Next-Generation Secure Public-Key Infrastructures

Paweł Szałachowski

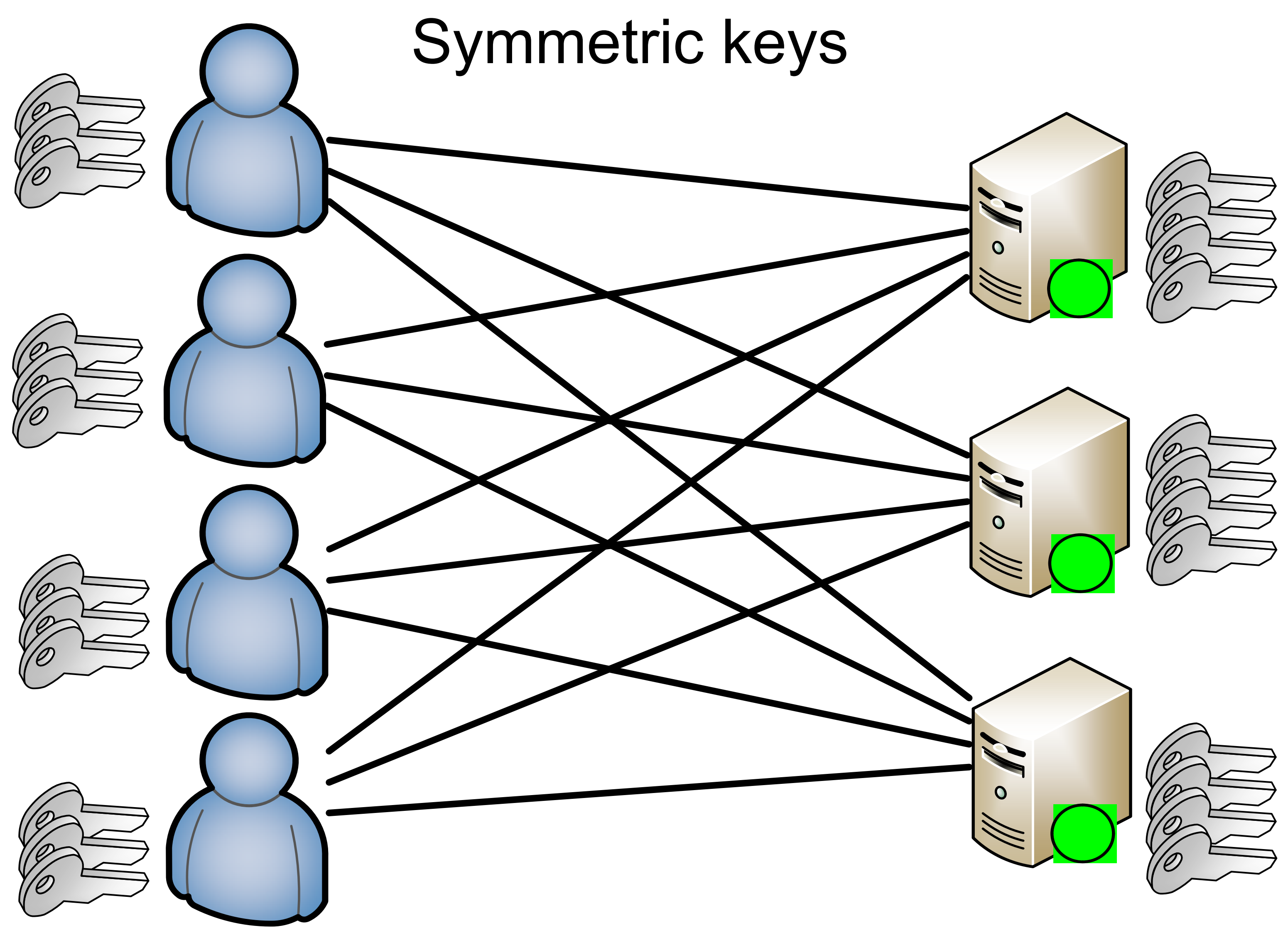
Network Security Group, ETH Zürich

ETH zürich

SCION

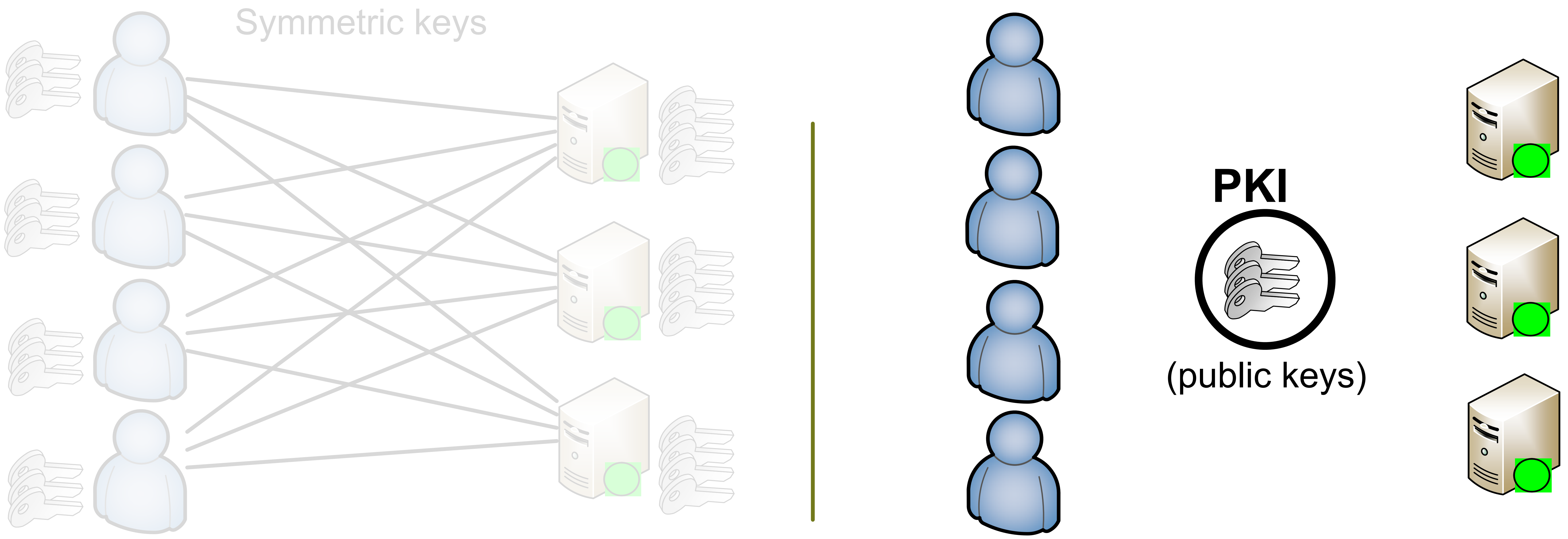
Public Key Infrastructure (PKI)

- Scalability issues with symmetric crypto
  - Distribution
  - Challenges in managing *n* secrets



Public Key Infrastructure (PKI)

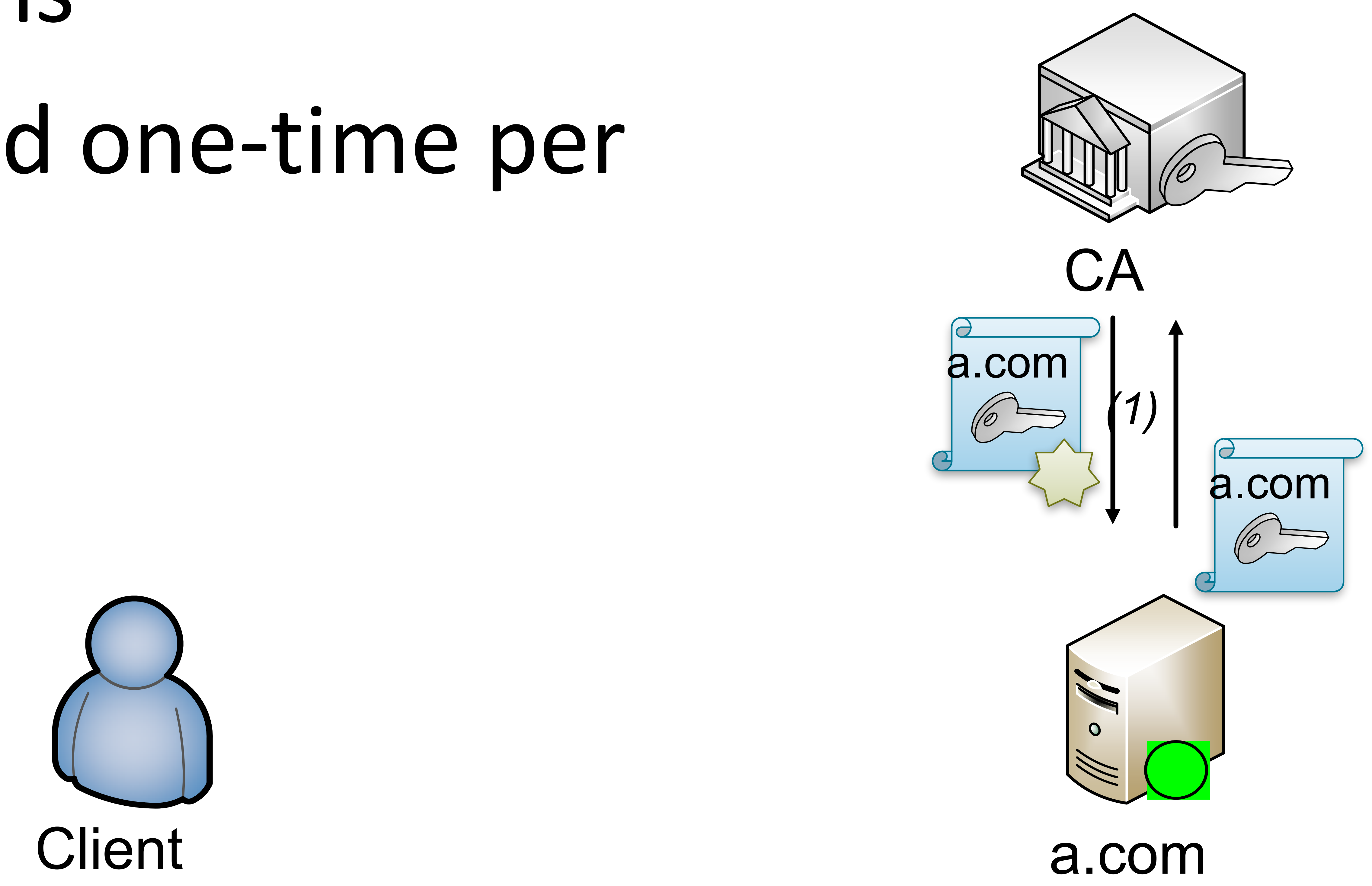
- Scalability issues with symmetric crypto
  - Distribution
  - Challenges in managing *n* secrets
- Asymmetric crypto (DH, RSA, …) solves the scalability problems, ... but creates a new one:
- **How to ensure that public-key is accessible and authentic ?**



ETH zürich

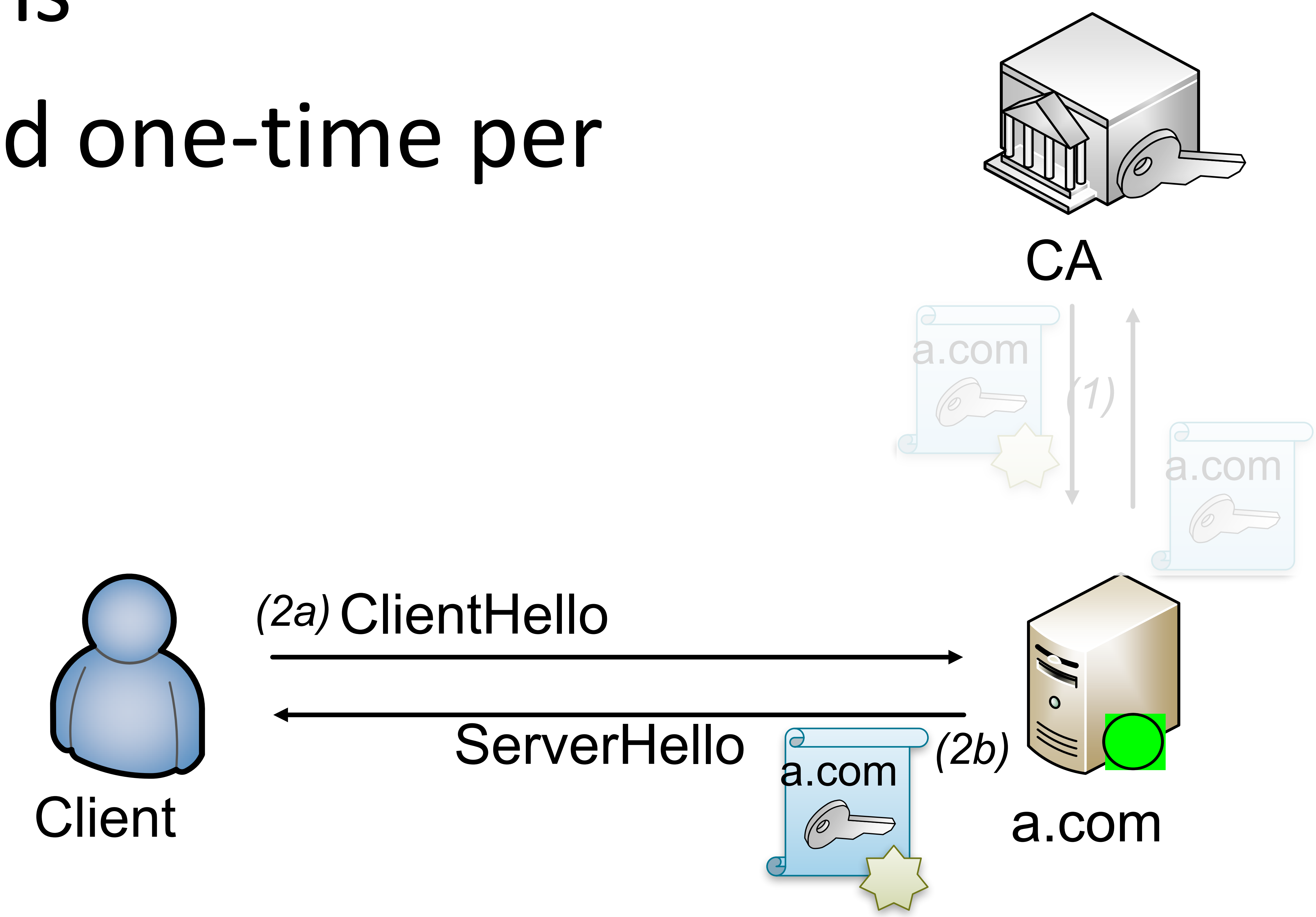
Current SSL/TLS PKI Model

- SSL/TLS Protocol
- Certification Authority (CA) is trusted by clients and domains
- Step *(1)* performed one-time per certificate



Current SSL/TLS PKI Model

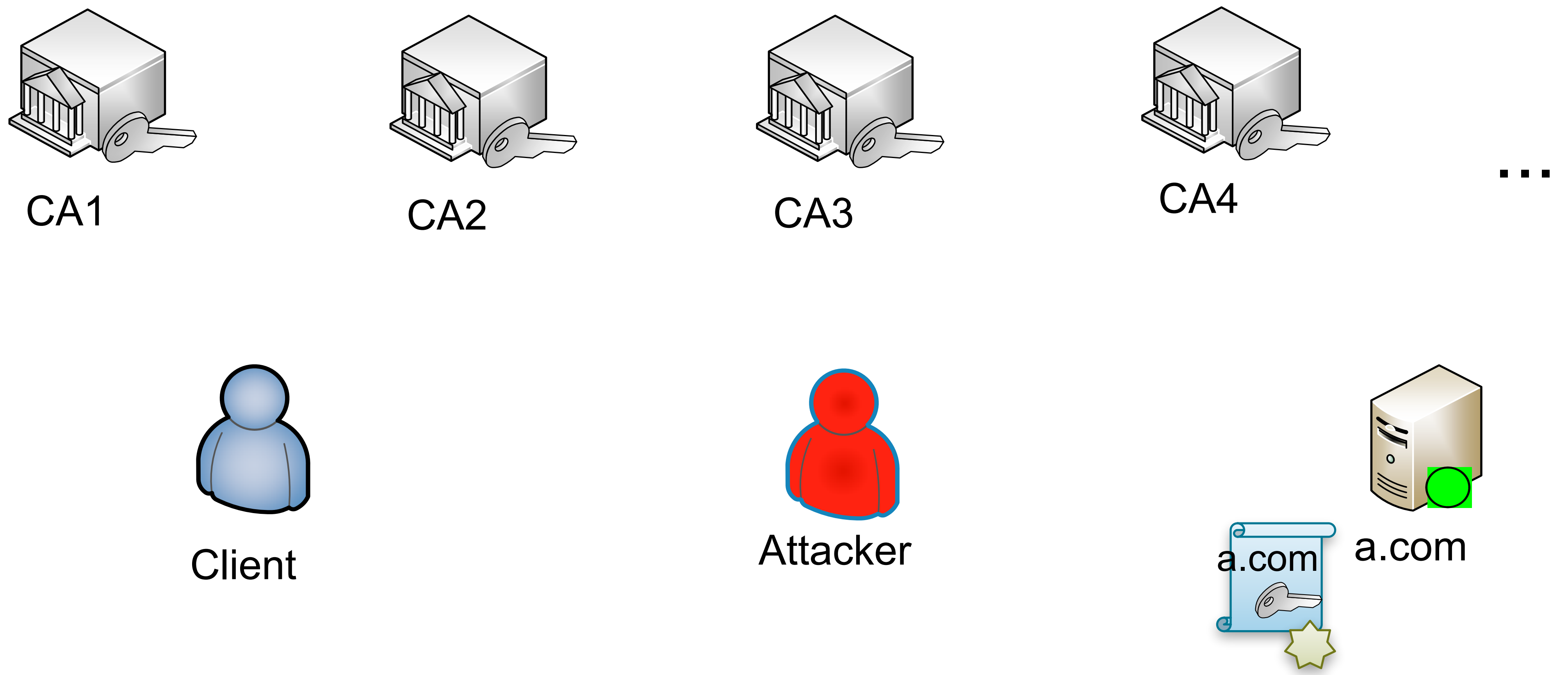
- SSL/TLS Protocol
- Certification Authority (CA) is trusted by clients and domains
- Step *(1)* performed one-time per certificate



Problem with current SSL/TLS PKI: Weak certificate authentication

- Certificates signed by single CA
  - Currently, cannot sign certificate by multiple CAs
- Weakest-link security with too many *trusted* entities
  - Current browsers trust ~1500 keys that can issue valid certificates

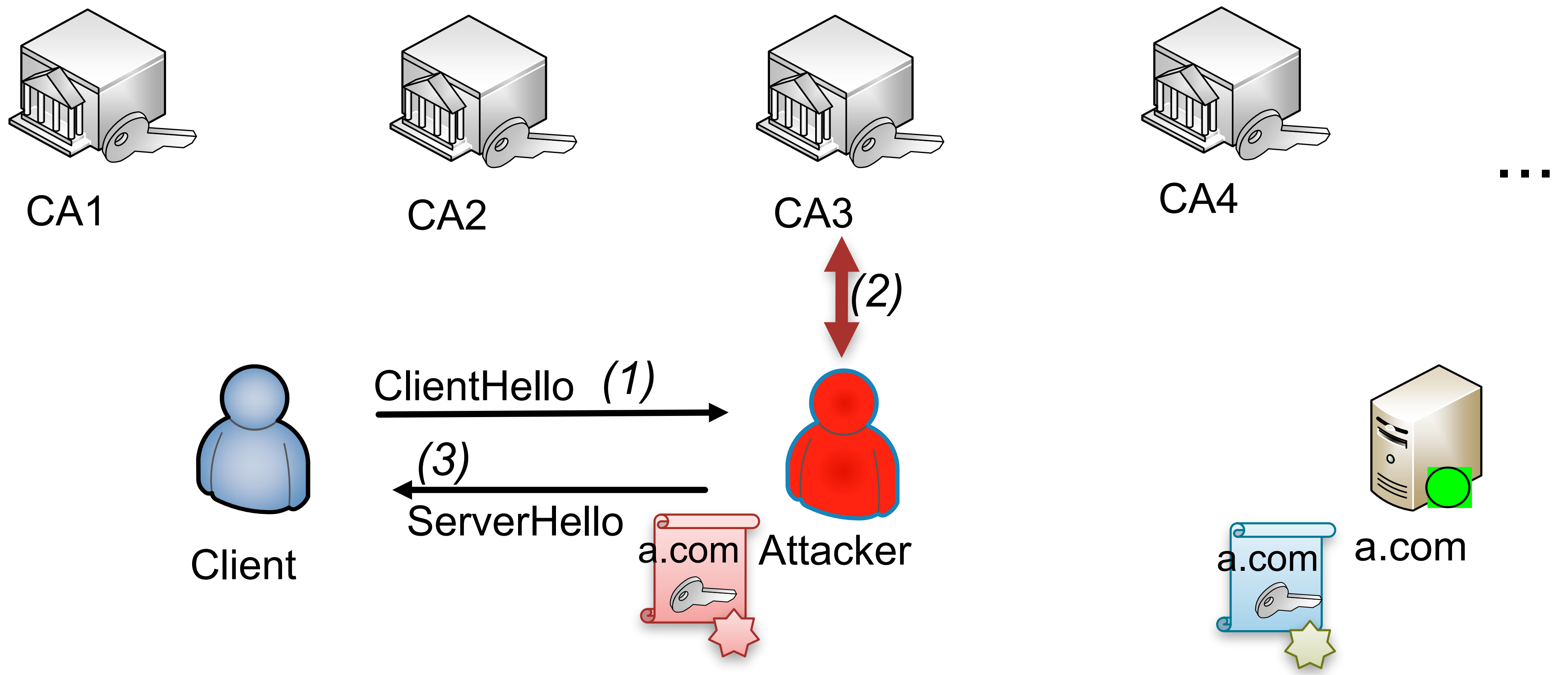
Man-In-The-Middle attack:



Problem with current SSL/TLS PKI: Weak certificate authentication

- Certificates signed by single CA
  - Currently, cannot sign certificate by multiple CAs
- Weakest-link security with too many *trusted* entities
  - Current browsers trust ~1500 keys that can issue valid certificates

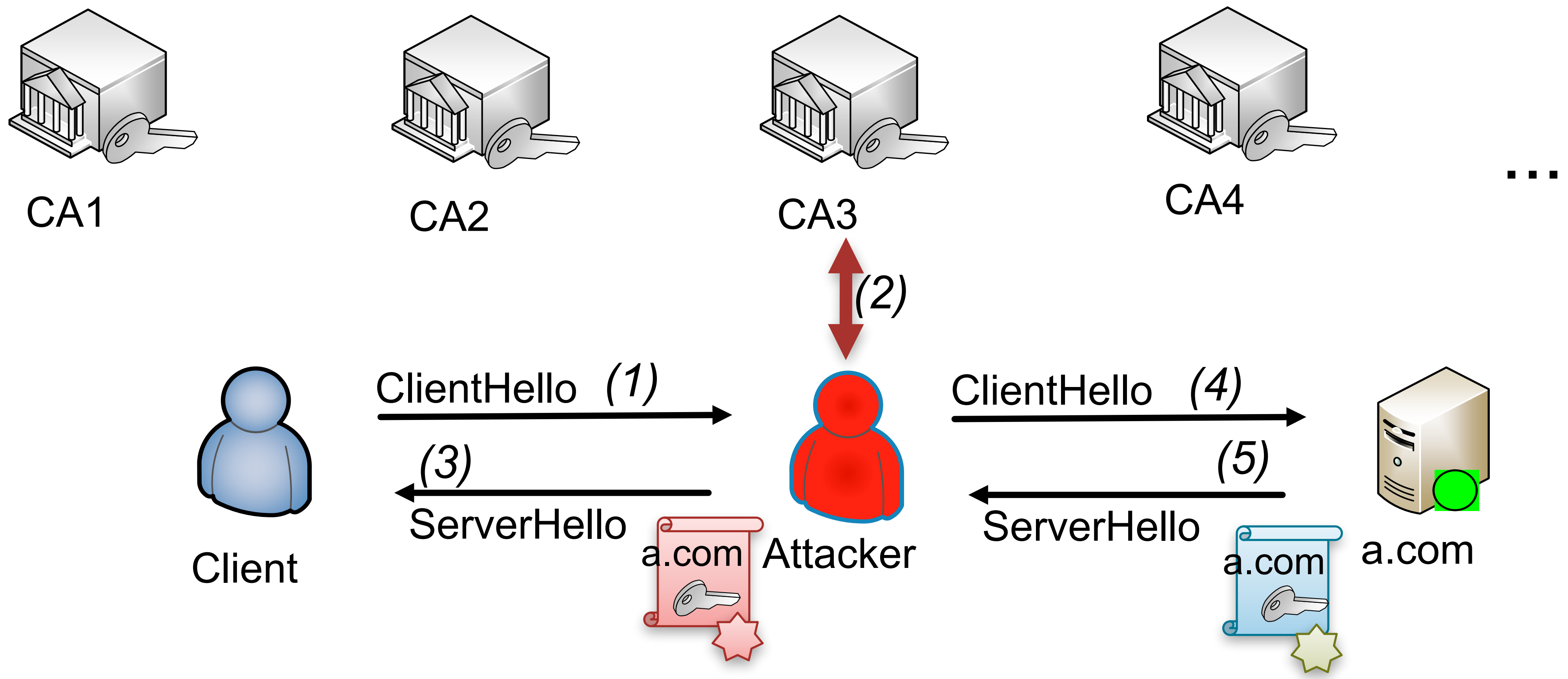
Man-In-The-Middle attack:



Problem with current SSL/TLS PKI: Weak certificate authentication

- Certificates signed by single CA
  - Currently, cannot sign certificate by multiple CAs
- Weakest-link security with too many *trusted* entities
  - Current browsers trust ~1500 keys that can issue valid certificates

Man-In-The-Middle attack:



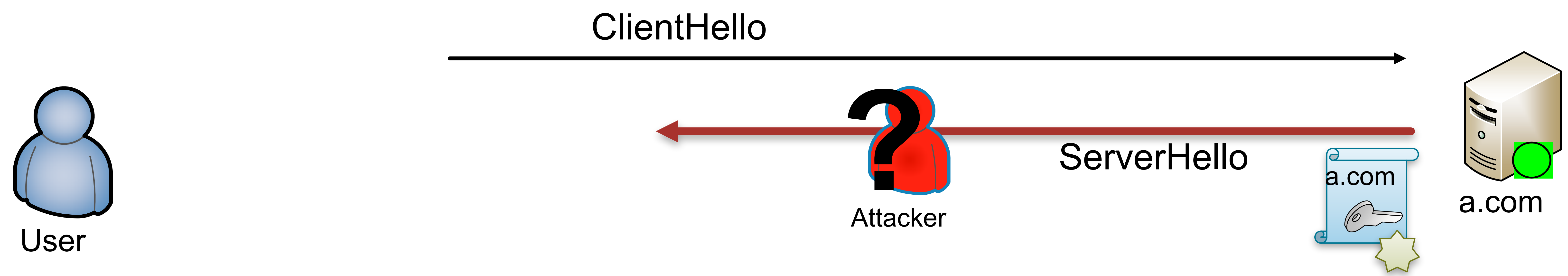
Problems with current SSL/TLS PKI

Problems with current SSL/TLS PKI

- Weakest-link security
- Revocation system is insecure and inefficient
  - Various schemes
  - Some CAs are *too-big-to-fail*
- Trust agility
  - Domains cannot state which CAs are trusted
- Transparency
  - CAs' actions are not transparent
- Imbalance
  - CAs have almost unlimited power
- Misconfigurations
  - SSLv2, weak crypto, NULL cipher suites

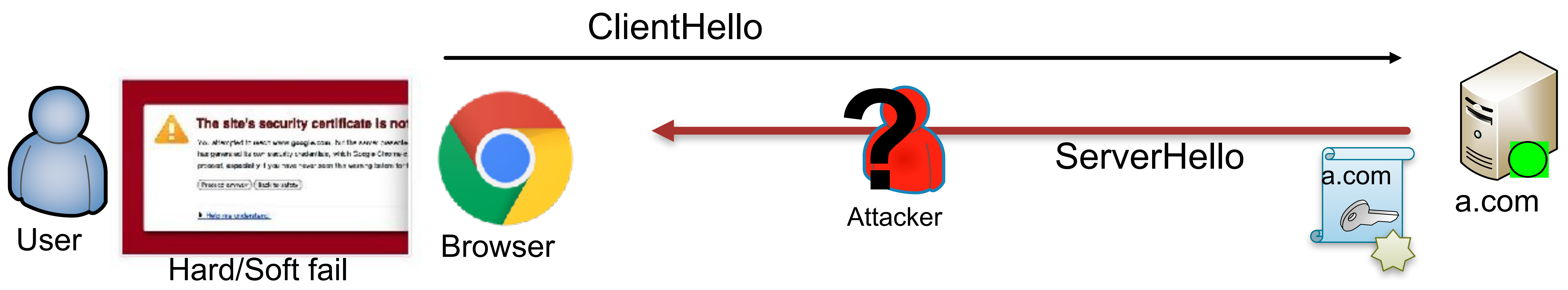
Problems with current SSL/TLS PKI: Security warnings and error handling

- Drawbacks of TLS error handling by browsers and users
  - Users prefer to ignore errors and visit web sites
  - Browsers prefer to avoid *hard fail* to cater to users
  - However *hard fail* is the only effective protection against an attack!
  - **Observation**: Domain should decide on error handling



Problems with current SSL/TLS PKI: Security warnings and error handling

- Drawbacks of TLS error handling by browsers and users
  - Users prefer to ignore errors and visit web sites
  - Browsers prefer to avoid *hard fail* to cater to users
  - However *hard fail* is the only effective protection against an attack!
  - **Observation**: Domain should decide on error handling

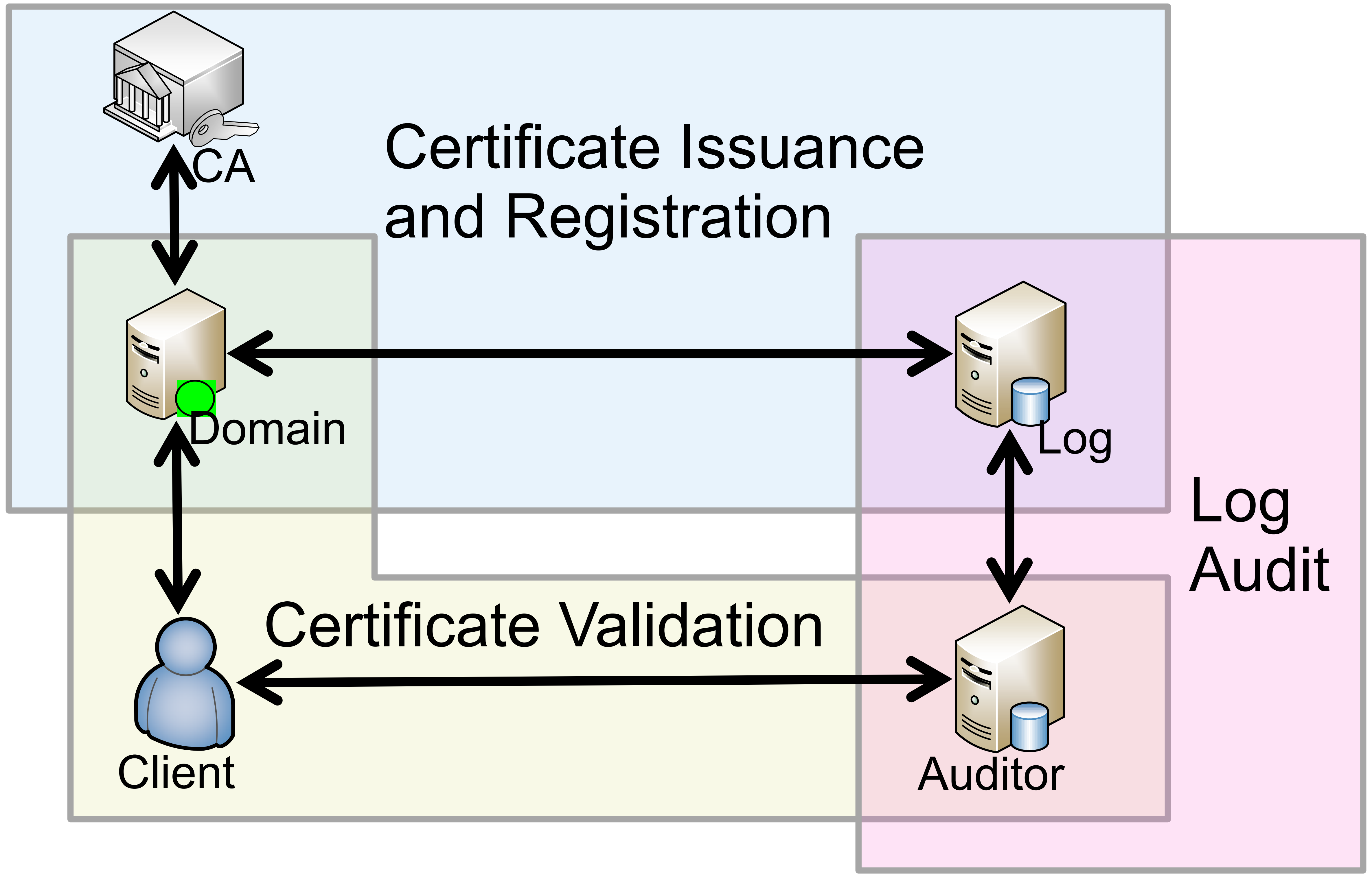


PoliCert: Secure and Flexible TLS Certificate Management [CCS'14]

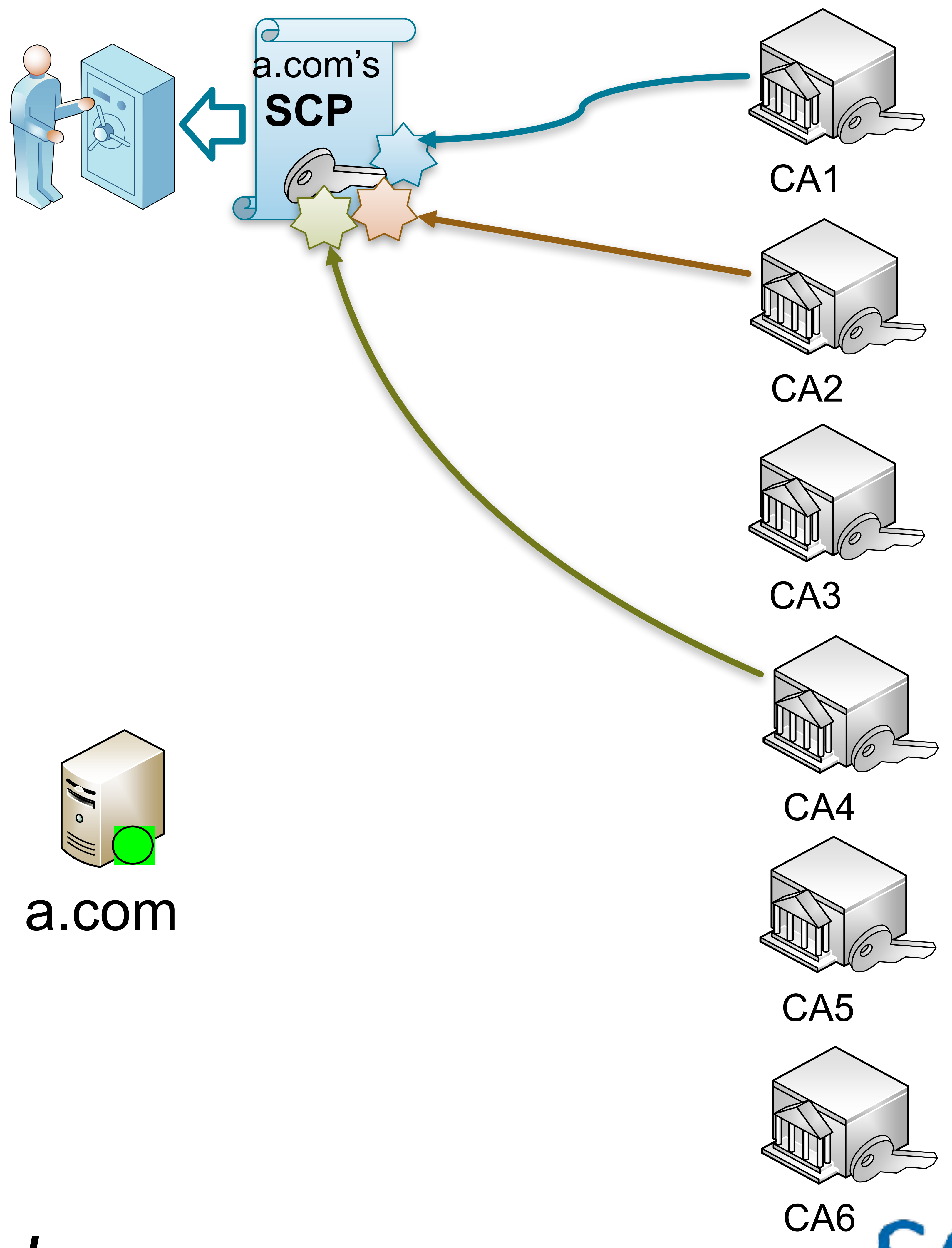
- Observation: many problems can be solved when domains can express their own security policies
  - Many domains have multiple certificates (and servers) and want to ensure consistent policy across all certificates (and servers)
  - Desire to enforce security policy for all subdomains
- **PoliCert** allows domains to express security policies (certificates, connections, policy inheritance rules for subdomains, and TLS error handling controls)
  - Subject Certificate Policy (**SCP**) – infrequently updated
  - Multi Signature Certificate (**MSC**) – frequently updated
- How to create and make policies accessible?

PoliCert: Parties

- Clients/CAs/Domains as today
- Logs are public and highly available
- Auditors monitor Logs

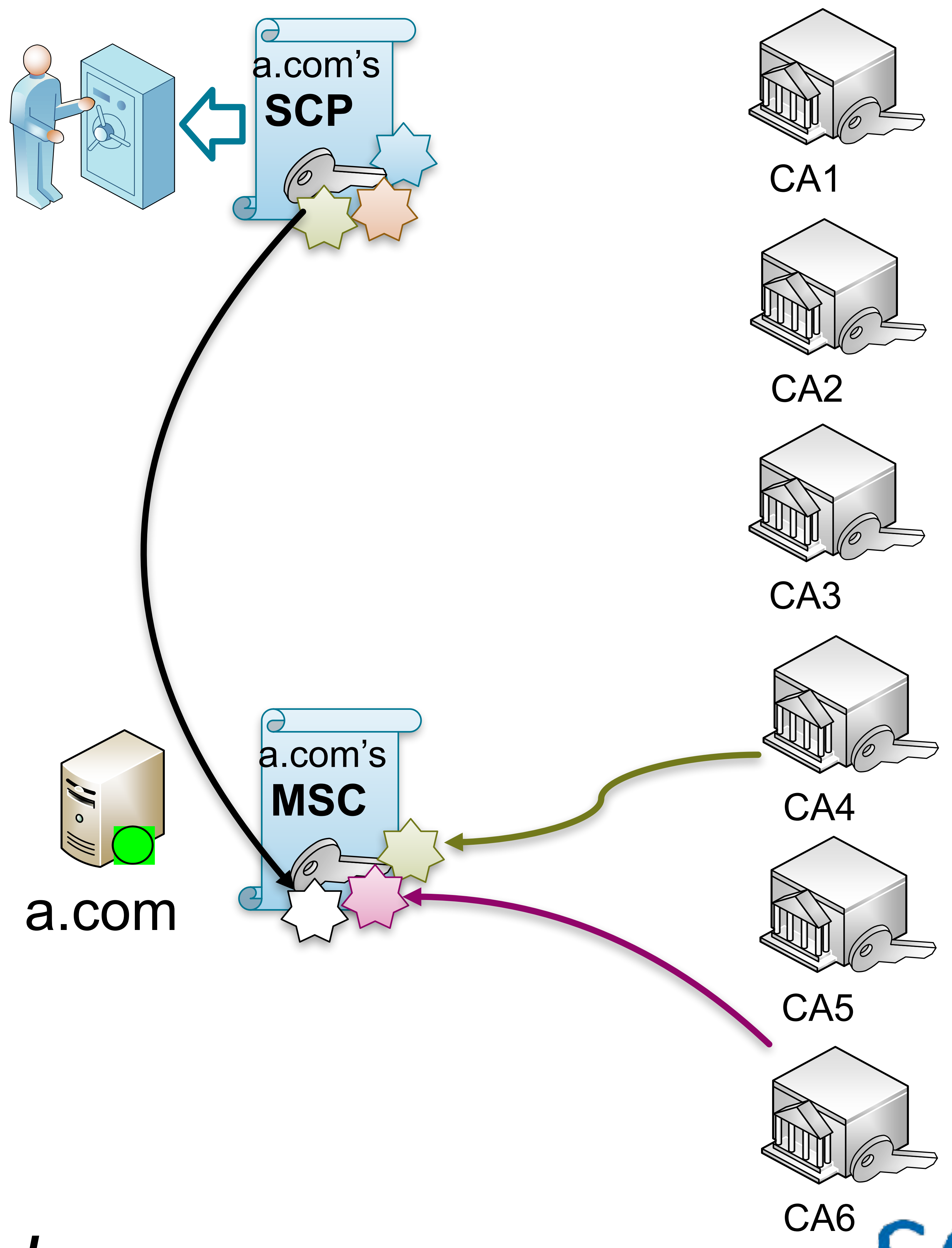


SCP and MSC Creation



- SCP (one per domain):
  - Used for management
  - Signed by long-term CAs' keys
  - Describes MSCs and connections:
    - Who is trusted by Domain (list of trusted CAs and Logs)?
    - When should MSC be accepted?
    - Security parameters of connection
    - Failure scenario (errors handling)
    - Inheritance (to enforce subdomains)
    - How can SCP be updated?
  - SCP's key can be stored off-line
- MSC (many per domain):
  - Used for TLS connection setup
  - Must be signed by SCP's key

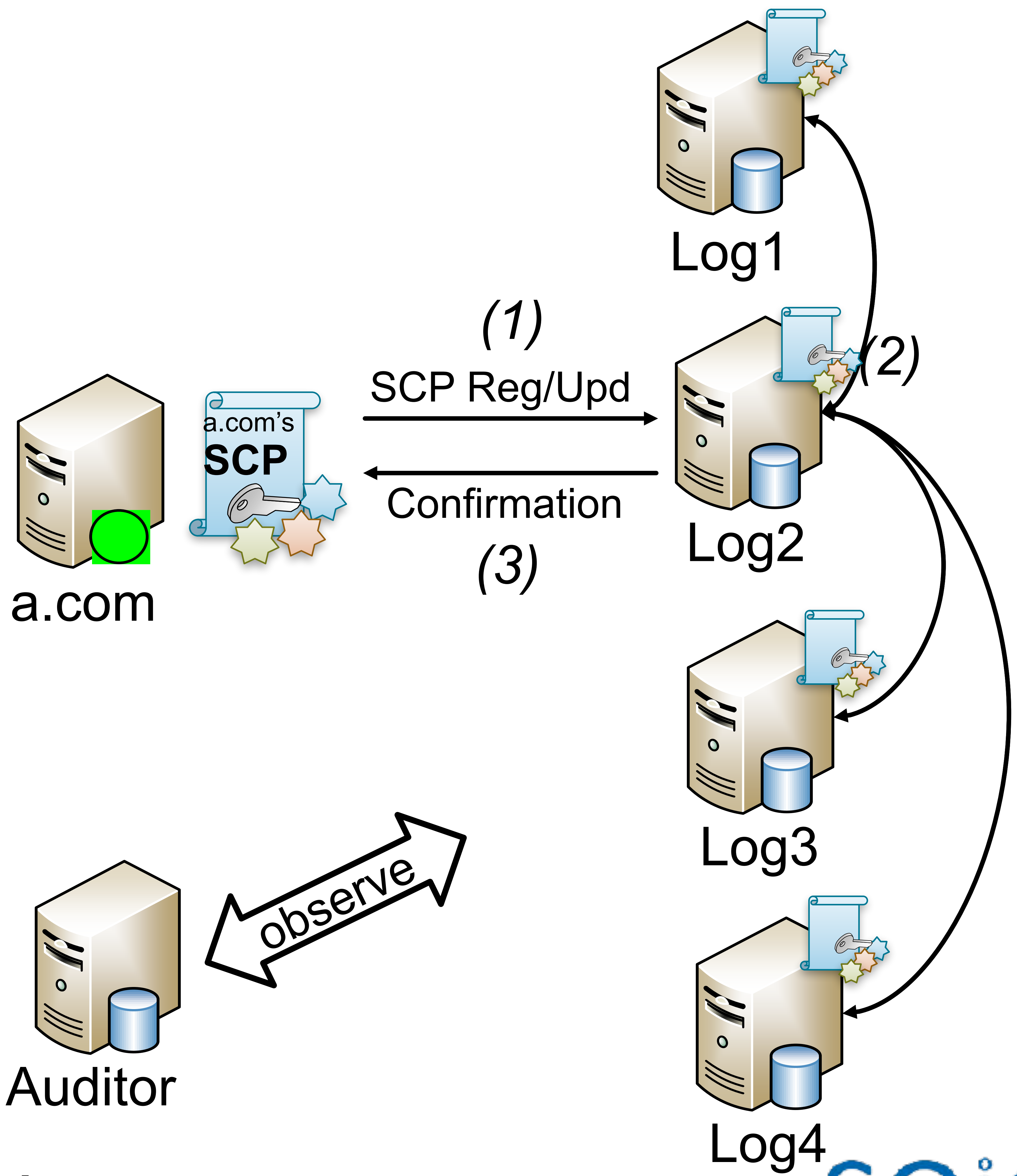
SCP and MSC Creation



- SCP (one per domain):
  - Used for management
  - Signed by long-term CAs' keys
  - Describes MSCs and connections:
    - Who is trusted by Domain (list of trusted CAs and Logs)?
    - When should MSC be accepted?
    - Security parameters of connection
    - Failure scenario (errors handling)
    - Inheritance (to enforce subdomains)
    - How can SCP be updated?
  - SCP's key can be stored off-line
- MSC (many per domain):
  - Used for TLS connection setup
  - Must be signed by SCP's key

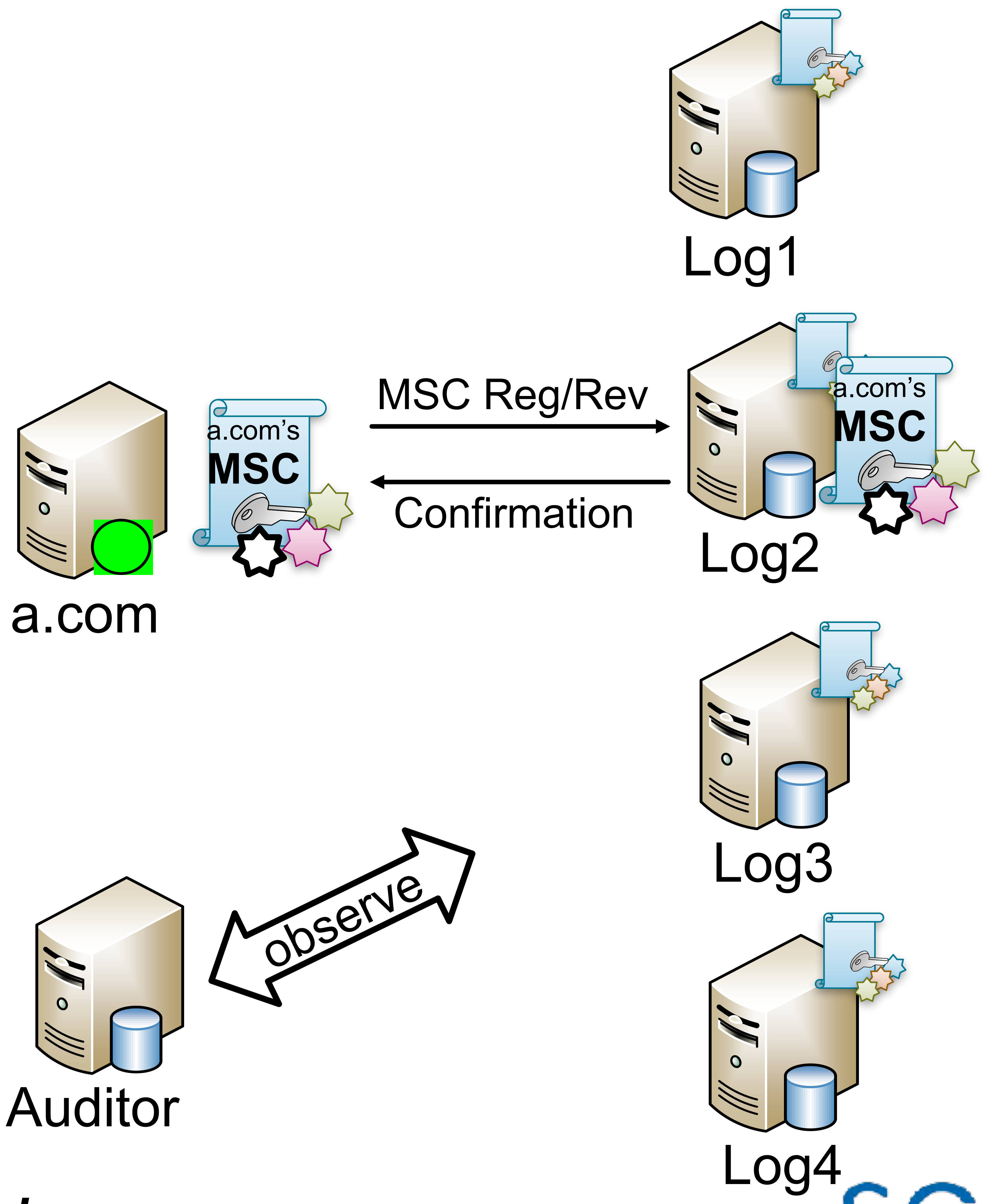
ETH zürich SCION

SCP Registration and Update



- Registration and update are synchronized among Logs (these operations are infrequent)
- Update must be be compliant with update parameters of current SCP

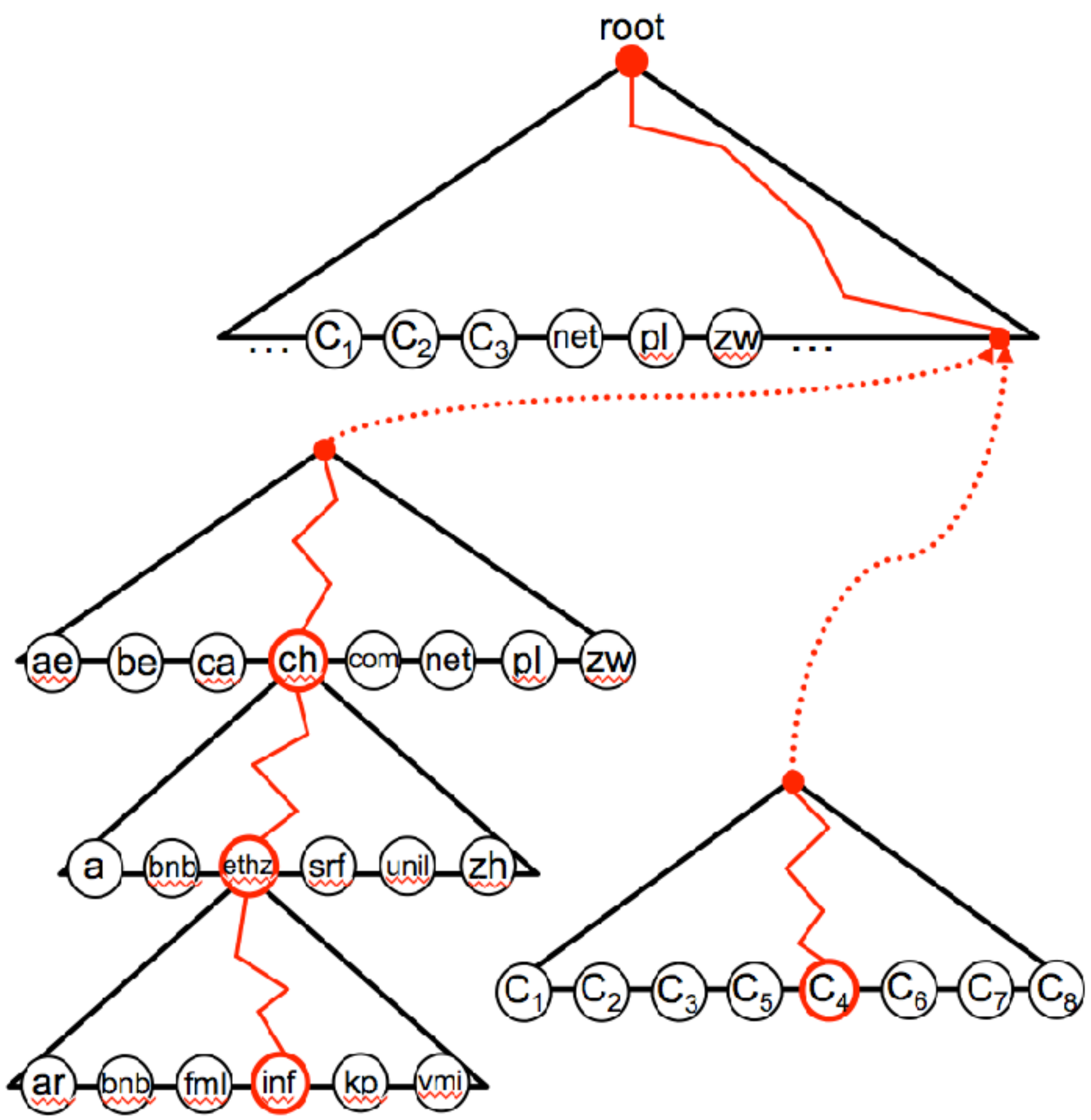
MSC Registration and Revocation



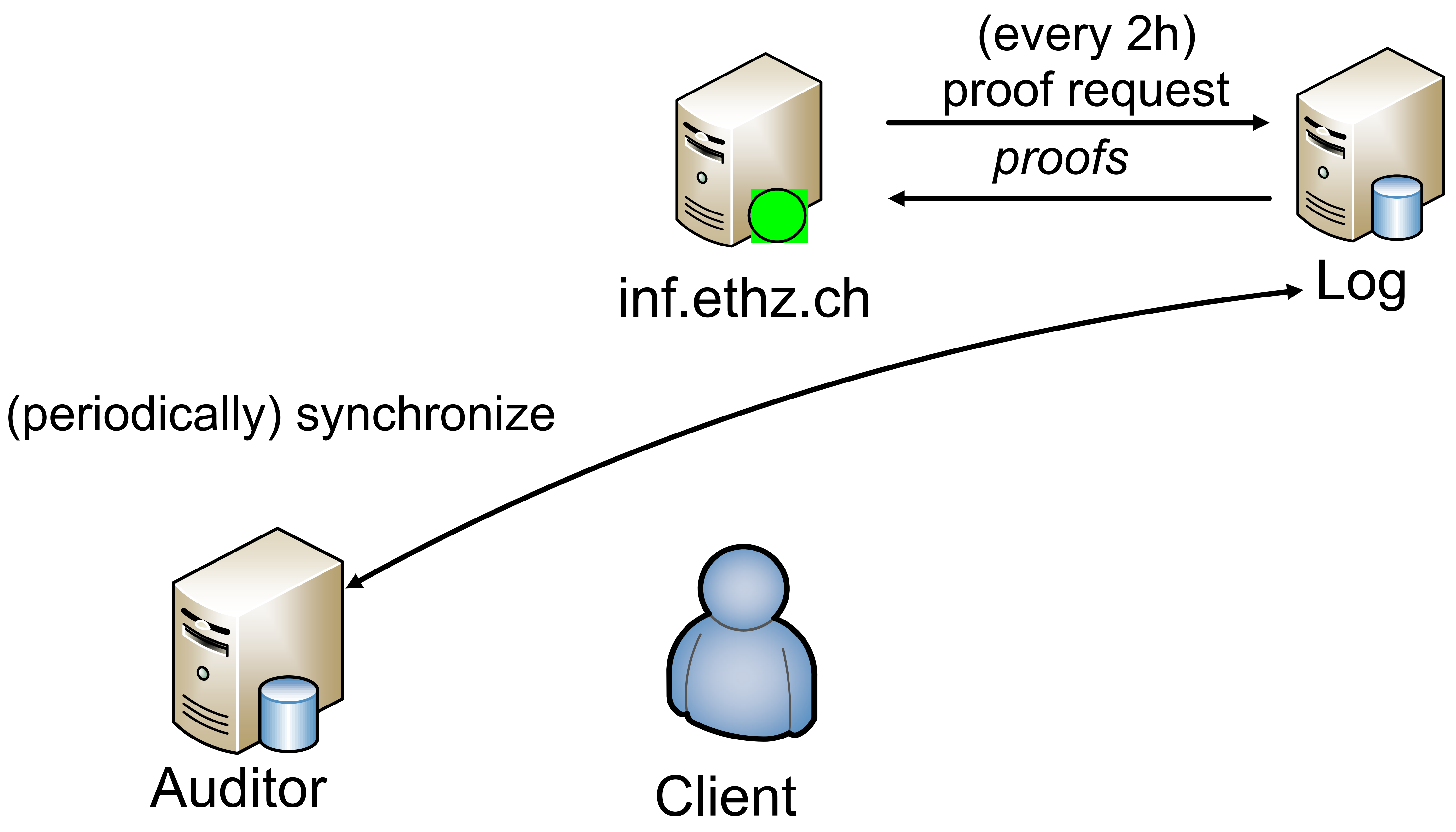
- Registration and revocation does not require any synchronization

Append-Only Log

- Log (on demand) can prove:
  - What is current SCP for a Domain
  - That MSC is logged and (not) revoked
  - That one snapshot of the log is an extension of another

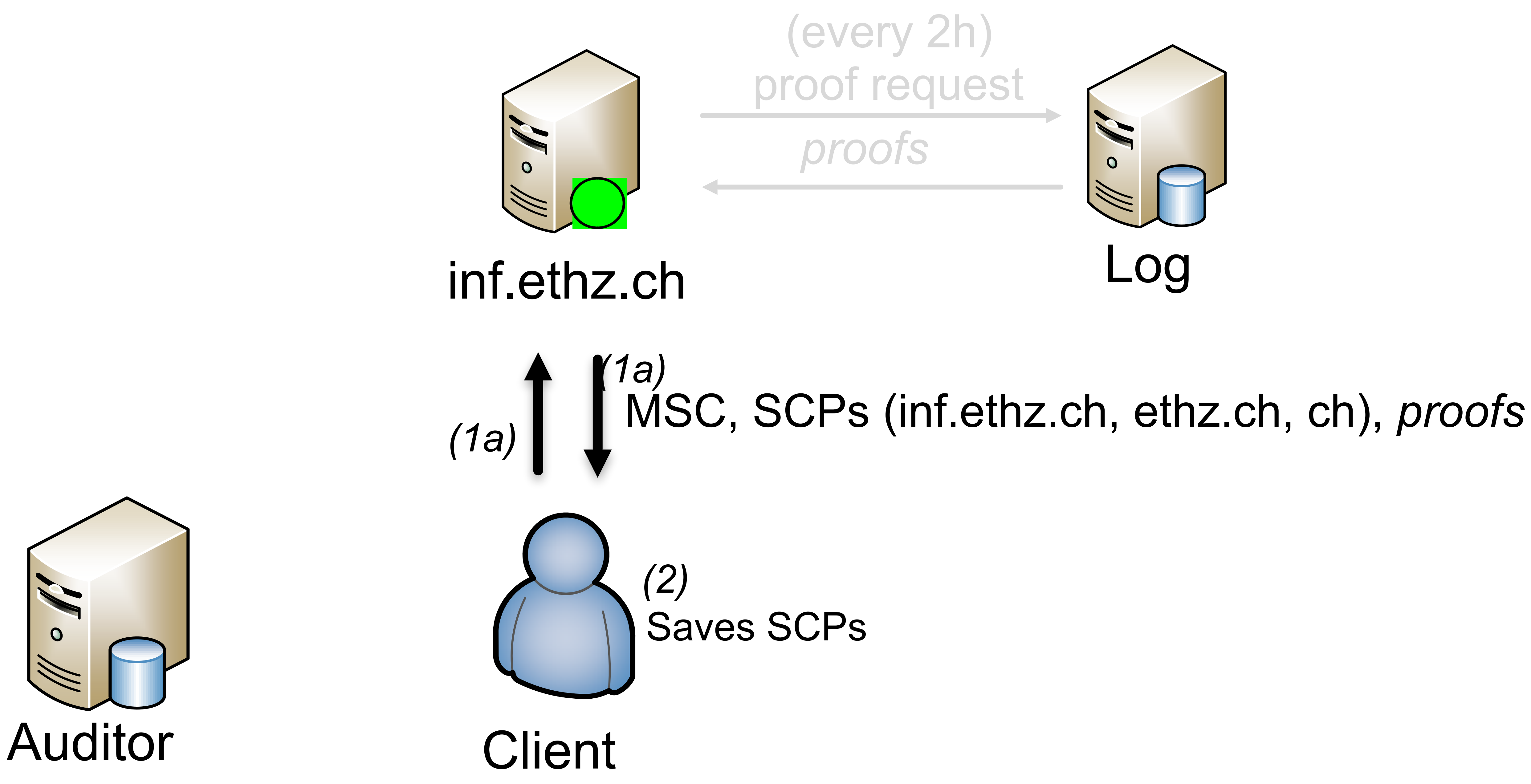


MSC validation



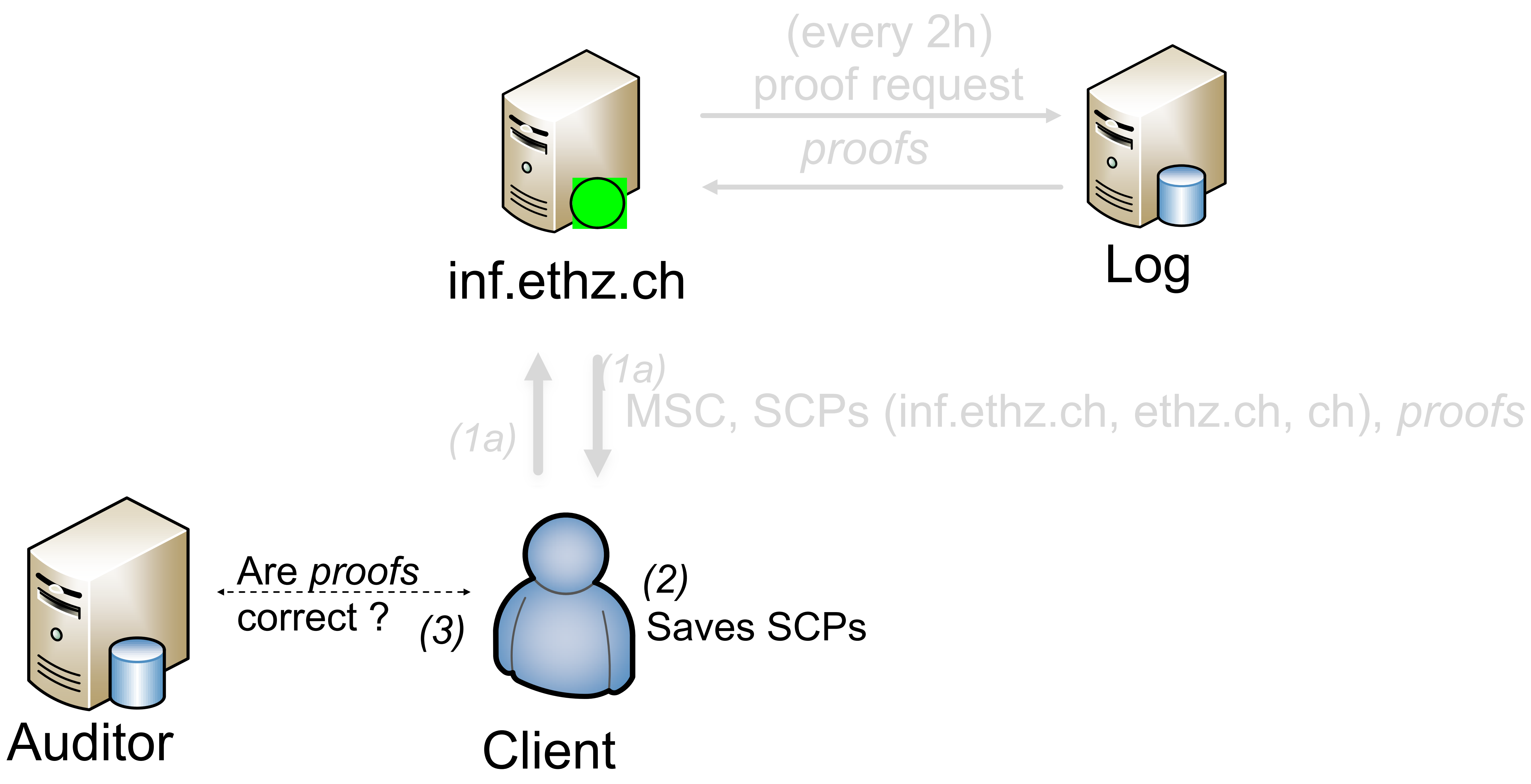
- Client checks if:
  - MSC and SCP are logged
  - MSC is not revoked
  - MSC is compliant with SCPs
- Client can contact Auditor to verify Log’s proofs

MSC validation



- Client checks if:
  - MSC and SCP are logged
  - MSC is not revoked
  - <u>MSC is compliant with SCPs</u>
- Client can contact Auditor to verify Log's proofs

MSC validation



- Client checks if:
  - MSC and SCP are logged
  - MSC is not revoked
  - MSC is compliant with SCPs
- Client can contact Auditor to verify Log's proofs

Parameters Inheritance

- SCPs can have parameters that are inherited by subdomains (i.e., subdomains have to adhere to them)
- In case of inheritance parameter can only be changed if it makes the parameter ***more secure***

inf.ethz.ch's policy		ethz.ch's policy		ch's policy
CA={A,B,C,D,E} SSL_SEC=Low ...	⇨	***CA**={A,B,C,D} ***SSL_SEC**=High ...	⇨	***CA**={B,C,D,E,F,G} ***SSL_SEC**=Medium ...

CA – list of trusted CAs
SSL_SEC – minimum security level of SSL/TLS connection
***PARAM** – value is inherited by subdomains

Parameters Inheritance

- SCPs can have parameters that are inherited by subdomains (i.e., subdomains have to adhere to them)
- In case of inheritance parameter can only be changed if it makes the parameter ***more secure***

inf.ethz.ch's policy

CA={A,B,C,D,E}
SSL_SEC=Low
...

ethz.ch's policy

***CA**={A,B,C,D}
***SSL_SEC**=High
...

ch's policy

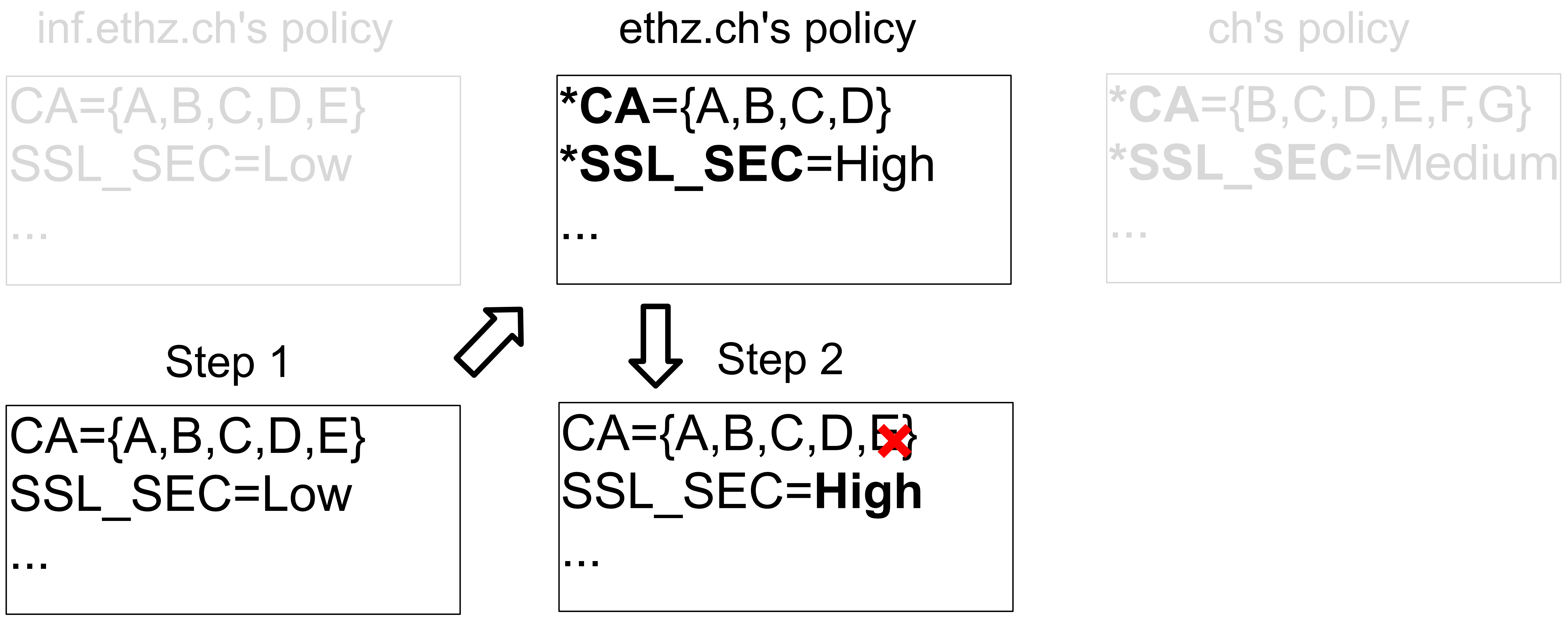
***CA**={B,C,D,E,F,G}
***SSL_SEC**=Medium
...

Step 1 ⇩

CA={A,B,C,D,E}
SSL_SEC=Low
...

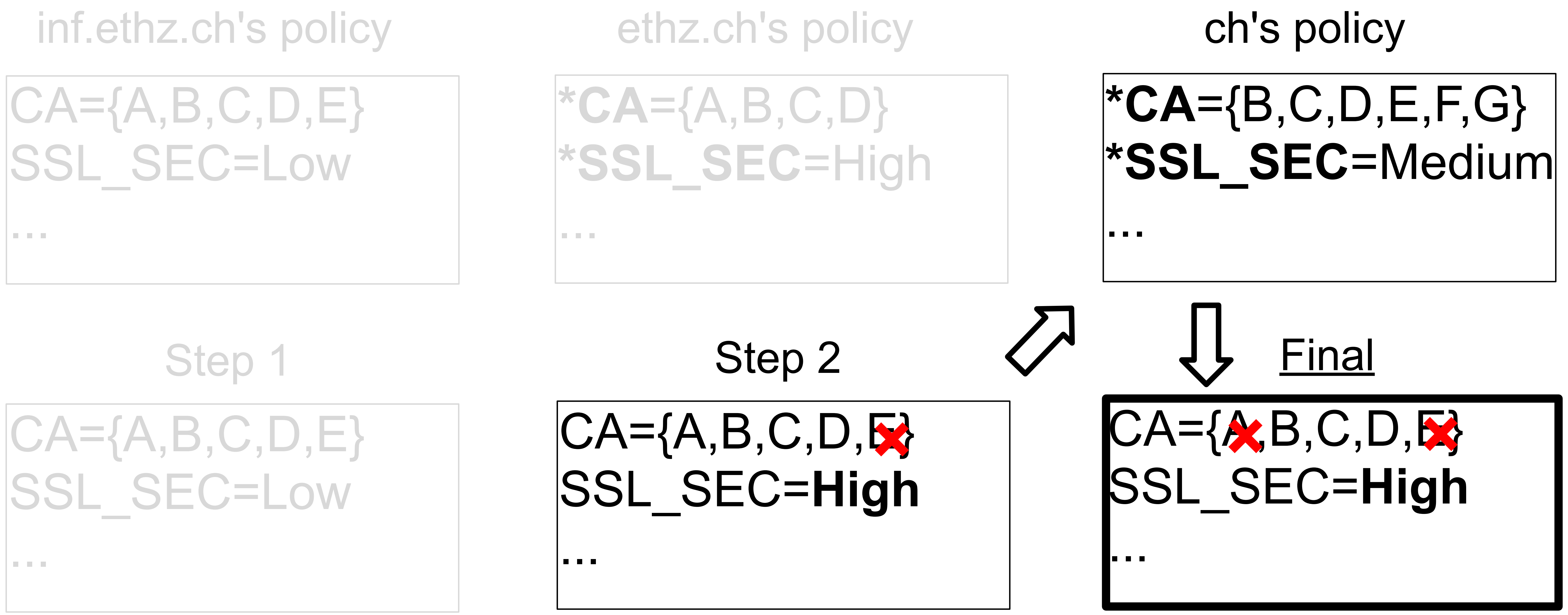
Parameters Inheritance

- SCPs can have parameters that are inherited by subdomains (i.e., subdomains have to adhere to them)
- In case of inheritance parameter can only be changed if it makes the parameter ***more secure***



Parameters Inheritance

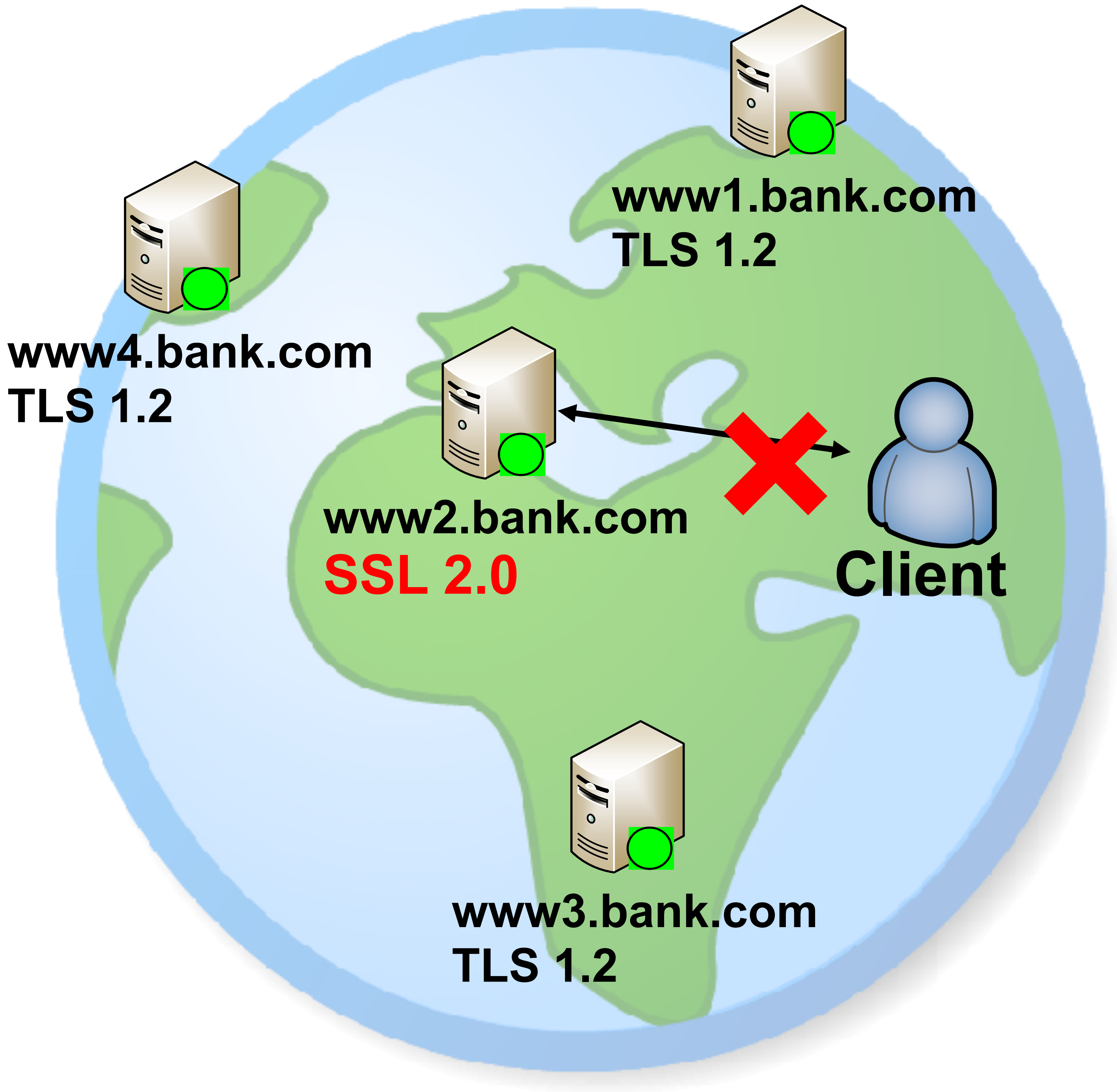
- SCPs can have parameters that are inherited by subdomains (i.e., subdomains have to adhere to them)
- In case of inheritance parameter can only be changed if it makes the parameter ***more secure***



Use Cases

bank.com's policy

***SSL_SEC**=High
***FAIL_SSL_SEC**=Hard
...



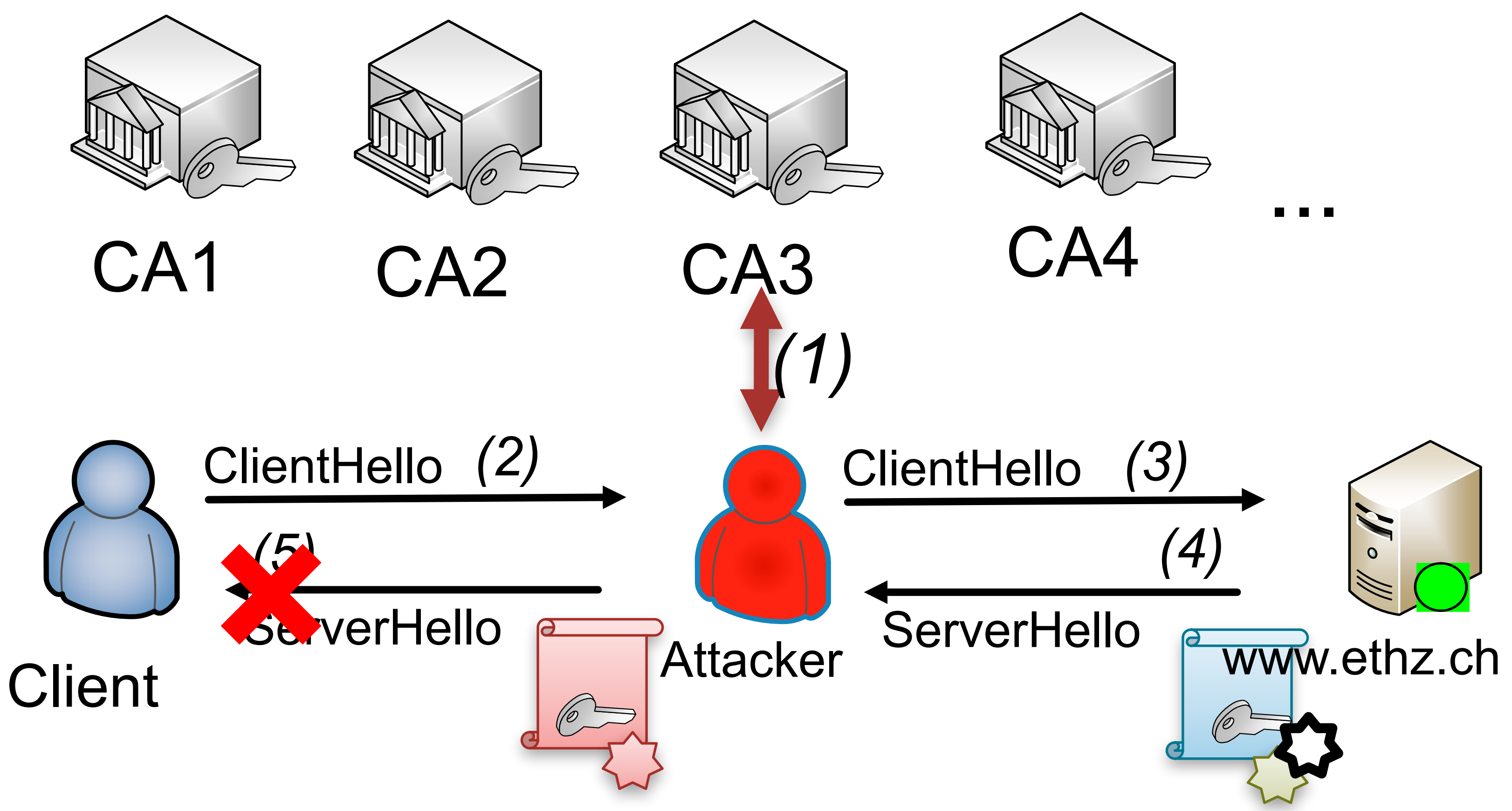
Use Cases

bank.com's policy

*SSL_SEC=High
*FAIL_SSL_SEC=Hard
…

www.ethz.ch's policy

CA={CA1, CA2, CA4}
…



Properties

- Transform weakest-link security into security of the selected trust roots
  - Multi-Signature Certificates (MSCs) by default instead of single weakest link
  - Impossible to create valid MSC without SCP's private key (offline)
- Expressiveness and trust agility
  - Control over certificates, connections, and error handling
  - Only selected entities are trusted, and all entities are verifiable
- Transparency
  - Policies, certificates, and revocations are logged
  - Potential attacks would be visible

Implementation

- SSL/TLS is unmodified
- SCPs and MSCs are implemented as concatenation of standard certificates
- Optimizations (SCPs' caching, MSC/SCP compression)
- Performance:

Log's side:

SCP registration/update:	**10*ms***
MSC registration:	**7*ms***
MSC revocation:	**5*ms***
Proof request:	**9*ms***

Browser's side:

Complete validation: **3ms**

Legacy certificate's validation in similar setting takes 0.7ms

Incremental deployment

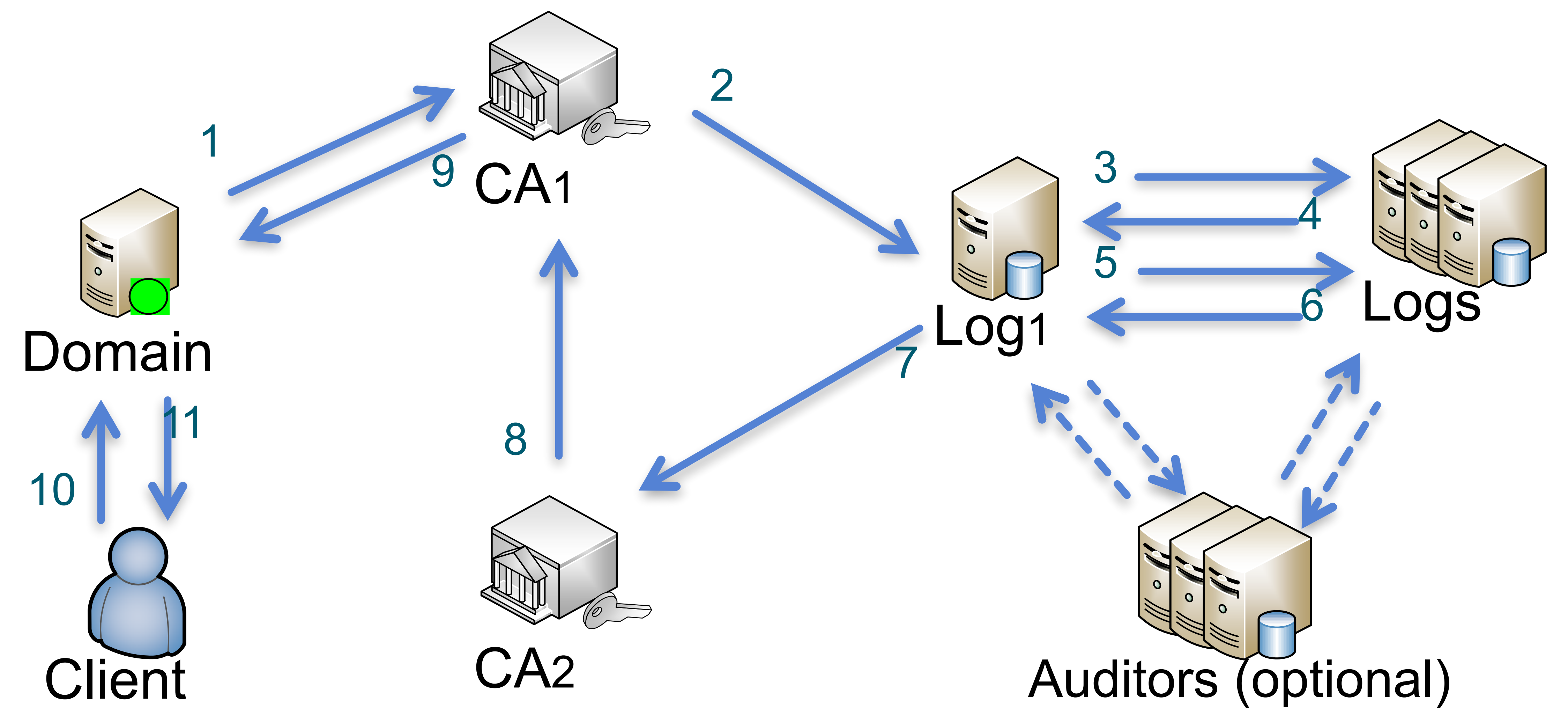
- Participants get benefits
- Others have no disadvantage
- One policy can cover all subdomains
- CAs without any changes
- MSC’s implementation works with legacy software

Remaining Challenges

- Corner cases: two compromised parties are enough to launch a successful attack
  - An adversary is able to compromise a CA and a log at the same time, and
  - the attacked client visits the targeted website for the first time.
- Protection from and detection of compromised logs
  - How to protect clients when logs and CAs are compromised?
  - How to make sure that logs behave correctly?
  - Currently auditors can only detect attacks (cannot prevent them)

ARPKI: Attack Resilient PKI [CCS'14, TDSC'16]

- Resilience for n-1 compromised entities
  - n is a parameter (security vs. efficiency)
- Message flow with CAs active in "on-line" actions
- Confirming is extended to n parties (one party is log and n-1 parties are different CAs)
- Co-design: formal specification and implementation are developed from a single design document



ARPKI: Operations

PWCert Generation

A : Set extensions, contact trusted CAs
: Combine multiple certificates into PWCert_A

PWCert Registration Request

1. $A \rightarrow CA_1$: $\text{RegReq} = \{PWCert_A, CA_1, ILS_1, CA_2\}_{K_A^{-1}}$
2. CA_1 : Verify signatures in RegReq
: Ensure $CA_1 \in \text{PWCert}_A$'s CA_LIST
: Add PWCert_A into a pending request list

$CA_1 \rightarrow ILS_1$: RegReq

ILS Synchronization

3. ILS_1 : Verify signatures in RegReq
: Ensure $ILS_1 \in \text{PWCert}_A$'s ILS_LIST
: Ensure ILS_1, CA_1, and CA_2 are different entities
: Ensure no PWCert was registered for A's domain

$ILS_1 \rightarrow ILS_n$: $\text{SynReq} = \{\text{RegReq}\}_{K_{ILS_1}^{-1}}$

4. ILS_n : Verify signatures in RegReq
: Ensure no PWCert was registered for A's domain

$ILS_n \rightarrow ILS_1$: $\text{SynResp} = \{H(\text{RegReq})\}_{K_{ILS_n}^{-1}}$

5. ILS_1 : Collect SynResp from at least a quorum of ILSes

$ILS_1 \rightarrow ILS_n$: $\text{SynCommit} = \{H(\text{RegReq})\}_{K_{ILS_1}^{-1}}$

6. $ILS_n \rightarrow ILS_1$: $\text{SynAck} = \{H(\text{RegReq})\}_{K_{ILS_n}^{-1}}$

Registration Confirmation

7. ILS_1 : Collect SynAck from at least a quorum of ILSes
: $\text{Accept} = \{H(PWCert_A)\}_{K_{ILS_1}^{-1}}$

$ILS_1 \rightarrow CA_2$: $\text{RegResp} = \{\text{Accept}, \text{RegReq}, List(\text{SynAck})\}_{K_{ILS_1}^{-1}}$

8. CA_2 : Verify signatures in RegResp
: Ensure $CA_2 \in \text{PWCert}_A$'s CA_LIST
: Ensure ILS_1, CA_1, and CA_2 are different entities

$CA_2 \rightarrow CA_1$: $\text{RegConf} = \{\{\text{Accept}\}_{K_{CA_2}^{-1}}, List(\text{SynAck})\}_{K_{CA_2}^{-1}}$

9. CA_1 : Verify signatures in RegConf
: Ensure ILS_1, CA_1, and CA_2 are different entities
: Remove PWCert_A from the pending request list

$CA_1 \rightarrow A$: $\{\{\text{Accept}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

A : Ensure ILS_1, CA_1, and CA_2 are different entities

TLS Connection

10. $C \rightarrow A$: TLS connection request
11. $A \rightarrow C$: PWCert_A, $\{\{\text{Accept}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

Update PWCert Generation

A : Set extensions for new key, contact trusted CAs
: Combine multiple certificates into a $\text{PWCert}_A{}'$

ILS PWCert Request

1. $A \rightarrow CA_1$: $\text{UpdateReq} = \{PWCert_A{}', CA_1, ILS_1, CA_2\}_{K_A^{-1}{}'}$
2. $CA_1 \rightarrow CA_2$: UpdateReq

ILS Synchronization

3. $ILS_1 \rightarrow ILS_n$: $\text{SynReq} = \{\text{UpdateReq}\}_{K_{ILS_1}^{-1}}$
4. $ILS_n \rightarrow ILS_1$: $\text{SynResp} = \{H(\text{UpdateReq})\}_{K_{ILS_n}^{-1}}$

Update Confirmation

7. ILS_1 : $\text{Accept} = \{H(PWCert_A{}'), T\}_{K_{ILS_1}^{-1}}$

$ILS_1 \rightarrow CA_1$: $\text{UpdateResp} = \{\text{Accept}, \text{UpdateReq}, List(\text{SynAck})\}_{K_{ILS_1}^{-1}}$

8. $CA_2 \rightarrow CA_1$: $\text{UpdateConf} = \{\text{Accept}_{K_{CA_2}^{-1}}, List(\text{SynResp})\}_{K_{CA_2}^{-1}}$
9. $CA_1 \rightarrow A$: $\{\{\text{Accept}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

TLS Connection

10. $C \rightarrow A$: TLS connection request
11. $A \rightarrow C$: $\text{PWCert}_A{}'$, $\{\{\text{Accept}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

ILS Confirmation Request

1. $A \rightarrow CA_1$: $\text{CCReq} = \{A, CA_1, ILS_1, CA_2\}_{K_A^{-1}}$
2. $CA_1 \rightarrow ILS_1$: CCReq

Proof Generation

7. $ILS_1 \rightarrow CA_2$: $\text{Proof} = \{List(HashVal)\}_{K_{ILS_1}^{-1}}, \{Root\}_{K_{ILS_1}^{-1}}$
8. $CA_2 \rightarrow CA_1$: $\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}$, Proof
9. $CA_1 \rightarrow A$: $\{\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$, Proof

TLS Connection

10. $C \rightarrow A$: TLS connection request
11. $A \rightarrow C$: $PWCert$, $\{\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$, Proof

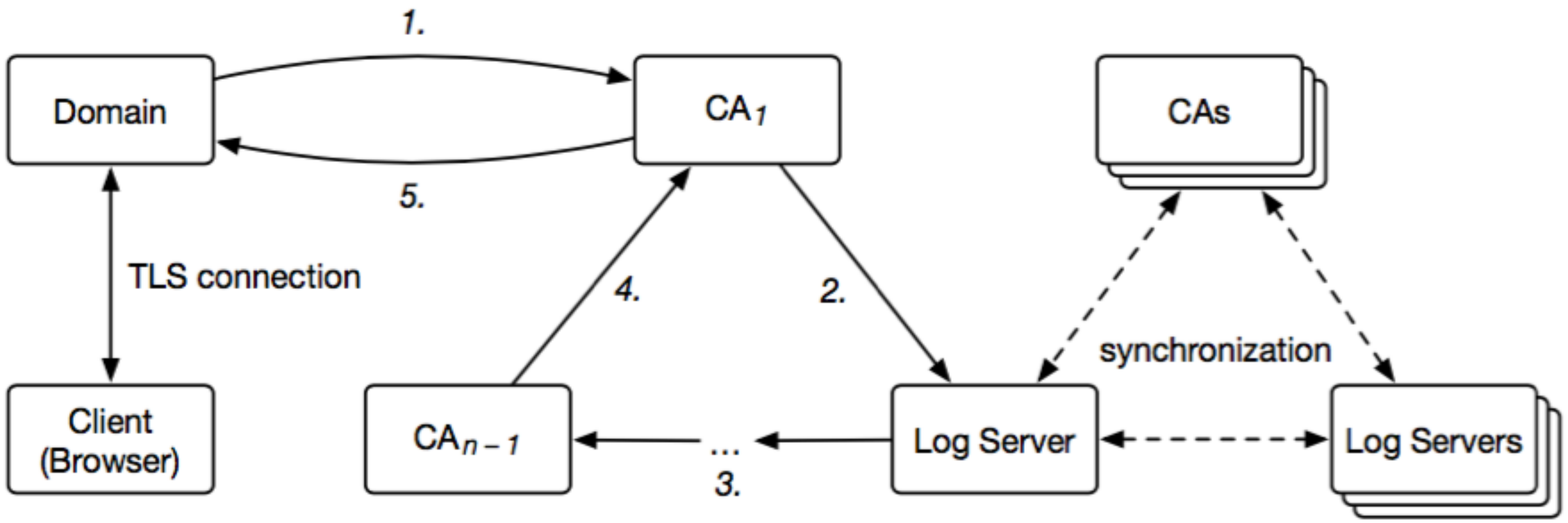
ARPKI: Formal verification

- Proof goal: Whenever (i) a domain A has been registered initially by an honest party with a certificate; and (ii) later a browser accepts a connection to domain A with some certificate (which may have been updated and hence differ from the original certificate), then the adversary does not know the private key for that certificate.
- Tamarin prover
- Full model is about 54000 characters – 23 rules, 1k loc
- 32GB+16 Cores (Xeon 2.7GHz) prove below lemma in 80 min

```
lemma main_prop:
  "( All cid a b reason oldkey key  #i1 #i2 #i3 #i4 .
       ( GEN_LTK(a,oldkey,'trusted') @i1              // 'Honest' agent
       & AskedForPWCert(a,oldkey) @i2                // domain has asked for a PWCert with this exact key
       & ReceivedPWCert(a,oldkey) @i3                // domain has confirmation that its PWCert with this
                                                      //     exact key has been processed.
       & ConnectionAccepted(cid,b,a,reason,key) @i4  // browser accepted connection, based on private key
                                                      //    'key' in for domain a.
       & i3 < i4)
       ==>
       ( (not (Ex #j. K(key) @j)) )                  // adversary cannot know that private key
   ) "
```

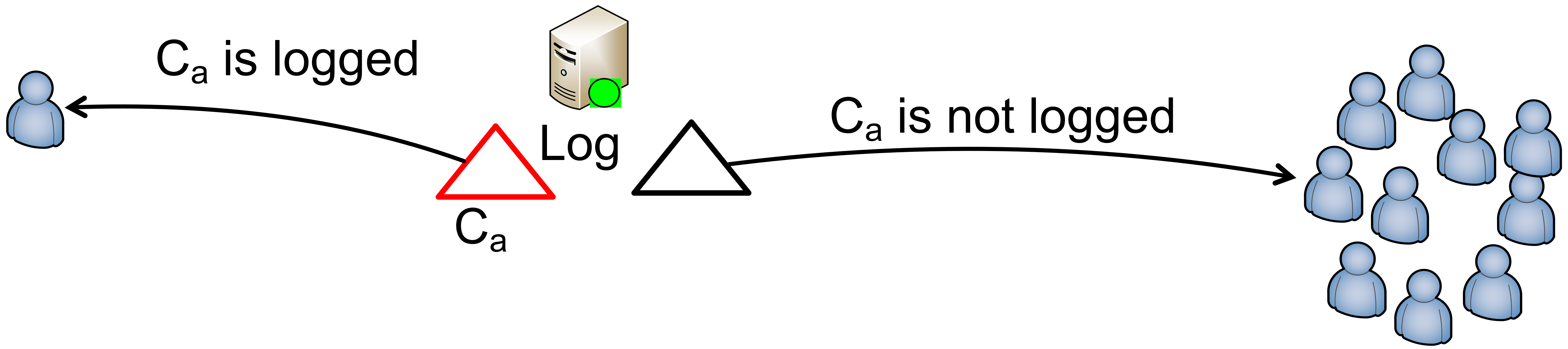
End-entity PKI in SCION

- SCPs confirmed by *n* trusted entities (the parameter is set by each SCION ISD)
  - SCPs have the same properties as certificates in ARPKI
- MSCs logged, non-revoked, and compliant with policies



Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs [CNS'15]

- Misbehavior detection (beyond *n* trusted entities)
  - *Who watches the watchman?* Equivocation attack (compromised PKI)
  - How to detect it?
  - Constraints: scalability, infrastructure, privacy, efficiency, effectiveness

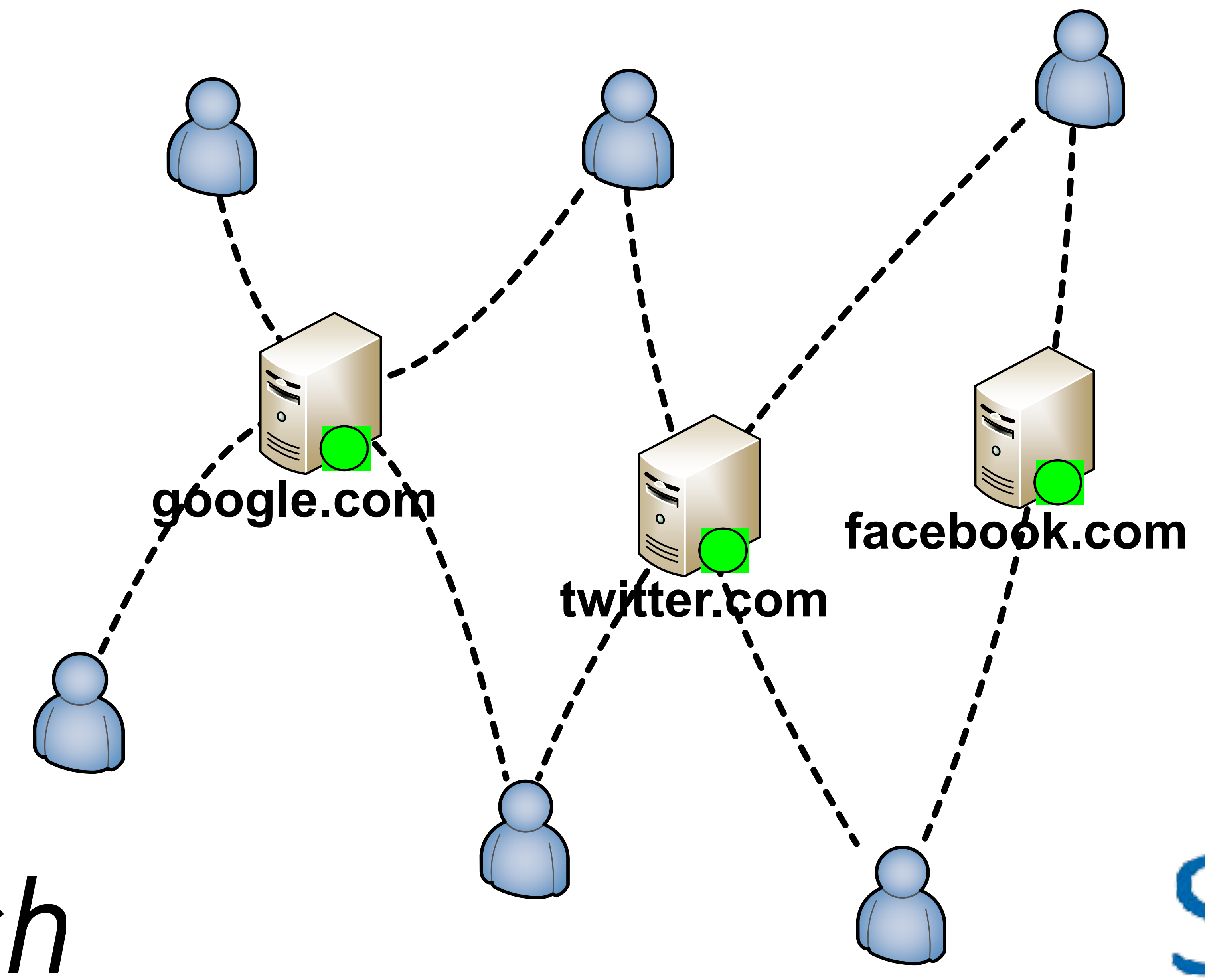


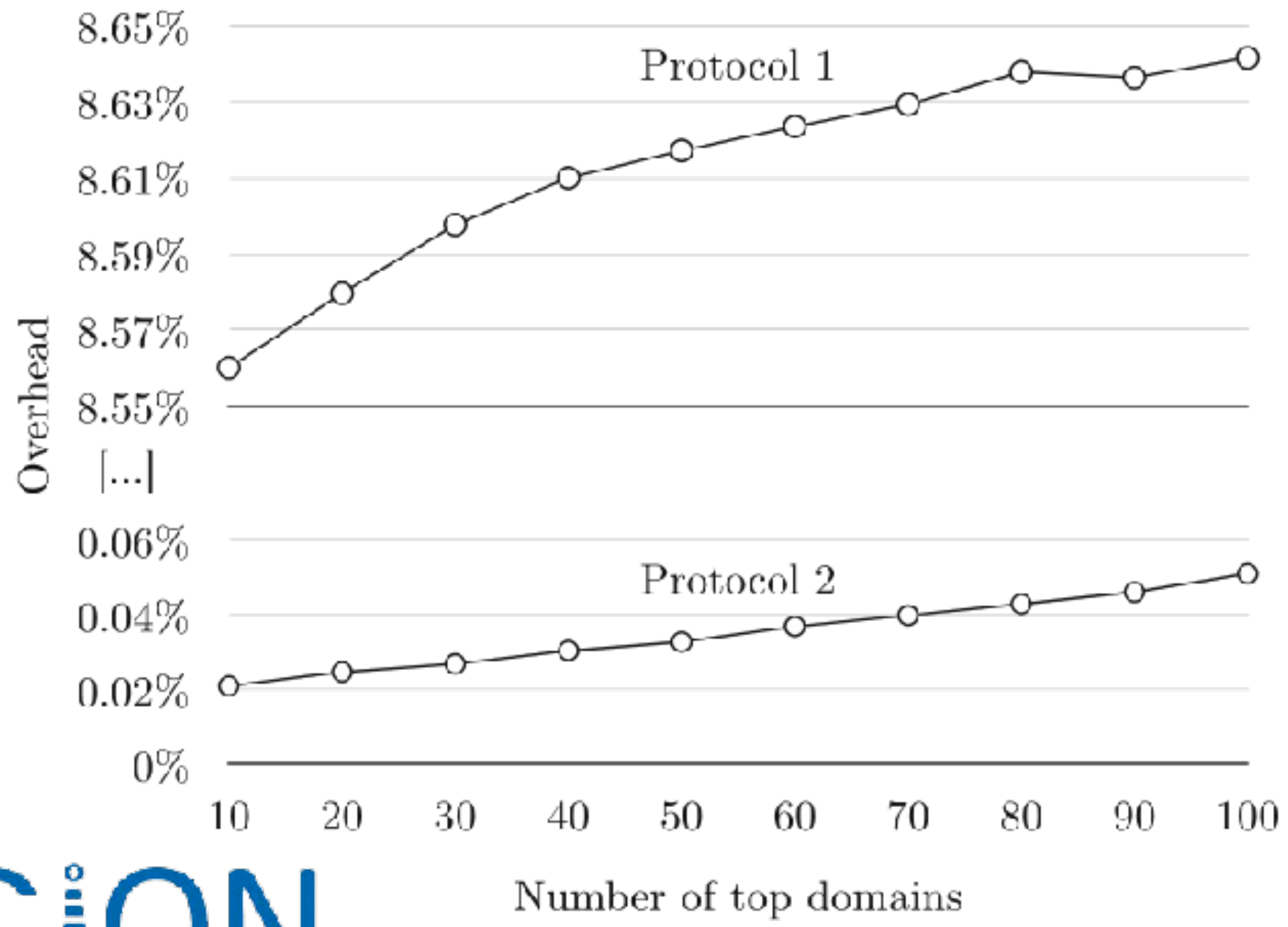
ETH zürich

SCION

Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs [CNS'15]

- Misbehavior detection (beyond *n* trusted entities)
  - *Who watches the watchman?* Equivocation attack (compromised PKI)
  - How to detect it?
  - Constraints: scalability, infrastructure, privacy, efficiency, effectiveness
- **Idea**: Clients exchange information using natural HTTPS traffic





Further Reading

P.Szalachowski, S.Matsumoto, A.Perrig "**PoliCert: Secure and Flexible TLS Certificate Management"**, *In Proc. of the ACM CCS*, 2014

D.Basin, C.Cremers, THJ.Kim, A. Perrig, R.Sasse, P.Szalachowski **„ARPKI: Attack Resilient Public-key Infrastructure."** *In Proc. of ACM CCS,* 2014.

L.Chuat, P.Szalachowski, A.Perrig, B.Laurie, E.Messeri **„Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs"** *In Proc. of IEEE CNS,* 2015

D.Basin, C.Cremers, THJ.Kim, A. Perrig, R.Sasse, P.Szalachowski **„Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure."** *In IEEE TDSC,* 2016

A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat. **„SCION: A Secure Internet Architecture."** Springer, 2017. (Chapter 4)