CSE 3400 - Lecture set 8: Public Key Infrastructure (PKI)

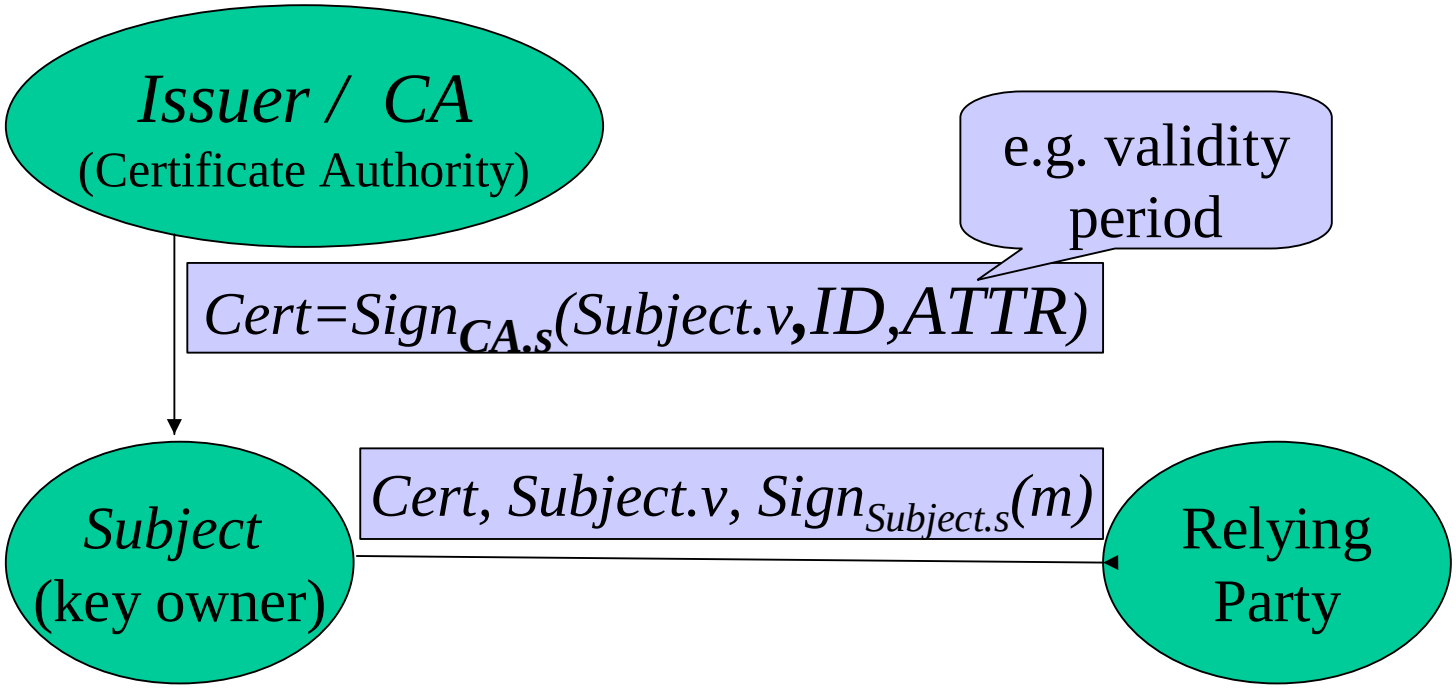
Last updated: 4/21/20

Public keys are very useful…

- Secure web connections
- Software signing (against malware)
- Secure messaging, email
- Crypto-currency, blockchains, financial crypto…
- But to use a PK, it must be authenticated
  - Mainly: signed by a **trusted Certificate Authority**
    - E.g., in TLS, browsers maintain list of ‘root CAs’
  - So far, not much use of personal certificates
    - Secure email: not widely deployed
    - Secure messaging: auth by provider (& user ??)

Public Key Certificates & Authorities

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



Can we trust a certificate / CA?

- Browser default: trust ~100 ‘root CAs’
  - Each can define ‘intermediate CAs’
- But failures have happened… E.g.:
  - 2001, VeriSign: attacker gets code-signing certs
  - 2008, Thawte: email-validation (attackers’ mailbox)
  - 2008,2011: Comodo not performing domain validation
  - 2011: DigiNotar compromised, >500 rogue certs
    - Incl. wildcard cert for Google for mass-interception in Iran
  - 2011: TurkTrust issued intermediate-CA certs to users
  - 2014: India CCA / NIC compromised (rogue certs)
  - 2015: Root CA CNNIC (China) issued CA-cert to MCS (Egypt) who issued rogue certs
  - 2015,17: Symantec issued unauthorized certs for >176 domains

PKI Lecture: Topics

- **X.509 Certificates**
- Revoking certificates
- Dealing with CA failures: CT (Certificate Transparency) and other defenses
- Conclusions and challenges…

The X.500 Global Directory Standard

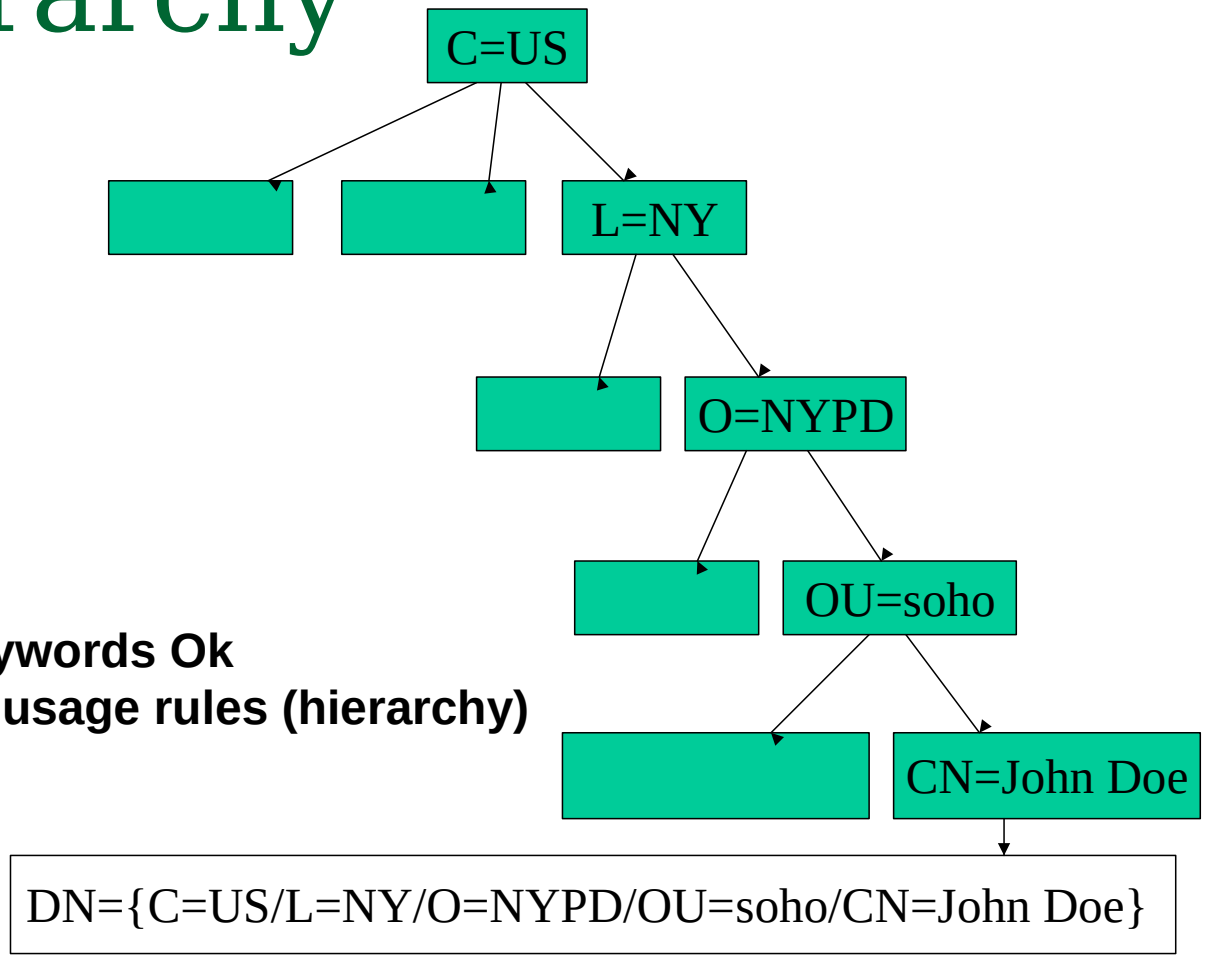
- X.500: an ITU standard, first issued 1988
  - ITU: International Telcos Union
- Idea: trusted global directory
  - Operated by hierarchy of trustworthy telcos
  - Never happened
    - Too complex, too revealing, too trusting of telcos
- Directory bind identifiers to attributes
  - Standard attributes (incl. public key)
  - Standard identifiers: Distinguished Names
    - Goal: unique, meaningful, decentralized identifiers

X.500 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:

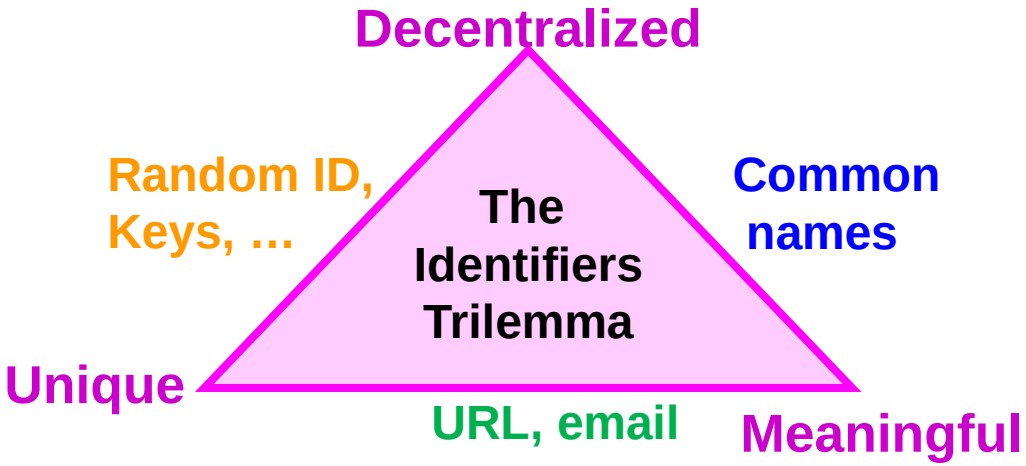
1. **Other keywords Ok**
2. **No strict usage rules (hierarchy)**

Goals for Identifiers in Certificates

- Meaningful (to humans)
  - Memorable, reputation, off-net, legal
- Unique identification of entity (owner)
- Decentralized - with Accountability: assigned by a trusted (certificate) authority
  - Accountability: identification of assigning authority

The Identifiers Trilemma

- Three goals: Meaningful, Unique, Decentralized
- Examples of achieving any two of the goals:
  - Unique + Meaningful: URL, email
  - Meaningful + Decentralized: common name
  - Unique + Decentralized: hash of key
- **Ensuring all three goals seems challenging!**



Distinguished Names - Evaluation

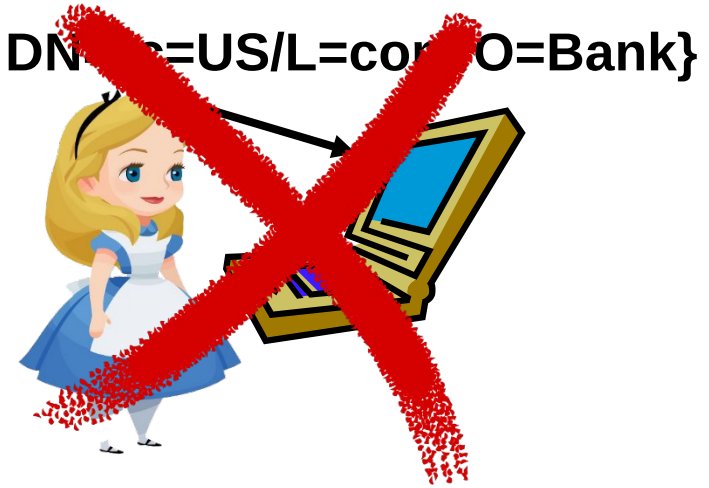
- **Decentralized?** 
  - Separate name spaces
- **Unique ?** 
  - Could be, if each name space has one issuer
  - TLS reality: browsers trust 100s of CAs for **every name**
- **Meaningful?** 
  - Usually: Julian Jones/UK/IBM
  - But not always: Julian Jones2/UK/IBM
    - Added ‘counter’ to distinguish mistakes, loss of meaning

Distinguished Names – More Problems

- Distinguished Name fields may expose
  - Organizational sensitive information (e.g. unit)
  - Privacy
- Handling changes in position, organizations
- Multiple, related hierarchies:
  - International organizations, divisions…
    - Julian Jones/UK/IBM or Julian Jones/IBM/UK ?
    - Julian Jones/Research/UK/IBM or Julian Jones/UK/Research/IBM ?

And: DNs aren't usable identifiers

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

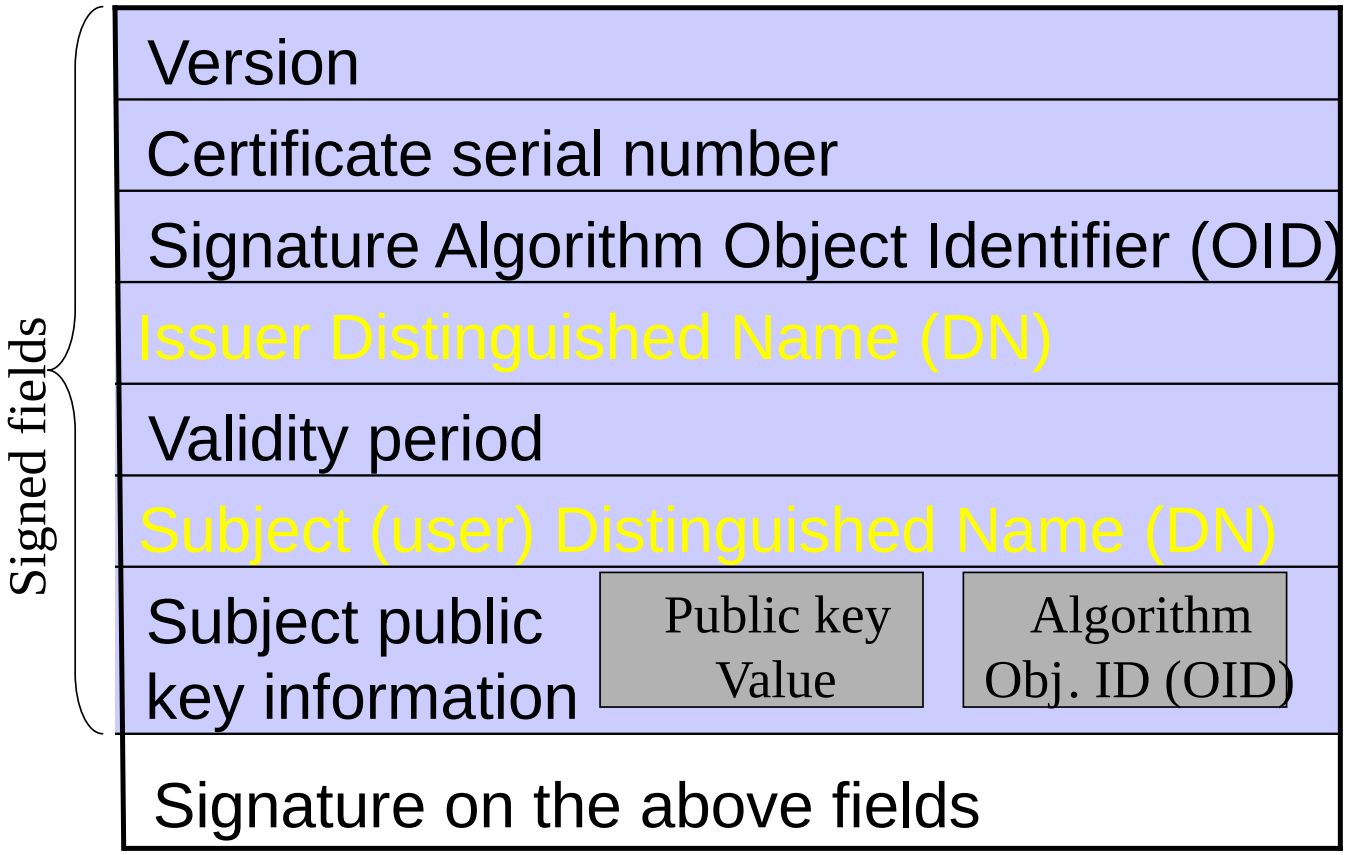


- Hopefully, they know the domain (in URL)
- Naming extensions: alternative names
  - DNS name: `cert.SubjectAltName.dNSname`
  - Wildcard domain names: *.bank.com

X.509 public key certificates

- X.509: authentication mechanisms of X.500
- Initially: Authenticate to Directory (PW, Pub key)
  - To maintain entity's record
- Later (and now): **X.509 certificate**
  - Signature binds **public key** to distinguished name (DN)
  - And to other attributes
    - Some defined in X.509 standard, others in `extensions`
- Used widely
  - SSL / TLS, code-signing, PGP, S/MIME, IP-Sec, …
  - In spite of complexity

Original (V1) X.509 Certs



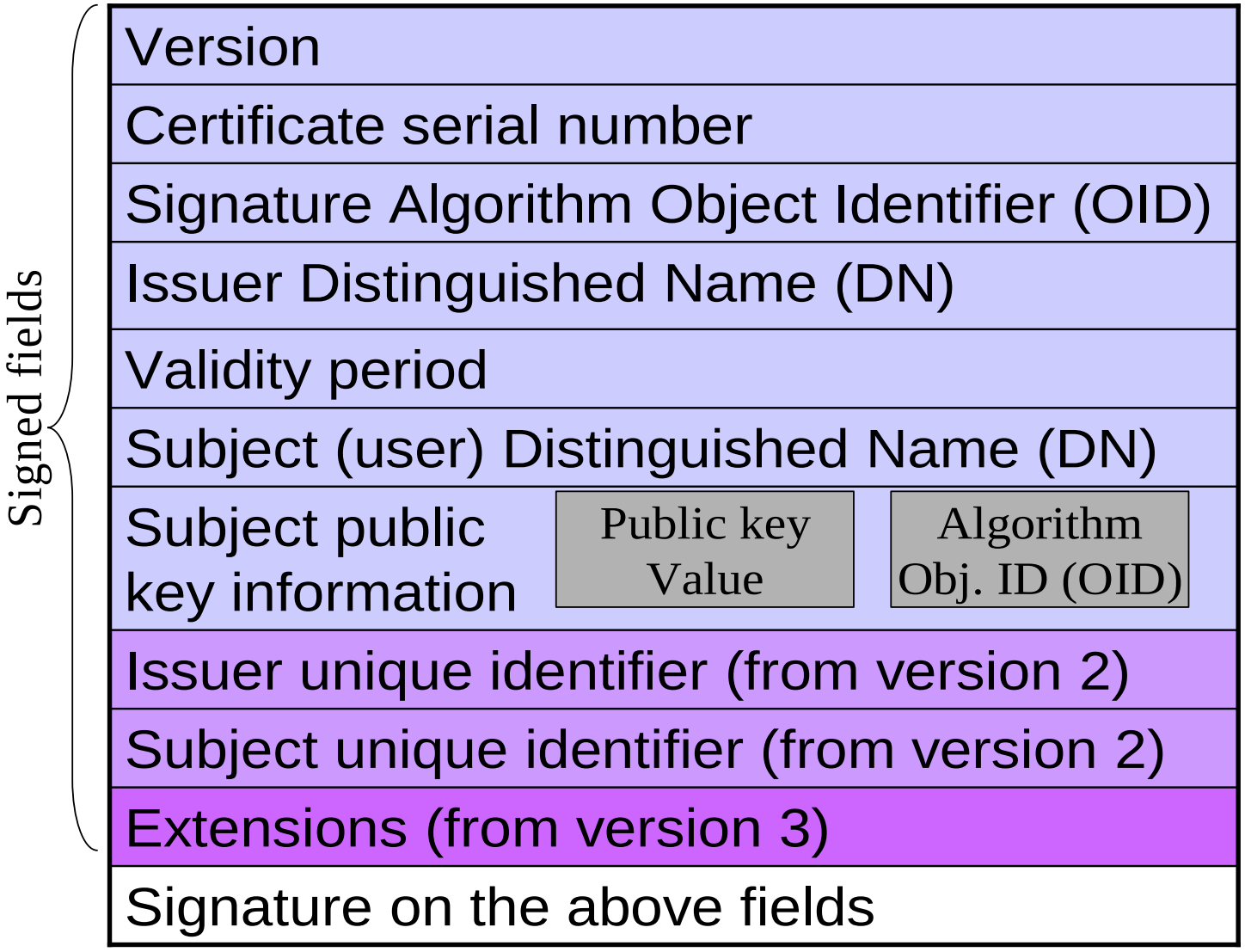
Object Identifiers (OID):

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

X.509 Certs & Subject Identifiers

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

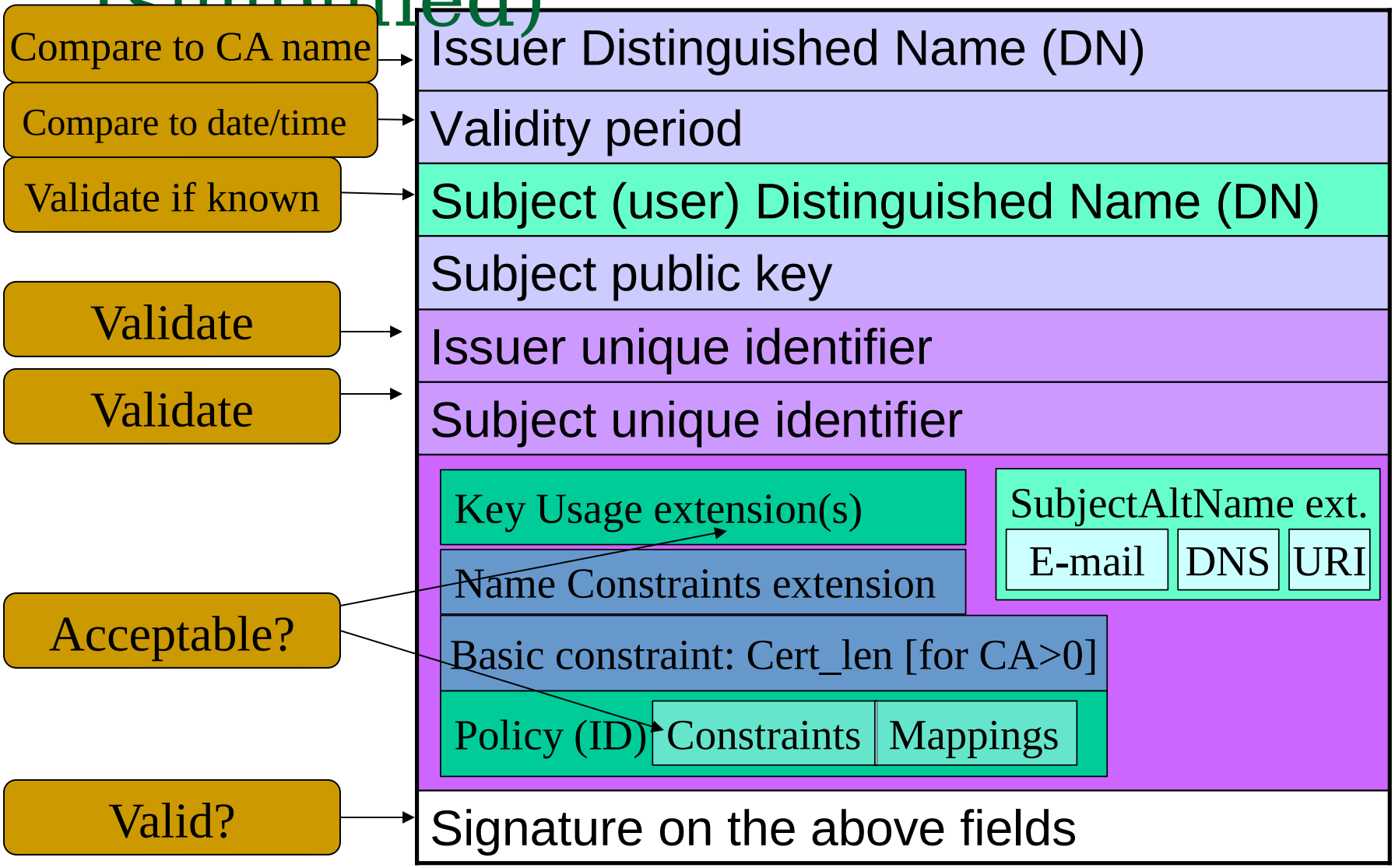
X.509 Public Key



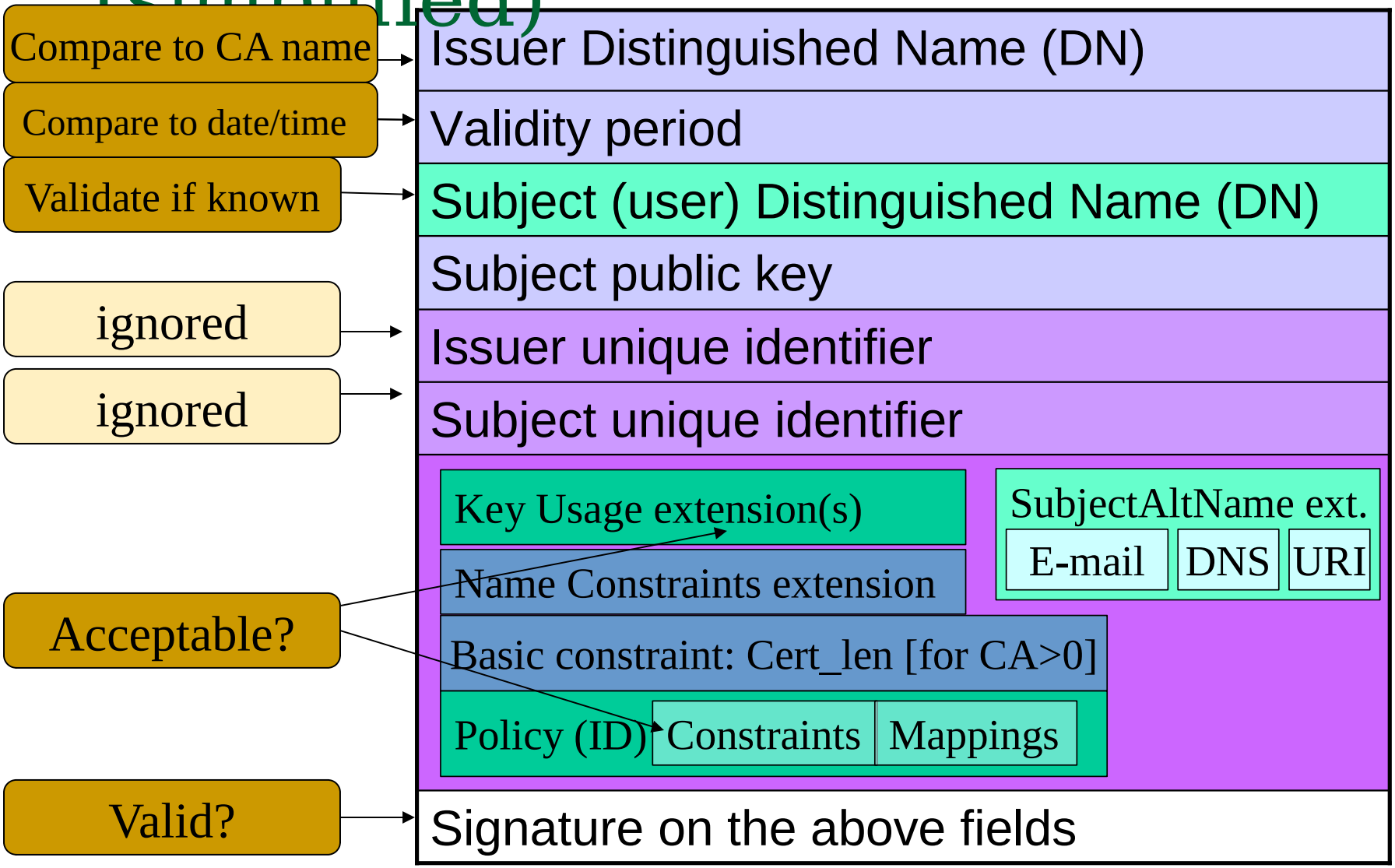
X.509 V3 Extensions Mechanism

- Each extension contains…
- Extension identifier
  - As an OID (Object Identifier)
  - E.g. `Naming constraints`
- Extension value
  - E.g. `Permit C=IL`, `Exclude dNSName=IBM.COM`
- **Criticality indicator**
  - **If critical**, relying parties MUST NOT use a certificate with any unknown critical extension
    - E.g. Naming constraints is `critical`
  - **If non-critical:** use certificate w/o unknown critical extensions; **ignore** unknown (non-critical) extensions

X.509 Certificate Validation (simplified)



PKIX Certificate Validation (simplified)



PKIX: Internet PK Infrastructure

- RFC 5280: X.509 + Internet extensions
- Addressing different goals:
  - **Improved naming:**
    Subject, Issuer AltName extensions
  - Establishing trust in CAs:
    Basic, Name and Policy Constraints extensions
  - Other goals and extensions

SubjectAltName (SAN) Extension

- Bound identities to the subject
  - In addition/instead of Subject Distinguished Name
  - Same extension may contain multiple SANs
- Goal: unique and meaningful names
  - Common: DNS name (dNSName), e.g., a.com
    - TLS/SSL allows wildcard domains (*.a.com)
      - Not addressed in PKIX (RFC5280); see RFC6125
  - Or: email address, IP address, URI, other
  - Don’t use (ignore) X.509v2 `unique identifiers’

IssuerAltName Extension

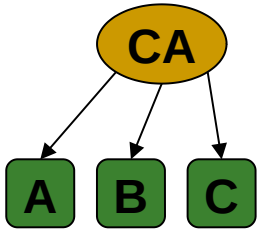
- Bound identities to the issuer
  - In addition/instead of Issuer Distinguished Name
- Same goal, syntax as SubjectAltName (SAN)

PKIX: Internet PK Infrastructure

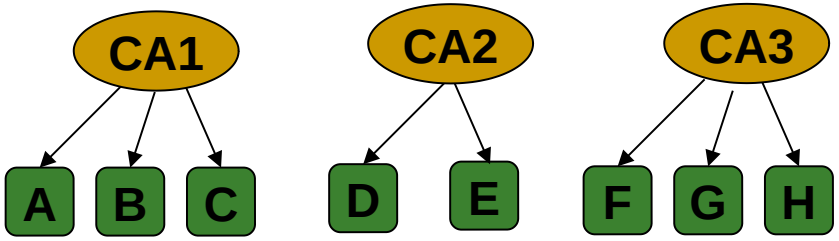
- RFC 5280: X.509 + Internet extensions
- Addressing different goals:
  - Improved naming:
    Subject, Issuer AltName extensions
  - **Establishing trust in CAs:**
    Basic, Name and Policy Constraints extensions
  - Other goals and extensions

Different types of PKIs

- **Simple PKI:** just one trusted CA:



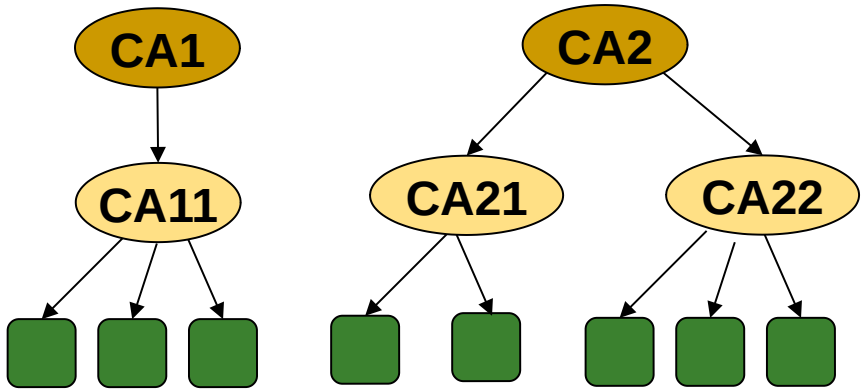
- **Multiple trusted CAs:**



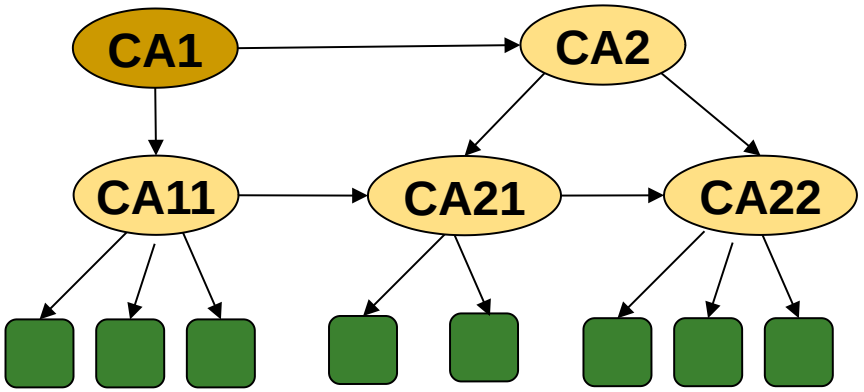
- Both **static**: CAs known, trusted in advance
- **Dynamic PKI:** establishing trust in ‘new’ CA, via certificates from ‘established’ CAs

Dynamic PKIs

- **Web/TLS PKI:** ‘root CAs’+‘intermediate CAs’:
  - Root CA issues cert for intermediate CAs
  - Supported by PKIX



- Or… **Web-of-Trust PKIs:**
  - Directed graph, not tree
  - Different variants/policies
    - Some supported by PKIX



Web of Trust PKI

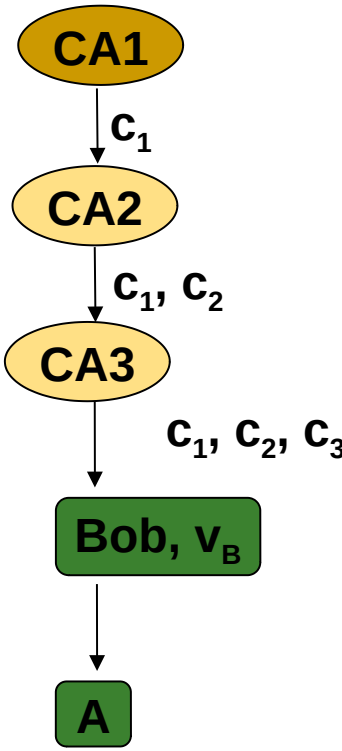
- PGP's friends-based Web-of-Trust:
  - Everyone is subject, CA and relying party
  - As a CA, certify (pk, name) for `friends'
  - As a subject, ask friends to sign for you
  - As a relying party, trust certificates from friends
    - Or also from friends-of-friends? Your policy….

PKIX PKI Trust Rules

- PKIX allows **path-based PKI policies**
  - A certificate path is a set $c_1,\ldots,c_n$ of certs, s.t. c_i is validated using $c_{i-1}.v$ (sig-valid-key in previous cert)
- Given cert-path $c_1,\ldots,c_n$, cert c_n is valid if:
  - The first cert c_1 was issued by a trusted CA,
  - **Basic, naming** and **policy constraints** satisfied
  - $c_1,\ldots,c_{n-1}$ are CA certs:
    - BasicConstraint.cA and keyUsage.keyCertSign are TRUE
    - □ All their subjects become trusted CA (‘dynamic’)

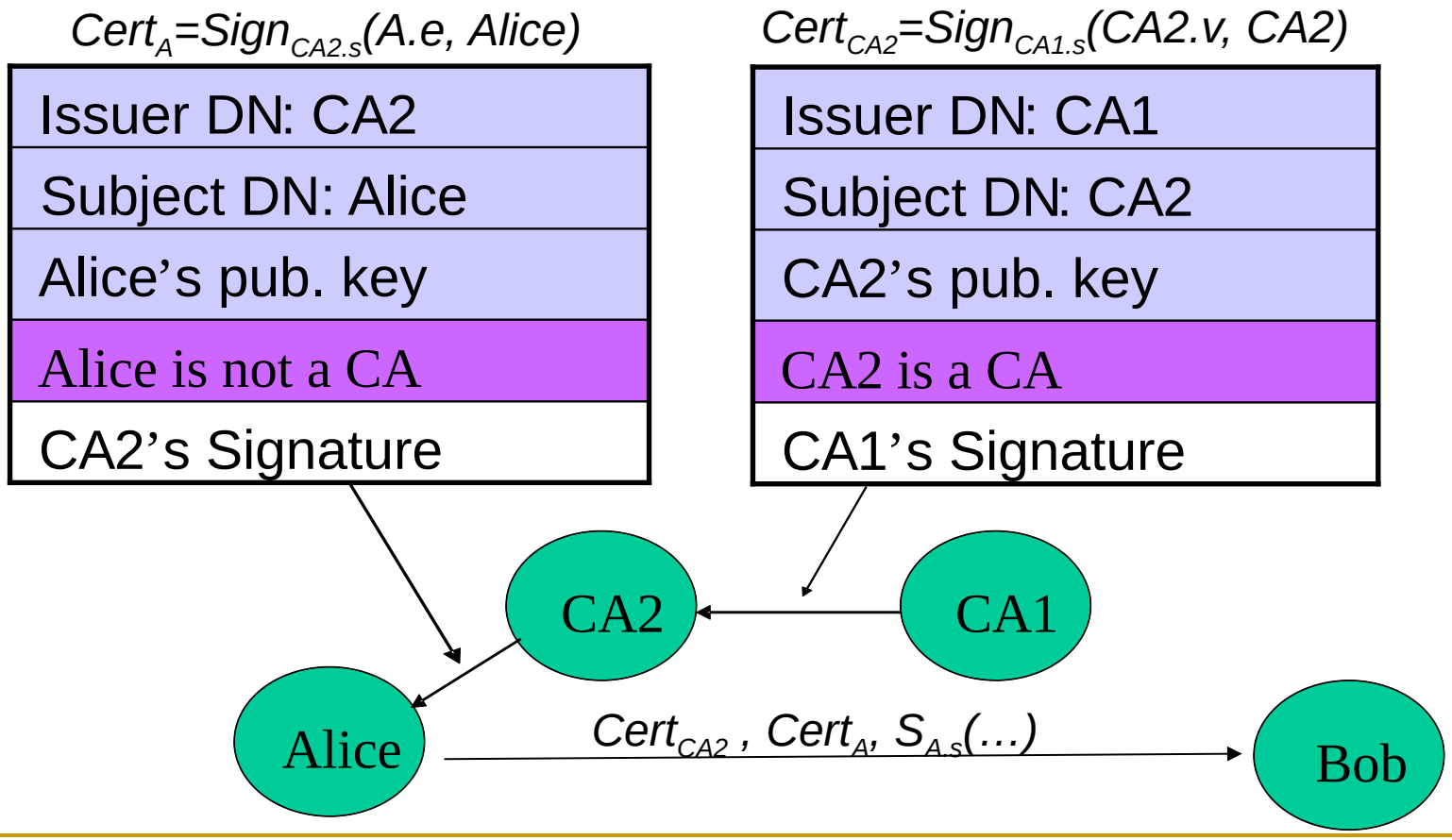
Establishing trust in cert in PKIX

- Relying party A trust c_3 , given also c_1, c_2 , if:
  - CA1 is a trust-anchor (‘root CA’)
  - $Valid_{CA1.v}(c_1)$ [signed by CA1.v]
    - Let v_{i+1} be the public key signed in c_i
  - $Valid_{v1}(c_2)$, $Valid_{v2}(c_3)$
    - Basic constraints:
      - c_1, c_2 mark CA2, CA3 (resp.) as CAs
      - Path-Length: in c_1 at least 2, in c_2 at least 1
    - Policy, naming constraints satisfied



Example: delegation of trust

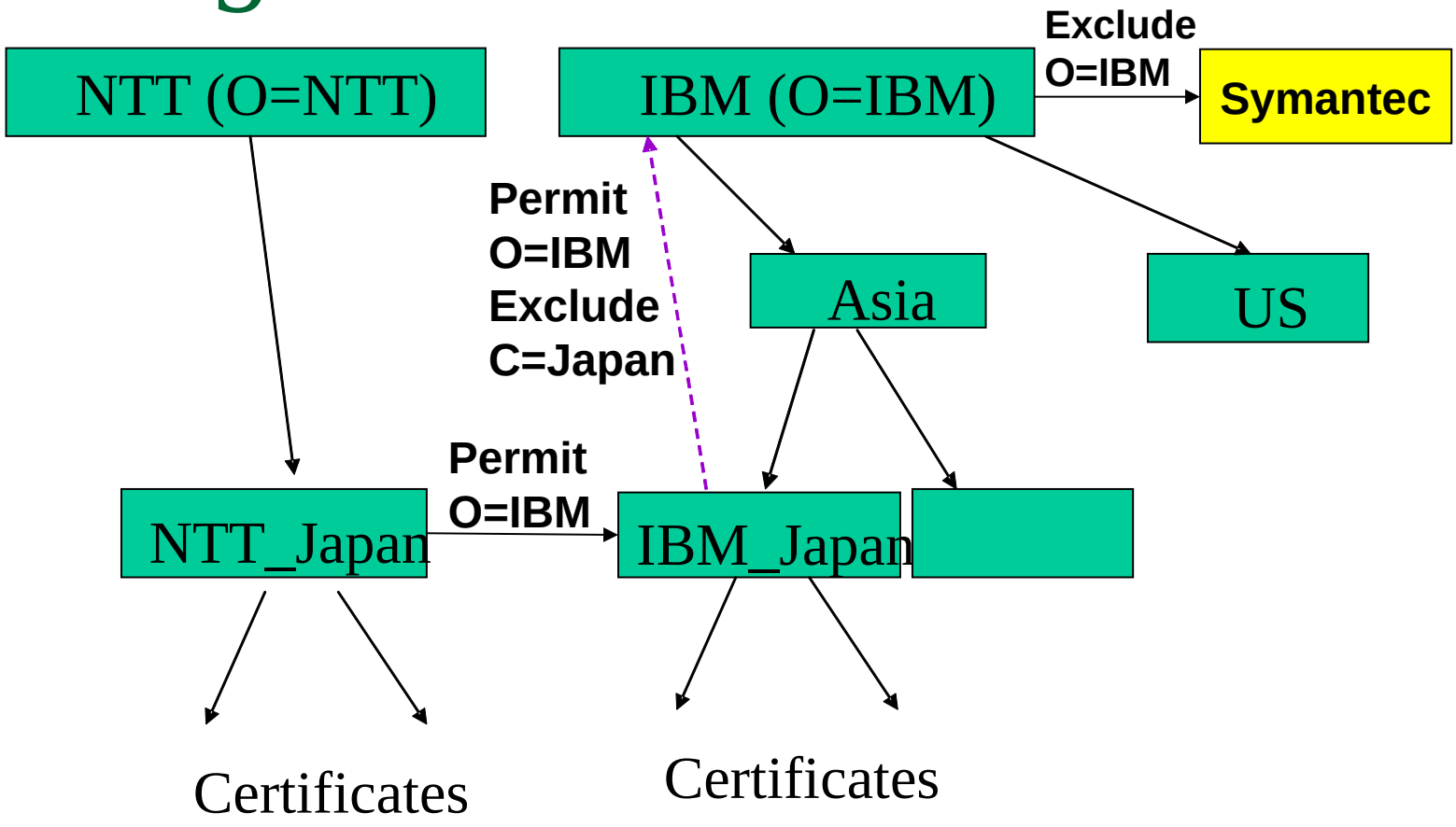
- CA1 `vouches for' CA2
- ⇒ Interoperability (if Bob knows only CA1)



Constraints

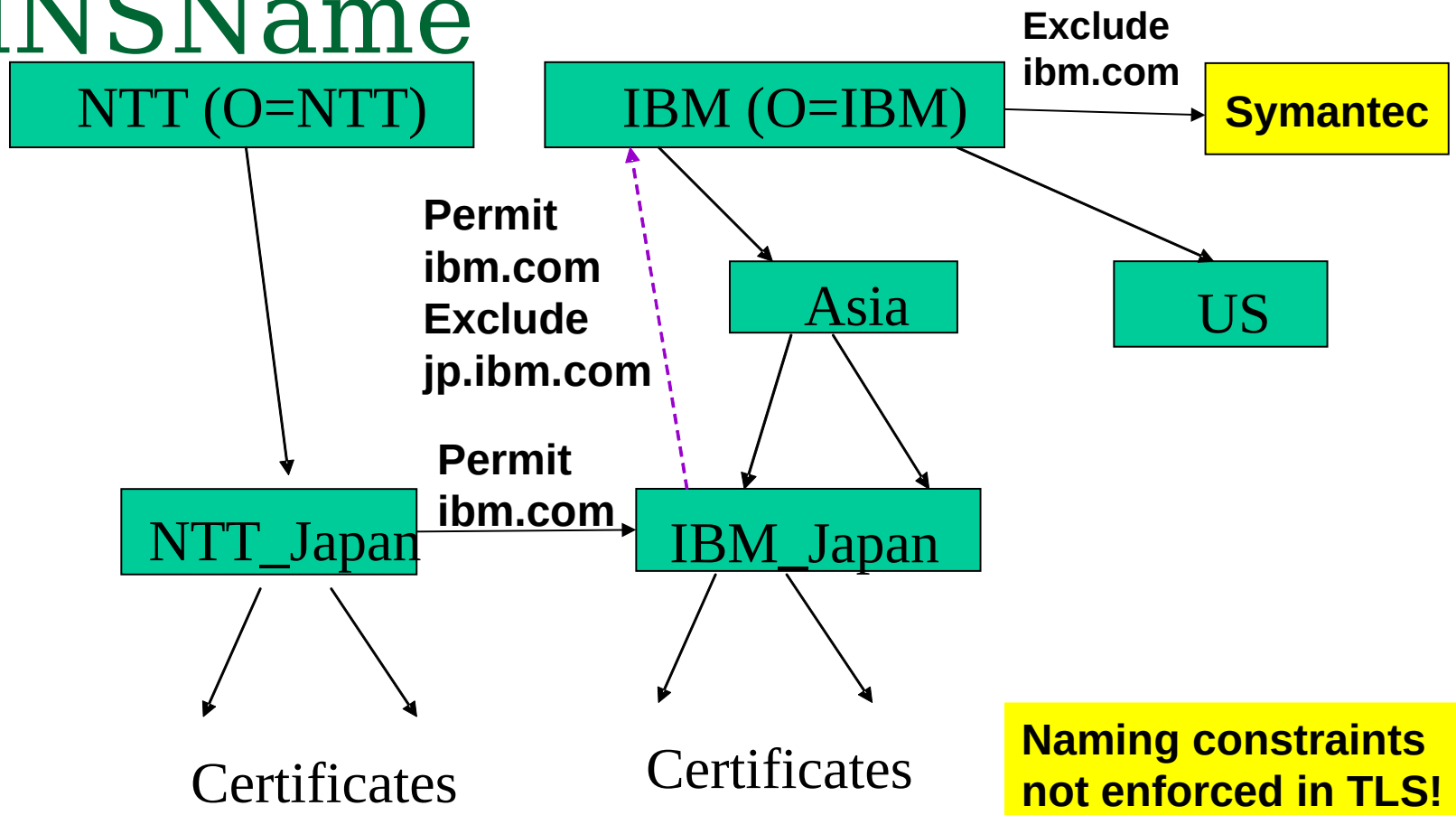
- Basic constraints:
  - Is the subject a CA? (default: FALSE)
  - Maximal depth of paths with this certificate?
    - pathLengthConstraint
- Policy constraints:
  - Interoperability of certificate-policy extensions
  - May allow/forbid `policy mappings' (extension, too)
- Naming constraints
  - Constraints on DN and SubjectAltName - in certs issued by subject
    - Only relevant when subject is a CA !
  - 'Permit' and 'Exclude'

Naming constraints on DN



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

Naming constraints on dNSName



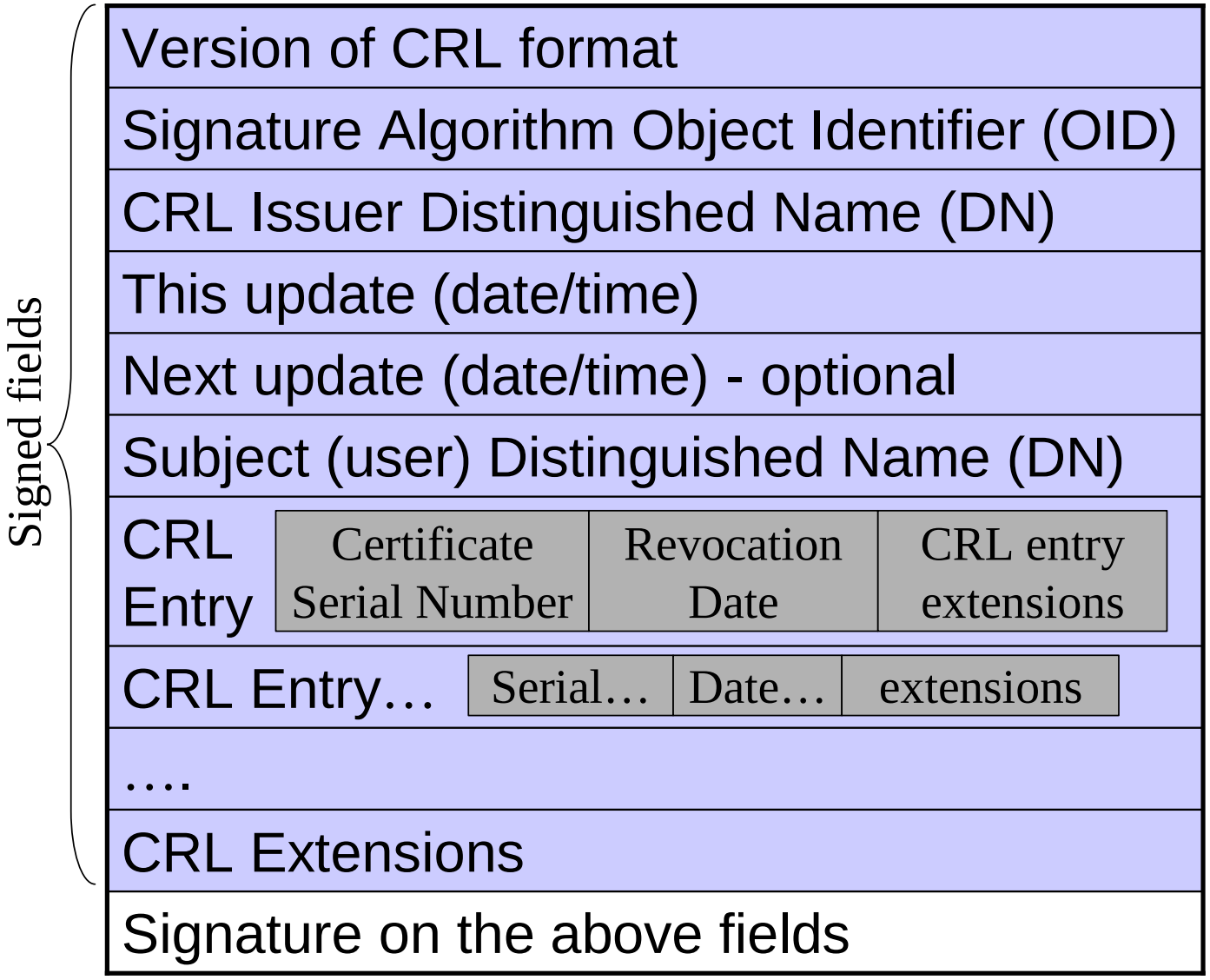
PKI Lecture: Topics

- X.509 Certificates
- **Revoking certificates**
- Dealing with CA failures: CT (Certificate Transparency) and other defenses
- Conclusions and challenges…

Certificate Revocation

- Reasons for revoking certificates
  - Key compromise
  - CA compromise
  - Affiliation changed (changing DN or other attribute)
  - Superseded (replaced)
  - Cessation – not longer needed
- How to inform relying parties? Few options…
  - 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

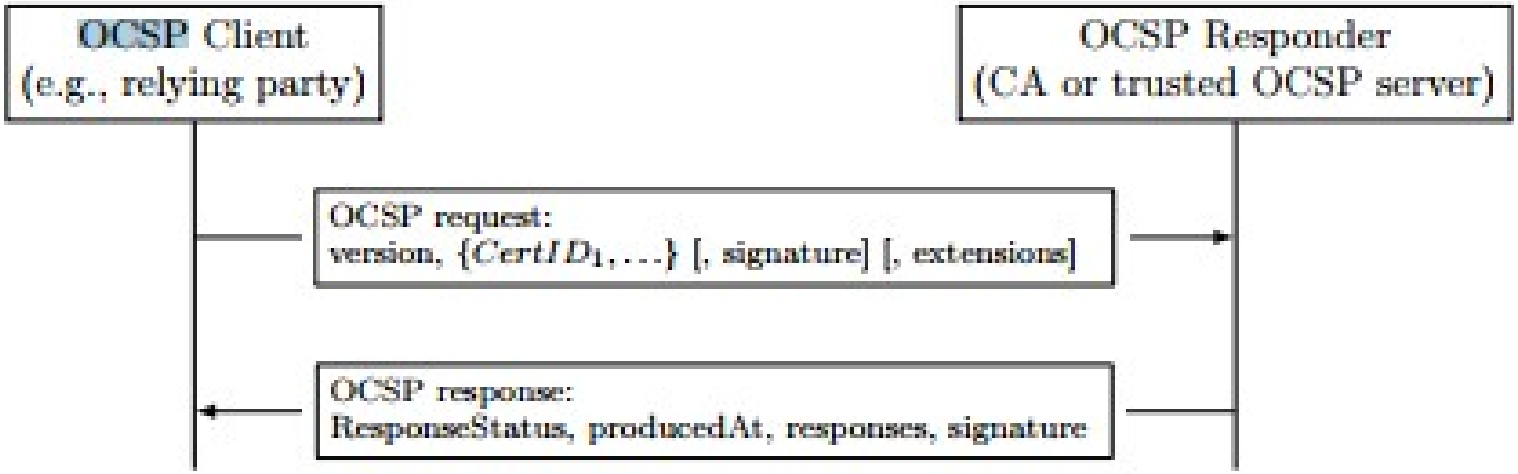


Revocation with CRLs is Difficult

- 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?
- Solutions:
  - More efficient CRL schemes
    - 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`
- Browsers mostly do not check CRLs. Instead:
  - Online Certificate Status Protocol (OCSP)
  - Short validity for certificates

Online Certificate Status Protocol (OCSP)

- Improve efficiency, freshness cf. to CRLs
- Client asks CA about cert during handshake
- CA signs response (real-time)



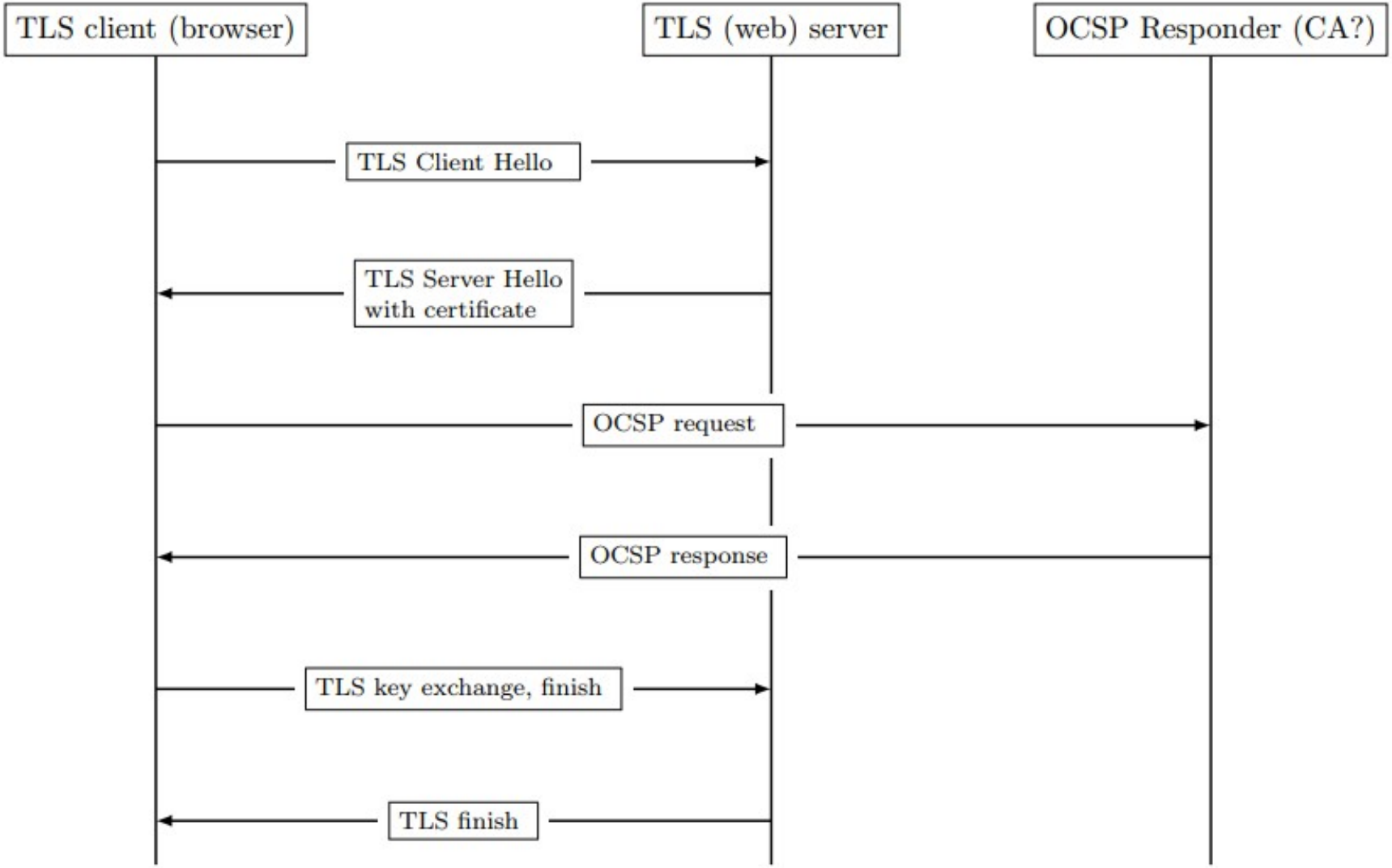


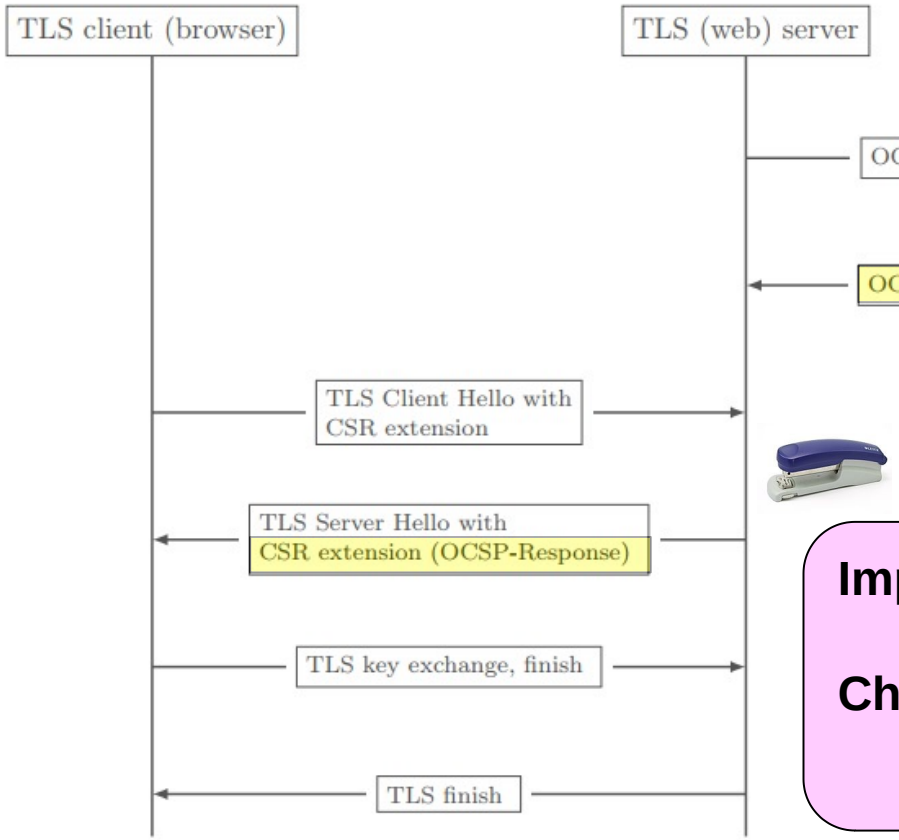
Figure 1: TLS handshake using OCSP (no stapling).

Online Certificate Status Protocol (OCSP)

- Client asks CA about cert during handshake
- CA signs response (real-time)
- Challenges:
  - Privacy: expose (domain, client) to CA
  - Load on CA
  - Delay (latency): on average, almost a second
  - Reliability: what if CA fails? No connectivity?
  - □ Most browsers skip OCSP or soft-fail: continue w/o OCSP response
- Better way to do OCSP?

Solution: OCSP-Stapling

Server runs OCSP, sends (`staples') the CA-signed response (CSR) during TLS handshake



Improve efficiency, privacy, reliability

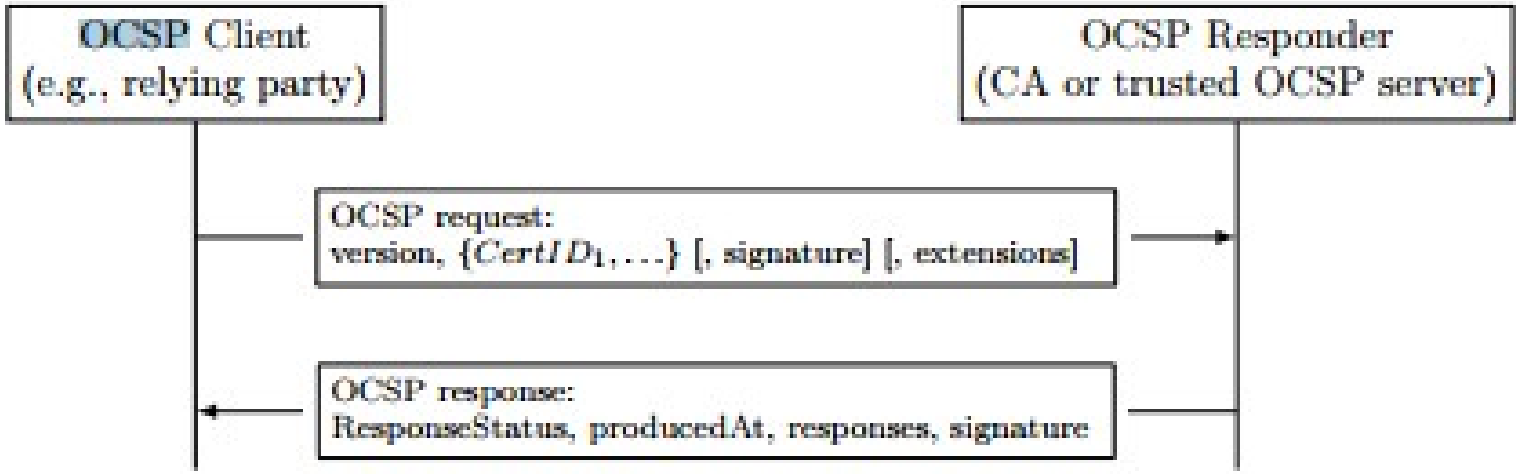
Challenge: many servers don't staple!
Or: staple `sometimes/usually'

OCSP-Stapling

- Server runs OCSP, sends (`staples') the CA-signed response during TLS handshake
- Challenge: many servers don't staple!
  - Or, worse: staple `sometimes/usually'
  - So, try OCSP? Connect anyway? Disconnect?
- Solution: `Must-staple' cert. extension
  - RFC 7633
  - Mark as not critical
    - As it may not be supported by some browsers

Online Certificate Status Protocol (OCSP)

- Most browsers don't use CRLs. Why?
  - Efficiency, freshness concerns
- OCSP: check cert-status `as needed'
- Signed responses (from trusted CA/server)



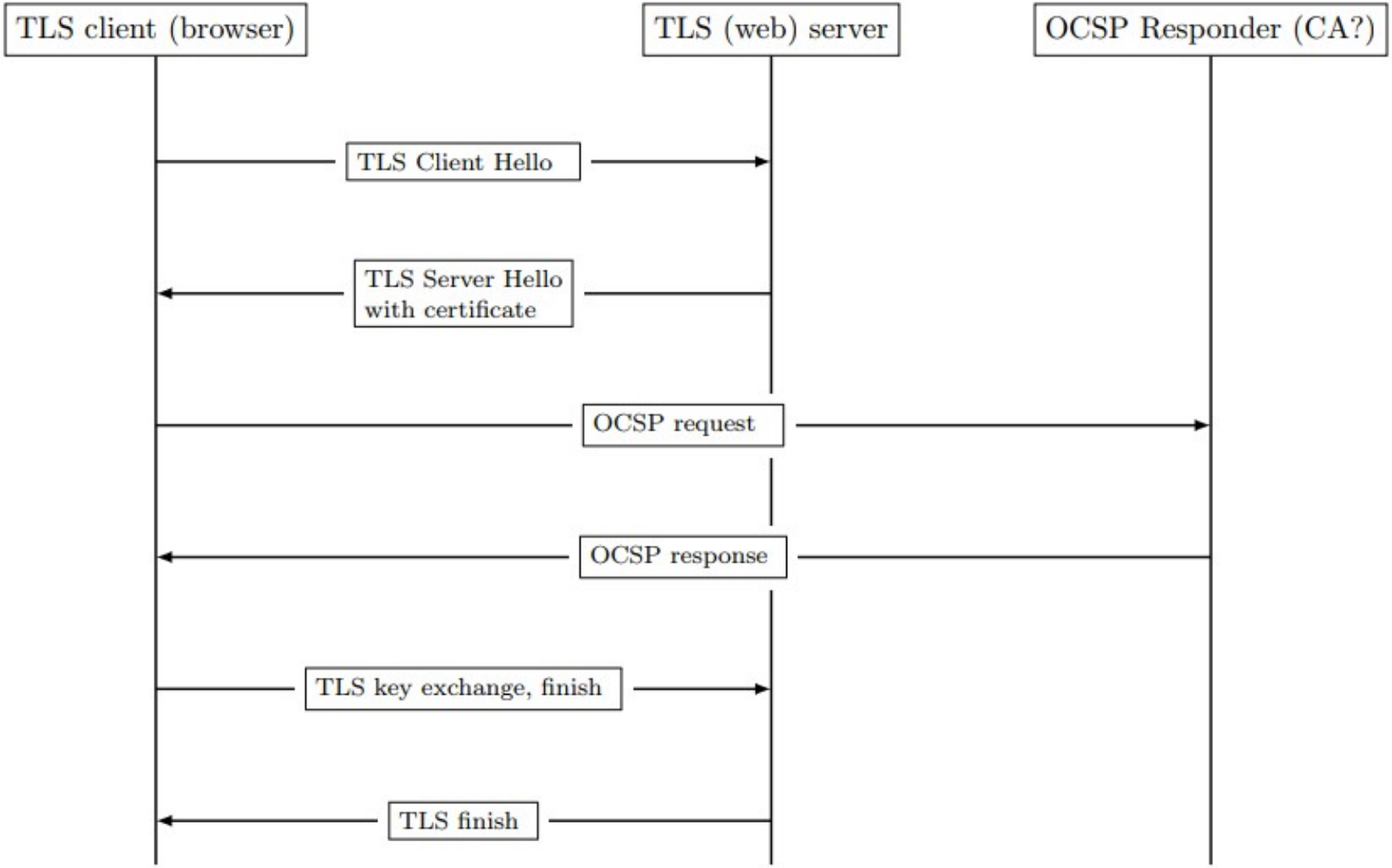
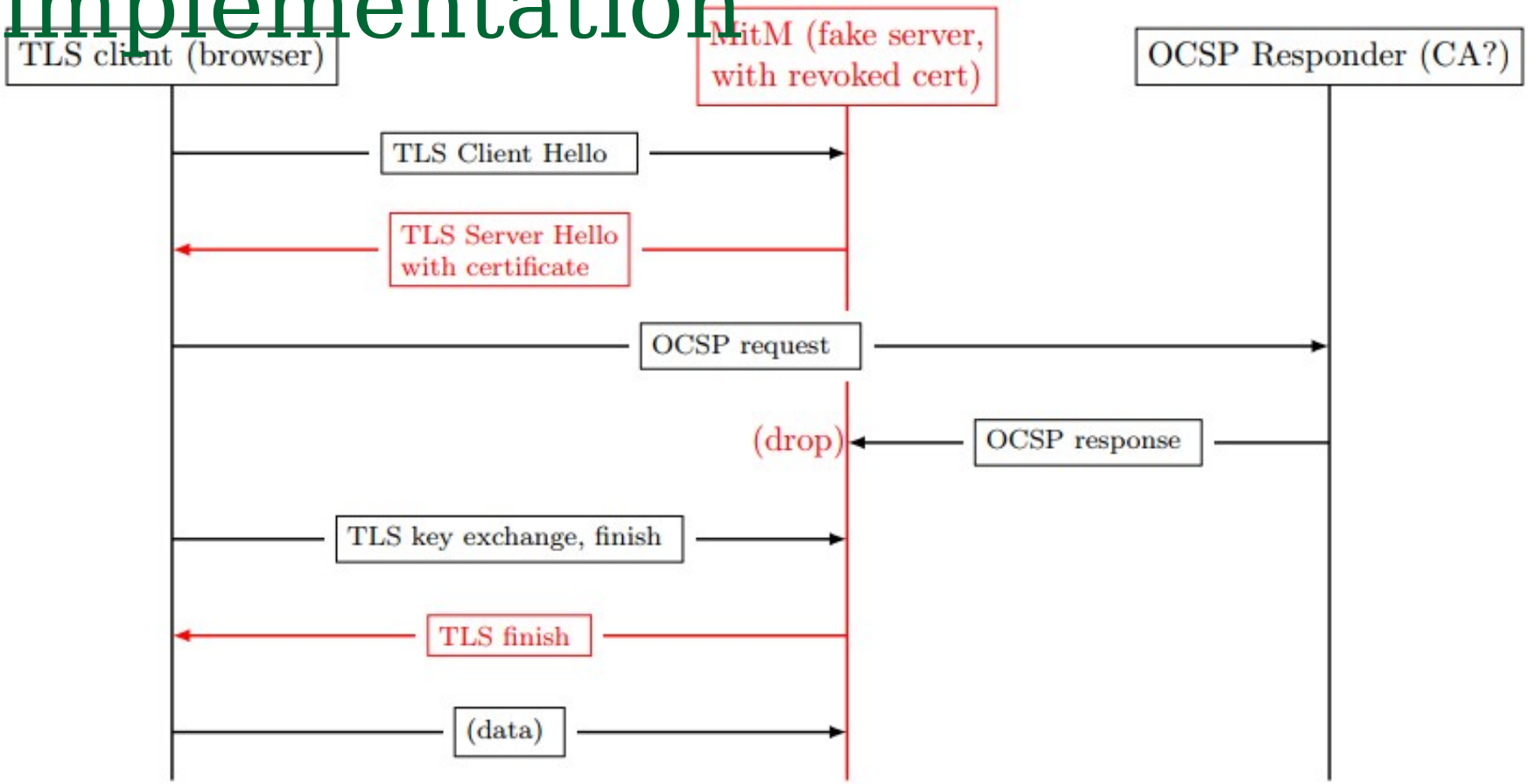


Figure 1: TLS handshake using OCSP (no stapling).

Online Certificate Status Protocol (OCSP)

- Client asks CA about cert during handshake
- CA signs response (real-time)
- Challenges:
  - Privacy: expose (domain, client) to CA
  - Load on CA
  - Delay (latency): on average, almost a second
  - Reliability: what if CA fails? No connectivity?
  - ⇨ Most browsers skip OCSP or soft-fail: continue w/o OCSP response
- Improved mechanism: OCSP-stapling

Attacking typical OCSP implementation



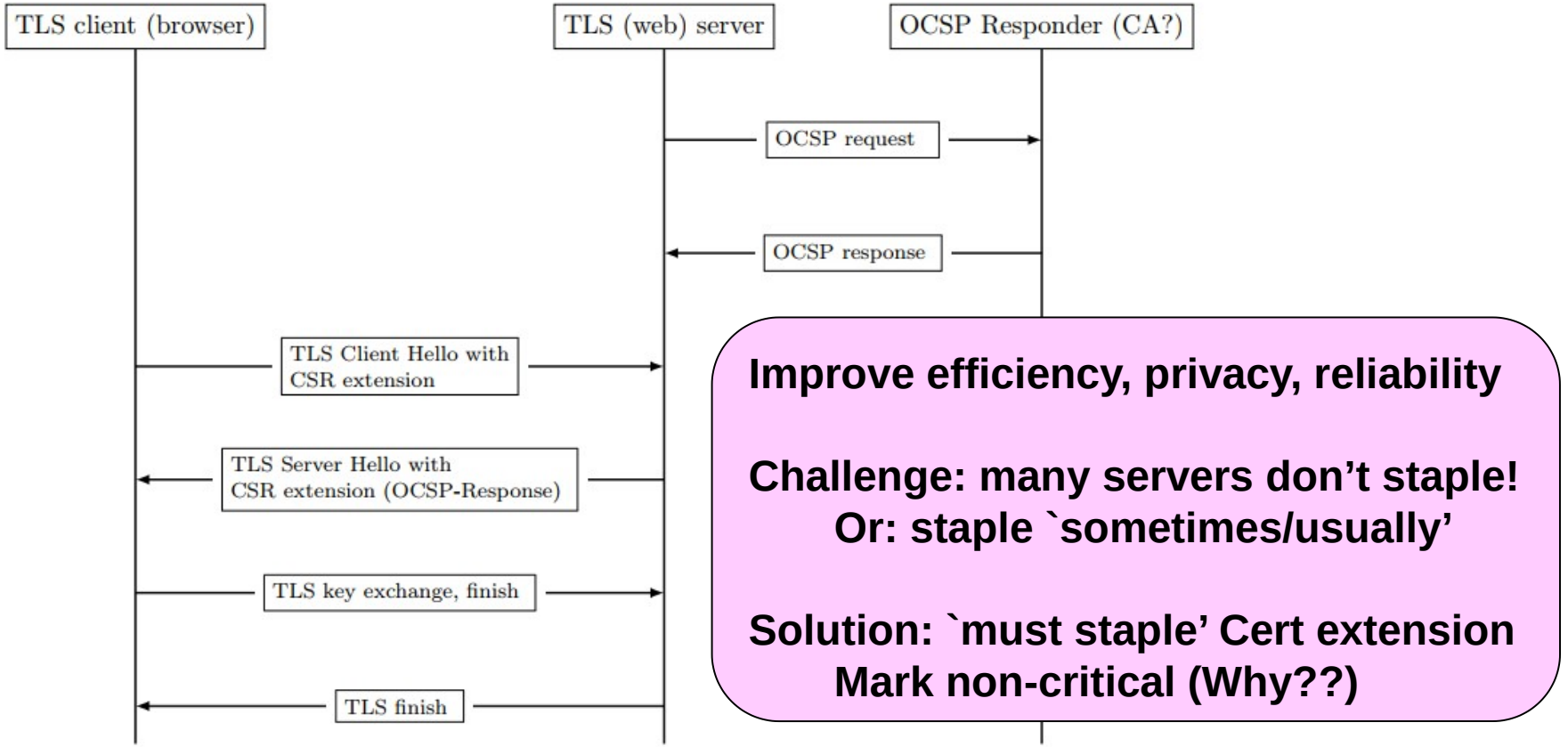
OCSP-Stapling

Server runs OCSP, sends (`staples') the CA-signed response during TLS handshake

Show sequence diagram for TLS handshake with OCSP-stapling

OCSP-Stapling

Server runs OCSP, sends (`staples') the CA-signed response during TLS handshake



Improve efficiency, privacy, reliability

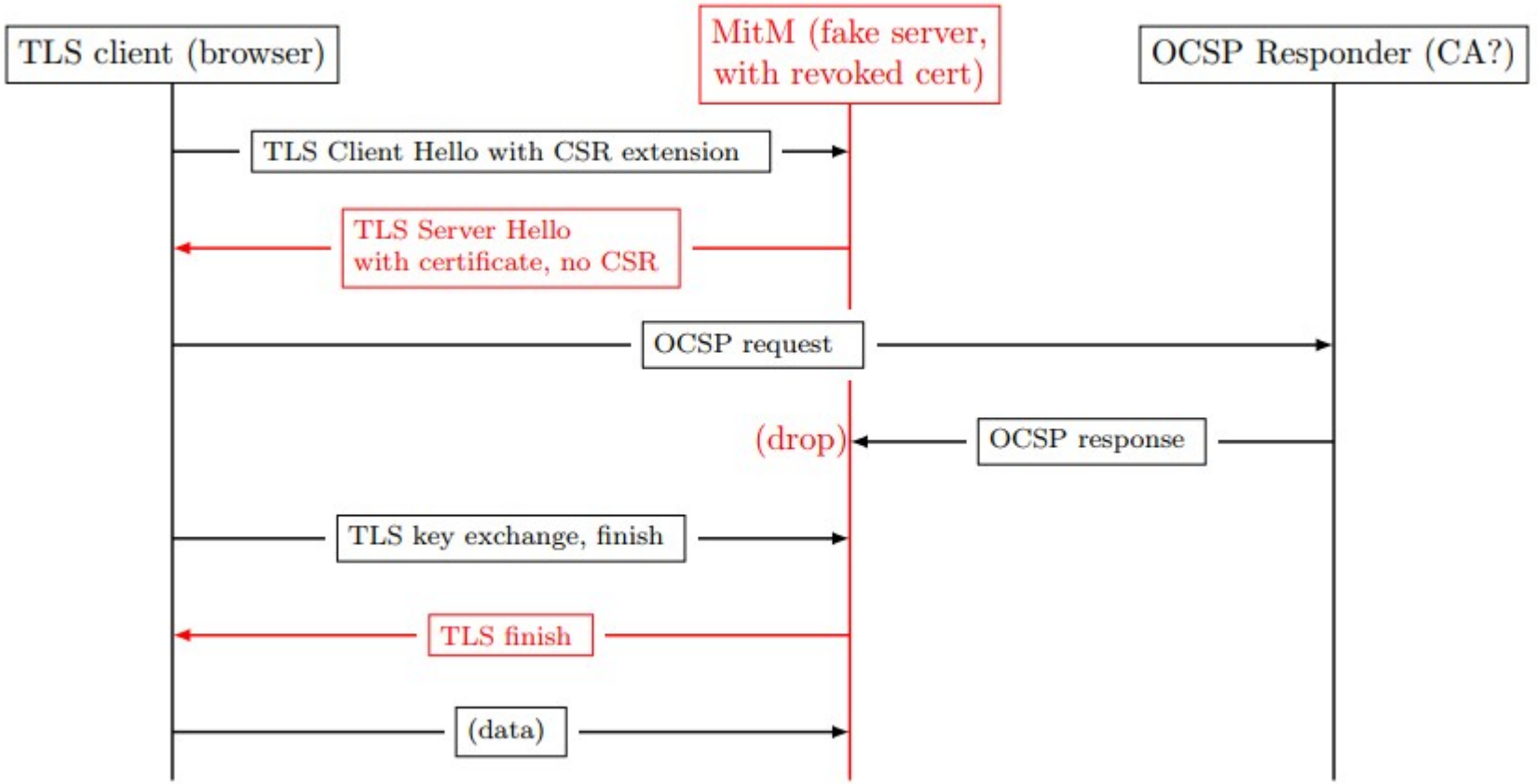
Challenge: many servers don't staple!
Or: staple `sometimes/usually'

Solution: `must staple' Cert extension
Mark non-critical (Why??)

Attacking… Stapled OCSP

- Stapled-OCSP is usually vulnerable!
- Idea of vulnerability:
  - Most servers don’t staple
  - And some staple… but not always
  - How can browser know if staple or not??
- Present attack on (typical) browser deployment of OCSP-stapling

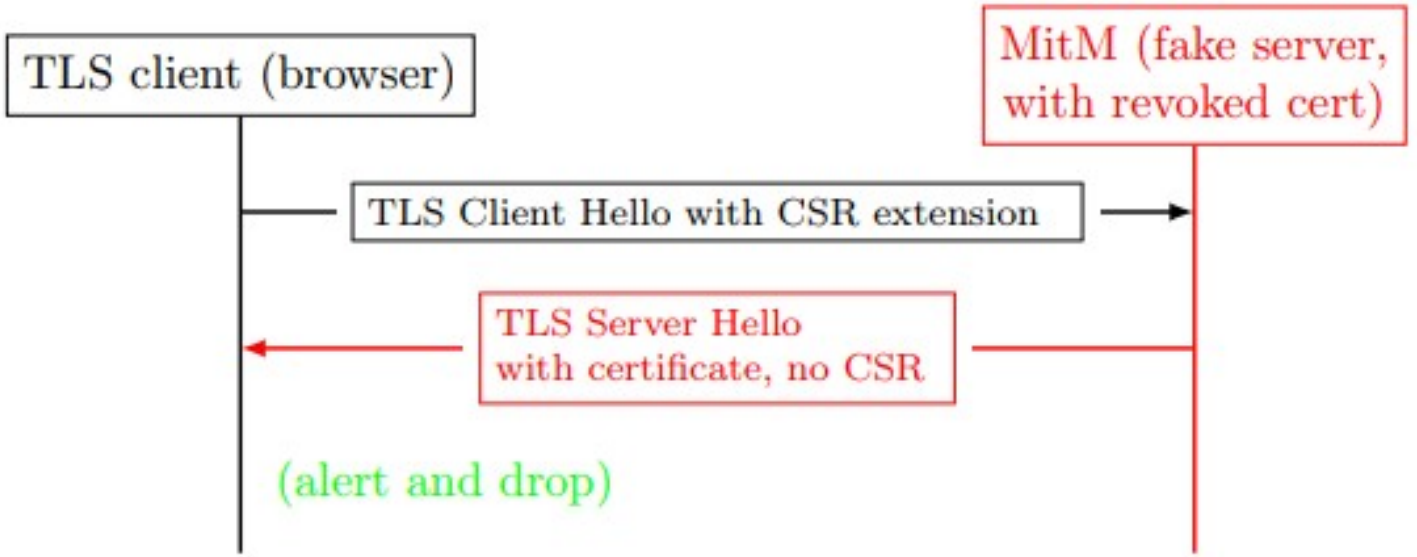
Attacking typical stapled-OCSP client



Must-Staple Certificate Extension

- Solution to attack on stapled-OCSP clients: site signals it **supports stapled OCSP**
- **How?** Add ‘must staple’ x.509 extension
  - A server sending this cert must staple!
- Note: extensions should be non-critical!
  - Else, a non-supporting client cannot use cert.
- How does this prevent attack?

OCSP with Must-Staple Extension



PKI Lecture: Topics

- X.509 Certificates
- Revoking certificates
- **Dealing with CA failures:**
  CT (Certificate Transparency) and defenses
- Conclusions and challenges…

Why and How CAs fail?

- Many CAs `trusted' in browsers (as root)
- 'Domain-Validated' certificates
- Several well-known failures
  - DigiNotar, Comodo, Stuxnet, …
- Every CA can certify any domain (name)
  - Naming constraints NOT applied (esp. to roots)
  - Equivocation: two certificates for same name (from same or different CAs)
  - To detect bad-CA: must find bad-certificate
    - Need: public, auditable log of certificates

Defenses against CA failures

- **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!
- Pre-load lists of (`important’) revocations
  - Chrome (`CRLset’), FF (`OneCRL’)
- Certificate Transparency (CT): **Accountability**

Certificate Transparency (CT)

- X.509, PKIX: CAs sign certificate
  - Given rogue certificates, we can know which CA issued it
    - A rogue certificate: same or too-similar name to that of some victim
    - Used to impersonate website, site malware, etc.
  - Then: ask CA to revoke it, or **stop trusting non-trustworthy CA**
  - All this – only **after** rogue certificate is detected !
- CT: early detection of rogue certificates
  - Soon after issued – possibly even before use!
  - Also: detect **at what time** the rogue cert was issued
    - Helps forensics, prevents rogue-CA from 'backdating' a certificate
  - Proposed by Google, after Chrome detected rogue Google cert
- Main idea and service: Certificate loggers
  - Maintain public log of all certificates issued (by subscribing CAs)
  - To begin, let's assume **trustworthy logger**

[RFC6962]

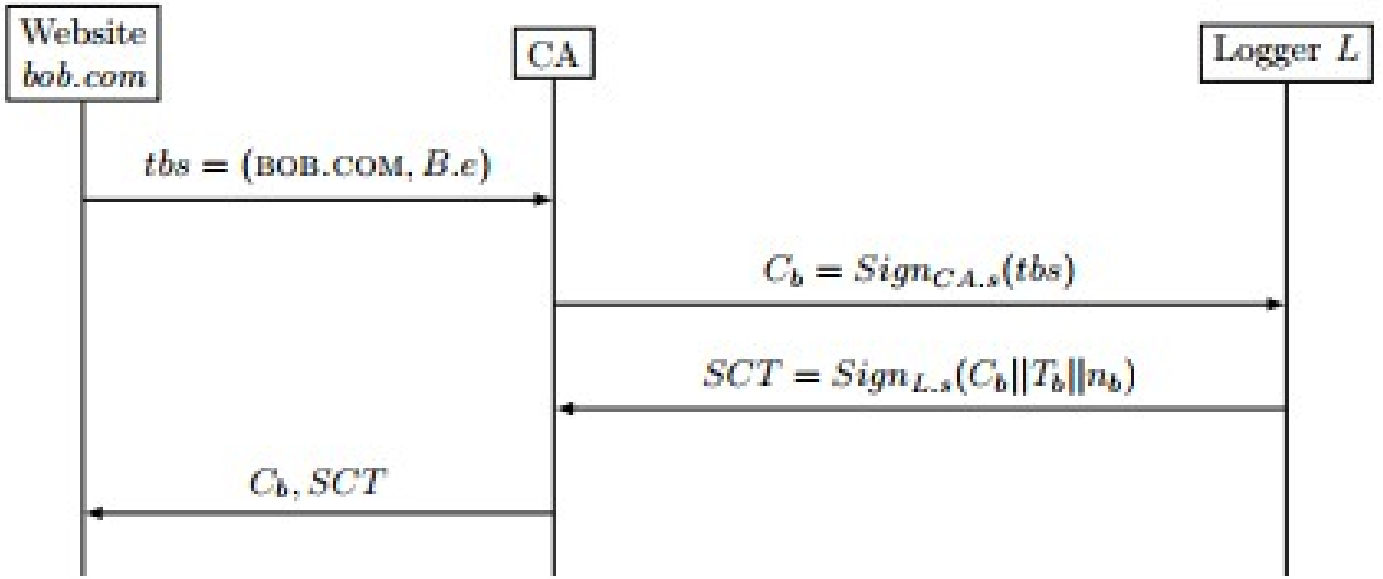
Simplified CT: Assume Trusted Logger

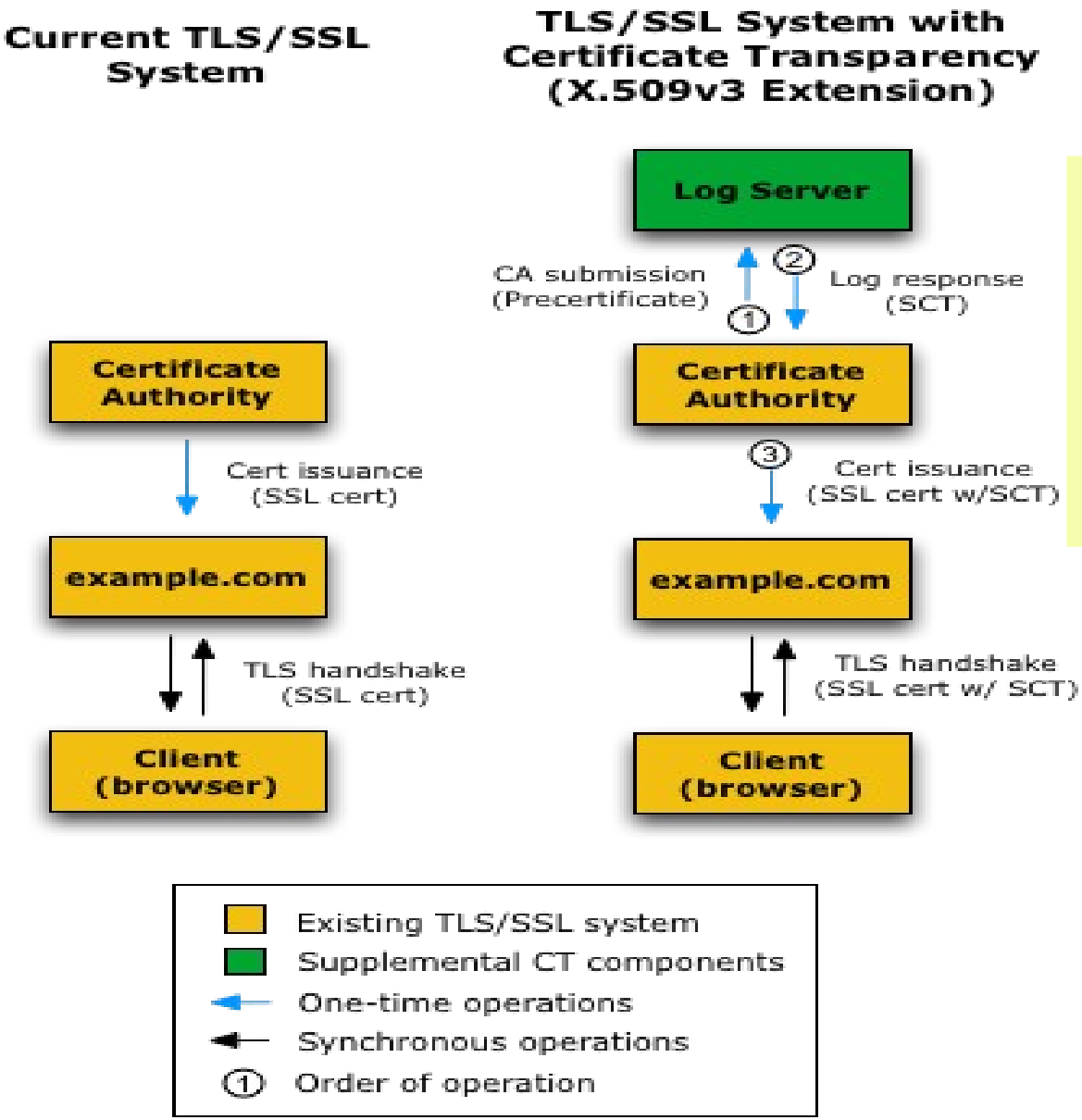
- Goal: early detection of rogue certs
  - Rogue cert: same or ‘similar’ to ‘owned’ name
- Simplified CT solution:
  - Issuer (CA) must send every cert to logger
  - Logger returns Signed Certificate Timestamp (SCT)
    - Validate that the cert was logged at given time
  - CA gives cert, SCT to subject
  - Subject sends SCT (with cert) to relying party
  - Relying party ‘knows’ cert was logged (and when)

Issuing a CT Certificate

- Subject, e.g. website, send request
  - Request contains ‘To Be Signed’ fields: name, public-key
- CA validates request, issues `regular’ cert
- Logger adds to log, signs and returns (signed) SCT
- CA sends cert + SCT to subject (e.g., website)

TBS=To Be Signed





SCT: Signed Certificate Timestamp (time that the certificate was added to log, serial number)

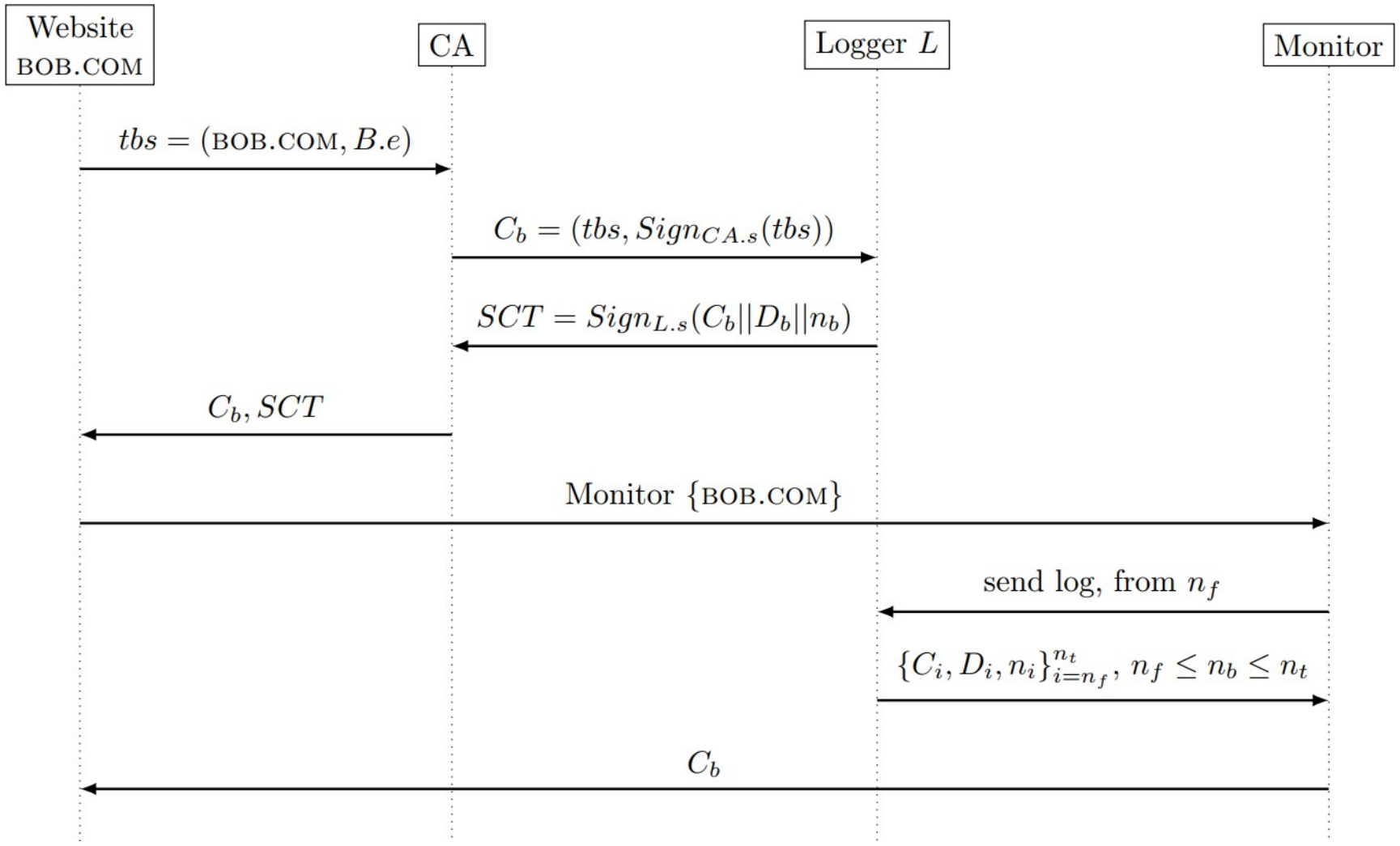
Simplified CT: Assume Trusted Logger

- Goal: early detection of rogue certs
  - Rogue cert: same or ‘similar’ to ‘owned’ name
- Simplified CT solution:
  - Issuer (CA) must send every cert to logger
  - Logger returns Signed Certificate Timestamp (SCT)
    - Validate that the cert was logged at given time
  - CA gives cert, SCT to subject
  - Subject sends SCT (with cert) to relying party
  - Relying party ‘knows’ cert was logged (and when)
- How do we use logs to detect rogue certs?

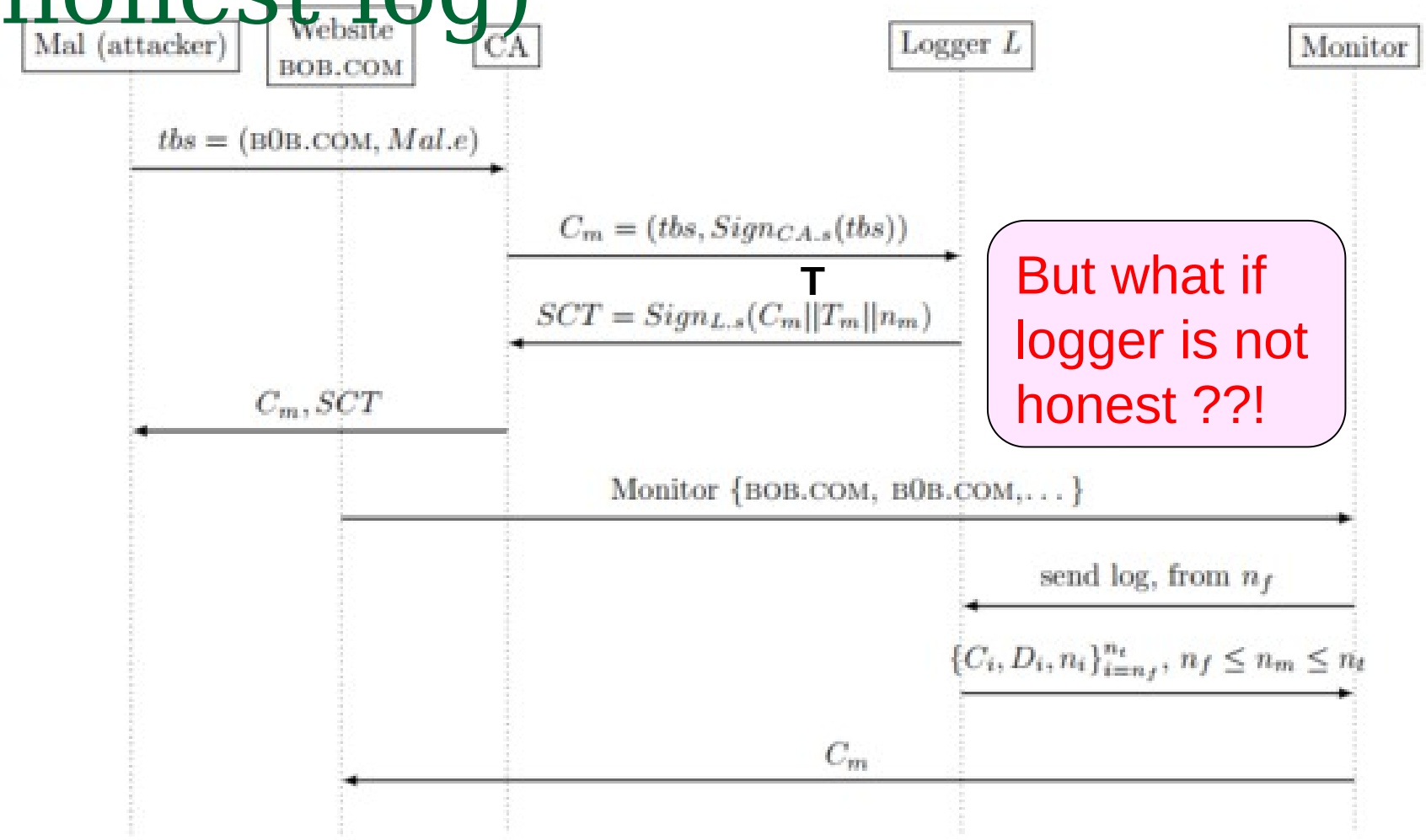
Detecting rogue certs in log: Monitors

- Goal: early detection of rogue certs **in log**
- Logs should be publicly available
- Name-owners can monitor the log
  - Download, check log for relevant names
  - □ high overhead!
- Instead: **monitors** do this (for many names)
  - Several such monitors, loggers already operate
  - Download only new certificates
    - And: ask log for seq# and/or date of last logged cert
    - Ask long to send range of certs: <from-to>
    - Optionally: maintain all certs (to check new names)

Monitoring a Log (trusted



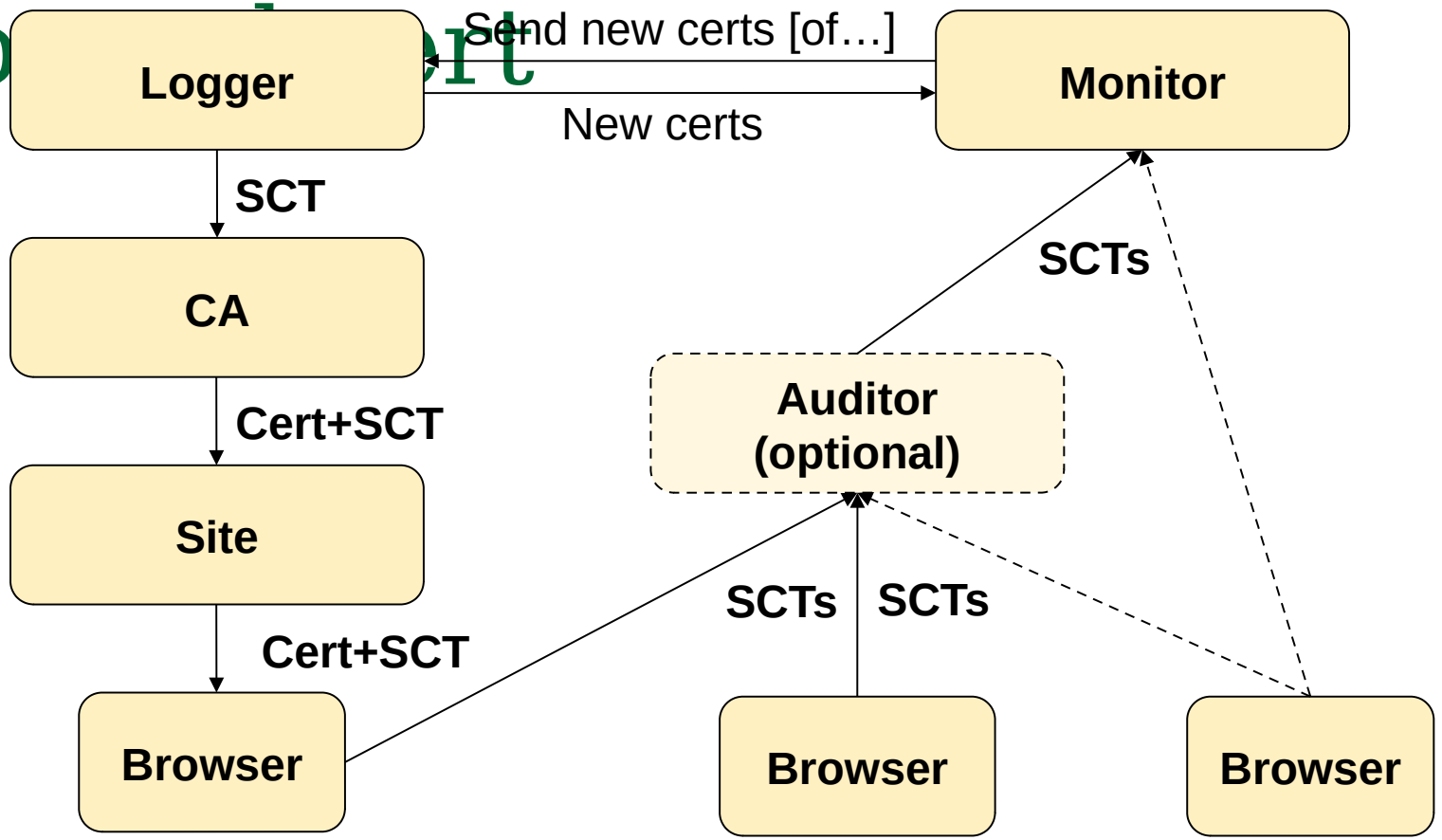
Detecting bad certificate (honest log)



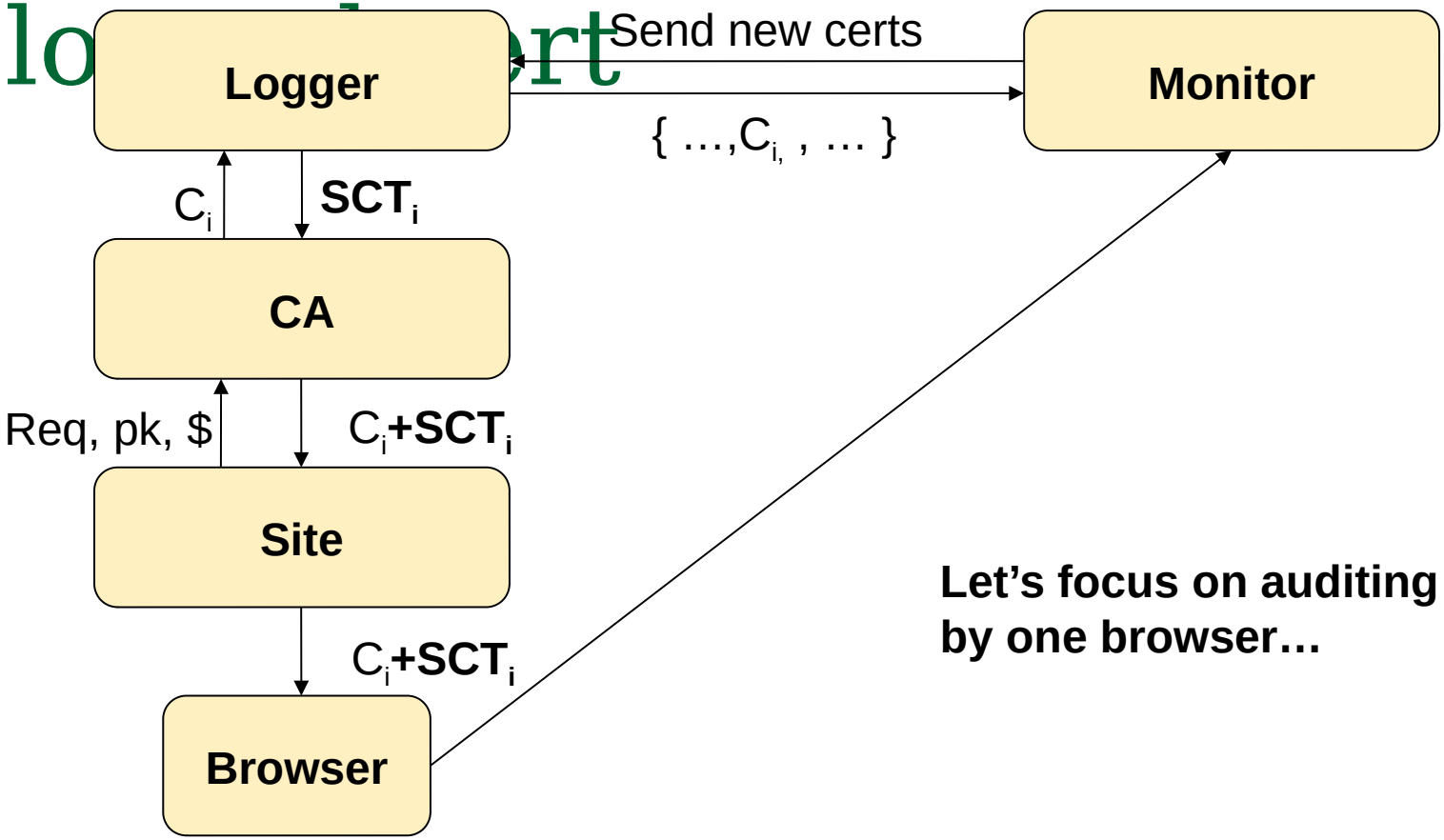
CT: Detecting Rogue Loggers

- Can we detect a rogue logger??
  - No; instead, send SCTs signed by few loggers
    - Overhead
    - How many and which loggers?
      - Current approach in Google's Chrome
      - Requires two SCTs, one of them by Google's logger
        - 'In Google we Trust' ?
  - **Yes: Audit to detect rogue loggers**
    - Relying-parties (clients) validate SCT is 'auditable'
      - Using 'stapled' Proof-of-Inclusion
    - And [Randomly? Always?] audit the SCT
      - Using (a trusted) monitor

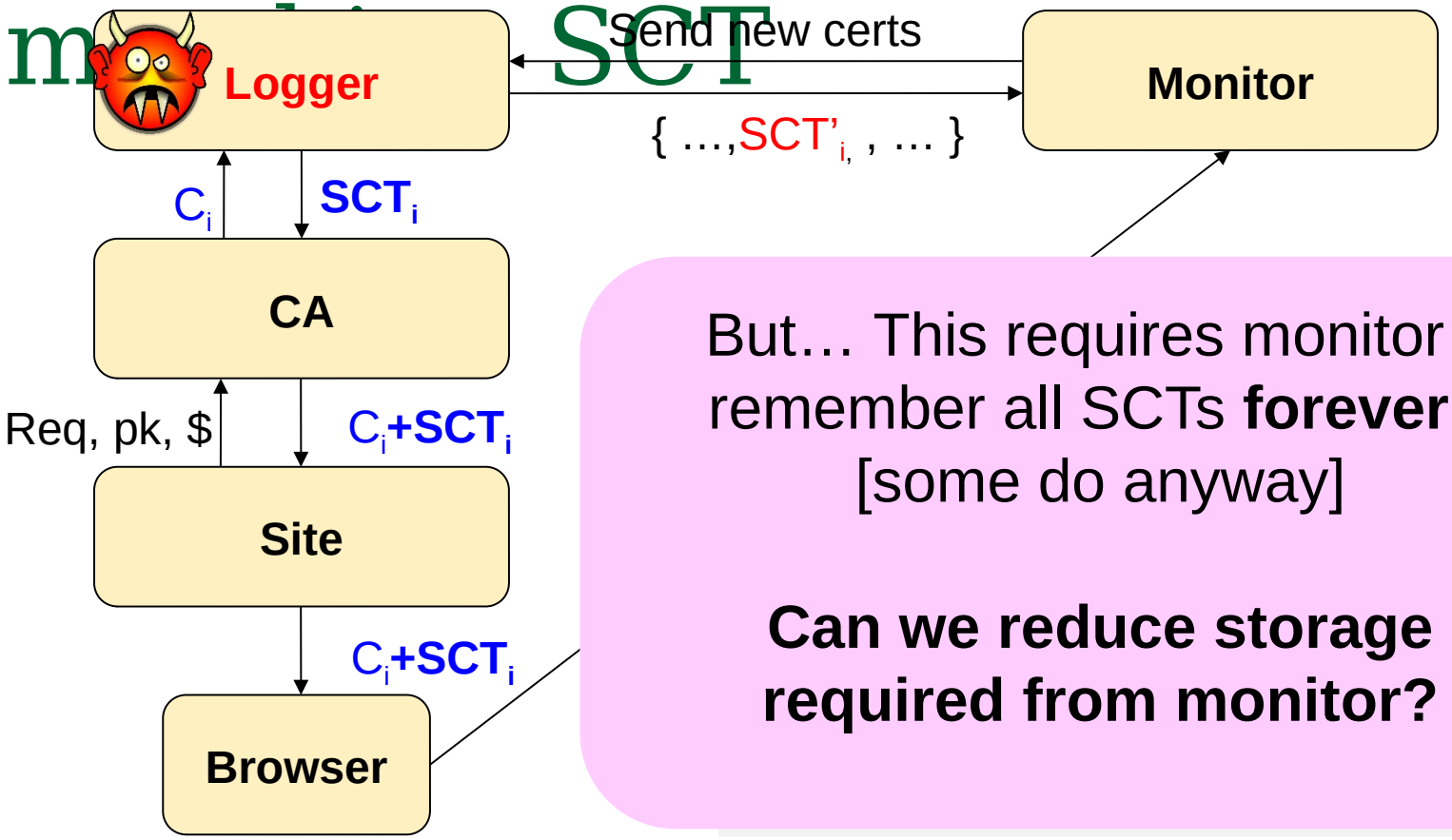
CT: Audit to detect non-log cert



CT: Audit to detect non-log[illegible]ert



Detect and 'prove' non-m… SCT



But… This requires monitor to remember all SCTs **forever**… [some do anyway]

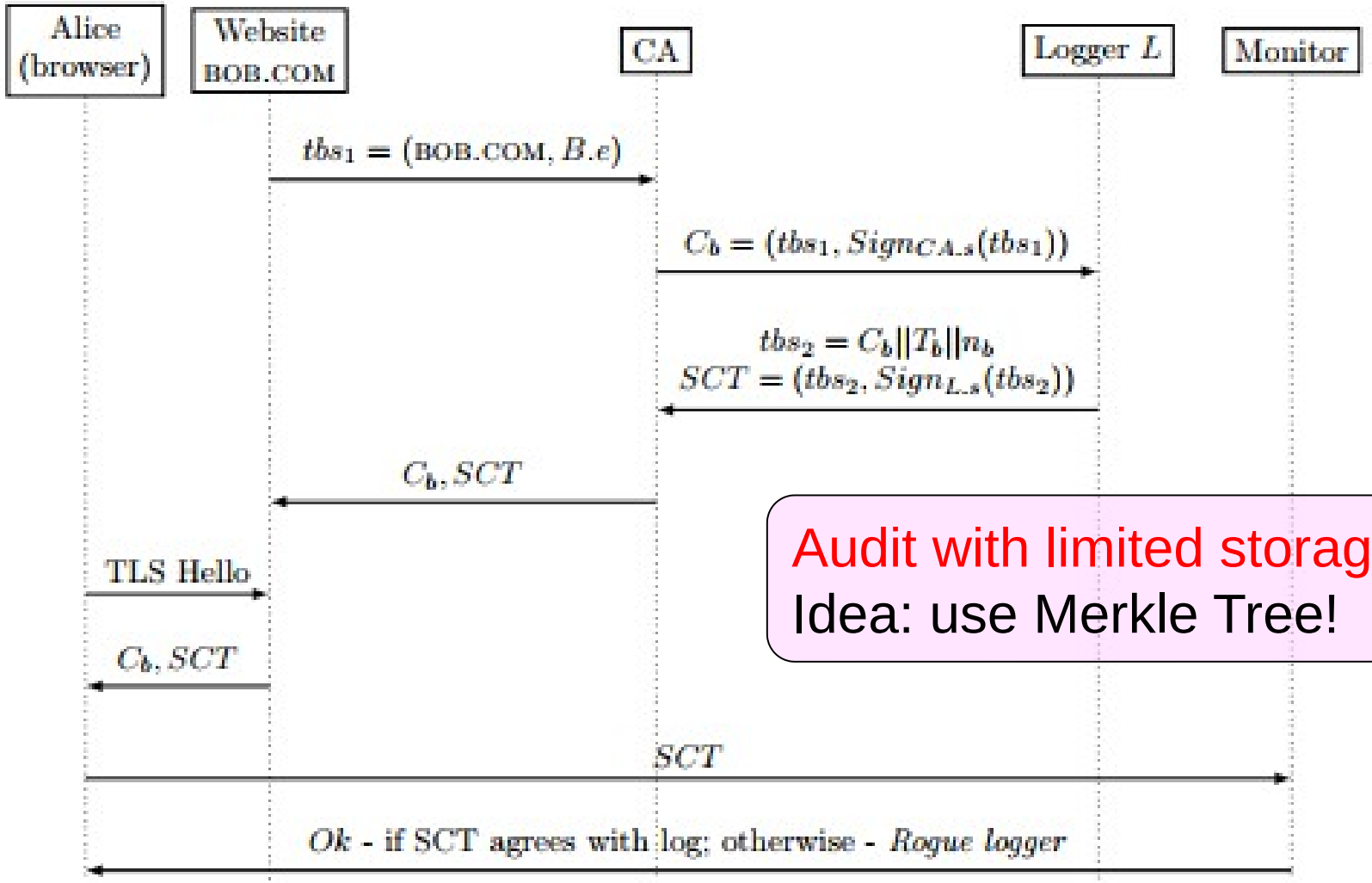
Can we reduce storage required from monitor?

$SCT'_i = Sign_{Log.s}(i,time,C'_i)$

$SCT_i = Sign_{Log.s}(i,time,C_i)$

Logger signed both SCT_i and SCT'_i , 'proving' its 'guilt'

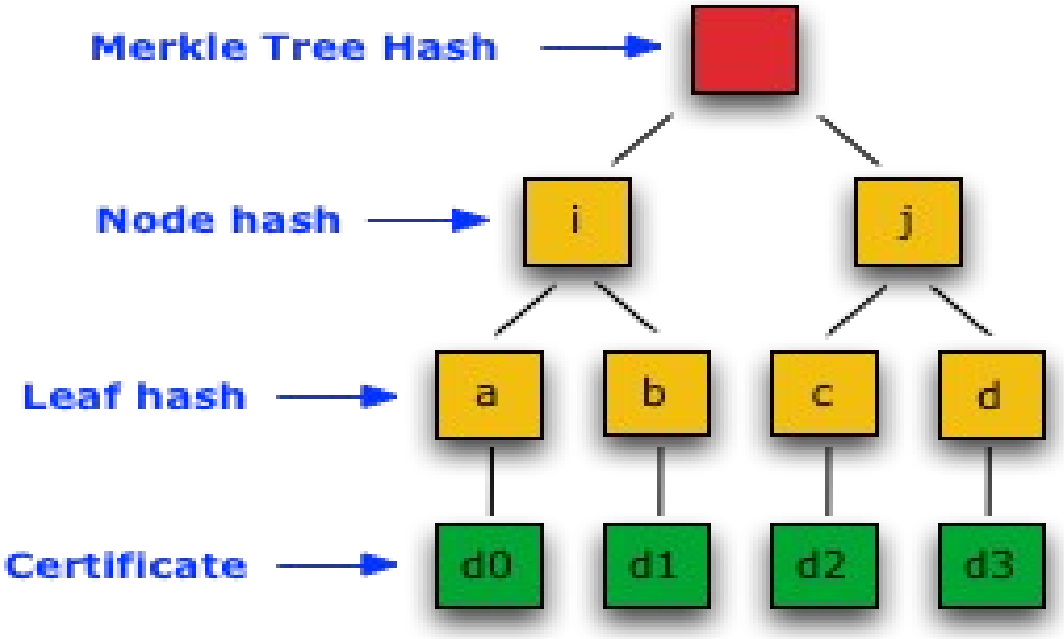
Auditing, simplified (monitor



Audit with limited storage?
Idea: use Merkle Tree!

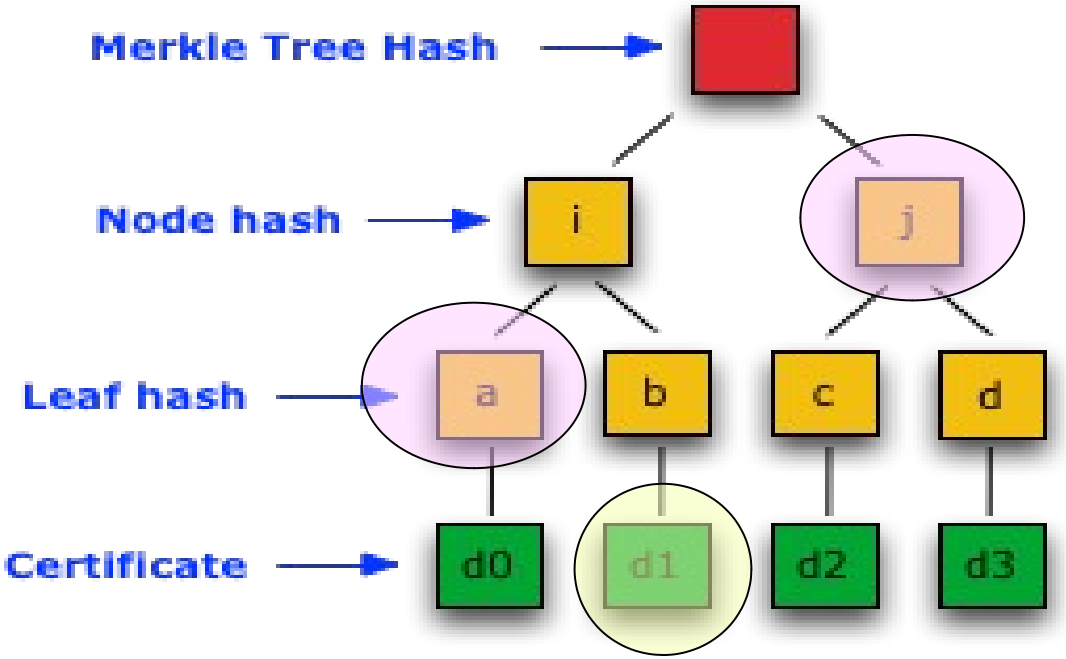
Limiting Monitor Storage: Merkle Tree

- Logs use Merkle hash Tree, send **Signed Tree Hash (STH)**
- Every node is hash of its children nodes.
- d0, d1, d2, d3 – certificates.

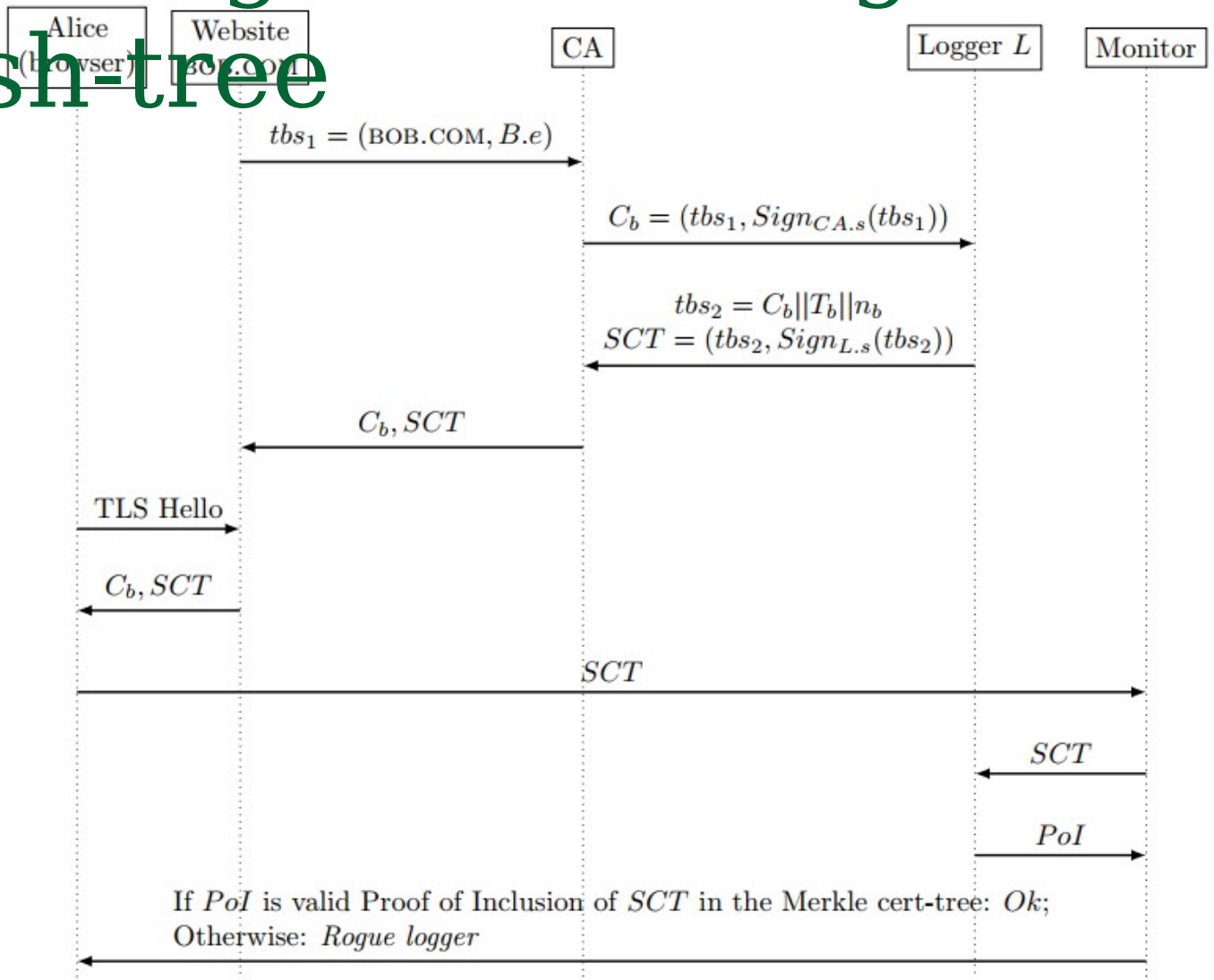


Limiting Monitor Storage: Merkle Tree

- Logs use Merkle hash Tree, send **Signed Tree Hash (STH)**
- Monitor has STH (from logger)
- Monitor can ask for **Proof of Inclusion (PoI)**
- **PoI for d1: {a,j}**



Auditing in CT, using Merkle hash-tree

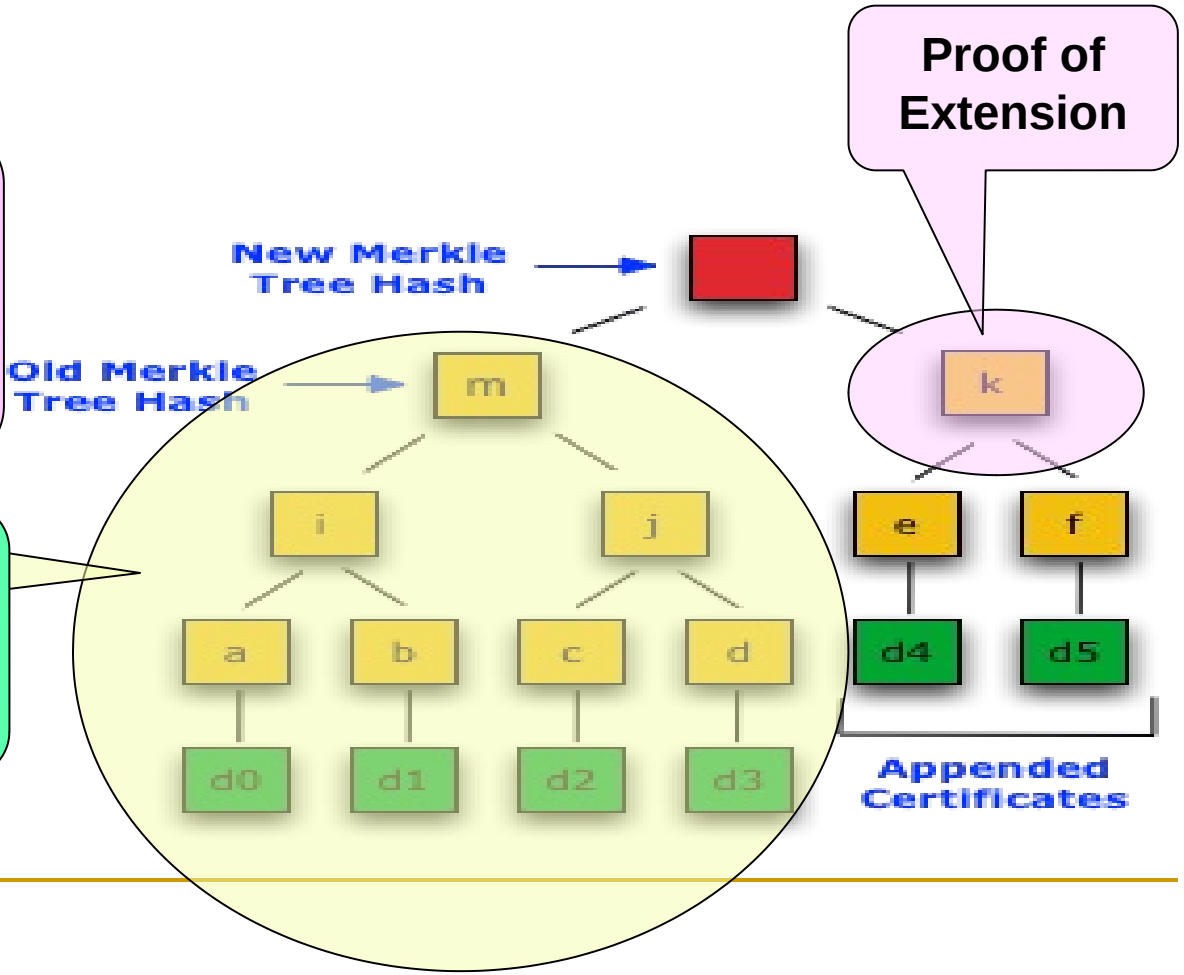


Monitors may store only tree head!

- Tree-head signed by logger
- Logger sends new tree-head, with **proof of extension** of old tree
- Append only

Risk: rogue logger sends different tree to different monitors
Must we check all monitors?

No; monitors gossip detect such rogue logger (details omitted… even from spec!)



So what are the benefits of CT?

- Benefits for websites:
  - Detect rogue certs, for domain
    - Or for `misleading' domain, often abused for phishing
  - Once detected, owners can mitigate risk
    - Demand revocation, removal of CA from browsers,…
- Benefit to users: less likely to fall victim…
- Benefit to trustworthy CAs: reduced competition from shady CAs
- Overall: more secure PKI !
- Concerns:
  - Not yet fully detecting rogue CAs [e.g., gossip not yet fixed]
  - Revocation is not transparent
  - Audit: concerns about overhead and privacy

PKI : Conclusions and Challenges

- Many PKI challenges remain…
- Revocation is still quite broken
  - Although OCSP-stapling is great step forward
  - E.g., incorrect ‘status Ok’ OCSP response from rogue CA
- Privacy: OCSP, CT-auditing exposes visited sites!
  - Must-Staple may be a solution (for OCSP; and CT too??)
  - Many other privacy concerns…
- Define, analyze PKI properties
- Client certificates…
- □ PKI still very active area of research