Network Security

<CH 10>

Youn Kyu Lee
Hongik University

Crypto Protocols

- TLS
  - SSL 2.0 (in 1995) by Netscape
  - SSL 3.0 (in 1996 ~ 2015 (officially deprecated by IETF))
  - TLS 1.0 (in 1999) as an upgrade from SSLv3.0
  - TLS 1.1 (in 2006)
  - TLS 1.2 (in 2008 ~ current): mostly used now
  - TLS 1.3 (in 2018 ~ current)

  these two are recommended as of today, whereas all other versions have been formally deprecated in 2018 by Apple, Google, Microsoft and Mozilla

- Supporting architecture: CA and PKI

What is TLS(formerly SSL)?

- TLS is the protocol used for majority of secure transactions on the Internet
- TLS(Transport Layer Security) and SSL(Secure Sockets Layer)

  Versions in TLS represented as 2-byte values

  SSL 3.0 is 3.0

  TLS 1.0 is 3.1, TLS 1.1 is 3.2, TLS 1.2 is 3.3, TLS 1.3 is 3.4

- if you want to buy a book at amazon.com :
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)

Security at the transport level

- Protecting data such as
  - the web page your browser is trying to load
  - the data your mobile app is displaying back to you
  - the form w/ sensitive information you are about to submit
- Security requirements at the transport level
  - confidentiality
    : C and S want to ensure that data exchanged is readable by them, and them only
    (no reading by anyone else)
  - integrity
    : C and S want to detect data communication errors or data tampering attempt by attackers
  - authentication
    : either C or S, or both C and S want to ensure they know who is on the other side of the
    communication

Security at the transport level

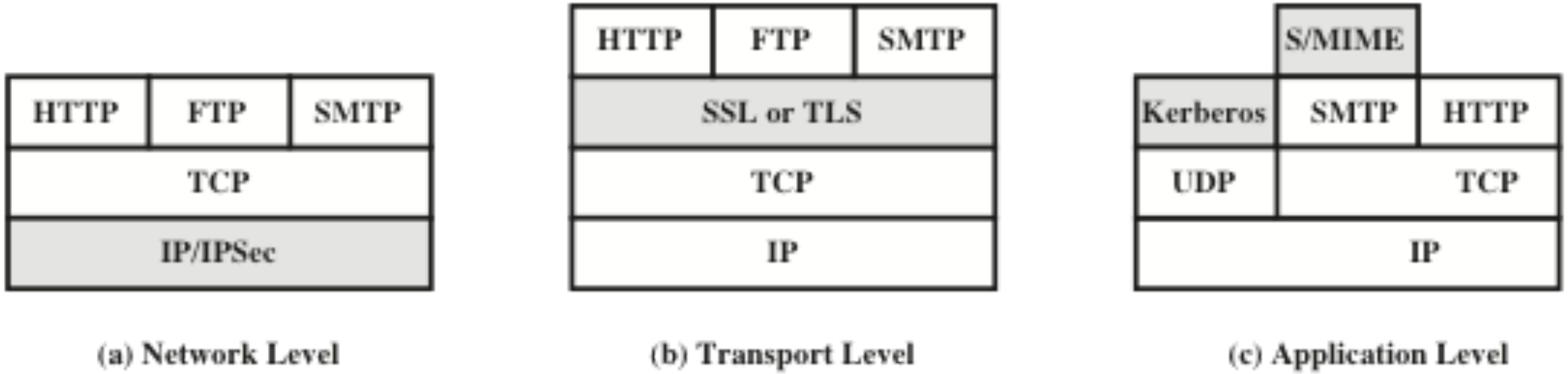


Figure 17.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

© 2017 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

TLS Handshake

- Based on Hybrid Cryptosystem for message protection:
  - public key crypto for secret sharing (slow)
  - symmetric key crypto for encrypting data exchanged (fast)

1) Cipher Suite negotiation

: C and S negotiate the protocol version and cipher suit to be used for secure communication

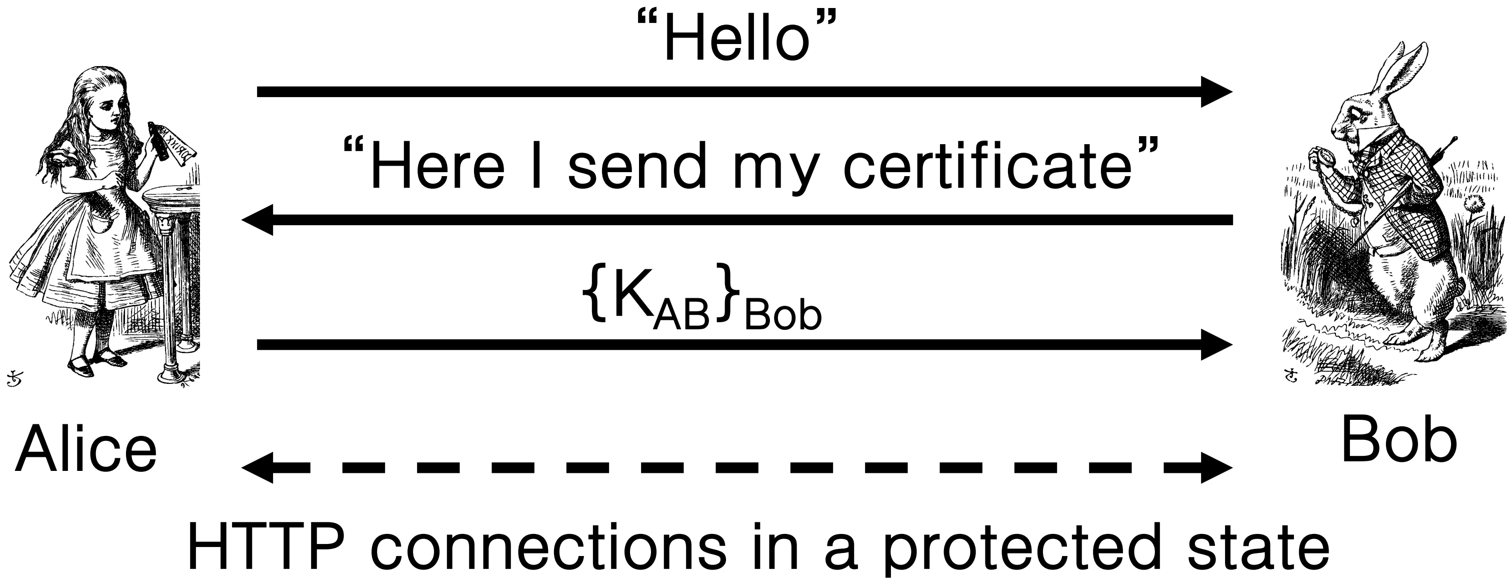
2) Authentication

: C (and sometimes S) validate they are establishing a connection with the intended recipient of the messages

3) Key Exchange

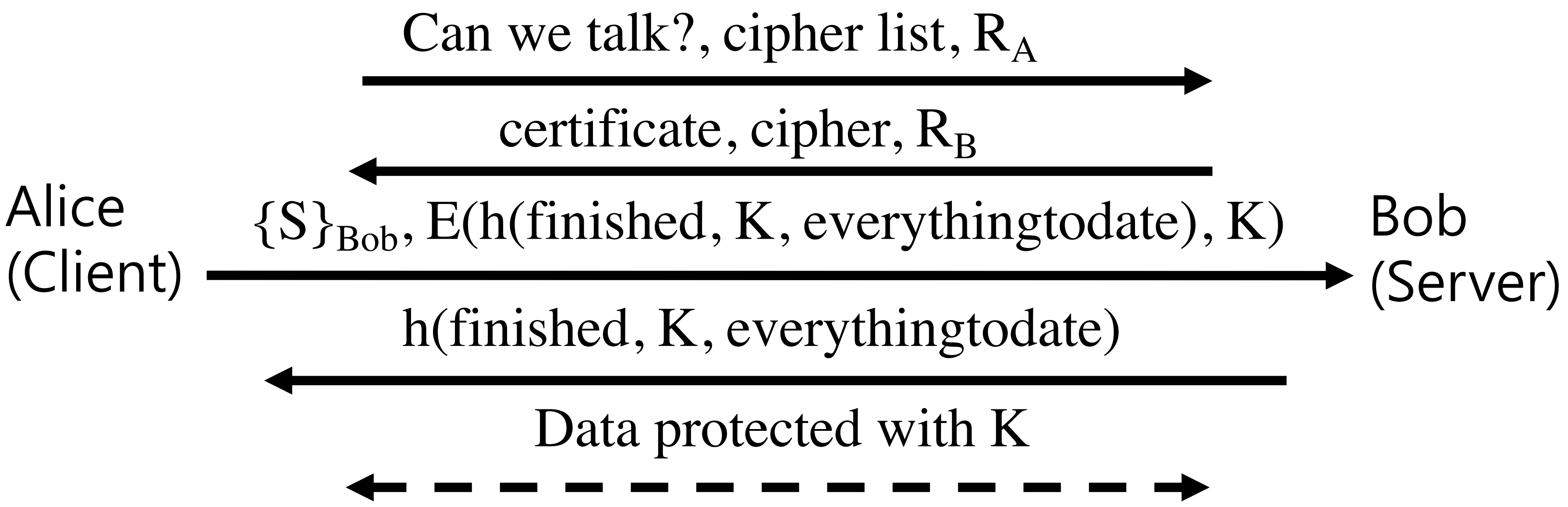
: C and S derive a session key that will be used for the symmetric encryption of the data, once TLS connection established

Simplified Protocol



- Can Alice be sure the person she's talking to is Bob?
- Can Bob be sure the person he's talking to is Alice?

Simplified Protocol



- S is known as pre-master secret
- $K = h(S, R_A, R_B)$: mater secret

1) Cipher Suite negotiation

- Cipher Suite: a collection of cryptographic functions and techniques
  - i) Key Exchange Algorithms: used to securely exchange secret to be used for key generation between C and S
  - ii) Authentication Algorithms: used to authenticate other party using asymmetric crypto and certificates
  - iii) Data Encryption Algorithms: used to encrypt/decrypt data
  - iv) Data Integrity Algorithms: used to detect data errors or data tampering attempts and for deriving key material

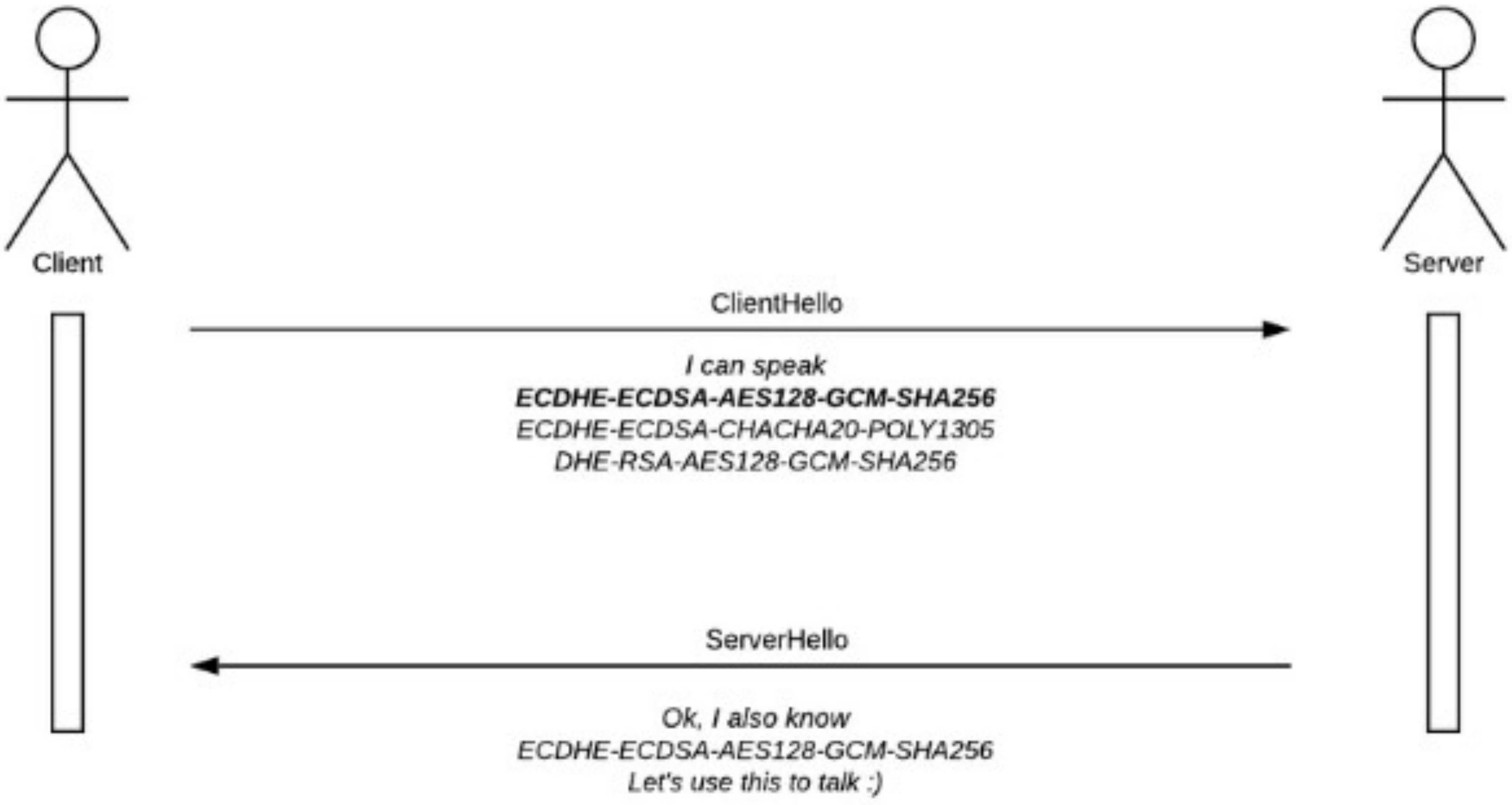
eg. TLS 1.2   ECDHE-ECDSA-AES128-GCM-SHA256

1) Cipher Suite negotiation

```
⌄ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 508
    Version: TLS 1.2 (0x0303)
  > Random: 1396873af8d56db07f55a31afba6c98a04e00025005764fe…
    Session ID Length: 32
    Session ID: fe329526917d48c5af72228bdcb801142894fe91f4a548f7…
    Cipher Suites Length: 34
  ⌄ Cipher Suites (17 suites)
      Cipher Suite: Reserved (GREASE) (0x3a3a)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
```

<Clienthello Packet>

TLS 1.2



TLS v 1.2 cipher suite negotiation

A modern overview of SSL/TLS - TLS 1.2 https://www.paolotagliaferri.com/an-overview-of-ssl-tls-secure-sockets-layer-transport-layer-security-tls-1-2/

2) Authentication

- Based on two types of digital signatures
  · RSA: for both key exchange(TLS 1.2 only) and signature
  · ECDSA: a variant of DSA(Digital Signature Algorithm, NIST standard) using elliptic curve cryptography
  · using Certificate(choice between RSA and ECDSA based)
  : C ensures that S can decrypt and use successfully the pre-master secret by possessing the right private key (if it can't the whole handshake will fail)

cf)

ECC key size	equivalent to	RSA key size
160-bit		1024-bit
224-bit		2048-bit
256-bit		3072-bit
384-bit		7680-bit
521-bit		15360-bit

3) Key Exchange

- (i) RSA Key Exchange(*removed in TLS 1.3*) , or
  (ii) DH Key Exchange

- General
  · C and S need to share a secret securely according to an agreed-upon method
  · once C and S share a value(so-called pre-master secret), both will derive the same keys from it using PRF or HKDF
  · 'ChangeCipherSpec': now I'm ready to apply our established protection scheme
  · 'Finished': first protected message to indicate the finish of my side of this handshake

3) Key Exchange

(i) RSA Key Exchange(*removed in TLS 1.3*)

- C verifies S's certificate by using the CA's public key and (other checks such as verification of the certificate chain of trust and certification revocation status)
- C prepares a pre-master secret, and encrypt it using the RSA public key in the certificate from the server
- S receives the key exchange and decrypts it using its RSA private key => now, C and S share a value(pre-master secret)
- lacks PFS

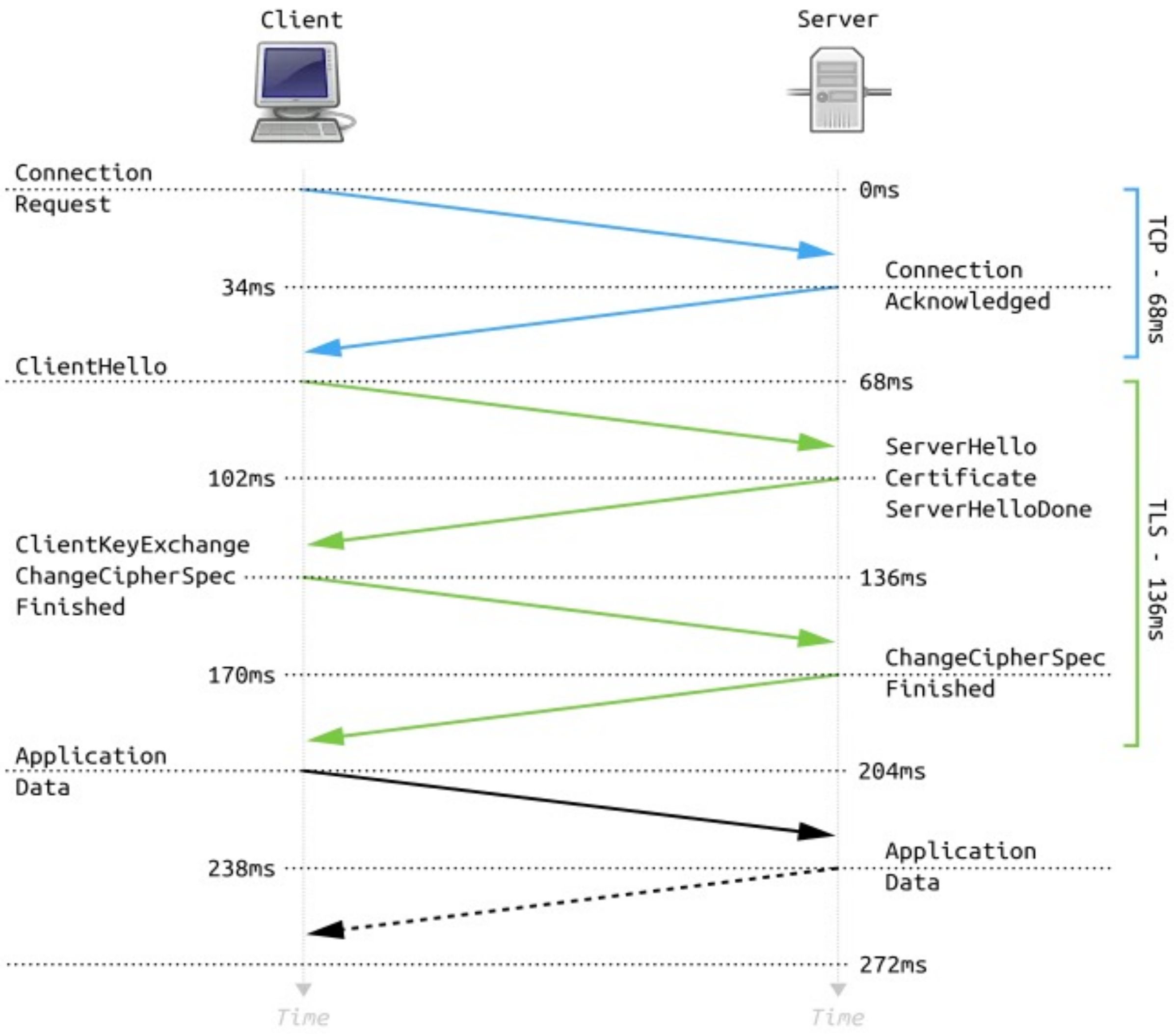
  what if an attacker has recorded a conversation between
  C and S and was able to steal S's decryption private key?

3) Key Exchange

(ii) DH Key Exchange

- both C an S send its public portion of DH exchange to the other and keep its private part as a secret
- each combines it private key with the other's public part to derive the same pre-master secret
- now, C and S share a value(pre-master secret) and from it both can derive the same keys
- it can be "Ephemeral" so that each session generate a different shared value (=> provides PFS)
- *TLS 1.3 allows only some fixed DH parameters known to be secure and only "Ephemeral" mode*

TLS 1.2

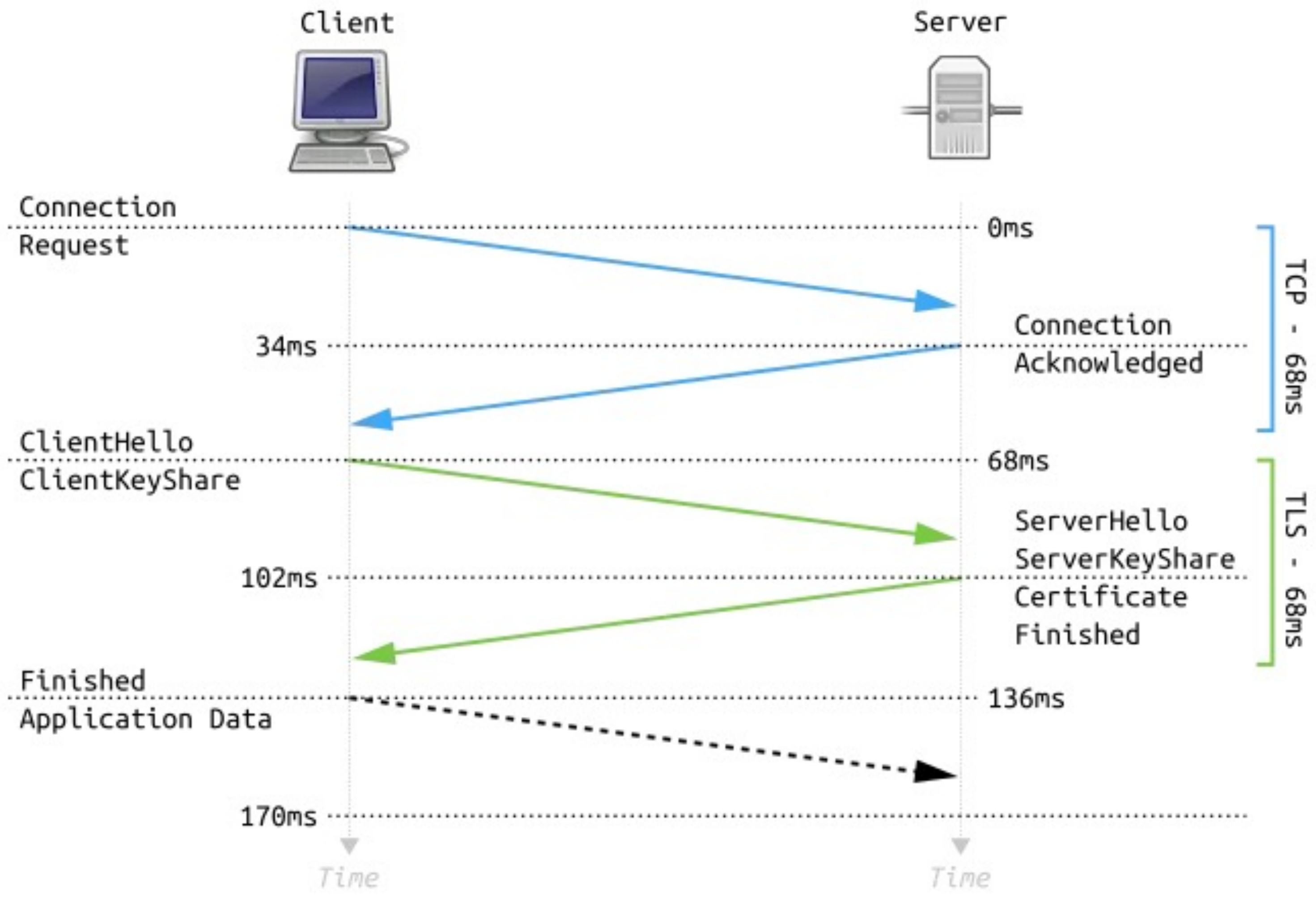


A modern overview of SSL/TLS - TLS 1.2 https://www.paolotagliaferri.com/an-overview-of-ssl-tls-secure-sockets-layer-transport-layer-security-tls-1-2/

TLS 1.3 (compared to TLS 1.2)

- Eliminates support for outmoded algorithms and ciphers
- Eliminates RSA key exchange, mandates PFS
  (DHE or ECDHE only; so will never send pre-master secret to S, and C and S computes it locally)
- Reduces the number of negotiations in the handshake
- Reduces the number of algorithms in a cipher suite
- Eliminates block mode ciphers and mandates AEAD bulk encryption (eg. AES256-GCM)
  vs. MAC-then-Encrypt in TLS 1.2 and earlier
- Uses HKDF cryptographic extraction and key derivation
- Signs the entire handshake, an improvement of TLS 1.2
- Supports additional elliptic curves(twisted Edwards curve) for signatures, EdDSA

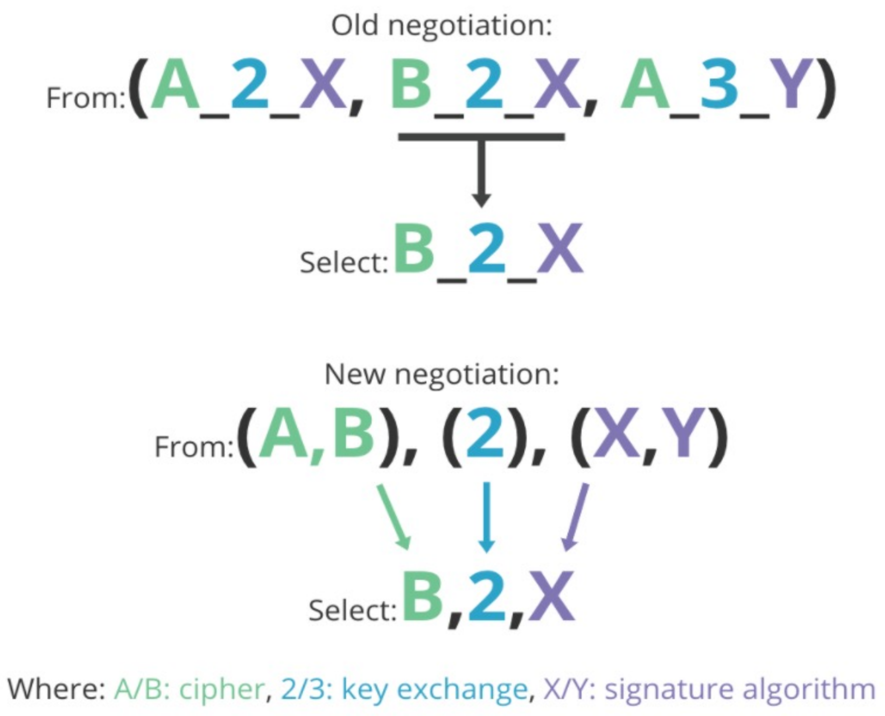
TLS 1.3



An overview of SSL/TLS -TLS 1.3 https://www.paolotagliaferri.com/overview-of-transport-layer-security-protocol-tls-1-3/

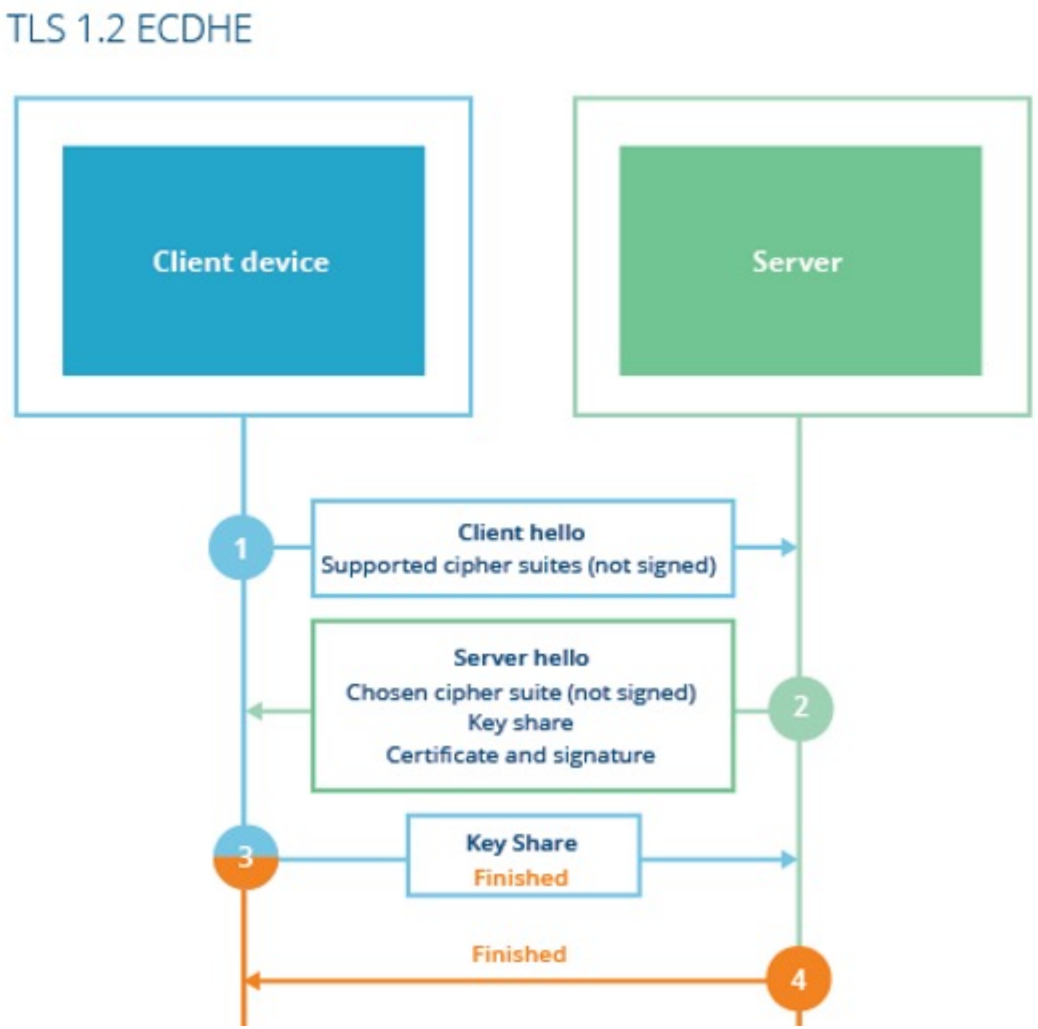
TLS 1.3

- Cipher : Key Exchange : Signature(Authentication)



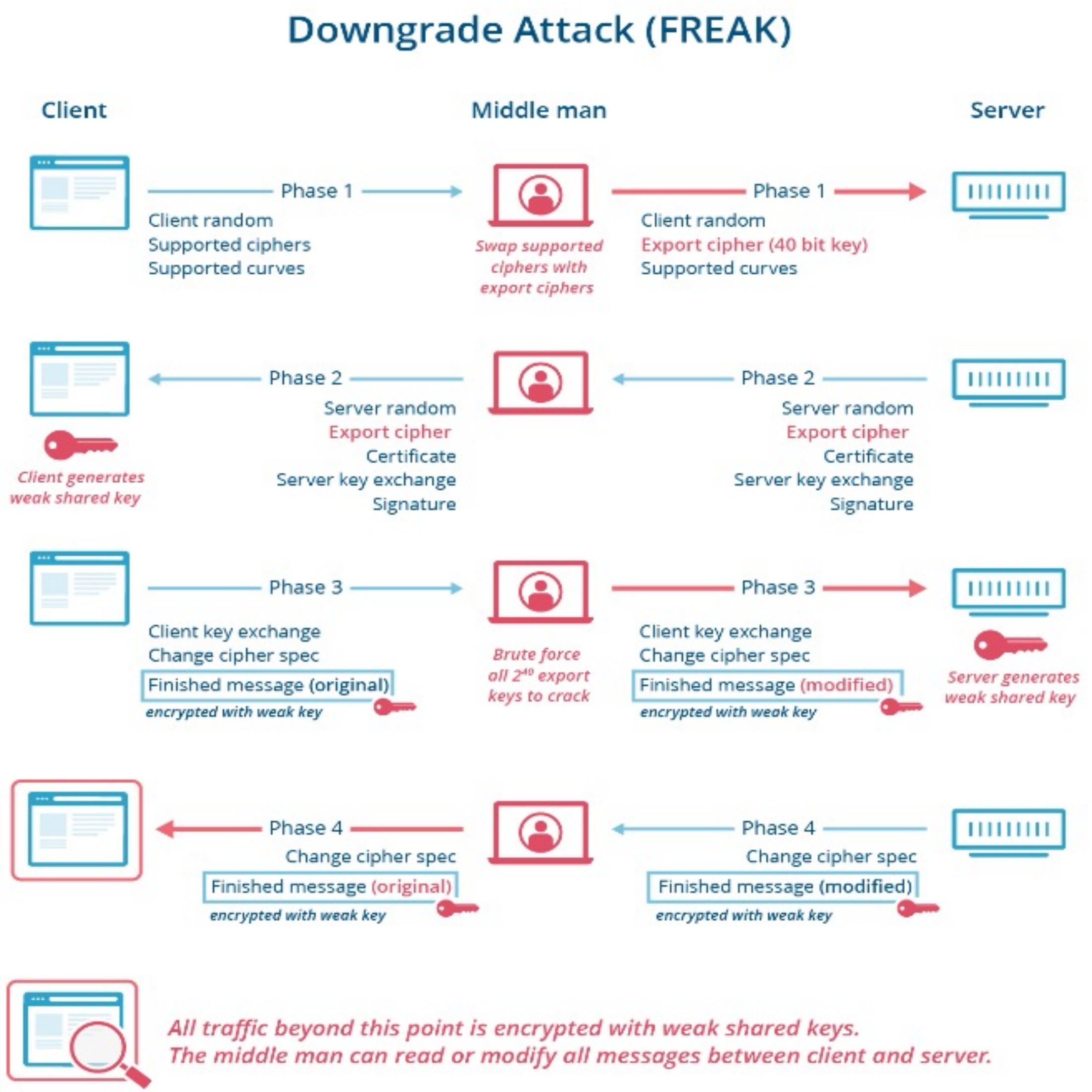
TLS 1.3

- Signing the entire transcript



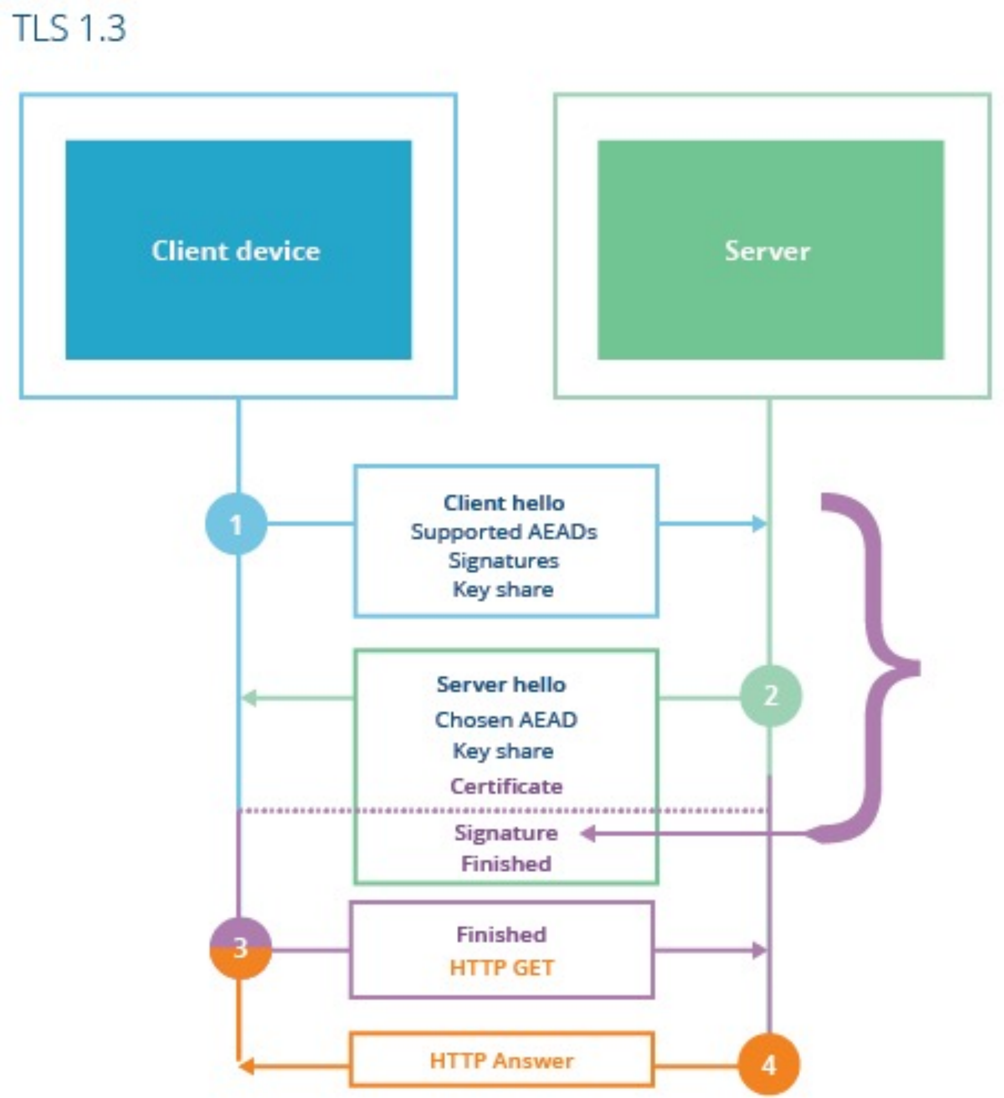
A Detailed Look at RFC 8446 (a.k.a. TLS 1.3) https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

FREAK



A Detailed Look at RFC 8446 (a.k.a. TLS 1.3) https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

TLS 1.3



A Detailed Look at RFC 8446 (a.k.a. TLS 1.3) https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

(Crypto) Supporting Architecture - CA and PKI

- In 1990s and early 2000s, lots of companies/organizations set up CAs and software firms such as Microsoft and Netscape embedded their public keys into their browsers

- Consider: a security agency can control a CA that would produce a certificate on `www.gmail.com` for the agency public key that the target's browser would accept

  (in 2011, attacker penetrated the DigiNotar (a Dutch CA) servers and made them have issued wildcard certificates for *.google.com, giving the attacker the ability to impersonate Google to any browser that trusted the certificate. It was reported that Iranian agents had monitored 300,000 Gmail users in Iran.)

(Crypto) Supporting Architecture - CA and PKI

- Open PKI vs. Closed PKI

  if you are building a service that government agencies are likely to attack, then it may be a good idea to keep your PKI closed, with a CA that runs on your premises

- Other Issues of trust

  - if you remove one of the root certificates from Firefox, then Mozilla silently replaces it. (no choice but to accept all) Same with Windows.
  - agency had its cert in Windows, but not in other browsers (resort to different surveillance method for Mac users)
  - users have been trained to ignore security warnings (such as on out-of-date certs)

(Crypto) Supporting Architecture - CA and PKI

- Other Issues of trust
  - certs bind a company name to a domain, but CAs usually are not validating thoroughly
    eg. they hand out certs after checking the applicant can answer email sent to the domain, …
  - certification revocation matters
    - · download CRL from the CA
      - and check any cert on which they are about to rely against the CRL
      - large CRLs lead network delay and congestion
    - · OCSP(Online Certificate Status Protocol)
      - for online status checking
      - more efficient and people are moving to OCSP from consulting CRLs

Q & A

aiclasshongik@gmail.com