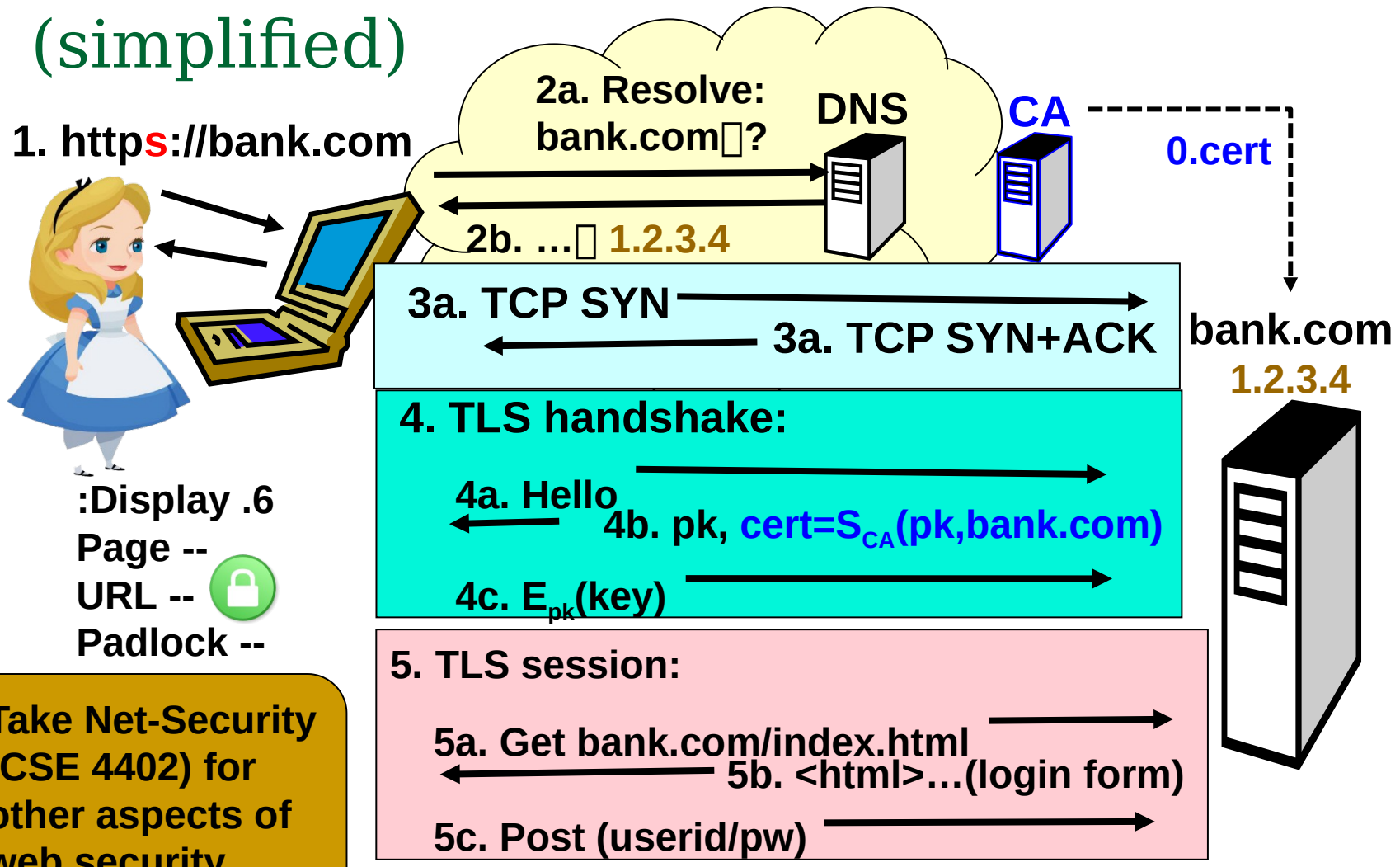
CSE 3400 - Lecture set 7: Transport Layer Security (TLS) and Secure Socket Layer (SSL)

Last updated: 3/31/20

Web Security with TLS/SSL

(simplified)



Take Net-Security (CSE 4402) for other aspects of web security

TLS/SSL: Security Goals

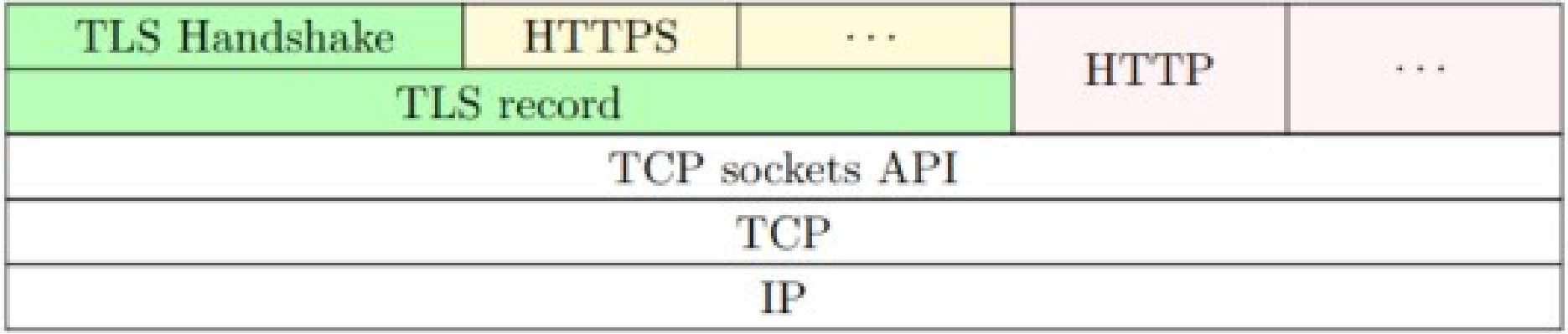
- Connection integrity and confidentiality
- Key exchange: setup shared key
- Server authentication
- Client authentication (optional – and rarely used)
- Cipher agility
- Robust crypto
- Perfect forward secrecy
- MitM attacker model

TLS/SSL: Engineering Goals

- Efficiency
  - Session resumption
  - Minimizing round trips
- Extensibility and versatility
- Ease of deployment and use

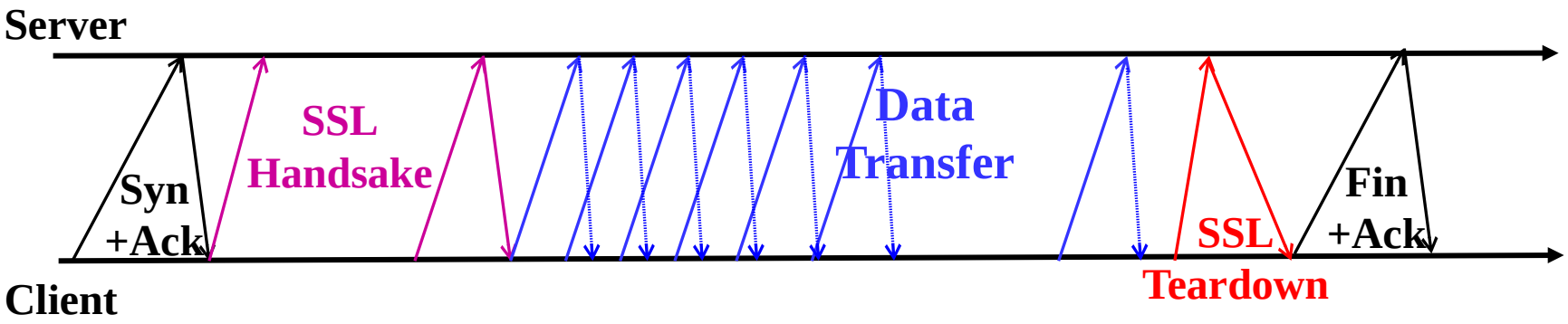
SSL/TLS Architecture

- SSL/TLS is built in two layers:
  - **Handshake Layer –** server[+client] auth, key exchange, cipher suite negotiation, extensions...
  - **Record Layer –**secure communication between client and server using exchanged session keys



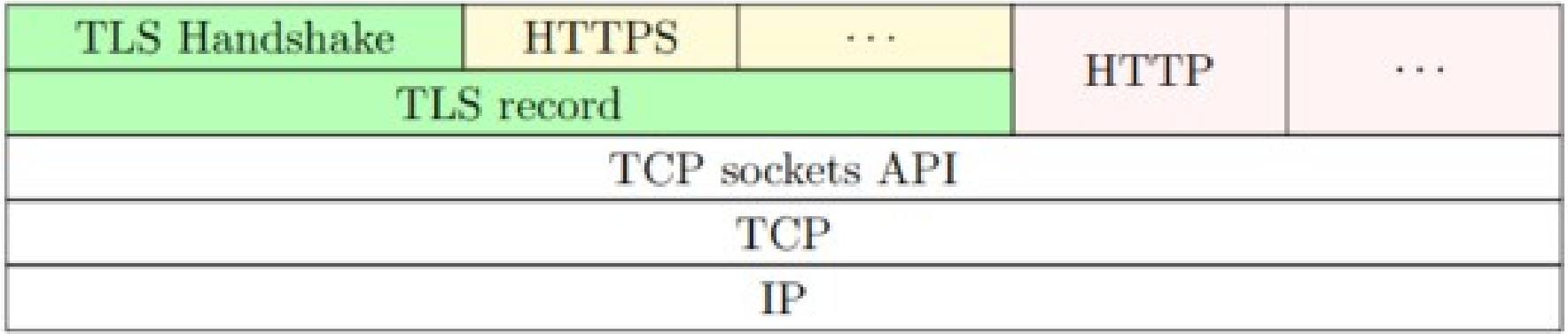
TLS/SSL Operation Phases (high level)

- TCP Connection setup (Syn+Ack)
- Handshake (key establishment)
  - Negotiate (agree on) algorithms, methods
  - Authenticate server and optionally client, establish keys
- Data transfer
- Secure Teardown (why?)
- TCP connection closure (Fin+Ack)



SSL/TLS and Applications

- SSL/TLS is (just) a library
- Deployed by applications, independent of OS
  - Easy to adopt widely deployed:
    web (https), email, SSH, mobile-apps, games…
- Risk: broken implementations

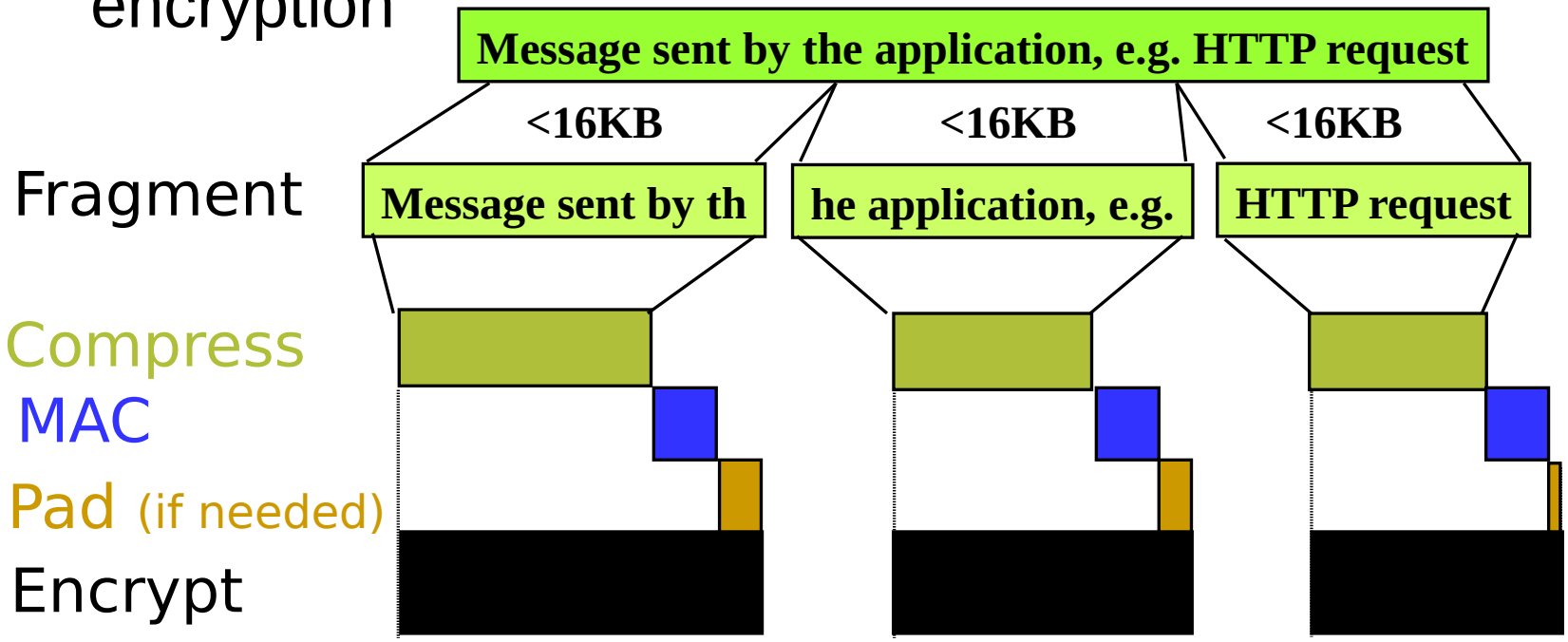


SSL Record Layer

- Assumes underlying reliable communication (TCP)
- Four services (in order):
  - Fragment: break TCP stream into fragments (<16KB)
    - Pipeline: send processed frag 1 while processing 2 and receiving 3
  - Compress (lossless) each fragment
    - Reduce processing, communication time
    - Ciphertext cannot be compressed – must compress before
    - Risk: exposure of amount of redundancy *compression attacks*
  - Authenticate: [seq#||type||version||length||comp_fragment]
  - Encrypt
    - After padding (if necessary)
- Finally, add header: type (protocol), version & length

TLS(till 1.3)/SSL Record Layer

- Assumes underlying reliable communication (TCP)
- Fragmentation, compression, authentication, encryption



Send each fragment via TCP

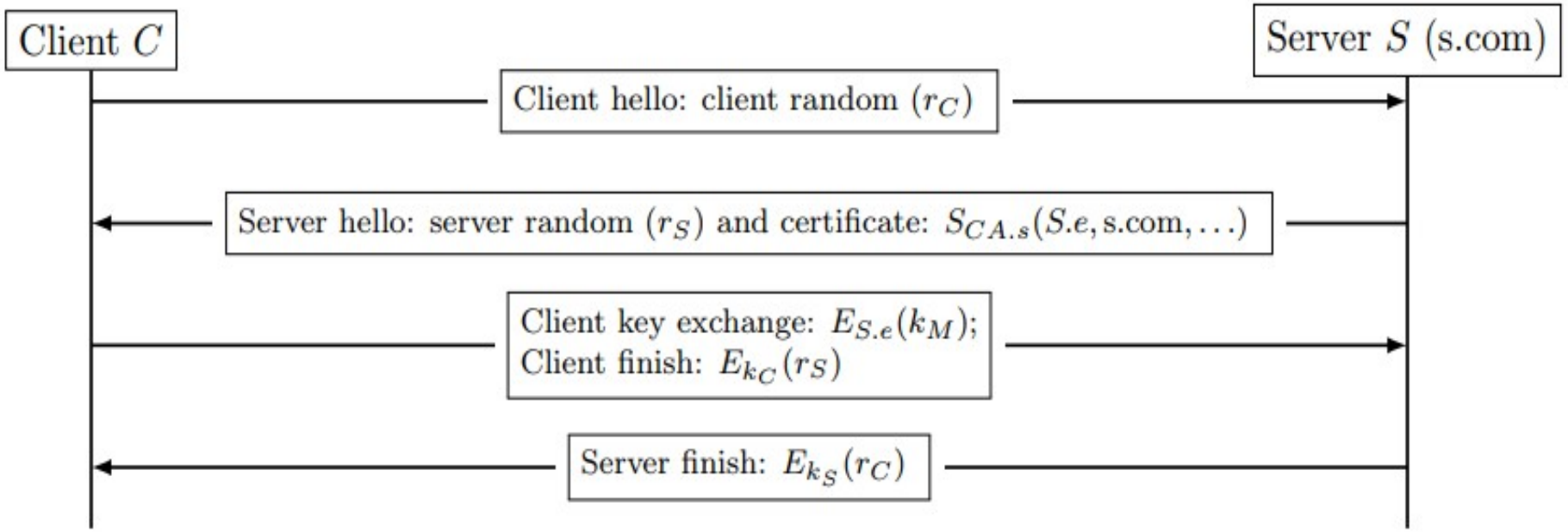
Record Layer Vulnerabilities

- Surprisingly many found, exploited!
- SSL, TLS1.0: vulnerable record protocol
  - Examples…
  - Attacks on RC4 to be avoided
  - CBC IV reuse in session (BEAST)
  - `MAC-then-Encrypt': padding attacks [Lucky13,POODLE, …]
  - Compress-then-encrypt: CRIME, TIME
- Our focus is handshake
  - Includes: downgrading to use vulnerable version!

SSL/TLS Handshake Protocol

- The beginning: SSLv2
  - SSLv1 was never published, released
- The evolution: from SSLv3 to TLS 1.2
  - TLS: the IETF version of SSL
- State-of-Art: TLS 1.3
  - Significant changes
- Our focus is on the handshake protocol

Simplified SSLv2 Handshake



- Key derivation in SSLv2:
  - Client randomly selects and sends to server
  - Client and server derive (directional) encryption keys:

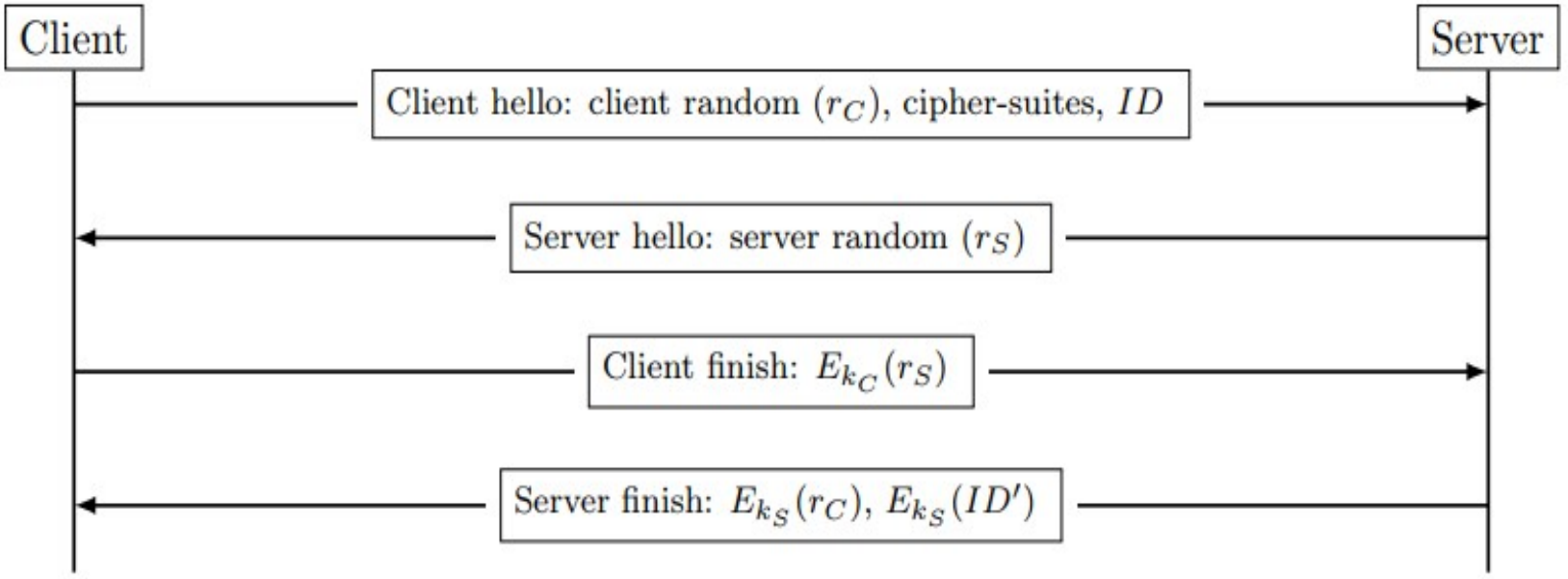
$$k_C = MD5(k_M || \text{“0”} || r_C || r_S) \qquad k_S = MD5(k_M || \text{“1”} || r_C || r_S)$$

SSLv2: important concepts

- Derive, from master key , two separate keys:
  - , for protecting traffic from client to server
  - , for protecting traffic from server to client
  - Nonces , protect against replay
    - Even if client reuses same PK encryption of
- Sessions: reusing public-key operations
- Cipher-agility
- Optional client authentication

SSLv2 Session Resumption

- Goal: cache shared master key (and *ID*)
  - Client identifies cached key by sending *ID* (if known)
  - If server knows *ID*, it sends only nonce (no cert req')
  - Server sends (new) identifier *ID'* at end of handshake

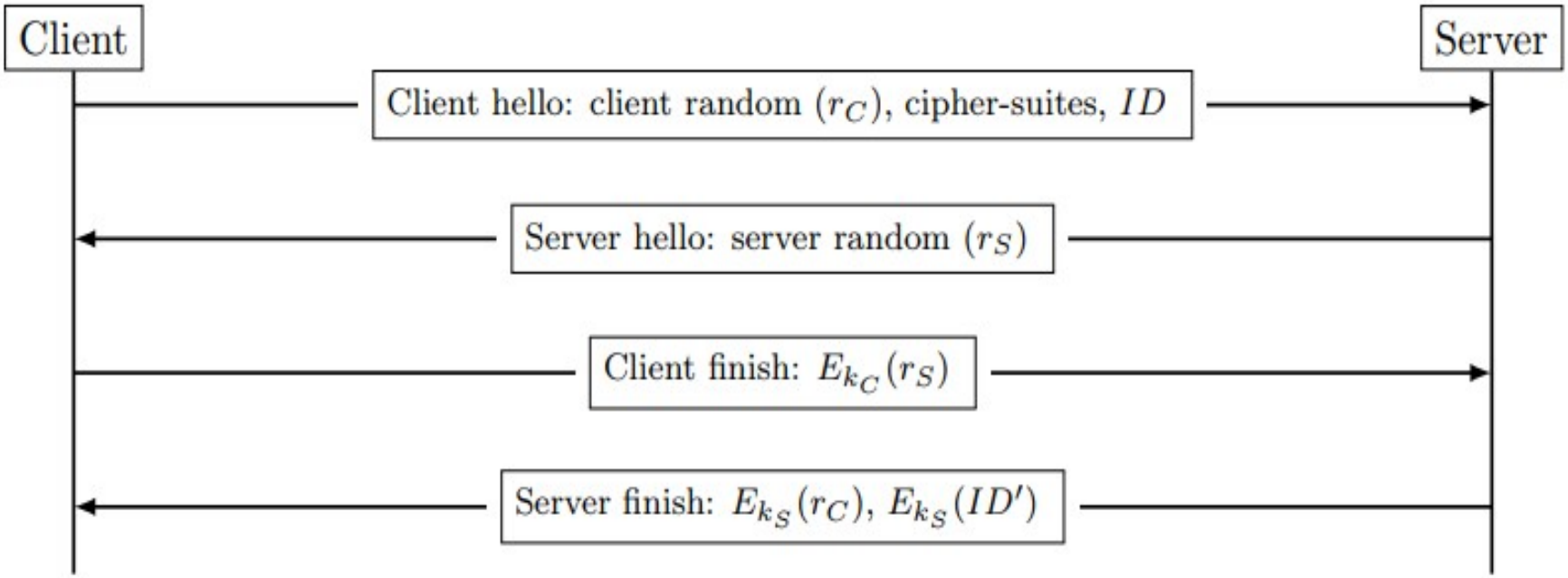


SSLv2 Session Resumption: Exercise

- Demonstrate (replay) MitM attack on SSLv2, if using a fixed value for:

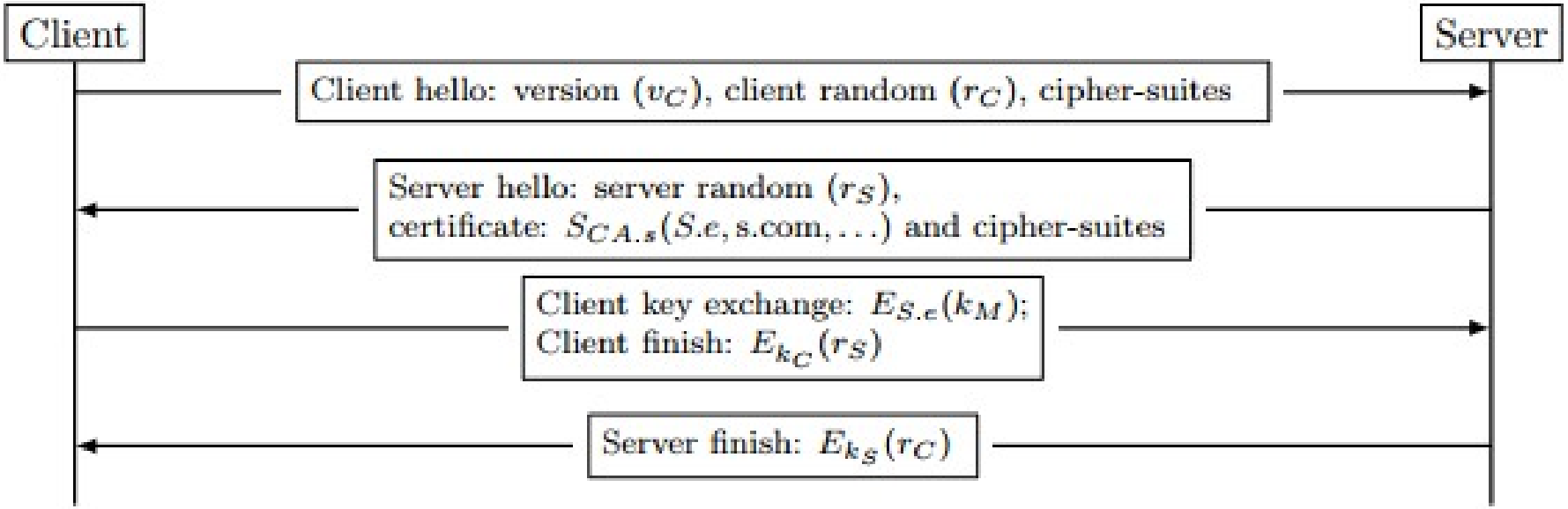
  (1) Client random,

  (2) Server random



SSLv2 Ciphersuite Negotiation

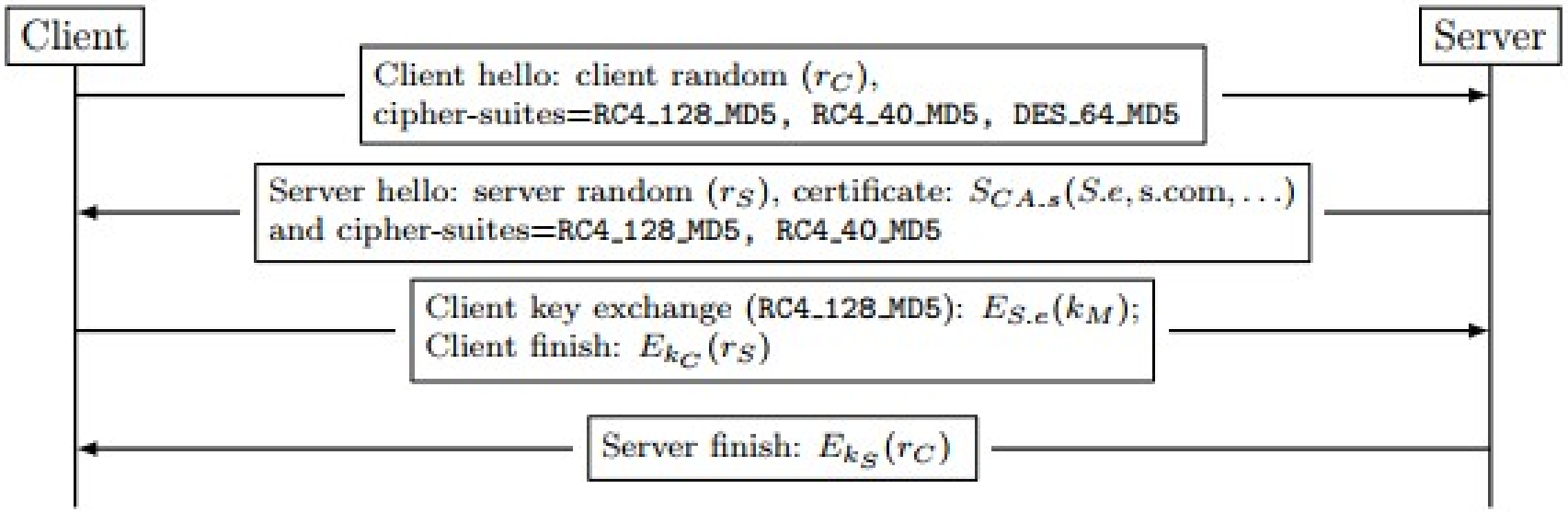
- Client, server sends cipher-suites
- Client specifies choice in client-key-exchange



- Example…

SSLv2 Ciphersuite Negotiation

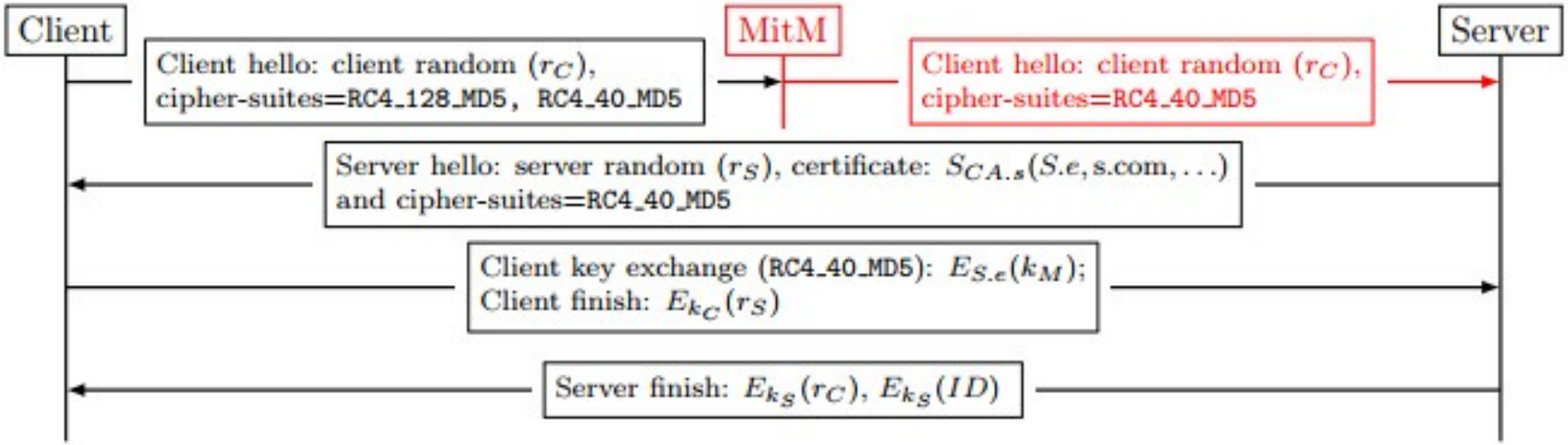
- Client, server sends cipher-suites
- Client specifies choice in client-key-exchange



- Example: RC4_128_MD5 chosen
- Vulnerable to downgrade attack!

SSLv2 Downgrade Attack

- Server and client tricked into using (insecure) 40-bit encryption (`export version')



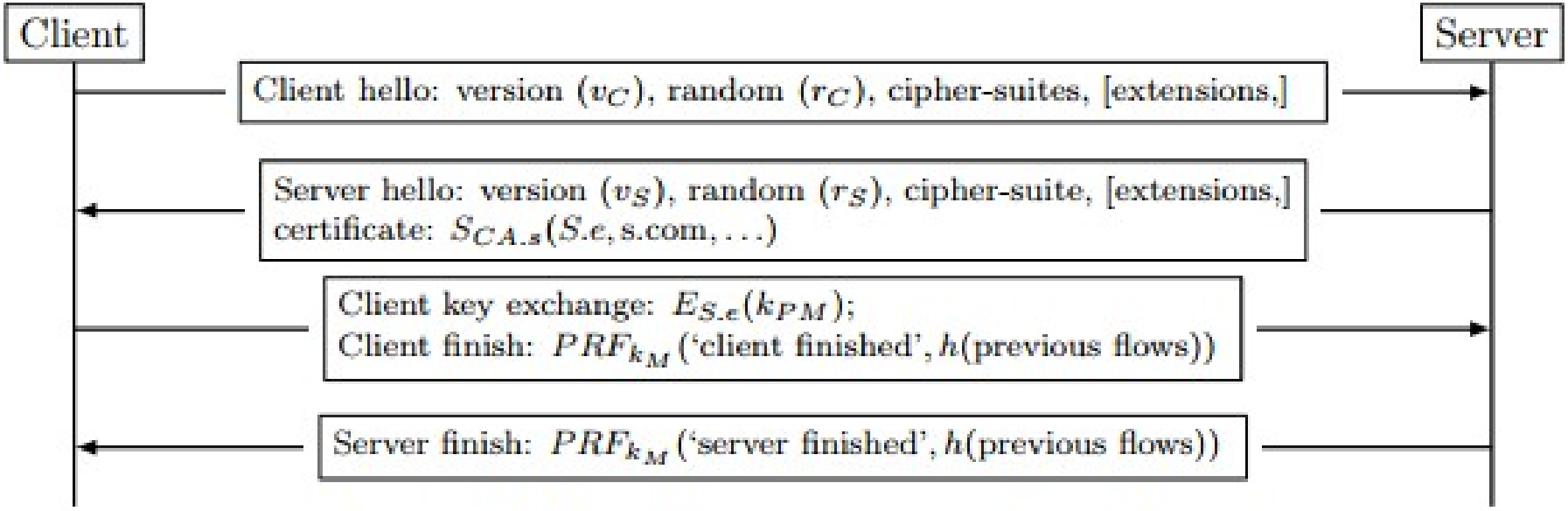
- Attacker may record connection and decrypt later – no need for real-time cryptanalysis!

The evolution: SSLv3, TLS1.0, 1.1, 1.2

- Main improvements:
  - Improved key derivation
    - Premaster key master key connection keys
  - Improved negotiation and handshake integrity
    - Prevents SSLv2 downgrade attack
    - Secure extensions, protocol-negotiation, & more
  - DH key exchange and PFS
    - SSLv2 allowed only RSA; TLS 1.3: only PFS
  - Session-ticket resumption

Basic RSA Handshake: SSL3-TLS1.2

- Used in SSLv3 and TLSv1.0, 1.1 and 1.2



$$k_M = PRF_{k_{PM}}(\text{“master secret”} \| r_C \| r_S)$$

$$key\text{-}block = PRF_{k_M}(\text{‘key expansion’} \| r_C \| r_S)$$

k_C^A	k_S^A	k_C^E	k_S^E	IV_C	IV_S

SSL3-TLS1.2: Key Derivation

- Handshake exchanges premaster key
- Derive master key:

$$k_M = PRF_{k_{PM}}(\text{“master secret”}||r_C||r_S)$$

  - Why this extra step of premaster key?
  - In case premaster key is not (fully) random
    - Weak randomness at a (weak) client
    - Weak client reuses same PK-encrypted key
    - DH-derived premaster key

SSL3-TLS1.2: Key Derivation

- Handshake exchanges premaster key
- Derive master key:

$$k_M = PRF_{k_{PM}}(\text{“master secret”}\|r_C\|r_S)$$

- Derive key block from master key:

$$key\text{-}block = PRF_{k_M}(\text{‘key expansion’}\|r_C\|r_S)$$

- Chop keys from key-block:

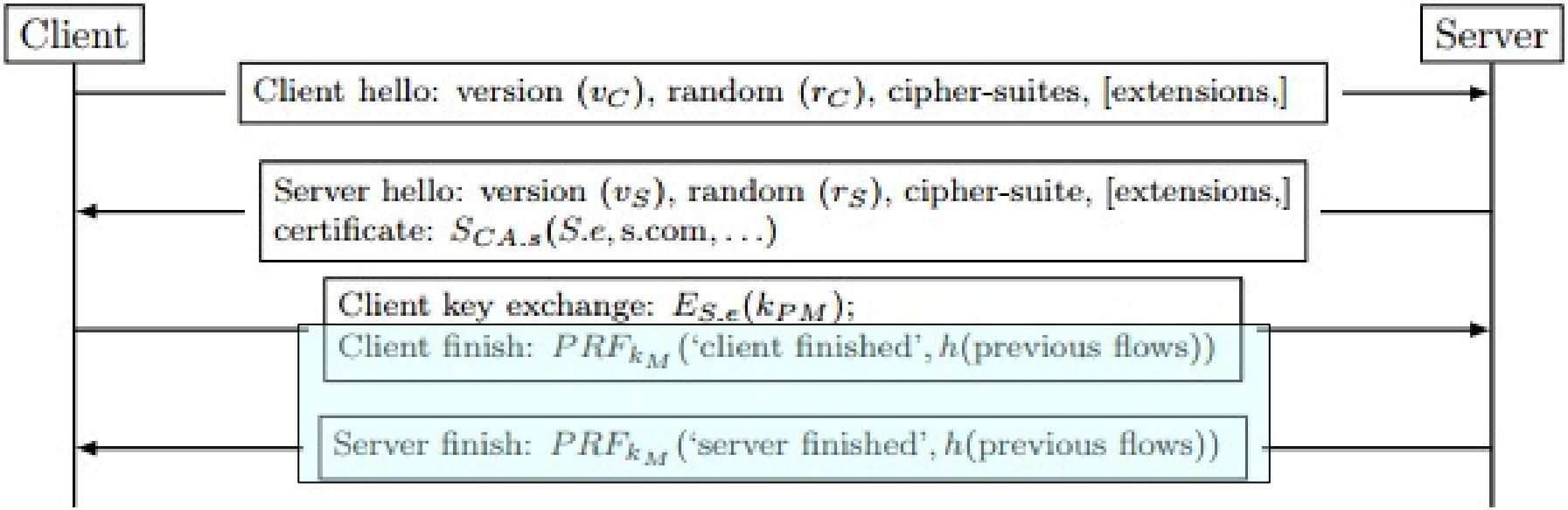
$key\text{-}block = PRF_{k_M}(\text{‘key expansion’}\|r_C\|r_S)$					
k_C^A	k_S^A	k_C^E	k_S^E	IV_C	IV_S

SSL3-TLS1.2: Agility and Integrity

- SSLv2: limited cipher-agility (ciphersuites)
  - And no integrity: vulnerable to downgrade attack
- SSLv3 to TLS1.2: integrity + improved agility:
  - Handshake integrity – foils downgrade attack!
  - Backwards compatibility
  - TLS extensions
  - Version-dependent key separation

SSL3-TLS1.2: Handshake integrity

- Foils the downgrade attack on SSLv2
- Extend the finish-message validation: authenticate entire previous handshake flows
  - Some differences between versions: simplified



SSL3-TLS1.2: Backwards compatibility

- Challenge: upgrading existing protocol
  - Unrealistic: all upgrade at same day
  - Backward compatibility: new (server, client) can still work with old (client, server)
    - Server selects version based on client's (in 'hello')
    - Downgrade prevented using 'finish' authentication
- Dilemmas for clients:
  - Some servers fail to respond to new handshake
  - 'Downgrade-dance' clients: try new versions, then older vulnerable!
    - Exercise 7.3

SSL3-TLS1.2: Backwards compatibility

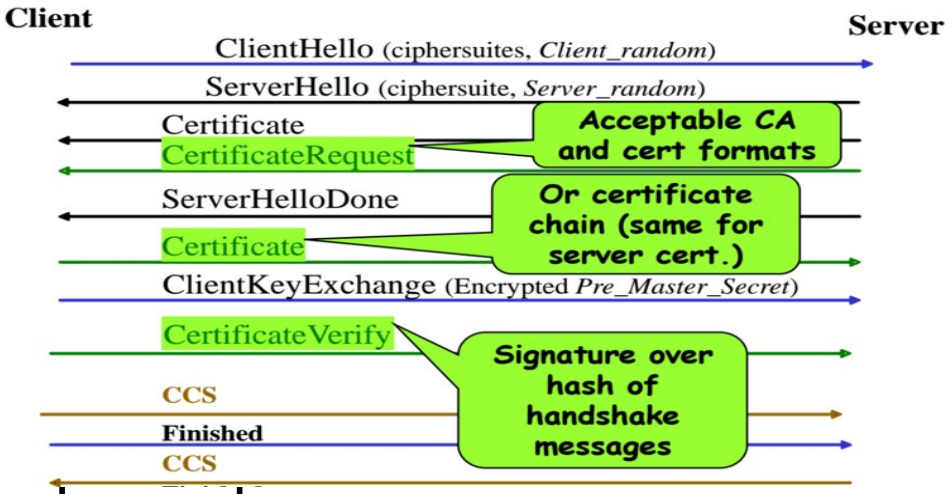
- Challenge: upgrading existing protocol
  - Unrealistic: all upgrade at same day
  - Backward compatibility: new (server, client) can still work with old (client, server)
    - Server selects version based on client's (in 'hello')
    - Downgrade prevented using 'finish' authentication
- Backwards compatibility vulnerabilities:
  - Downgrade-dancing clients
  - Downgrade to SSLv2 (no integrity!)
    - Disallowed – in SSL3, allow with 'trick' / vulnerable
  - Immediate discovery of key forge MAC [LOGJAM]

Advanced Handshake Features

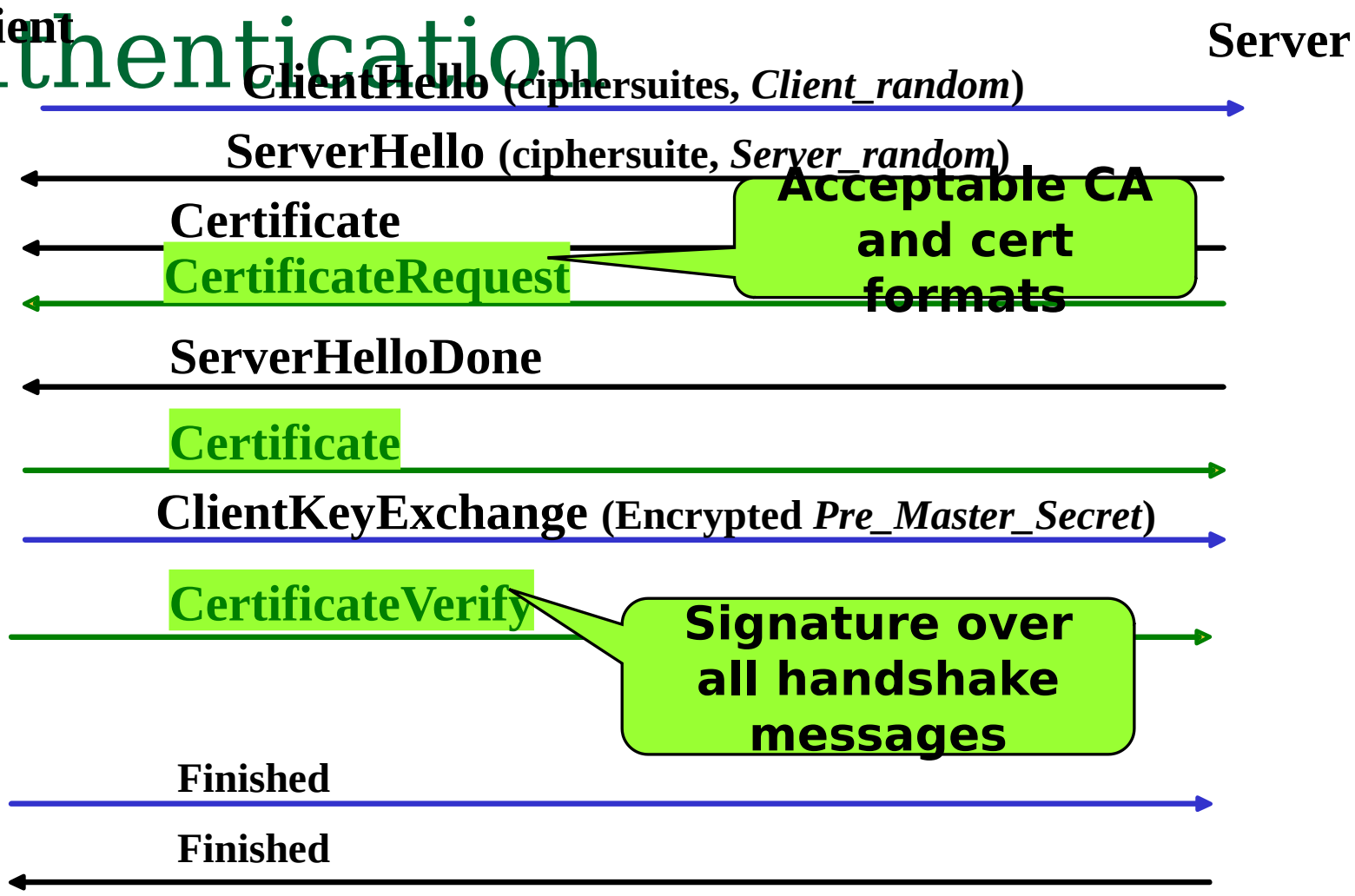
- Client authentication
- Perfect Forward Secrecy (PFS) - ephemeral keys
- Session resumption (ID-based, ticket)
- TLS 1.3 handshakes

TLS/SSL Client Authentication

- Usually, TLS/SSL used only with server PK
  - Only allows client to authenticate server
  - Client authentication: encrypt secret (pw, cookie)
- But TLS/SSL also allows client certificates
- How?
  - Client authenticates by signing with certified PK
- Easy – no PW!
- But: PKI challenges, device dependency
- ⇒ Limited use, mainly within organization/community



TLS/SSL Client Authentication



SSL Client Authentication: Issues

- Which identifier?
  - No global, unique namespace
  - Result: each server use its own client names, certificates
- Support for mobility of cert and key…
  - Smartcard, USB `stick`?

- Rarely used

Advanced Handshake Features

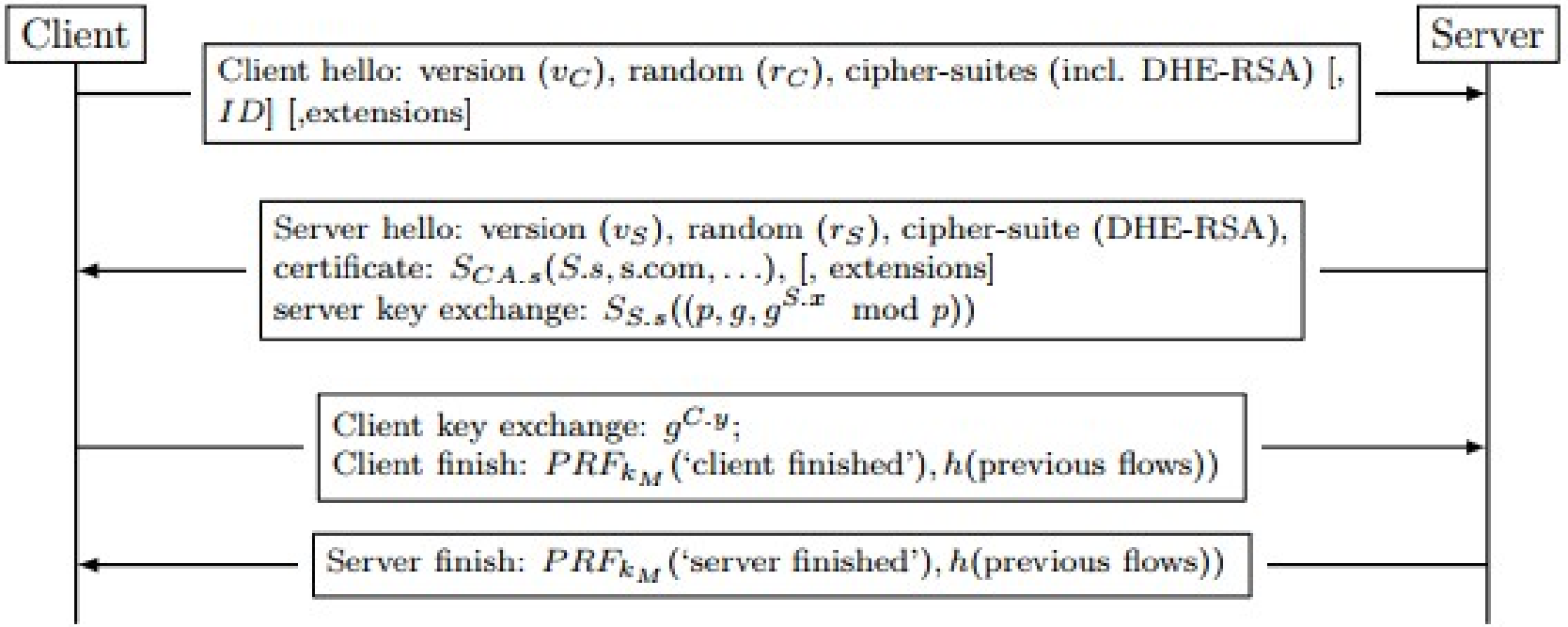
- Client authentication
- **Perfect Forward Secrecy (PFS)**
  - ephemeral keys
- Session resumption (ID-based, ticket)
- TLS 1.3 handshakes

Ephemeral public keys

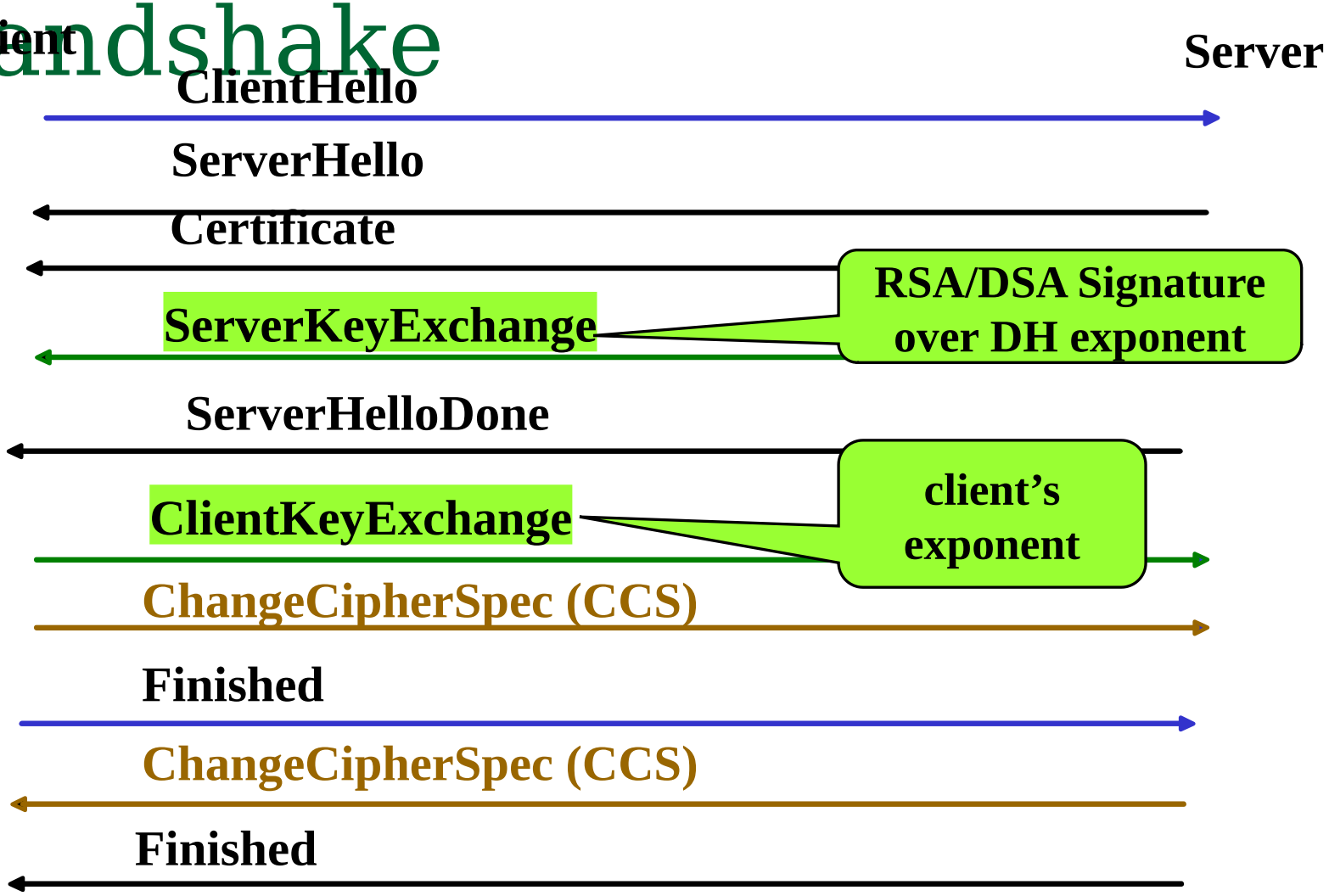
- Ephemeral keys: per-connection
  - Per-connection public keys ? Why?
- Motivations?
  - Perfect forward security: present traffic immune from future exposure – incl. of past keys
  - Historical: ‘export-grade’ (weak) keys [512b RSA]
- How?
  - Diffie-Hellman key exchange
  - Authenticated using long-term keys
- [Mozilla’15]: ~95% of ciphersuites negotiated

TLS/SSL Handshake: Ephemeral DH

- Server signs a DH exponent
  - E.g., using RSA signtures
- Client just sends DH exponent



TLS/SSL Ephemeral PK Handshake

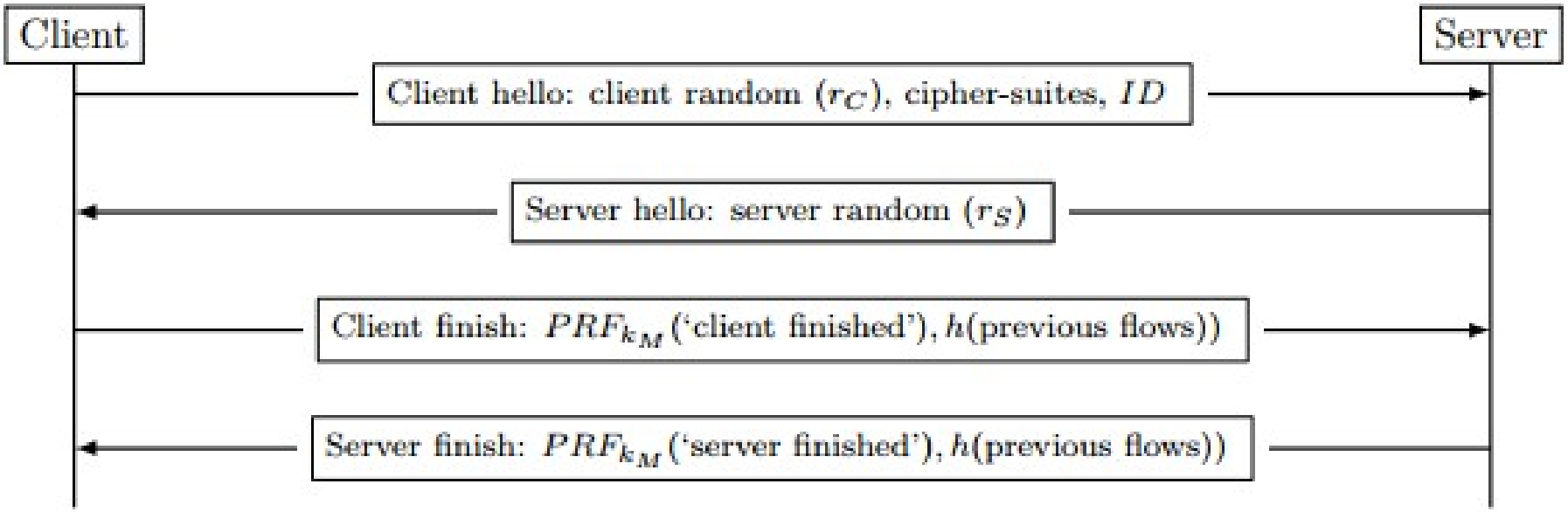


Advanced Handshake Features

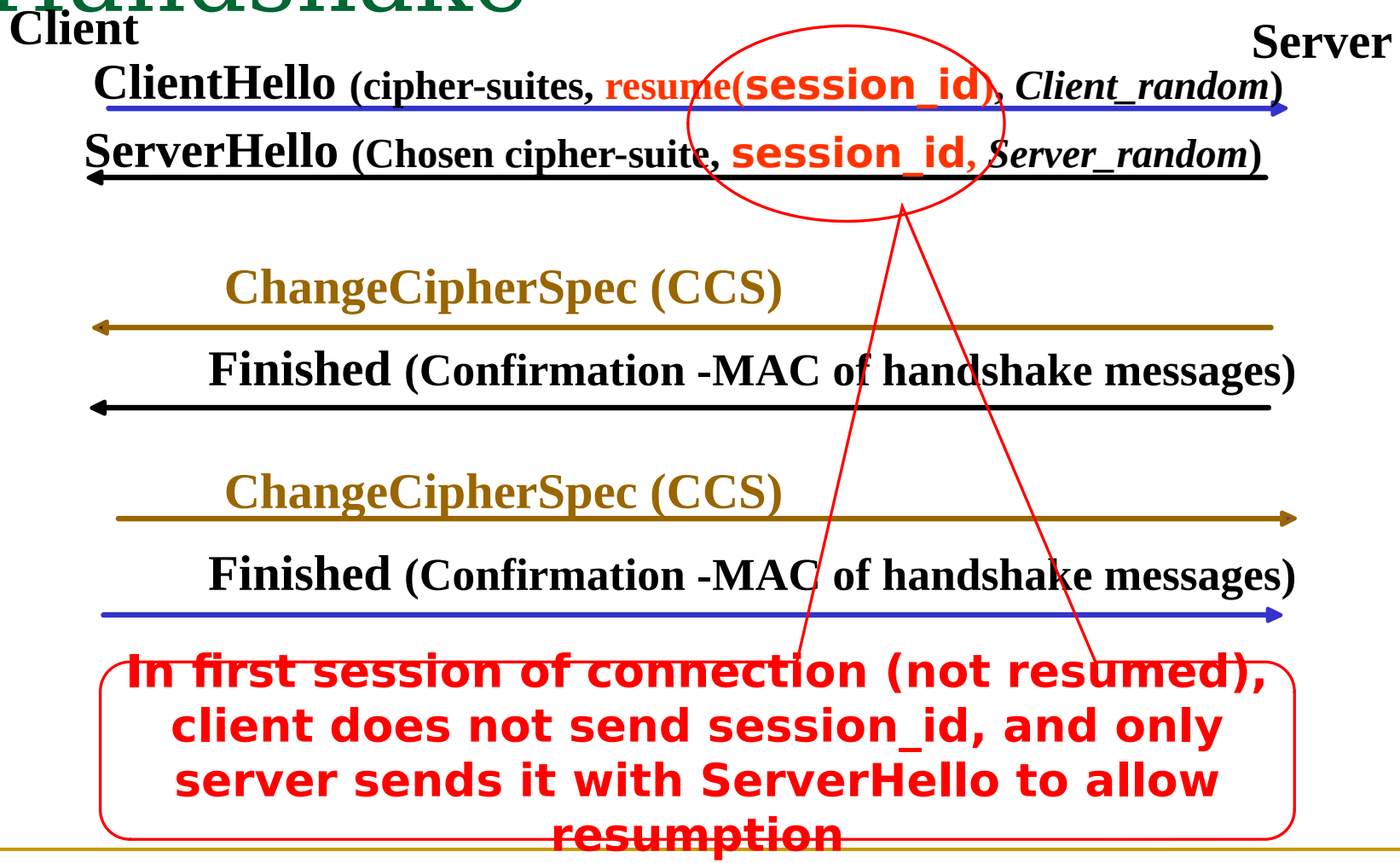
- Client authentication
- Perfect Forward Secrecy (PFS) - ephemeral keys
- Session resumption (ID-based, ticket)
- TLS 1.3 handshakes

ID-based Session Resumption (SSLv3-TLS1.2)

- Idea: server, client store (ID, key) per peer
- Reuse in new connections btw same pair
- Saves PK operations (CPU, BW)



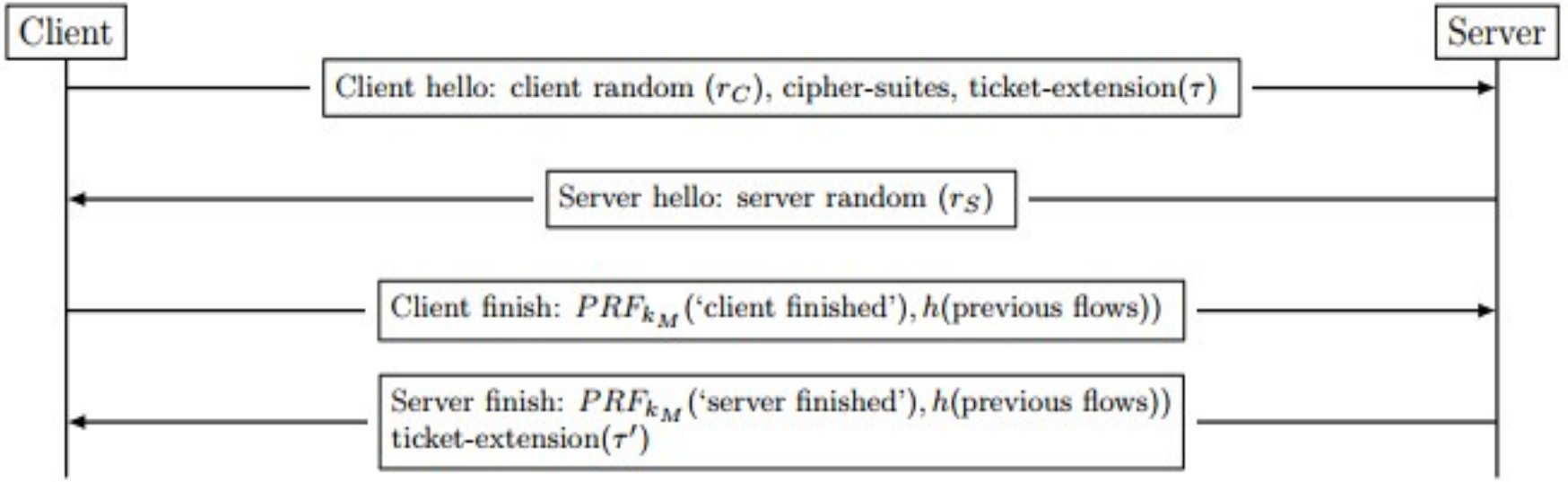
Session-ID Resumption Handshake



Session Resumption Issues

- Need to keep state, lookup ID…
  - Overhead (small cache: less effective)
  - Need to share among (many!) replicates of server
  - For PFS: ensure keys disappear after ‘period’
- Solution: Client-side caching (Session-Ticket Hello Extension)
  - Ticket contains master key, encrypted by a secret session ticket key, known (only) to server
    - Share with other servers of this site
    - Change periodically to enforce PFS
  - Uses TLS extension (not in SSL)

Session-Ticket Resumption



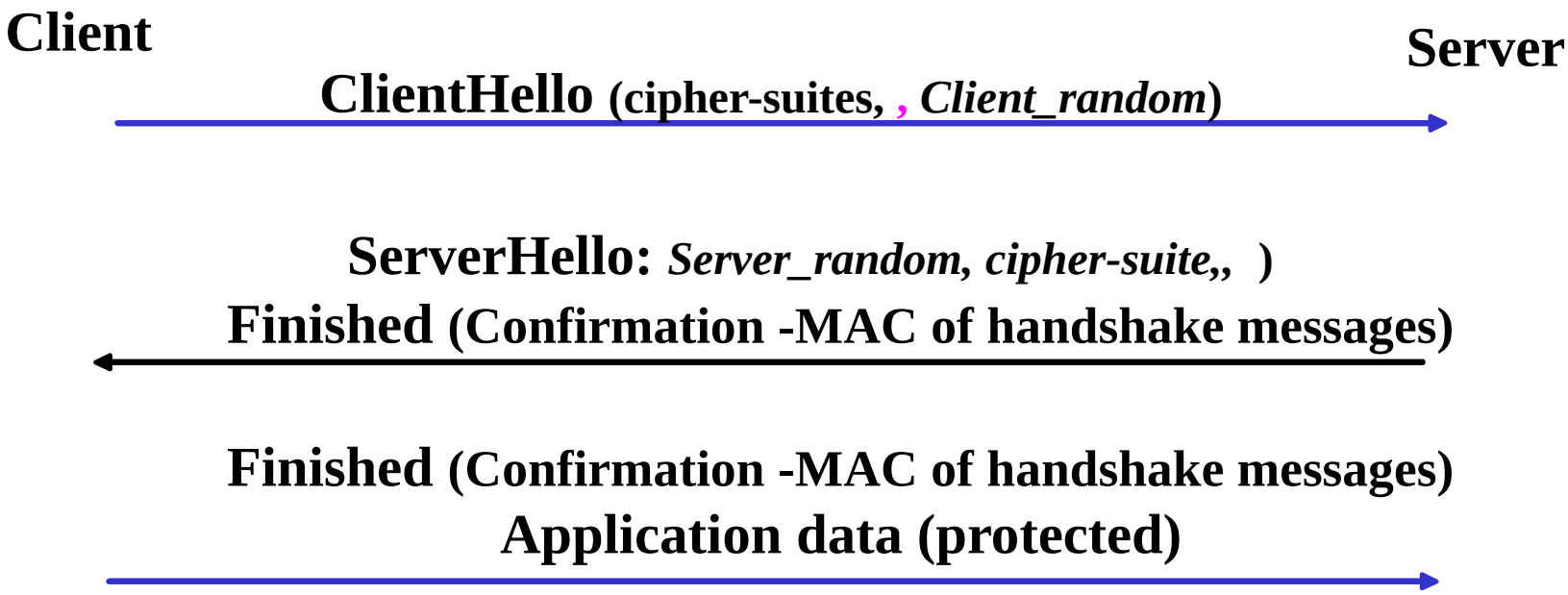
- To preserve PFS:
  - Tickets ‘expire’ after ‘time period’ (e.g., 24 hours)
  - Ticket-key changed rapidly (e.g., every hour or few)
  - Ticket-key erased after `time period’ ends (e.g., daily)
- Problem: many servers do not limit ticket-key lifetime

Advanced Handshake Features

- Client authentication
- Perfect Forward Secrecy (PFS) - ephemeral keys
- Session resumption (ID-based, ticket)
- **TLS 1.3 handshakes**

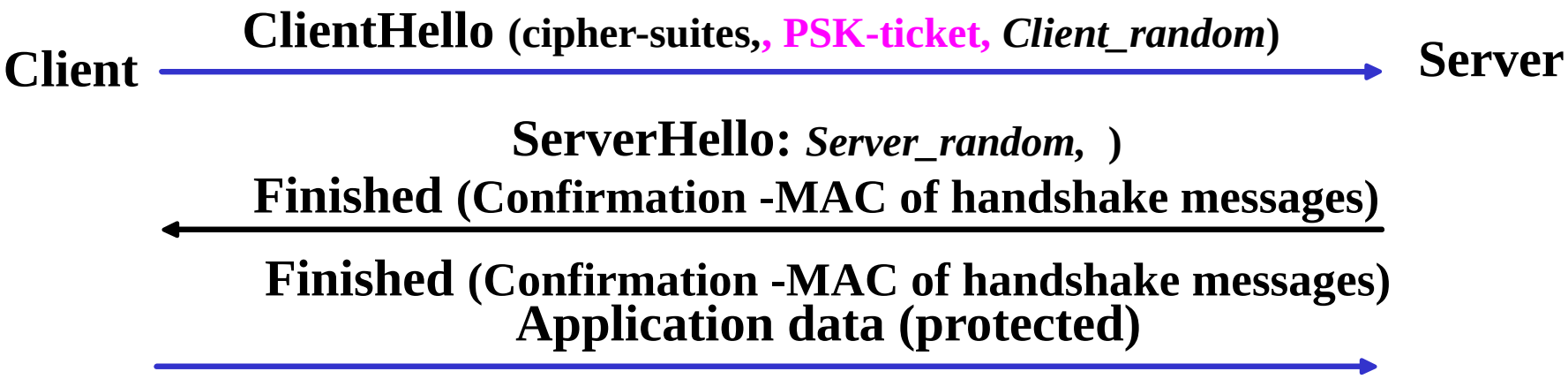
TLS 1.3 'Full handshake': 1-RTT

- No RSA: only DH + signature by server
- 1-RTT: client sends key-share in Hello !
  - So: a key-share per each cipher-suite option sent by client



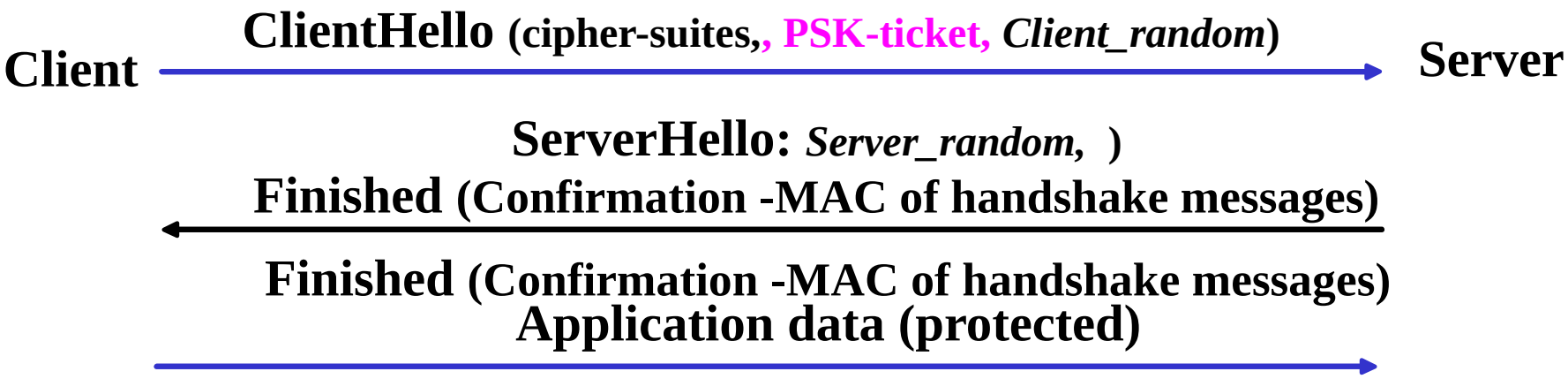
TLS 1.3 Session Resumption: PSK

- Resume only using Pre-Shared Key (PSK)
  - Essentially, build-in ticket mechanism
  - Optional use of DH for PFS (ephemeral key)
    - Add to client-hello
    - Use PSK to authenticate key-shares and derive key



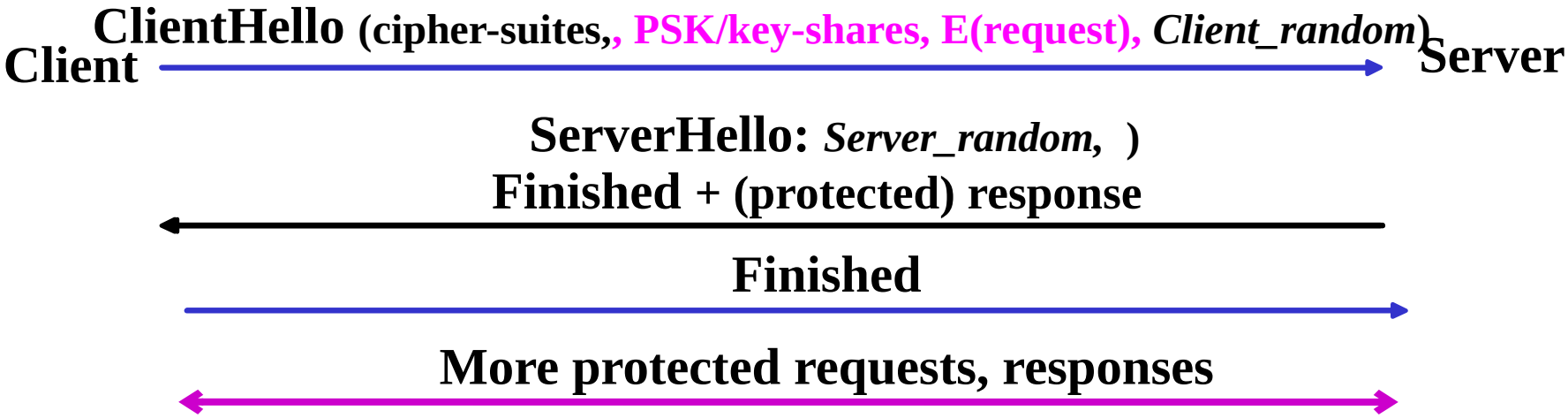
TLS 1.3 Session Resumption: PSK

- Resume only using Pre-Shared Key (PSK)
  - Essentially, build-in ticket mechanism
  - Optional use of DH for PFS (ephemeral key)
    - Add to client-hello
    - Use PSK to authenticate key-shares and derive key



TLS 1.3 Session Resumption: 0-RTT

- TLS 1.3 even supports 0-RTT!



TLS/SSL: Conclusion

- TLS/SSL: a mature, widely used crypto protocol
- Many features, vulnerabilities, fixes, versions
- Many downgrade attacks
  - More foresight, scrutiny would have saved a lot!
- Extensibility by design principle: build into design mechanisms for secure extensions, downward-compatible versions, and negotiation
- Improved key-separation: use independent keys for each different crypto scheme or version, and different types/sources of plaintext.

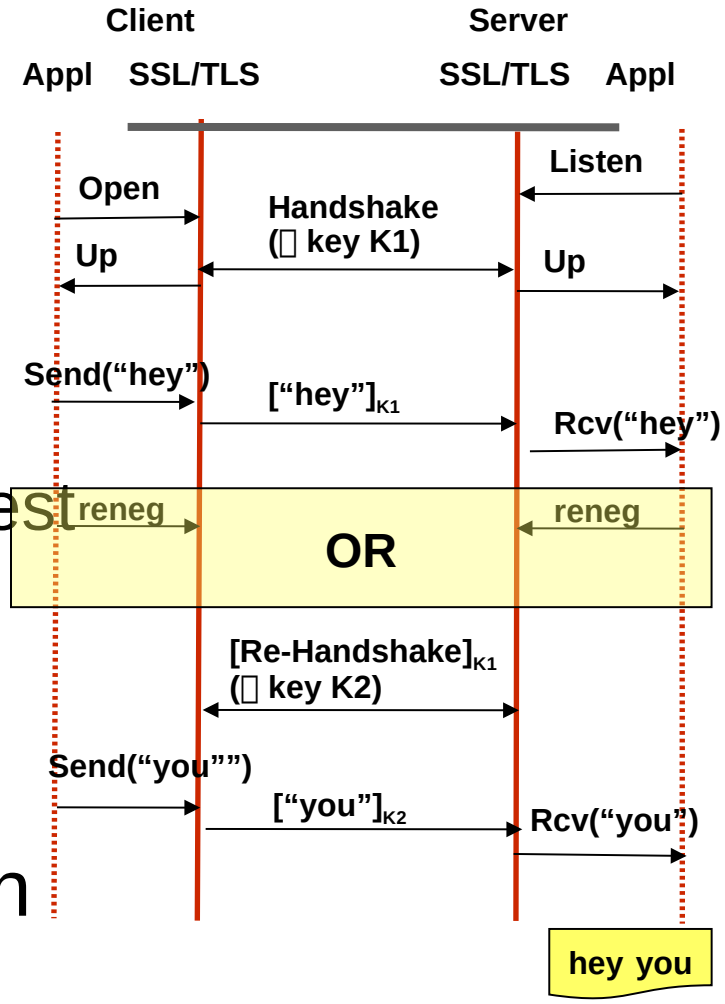
Extras

Renegotiation Handshake

- Client, server can initiate re-negotiate
- Why?
  - Refresh keys (by time or use, e.g. if counter overflows)
  - Change cipherspec (e.g. to more secure)
  - Request client authentication (certificate)
- How?
  - [Server: Hello_Request], Client: send Client_Hello
  - Protected by existing keys
- Vulnerabilities ‘patch’ removed in TLS 1.3
  - Details follow … or skip

Renegotiation: Application View

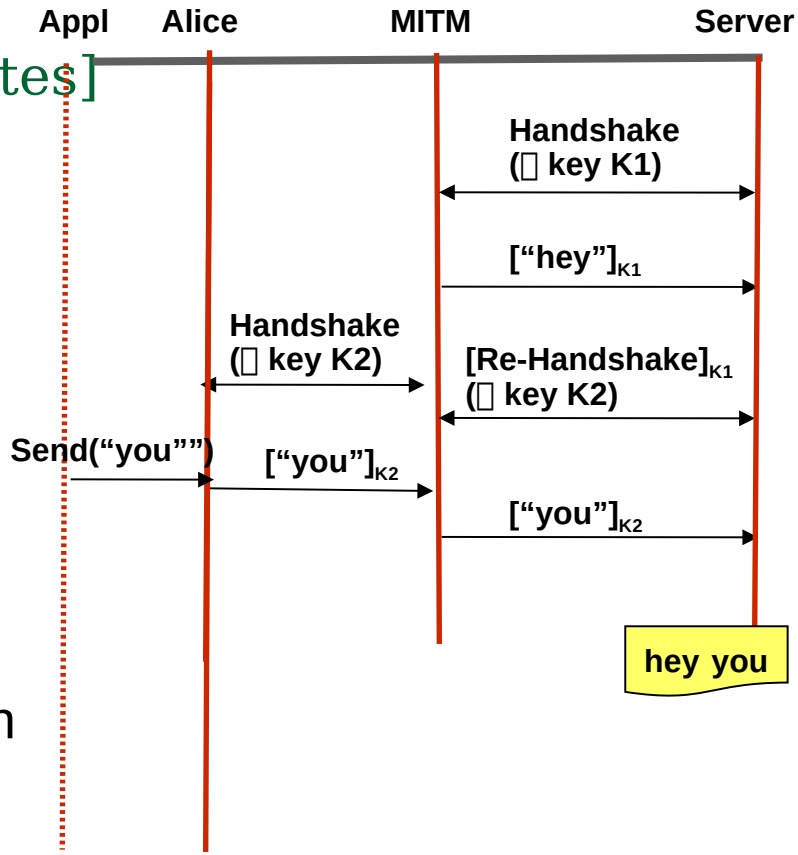
- Record layer carries application+handshake
- Renegotiation initiated by client or by server
  - To change keys / cipher, request client cert, …
  - Each may refuse
- Most servers don't separate appl data after re-negotiation
- Two attacks… (follow)



Renegotiation MITM Prefix Inject Attacks

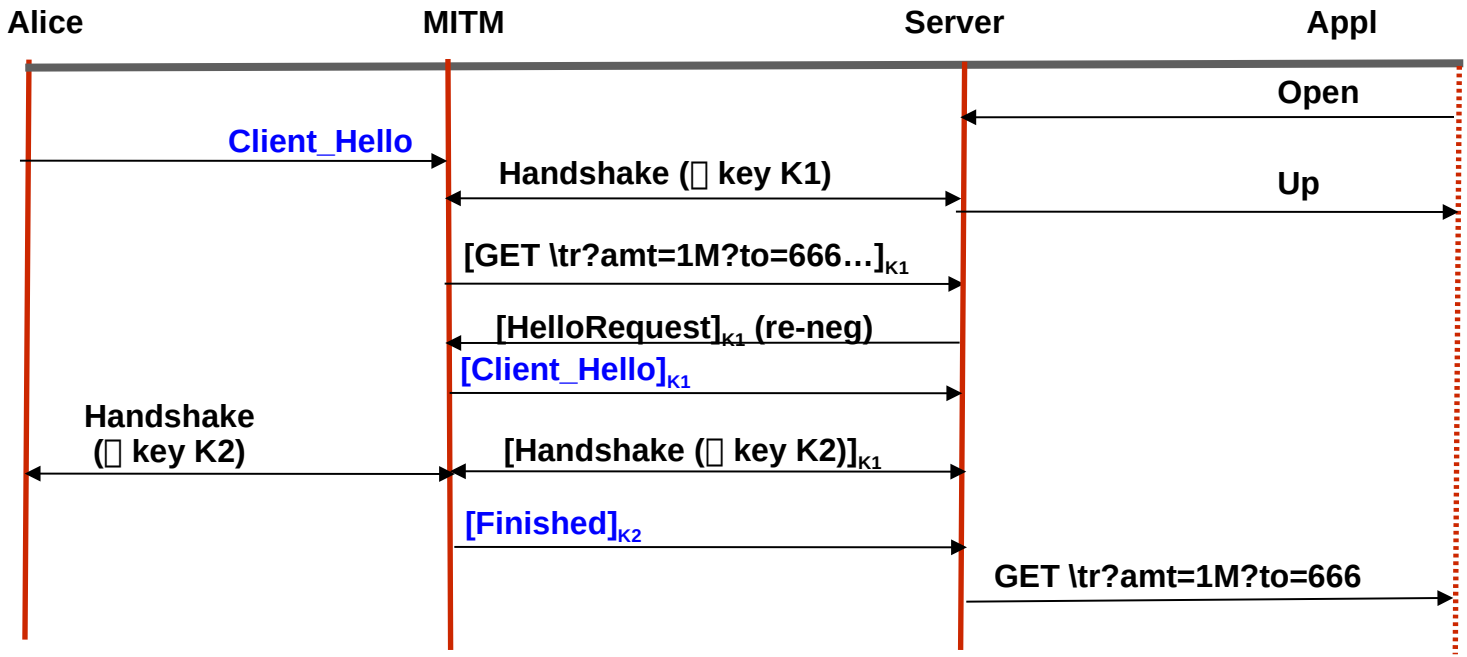
- Assume: MITM attacker, and:
  - Server agrees/inits re-negot.
  - No `marking` after re-negot.
- Result: inject <u>arbitrary prefix</u>
- So what?
  - Break cookie/http client auth
  - Expose cookie / other secret
  - Break SSL/TLS client auth
    - Since (most) servers do client-auth *after* request…
- Defense: RFC5746, TLS Renegotiation Indication Extension

[Ray, Dispensa, Rescorla 09 – see notes]



RenegotiationAttack on SSL Client Auth

- Many servers renegotiate for client cert, only for certain pages/requests
- After authentication – they use request received before
  - Since there is no way in HTTP to ask client to resend
- This is vulnerable… easy MITM attack!



Key-Establishment, PKI & SSL/TLS

- Overview of SSL/TLS (and use of certificates)
- Basic Public Key Infrastructure (PKI)
- TLS/SSL Key-Establishment
- Delegated to Network Security (4402):
  - `Advanced' PKI and TLS
    - Including record layer and cryptography, attacks

Key-Establishment, PKI & SSL/TLS

- Overview
- TLS/SSL Key-Establishment (handshake)
- **TLS/SSL: record layer and cryptography**
  - Key derivation
- PKI (Public Key Infrastructure)

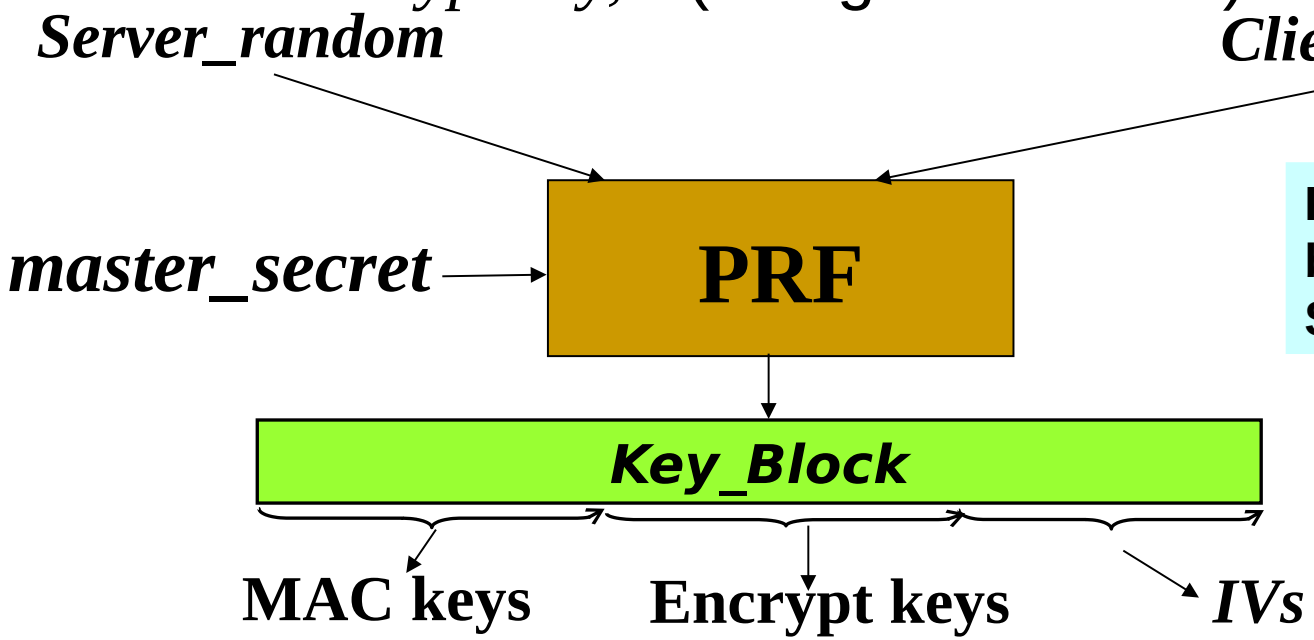
Extract-then-Expand Key Derivation

- Goals:
  - Multiple, independent, secret random keys
    - Exposure of some keys will not expose others
  - From one imperfectly-random shared secret
    - Called pre-master-secret ()
    - Typically, DH-exchanged
- Extract-then-Expand Key Derivation:
  - Extract random master key:
    - Where is public random bits, e.g., client-random
  - Expand:
    - : context/goal of key, e.g., “encrypt to client”
  - From TLS1.3. (Earlier versions a bit different.)

Deriving Connection Keys, IVs

Key_Block = PRF $_{master\_secret}$ *(“key expansion”|| Server_random || Client_random)*

Split *Key_Block* to *ClientMACKey, serverMACKey, ClientEncryptKey,…*(using fixed order)

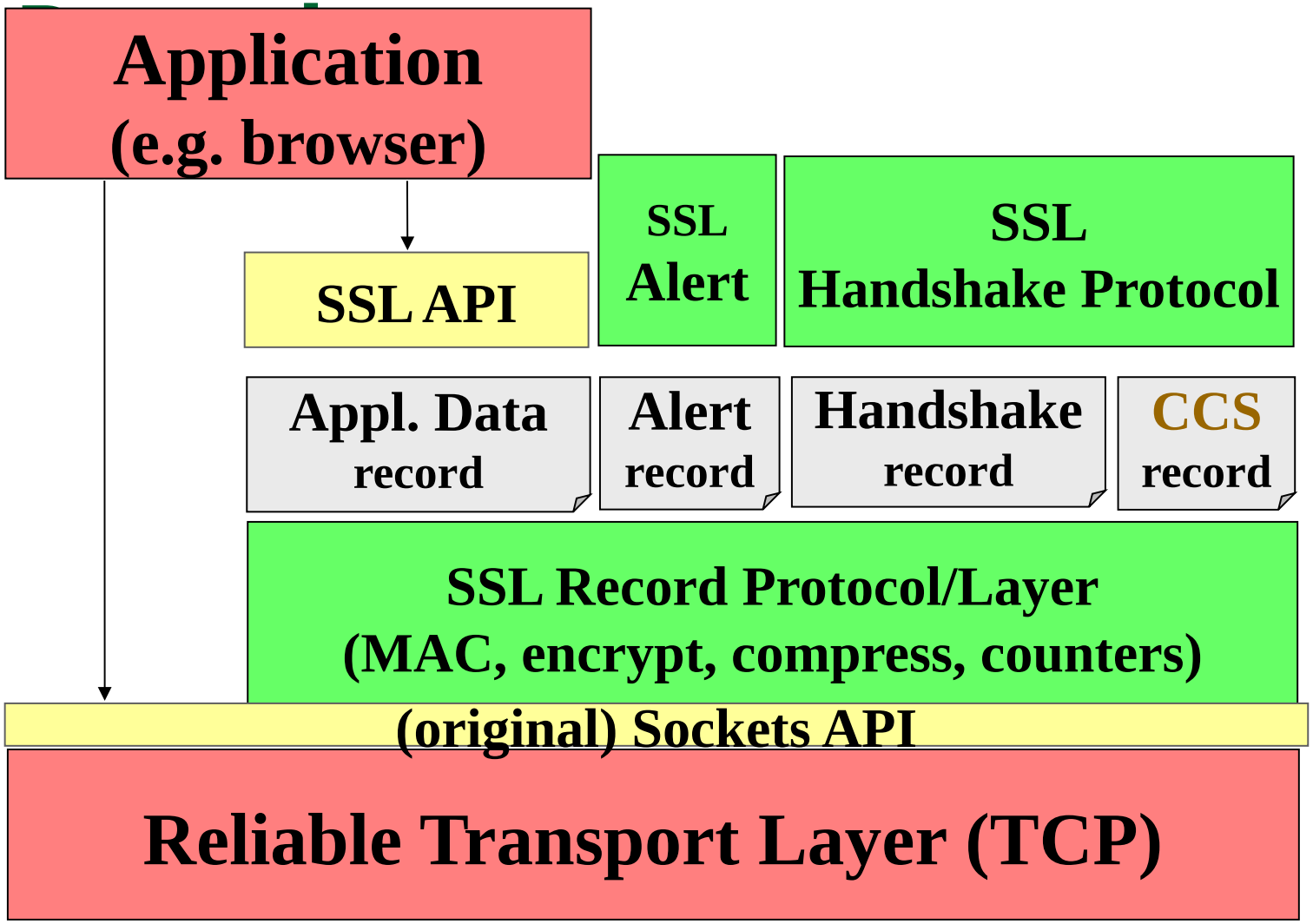


PRF details differ btw TLS and SSL3… So: presented later.

Handshake Protocol

Message	M?	From	Meaning/Contains
HelloReq.	O	Srvr	Inform client to begin
ClientHello	M	Clnt	Version, *client_random, session_ID,* algorithms
ServerHello	M	Srvr	Version, *server_random, session_ID,* algorithms
Certificate	O	Both	X.509 certificate
ServerKeyExchng	O	Srvr	Ephemeral server pub key (this session only)
Cert. Request	O	Srvr	Cert. type (RSA/DSS,Sign/DH), CAs
ClientKeyExchang	M	Clnt	Encrypted *pre_master_key*
Cert. verify	O	Clnt	Sign previous messages
Finished	M	Both	MAC on entire handshake

SSL Protocols, Layers and



TLS 1.3 Record Layer

- Single simple secure construction
  - Avoids many vulnerabilities, including:
    - Padding
    - Side-channels: Lucky13, POODLE
- Use AEAD: (shared-key) Authenticated Encryption with Associated Data
  - Associated-data is only authenticated (public)
  - Few algorithms, e.g. AES-GCM

SSL/TLS Overhead ?

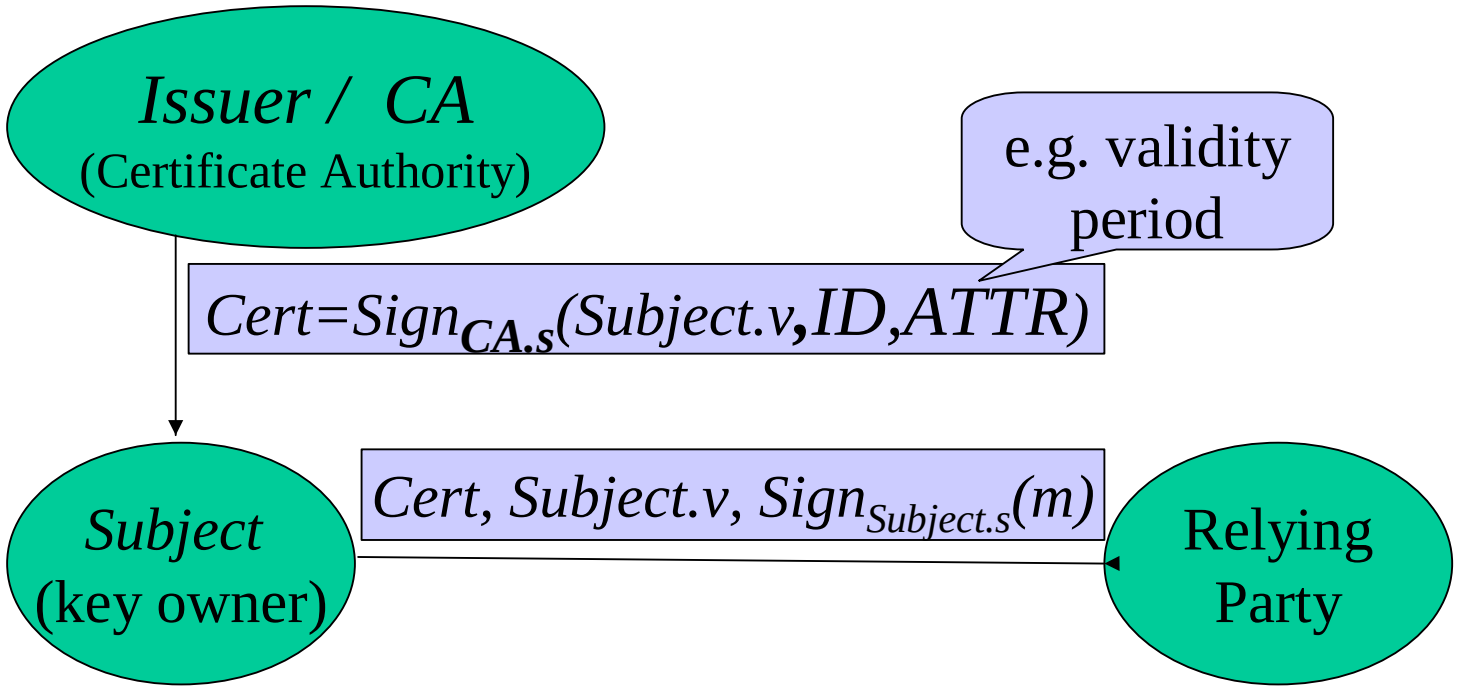
- Handshake
  - Exchanges (Round Trips)
  - Public key operations
- Record (data transfer)
  - Encryption etc.
  - Inability to cache (proxy)
- Caching non-confidential info with SSL?
  - Use SSL to transfer just HTML with script
  - Script downloads, authenticates rest of page

Key-Establishment, PKI & SSL/TLS

- TLS/SSL Key-Establishment (handshake)
- **PKI**
  - **Basics: X.509 and PKIX**
  - Defenses against Corrupt CA & Equivocation
  - Certificate revocation

Public Key Certificates & Authorities

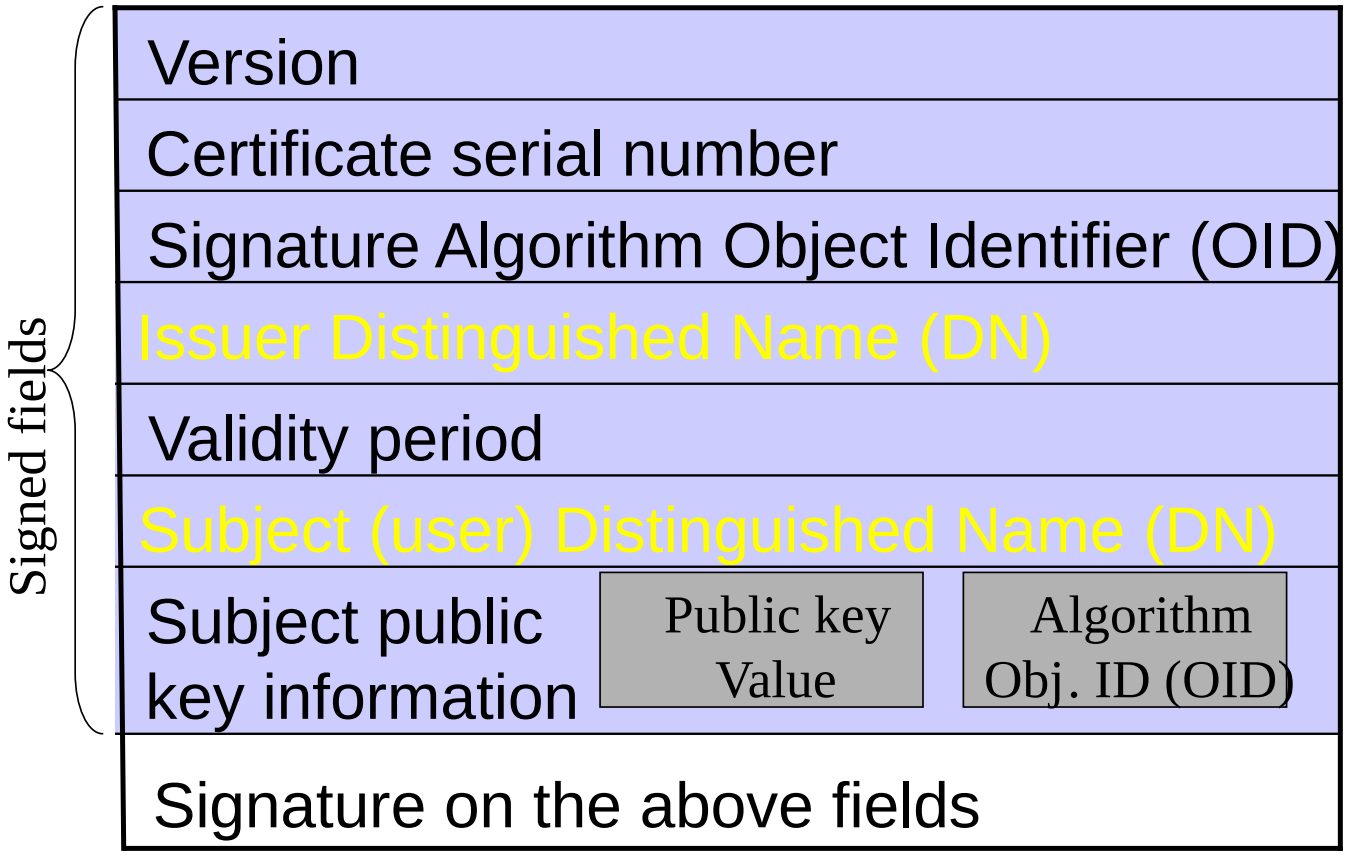
- *Certificate:* signature by Certificate Authority (CA) over subject's public key and attributes
- Attributes:
  - Validated by CA (liability?)
  - Used by **relying party** for decisions (e.g., use this website?)
  - **Questions:** Attributes? Identifiers? Format? …



X.509 public key certificates

- Public key **signed** by (trusted) **issuer (CA)**
  - Certificate: signed public key (and attributes)
  - CA: Certificate Authority (issuer)
- **X.509:** ITU's standard for certificates & usage
  - Widely adopted – in spite of complexity
- Main outcome of X.500 standard
  - ITU: International Telcos Union
  - Goal: trusted, centralized 'phone directory'
  - Global directory? No; but – X.509 widely used
    - Why global directory failed? Too complex, revealing
  - Identifiers: distinguished names
    - Goals: unique, meaningful, decentralized identifiers

Original (V1) X.509 Certs



Object Identifiers (OID):

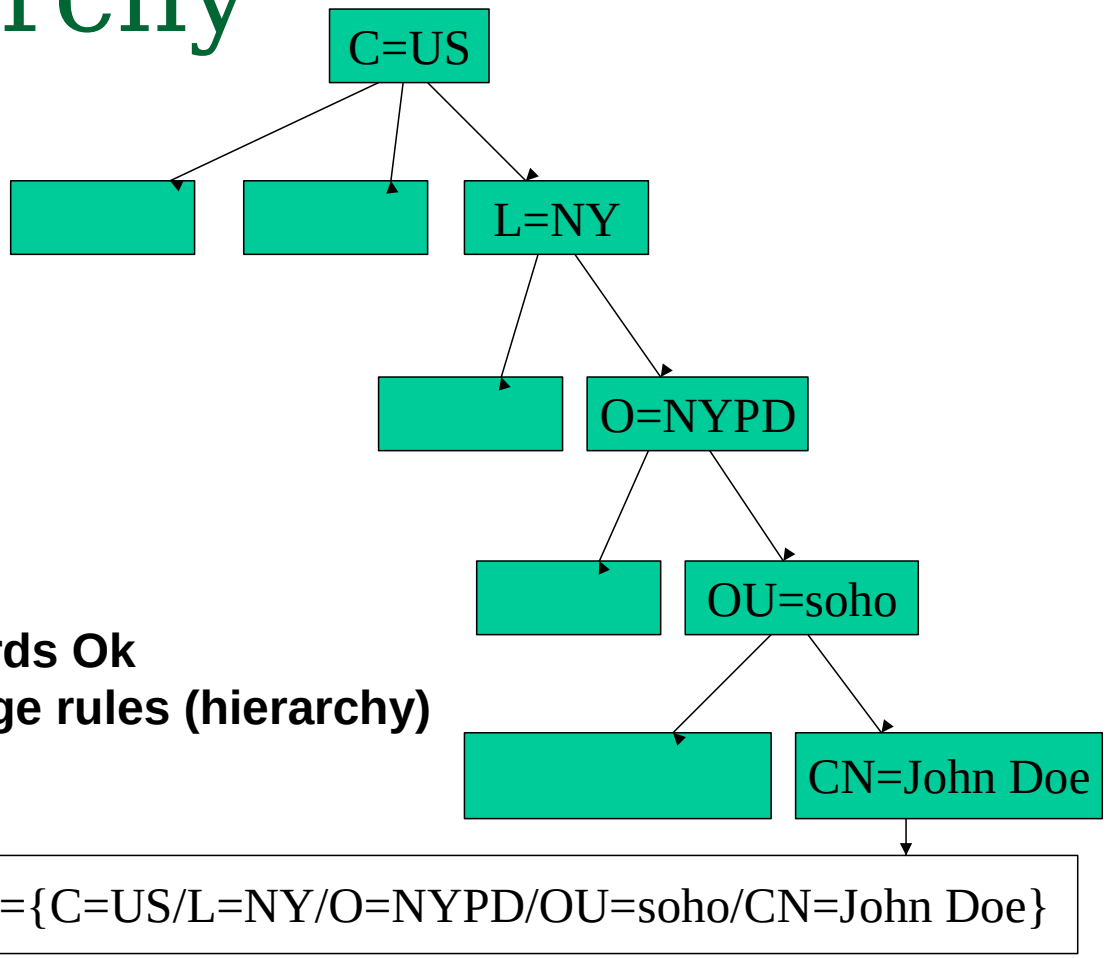
- Global, unique identifiers
- Sequence of numbers, e.g.: 1.16.840.1.45.33
  - Hierarchical

X.509 Distinguished Names (DN)

- Goal: meaningful, unique and decentralized identifiers
- Sequence of keywords, a string value for each of them
- Distributed directory, responsibility *hierarchical DN*

Keyword	Meaning
C	Country
L	Locality name
O	Organization name
OU	Organization Unit name
CN	Common Name

Distinguished Name (DN) Hierarchy

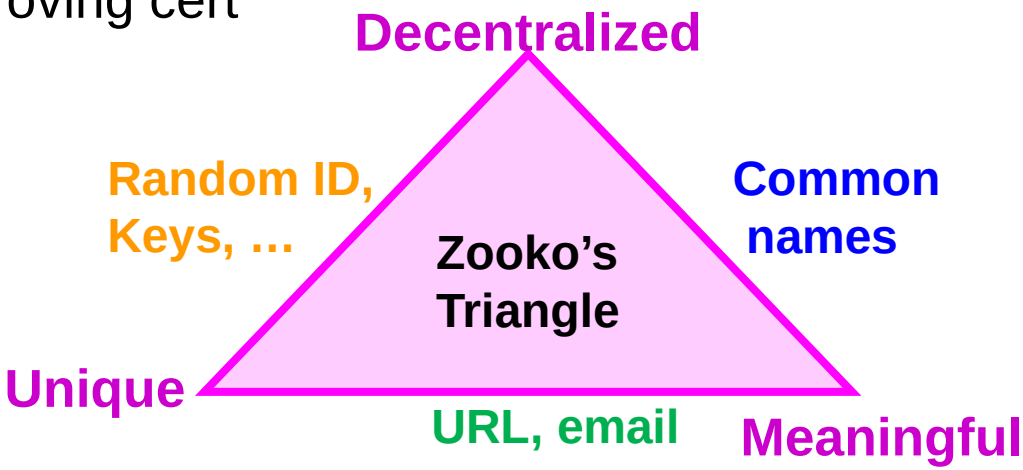


Comments:

1. **Other keywords Ok**
2. **No strict usage rules (hierarchy)**

Goals for Identifiers in Certificates

- Meaningful (to humans)
  - Memorable, reputation, off-net, legal
- Unique identification of entity (owner)
- Decentralized - with Accountability: assigned by any trusted certificate authority
  - Accountability: CA approving cert
- Pairs are easy:
  - Unique + Meaningful
  - Meaningful + Decentalized
  - Unique + Decentralized
- Zooko: can't have all three properties



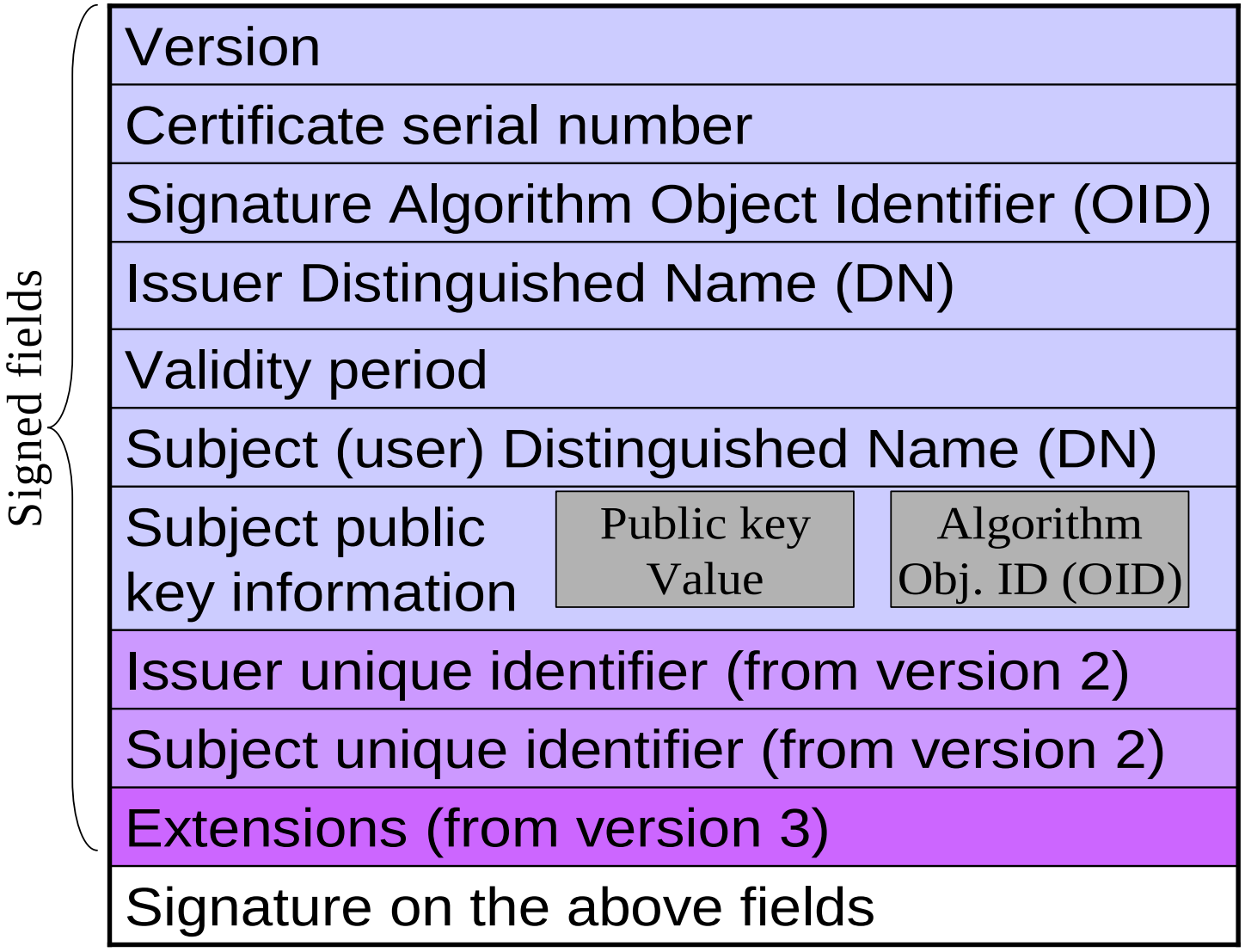
Distinguished Names - Evaluation

- **Decentralized?**
  - Sure: any CA can select DN for its customers, sign cert
- **Unique ?**
  - Could be, if each name space has one issuer
  - TLS reality: browsers trust 100s of CAs for **all** DN
- **Meaningful?**
  - Usually: Julian Jones/UK/IBM
  - But not always: Julian Jones2/UK/IBM
    - Added ‘counter’ to distinguish mistakes, loss of meaning
- X.509 response: v2: unique ID, v3: extensions

X.509 Certs & Subject Identifiers

- V1: Distinguished Name (for subject & issuer)
- V2: unique identifiers (for subject & issuer)
- V3: extensions
  - PKIX standard: SubjectAltName extension
    - Including DNSname
    - PKIX: Public Key Infrastructure working group of IETF
    - Widely adopted, including in SSL/TLS (& https)

X.509 Public Key

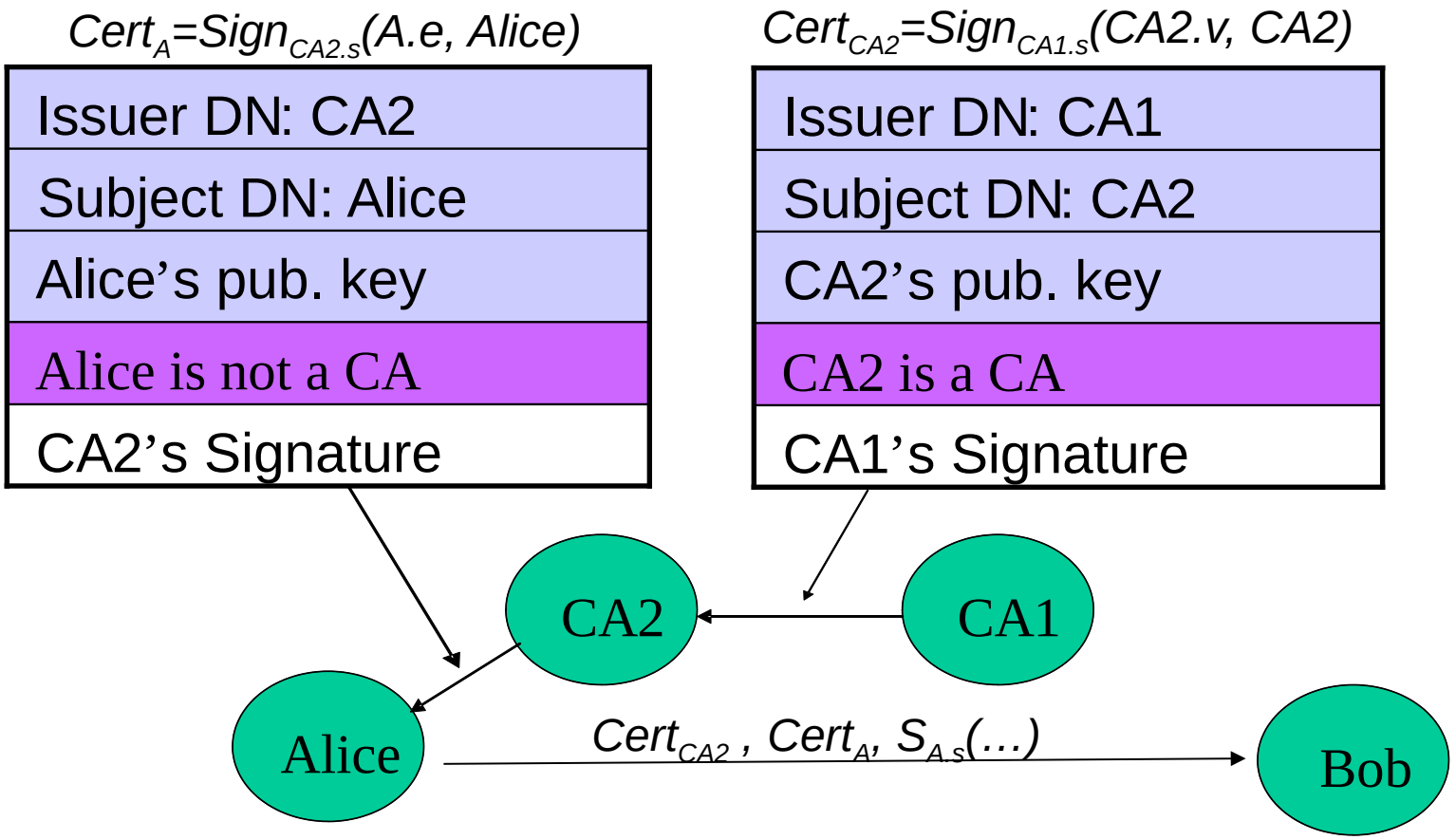


X.509 V3 Extensions Mechanism

- Each extension contains…
- Extension identifier
  - As an OID (Object Identifier)
  - E.g. `Naming constraints`
- Extension value
  - E.g. `Include C=IL`, `exclude dNSName=*.IBM.COM`
- Criticality indicator
  - If critical, relying parties MUST understand extension to use certificate
    - E.g. Naming constraints is `critical`
  - If non-critical, Ok to use certificate and ignore extension

Certificate Path

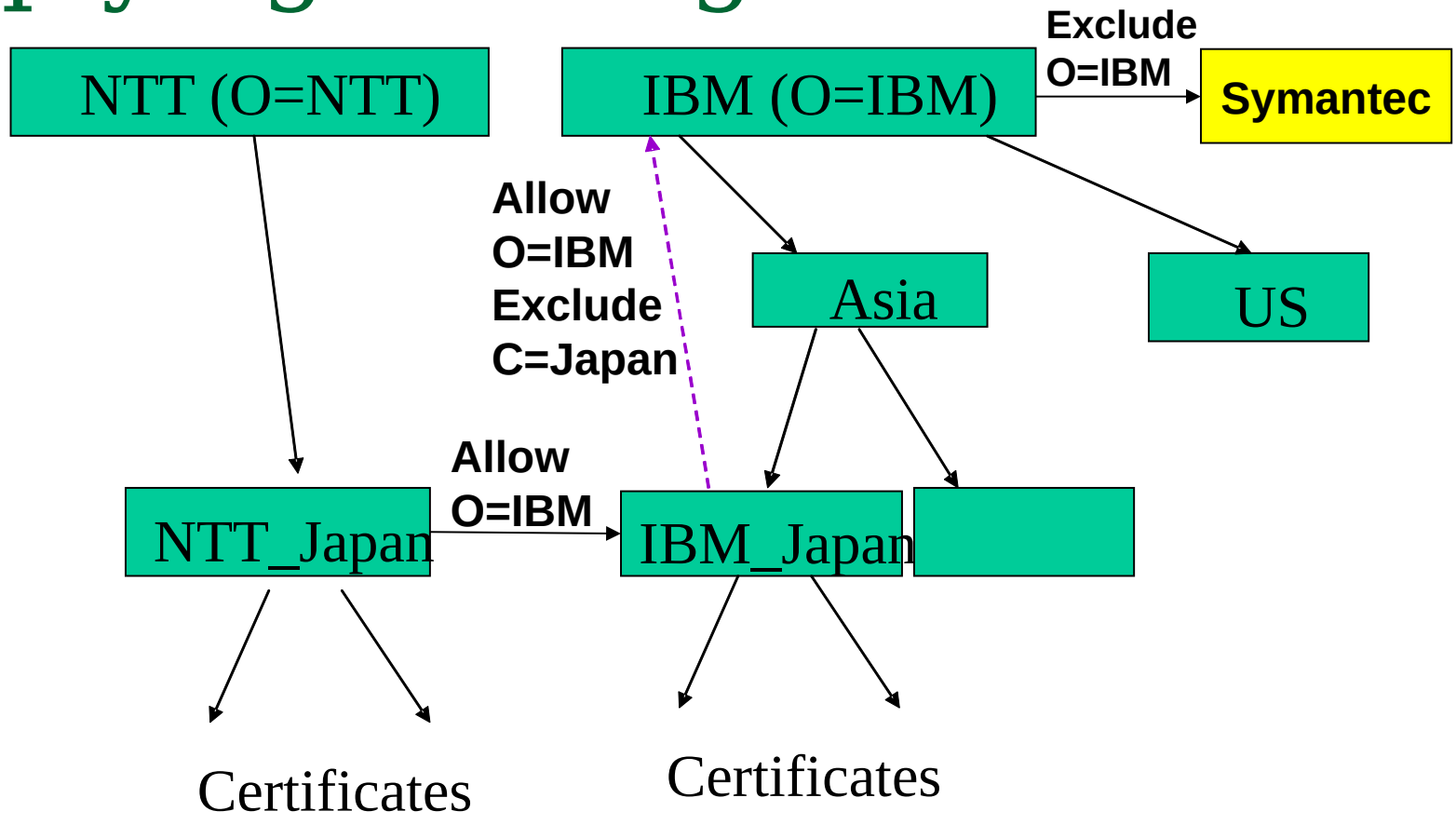
- Suppose relying party (browser) does not trust subject's CA…
- Solution: Certificate Path – a trusted CA certifies subject's CA



X.509v3/PKIX Standard Extensions

- Most important: Naming and Constraints extensions
- Certification path constraints extensions:
  - Basic constraints:
    - Goal: mark the (normal) case: subject isn't CA
    - `CA:` Subject is CA or end entity
    - `CertPathLength`
  - Naming_constraints
    - Constraints on `DN` - in certs issued by subject
      - Only relevant when subject is a CA !
    - 'Allow' and 'Exclude'

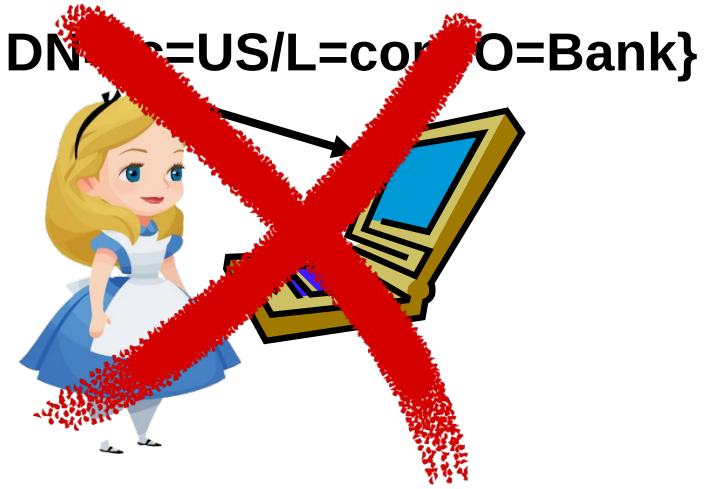
Applying naming constraints



- **NTT JP allows IBM JP to certify IBMers**
- **IBM JP allows IBM to certify all IBMers, except of IBM JP**
- **IBM trusts Symantec's certificates, except for O=IBM**

Reality: DNs aren't usable identifiers

- Relying parties (users) don't know the DN



- Hopefully, they know the domain (in URL)
- Naming extensions: alternative names
  - For TLS: `cert.SubjectAltName.dNSname`
  - Possible values: bank.com, *.bank.com (wildcard), …
  - May use also in naming constraints

SSL / TLS PKI Challenges

- Many CAs `trusted' in browsers
- Every CA can certify any domain (name)
  - Since naming constraints NOT used
  - Two CAs can same name (equivocation)
  - To detect bad-CA: must find bad-certificate
    - No public, auditable log of certificates
- Several well-known failures
  - DigiNotar, Comodo, Stuxnet, …

Key-Establishment, PKI & SSL/TLS

- Overview of SSL/TLS
- Basic Public Key Infrastructure (PKI)
- **TLS/SSL Key-Establishment**
- Delegated to Network Security (4402):
  - `Advanced' PKI and TLS
    - Including record layer and cryptography, attacks

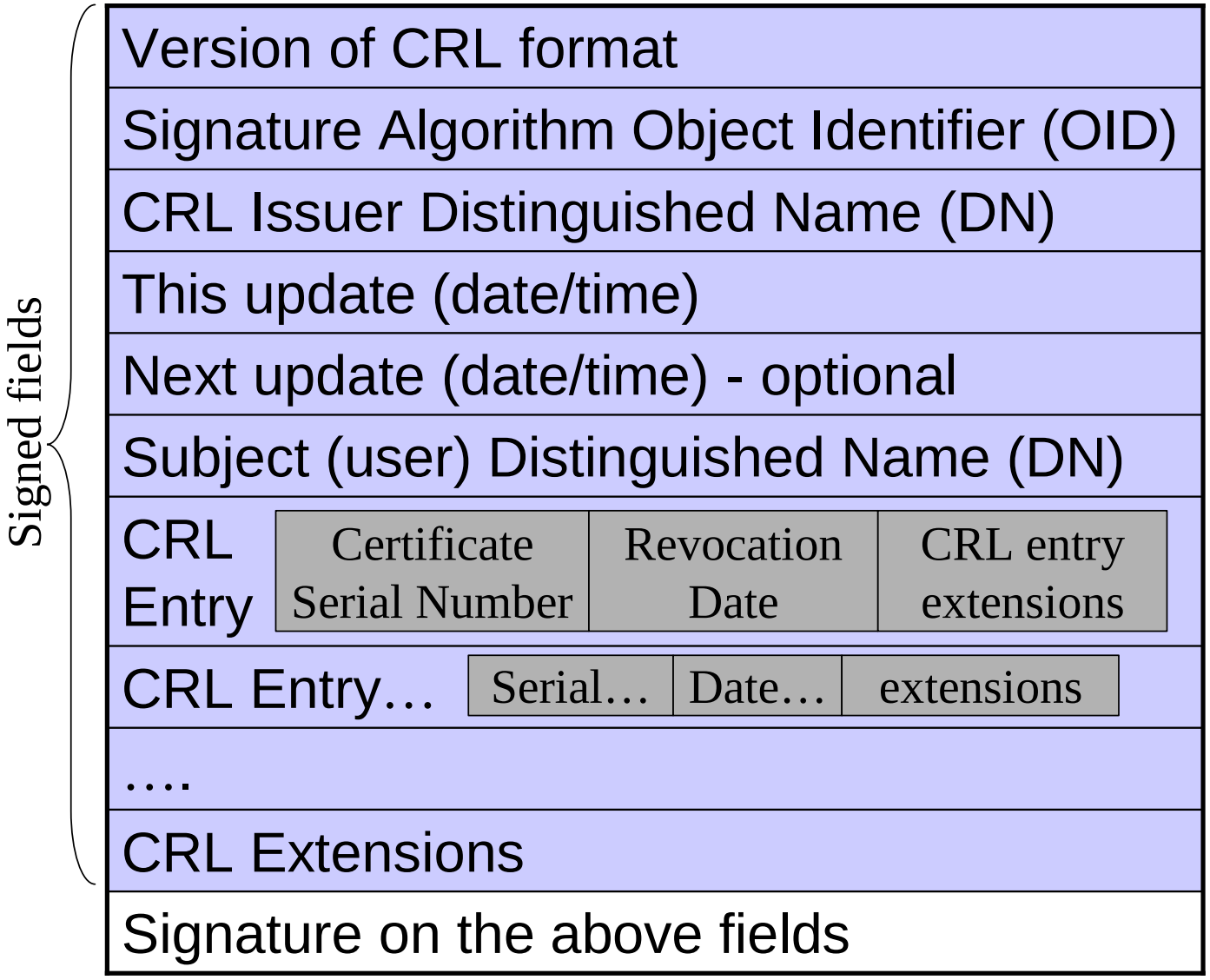
Key-Establishment, PKI & SSL/TLS

- Overview
- PKI basics: X.509 and PKIX
- TLS/SSL Key-Establishment (handshake)
- TLS/SSL: record layer and cryptography
- **PKI – in depth**
  - Certificate revocation
  - Defenses against Corrupt CA
    - Mainly: Certificate Transparency

Certificate Revocation

- Reasons for revoking certificate
  - Key compromise
  - CA compromise
  - Affiliation changed (changing DN or other attribute)
  - Superseded (replaced)
  - Cessation – not longer needed
- How to inform relying parties?
  - Do not inform – wait for end of (short?) validity period
  - Distribute *Certificate Revocation List (CRL)*
  - Ask - Online Certificate Status Protocol (OCSP)
  - Skip details

X.509 CRL Format



Revocation is Difficult

- If CRLs contain all revoked certificates (which did not expire)… it may be huge!
- CRLs are (also) not immediate
  - Who is responsible until CRL is distributed?
  - What is the impact on non-repudiation?
- Solutions:
  - Online Certificate Status Protocol (OCSP)
  - More efficient CRL schemes (usually CRL extensions)
    - CRL distribution point – split certificates to several CRLs
    - Authorities Revocation List (ARL): list only revoked CAs
    - Delta CRL – only new revocations since last `base CRL`
    - Certificate Revocation Tree (more later)
  - Short validity for certificates

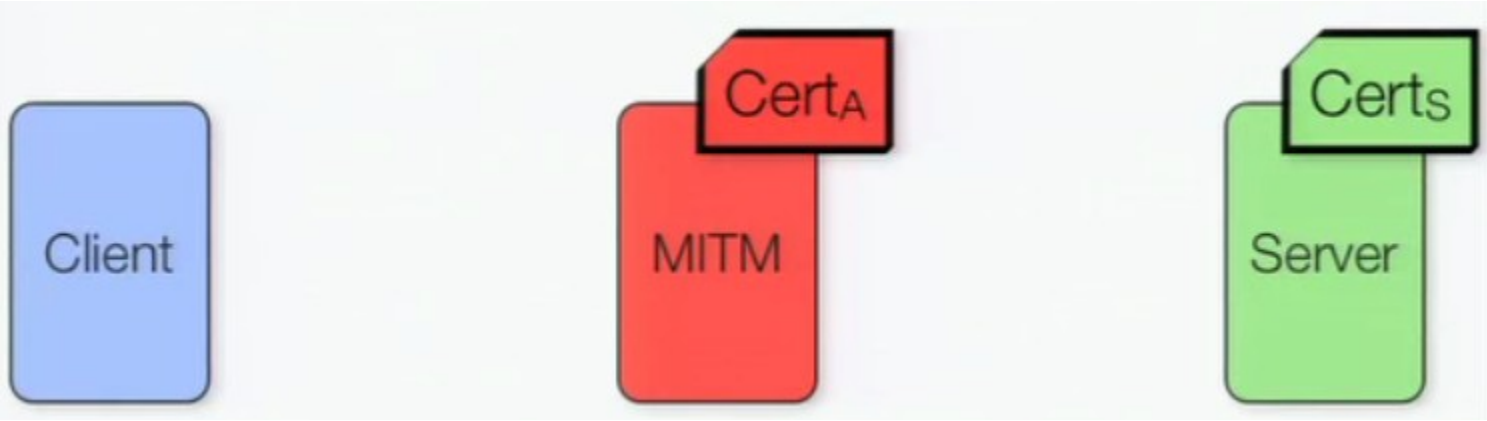
Short-Term Certificates

- Idea: short validity period of certificates, so no need to revoke them
- Concern: overhead of signing many certificates each (short) period
- Solutions:
  - Extend many certs with one signature: *hash tree*
    - $Sign_{CA.s}(date, valid{:}h(h(cert_A),h(cert_B),\ldots))$
    - Certificate revocation tree:
      $Sign_{CA.s}(date, all\ except{:}h(h(cert_A),h(cert_B),\ldots))$
  - Certificates includes a *hash chain*, e.g. for Jan 2005:
    $Cert_A = Sign_{CA.s}(A.s, "Alice", 2005, h^{(11)}(x)))$
    - And for Feb 2005: $Cert_A$, $h^{(10)}(x)$
    - Validate incoming $Cert_A$, h_{10} by $h^{(11)}(x)=h(h_{10})$
    - Security based on random choice of x and h being one-way premutation
  - Often, requiring frequent CRL is more efficient

SSL / TLS PKI Challenges

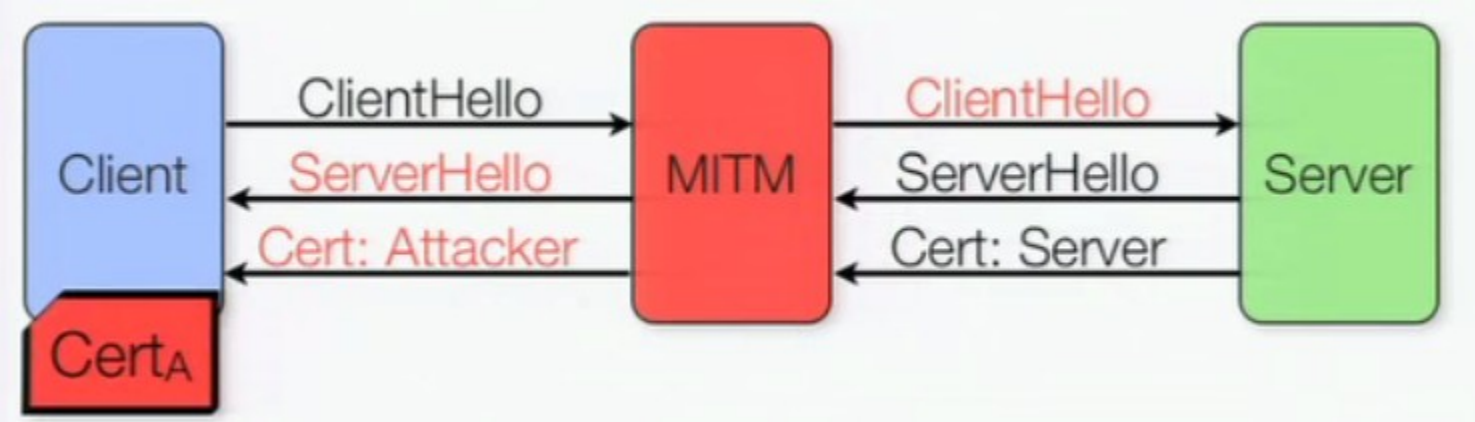
- Many CAs `trusted' in browsers
- Naming constraints NOT used
  - Every CA can certify any domain
- Several well-known failures
- DigiNotar, Comodo, Stuxnet, …

TLS Interception / MitM Attack



CertA is a fake-but-valid certificate for the identity of Server

TLS Interception / MitM Attack



Interception is used ethically, by 'locally' adding a CA, by many organizations, for filtering SSL/TLS traffic from malware, etc.

But also by attackers…

Defenses against Corrupt CAs

- **Use naming constraints to limit risk**
  - who can issue global TLDs (.com, etc.)?
- **‘Burned-in’ public keys (e.g., for Google)**
  - Detected MitM in Iran, using DigiNotar CA
- **Certificate / public-key pinning (HPKP)**
  - Server: I always use this PK / Cert / Chain
  - Client: remember and implement!
- Certificate Transparency (CT): Accountability
- Origin-bound certificates

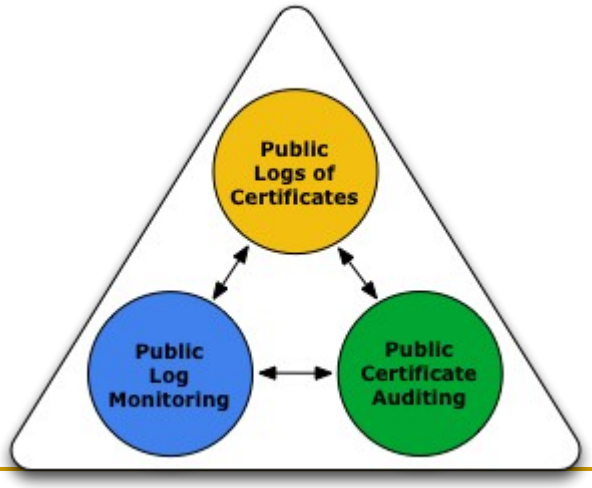
Defenses against Corrupt CAs

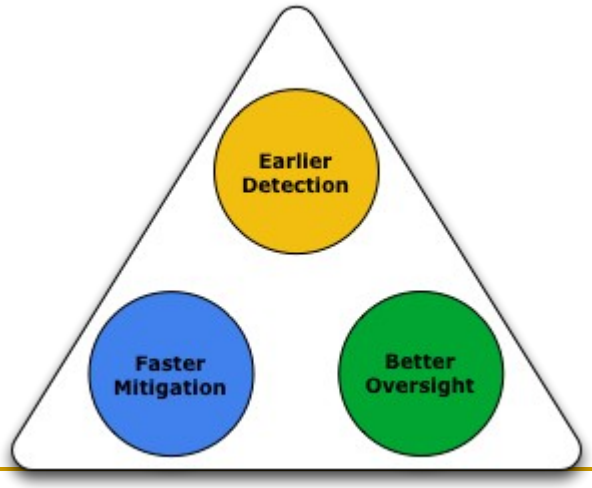
- Use naming constraints to limit risk
- ‘Burned-in’ public keys (e.g., for Google)
- Certificate / public-key pinning (HPKP)
- **Certificate Transparency (CT): Accountability**
- Threshold schemes

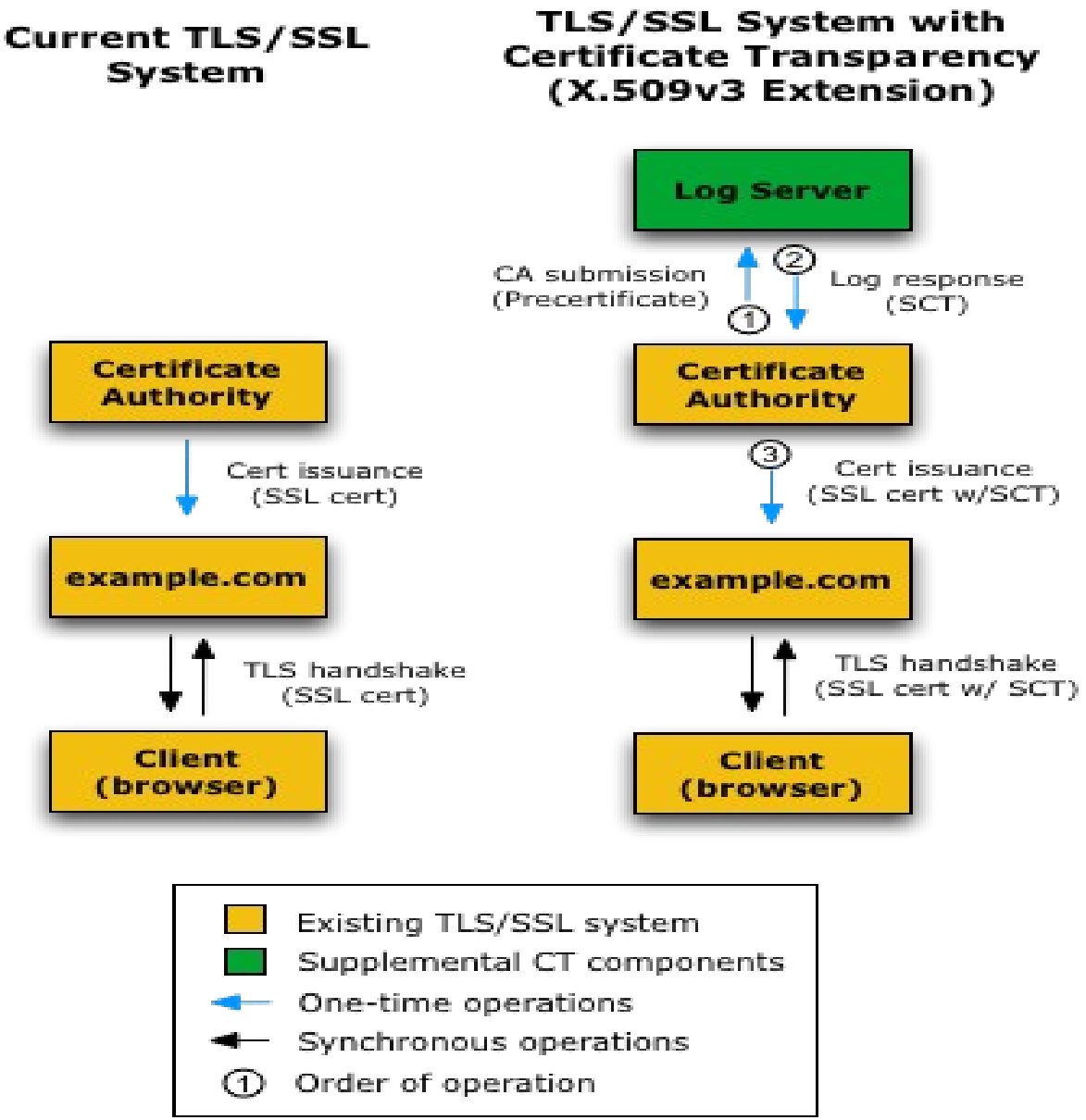
Certificate Transparency brings CA Accountability

- Certificate Logs
- Monitors [e.g., CA]
- Auditors [browser]

- Early detection of mis-issued certificates, malicious certificates, and rogue CAs.
- Faster mitigation after suspect certificates or CAs are detected.
- Better oversight of the entire TLS/SSL system.



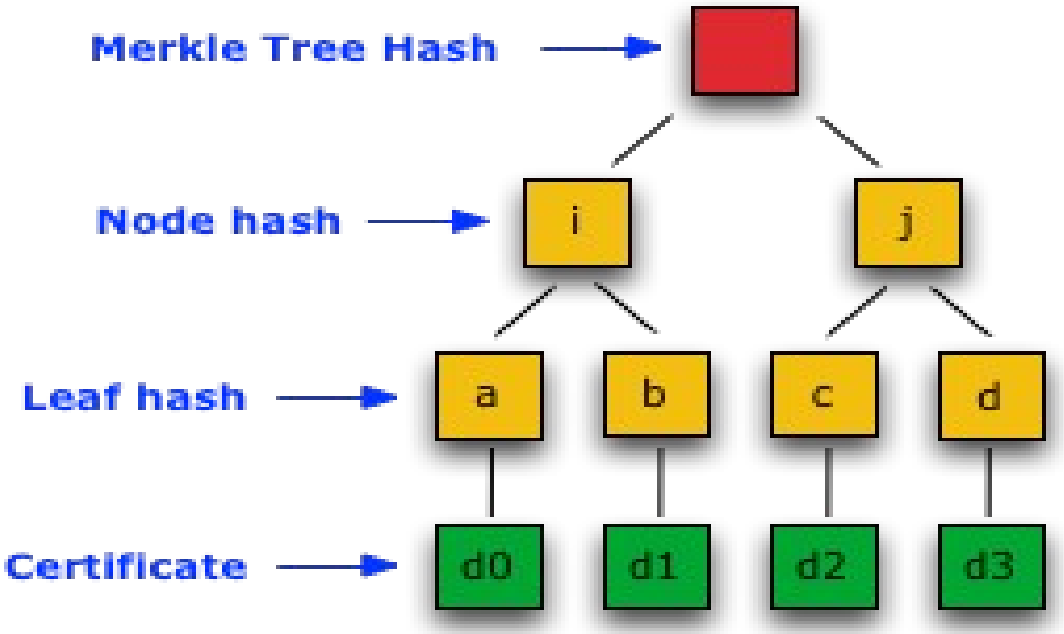




SCT: Signed Certificate Timestamp (time cert added to log)

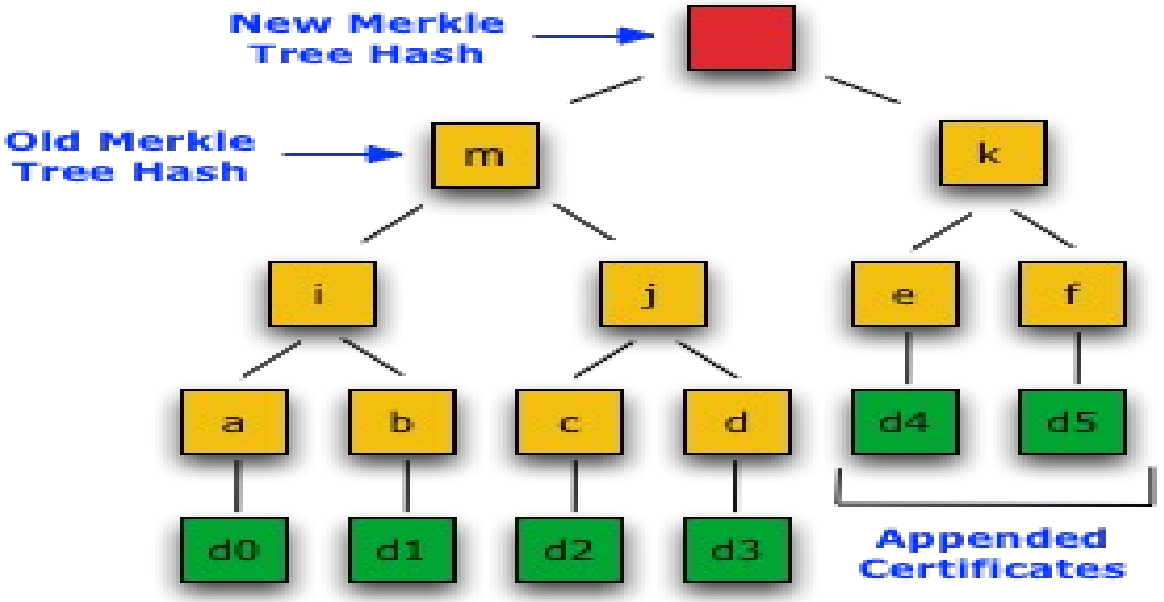
Merkle Tree - Log

- Logs use Merkle hash Tree.
- Every node is labeled with hash of labels of its children nodes.
- d0, d1, d2, d3 – certificates.



Merkle Tree for Certificate Transparency

- Accept a cert only with proof of existence in tree
- Accept new hash-of-tree only with proof of extension of old hash-tree
- Append only
- Cryptographically assured
- Publicly auditable



Key Establishment & PKI : Conclusions

- Key Establishment: use PKs, cert for ‘handshake’
  - SSL/TLS: mature standard, widely used
    - Many vulnerabilities in older versions
    - Slow adoption of newer versions
- PKI & Trust: still challenging, active areas
  - SSL/TLS certs: too many legit CAs, no naming constraints
  - How to deal with rogue CAs?
    - Certificate Transparency (accountability) ?
  - Client certificates (authentication) ???