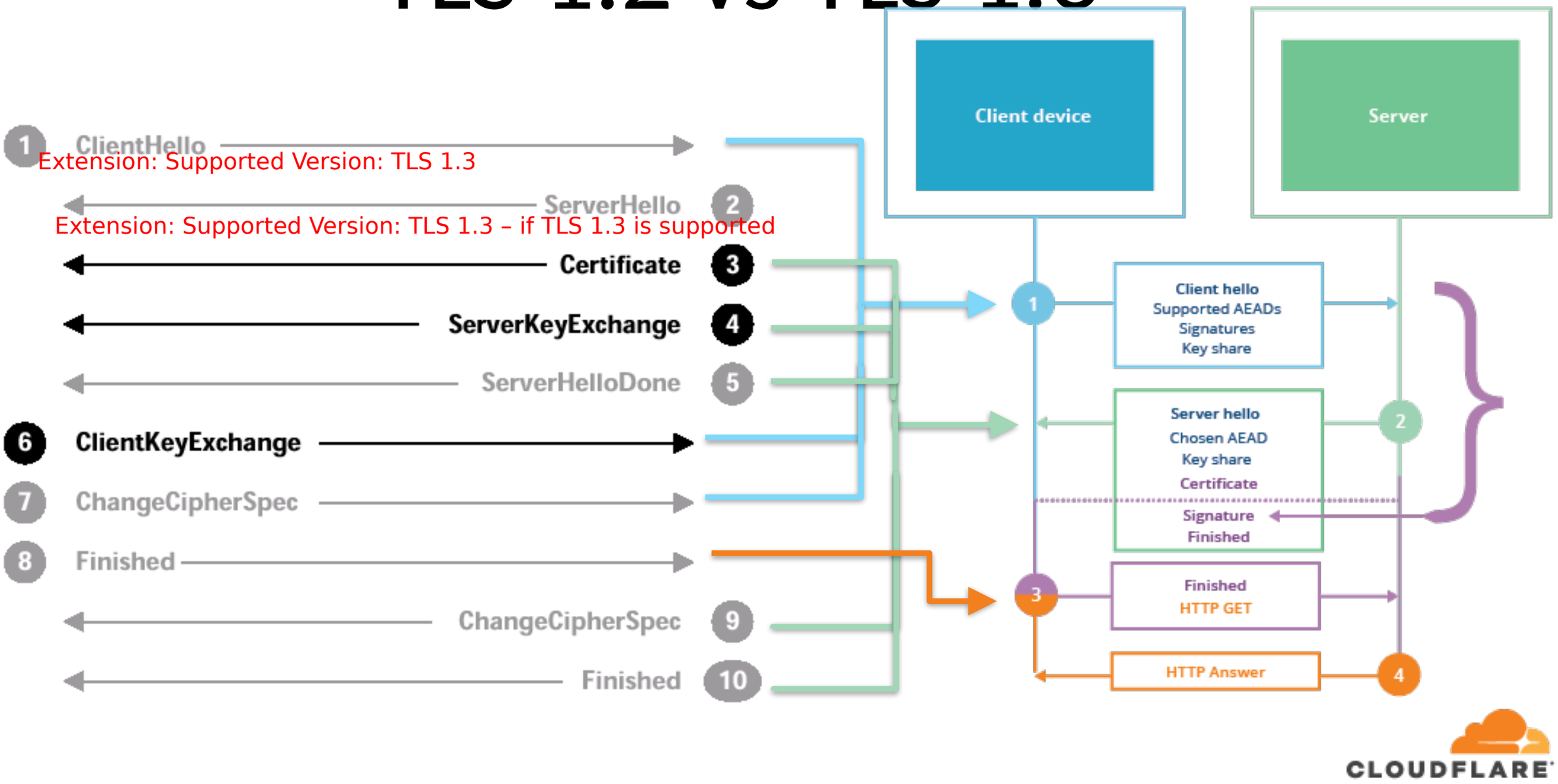
Updates in TLS 1.3



CLOUDFLARE

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

TLS 1.2 vs TLS 1.3



Security Analysis of SSL

- TLS is an extensible framework
  - TLS been audited by security research and in practice for many years
  - Flawed algorithms and procedures can be modified with RFCs and extensions
- Authentication Issues:
  - Depends on the validity of certificates and certificate revocation
  - Unfortunately, there isn't any dependable infrastructure to enforce prudent certification methods
    - E.g.: Valid certificates issued to false Microsoft, Inc.
    - IE bug was it failed to validate the entire certificate chain, instead it stopped validation after one parent certificate.
      - Certificate field “basic constraints” which specifies if the certificate belongs to a CA
      - Non CAs signing child certificates
    - Certificate Revocation Lists (CRLs) not commonly checked by browsers
- Key Exchange Issues:
  - Avoid the use of anonymous Diffie-Hellman **`dh_anon`** (MITM attack)
  - Choice of “strong” public parameters in Ephemeral Diffie-Hellman
    - Choosing very small prime numbers (like 3 or worst 2!)
- Proper padding, salt, and IVs
- Using broken or weak algorithms
  - Using “export” versions with 40-bit key size
  - Symmetric < 128 bit
  - Asymmetric and key exchange < 1024 bits
- Privacy Issues:
  - Remote Timing Attacks
  - Traffic Analysis

Vulnerabilities in SSL/TLS

- Physical Attacks
  - Private keys can be physically stolen from bribery, exploitation, or simply unsecured backups
  - Does not protect against compromised hosts or application vulnerabilities
- Improper validation of certificate fields
  - No validation “not before” and “not after” is based on client’s time.
  - Allows an attacker to use an valid expired certificate that’s weaker
- User Errors
  - User ignore certificate warnings
- SSLStrip
  - MITM attack that replaces encrypted versions with non-encrypted versions

Adversary-in-the-Middle Attacks

- Installing malicious root CA in user's certificate store
  - Commonly done in corporate environments to track user's Internet usage
  - Used by several malware
- A Simple DoS
  - Send an invalid SSL packet to the server or client, by predicting subsequent TCP values
  - SSL will ignore the packet but TCP will accept it
  - Then, when the real SSL packet comes in TCP will ignore it thinking it is a duplicate and SSL layer will never receive the packet subsequently resulting in a connection termination.