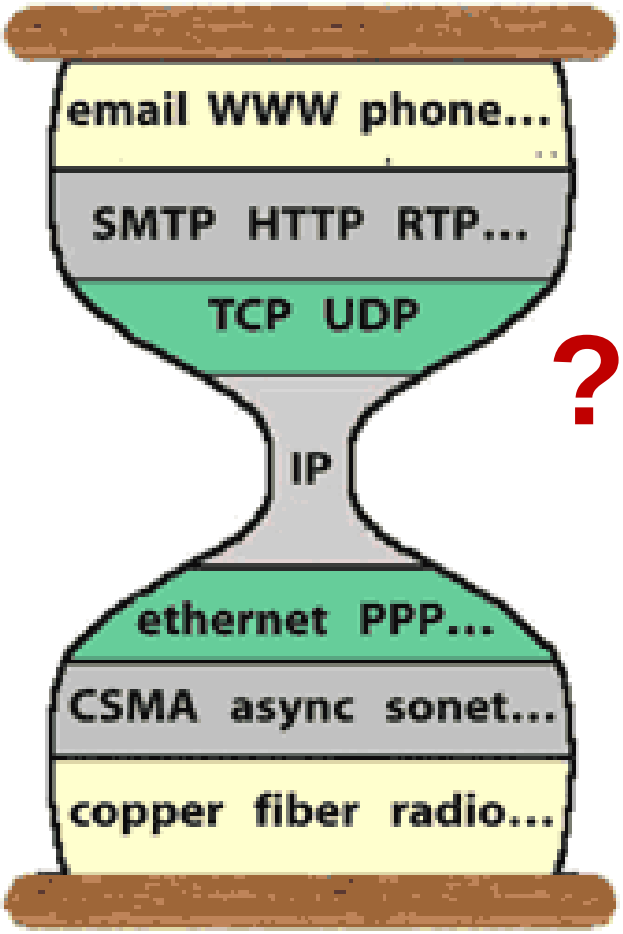
Lecture 11: Security

FYI



8.1 Network Security: overview

8.2 basics of Cryptography

- Symmetric Key Cryptography
- Public Key Encryption

8.3 Message Integrity and Digital Signatures

- Cryptographic Hash Functions
- Message Authentication Code
- 8Digital Signatures

What is network security

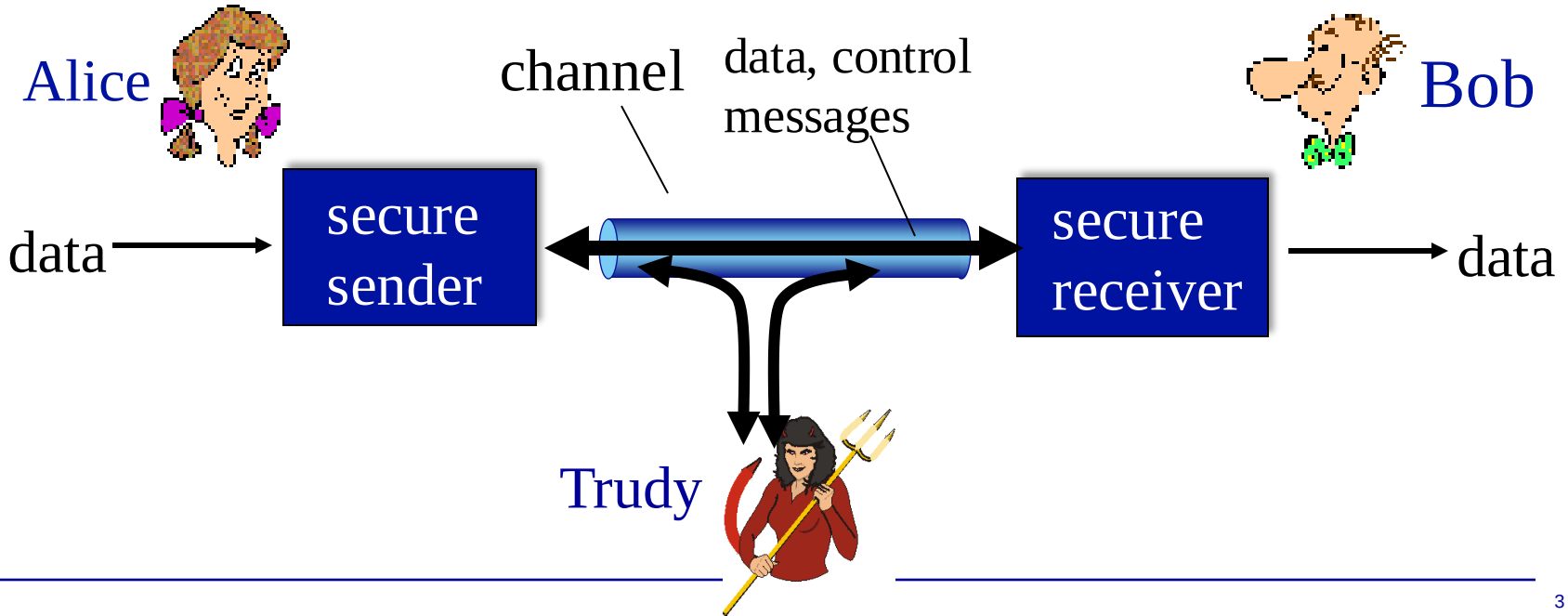
- *Confidentiality*: only intended receiver(s) can see message contents
  - Sender encrypts message
  - Each receiver decrypts message
- *Authentication*: receiver can confirm the identity of the producer of received message
- *Data integrity*: any changes to the message (in transit, or afterwards) can be detected
- *Availability*: services/data available to users
  - biggest threat to availability today: DDoS (distributed denial of service)

Utilize cryptographic technologies

The basic network security model

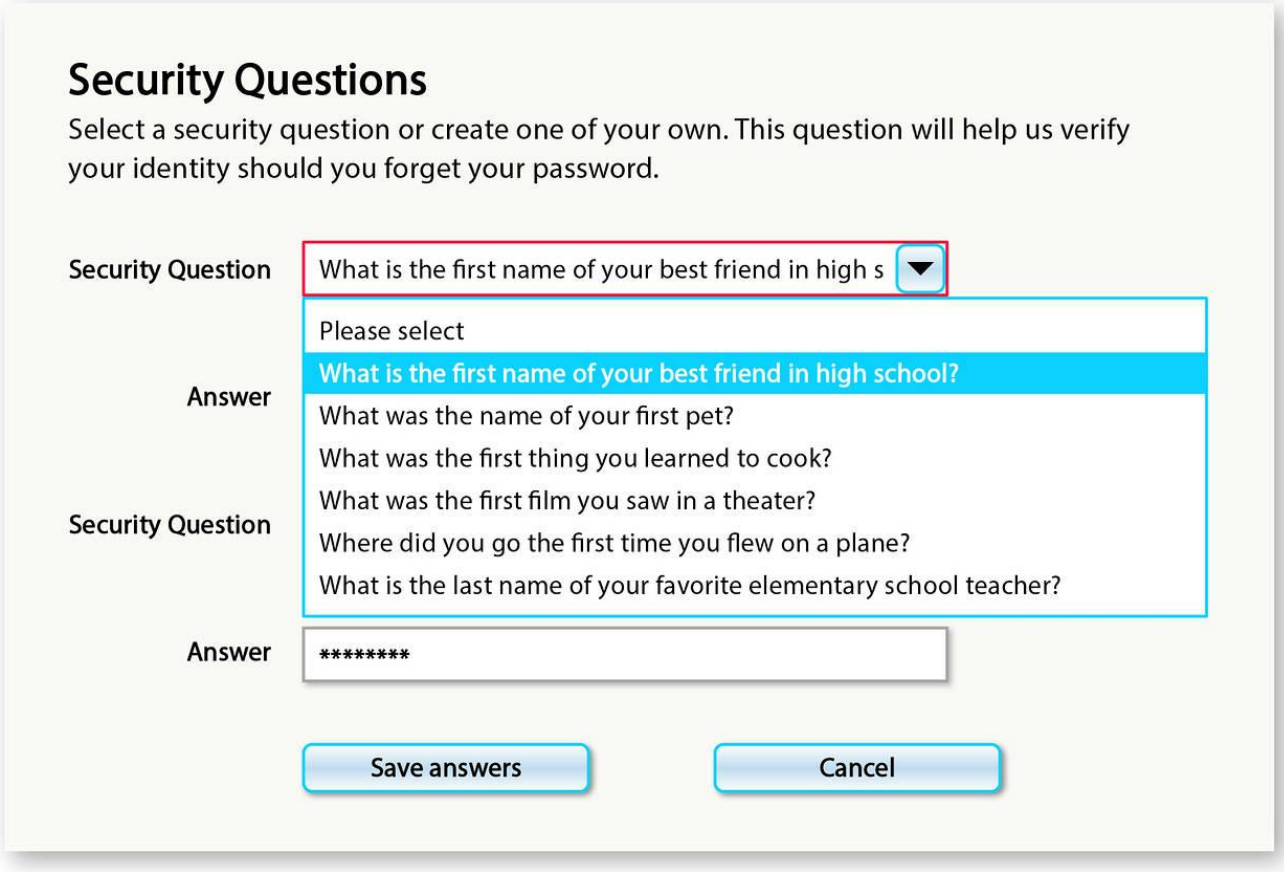
Friends and enemies

- *Alice* and *Bob* (2 communicating entities) want to communicate *securely*
  - *Web browser and server*
  - *on-line banking client and server*
- Trudy (intruder) may intercept, delete, add, or modify messages



(in general) How to protect secret…?

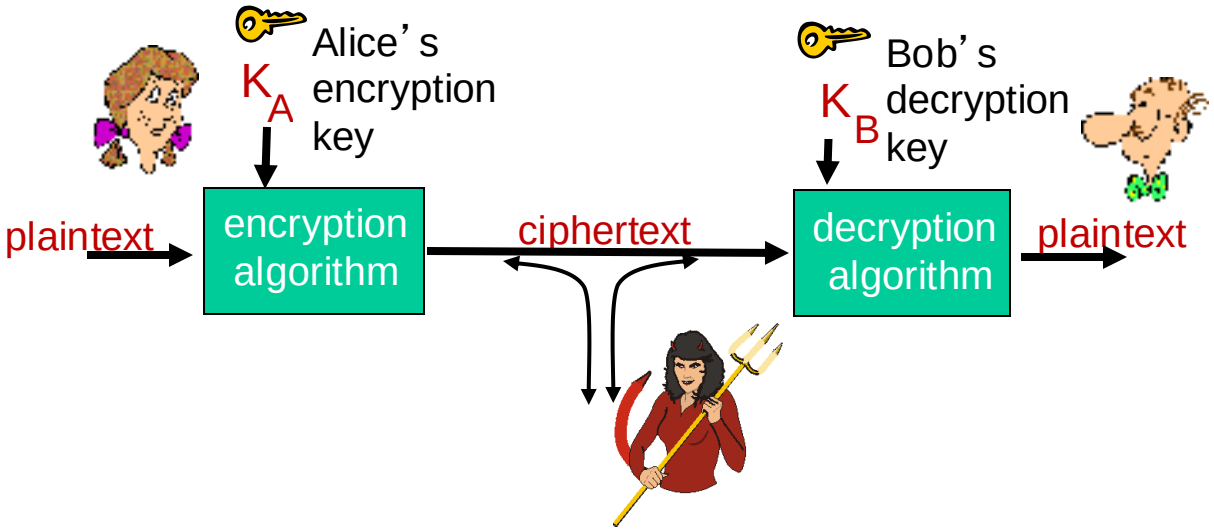
- (relatively) easy to make, hard to ~~remember~~ guess
  - Password
  - Codebook
  - Security questions…



(in general) How to protect secret…? mathematically

- One example: factoring is slow, multiplication is easy
  - 91 = 7 * 13
  - A magic number pair: 5, 29
    - **67** (C in ASCII), multiplies itself, takes remainder of 91, repeat 5 times
    - 67 * 67 = 4489, remainder 30
    - 30 * 67 = 2010, remainder 8
    - 8 * 67 = 536, remainder 81
    - 81 * 67 = 5427, remainder **58**
    - 58, multiplies itself, takes remainder of 91, repeat 29 times
    - …
    - 9 * 58 = 522, remainder **67**

The language of cryptography



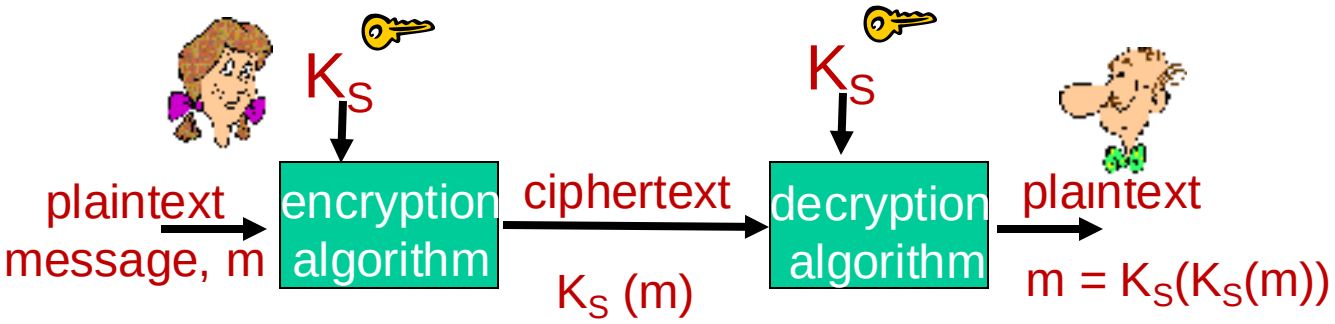
m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

8-6

Symmetric key cryptography



Bob and Alice share the same (symmetric) key: K_s

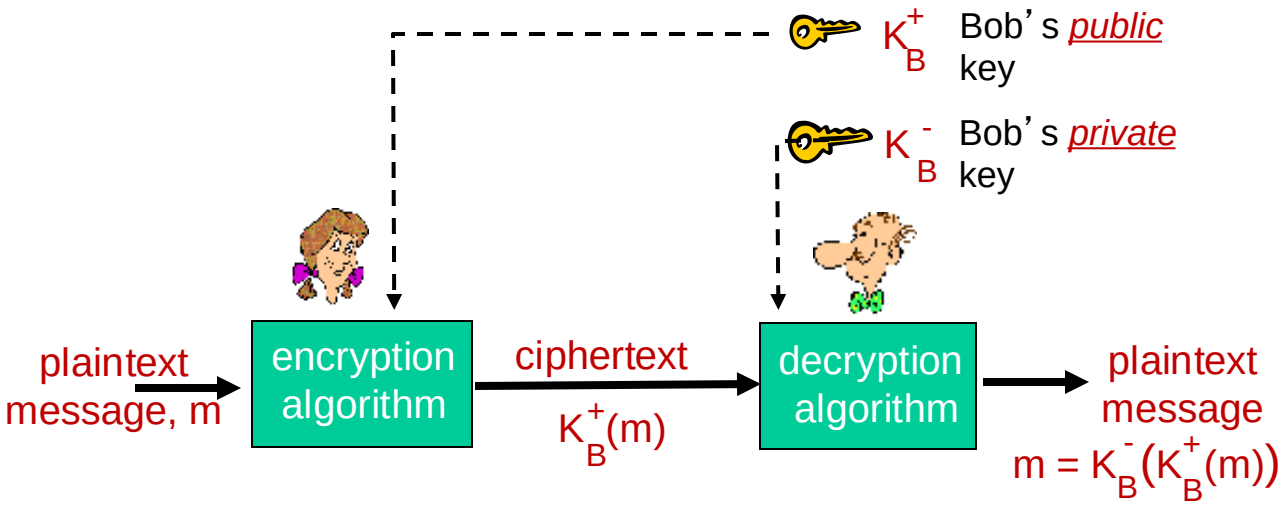
Q: how do Bob and Alice agree on the key value in a *secure* way?

- Especially if they do not meet in person ?

8-7

Public Key Cryptography

- Radically different approach [Diffie-Hellman76, RSA78]
  - *RSA*: Rivest, Shamir, Adelson algorithm
- Sender and receiver do not share secret key
- Each of them produce a pair of keys
  - public key: known to all
  - private key: known only to oneself



Public key encryption algorithms

Requirements:

1. need $K_B(\bullet)^+$ and $K_B(\bullet)^-$ such that

$$K_B^-(K_B^+(\text{m})) = \text{m}$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: an important property

result is the same

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

- Very useful property
  - Confidentiality: using K_B^+ to encrypt a private message to Bob
  - Nonrepudiation: if Bob uses K_B^- to encrypt a message, everyone can use K_B^+ to prove that Bob produced it
- One problem: using public-key to encrypt long messages is computationally expensive

RSA: Creating public/private key pair

1. choose two large prime numbers *p, q.* (e.g., 1024 bits each)

2. compute *n* = *pq*, *z* = *(p-1)(q-1*)

3. choose *e (*with *e* < *n)* that has no common factors with z (*e*, *z* are "relatively prime").

4. choose *d* such that *ed-1* is exactly divisible by *z*. (in other words: *ed* mod *z* = *1*).

5. *public* key is *(n,e). private* key is *(n,d,p,q).*

K_B^+ K_B^-

Private key can derive the public part, but not the other way

(in general) How to protect secret…? mathematically

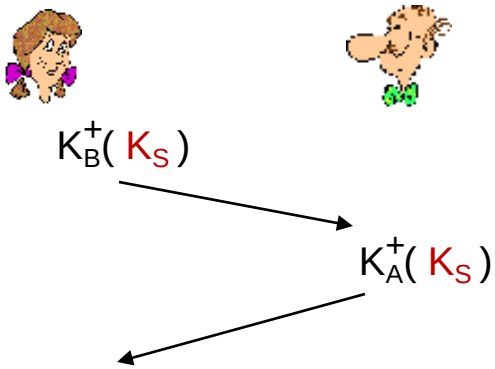
- One example: factoring is slow, multiplication is easy
  - 91 = 7 * 13
  - A magic number pair: 5, 29
    - **67** (C in ASCII), multiplies itself, takes remainder of 91, repeat 5 times
    - 67 * 67 = 4489, remainder 30
    - 30 * 67 = 2010, remainder 8
    - 8 * 67 = 536, remainder 81
    - 81 * 67 = 5427, remainder **58**
    - 58, multiplies itself, takes remainder of 91, repeat 29 times
    - …
    - 9 * 58 = 522, remainder **67**

Public key: (91, 5) $ed \bmod (p-1)(q-1) = 1$ $91 = pq$

Private key: (91, 29, 7, 13) $29e \bmod (7-1)(13-1) = 1$ $5d \bmod (p-1)(q-1) = 1$

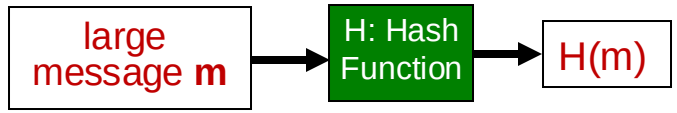
Use Public Key Crypto to Obtain Session Keys

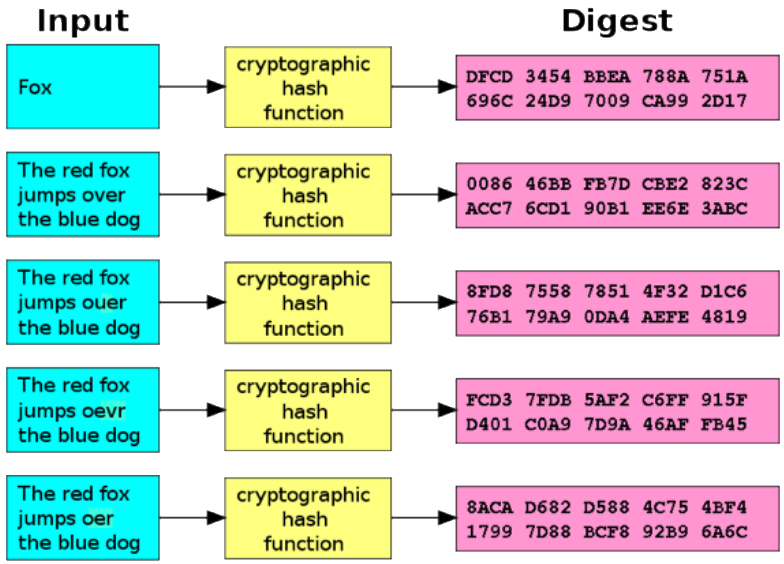
- Assuming that Alice and Bob know each other's public keys:
- Alice picks a symmetric key KS , uses Bob's public key to encrypt KS , send to Bob
- Bob uses his private key to decrypt Alice's msg, gets the symmetric key KS
- Once both Alice and Bob have KS , they can start using symmetric key cryptography to communicate
- Q: how can Bob know for sure the msg is sent by Alice?
- Next: digital signature by public key
  - Need to understand *crypto hash* first



Cryptographic hash function/message digest

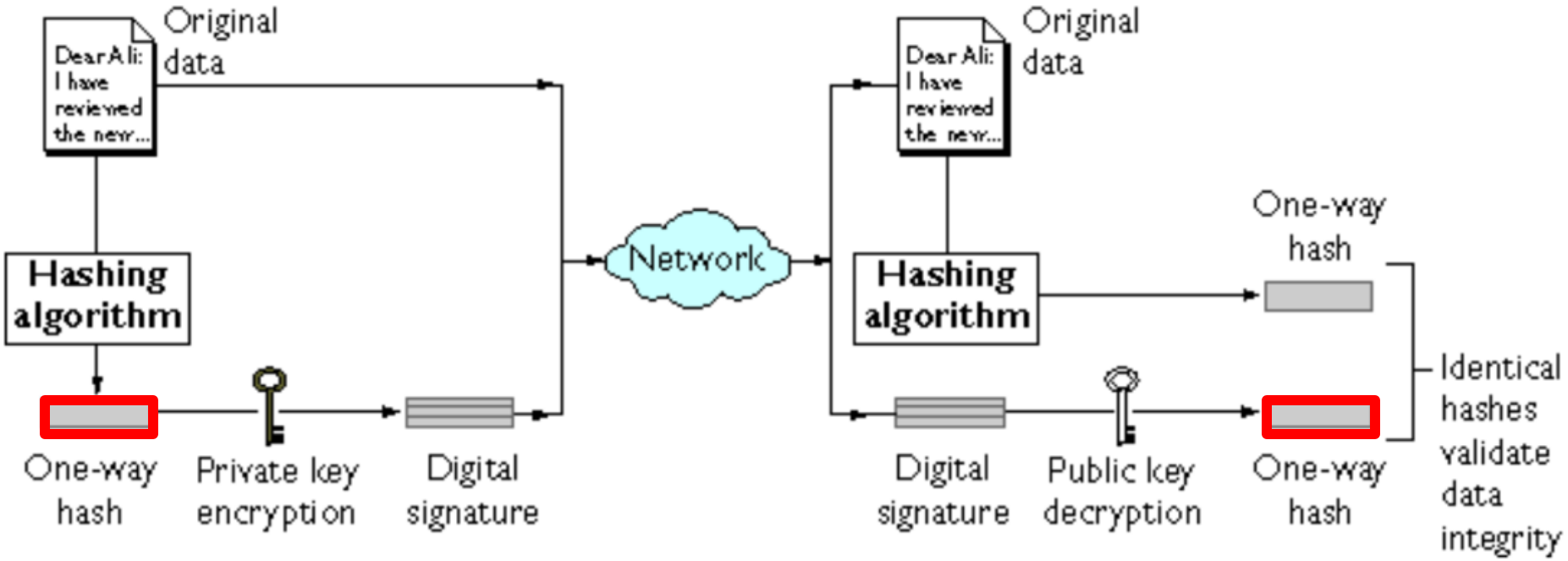
- *goal:* map a (potentially long) variable length message to a fixed-length, easy-to-compute digital "fingerprint"
- Desired properties of hash function
  - Deterministic
  - Given $H(m)$, it is infeasible to generate a message that yields $H(m)$
  - It is infeasible to find m_1, m_2 with the same hash value
  - A small change from m to m' should lead to big change in $H(m')$, uncorrelated with $H(m)$





Digital Signature by Public key crypto

- The sender has a pair of keys: public, private
  - Cryptosign(private key, data) → signature of data
  - Cryptoverify(public key, data, signature) →validation

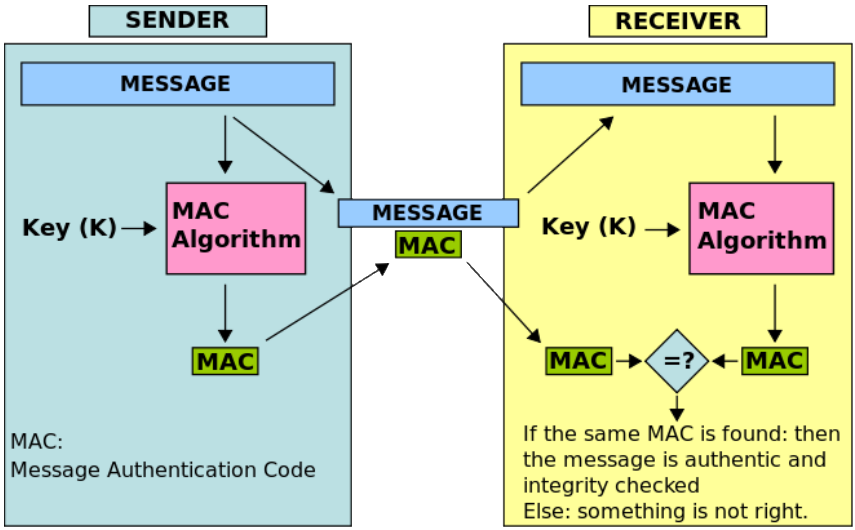


Message authentication code (MAC)

- A message authentication code consists of three algorithms:
  - A key generation algorithm selects a key from the key space uniformly at random.
  - given the key and the message: a **symmetric** signing algorithm efficiently returns a tag (or the MAC).
  - given the key and the tag, a verifying algorithm efficiently verifies the authenticity of the message
- MAC vs hash: stronger protection
- MAC vs signature: authentication without identity
  - the same secret key used for MAC generation and msg verification

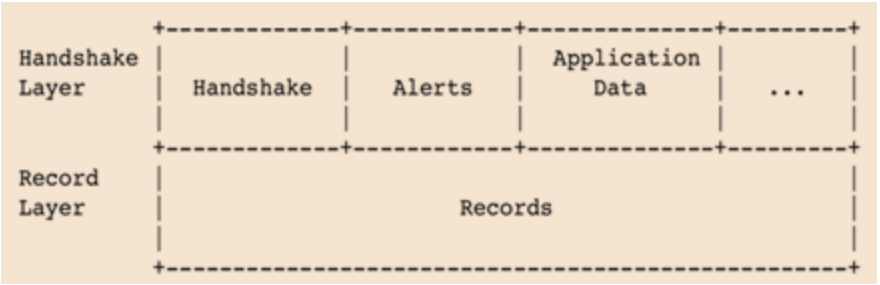
Transport Layer Security (TLS)

- Goals
  - provides data confidentiality using symmetric key cryptography
  - provides data integrity using a keyed message authentication checksum (MAC)
- Can we encrypt data in byte stream as we write data into TCP?
  - where to put the MAC?
  - If put at the end of a TCP connection: no message integrity checking until all data processed.



TLS data records

- Break byte stream to series of records
  - Each record carries a MAC
  - Receiver checks each record as it arrives
- TLS consists of two primary components
  - A handshake protocol that authenticates the communicating parties, negotiates cryptographic parameters, and establishes shared keying material.
  - A record protocol that uses the parameters from the handshake protocol to protect traffic
    - Divide app data into records, each independently protected



TLS Record Protocol

Sender:

- Read the messages for transmit
- Fragment messages into chunks of data
- Encrypt the data
- Calculate the MAC
- Transmit the resulting data to the peer

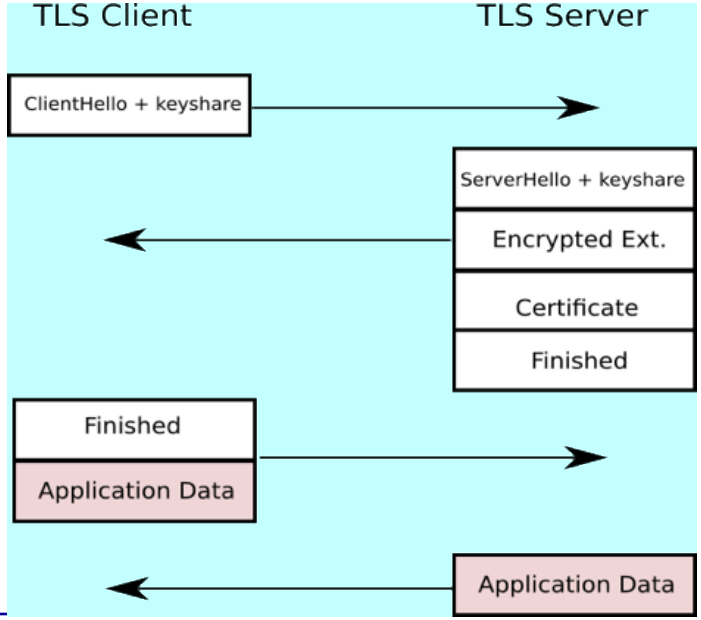
Receiver:

- Read received data from the peer
- Verify the MAC
- Decrypt the data
- Reassemble fragments back to the message
- Deliver the message to upper protocol layers

Reliable data delivery (by TCP)

TLS Handshake Protocol

- A client connecting to a TLS-enabled server presents a list of supported cipher suites (ciphers and hash functions).
- The server notifies the client its picks of cipher and hash function
- The server provides a digital certificate.
  - The certificate contains the server name
  - the trusted certificate authority (CA) that vouches for the authenticity of the certificate
  - the server's public key.
- The client confirms the validity of the server certificate before proceeding.

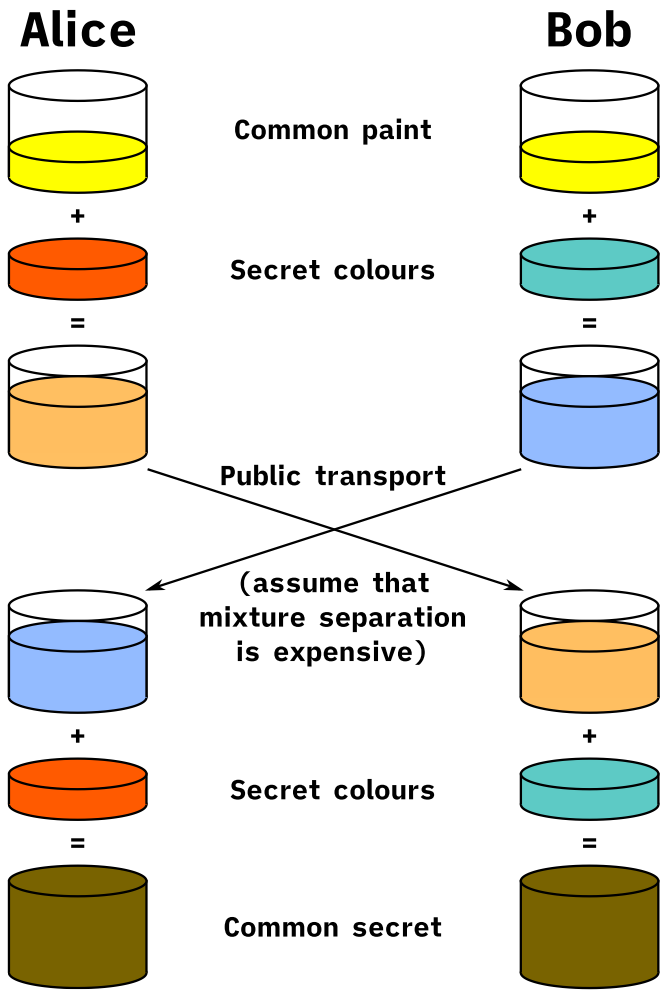


TLS Handshake Protocol

- A client connecting to a TLS-enabled server presents a list of supported cipher suites (ciphers and hash functions).
- The server notifies the client its picks of cipher and hash function
- The server provides a digital certificate.
  - The certificate contains the server name
  - the trusted certificate authority (CA) that vouches for the authenticity of the certificate
  - the server's public key.
- The client confirms the validity of the certificate before proceeding.
- To generate the session keys used for the secure connection, the client either:
  - encrypts a random number with the server's public key and sends the result to the server; both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session
  - uses Diffie–Hellman key exchange to securely generate a random and unique session key for encryption and decryption
    - additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party.

Diffie–Hellman key exchange in picture

- Alice and Bob agree on an arbitrary starting color
- Each selects a secret color that they keep to themselves
- Each mixes the secret color together with the mutually shared color
- Exchange the mixed colors.
- Each mixes together the received mixed color with own private color, obtaining an identical secret share between the two



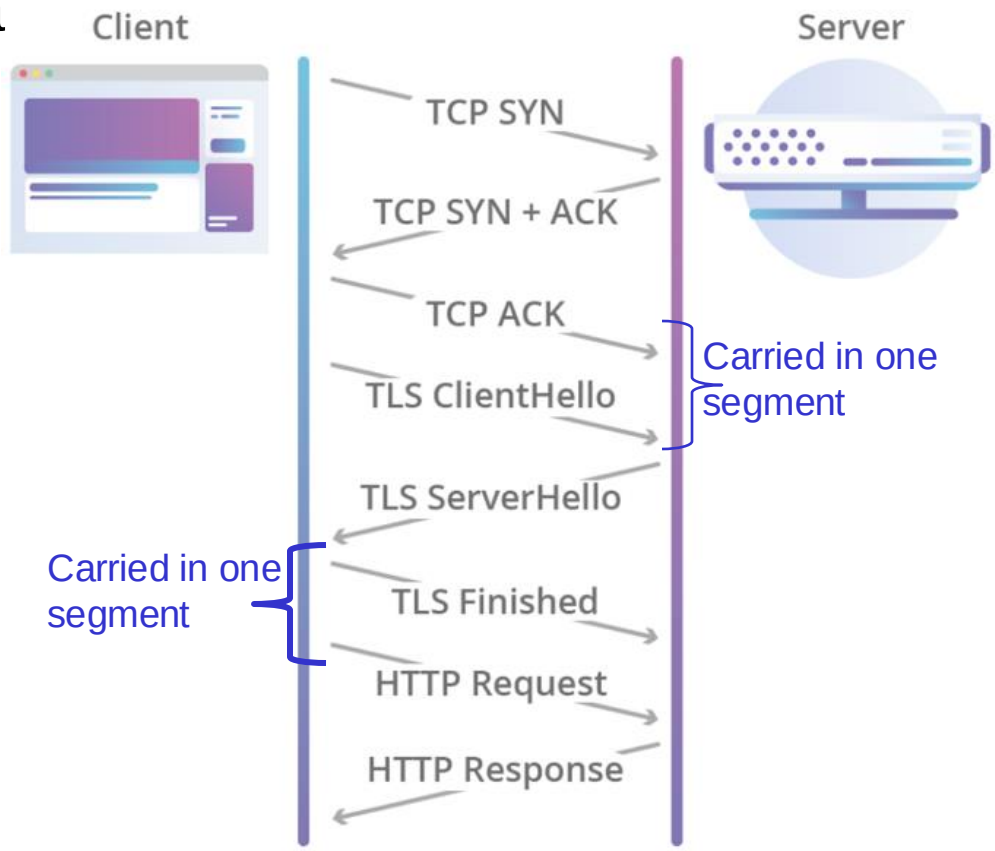
Mathematically speaking

- A and B agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
  - a number g is a **primitive root modulo *n*** if every number a coprime to n is congruent to a power of g modulo n
    - A and B are **coprime** if both of them can only divided by 1
    - a and b are said to be *congruent modulo* n, if their difference $a - b$ is an integer multiple of n (that is, if there is an integer k such that $a - b = kn$)
- A chooses a secret integer $\boldsymbol{a} = 4$, then sends B $A = g^{\boldsymbol{a}} \bmod p$
  - $A = 5^{\boldsymbol{4}} \bmod 23 = 4$
- B chooses a secret integer $\boldsymbol{b} = 3$, then sends A $B = g^{\boldsymbol{b}} \bmod p$
  - $B = 5^{\boldsymbol{3}} \bmod 23 = 10$
- A computes $\boldsymbol{s} = B^{\boldsymbol{a}} \bmod p$
  - $\boldsymbol{s} = 10^{\boldsymbol{4}} \bmod 23 = 18$
- B computes $\boldsymbol{s} = A^{\boldsymbol{b}} \bmod p$
  - $\boldsymbol{s} = 4^{\boldsymbol{3}} \bmod 23 = 18$
- A and B now share a secret (the number 18)

Key math motivation:
Given $g^x = y \bmod p$
From x to y is easy, y to x is very hard

TLS over TCP

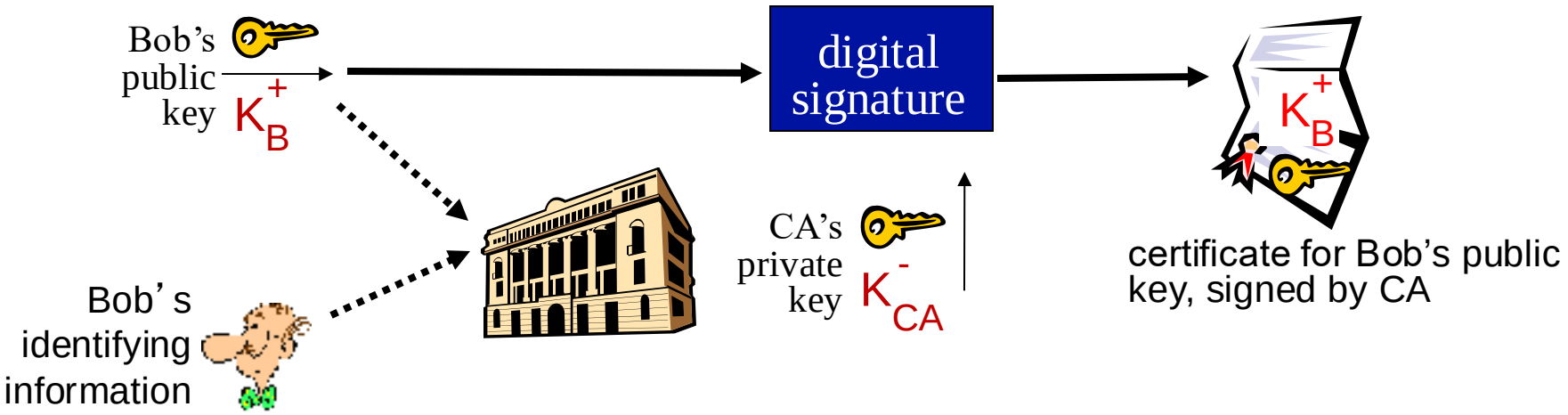
- 3-way handshake to set up a TCP connection
- Then TLS can start its own handshake
- Then send application data using TLS record protocol



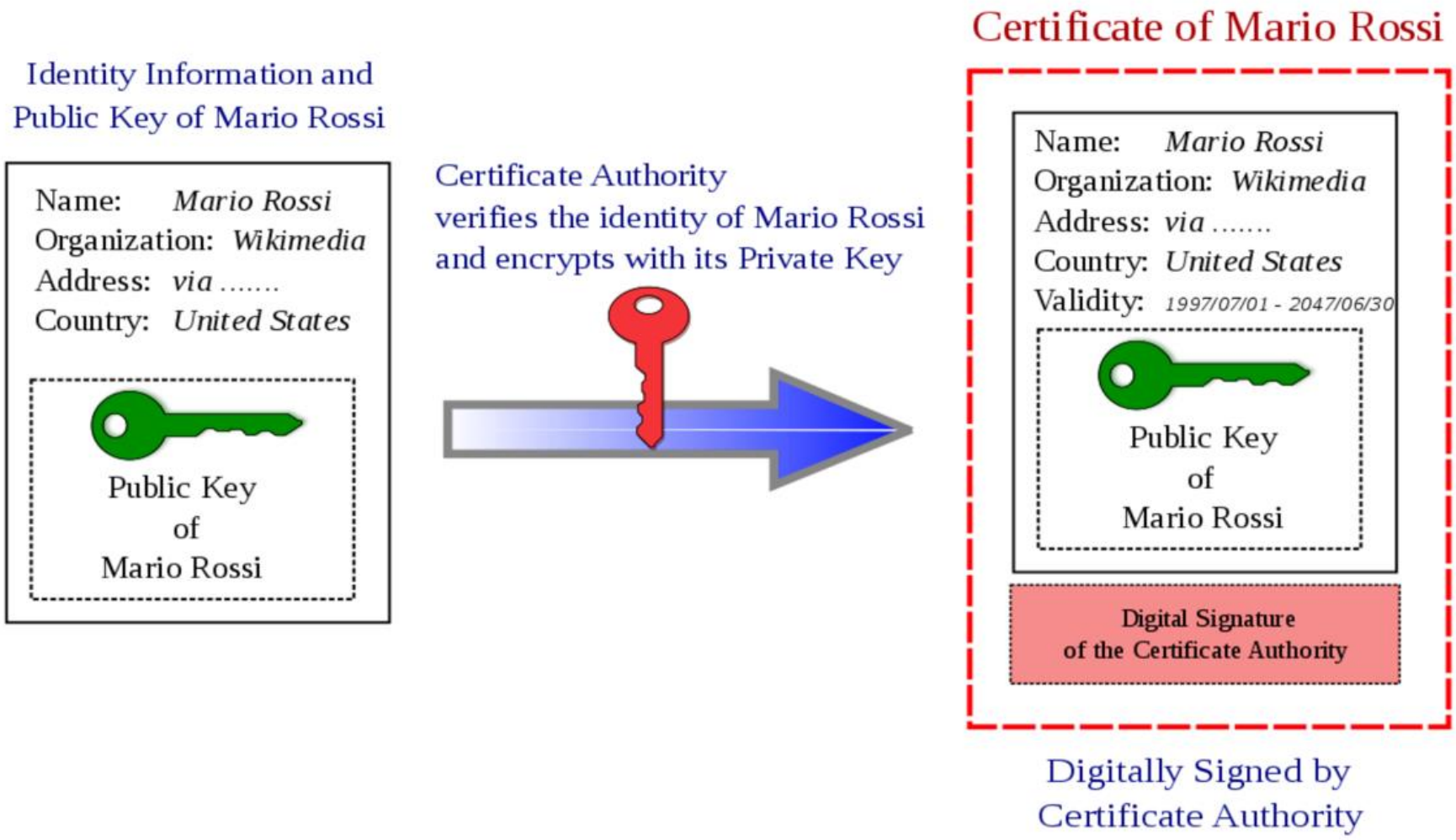
One more question: how does a client verify the server's certificate?

Public Key Certification authorities

- *Certification Authority (CA):* binds public key to particular entity, E.
- Entity E (person, website) 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"



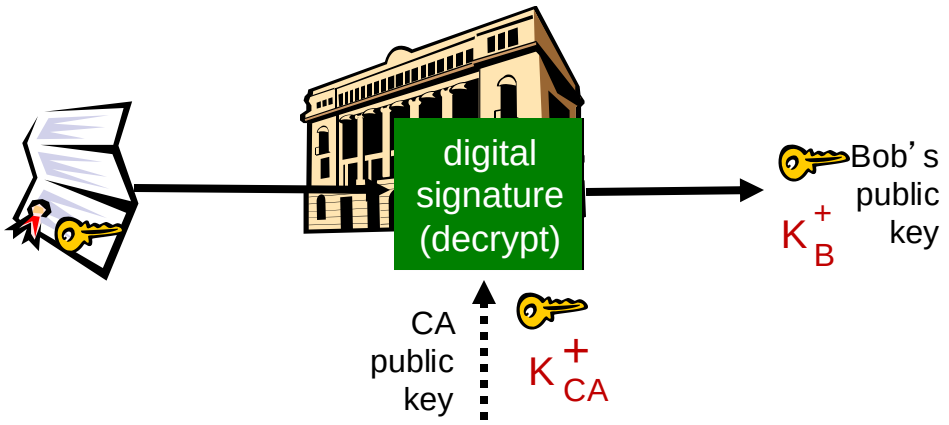
Obtaining a Public key certificate



Certification Authorities

When Alice wants Bob's public key:

- Gets Bob's certificate (from Bob or elsewhere)
- Apply CA's public key to Bob's certificate, to validate Bob's public key



Two more (and *most important*) questions

- Who are those trusted CAs? Who choose them?
- How do end hosts get the CAs' public keys *a prior* and in a secure way?
  - Which CAs' keys that one must have?

Certification authorities

- Commercial certificate providers

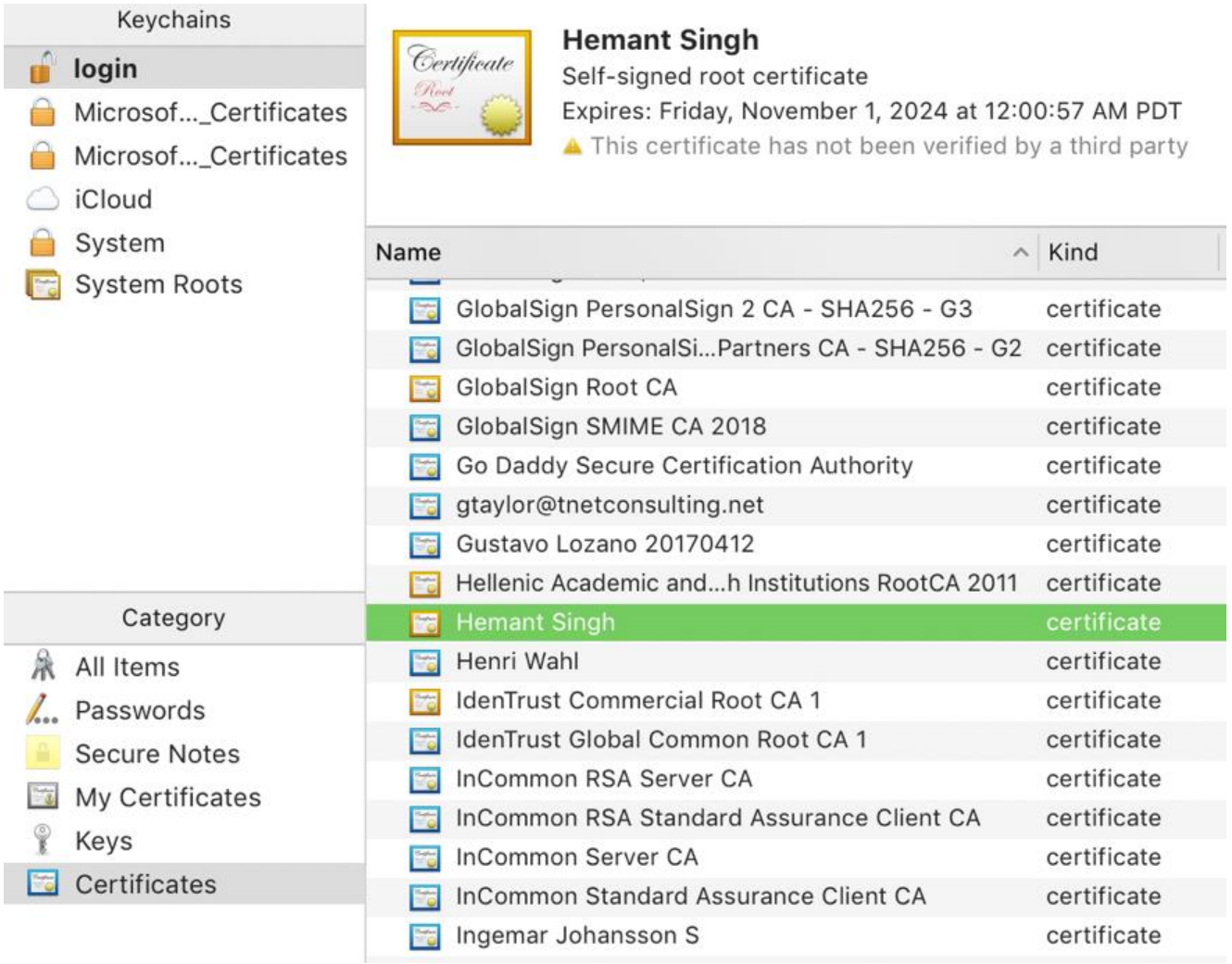
- Free CA: Let's Encrypt

- How do the certificates for all the CAs get into your phones/ computers today?
  - Through browser/operating system
    - Or anti-virus packages/manual configurations

A CA's certificate: self-signed

Who Decides Whom You Trust ?

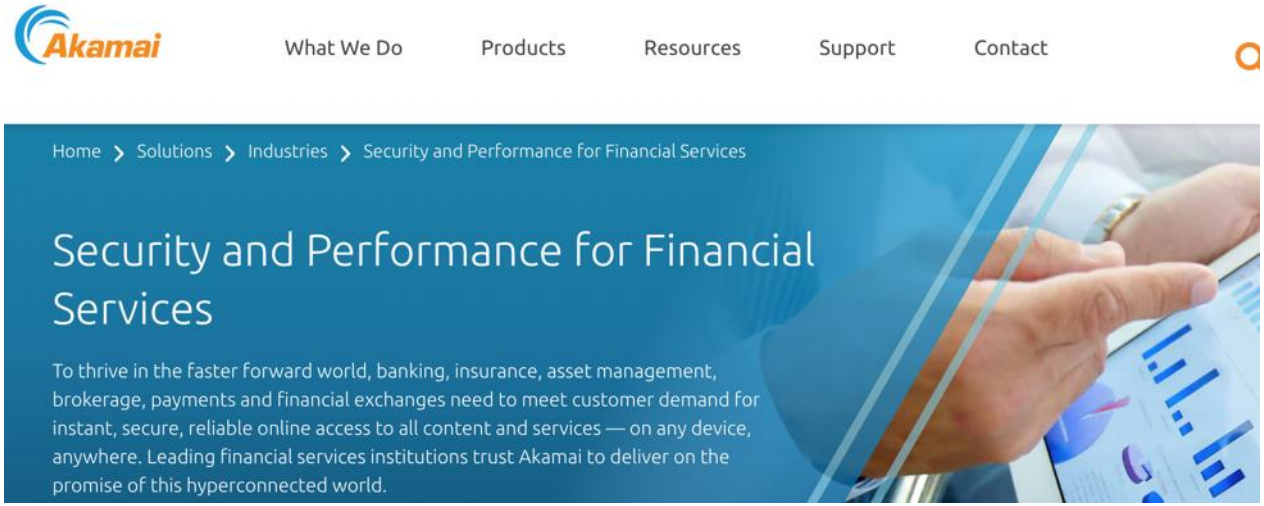
From MACOS Keychain access:



More food for thought

- Ideally how would you want to manage your trust?
- Assume you trust commercial CAs: TLS secures the communication channel between 2 computers; the data is out of protection when out of the TLS connection – a problem?
  - If so, how to solve this problem?
- Online banking: does your laptop really connect to your bank?

Is today's Internet more secure than 10 years ago? Why? or why not?



There is no end-to-end security in today's network applications in general.

More food for thought

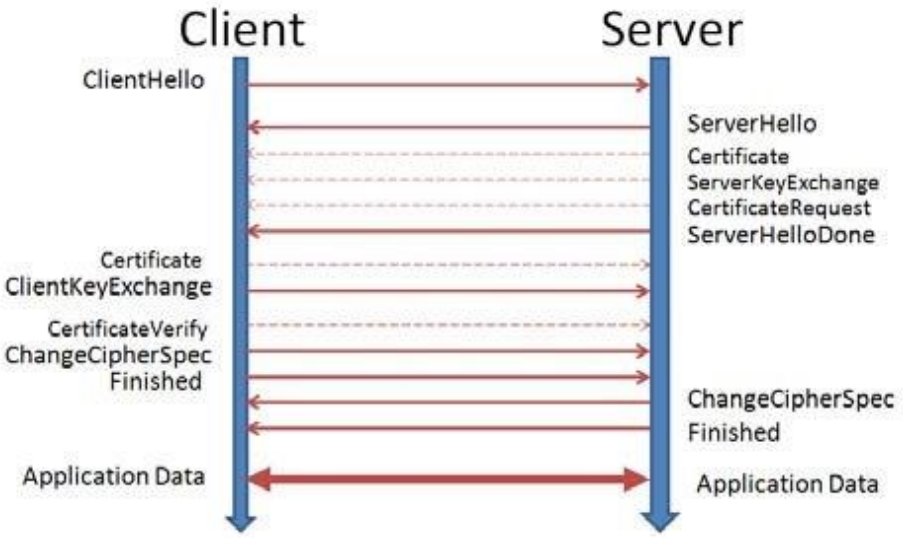
- Now looking back, what HTTPS secures?
  - Confidentiality
  - Integrity
  - Authenticity
- Why we still have HTTPS-enabled phishing websites?
  - paypal.secure-login.com?

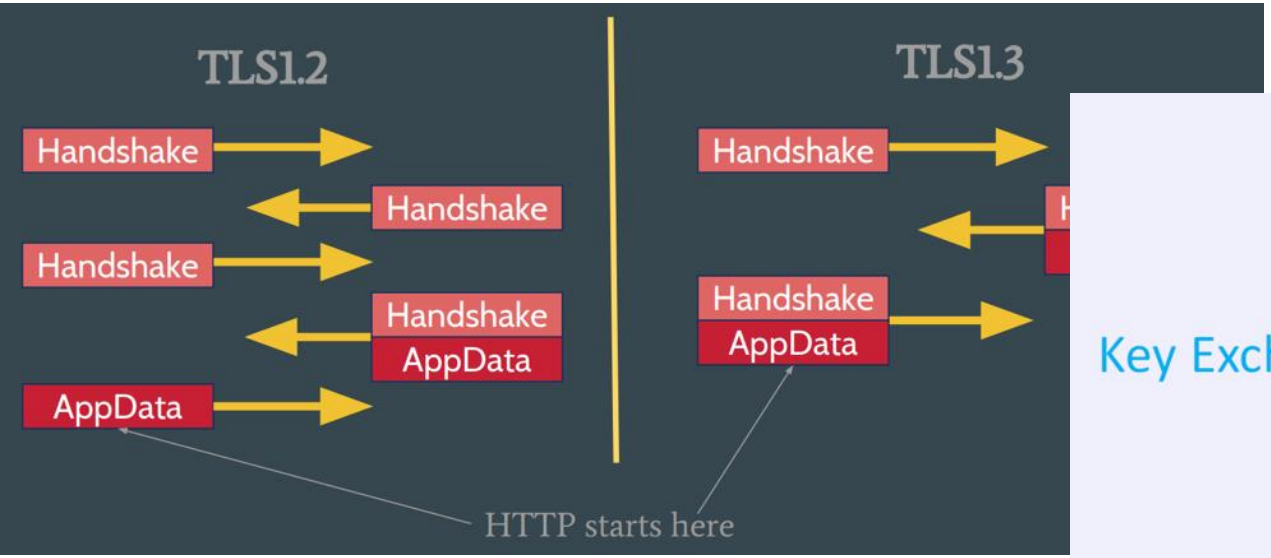
MAC VERSUS HMAC

MAC	HMAC
Short piece of information used to authenticate a message	Specific type of MAC that involves a cryptographic hash function and a secret cryptographic key
MAC stands for Message Authentication Code	HMAC stands for Hash based Message Authentication Code

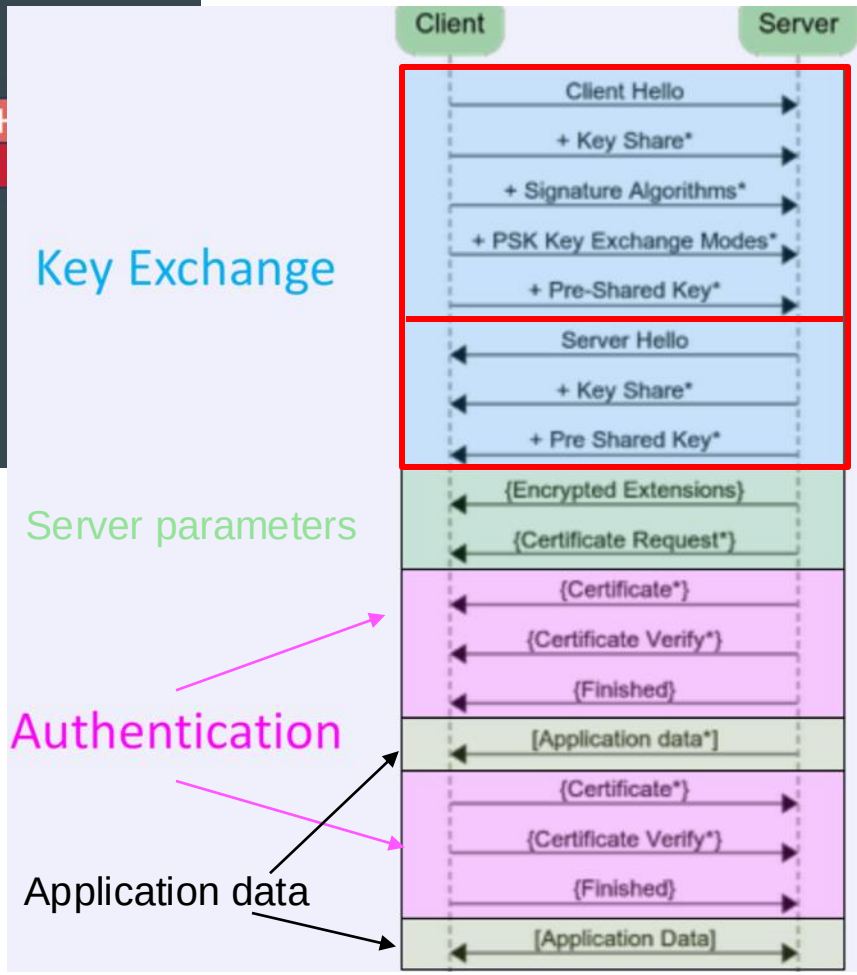
TLS Handshake Protocol

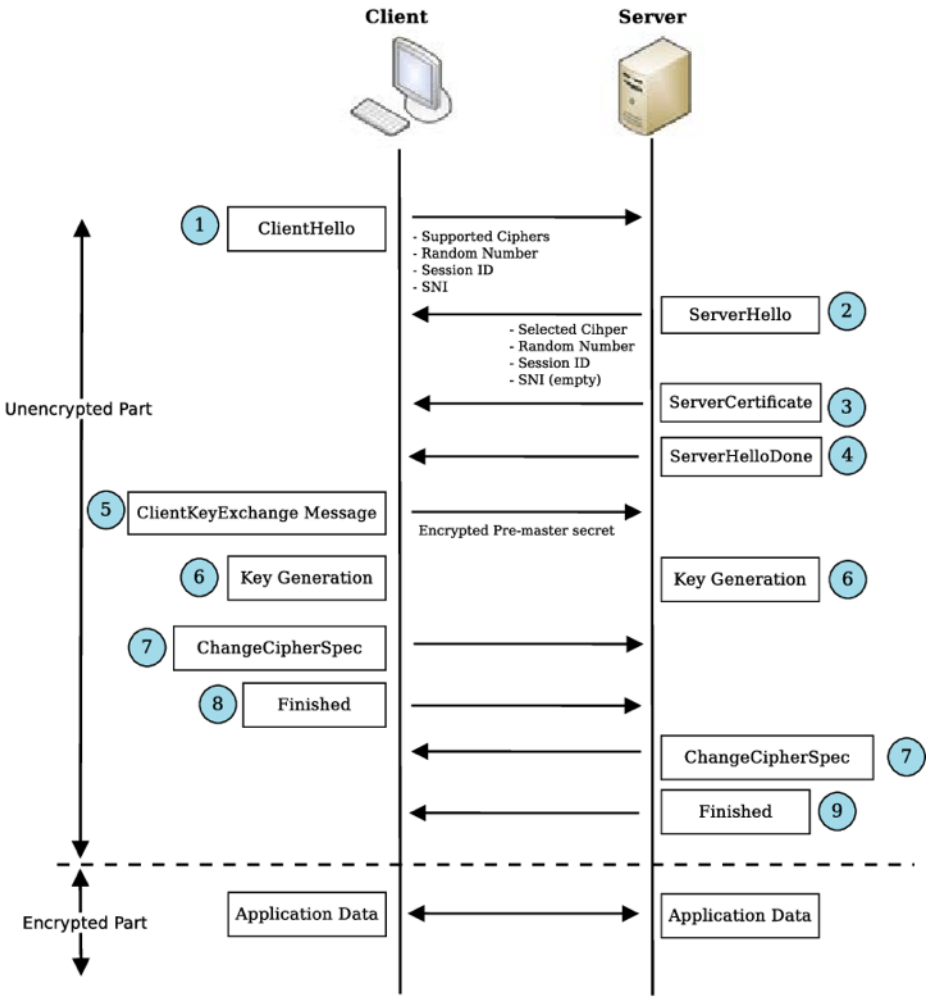
- Runs over the TLS Record Protocol
- Three goals:
  – Agree a cipher suite.
  – Agree a master secret (Diffie–Hellman)
  – Establish trust between Client & Server.

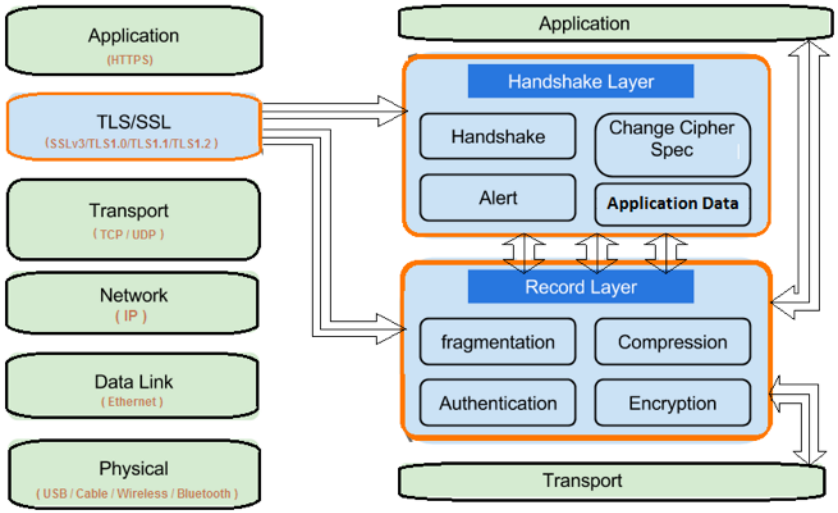


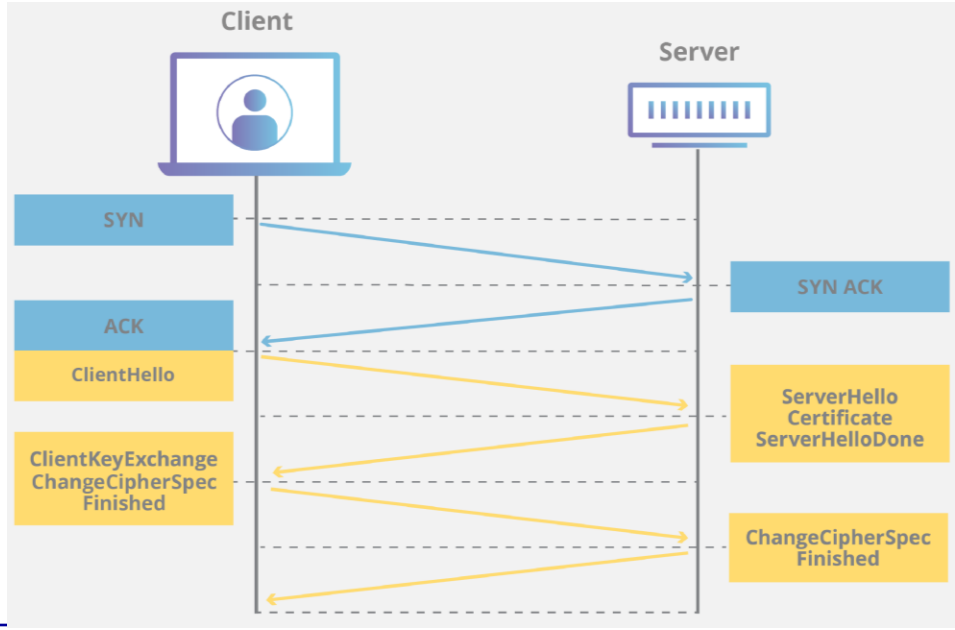


Server parameters
1. Encrypted Extensions
• Server Name
• Message Length
• ...*and optionally many more*
2. Cert request




Client
Server
1
ClientHello
- Supported Ciphers
- Random Number
- Session ID
- SNI
ServerHello
2
- Selected Cihper
- Random Number
- Session ID
- SNI (empty)
ServerCertificate
3
ServerHelloDone
4
Unencrypted Part
5
ClientKeyExchange Message
Encrypted Pre-master secret
6
Key Generation
Key Generation
6
7
ChangeCipherSpec
8
Finished
ChangeCipherSpec
7
Finished
9
Encrypted Part
Application Data
Application Data


Application
(HTTPS)
TLS/SSL
(SSLv3/TLS1.0/TLS1.1/TLS1.2)
Transport
(TCP / UDP)
Network
(IP)
Data Link
(Ethernet)
Physical
(USB / Cable / Wireless / Bluetooth)
Application
Handshake Layer
Handshake
Change Cipher Spec
Alert
Application Data
Record Layer
fragmentation
Compression
Authentication
Encryption
Transport


Client
Server
SYN
SYN ACK
ACK
ClientHello
ServerHello
Certificate
ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished