Public Key Infrastructure (PKI)
Tutorial for CANS'20
Day 2: Revocation and Merkle Digest Schemes

Amir Herzberg
University of Connecticut

See ch. 8 of 'Applied Intro to Cryptography',
available at my site: .

PKI Tutorial – CANS'20: Agenda

- **Day 1: Introduction, X.509 and constraints**
- **Day 2: Revocations and Merkle Digests**
  - **The certificate revocation challenge**
  - Pre-fetching revocations: CRL, VRL, CRV
  - Just-in-Time fetching: OCSP and variants
- Day 3: CA failures + Certificate Transparency
- Conclusions, directions and challenges

Certificate Revocation

- Sometimes, certificates must be revoked
- Mainly, for security:
  - Key compromise: revoke relevant certificate
  - CA compromise: revoke **all** certificates it issued
  - Sometimes, for 'administrative' reasons
- Challenge: inform relying parties, provide **PoNR**
  - PoNR: Proof-of-Non-Revocation (e.g., for signed document)
  - Inform – when? **Pre-fetch** (e.g., daily) or **'just-in-time'** (before using the certificate)?

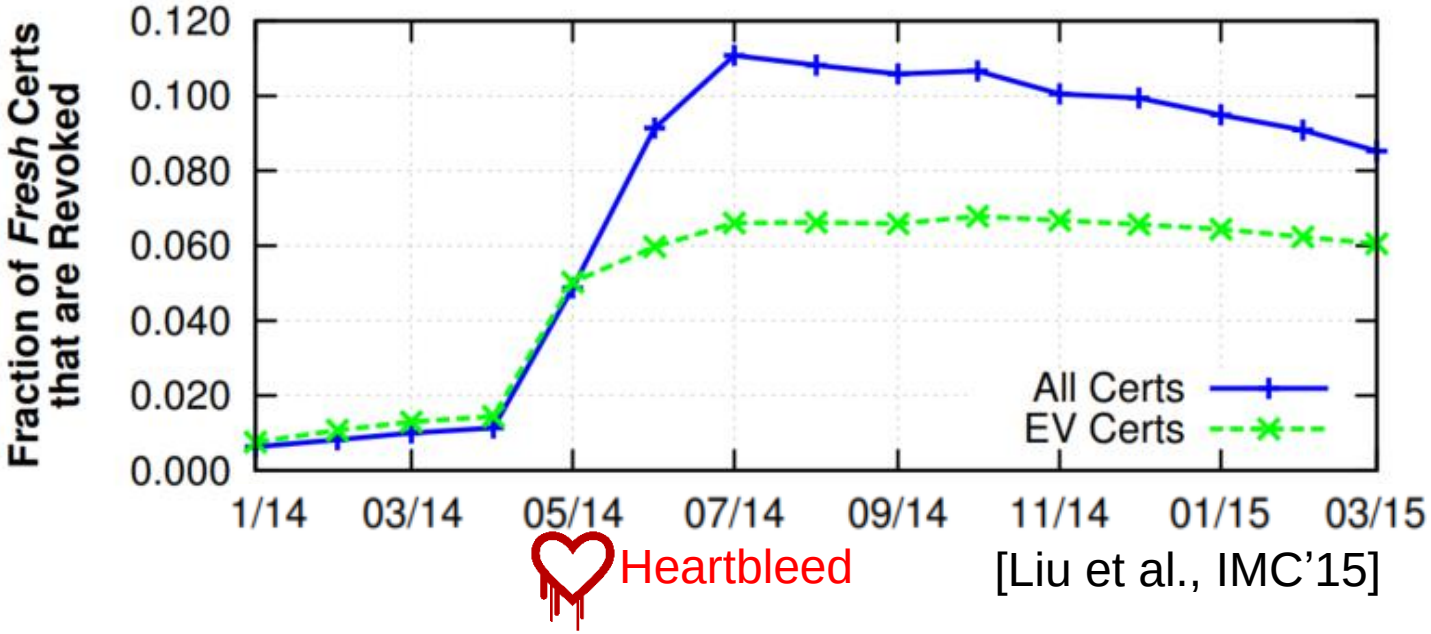
'TO PRE-FETCH OR NOT TO PRE-FETCH? THAT IS THE QUESTION.'

Certificate Revocation List (CRL)

- CA signs list of revoked certificates:
  $Sign_{CA.s}[\{Cert\#, Date\}, Issued, Expires, Extensions]$
  - One signature 'covers' all revocations in the CRL!
- Option 1: prefetch, i.e., download before $Next$
  - Cons: many CAs ➔ **lots** of download, storage…
  - Maybe no website using this CA ?
- Option 2: 'just-in-time' – upon validating certificate
  - Common design
  - Con: delay on entering site (and possible failure, too)
- Overall, seems fine, assuming revocations are rare
- **But are revocations really rare???**

Reality: Revocations Quite Common

- Significant fraction of certificates may be revoked at given time



[Liu et al., IMC'15]

- More efficient ways to revoke?

More efficient revocation

- Significant time btw CRLs ➔ freshness concern
- More efficient CRL schemes
  - CRL distribution point – split certificates to several CRLs
    - Tradeoff between number of signatures and size of CRL
    - Helps only if we load CRLs only 'as needed'
  - Authorities Revocation List (ARL): list only revoked CAs
    - Can be maintained better, e.g., pre-fetched
  - Delta CRL – only new revocations since last 'base CRL'
    - Smaller downloads, but harder to prove non-revocation
- Or: **Vendor's Revocation List (VRL)**
  - Revoked certificates of **all** CAs (maintained by vendor)
  - Main current revocation mechanism (most browsers)
  - May not contain all revoked certificates, though…
- Or, let's revoke using CRV, not CRL !

Let's Revoke with CRVs!

[Smith, Dickinson, Seamons] NDSS'20

- A 'revoked-bit-vector' instead of CRL
- CRV: Certificate Revocation **Vector**
  - Add **revocation number** extension to each certificate, counting certs issued by this CA, with same expiration date
  - $Cert[CA, d_{EXP}, r]$: cert with revocation number r expiring at d_{EXP}
  - $CRV[CA, d_{EXP}, r] = 1$ if $Cert[CA, d_{EXP}, r]$ was revoked
- Browsers fetch signed-CRVs from CA (daily)
- For further efficiency:
  - Compress, using the fact that most certs are not revoked
  - By sending lengths of 0-bit (non-revoked) sequences
  - And by sending 'Delta-CRVs': only revocations from yesterday
  - Length of update: up to 22KB for 90M certificates
- See paper for details, variants and beautiful graphs…

Let's Revoke with CRVs!

[Smith, Dickinson, Seamons] NDSS'20

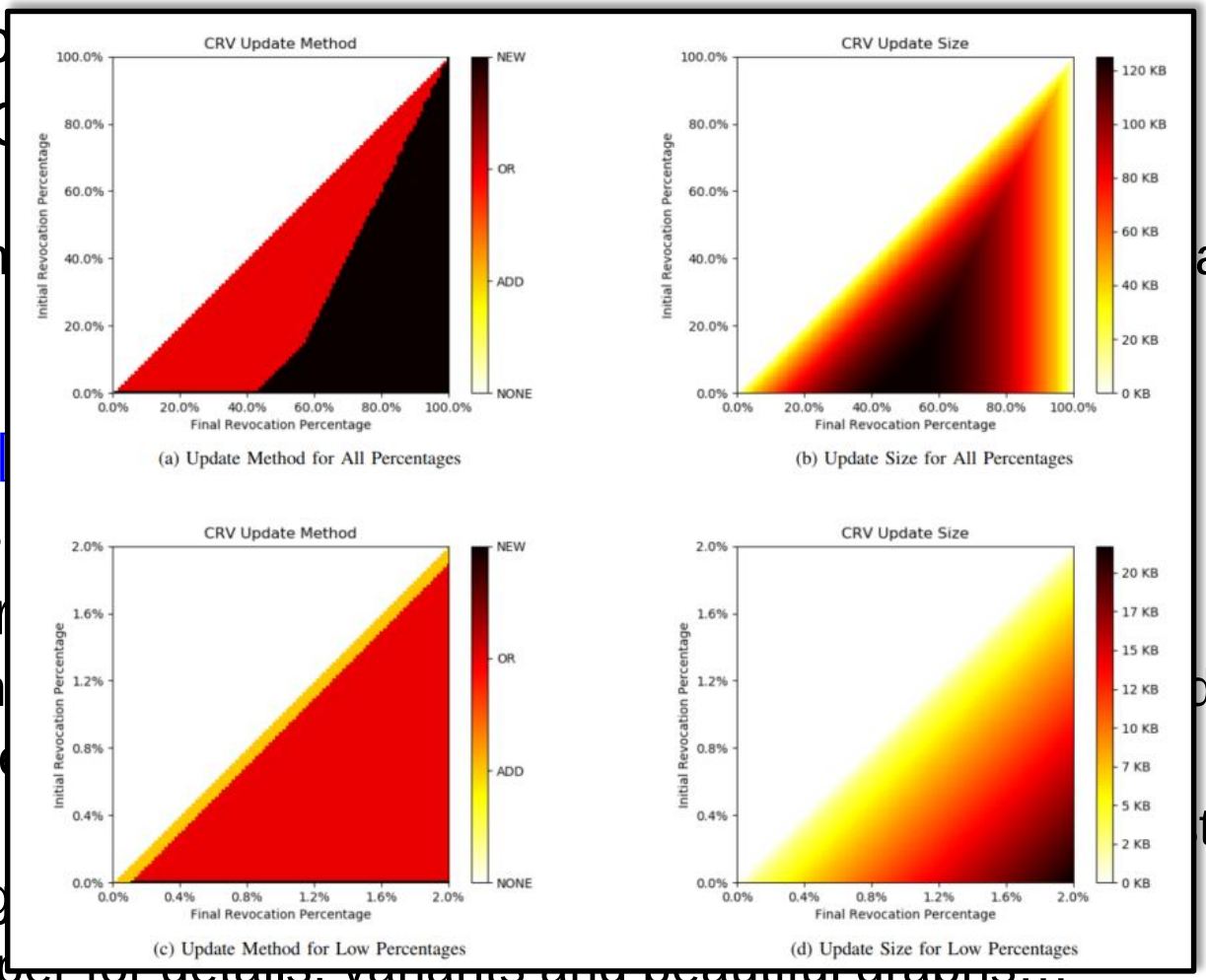
- A 'revo
- CRV: C
  - Add coun
  - $Cert$ d_{EXP}
  - CRV
- Brows
- For fur
  - Com
  - By s
  - And
  - Leng
- See pap

ate

at

d

terday



Let's Revoke: read the fine print…

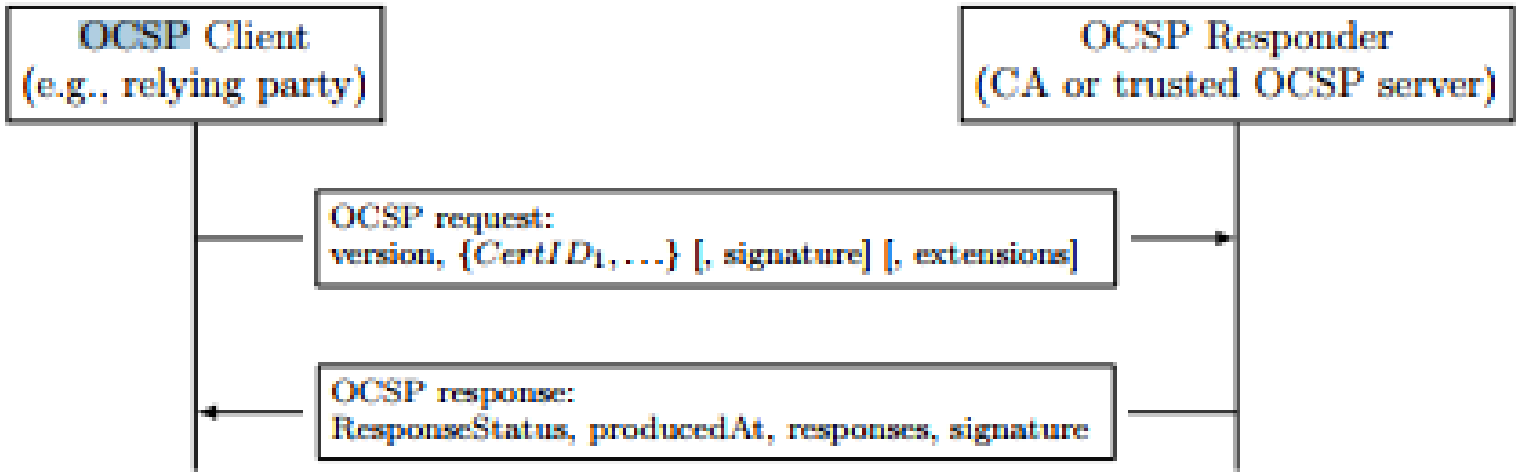
- In rough order of increasing difficulty…
- Revocation numbers **potential exposure** :
  - Expose number of certs from CA
  - X.509 serial numbers are random!
- Requires a **new extension** to the certificates
- CRV is per-CA and per expiration date
  - Web-PKI: 100s CAs , many expiration days
  - ➔ **Many many CRVs (>10,000 for sure)**
  - **Sent to every relying party (browser) daily…**
- ➔ **Still high overhead**
- **Maybe we shouldn't pre-fetch?**

PKI Tutorial – CANS'20: Agenda

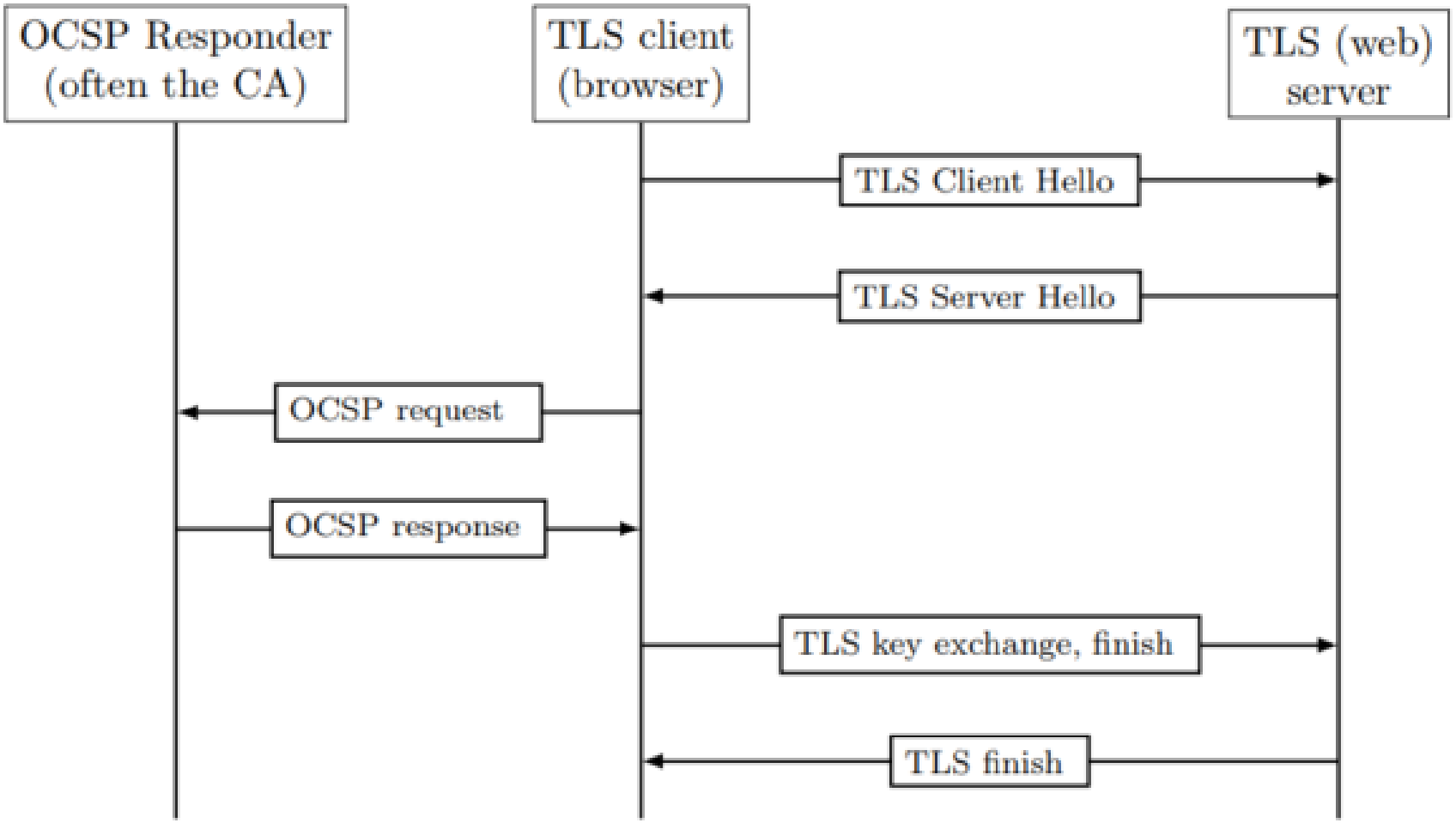
- **Day 1: Introduction, X.509 and constraints**
- **Day 2: Revocations and Merkle Digests**
  - The certificate revocation challenge
  - Pre-fetching revocations: CRL, VRL, CRV
  - **Just-in-Time fetching: OCSP and variants**
- Day 3: CA failures + Certificate Transparency
- Conclusions, directions and challenges

Online Certificate Status Protocol (OCSP)

- Most browsers don't pre-fetch most certificates:
  - Don't use CRLs due to efficiency, freshness concerns
  - Vendors lists (OneCRL, etc.): only some certificates
  - CRVs: not deployed (and concerns?)
- OCSP: 'just-in-time' check for revocation
- Signed responses (from trusted CA/server)



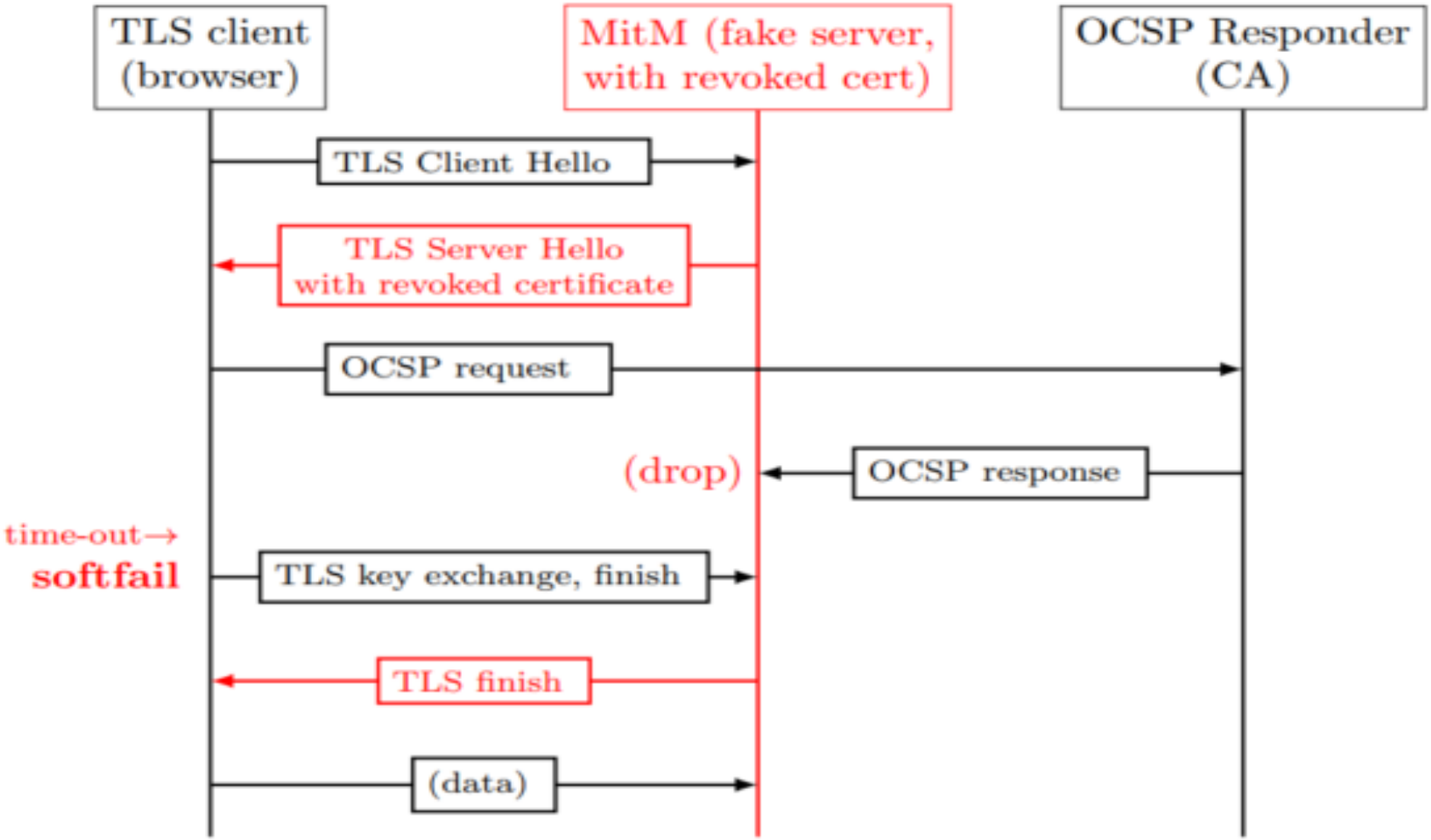
‘Classic’ use of OCSP by TLS Client



‘Classic’ OCSP: Delay and Loss Concerns

- Client asks CA about cert during handshake
- CA signs response (real-time)
- Delay
  - Significant added delay to page load
- Reliability
  - What to do if no response (loss / no connectivity)?
    - Resend request: more overhead on client, CA and network
  - How much to wait before determining loss?
    - Short timeout: easy to circumvent with DDoS
    - Long timeout: even longer delay on page load upon loss
- ➔ Most browsers soft-fail: continue w/o OCSP response
  - Hmm… is this secure ?

MitM Attack on Soft-Fail 'classic' OCSP

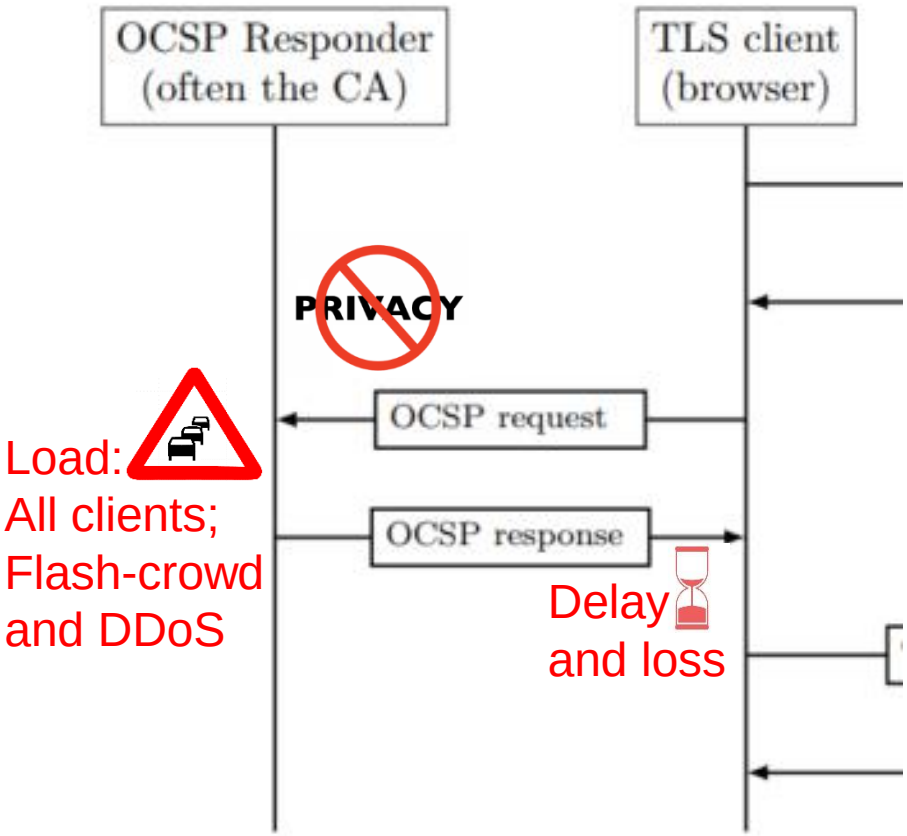


Soft-Fail is too Soft. Why do it ???

- Why not deploy OCSP without soft-fail?
  - Foiling the MitM soft-fail attack !
- Hard-Fail: browser refuses connection unless/until receiving (good) OCSP response
- Possible answers?
  - Good idea. Google, MS and Apple are dummies.
  - No way, users will switch browsers.
- Principle: User Experience (UX) > Security
  - ‘Precedence rule’

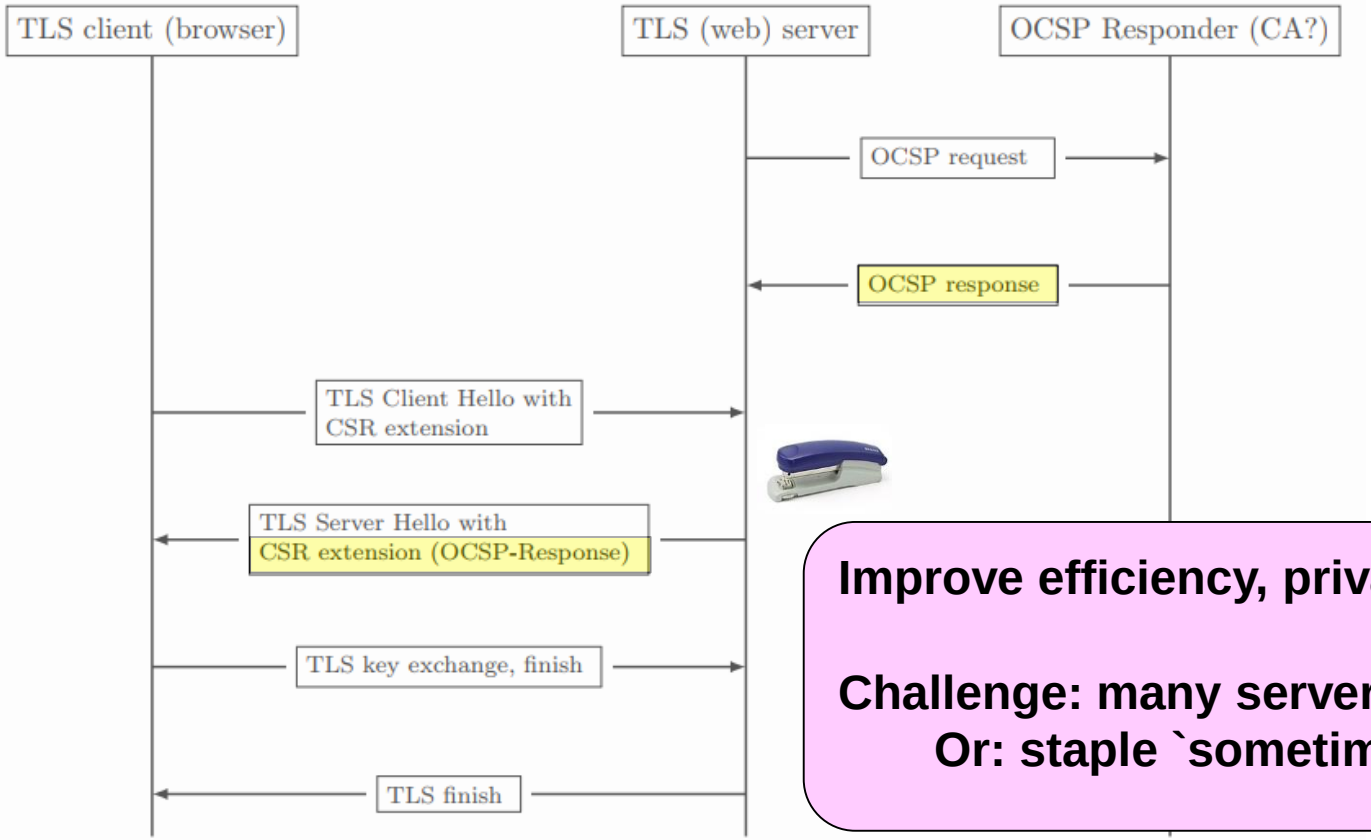
'Classic' use of OCSP: **Three Concerns**

- Delay and Reliability
  - Significant added delay to page load
  - Soft-fail ➔ vulnerability
  - Hard-fail ➔ connection may fail due to loss / delay
- Privacy : exposes (domain, client) to CA
- Load and DDoS on CA:
  - Many clients (all browsers!)
  - Potentially 'together': flash crowds
  - Easy for abuse with DDoS



OCSP-Stapling

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

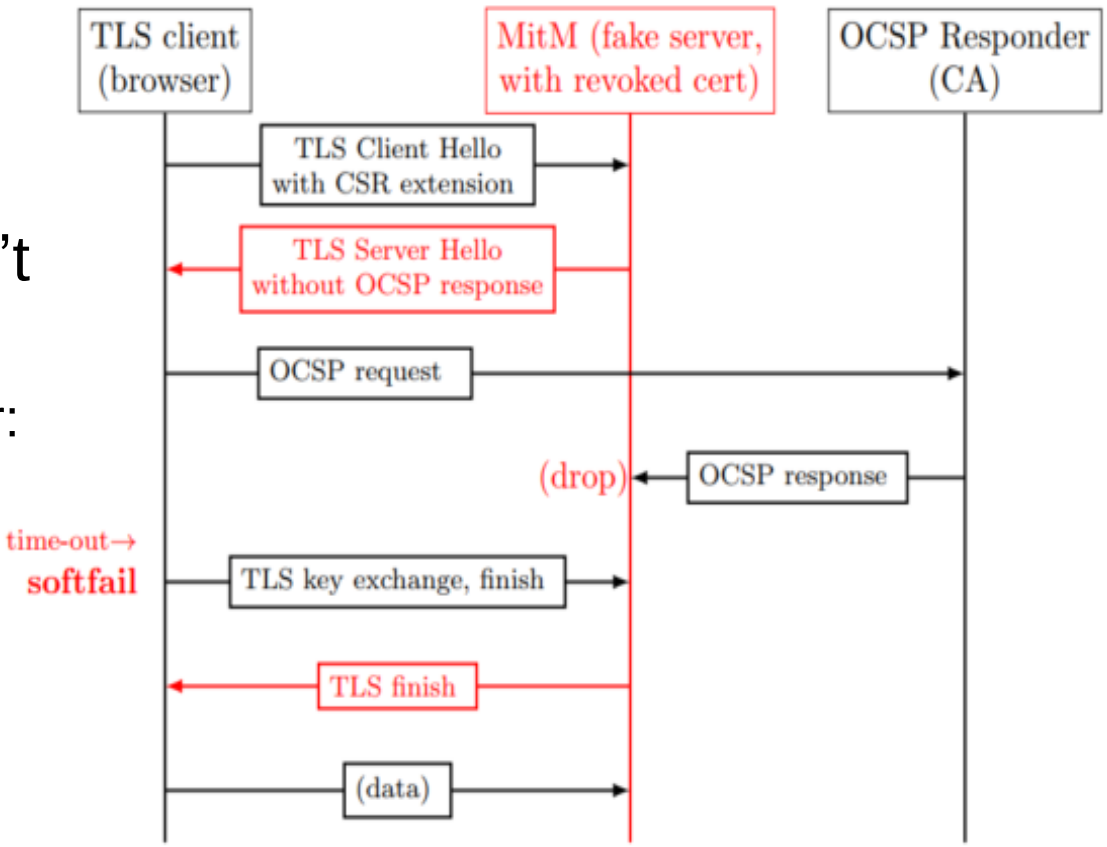


Improve efficiency, privacy, reliability

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

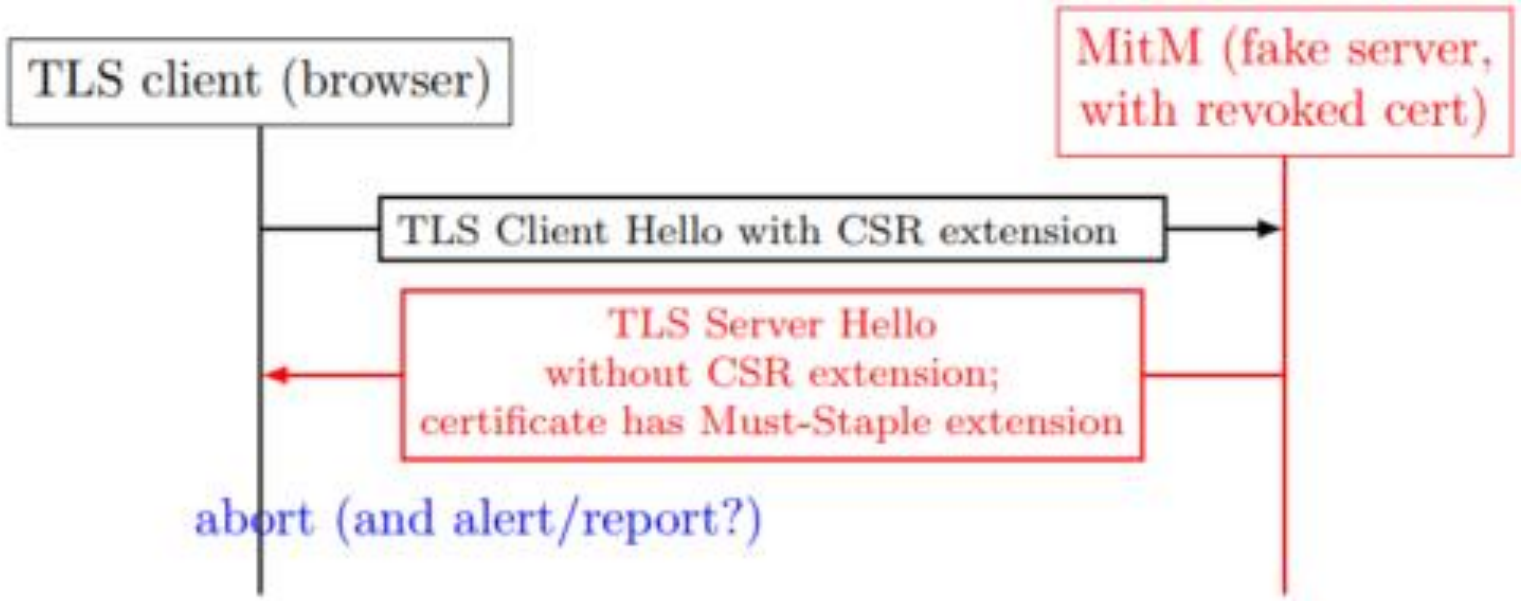
OCSP-Stapling: what if not stapled?

- OCSP-stapling: server should send (`staple') CA-signed OCSP response, with certificate
- But many servers don't (always) staple!
  - Don't support OCSP, or: support, but not always
  - So, try 'classic' OCSP?
  - If no response… soft-fail?
  - ➔ similar MitM attack !



OCSP: 'Must-Staple' X.509 extension

- If server's certificate contains 'must-staple' extension, client will **hard-fail** if an OCSP response isn't stapled
- Mark as not critical X.509 extensions
  - Since it may not be supported by some browsers



Optimization to OCSP

- OCSP stapling reduces overhead: one signature, response per website (subject)
- Still, high overhead:
  - Separate signature and message per website
- Two types of optimizations:
  - **Hash-chain:** use hashing to reduce signing
    - CA adds to OCSP response $h^{(n)}(x)$ for random x
    - Where $h^{(n)}(x) = h\left(h^{(n-1)}(x)\right)$, $h^{(1)}(x) = h(x)$
  - **Merkle-digest:** same signature for many sites
    - Three methods…
    - Quick recap of this widely-used by rarely defined scheme…

Merkle Digest Schemes

- Digest function Δ: $\{m_i \epsilon \{0,1\}^*\} \rightarrow \{0,1\}^*$
  - Collision-resistance requirement
- Validation of Inclusion: PoI and $VerPoI$
  - PoI function: compute Proof of Inclusion
  - $VerPoI$ function: verify PoI
  - Both: mandatory and optimized
  - Optional, also Proof-of-Non-Inclusion (PoNI)
- Extending the Sequence: PoC and $VerPoC$
  - PoC: Proof of Consistency (from old digest to new)
  - $VerPoC$ function: verify PoC
  - Optional

Merkle digest scheme: definition

Definition 4.13 (Merkle digest scheme). *A* Merkle digest scheme $\mathcal{M}$ *is a tuple of three PPT functions* $(\mathcal{M}.\Delta, \mathcal{M}.PoI, \mathcal{M}.VerPoI)$, *where:*

$\mathcal{M}.\Delta$ *is the* Merkle tree digest *function, whose input is a sequence of messages* $M = \{m_i \in \{0,1\}^*\}_i$ *and whose output is an* n*-bit* digest*:* $\mathcal{M}.\Delta : (\{0,1\}^*)^* \to \{0,1\}^n$.

$\mathcal{M}.PoI$ *is the* Proof-of-Inclusion *function, whose input is a sequence of messages* $M = \{m_i \in \{0,1\}^*\}_i$, an integer $i \in [1, |M|]$ *(the index of one message in* M*), and whose output is a* Proof-of-Inclusion *(PoI):* $\mathcal{M}.PoI : (\{0,1\}^*)^* \times \mathbb{N} \to \{0,1\}^*$.

$\mathcal{M}.VerPoI$ *is the* Verify-Proof-of-Inclusion *predicate, whose inputs are digest* $d \in \{0,1\}^n$, *message* $m \in \{0,1\}^*$, *index* $i \in \mathbb{N}$, *proof* $p \in \{0,1\}^*$, *and whose output is a bit (*1 *for 'true' or* 0 *for 'false'):* $\mathcal{M}.VerPoI : \{0,1\}^n \times \{0,1\}^* \times \mathbb{N} \times \{0,1\}^* \to \{0,1\}$.

Merkle digest: correctness and security

A Merkle digest scheme $\mathcal{M}$ *is* correct *if for every sequence of messages* $M = \{m_i \in \{0,1\}^*\}_i$ *and every index* $i \in [1, |M|]$, *the Proof-of-Inclusion verifies correctly, i.e.:*

$$\mathcal{M}.VerPoI(\mathcal{M}.\Delta(M), m_i, i, \mathcal{M}.PoI(M,i)) \tag{4.25}$$

A Merkle digest scheme $\mathcal{M}$ *is* secure *if for every efficient (PPT) algorithm* $\mathcal{A}$, *both the* collision advantage $\varepsilon^{Coll}_{\mathcal{M},\mathcal{A}}(n)$ *and the* PoI advantage $\varepsilon^{PoI}_{\mathcal{M},\mathcal{A}}(n)$ *are* negligible *in* n, *i.e., smaller than any positive polynomial for sufficiently large* n *(as* $n \to \infty$*), where:*

$$\varepsilon^{Coll}_{\mathcal{M},\mathcal{A}}(n) \equiv \Pr\left[\begin{array}{c}(x,x') \leftarrow \mathcal{A}(1^n)\ s.t.\ (x \neq x') \\ \wedge(\mathcal{M}.\Delta(x) = \mathcal{M}.\Delta(x'))\end{array}\right]$$

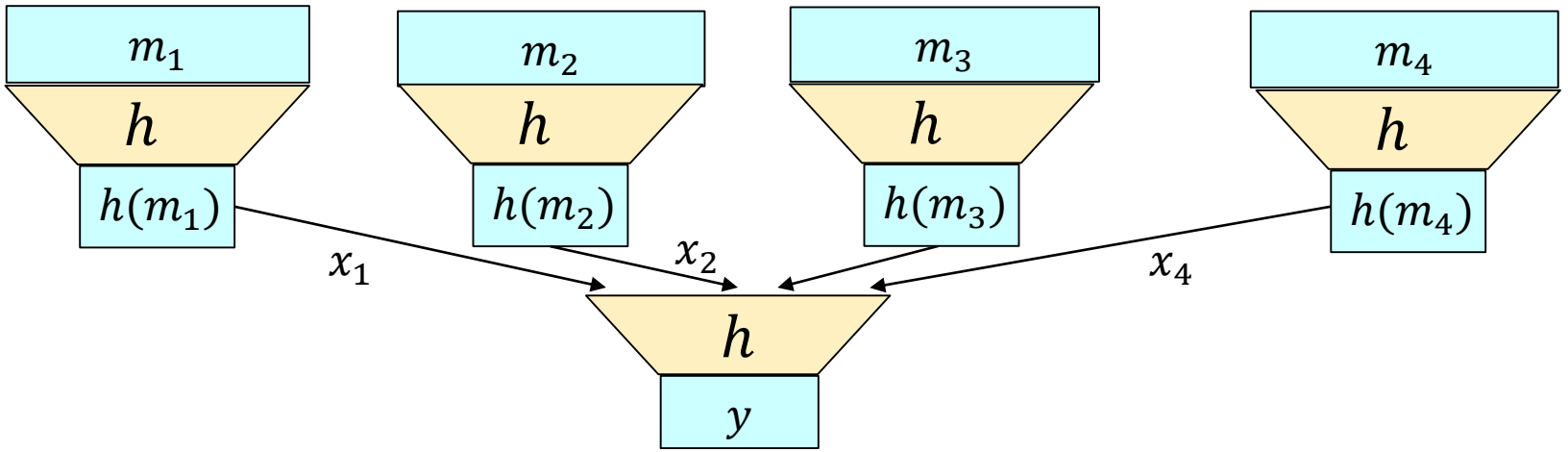
$$\varepsilon^{PoI}_{\mathcal{M},\mathcal{A}}(n) \equiv \Pr\left[\begin{array}{c}(d,m,i,p) \leftarrow \mathcal{A}(1^n)\ s.t.\ \mathcal{M}.VerPoI(d,m,i,p) \wedge \\ (\nexists x \in D)(d = \mathcal{M}.\Delta(x))\end{array}\right]$$

Two-layered Merkle tree

$$2l\mathcal{MT}.\Delta(m_1,\ldots,m_l) \equiv h\left[h(m_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} h(m_l)\right]$$

$$2l\mathcal{MT}.PoI((m_1,\ldots,m_l),j) \equiv \{h(m_i)\}_{i=1}^{l}$$

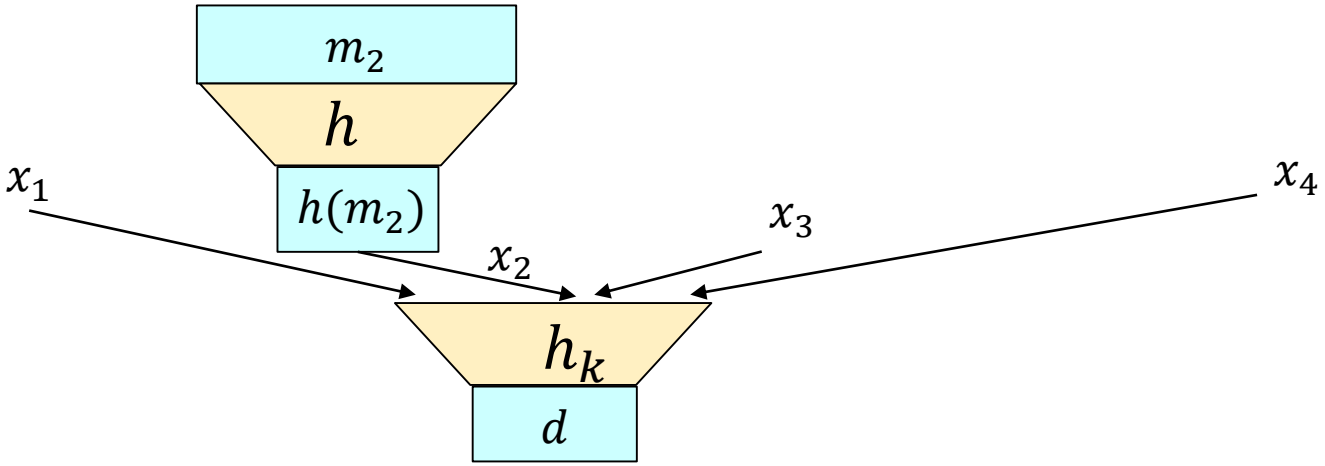
$$2l\mathcal{MT}.VerPoI(d,m,i,\{x_i\}_{i=1}^{l}) \equiv \left[\begin{array}{c}\textsc{True } if\ x_i = h(m),\ and \\ d = h(x_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} x_l)\end{array}\right]$$



Allows each user to receive, validate only required items. How?

To verify inclusion of m_2 ...

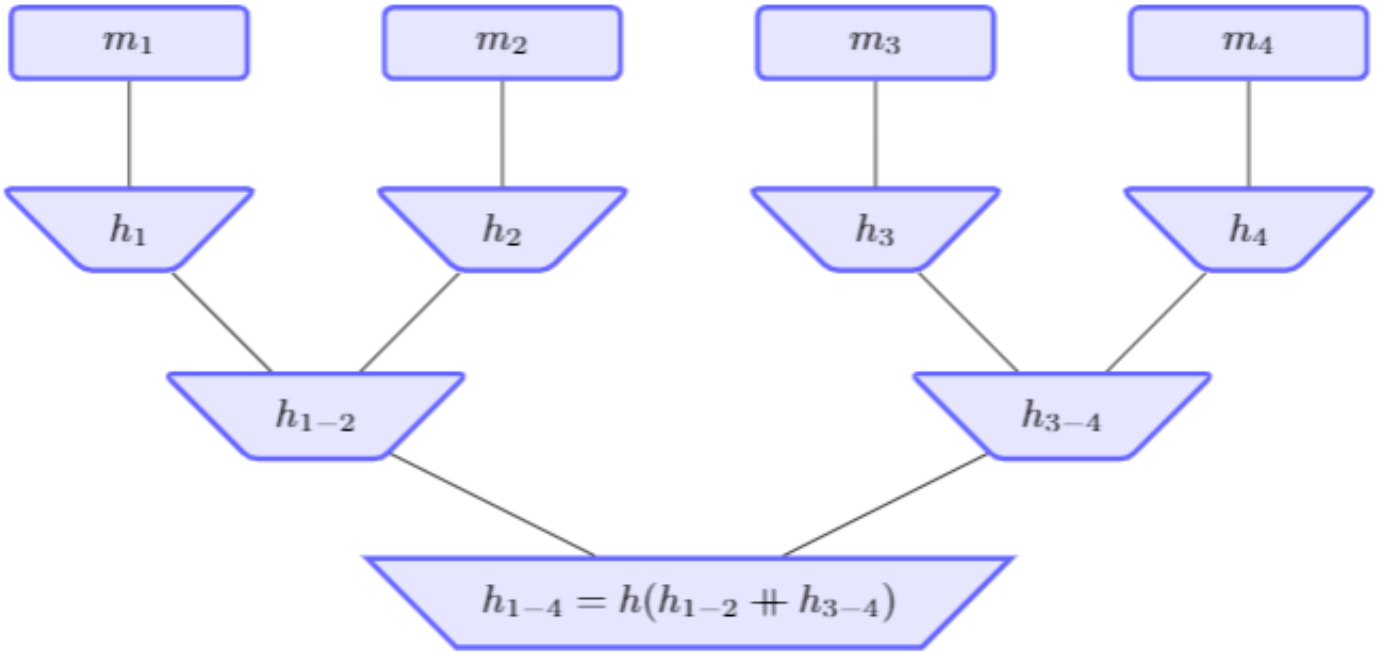
$$
\begin{aligned}
2l\mathcal{MT}.\Delta(m_1,\ldots,m_l) &\equiv h\left[h(m_1) \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} h(m_l)\right] \\
2l\mathcal{MT}.PoI((m_1,\ldots,m_l),j) &\equiv \{h(m_i)\}_{i=1}^{l} \\
2l\mathcal{MT}.VerPoI(d,m,i,\{x_i\}_{i=1}^{l}) &\equiv \left[\begin{array}{c} \textsc{True} \text{ if } x_i = h(m), \text{ and} \\ d = h(x_1 \mathbin{+\!\!+} \ldots \mathbin{+\!\!+} x_l) \end{array}\right]
\end{aligned}
$$



Receive and validate only m_2. Other hashes still required, though.

The Merkle Tree Construction

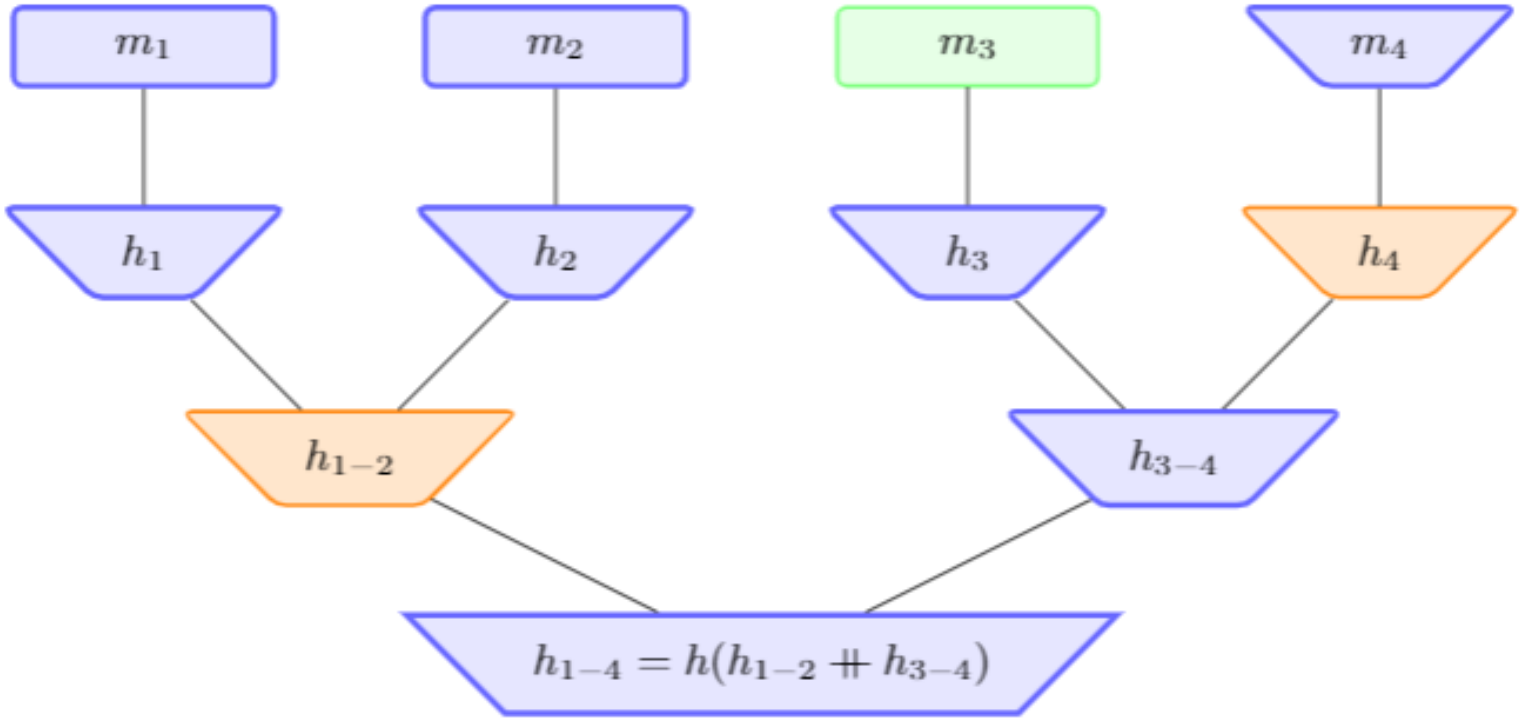
- Reduce length of ‘proofs’ – send less hashes of ‘other msgs’



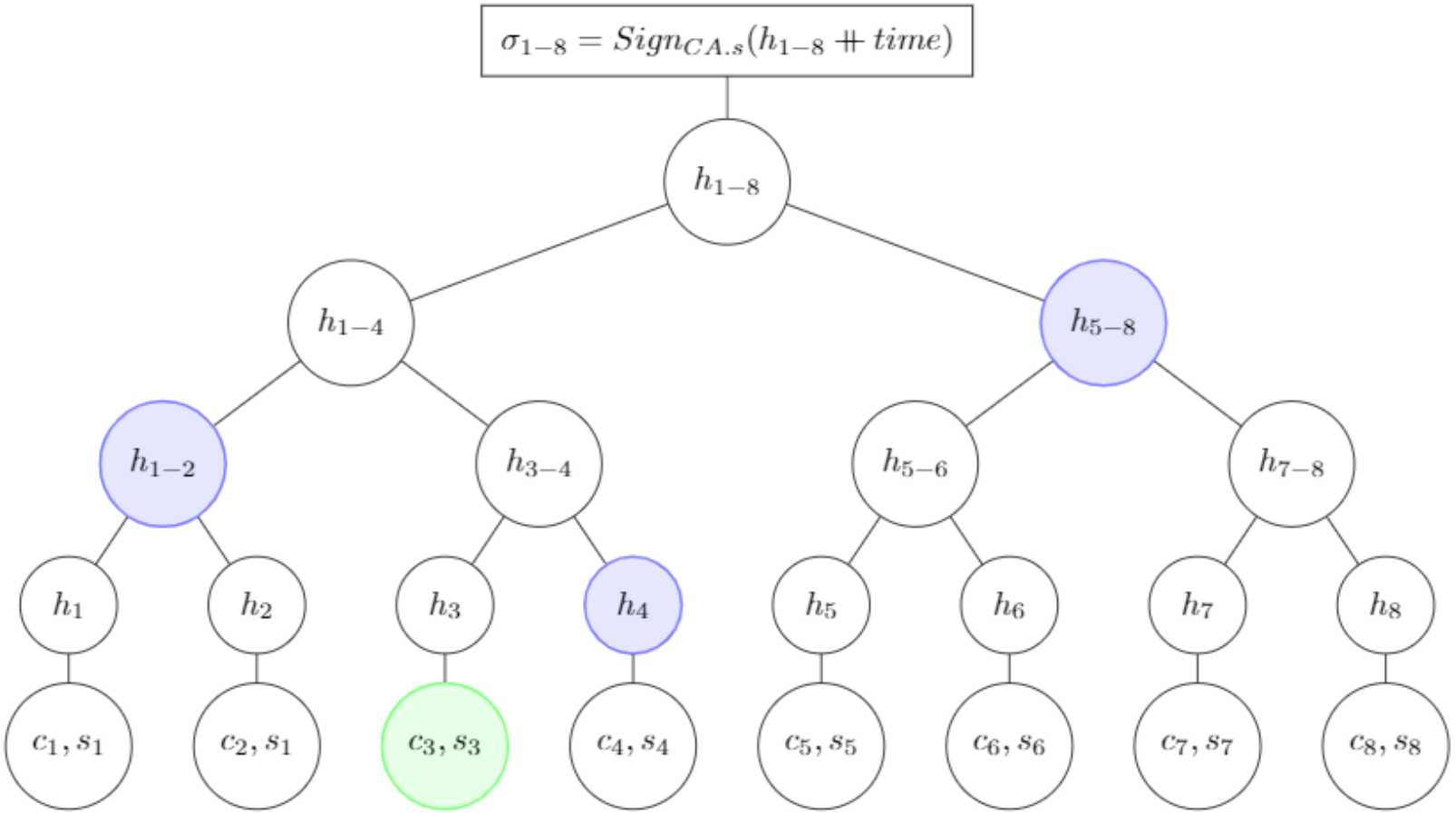
$$\mathcal{MT}.\Delta(M) \equiv \begin{cases} \text{If } L = 0: & h(m_1) \\ \text{Else} & h\left(\mathcal{MT}.\Delta\left(m_1, \ldots, m_{2^{L-1}}\right) + \right. \\ & \left. + \mathcal{MT}.\Delta\left(m_{2^{L-1}+1}, \ldots, m_{2^L}\right)\right) \end{cases}$$

Merkle Tree: Proof of Inclusion (PoI)

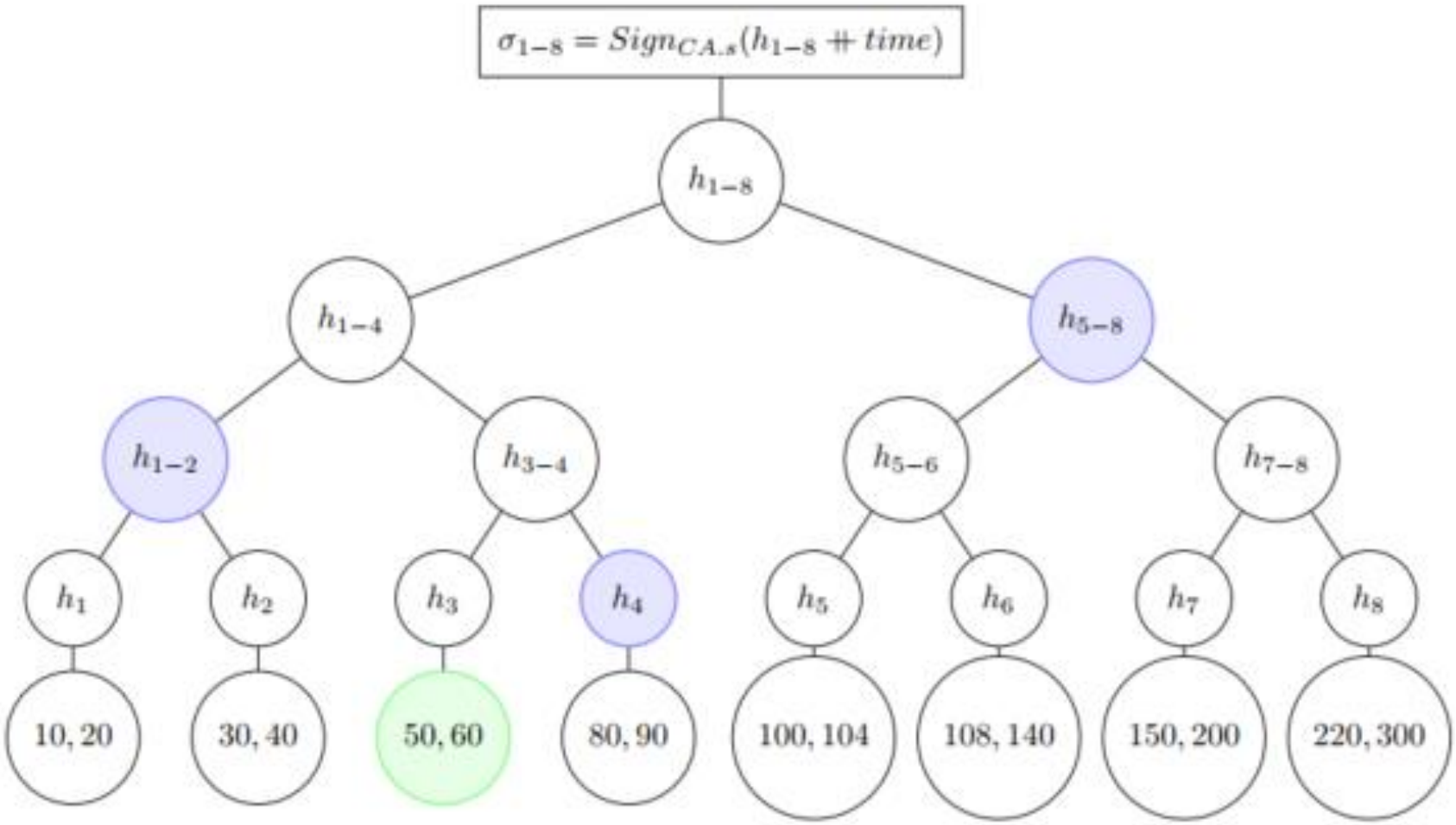
- Proof of Inclusion (PoI) of m_3 consists of:
  - $h_{1-2} = h\big(h(m_1)||h(m_2)\big)$
  - $h_4 = h(m_4)$



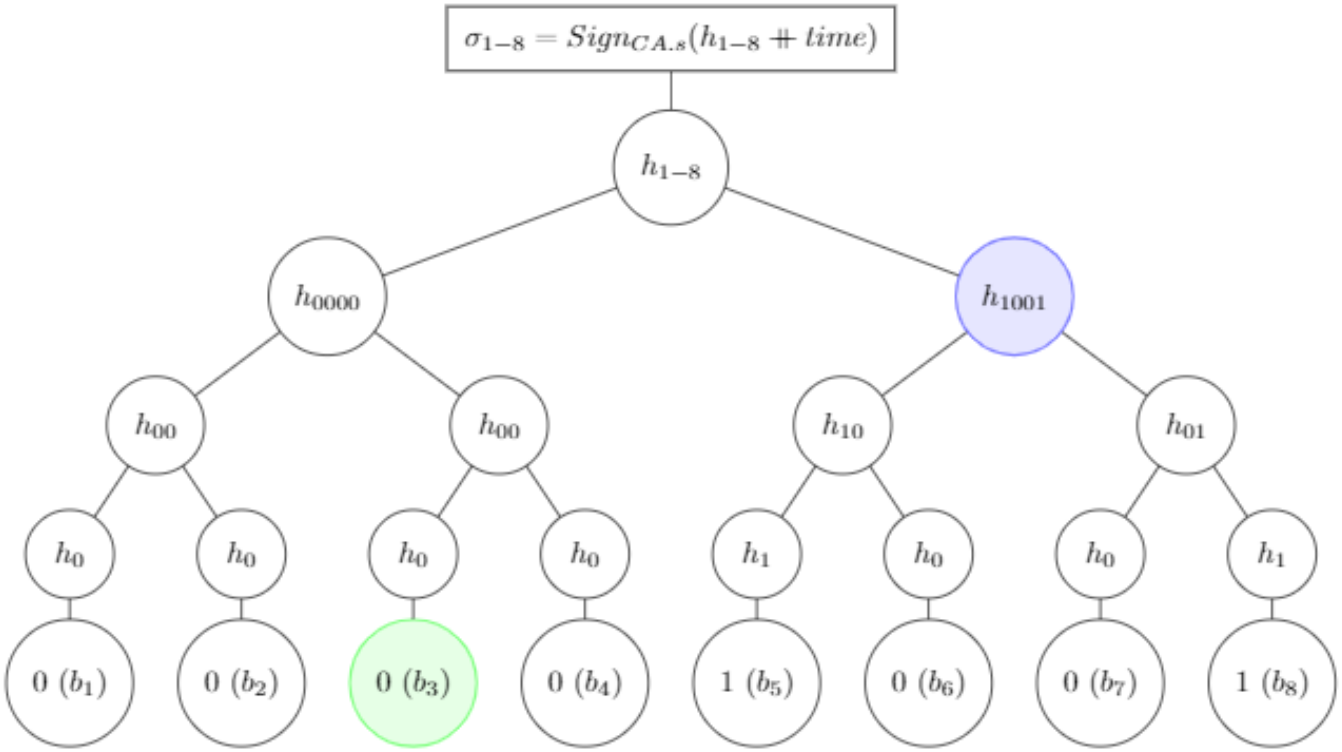
Merkle-tree Solution 1/3: Tree of Certificate Statuses, and Proof-of-Inclusion



Merkle-tree Solution 2/3: Tree of Revoked Certificates, and Proof-of-Non-Inclusion



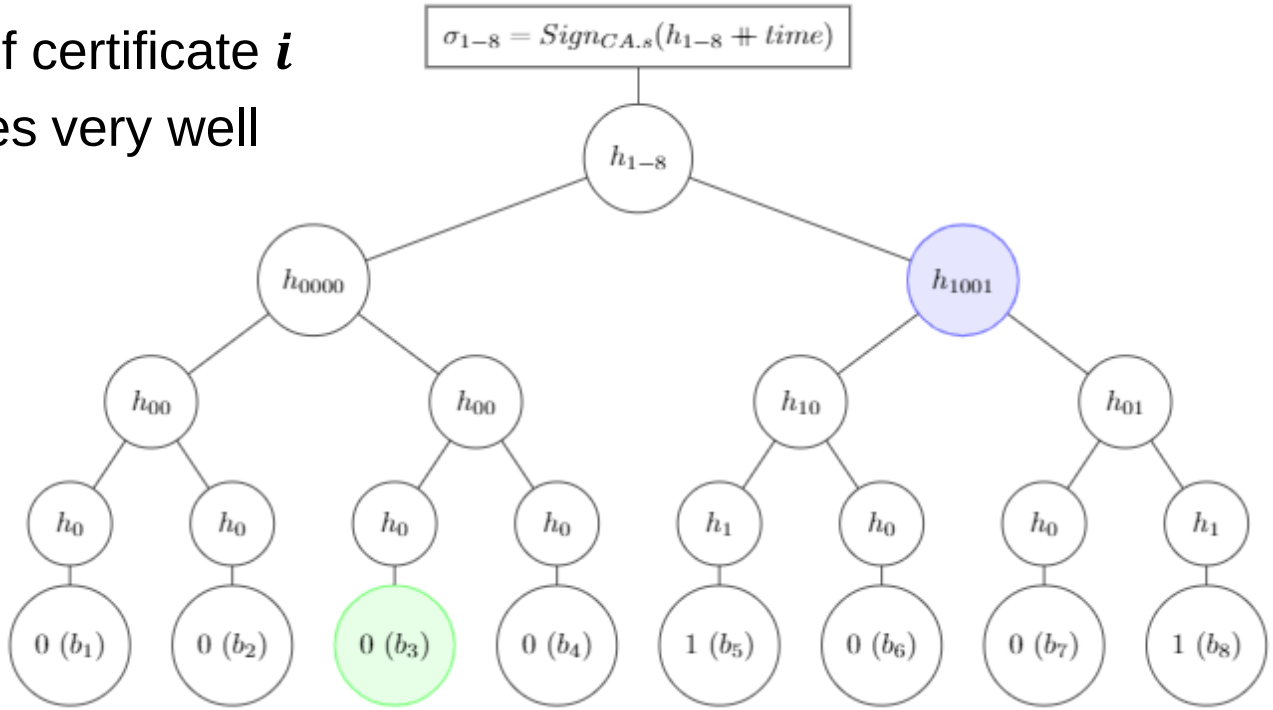
Merkle-tree Solution 2/3: Signed Revocation-bit Merkle-Tree



- Further optimizations: don't send zero-hashes; batching: many certs in each leaf

Signed Revocations-Status Merkle-Tree

- b_i: status of certificate i
- Compresses very well



- Further optimizations:
  - don't send zero-hashes
  - batching: many certs in each leaf
- Very efficient
  - One signature !
  - Short message
  - Quick validation

Short-Term Certificates vs. OCSP

- Idea: every method (e.g., OCSP) has short validity period
- So: issue certs with short validity-period – never revoke!
- A simple solution available **today**
- Optimizations possible – just like for OCSP
  - Using X509 extensions
- E.g.: Hash-chain short-term certificate renewal
  - Yearly-signed certificate, monthly-preimage-renewal
  - December: sign new yearly cert, with $h^{(12)}(x)$
    - Random x
    - Each month, expose a preimage: $h^{(11)}(x)$, $h^{(10)}(x)$, ...
      - Validate extension, e.g.: $h^{(11)}(x)=h(h^{(10)}(x))$

Note: Revocation Assume Honest CA!

- A rogue CA can fail to revoke, allowing attacker to use exposed key
- Just one of the many possible attacks of a rogue CA…
- Next topic!!

PKI Tutorial – CANS'20: Agenda

- **Day 1: Introduction, X.509 and constraints**
- **Day 2: Revocations and Merkle Digests**
- **Day 3: CA failures and Certificate Transparency**
- Conclusions, directions and challenges