CSE 3400 - Introduction to Computer & Network Security
(aka: Introduction to Cybersecurity)

Lecture 13
Public Key Infrastructure – Part II

Ghada Almashaqbeh
UConn

From Textbook Slides by Prof. Amir Herzberg
UConn

Outline

- Certificate revocation.
- Dealing with CA failures.

Certificate Revocation

Certificate Revocation

- Reasons for revoking certificates
  - Security issues:
    - Key compromise, CA compromise
  - Administrative issues:
    - Affiliation changed (changing DN or other attribute), public key has been replaced, subject has ceased operation (company dissolving).
- How to inform relying parties? Few options usually under three categories:
  - Prefetch: have revocation info in advance.
  - As-needed: ask for this info when a receiving a certificate and want to validate.
  - Neither: does not fall under any of the above, usually called network-assisted techniques.

Certificate Revocation Techniques

- Prefetch:
  - Cons: higher storage and communication overhead,
  - Pros: lower response delay
- As needed:
  - Cons: higher response delays, reliability issues, privacy concerns.
  - Pros: lower storage and communication overhead
- We will start with studying two techniques:
  - Distribute ***Certificate Revocation List (CRL) --*** *Prefetch*
    - This is part of the X.509 standard.
  - Ask - ***Online Certificate Status Protocol (OCSP) –*** *As needed*

CRLs

- A certificate revocation list (CRL) is simply a list of revoked certificates.
- Distributed periodically by CAs.
  - See next slide for its format.
- If CRLs contain all revoked certificates (which did not expire)… it may be huge!
  - Yes, large storage and communication overhead.
- CRLs are not immediate
  - Who is responsible until CRL is distributed?
  - Frequent CRLs → even more overhead!

X.509 CRL Format

Signed fields:

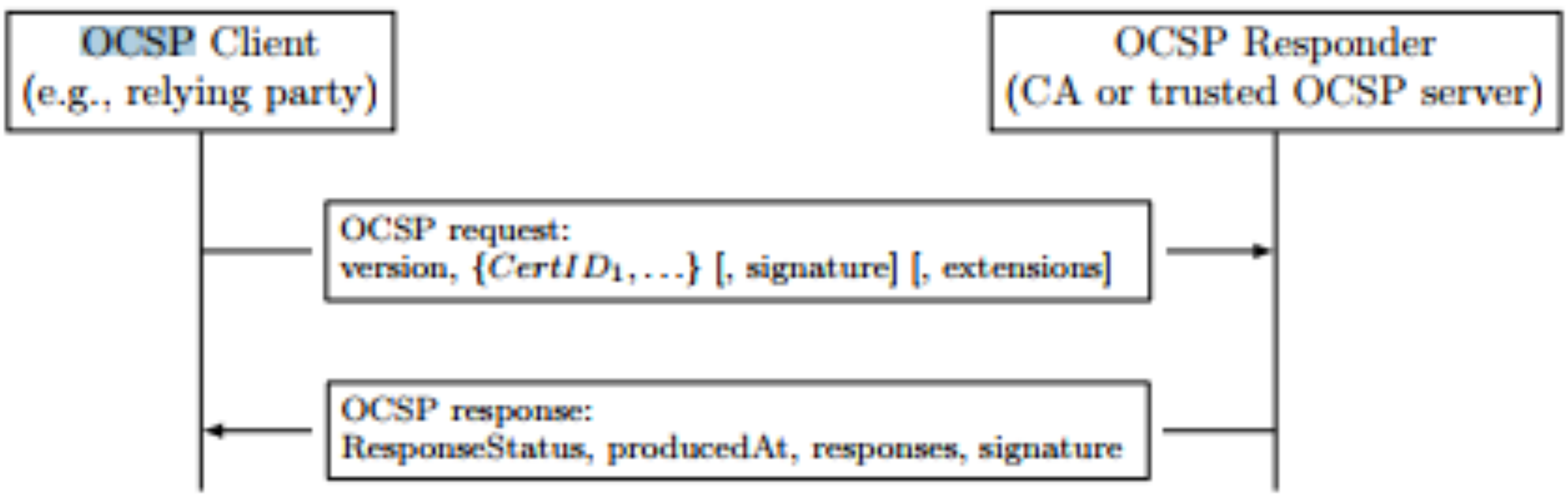
Version of CRL format			
Signature Algorithm Object Identifier (OID)			
CRL Issuer Distinguished Name (DN)			
This update (date/time)			
Next update (date/time) - optional			
Subject (user) Distinguished Name (DN)			
CRL Entry	Certificate Serial Number	Revocation Date	CRL entry extensions
CRL Entry…	Serial…	Date…	extensions
….			
CRL Extensions			
Signature on the above fields			

CRLs Optimization 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`
    - Need to keep CRLs for long period to check deltas → complicates implementation
- Browsers mostly do not check CRLs. Instead they usually use:
  - Online Certificate Status Protocol (OCSP)

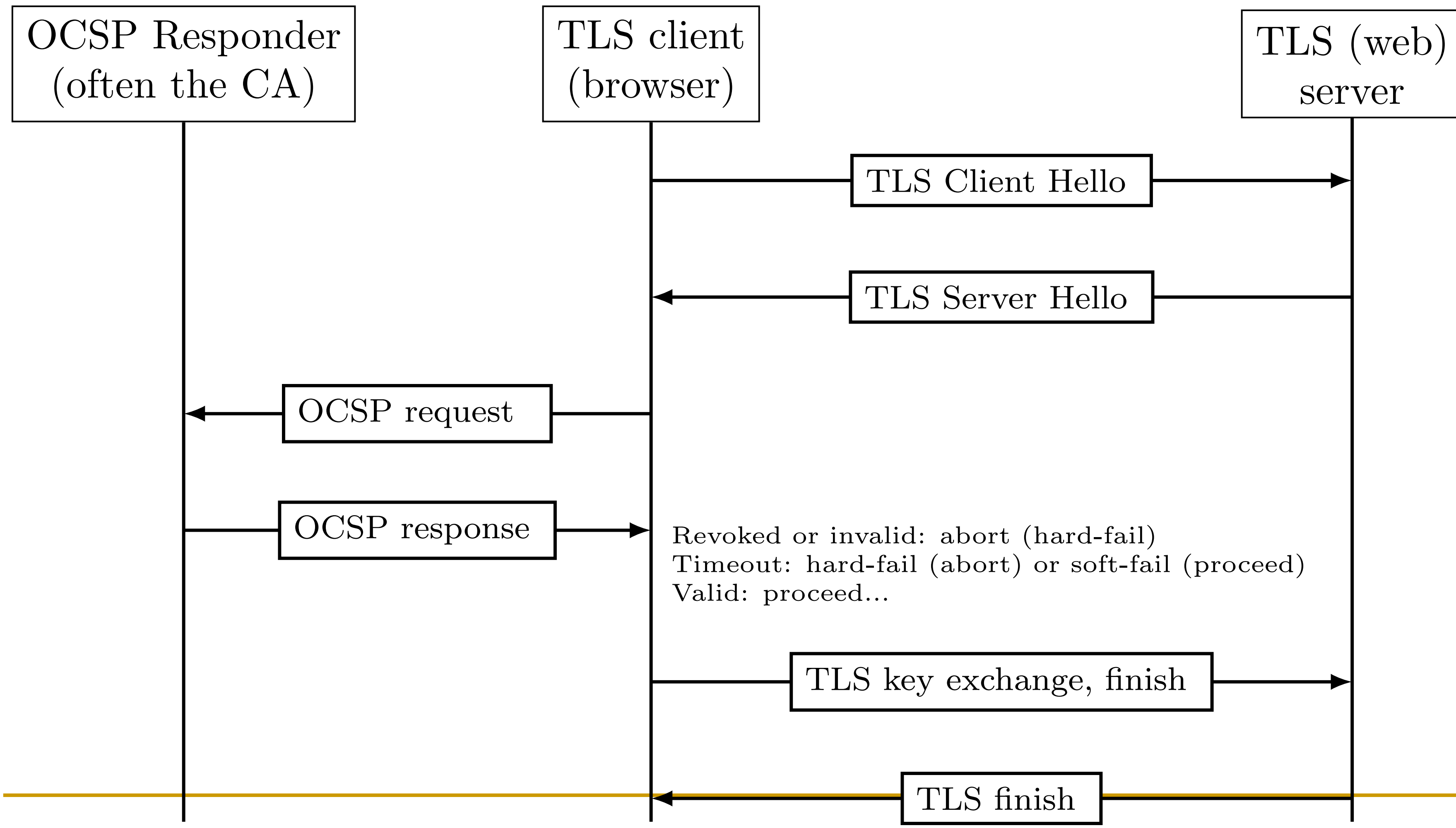
Online Certificate Status Protocol (OCSP)

- Improve efficiency and freshness compared to CRLs
- Client asks CA about cert during handshake
- CA signs response (real-time)



Example - TLS Handshake with OCSP

Classical OCSP – Browser sends requests



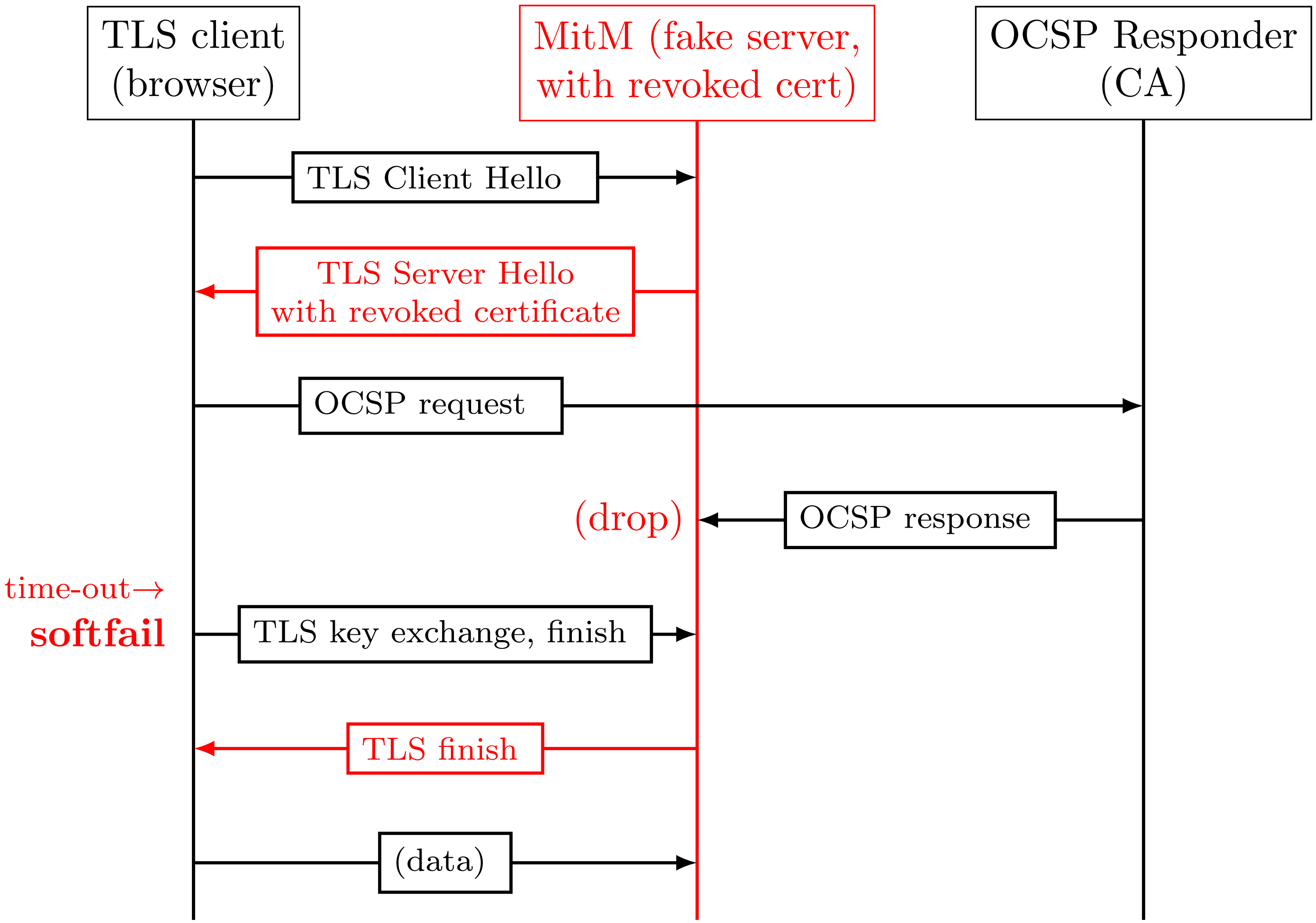
OCSP Challenges

- Cons of as-needed mechanisms mentioned before:
  - Privacy (expose domain and client to CA), load on CA, response delay, reliability (what if CA fails).
- We will elaborate more on:
  - Ambiguity:
    - When an OCSP server (or CA) cannot resolve the request, it replies with ”certificate status is unknown”.
  - Reliability or failed requests.
    - Client failed to establish a connection with the OCSP server.
    - Or client’s request is invalid (not signed, or not authorized).

Ambiguous/Failed OCSP Responses

- What should the client do?
  - Wait forever – unrealistic!
  - Hard-fail: terminate the connection since certificate is unknown.
    - Safe!
  - Ask user: application display a message asking the user how to proceed.
  - soft-fail: pretend that a response has been received and continue as the cert is not revoked.
    - Common choice for browsers!
    - But, a man in the middle attacker who may block the OCSP response to make a revoked cert go through?
    - See next slide.

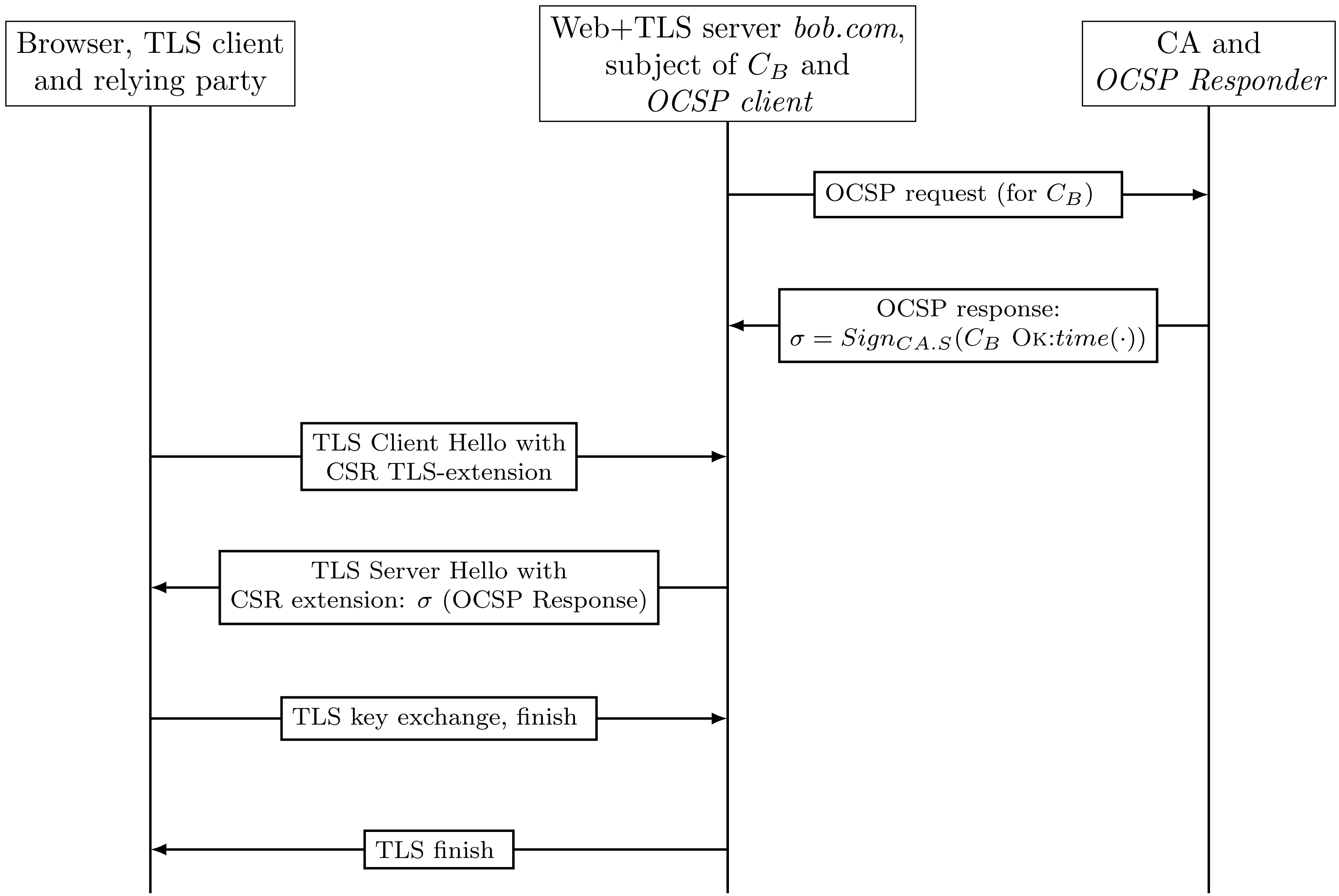
MitM soft-fail attack on ‘classic OCSP’



Classic OCSP is Problematic → Use OCSP-Stapling

- Subject of the certificate (aka web server) sends OCSP requests instead of the relying party (aka browser).
  - So subject acts as the OCSP client.
  - It receives a signed response back (signed by the CA and includes a timestamp) which it forwards to any relying party initiating a connection.
  - Browser accepts if signature is valid and time is recent enough.
- Solves:
  - Privacy – the CA no longer knows about browsers access pattern
  - Reduce load – one request per website rather than many requests from browsers.
  - CA limits OCSP service to subjects – easier to manage.

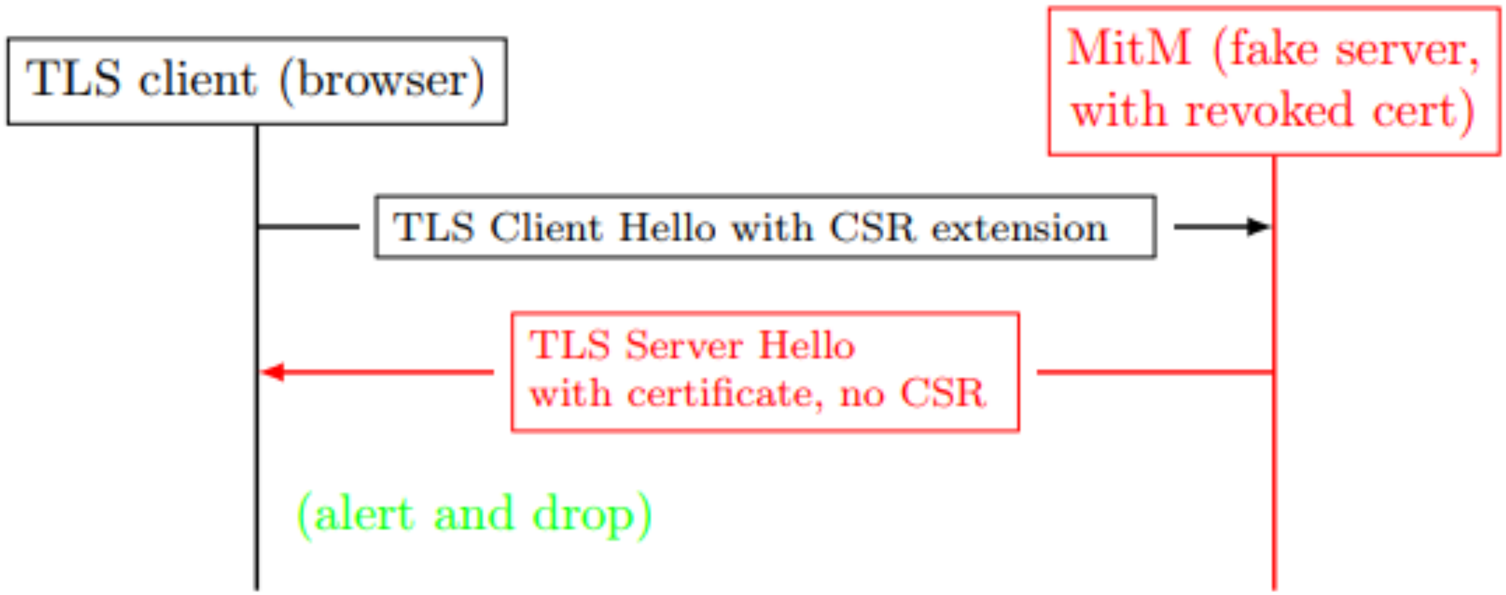
OCSP-Stapling



OCSP-Stapling

- Challenge: many servers don't staple!
  - Or, worse: staple `sometimes/usually'
  - So, try OCSP? Connect anyway? Disconnect?
    - Usually browsers attempt to do classical OCSP then if no response, resort to soft-fail.
    - So we are back to the MitM attack described before.
- Solution: `Must-staple' cert. extension
  - RFC 7633
  - Mark as not critical
    - As it may not be supported by some browsers

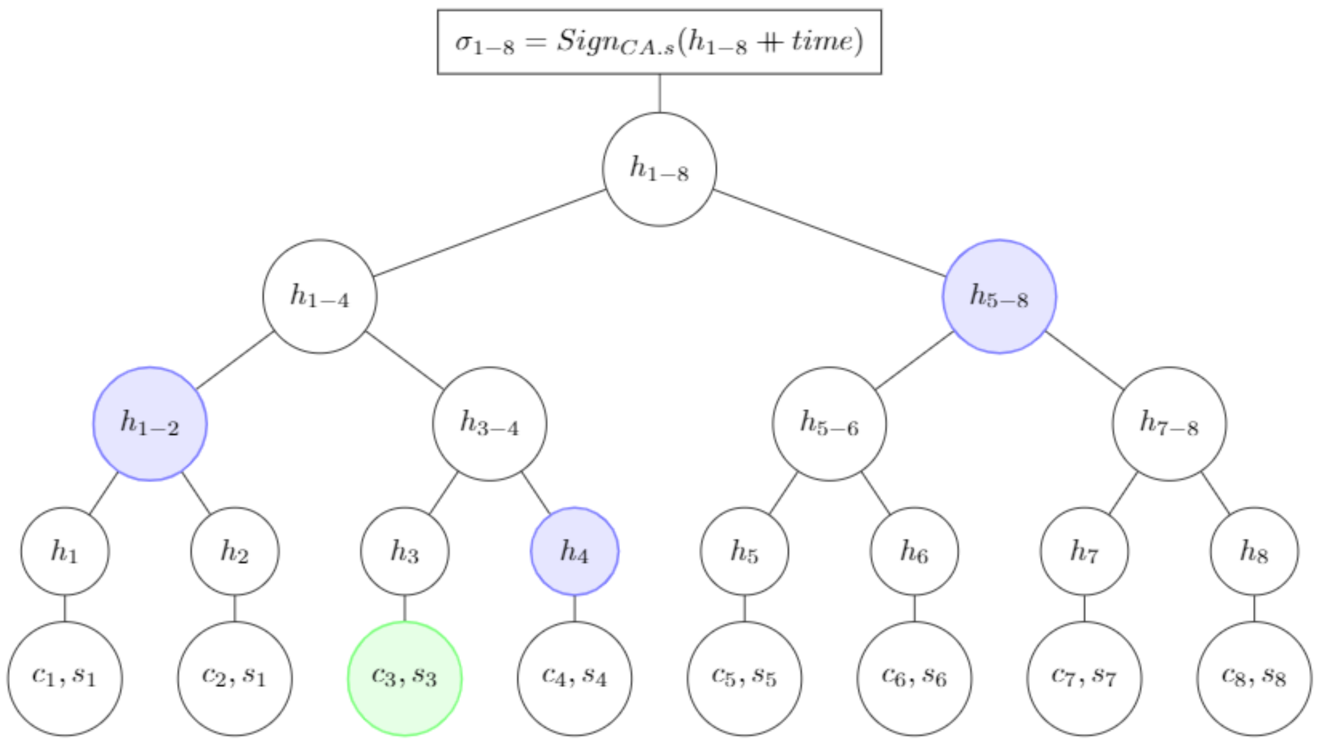
OCSP with Must-Staple Extension



Principle:
Defenses should not be bypassed due to failures.
If defenses are bypassed upon failure, attacker will cause failures to bypass defenses.

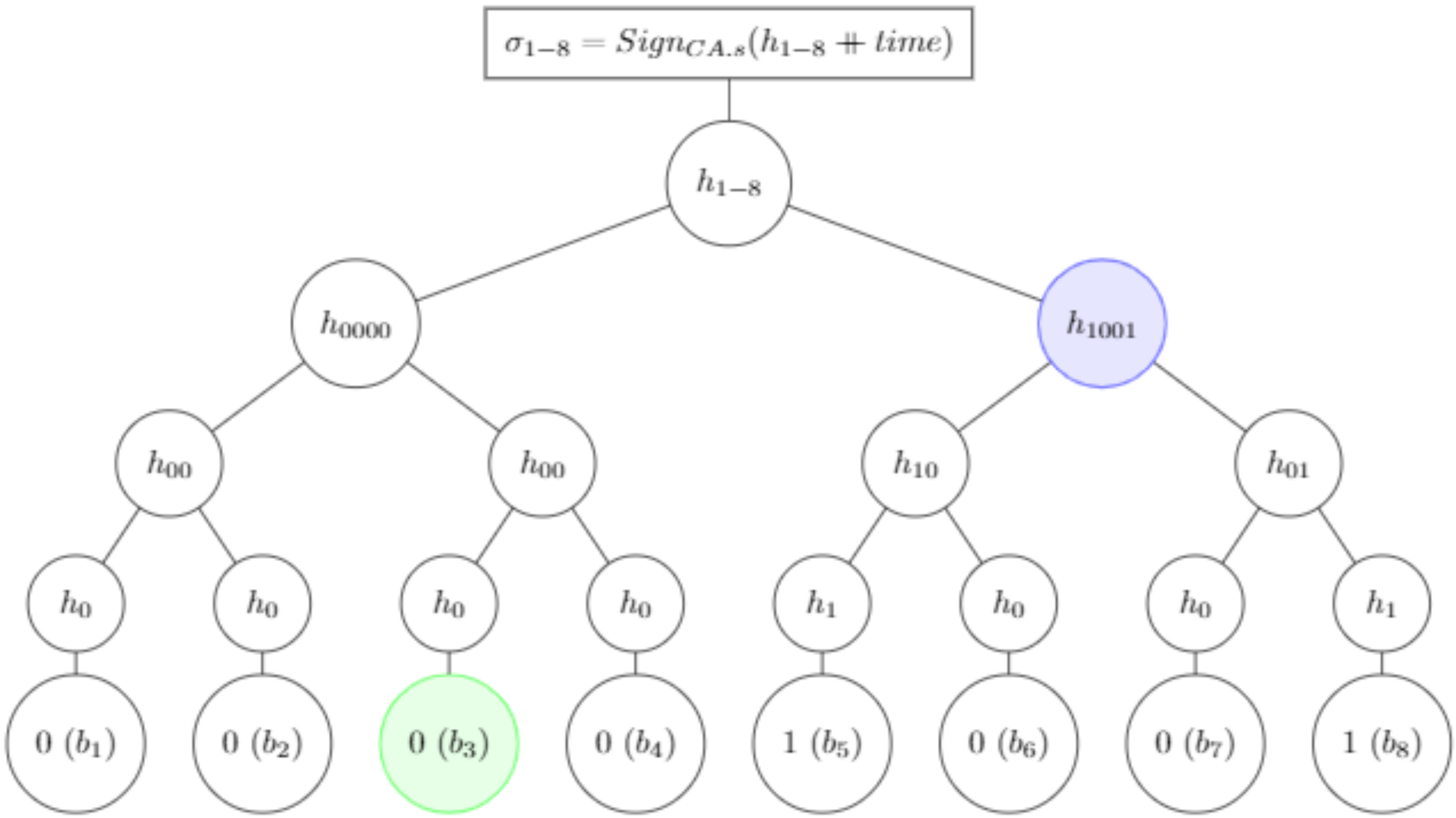
Optimizing OCSP Responses (1)

- OCSP overhead is high – esp. if frequent
- Several optimizations possible, e.g.:
  Merkle digest-tree and Proof-of-Inclusion (PoI).
  - Saves number of needed signatures (recall, public key cryptography is expensive)!



Signed Revocations-Status Merkle-Tree

- A further optimization: send digest and PoI in **revocations-status** Merkle tree:



Dealing with CA Failures

Why and How CAs Fail?

- Many CAs `trusted' in browsers (as root)
- Several well-known failures
  - DigiNotar, Comodo, Stuxnet, …
- Every CA can certify any domain (name)
  - Name constraints NOT applied (esp. to roots)
  - Some CAs may be negligible or even rogue
- Bad certificates:
  - Equivocation: rogue certificates for same name as a legit cert
  - Misleading certificates, e.g., similar name
- Can we improve defenses against bad CA?

Defenses Against CA Failures

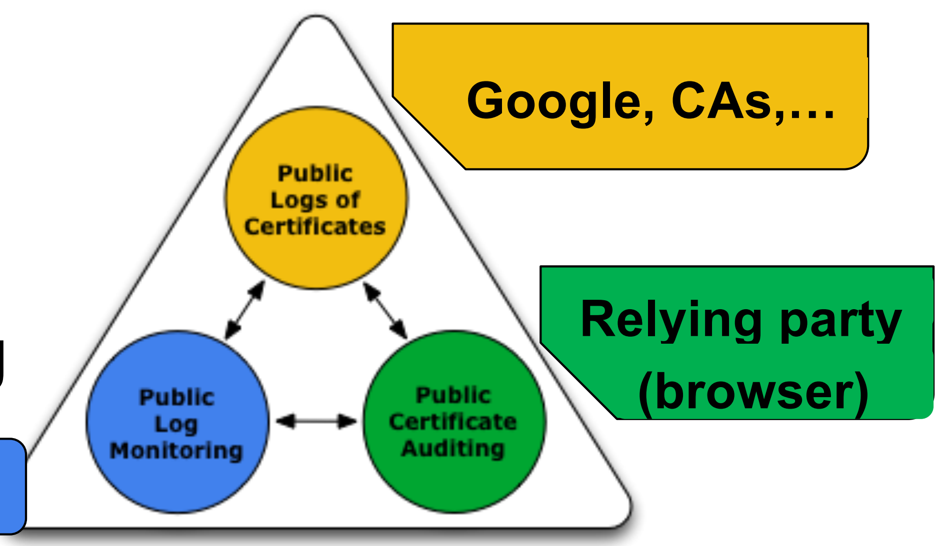
- **Use name constraints to limit risk**
  - who can issue global TLDs (.com, etc.)?
- **Static key pinning:** `burned-in' public keys
  - Detected MitM in Iran: rogue DigiNotar cert of Google
  - Limited: changing keys? Which keys to preload ?
- **Dynamic Pinning: HTTP Public-Key Pinning (HPKP)**
  - Server: I always use this PK / Cert / Chain
  - Client: remember, implement, detect & report attacks
  - Concerns: key loss/exposure, changing keys (recover security)
- Still, **Trust-On-First-Use (TOFU)** can be helpful
  - E.g. for security policies: OCSP-must-stapling or CAs-pinning
- Certificate Transparency (CT): **Accountability**
  - **Public, auditable certificates log**

Certificate Transparency (CT) [RFC6962]

- X.509: CAs sign cert
  - **Accountability**: identify issuer, **given** (rogue) cert
- Challenge: find rogue cert
  - Unrealistic to expect relying parties to detect !
  - Google detected in Iran - since Chrome had pinned Google's PK
- Proposed solution: Certificate Transparency
- Functions: **Logging, Monitoring and Auditing**

Three types of entities:

- **Loggers** provide public logs of certificates
- **Monitors** monitor certificates logged for detection of suspect certificates
- **Auditing** (auditors?): check for misbehaving loggers

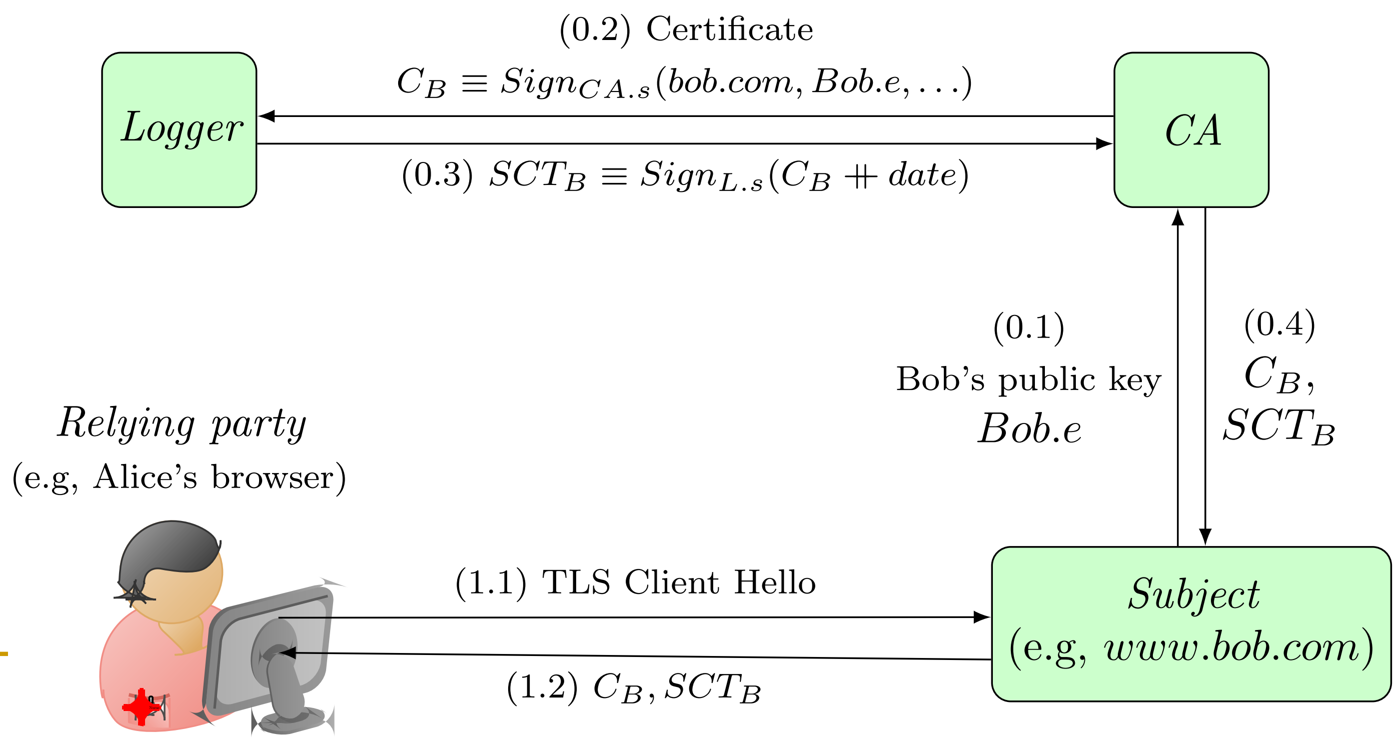


Certificate Transparency (CT): Goals

- → Easier to detect, revoke rogue certificates
- → Easier to detect, dis-trust rogue CAs:
  No (real) accountability without transparency!
- What about rogue loggers ?
- Option 1: Honest-Logger CT (HL-CT) [RFC6962]
  - Assume honest logger [out of two loggers – redundancy]
- Option 2: NTTP-Secure CT (NS-CT):
  - Monitors, relying-parties detect misbehaving loggers
  - Relying party decides which **monitor(s)** it relies on (trusts) !
  - Original CT goal: ‘no trusted third party’

Honest-Logger CT: Issuing Certificate

- Subject, e.g. website, sends request
  - Request contains 'To Be Signed' fields: name, public-key
- CA validates request, signs cert, sends to **logger**
- Logger adds cert to log, signs and returns (signed) **SCT**
- CA sends cert + SCT to subject (e.g., website)



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

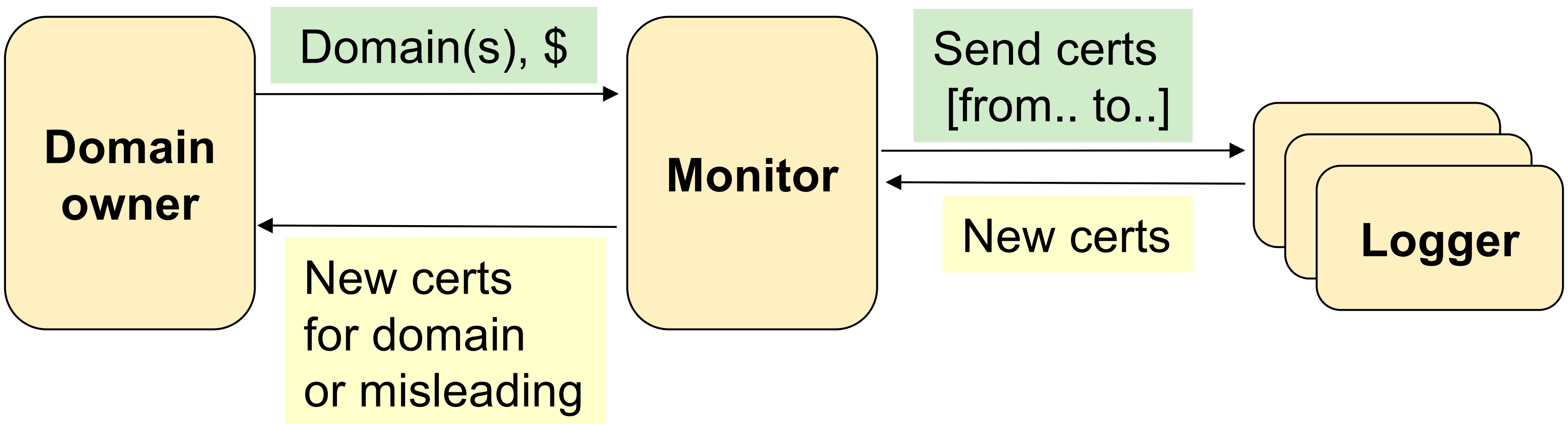
- Download, check log for relevant names
- ☹ high overhead to everyone!

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 log to send range of certs: <from-to>
  - Optionally: maintain all certs (to check new names)

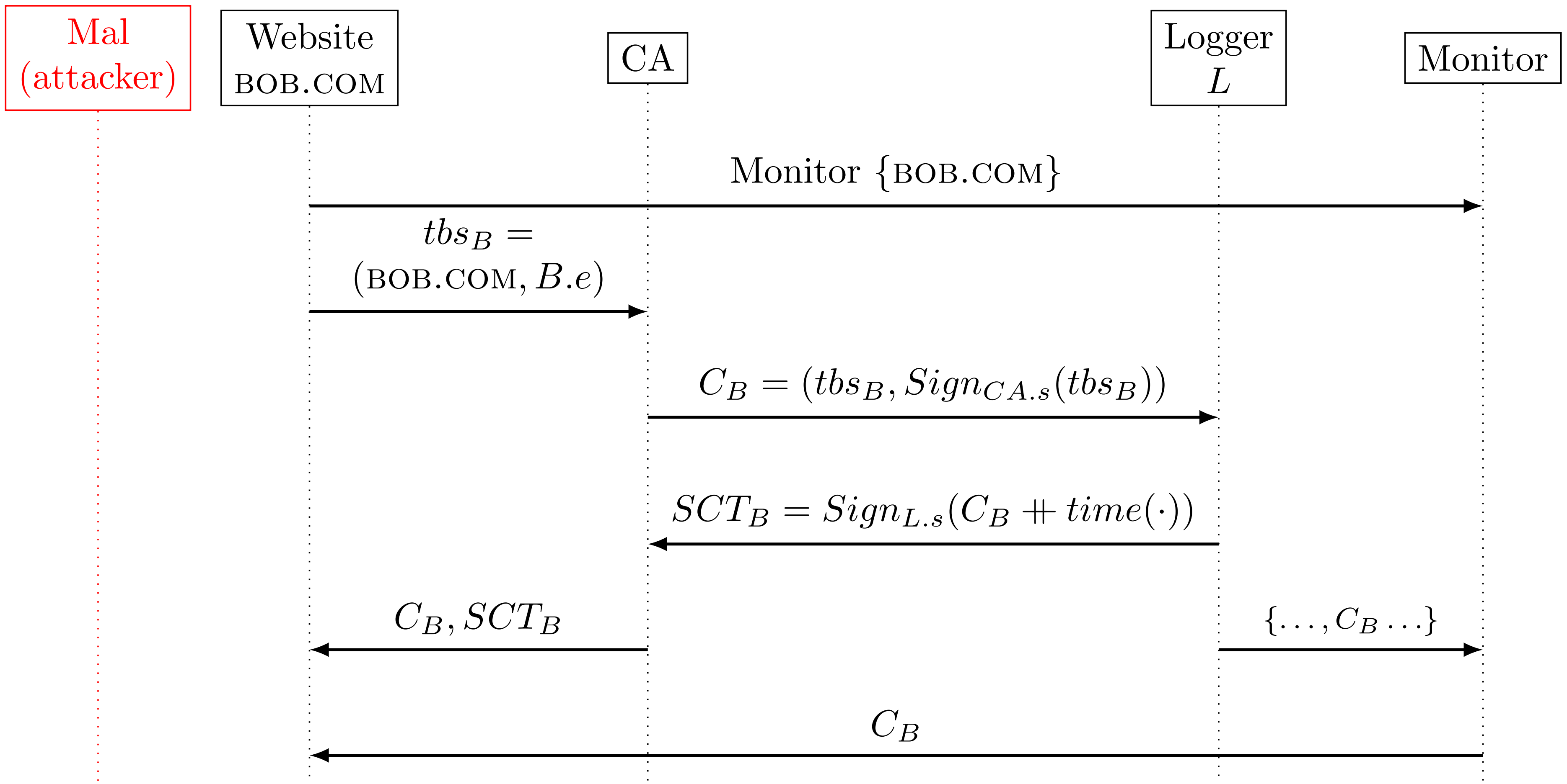
Monitor Detects Rogue Certificates

- Owner asks to monitor relevant domain names

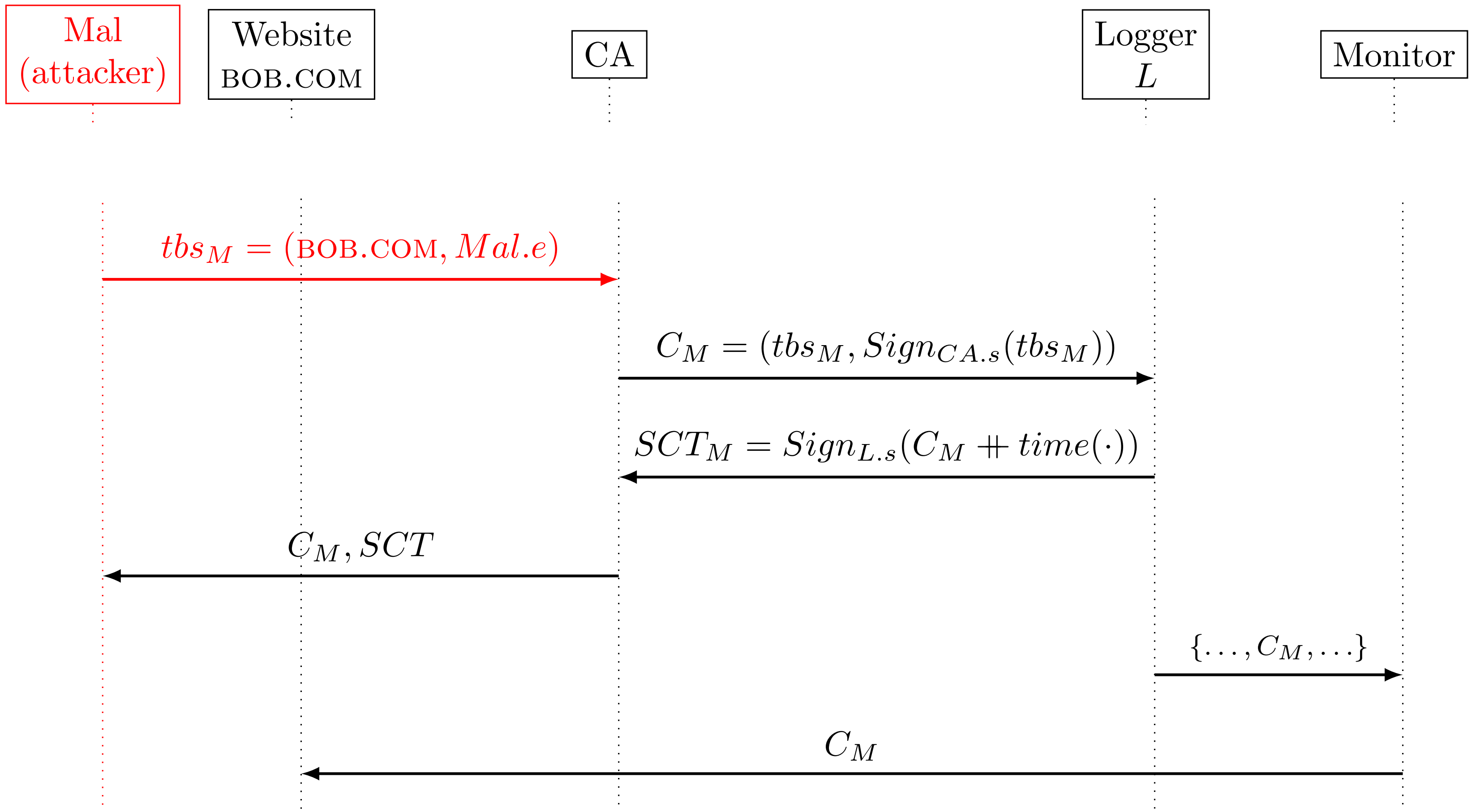


- Monitor asks for certs [Range, e.g., all new]
  - Usually periodically; assume daily (typical)
- Monitor sends to owner new certs for same domain name
  - Or suspect as misleading: combo, homographic, similar,…

Monitoring in Honest-Logger CT

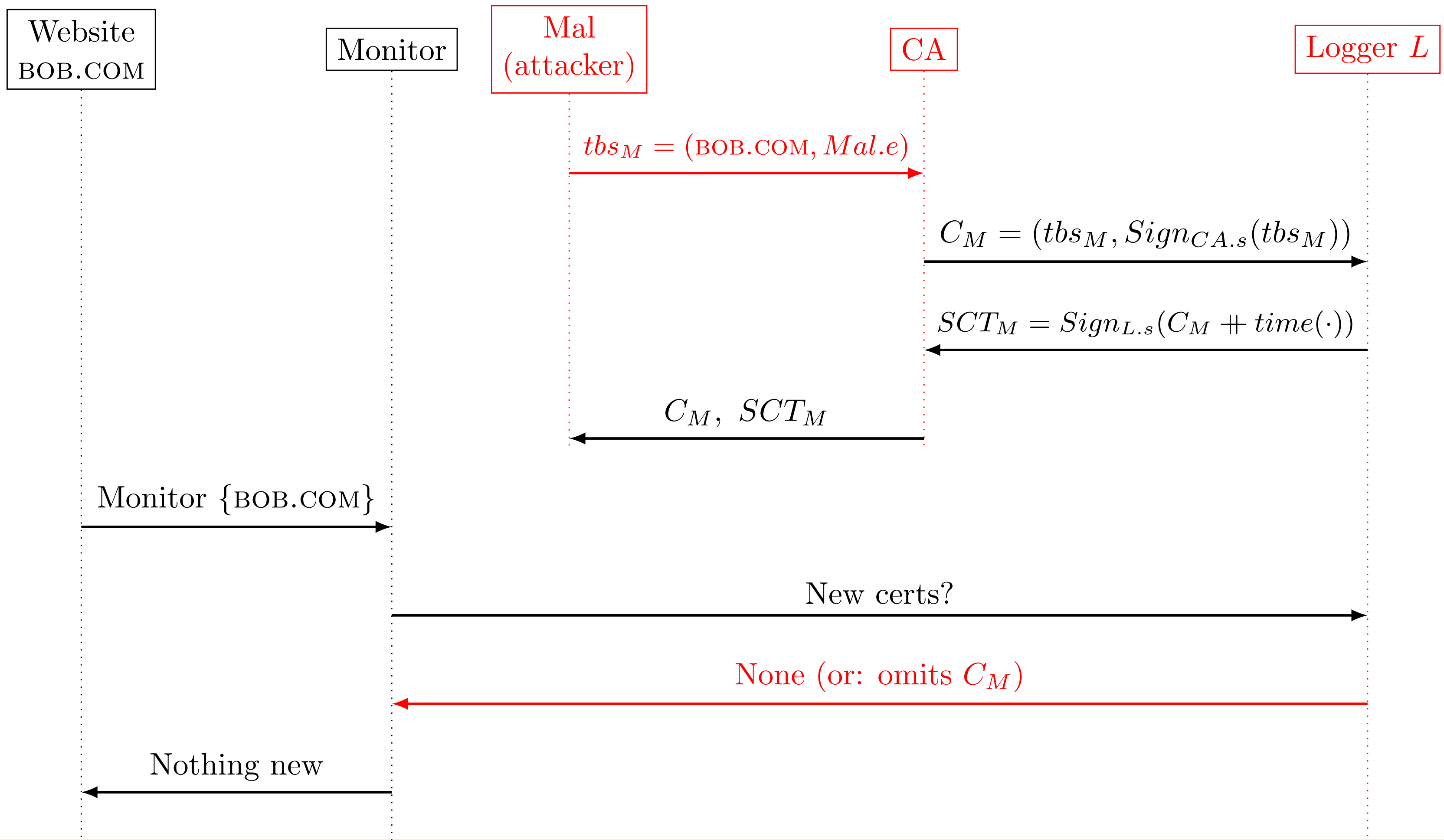


HL-CT: Detecting Rogue Certificate



HL-CT: Omitted-Cert Attack by Rogue Logger

- Collusion of rogue CA and rogue Logger



Security against **Logger-CA Collusion**: two options

- **Option 1: redundancy**: SCTs signed by multiple loggers
    - Current approach in Google's Chrome; req's two SCTS (one Google)
    - If you require more redundancy… good luck finding certificates!
  - How many loggers? Which loggers? Overhead ?
    - Google's logger + a logger selected by (untrusted??) CA
- **Option 2: avoid 'honest-logger' assumption**
  - Two variants: AnG-CT and NS-CT
- AnG-CT: Audit and Gossip to detect rogue logger
  - Roughly follows RFC6962 and original CT publications
  - Complex, expose user privacy, … : see textbook if interested
- NTTP-Secure CT (NS-CT):
  - Ensures `no trusted third party' by Proofs-of-Misbehavior (PoM)

Audit-and-Gossip (AnG) Certificate Transparency

- Logger keeps certs in Merkle tree
  - Protocol uses digest, PoI and PoC mechanisms
  - Signed, timestamped digest: Signed Tree Head (STH)
- Logger must respond to several **audit requests**:
  - Request for STH+PoI, for given certificate
  - Request for PoC, for given pair of STHs
  - Request for current STH
  - Request for certificates, logged between given start/end times
- **Gossip:** sharing of STHs among entities
  - To detect 'split world attack': different STHs to different entities
- Textbook interpretation of `original' CT
  - Using Audit and Gossip to detect rogue loggers
  - No complete spec published until now.

What is missing in AnG-CT ?

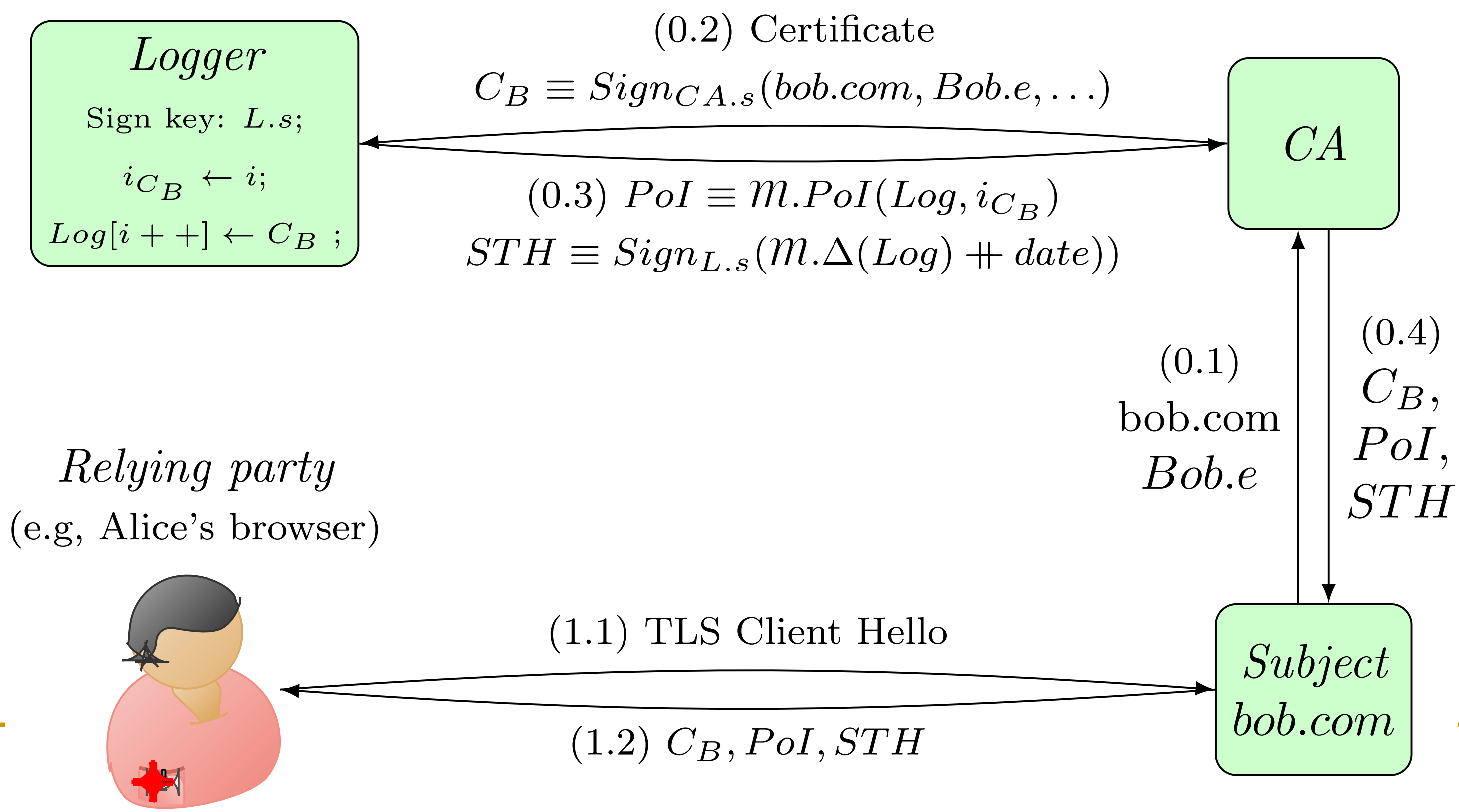
- AnG-CT may fail to provide Proof-of-Misbehavior
  - Logger never sends the STH for a rogue SCT !
  - Relying party receives no response… what can it do ??
  - Or, never responds to request for PoC for ‘rogue STH’…
- Also: AnG-CT does not address revocation transparency ➔ vulnerable to ‘zombie certificate attack’: send ‘valid’ response to OCSP query for a revoked certificate
- And: AnG-CT relying parties expose visited website
- Next: NTTP-Secure (NS) Certificate Transparency
  - Also based on Audit and Gossip, but addressing above issues
  - NTTP = No Trusted Third Party (e.g., logger not trusted)
  - Simplified: no SCT, logger responds only daily with STH (can be changed to give SCT for immediate responses, certs)

NTTP-Secure CT: Goals and Assumptions

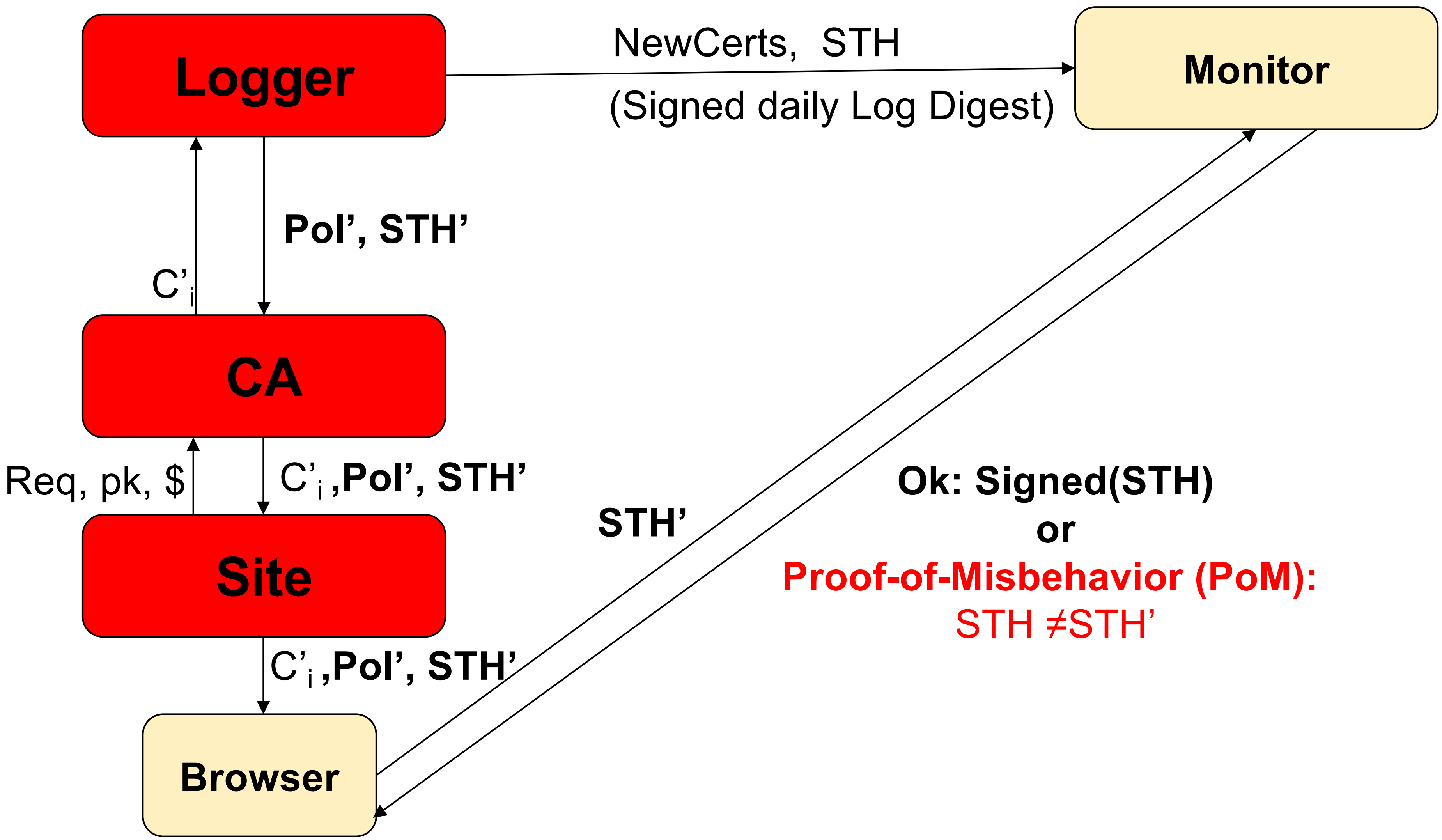
- **Secure against collusions of any set of parties**
  - Up to threshold t (maximal number of colluding parties)
- The **rapid rogue certificate mitigation property**: when a relying party receives a ‘valid’ certificate, then:
  - Every monitor received this certificate, or
  - Every monitor has Proof-of-Misbehavior of the logger.
- **The no false convictions property: an honest entity is never considered corrupt.**
- Simplifications/assumptions:
  - Reliable communication between entities, synchronized clocks
    - We ignore delays and clock-skews, easy to handle these details
  - There are at least $2t + 1$ monitors, and at most t faulty.
  - All monitors observe all loggers.

NS-CT : Simplification and Issue Process

- Loggers issue Signed Tree Head (STH) every 24 hours
  - And `immediately' provide it to all monitors
  - Response to CA includes STH and **Proof-of-Inclusion (PoI)**
  - CA, subject, relying party validate STH and PoI
  - Issue process almost unchanged – but takes 24 hours…



NS-CT: **Audit** detects an omitted cert



Summary: benefits of CT

- Benefits for websites:
  - Detect rogue certs for domain (same or misleading)
  - Once detected, owners can mitigate risk
    - Demand revocation, removal of CA from root program, …
- Benefit to users: less likely to fall victim…
- Benefit to trustworthy CAs: reduced competition from shady CAs
- Overall: more secure PKI !
- Not covered here:
  - Auditing and proving misbehavior of rogue monitors
  - The Zombie-certificate attack
  - Transparent revocation in NS-CT (prevents Zombie-cert attack)

Covered Material From the Textbook

- Chapter 8:
  - Sections 8.4, 8.5, and 8.6

Thank You!