dRR: A Decentralized, Scalable, and Auditable Architecture for RPKI Repository

NDSS ’24

Yingying Su[∗‡], Dan Li[∗†], Li Chen[†], Qi Li[∗†], and Sitong Ling[∗]

[∗]Tsinghua University, [†]Zhongguancun Laboratory, [‡]BNRist

mhkang 2024-05-01

Resource Public Key Infrastructure (RPKI)

- **RPKI** is an **infrastructure for securing Internet number resources** (e.g., IP prefixes or AS numbers) and improving security of BGP routing

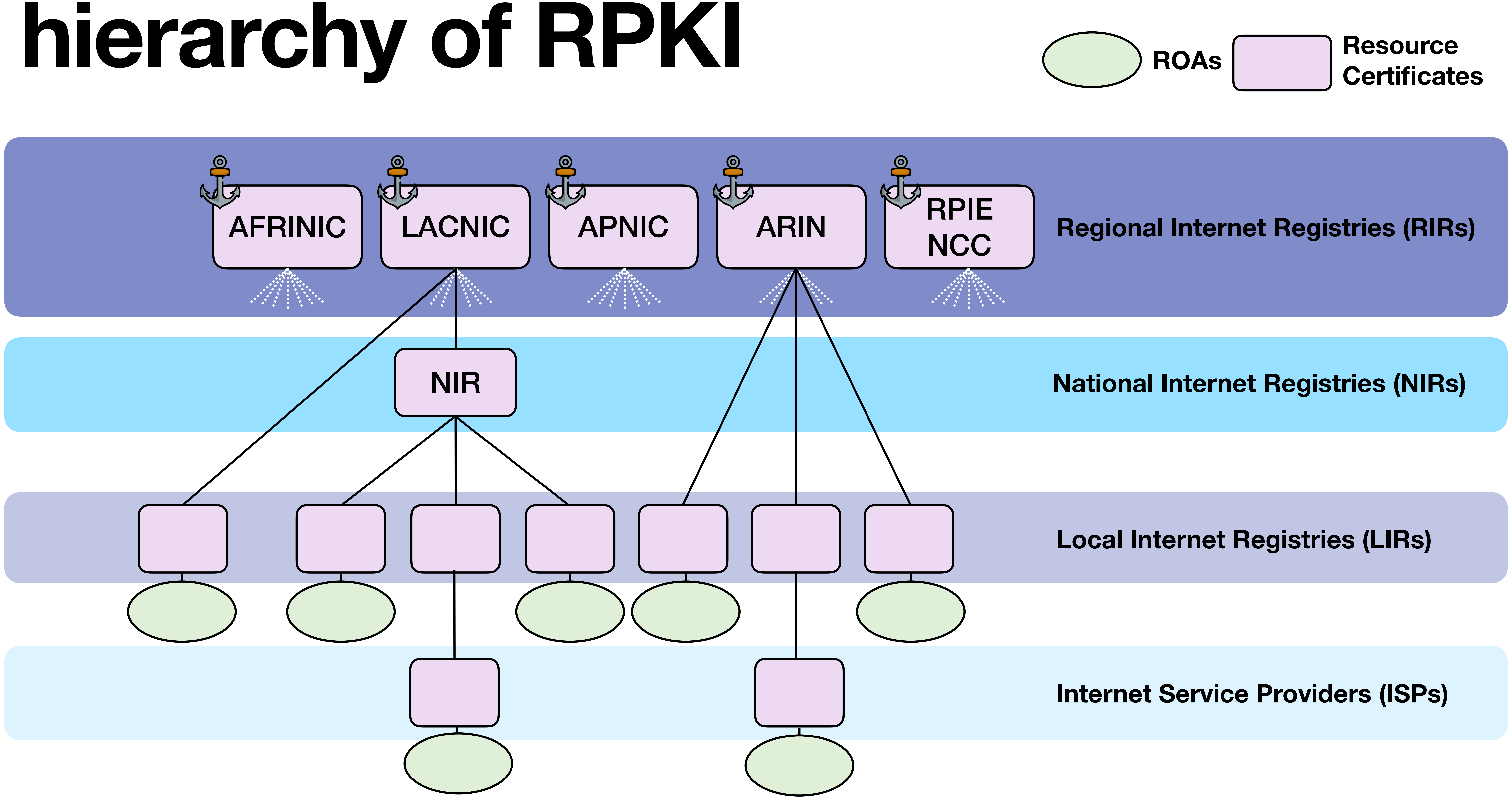
- Two key objects in RPKI
  - Resource Certificate (RC): enables resource holders to assert their legitimate ownership of Internet number resources
  - Route Origin Authorization (ROA): provides a binding of IP prefixes to their legitimate origin ASes

Resource Public Key Infrastructure (RPKI)

- RPKI is an infrastructure for securing Internet number resources (e.g., IP prefixes or AS numbers) and improving security of BGP routing

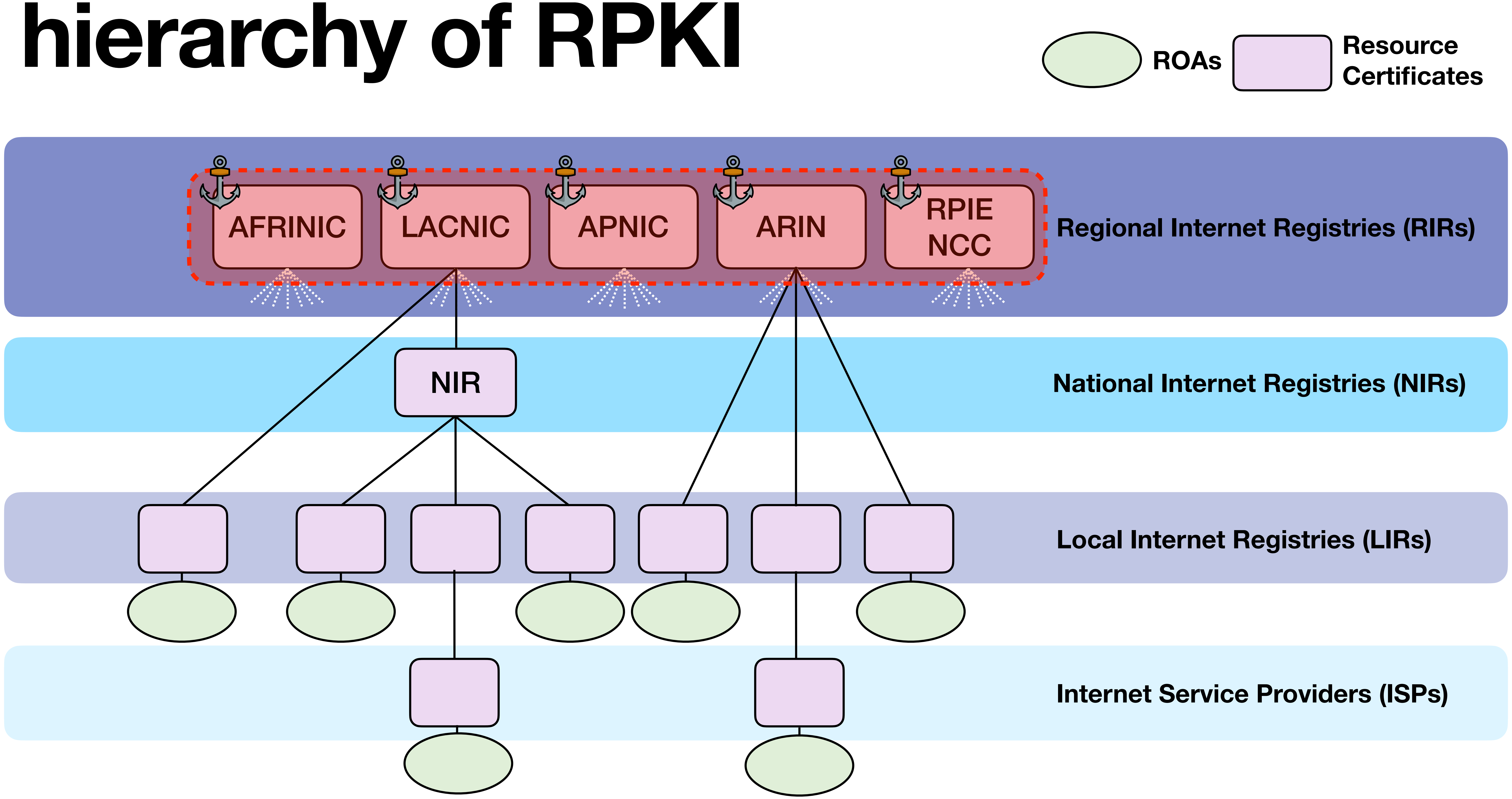
- **Two key objects in RPKI**
  - **Resource Certificate (RC)**: enables resource holders to assert their legitimate ownership of Internet number resources
  - **Route Origin Authorization (ROA)**: provides a binding of IP prefixes to their legitimate origin ASes

The hierarchy of RPKI


ROAs
Resource Certificates
AFRINIC
LACNIC
APNIC
ARIN
RPIE NCC
Regional Internet Registries (RIRs)
NIR
National Internet Registries (NIRs)
Local Internet Registries (LIRs)
Internet Service Providers (ISPs)

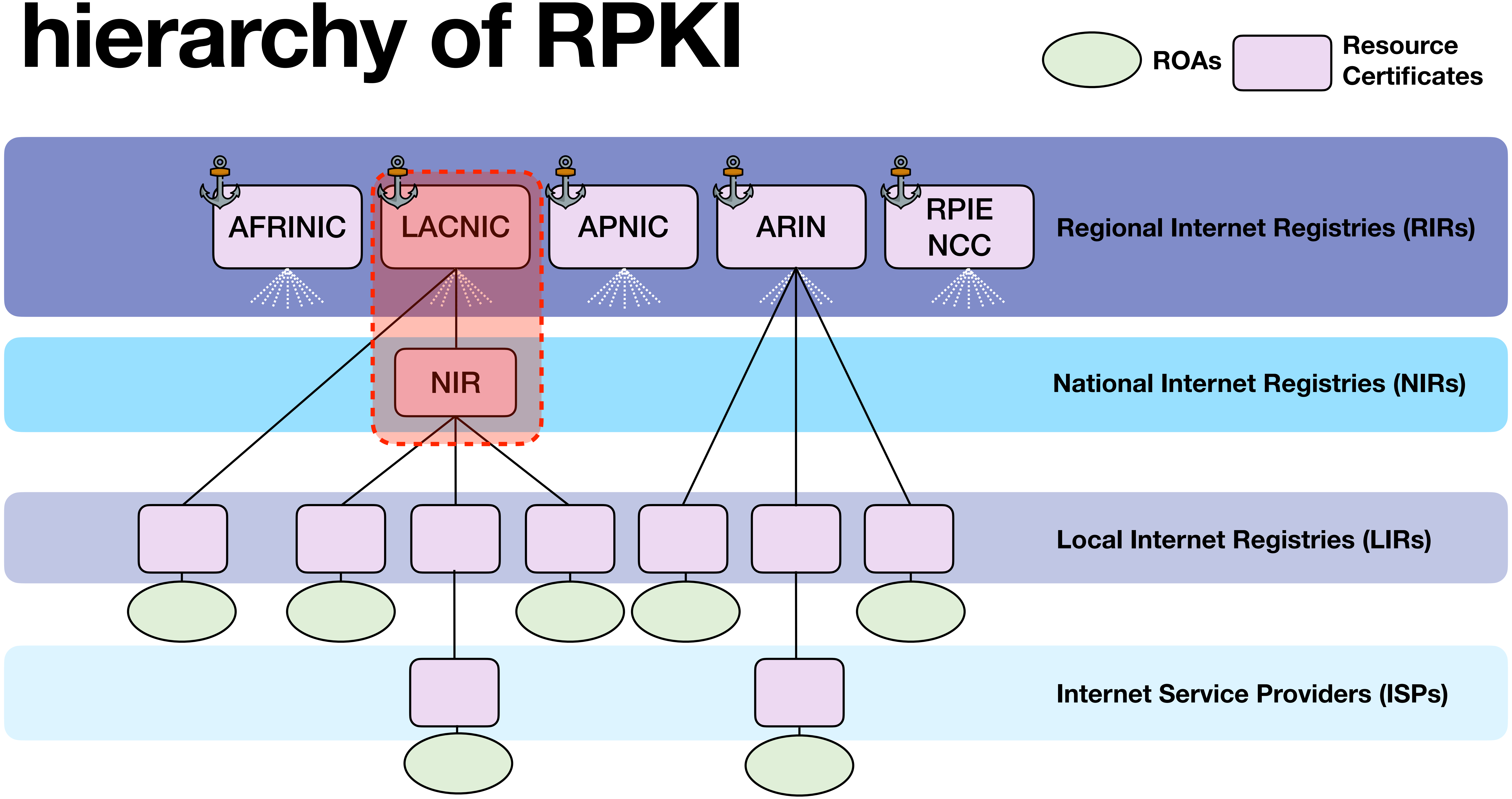
- RPKI is structured hierarchically
  - five RIRs are root CAs and NIRs/LIRs/ISPs are sub-CAs

The hierarchy of RPKI


ROAs
Resource Certificates
AFRINIC
LACNIC
APNIC
ARIN
RPIE NCC
Regional Internet Registries (RIRs)
NIR
National Internet Registries (NIRs)
Local Internet Registries (LIRs)
Internet Service Providers (ISPs)

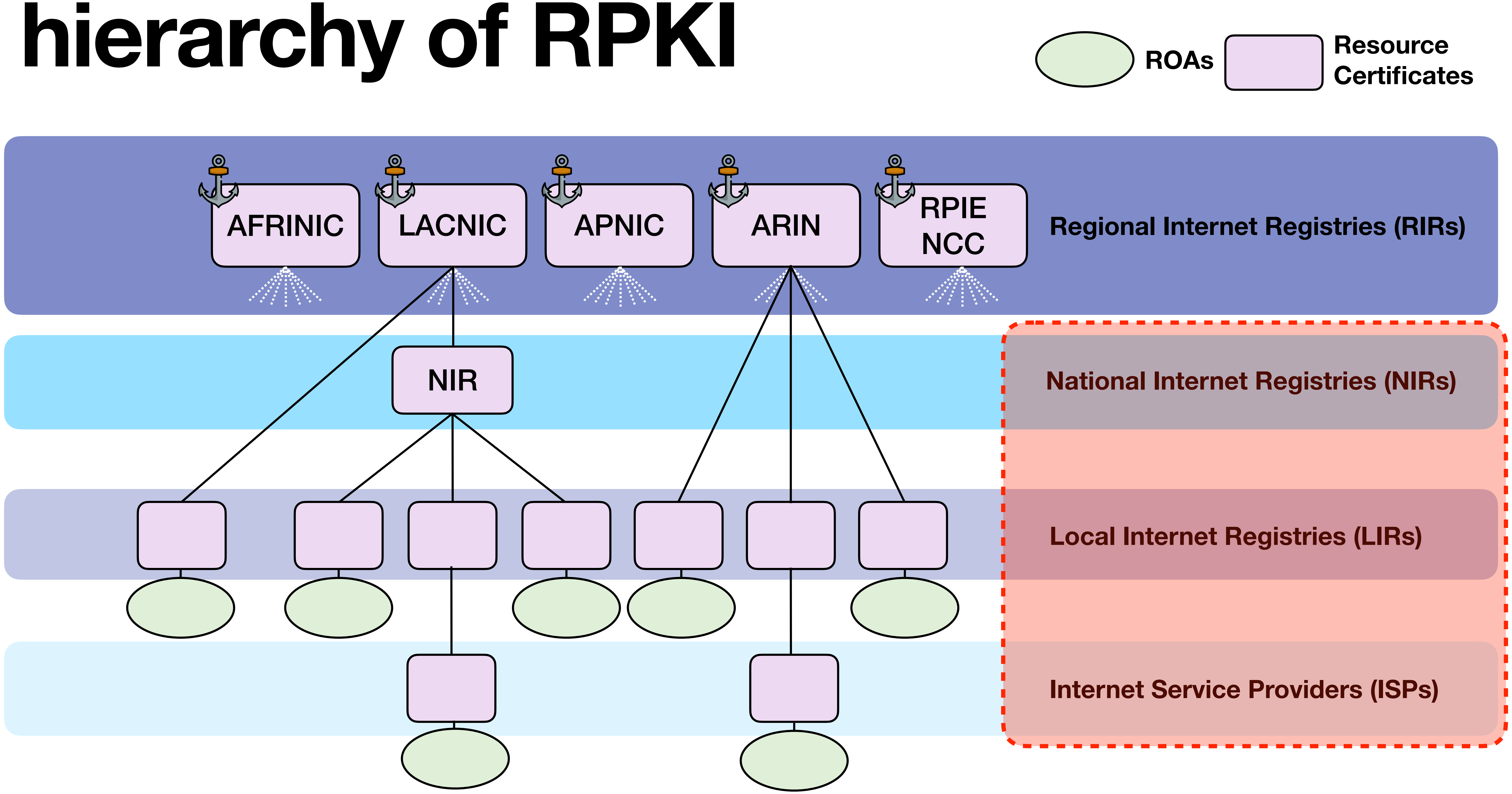
- RPKI is structured hierarchically
  - five RIRs are root CAs and NIRs/LIRs/ISPs are sub-CAs

The hierarchy of RPKI


ROAs
Resource Certificates
AFRINIC
LACNIC
APNIC
ARIN
RPIE NCC
Regional Internet Registries (RIRs)
NIR
National Internet Registries (NIRs)
Local Internet Registries (LIRs)
Internet Service Providers (ISPs)

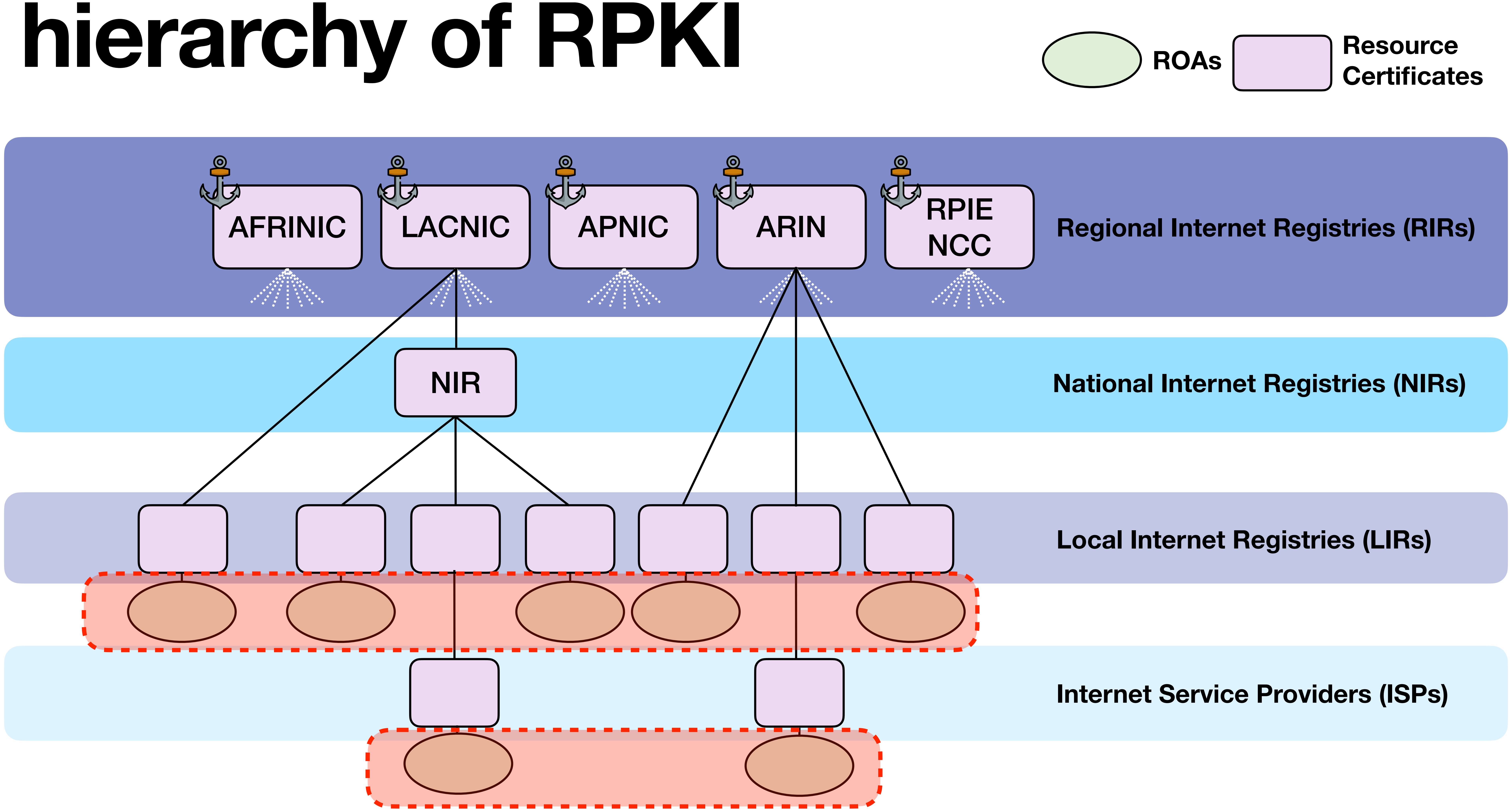
- RPKI is structured hierarchically
  - five RIRs are root CAs and NIRs/LIRs/ISPs are sub-CAs

The hierarchy of RPKI


ROAs
Resource Certificates
AFRINIC
LACNIC
APNIC
ARIN
RPIE NCC
Regional Internet Registries (RIRs)
NIR
National Internet Registries (NIRs)
Local Internet Registries (LIRs)
Internet Service Providers (ISPs)

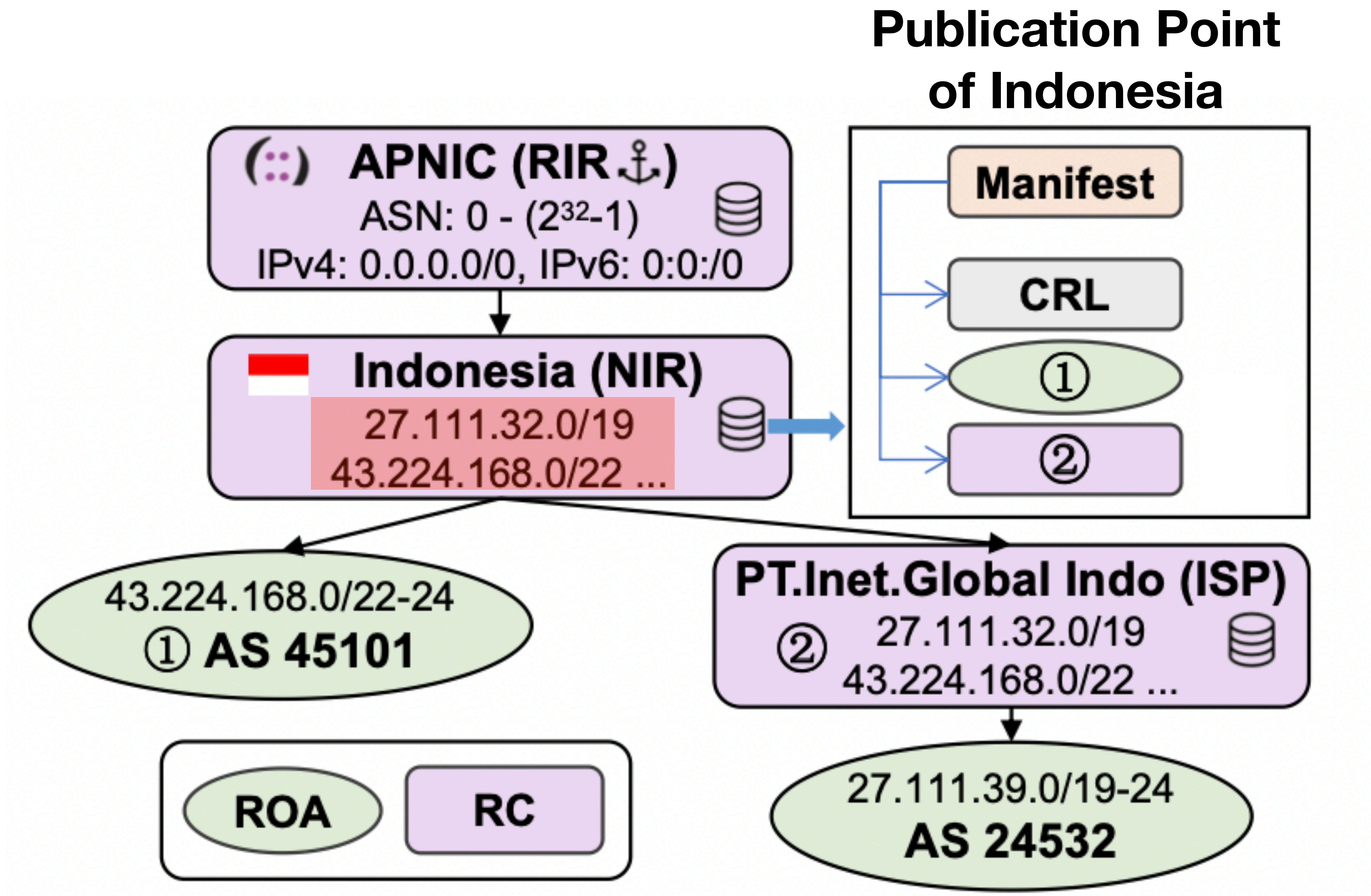
- RPKI is structured hierarchically
  - five RIRs are root CAs and NIRs/LIRs/ISPs are sub-CAs

The hierarchy of RPKI

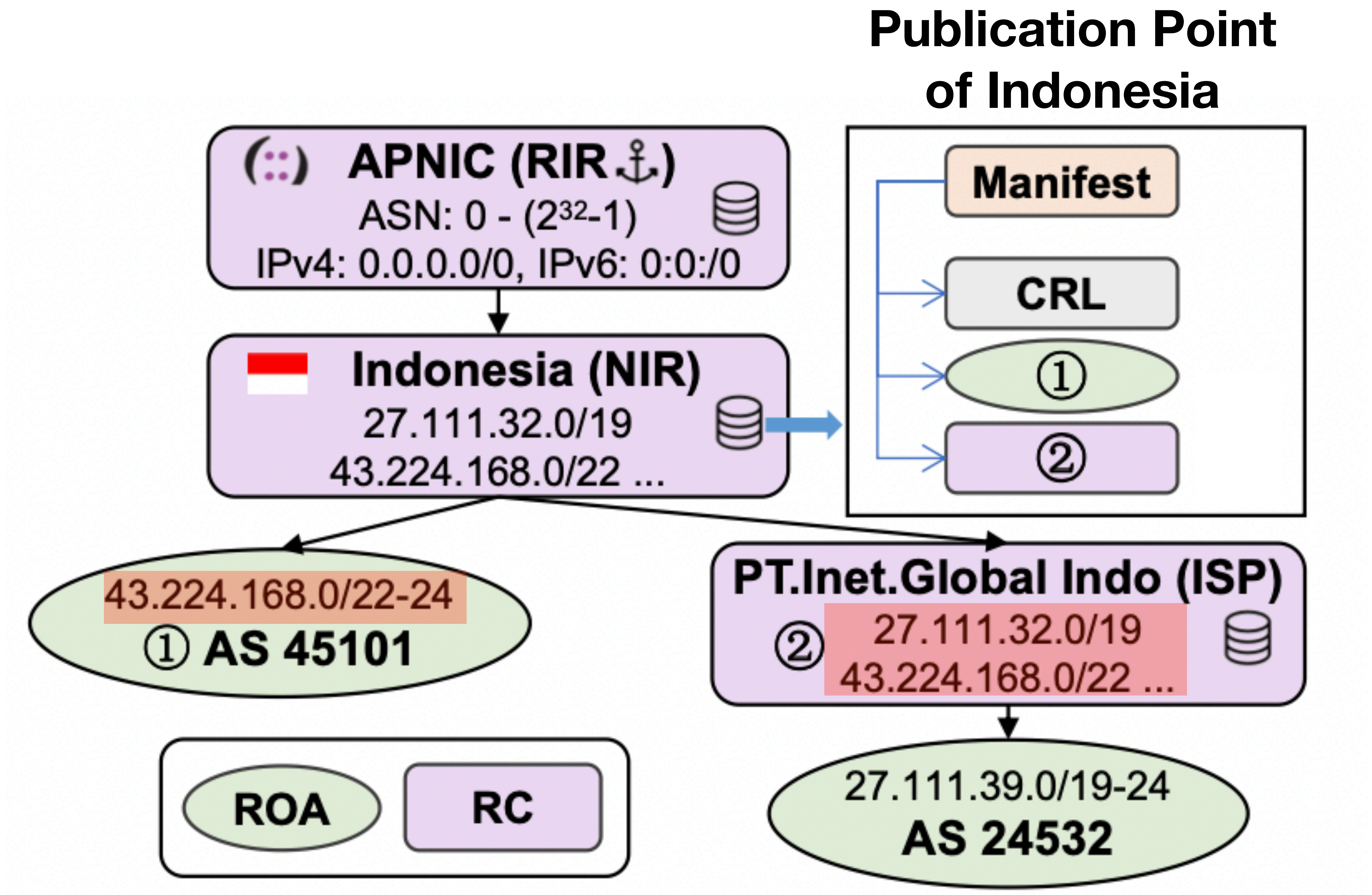

ROAs
Resource Certificates
AFRINIC
LACNIC
APNIC
ARIN
RPIE NCC
Regional Internet Registries (RIRs)
NIR
National Internet Registries (NIRs)
Local Internet Registries (LIRs)
Internet Service Providers (ISPs)

- RPKI is structured hierarchically
  - five RIRs are root CAs and NIRs/LIRs/ISPs are sub-CAs → generate ROAs

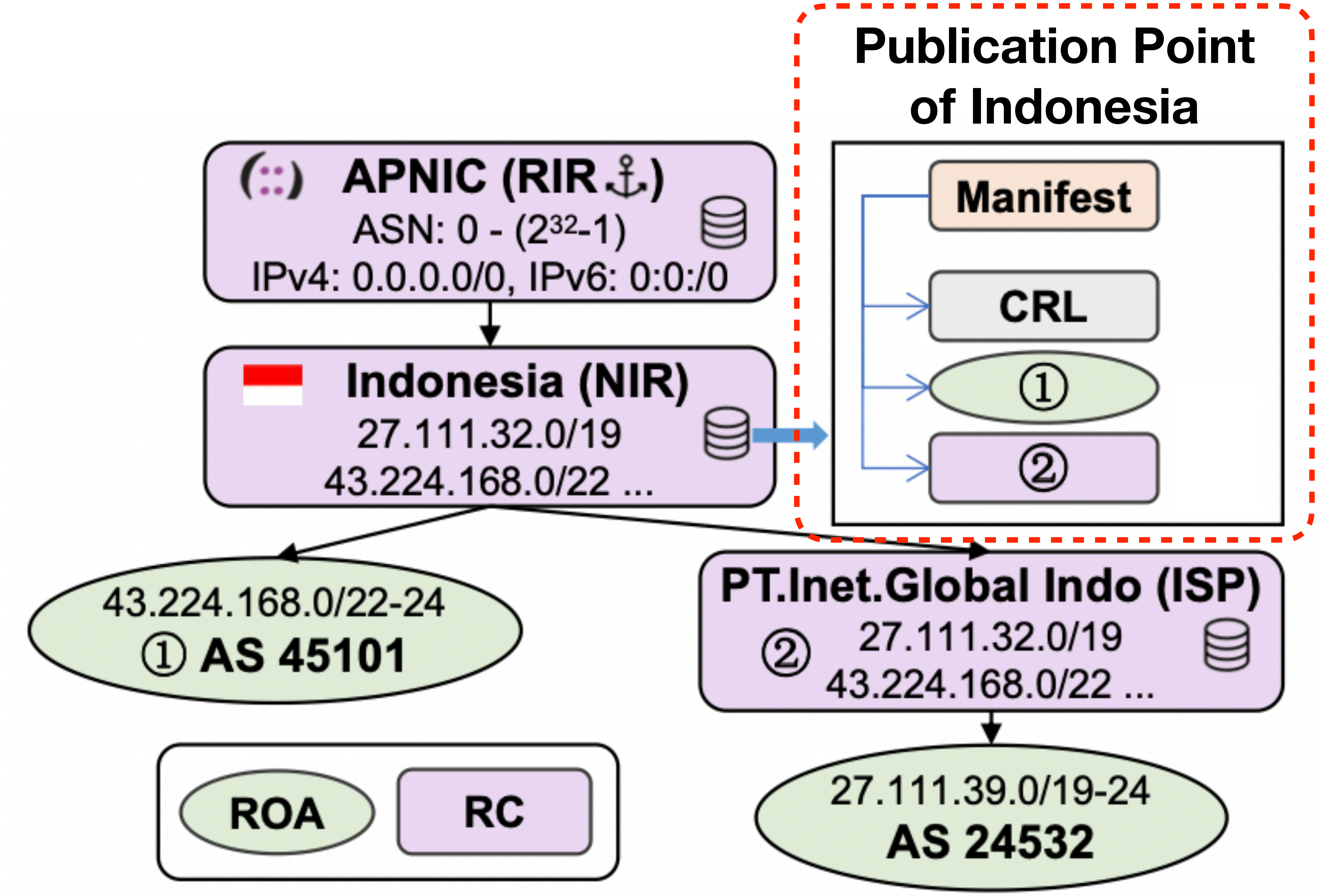
Example of RPKI hierarchical structure



Example of RPKI hierarchical structure

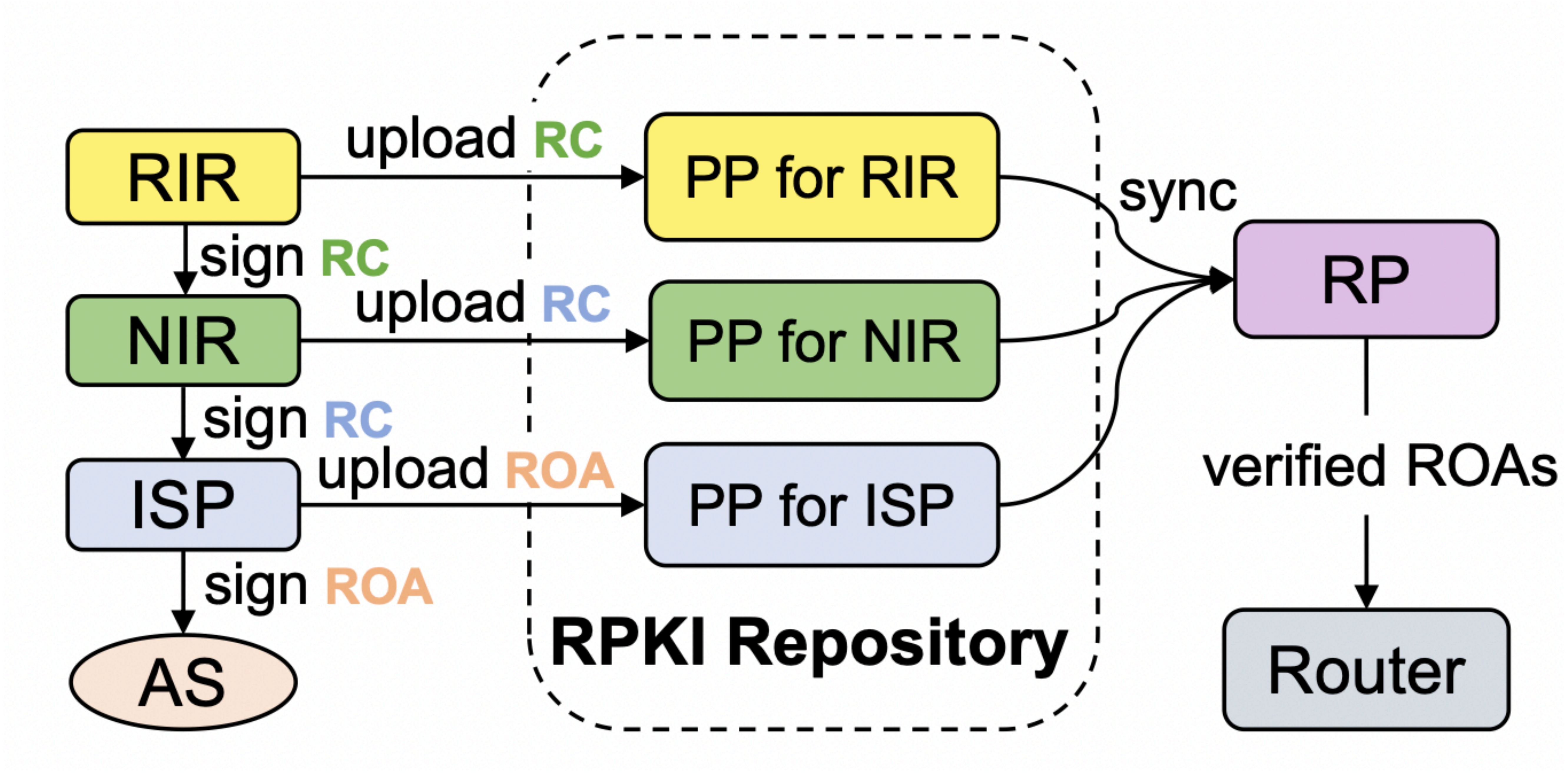


Example of RPKI hierarchical structure



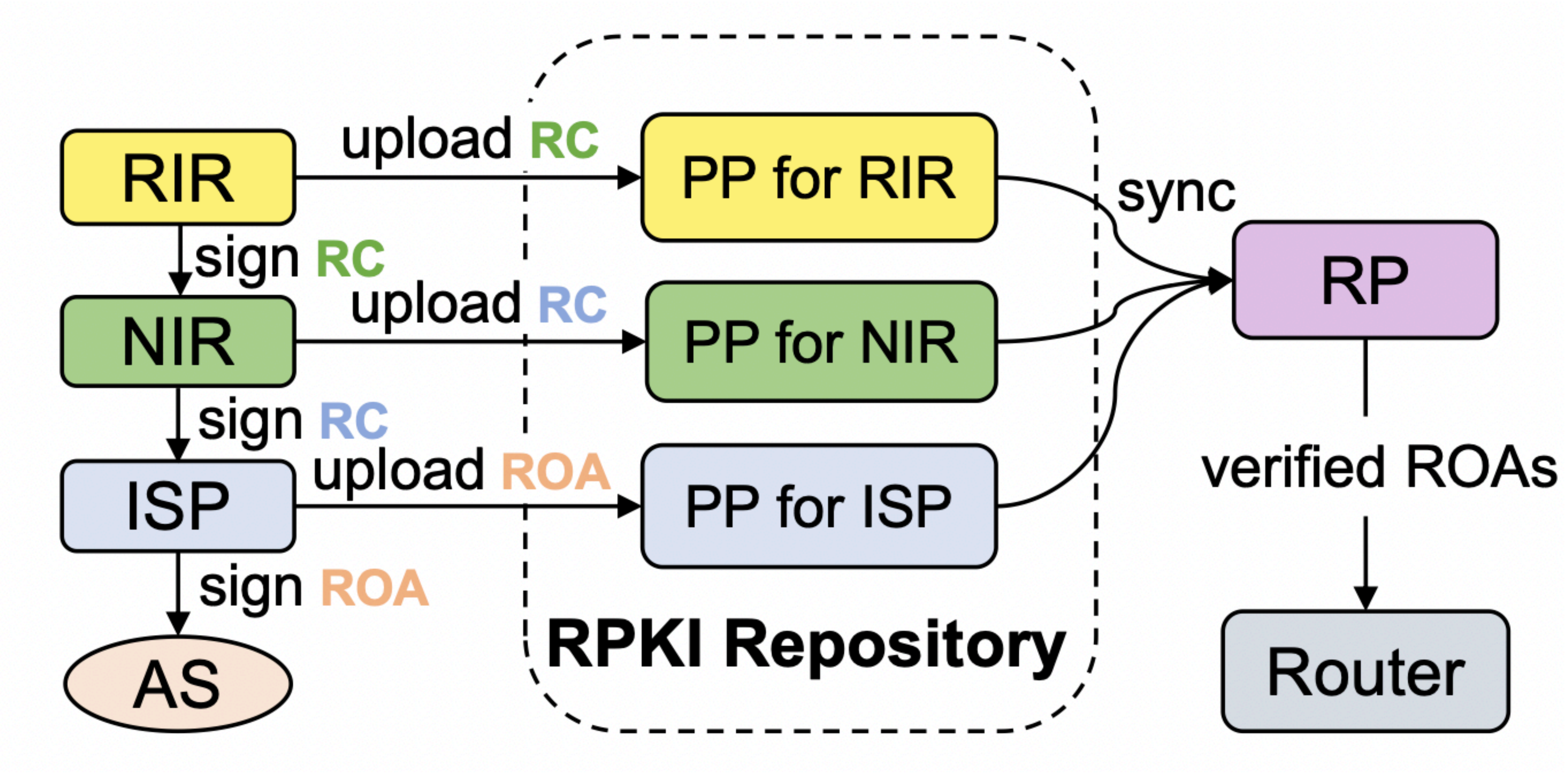
Each CA runs
a Publication Point (PP)
to store RCs and ROAs
issues for Internet
Number Resource
holders (INR holders)

RPKI Repository



- **All PPs collectively form the RPKI Repository**
    - each CA's PP exclusively stores the RPKI objects issued by the respective CA
- Relying Parties (RPs) periodically traverse all PPs, download and validate all RPKI objects

RPKI Repository



- All PPs collectively form the RPKI Repository
  - each CA's PP exclusively stores the RPKI objects issued by the respective CA
- **Relying Parties (RPs) periodically traverse all PPs**, download and validate all RPKI objects

Three Key Problems of RPKI Repository

P1. Unilateral Reliance on RPKI Authority

- CAs can unilaterally undermine any RPKI objects without INR holders' consent

P2. Single Point of Failure

- Any PPs' failure will hinder RPs from obtaining complete RPKI object views
- Introduce inter-dependency between the accessibility of a PP and the reachability of the PP's AS

P3. Poor Scalability

- RP local cache refresh involves traversing all PPs to fetch updated data
- The number of PPs is expected to increase dramatically with the further deployment of RPKI

Three Key Problems of RPKI Repository

P1. Unilateral Reliance on RPKI Authority

- **CAs can unilaterally undermine any RPKI objects without INR holders' consent**

P2. Single Point of Failure

- Any PPs' failure will hinder RPs from obtaining complete RPKI object views
- Introduce inter-dependency between the accessibility of a PP and the reachability of the PP's AS

P3. Poor Scalability

- RP local cache refresh involves traversing all PPs to fetch updated data
- The number of PPs is expected to increase dramatically with the further deployment of RPKI

Three Key Problems of RPKI Repository

P1. Unilateral Reliance on RPKI Authority

- CAs can unilaterally undermine any RPKI objects without INR holders’ consent

P2. Single Point of Failure

- **Any PPs’ failure will hinder RPs from obtaining complete RPKI object views**
- Introduce **inter-dependency** between the **accessibility** of a PP and the **reachability** of the PP’s AS

P3. Poor Scalability

- RP local cache refresh involves traversing all PPs to fetch updated data
- The number of PPs is expected to increase dramatically with the further deployment of RPKI

Three Key Problems of RPKI Repository

P1. Unilateral Reliance on RPKI Authority

- CAs can unilaterally undermine any RPKI objects without INR holders' consent

P2. Single Point of Failure

- Any PPs' failure will hinder RPs from obtaining complete RPKI object views
- Introduce inter-dependency between the accessibility of a PP and the reachability of the PP's AS

P3. Poor Scalability

- **RP local cache refresh involves traversing all PPs to fetch updated data**
- The **number of PPs** is expected to **increase dramatically** with the further deployment of RPKI

Data-driven Security Analysis

P1. “Unilateral Reliance on RPKI Authority"

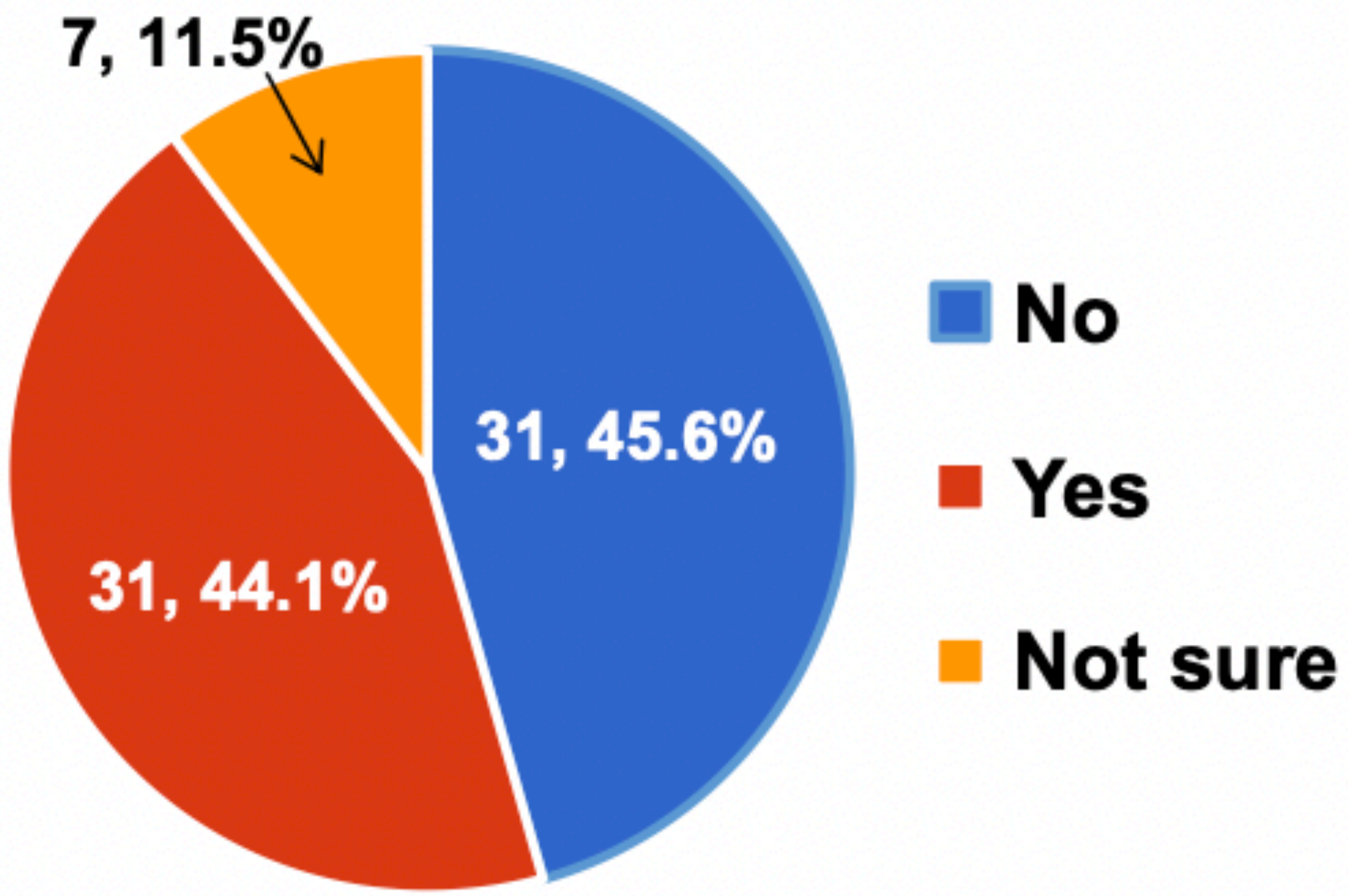


Fig. 21: For P1. Are you worried that RPKI authorities maliciously compromise your certificates, which could affect the legitimacy of your BGP updates? (w/ROA).

- Real-world concerns
  - **44% of AS operators** expressed **concerns** about **malicious authorities**
  - **two operators** consider the threat from authorities to be **the most serious problem**
  - **one operator had lost all their ROAs** due to administrative/human reasons

responses from administrators of 68 ASes that have deployed ROA and 35 ASes that have not deployed ROA

Data-driven Security Analysis

P2. “Vulnerable to Single Point of Failure"

- **only 8** out of 61 PPs are **hosted in CDNs***
  - hosted in Cloudflare AS13335 or Amazon AS16509

*** RPKI Repository Delta Protocol (RRDP)**

- used by Relying Parties to retrieve the RPKI objects from the RPKI repository,
- designed to leverage CDN infrastructure for resilient service

- **58** out of 61 PPs are **hosted in a single AS**
  - The accessibility of these PPs is highly dependent on the reachability of a single AS

- **14** PPs **carry the ROAs of the ASes where PPs are located**
  - The accessibility of these PPs will form a circular dependency on the reachability of the ASes

Data-driven Security Analysis

P3. “Poor Scalability"

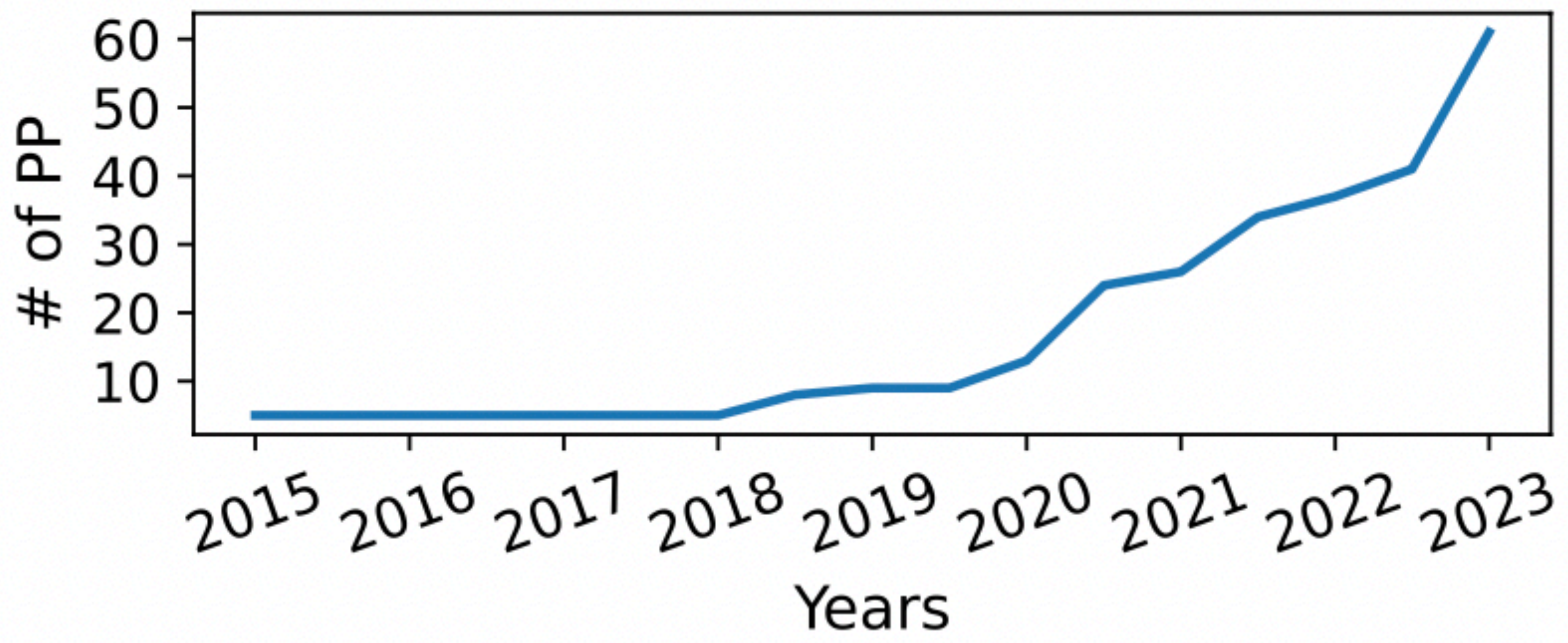


Fig. 4: The number of independent PPs over nine years.

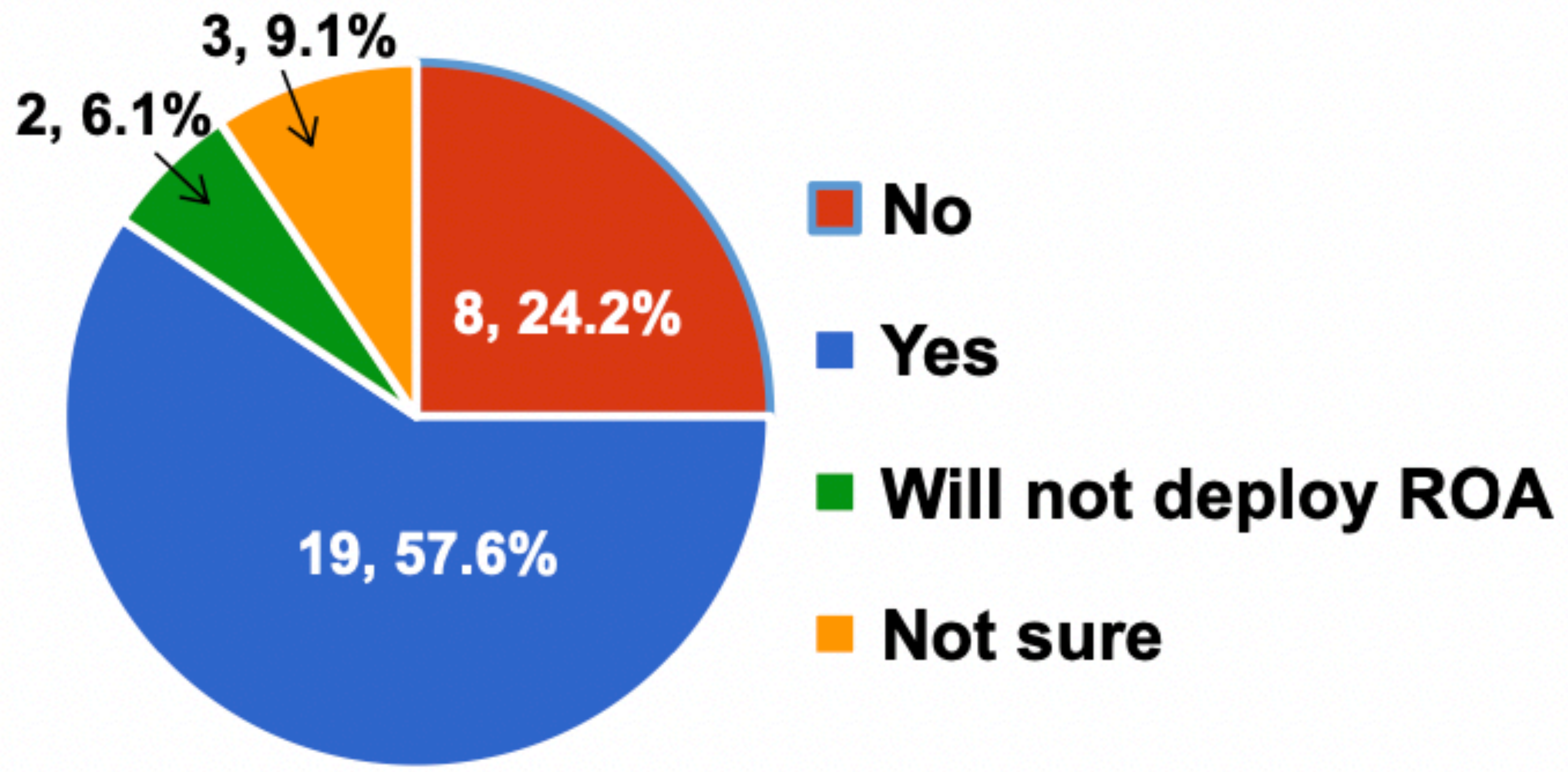


Fig. 22: For P3. If you deploy ROA in the future, would you consider adopting delegated RPKI and running your own PP? (wo/ROA).

- Analysis
  - **the number of PPs** has **grown** more than **12 times**
  - **many AS operators** are **considering running PPs**
  - when RPKI is fully deployed, **the number of PPs will inevitably increase**
- Potential Problems
  - threaten the scalability of RPKI
  - increase the cost of RP refreshing

Design Goal of dRR *(**d**ecentralized **R**PKI **R**epository)*

P1

- **Defend against RPKI authorities' malicious behavior**
  - Allow RPs verify certificate status
  - Allow resource holders verify the integrity of RPKI views
  - RPKI historical data can be audited

P2

- **Defend against single point of failure**
  - Truly distributed data storage
  - PP accessibility is independent of AS accessibility

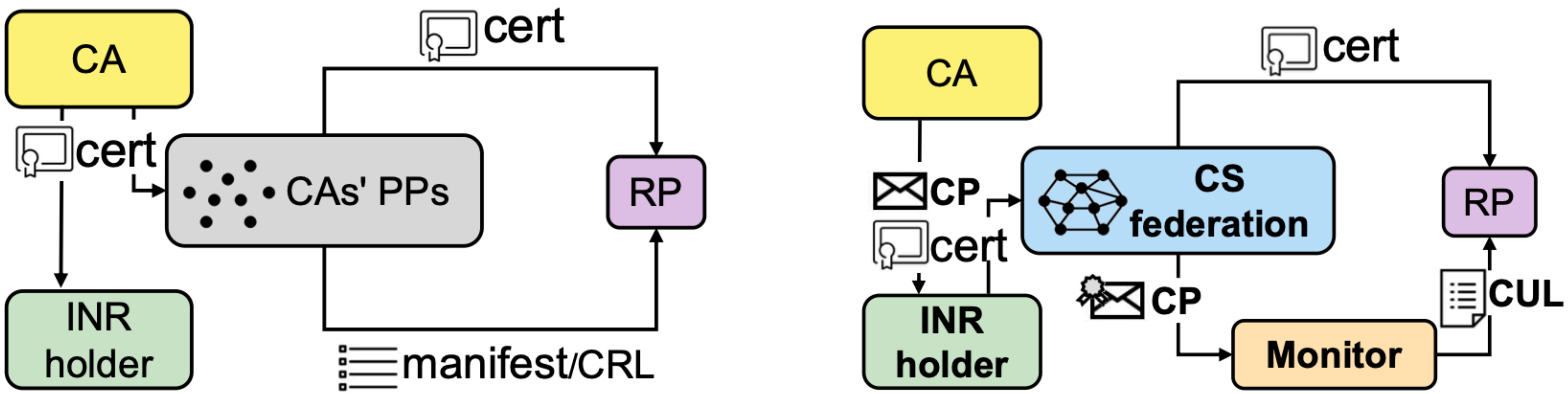
P3

- **Prevent unlimited growth in the number of PPs**
  - Improve the reliability of RPKI Repository system

Be compatible with RPKI architecture and support incremental deployment

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI

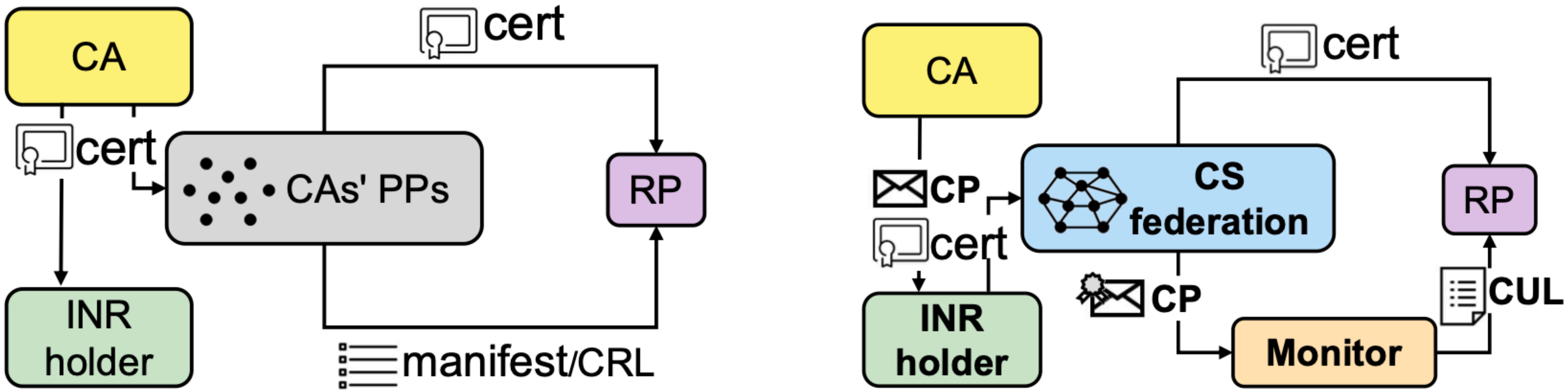


(a) the current RPKI Repository.

(b) dRR

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI

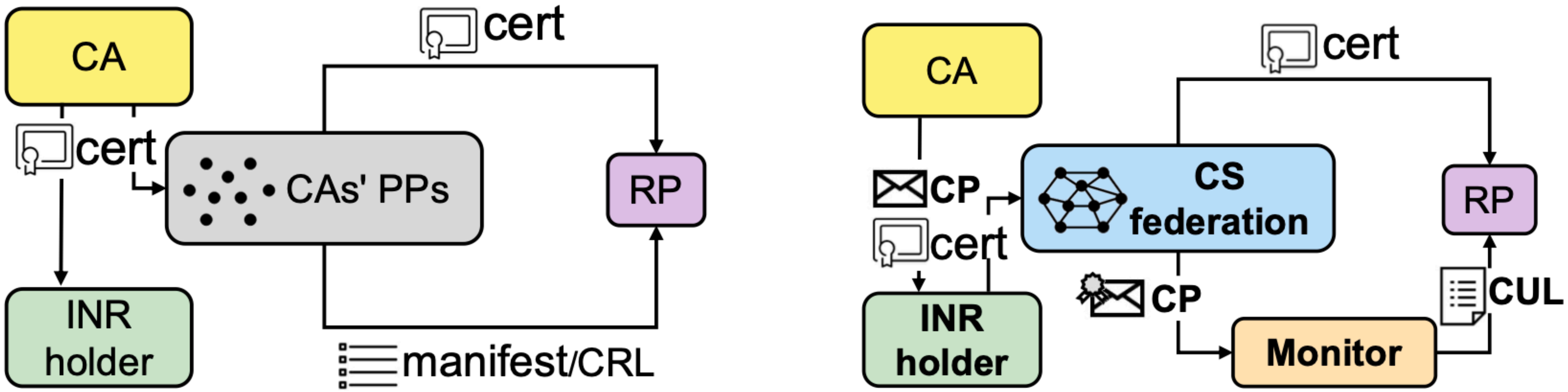


(a) the current RPKI Repository. (b) dRR

Key new entities for dRR: Certificate Server (CS) Federation and Monitor

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI



(a) the current RPKI Repository.

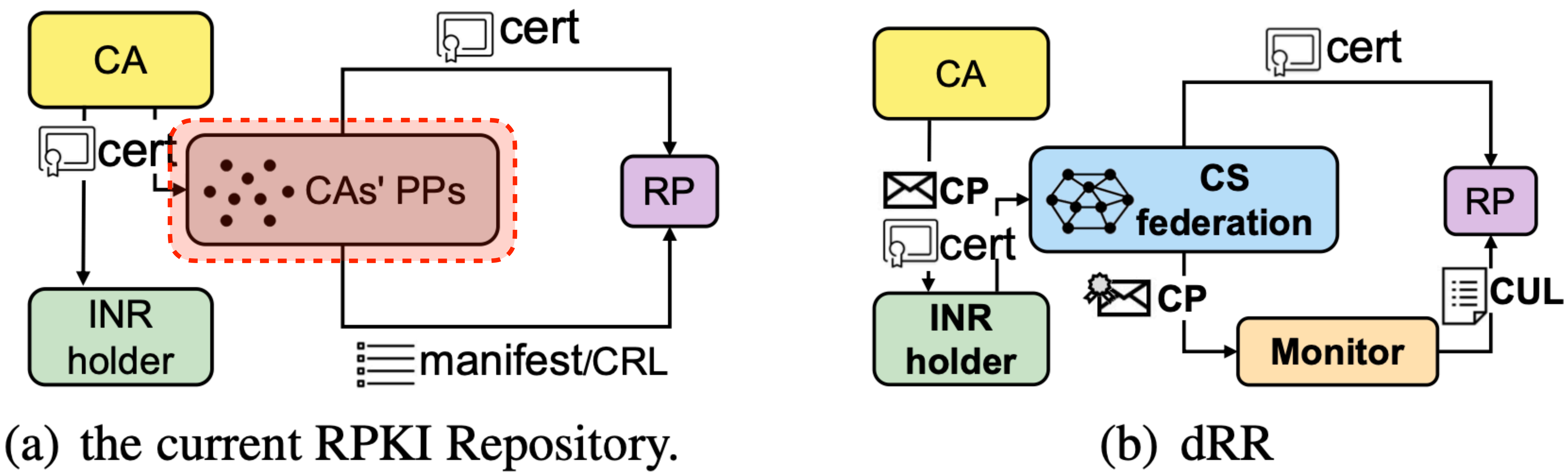
(b) dRR

Key new entities for dRR: Certificate Server (CS) Federation and Monitor

Key new objects for dRR: Certificate Policy (CP) and Certificate Update List (CUL)

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI



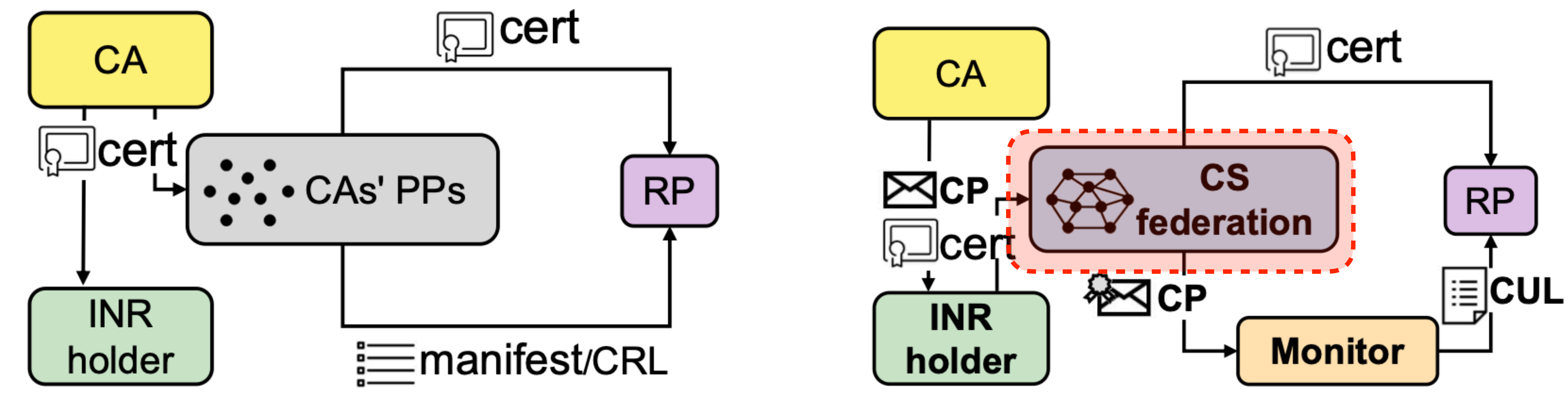
(a) the current RPKI Repository. (b) dRR

Key new entities for dRR: Certificate Server (CS) Federation and Monitor

Key new objects for dRR: Certificate Policy (CP) and Certificate Update List (CUL)

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI



(a) the current RPKI Repository.

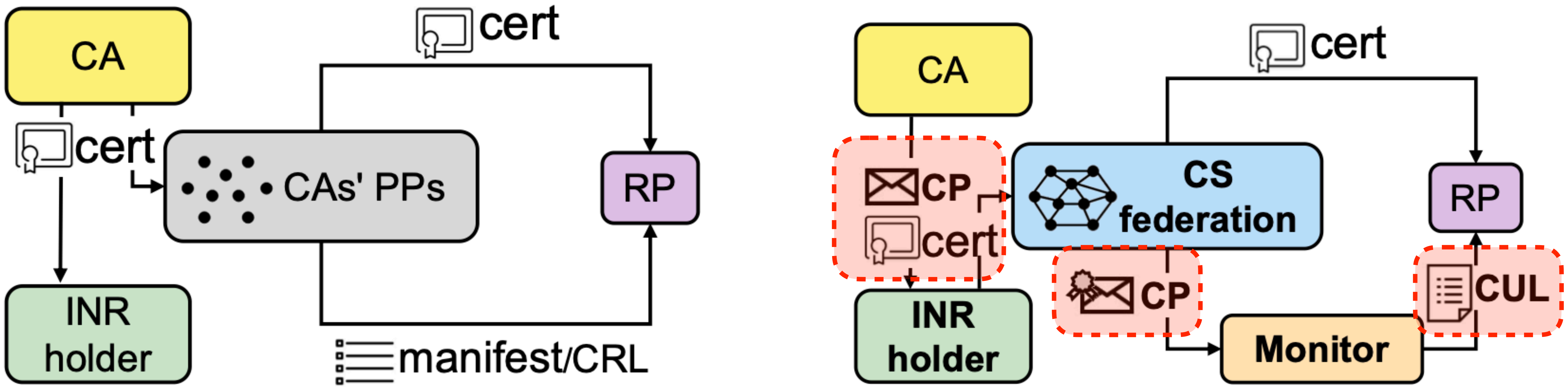
(b) dRR

Key new entities for dRR: Certificate Server (CS) Federation and Monitor

Key new objects for dRR: Certificate Policy (CP) and Certificate Update List (CUL)

Key Idea of dRR

- **Separating RPKI object distribution from signing**
  - decouple PP and RPKI Authority and design a third-party repository for RPKI



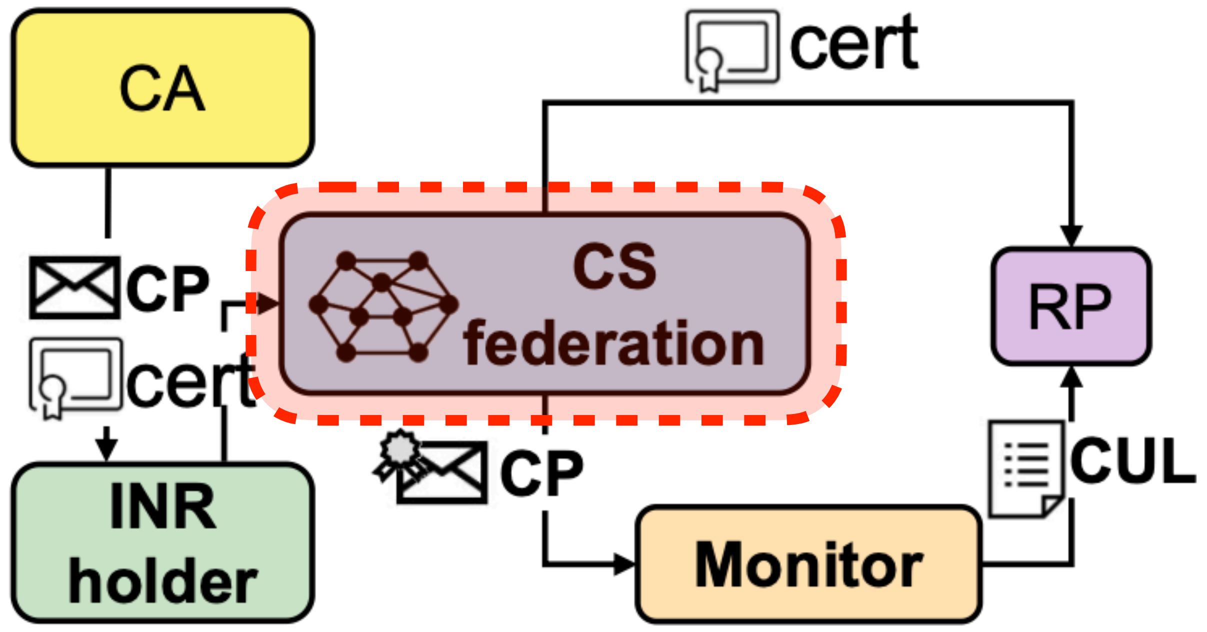
(a) the current RPKI Repository.

(b) dRR

Key new entities for dRR: Certificate Server (CS) Federation and Monitor

Key new objects for dRR: Certificate Policy (CP) and Certificate Update List (CUL)

Certificate Server (CS) Federation

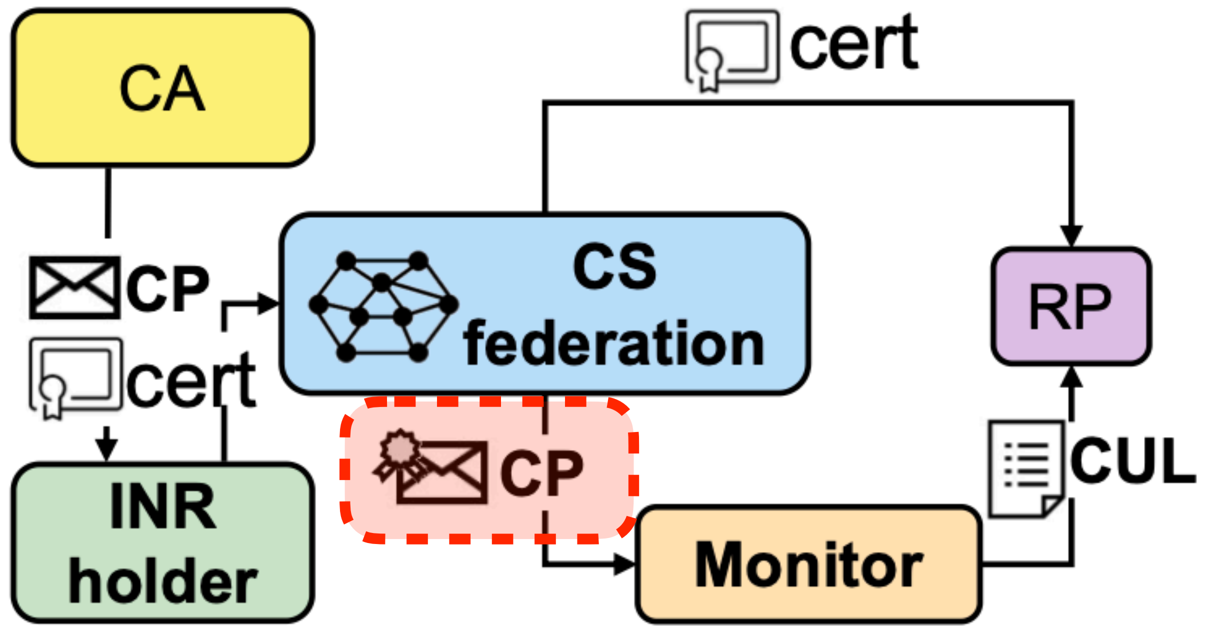


(b) dRR

- Hosting resource certificates and ROAs for resource holders

- Two main improvements of the CS, compared to the traditional publication point (PP)
  - **independent of CAs**, all certificate servers are equal and together form the CS federation
  - **resource holders can freely choose any CSs** they trust to provide certificate hosting services for them
    - not only host the certificates, but also **publicize certificate policies**

Certificate Policy (CP)



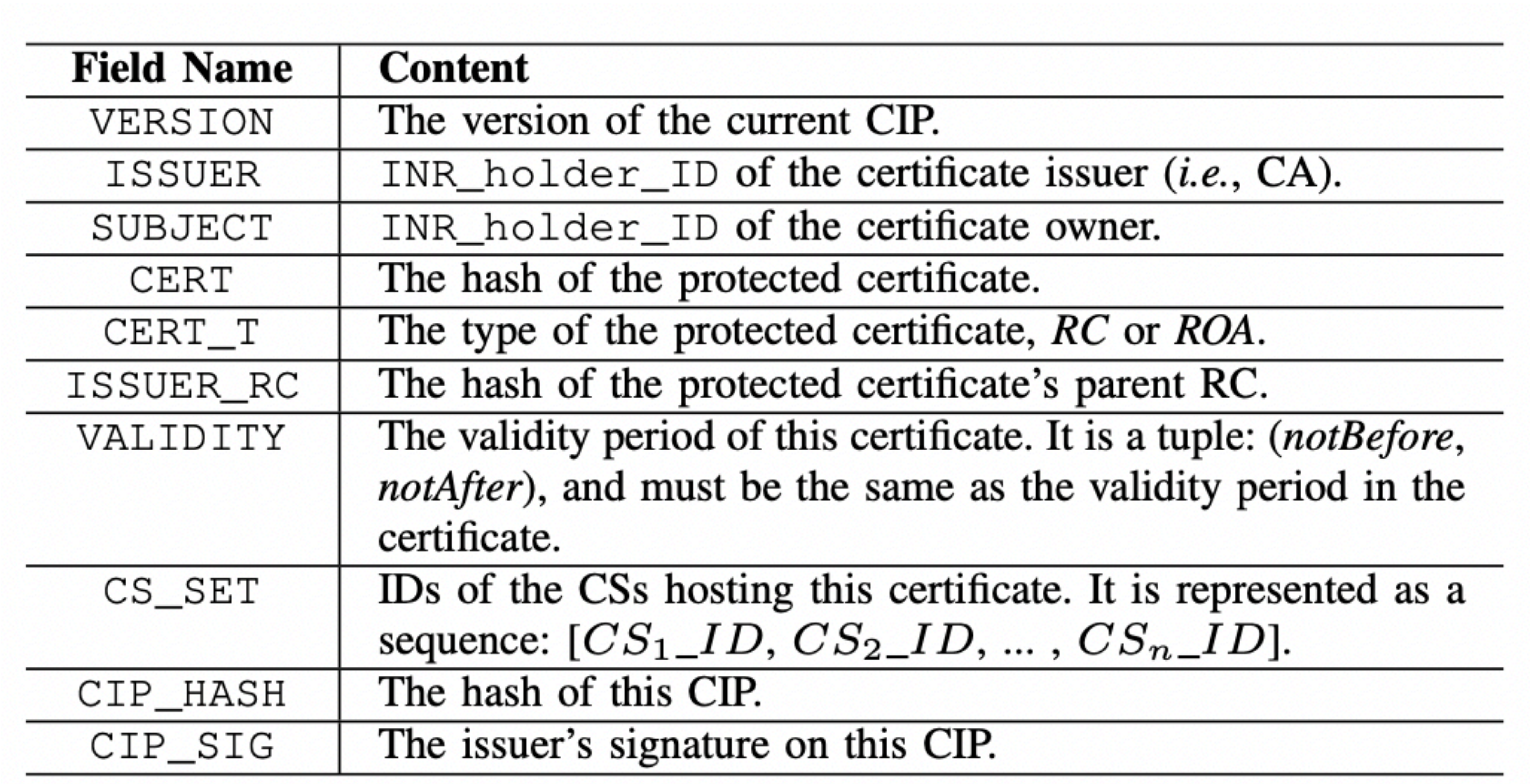
(b) dRR

- Any certificate issuance and revocation will be publicized in the CS federation in the form of Certificate Policies

- Two types of CPs
  - **certificate issuance policy (CIP)**
  - **certificate revocation policy (CRP)**

Certificate Issuance Policy (CIP)

- CAs provide CIPs to resource holders to **prove the authenticity of the issuance** of certificates

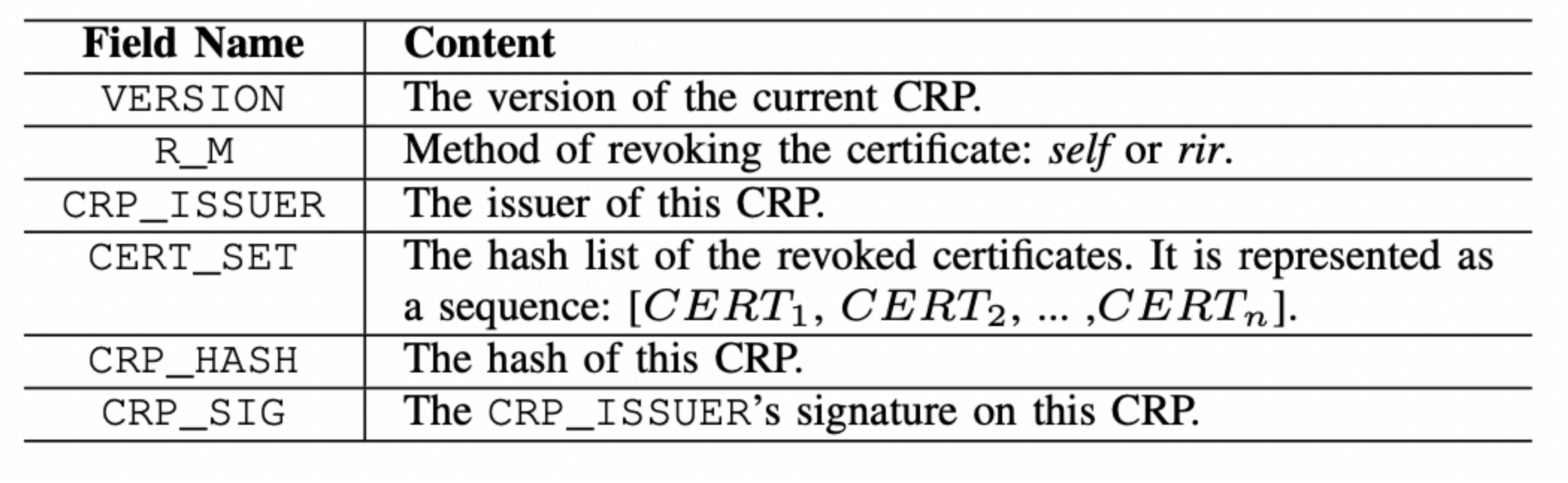


Field Name	Content
`VERSION`	The version of the current CIP.
`ISSUER`	`INR_holder_ID` of the certificate issuer (*i.e.*, CA).
`SUBJECT`	`INR_holder_ID` of the certificate owner.
`CERT`	The hash of the protected certificate.
`CERT_T`	The type of the protected certificate, *RC* or *ROA*.
`ISSUER_RC`	The hash of the protected certificate's parent RC.
`VALIDITY`	The validity period of this certificate. It is a tuple: (*notBefore*, *notAfter*), and must be the same as the validity period in the certificate.
`CS_SET`	IDs of the CSs hosting this certificate. It is represented as a sequence: [CS_1_ID, CS_2_ID, ... , CS_n_ID].
`CIP_HASH`	The hash of this CIP.
`CIP_SIG`	The issuer's signature on this CIP.

Certificate Revocation Policy (CRP)

- Resource holders provide CRPs to confirm that **the revocation has obtained the consent** of all affected parties

- **Five RIRs** can jointly sign CRPs for **mandatory certificate revocation**



Field Name	Content
`VERSION`	The version of the current CRP.
`R_M`	Method of revoking the certificate: *self* or *rir*.
`CRP_ISSUER`	The issuer of this CRP.
`CERT_SET`	The hash list of the revoked certificates. It is represented as a sequence: [$CERT_1$, $CERT_2$, ... ,$CERT_n$].
`CRP_HASH`	The hash of this CRP.
`CRP_SIG`	The `CRP_ISSUER`'s signature on this CRP.

The Global Ledger Maintained by the CS Federation

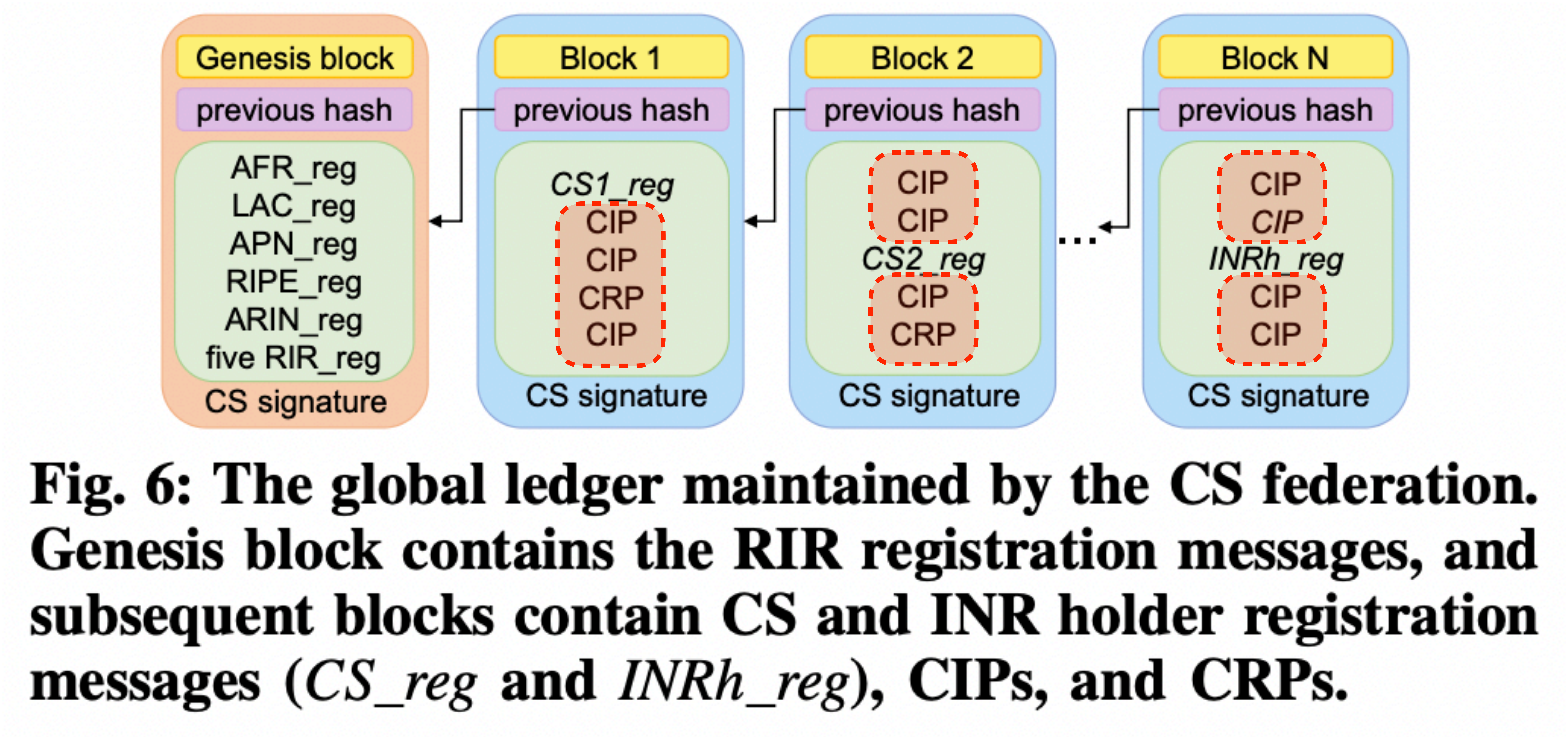


Fig. 6: The global ledger maintained by the CS federation. Genesis block contains the RIR registration messages, and subsequent blocks contain CS and INR holder registration messages (*CS_reg* and *INRh_reg*), CIPs, and CRPs.

The Global Ledger Maintained by the CS Federation

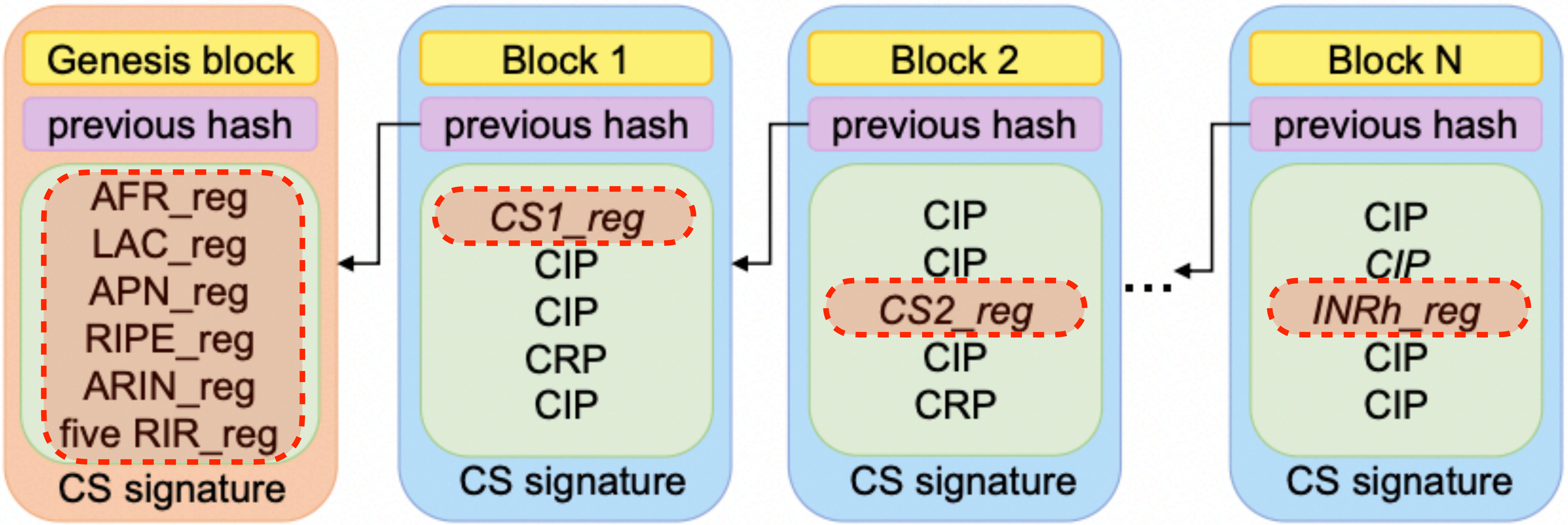
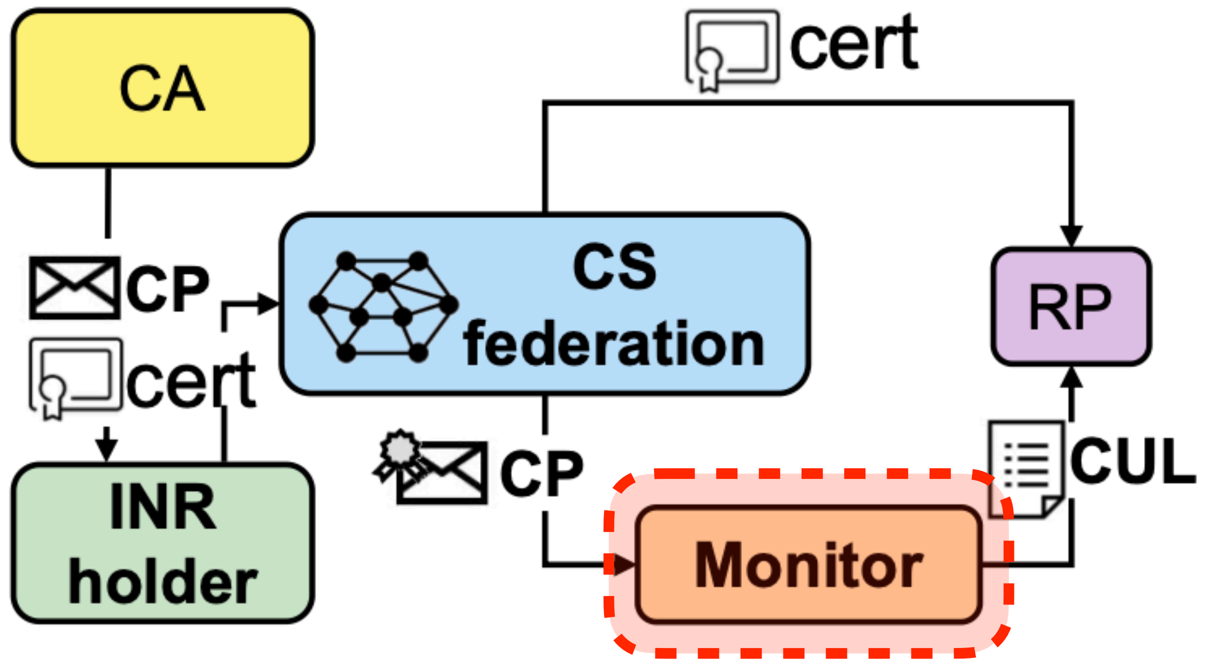


Fig. 6: The global ledger maintained by the CS federation. Genesis block contains the RIR registration messages, and subsequent blocks contain CS and INR holder registration messages (*CS_reg* and *INRh_reg*), CIPs, and CRPs.

Monitor

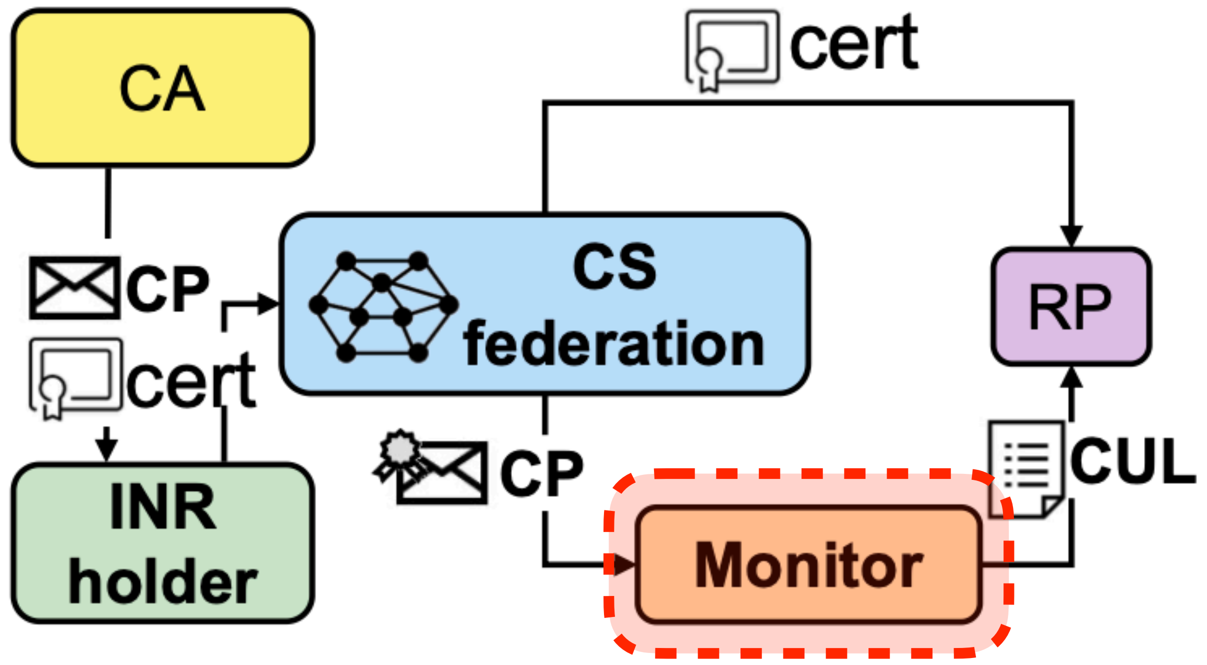


(b) dRR

- The Monitor in dRR provides proofs of the status of a specific certificate and the trustworthiness of the monitor
    - proof of presence, proof of absence, proof of consistency

- M-Tree
    - a Merkle Hash Tree (MHT) with leaf nodes containing CPs
    - generate a *commitment* (root hash of the tree) after inserting a block’s CIPs and CRPs
    - the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes
    - the revocation of the certificates in the newly added CRP is recorded by modifying the CIP entries of the respective certificates

Monitor

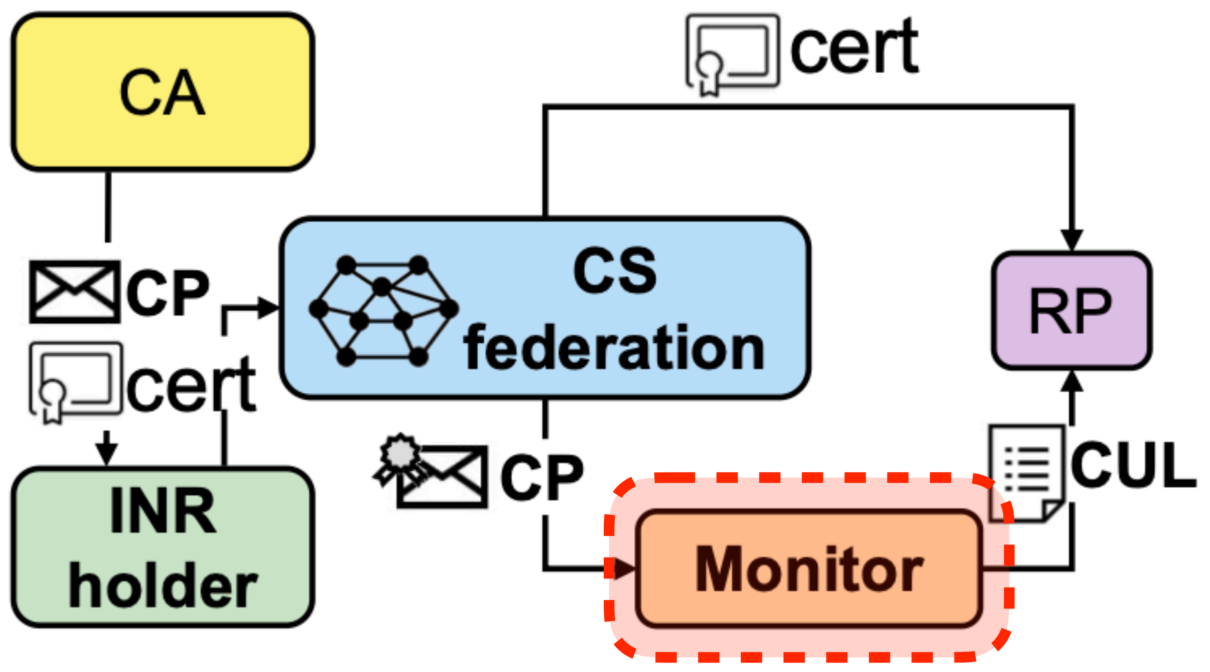


(b) dRR

- The Monitor in dRR provides proofs of the status of a specific certificate and the trustworthiness of the monitor
  - proof of presence, proof of absence, proof of consistency

- **M-Tree**
  - **a Merkle Hash Tree (MHT) with leaf nodes containing CPs**
  - generate a *commitment* (root hash of the tree) after inserting a block's CIPs and CRPs
  - the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes
  - the revocation of the certificates in the newly added CRP is recorded by modifying the CIP entries of the respective certificates

Monitor

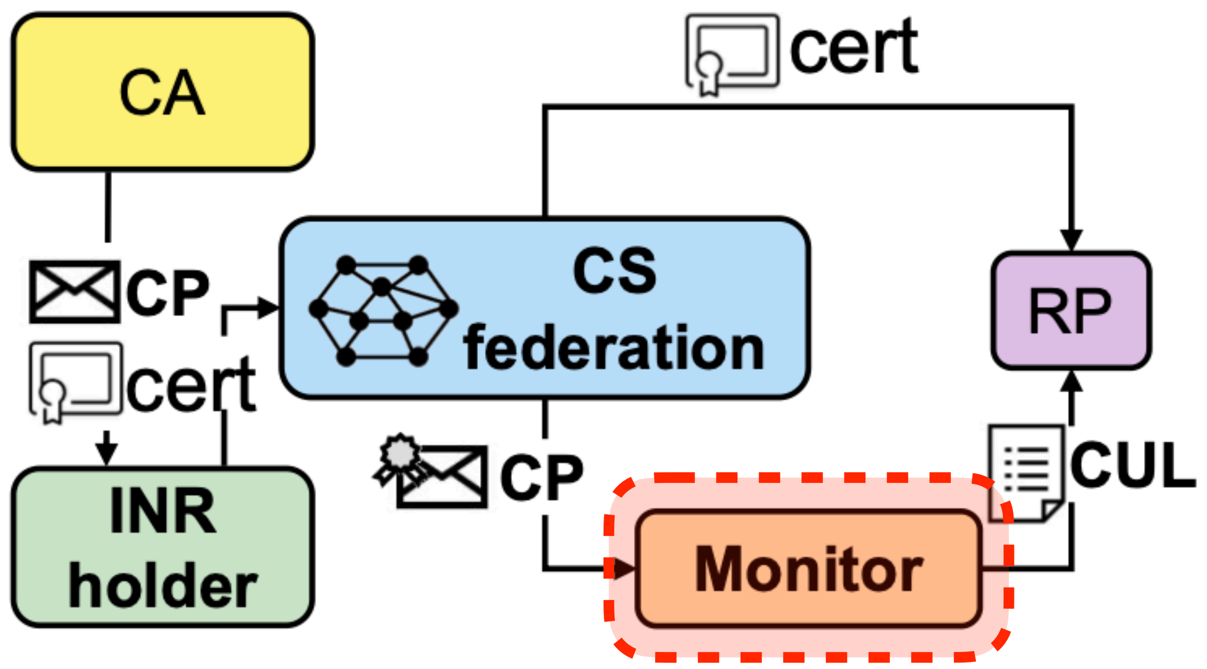


(b) dRR

- The Monitor in dRR provides proofs of the status of a specific certificate and the trustworthiness of the monitor
  - proof of presence, proof of absence, proof of consistency

- **M-Tree**
  - a Merkle Hash Tree (MHT) with leaf nodes containing CPs
  - **generate a *commitment* (root hash of the tree) after inserting a block's CIPs and CRPs**
  - the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes
  - the revocation of the certificates in the newly added CRP is recorded by modifying the CIP entries of the respective certificates

Monitor

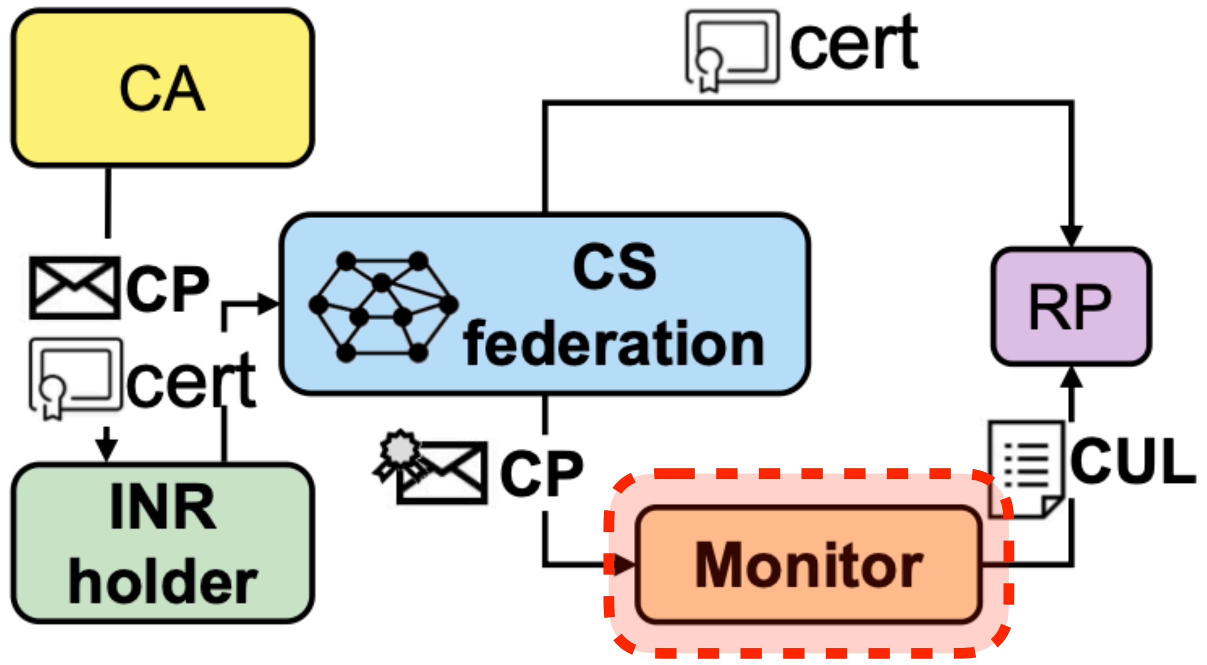


(b) dRR

- The Monitor in dRR provides proofs of the status of a specific certificate and the trustworthiness of the monitor
  - proof of presence, proof of absence, proof of consistency

- **M-Tree**
  - a Merkle Hash Tree (MHT) with leaf nodes containing CPs
  - generate a *commitment* (root hash of the tree) after inserting a block's CIPs and CRPs
  - **the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes**
  - the revocation of the certificates in the newly added CRP is recorded by modifying the CIP entries of the respective certificates

Monitor



(b) dRR

- The Monitor in dRR provides proofs of the status of a specific certificate and the trustworthiness of the monitor
    - proof of presence, proof of absence, proof of consistency

- **M-Tree**
    - a Merkle Hash Tree (MHT) with leaf nodes containing CPs
    - generate a *commitment* (root hash of the tree) after inserting a block's CIPs and CRPs
    - the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes
    - **the revocation of the certificates in the newly added CRP is recorded by modifying the CIP entries of the respective certificates**

Proof of Presence

- **A pruned tree** that contains the leaf entry of the requested certificate and the intermediate nodes needed to reconstruct the commitment

- Verification process
  - an INR holder asks whether a certificate exists
  - the Monitor will return a proof of presence
  - the INR holder reconstructs a *commitment* C'
  - then the INR holder accesses *the commitment update files* provided by other Monitors to check the authenticity of C'

Proof of Presence

- A pruned tree that contains the leaf entry of the requested certificate and the intermediate nodes needed to reconstruct the commitment

- **Verification process**
  - **an INR holder asks whether a certificate exists**
  - the Monitor will return a proof of presence
  - the INR holder reconstructs a *commitment C'*
  - then the INR holder accesses *the commitment update files* provided by other Monitors to check the authenticity of C'

Proof of Presence

- A pruned tree that contains the leaf entry of the requested certificate and the intermediate nodes needed to reconstruct the commitment

- **Verification process**
  - an INR holder asks whether a certificate exists
  - **the Monitor will return a proof of presence**
  - the INR holder reconstructs a *commitment C′*
  - then the INR holder accesses *the commitment update files* provided by other Monitors to check the authenticity of *C′*

Proof of Presence

- A pruned tree that contains the leaf entry of the requested certificate and the intermediate nodes needed to reconstruct the commitment

- **Verification process**
  - an INR holder asks whether a certificate exists
  - the Monitor will return a proof of presence
  - **the INR holder reconstructs a *commitment C′***
  - then the INR holder accesses *the commitment update files* provided by other Monitors to check the authenticity of *C′*

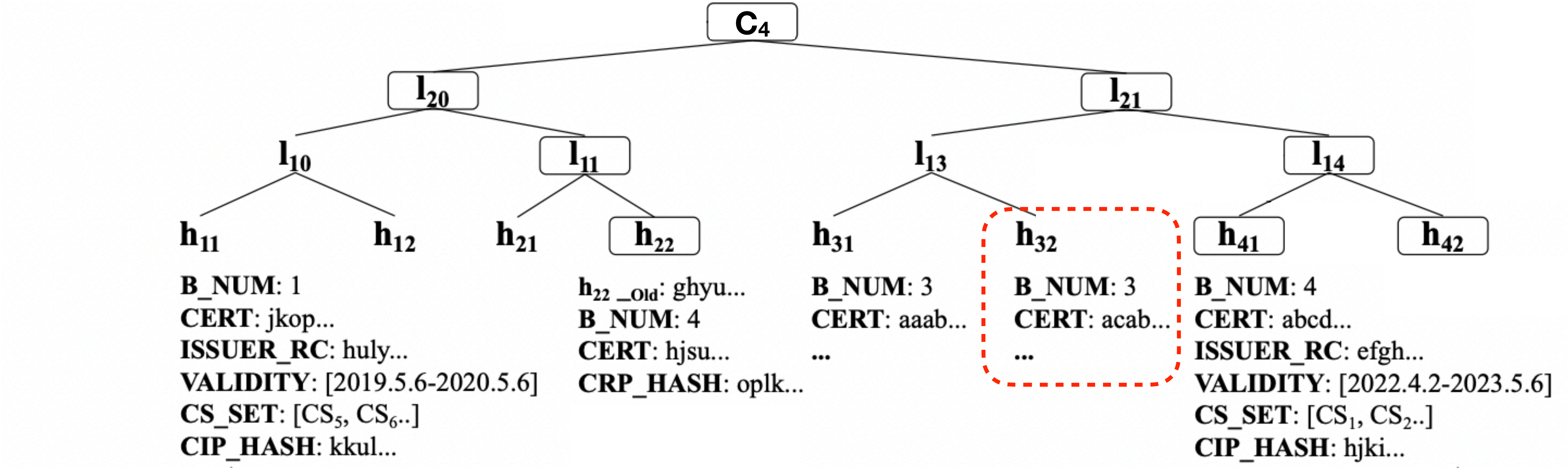
Proof of Presence

- A pruned tree that contains the leaf entry of the requested certificate and the intermediate nodes needed to reconstruct the commitment

- **Verification process**
  - an INR holder asks whether a certificate exists
  - the Monitor will return a proof of presence
  - the INR holder reconstructs a *commitment C′*
  - **then the INR holder accesses *the commitment update files* provided by other Monitors to check the authenticity of *C′***

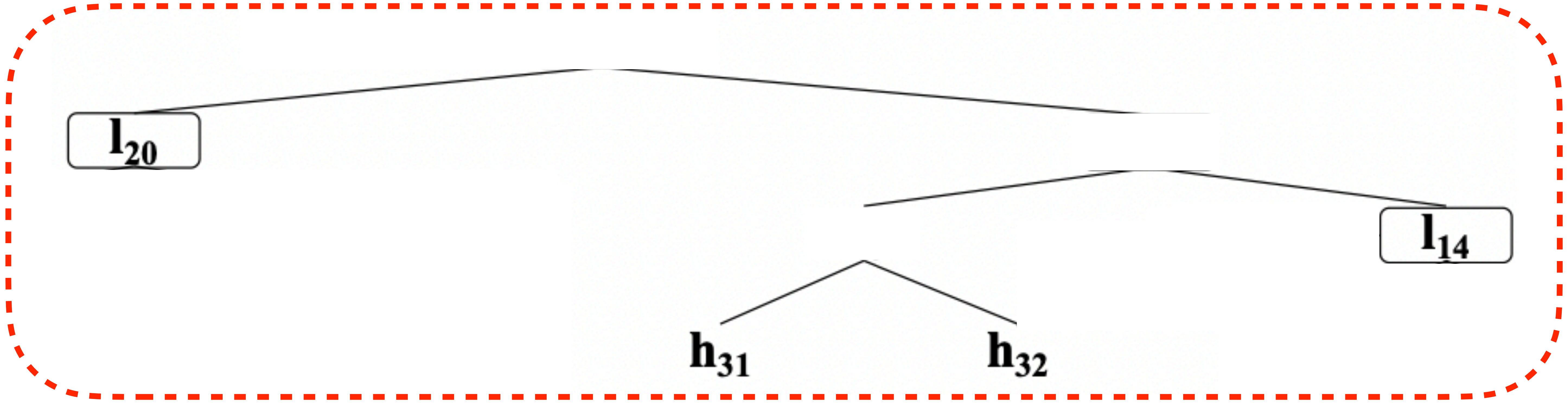
Proof of Presence: example

- INR holder asks whether the certificate in h32=⟨B_NUM = 3,CERT = acab…⟩ exists



Proof of Presence: example

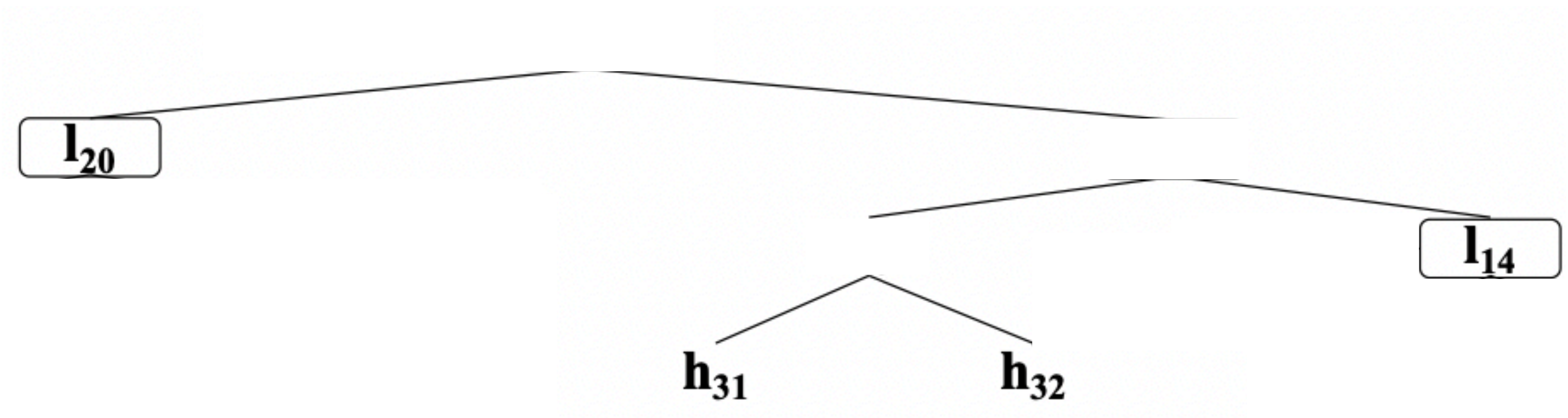
- The Monitor will return a pruned tree that contains the entry of h_{32} and the hash of h_{31}, intermediate nodes l_{14} and l_{20} to the INR holder



proof of presence

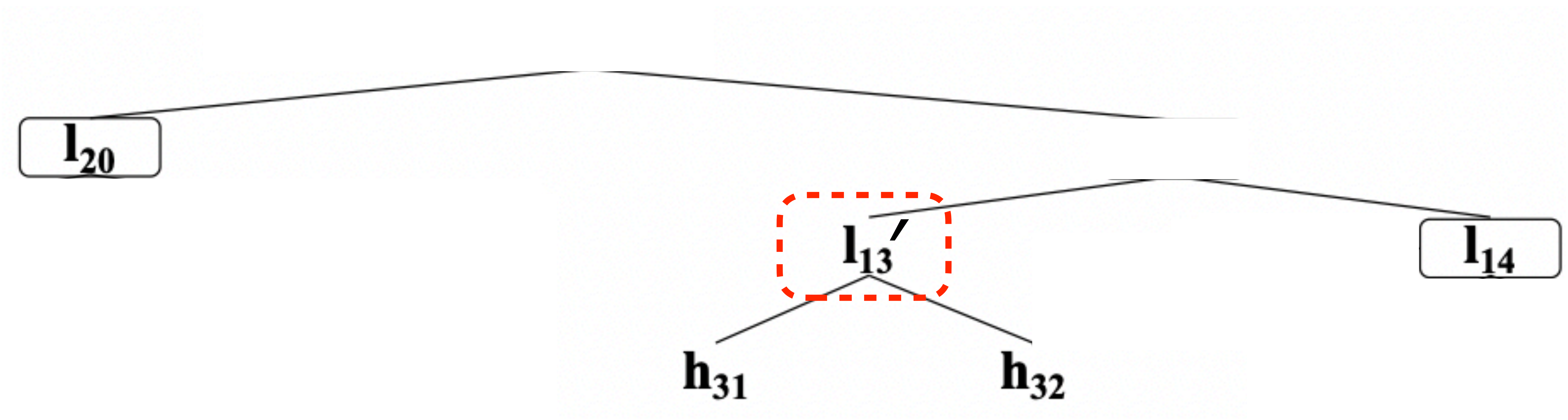
Proof of Presence: example

- The INR holder can easily reconstruct the *commitment C_4'*



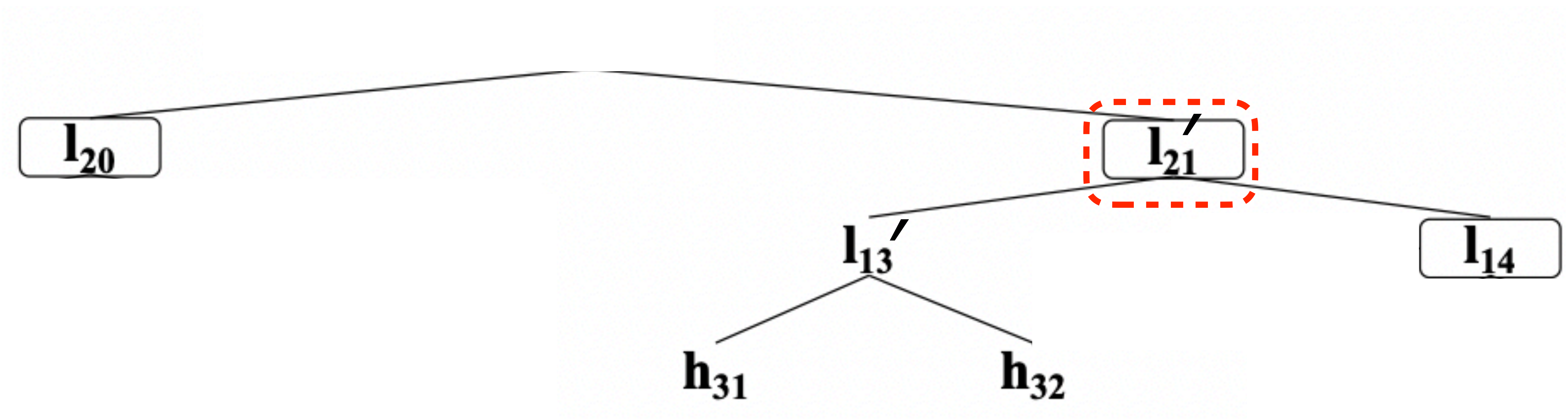
Proof of Presence: example

- The INR holder can easily reconstruct the *commitment C_4'*



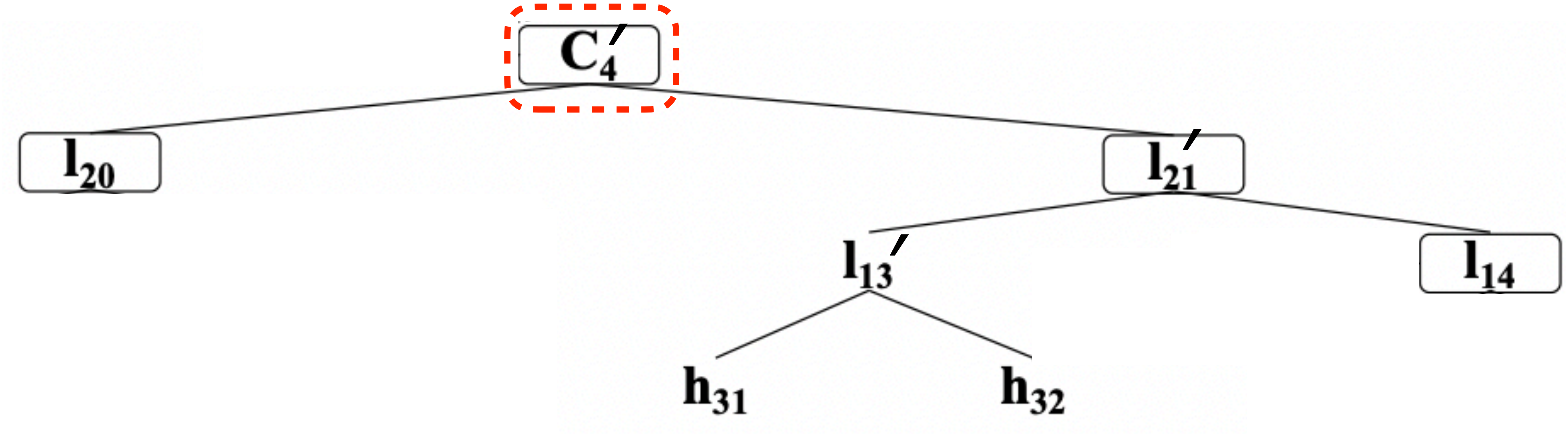
Proof of Presence: example

- The INR holder can easily reconstruct the *commitment C_4'*



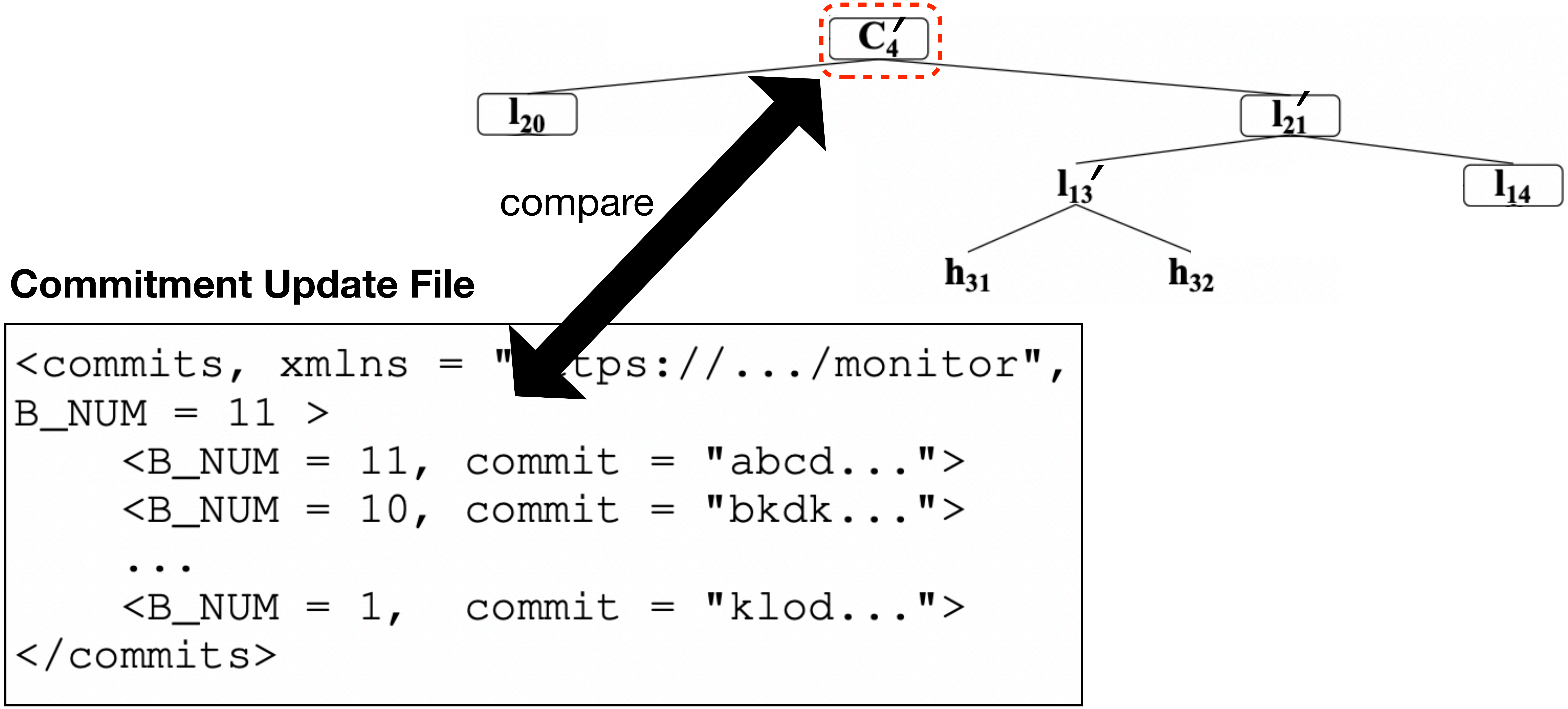
Proof of Presence: example

- The INR holder can easily reconstruct the *commitment C_4'*



Proof of Presence: example

- Then the INR holder accesses the commitment update files provided by other Monitors to check the authenticity of C_4'



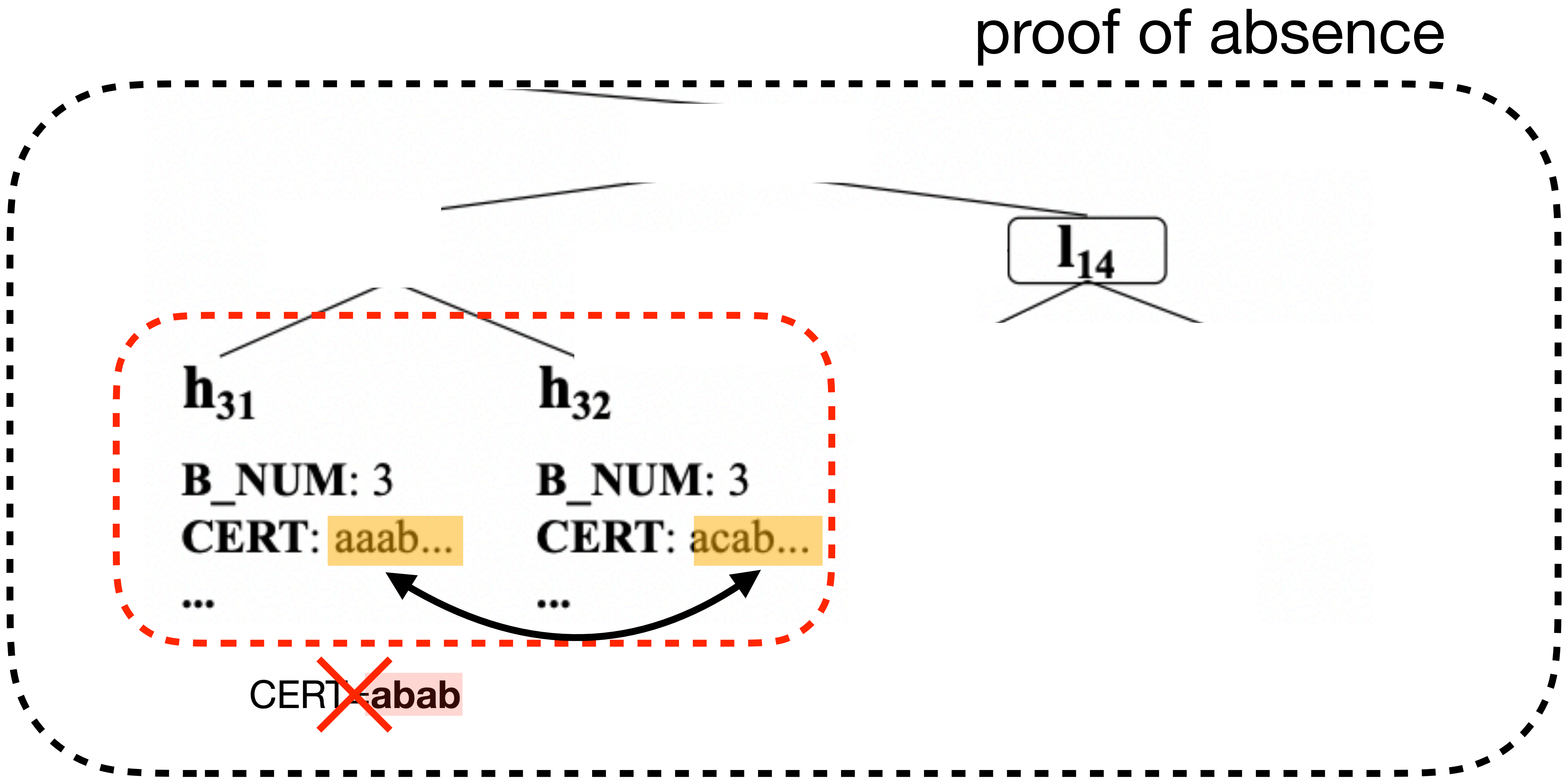
Commitment Update File

```
<commits, xmlns = "  tps://.../monitor",
B_NUM = 11 >
    <B_NUM = 11, commit = "abcd...">
    <B_NUM = 10, commit = "bkdk...">
    ...
    <B_NUM = 1,  commit = "klod...">
</commits>
```

Proof of Absence

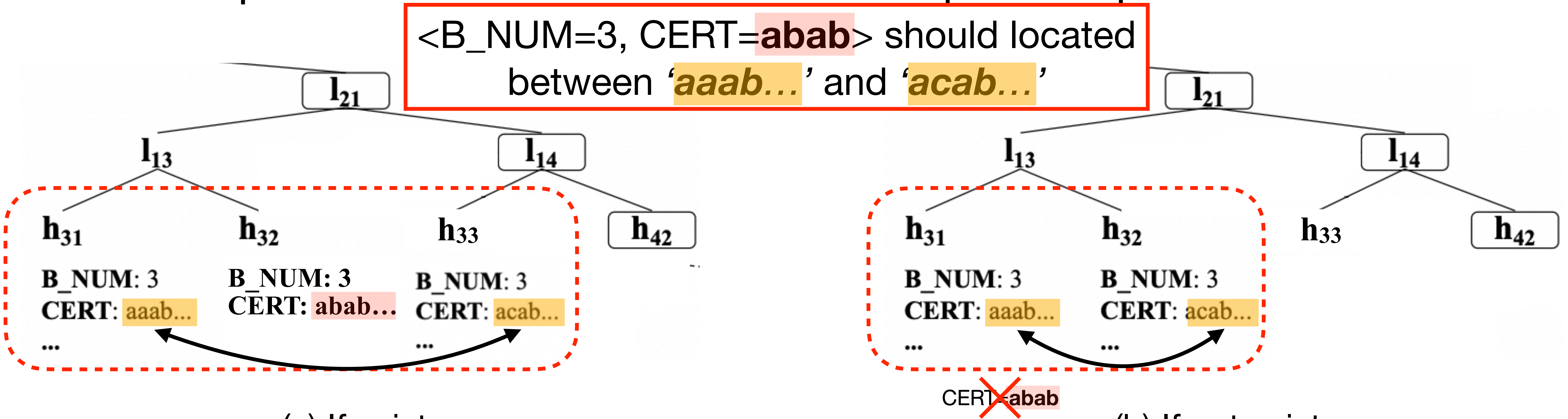
- A pruned tree
  - contains **two consecutive leaf nodes** where **the hash of the queried certificate is between the hashes of them**
  - and the intermediate nodes needed to reconstruct the commitment
- Verification process is the same as that of the proof of presence

<B_NUM=3, CERT=**abab**> ?



Proof of Absence

- A pruned tree
  - contains **two consecutive leaf nodes** where **the hash of the queried certificate is between the hashes of them**
  - and the intermediate nodes needed to reconstruct the commitment
- Verification process is the same as that of the proof of presence



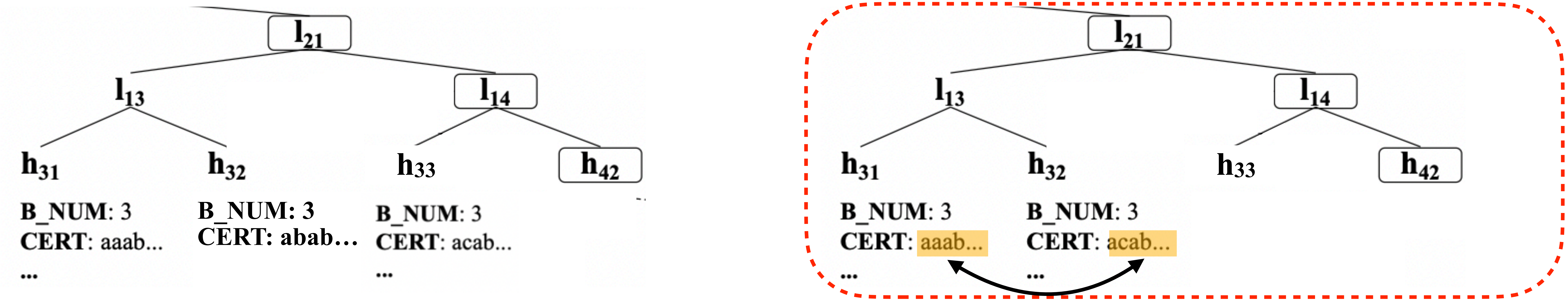
<B_NUM=3, CERT=**abab**> should located between '***aaab…***' and '***acab…***'

(a) If exist

(b) If not exist

Proof of Absence

- A pruned tree
  - contains **two consecutive leaf nodes** where **the hash of the queried certificate is between the hashes of them**
  - and the intermediate nodes needed to reconstruct the commitment
- Verification process is the same as that of the proof of presence



(a) If exist

no such certificate exists → proof of absence

(b) If not exist

Proof of Consistency

- A pruned tree
  - contains a **Certificate Update List** (**CUL**, a list of newly inserted or updated entries) and a proof (hashes needed to reconstruct a commitment)
  - prove that the commitment of the current M-tree is indeed evolved from the previous commitment

Proof of Consistency

- Verification process
  - **A Relying Party (RP) submits <B_num=3, c=C_3>, the RP has completed the synchronization of the first three blocks**
  - The Monitor will return a proof of consistency whose commitment is C_4
  - The RP verify that the reconstructed commitment C'_4 is trusted
  - The RP verify that the reconstructed commitment C'_4 is evolved from C_3

Proof of Consistency

- Verification process
  - A Relying Party (RP) submits <B_num=3, c=C_3>, the RP has completed the synchronization of the first three blocks
  - **The Monitor will return a proof of consistency whose commitment is C_4**
  - The RP verify that the reconstructed commitment C'_4 is trusted
  - The RP verify that the reconstructed commitment C'_4 is evolved from C_3

Proof of Consistency

- Verification process
  - A Relying Party (RP) submits <B_num=3, c=C_3>, the RP has completed the synchronization of the first three blocks
  - The Monitor will return a proof of consistency whose commitment is C_4
  - **The RP verify that the reconstructed commitment C'_4 is trusted**
  - The RP verify that the reconstructed commitment C'_4 is evolved from C_3

Proof of Consistency

- Verification process
  - A Relying Party (RP) submits <B_num=3, c=C_3>, the RP has completed the synchronization of the first three blocks
  - The Monitor will return a proof of consistency whose commitment is C_4
  - The RP verify the authenticity of the reconstructed commitment C'_4
  - **The RP verify whether the reconstructed commitment C'_4 is evolved from C_3**

Proof of Consistency

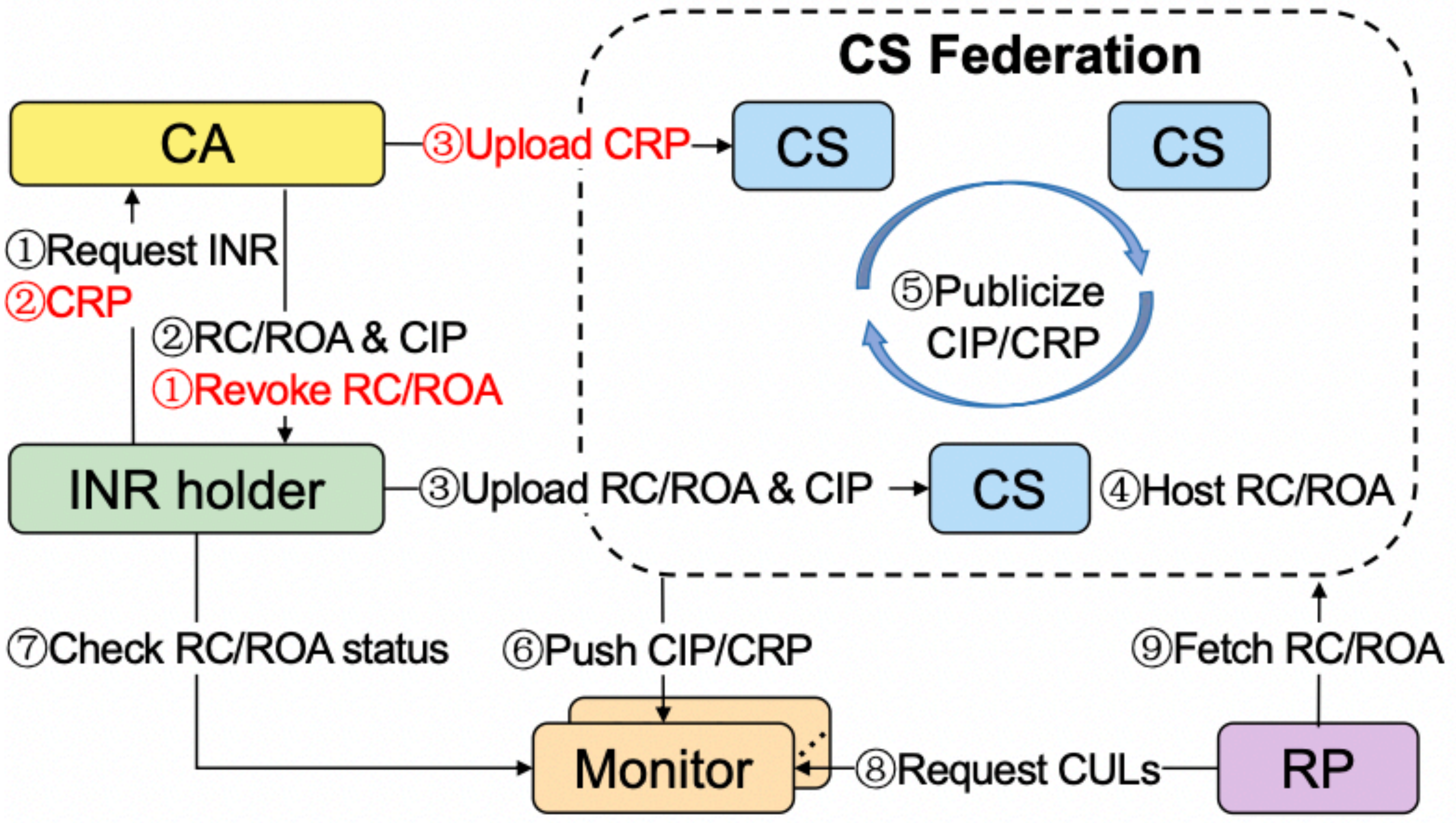
- Verification process
  - A Relying Party (RP) submits <B_num=3, c=C_3>, the RP has completed the synchronization of the first three blocks
  - The Monitor will return a proof of consistency whose commitment is C_4
  - The RP verify the authenticity of the reconstructed commitment C'_4
  - **The RP verify whether the reconstructed commitment C'_4 is evolved from C_3**

    deletes inserted entries and reverts the updated entries from the pruned tree

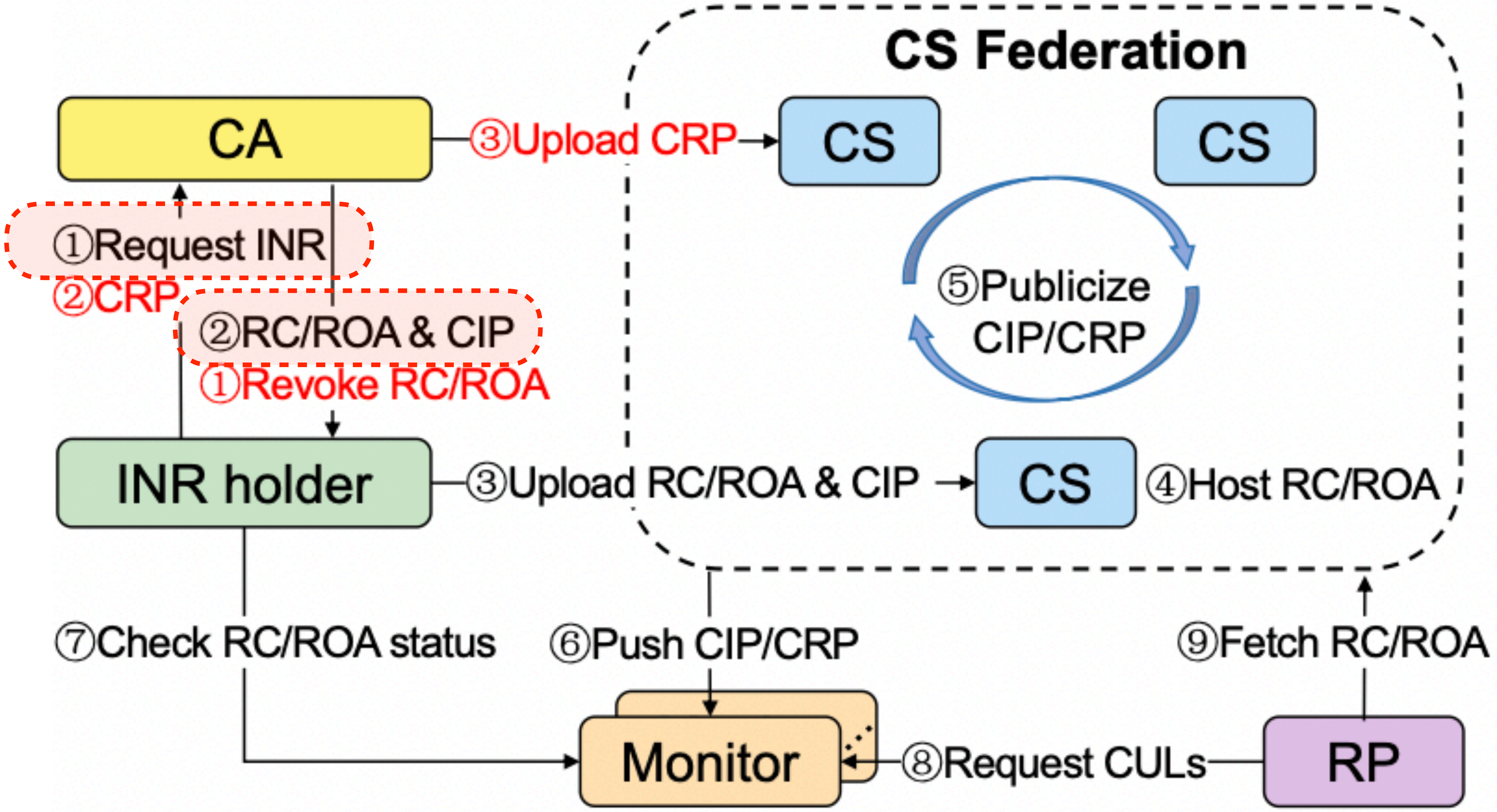
    re-calculates the commitments C′3

    checks whether it is equal to C3

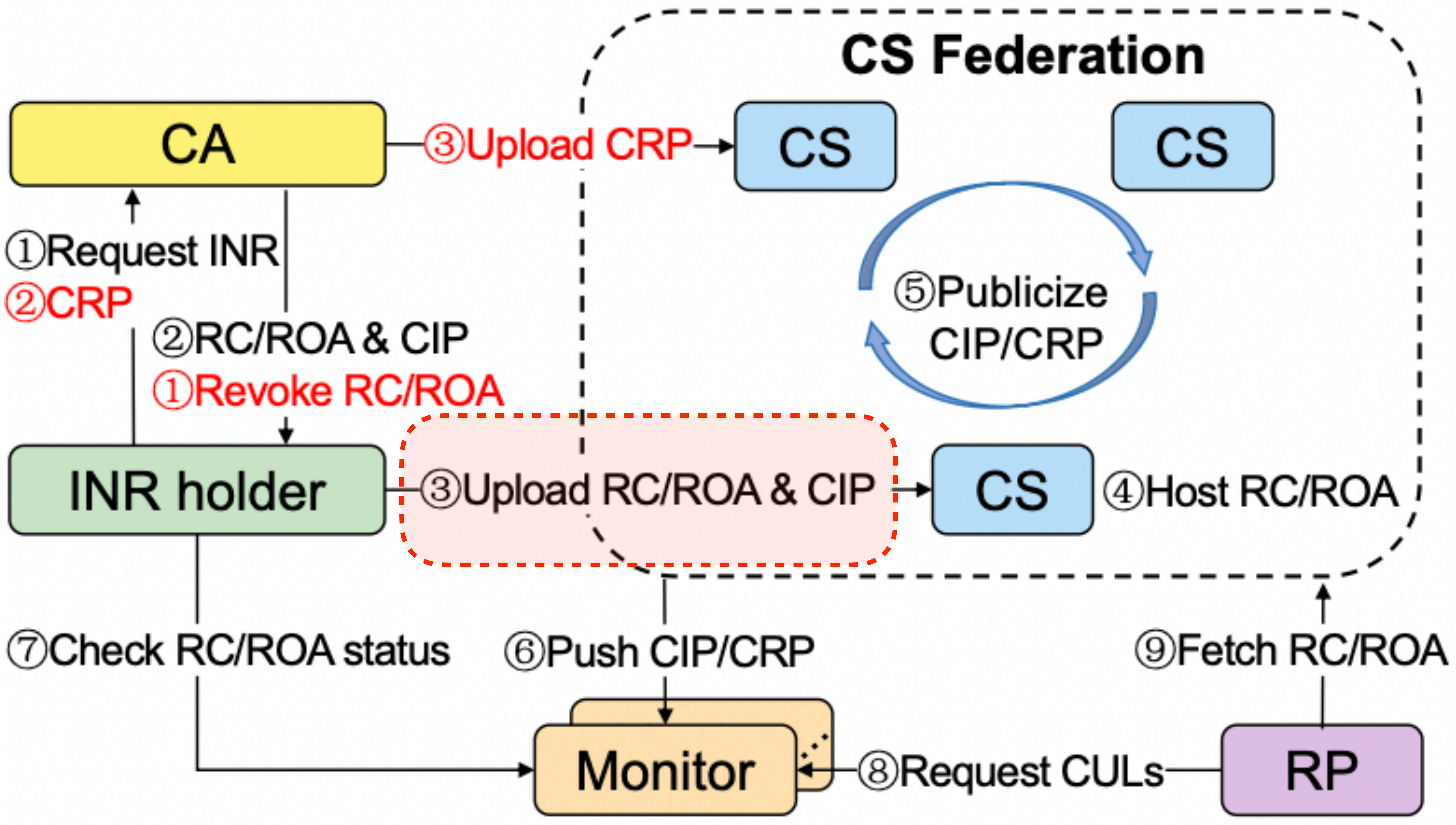
dRR Workflow



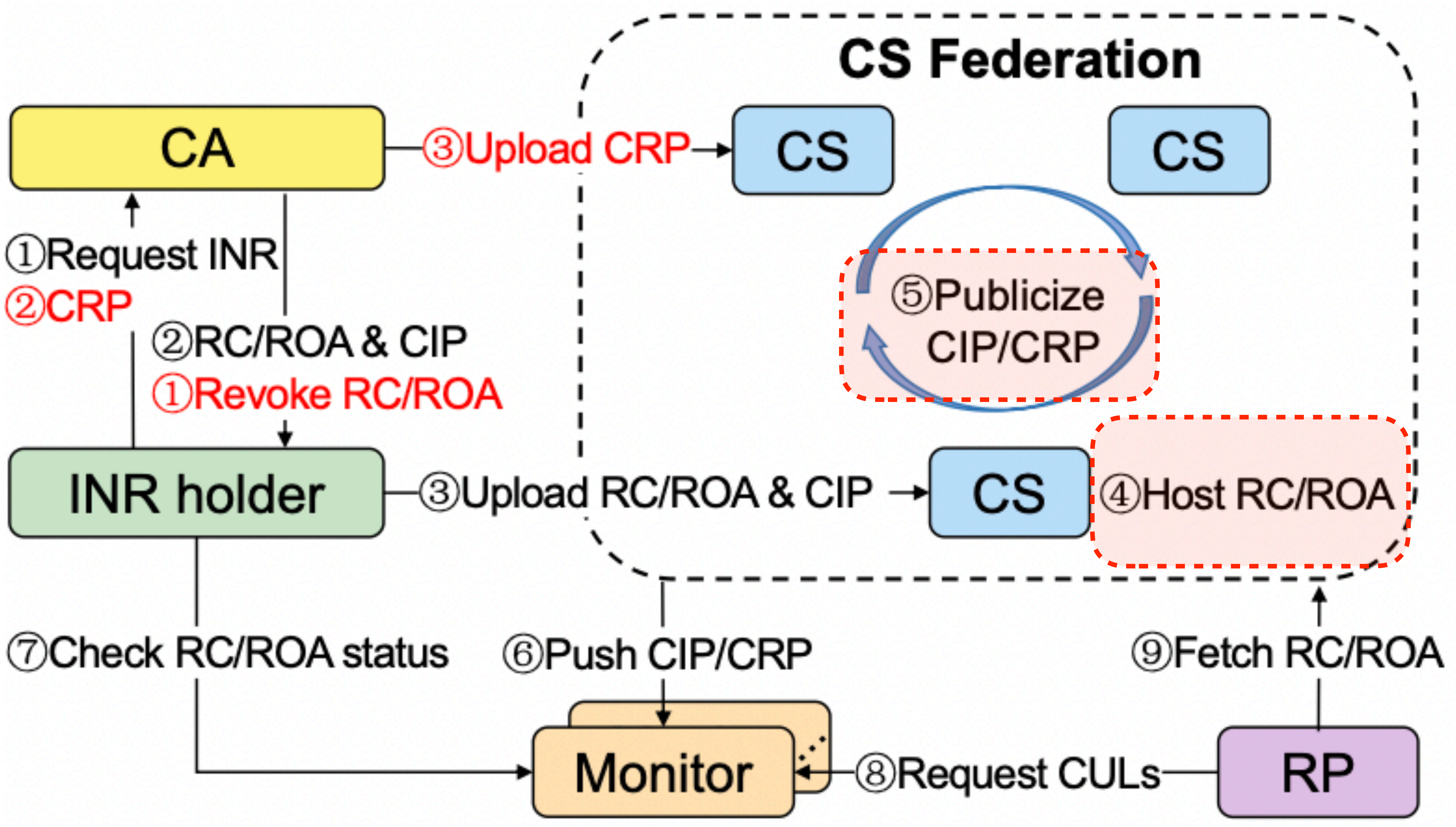
dRR Workflow



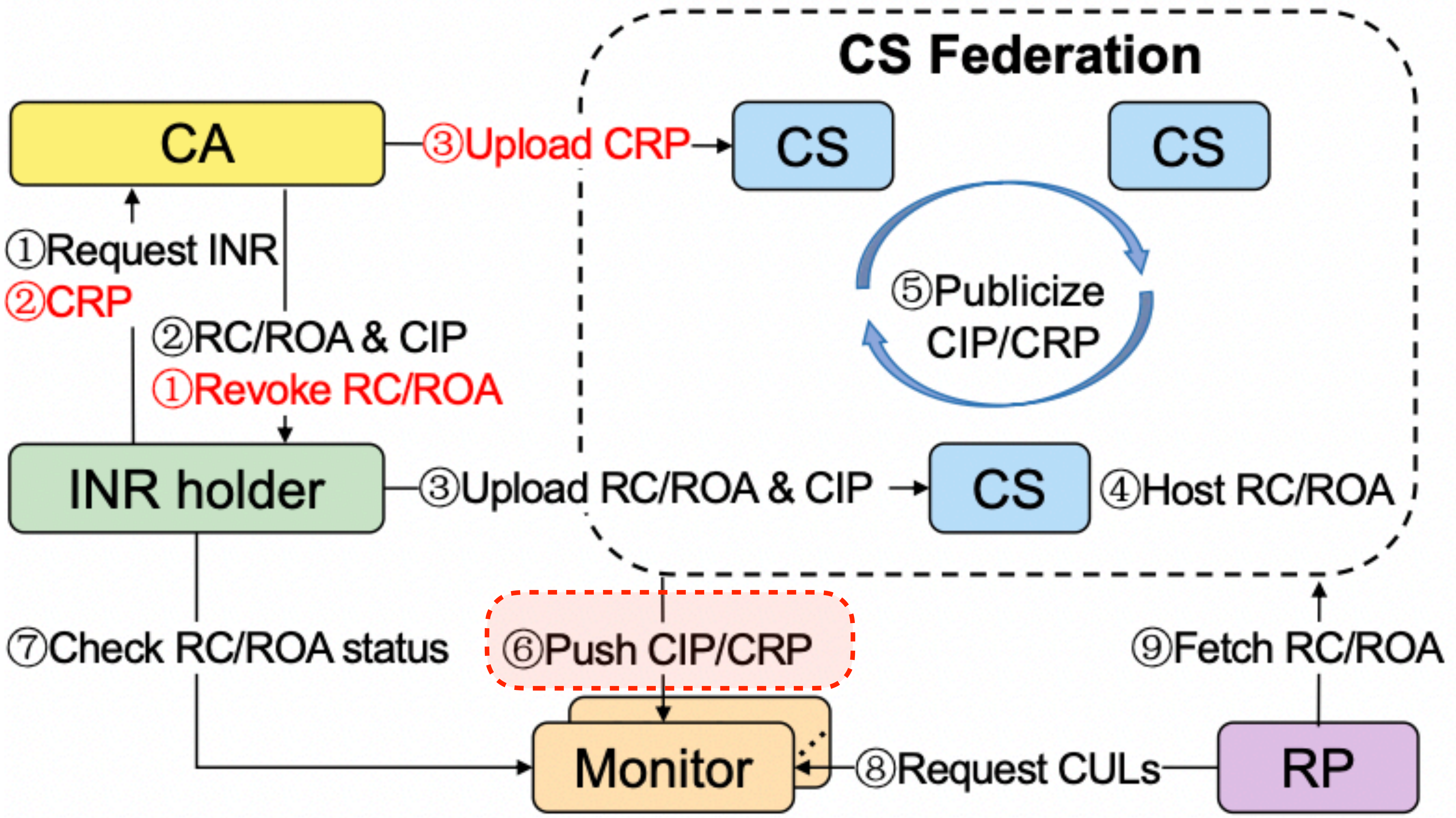
dRR Workflow



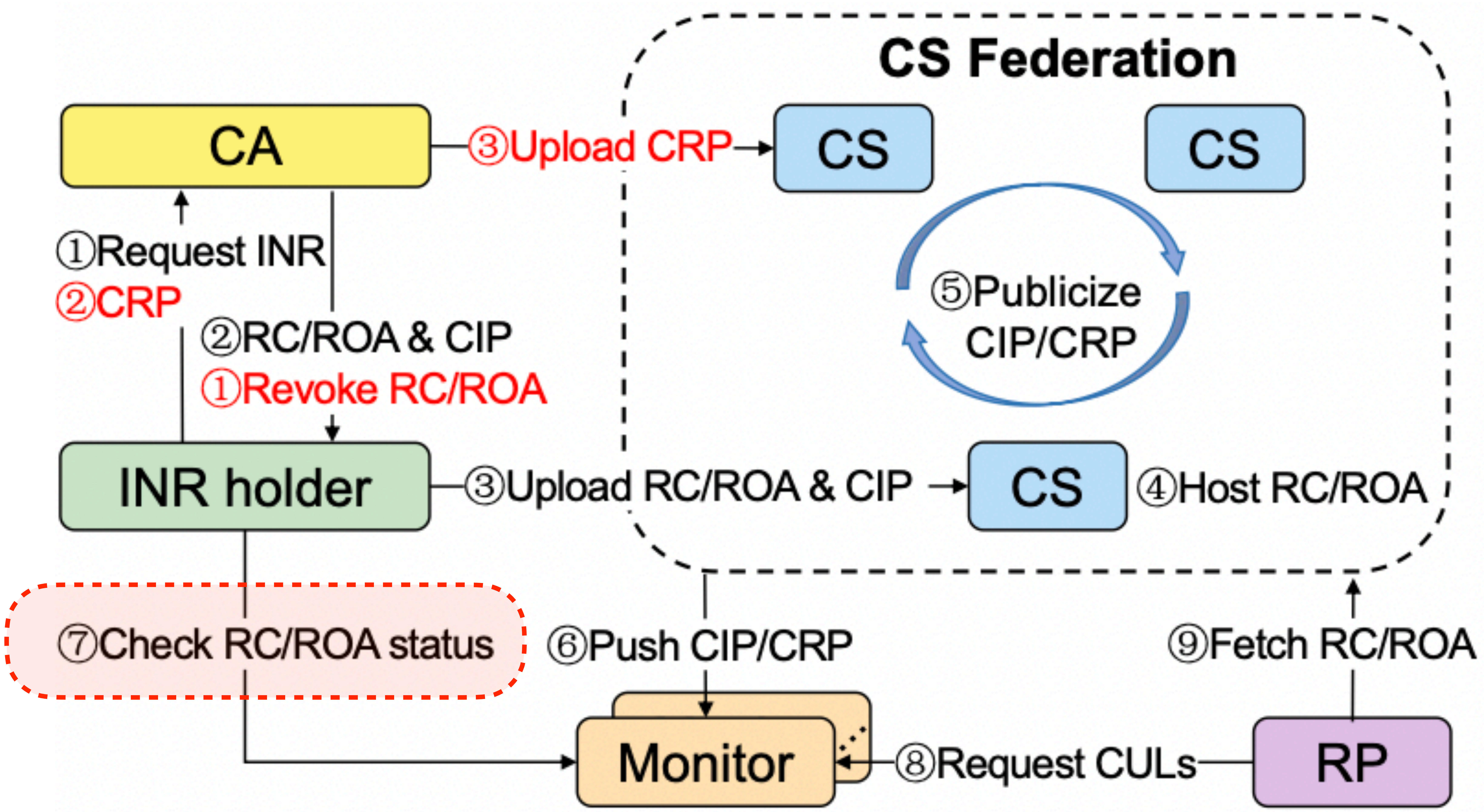
dRR Workflow



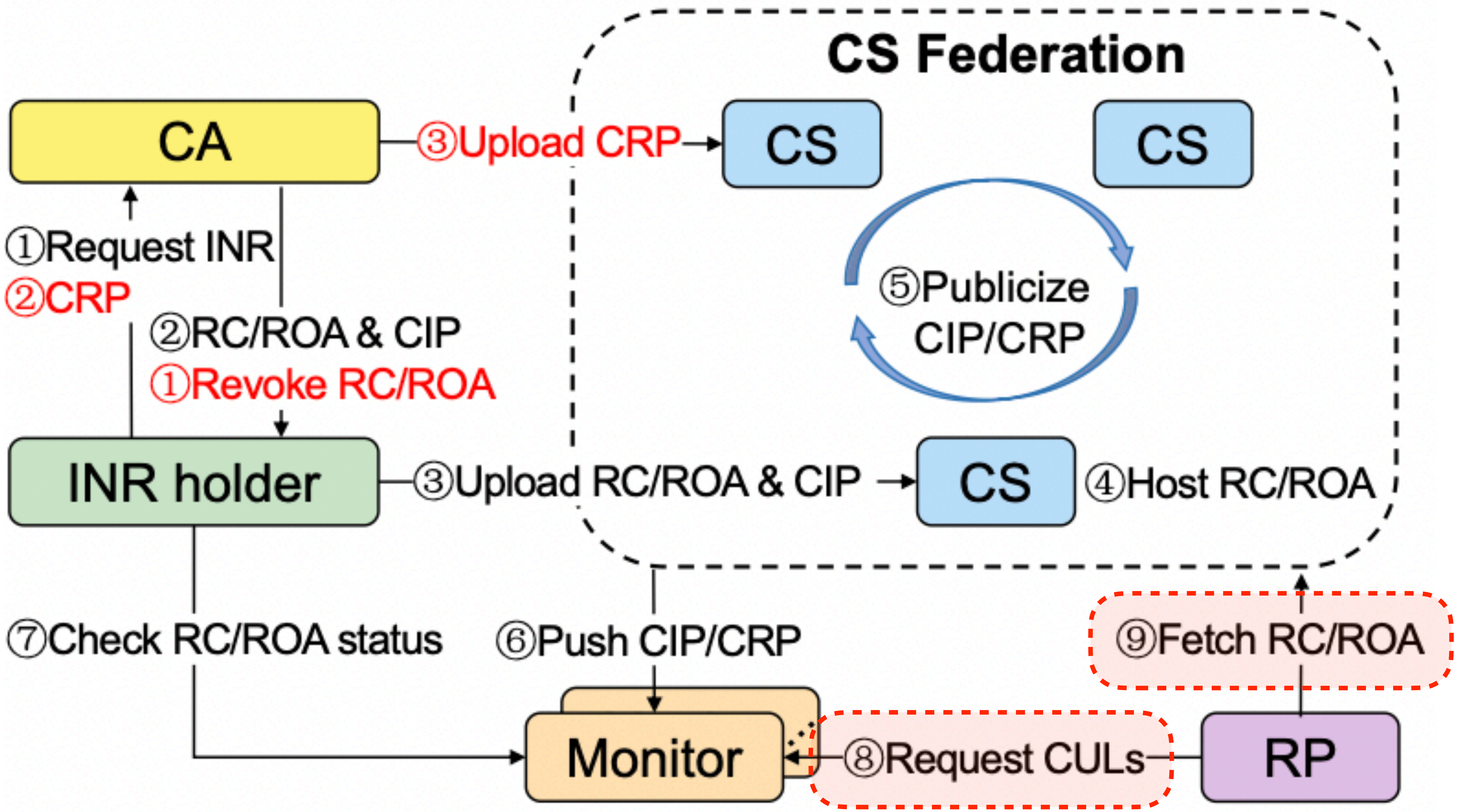
dRR Workflow



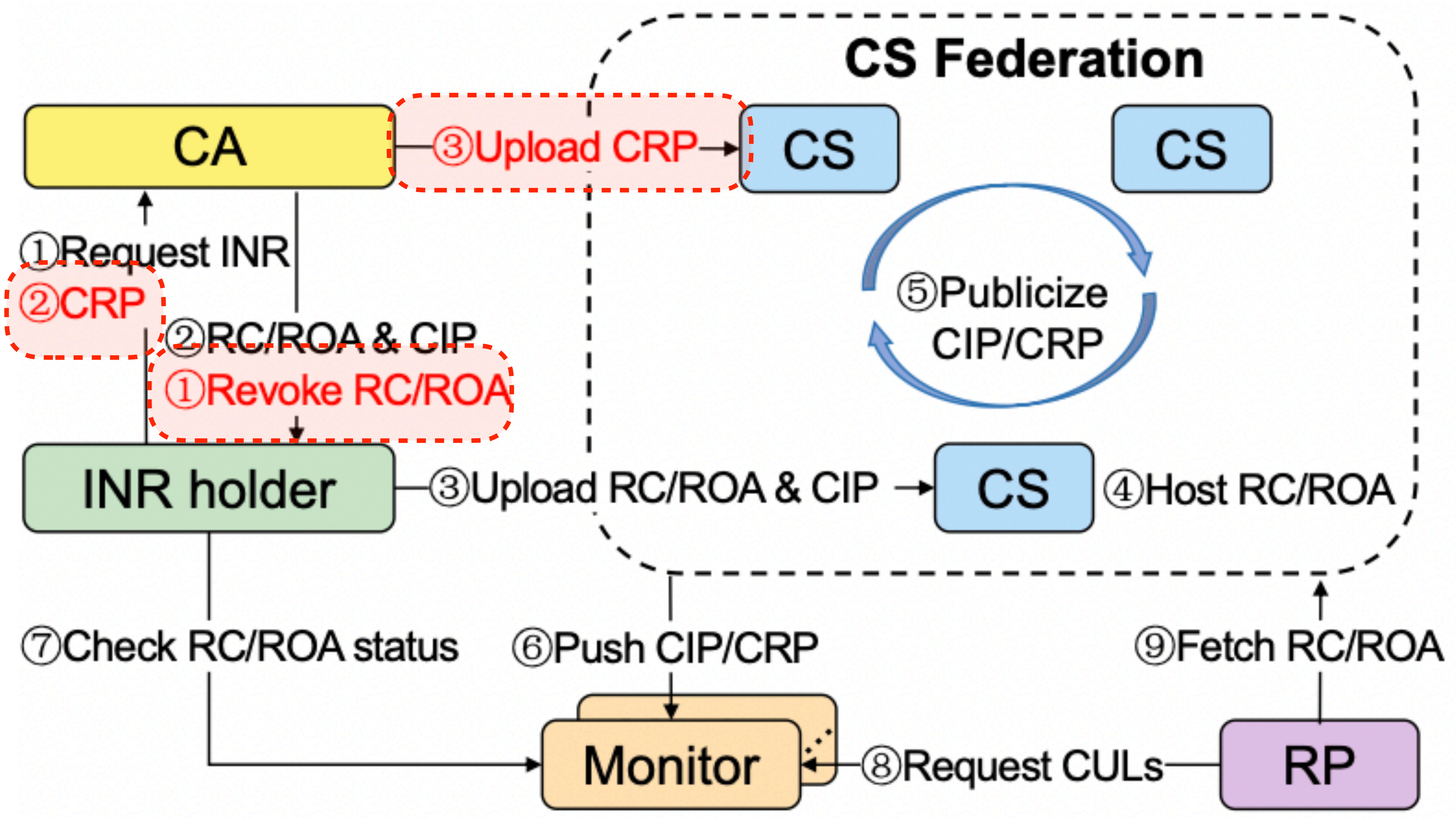
dRR Workflow



dRR Workflow



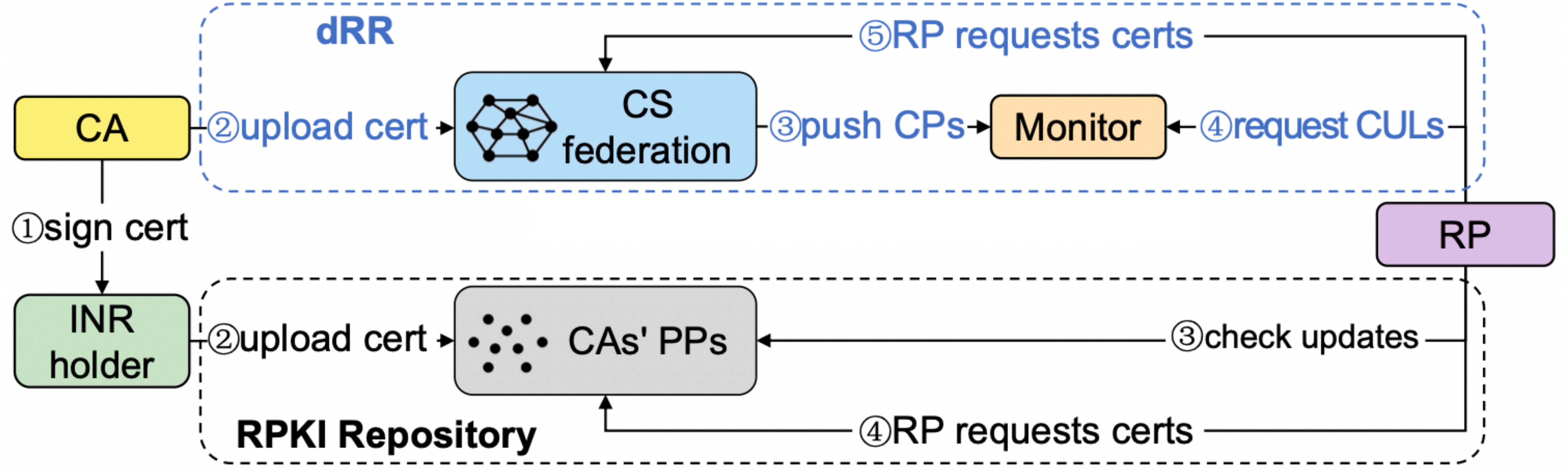
dRR Workflow



Evaluation

- Global Testbed
  - 100 server nodes across 15 countries
  - 50 nodes for CS federation and 50 nodes for Monitors

- Goal: evaluate the overhead of dRR



Evaluating CS Federation

- **Metrics**
  - throughput of the CS federation
  - the latency from a submission of a certificate policy to the confirmation
- **Baseline:**
  - the frequency of the issuance and the revocation in the current RPKI system
  - the **peak** reaches **60k/day (issued 30K + revoked 30K)**

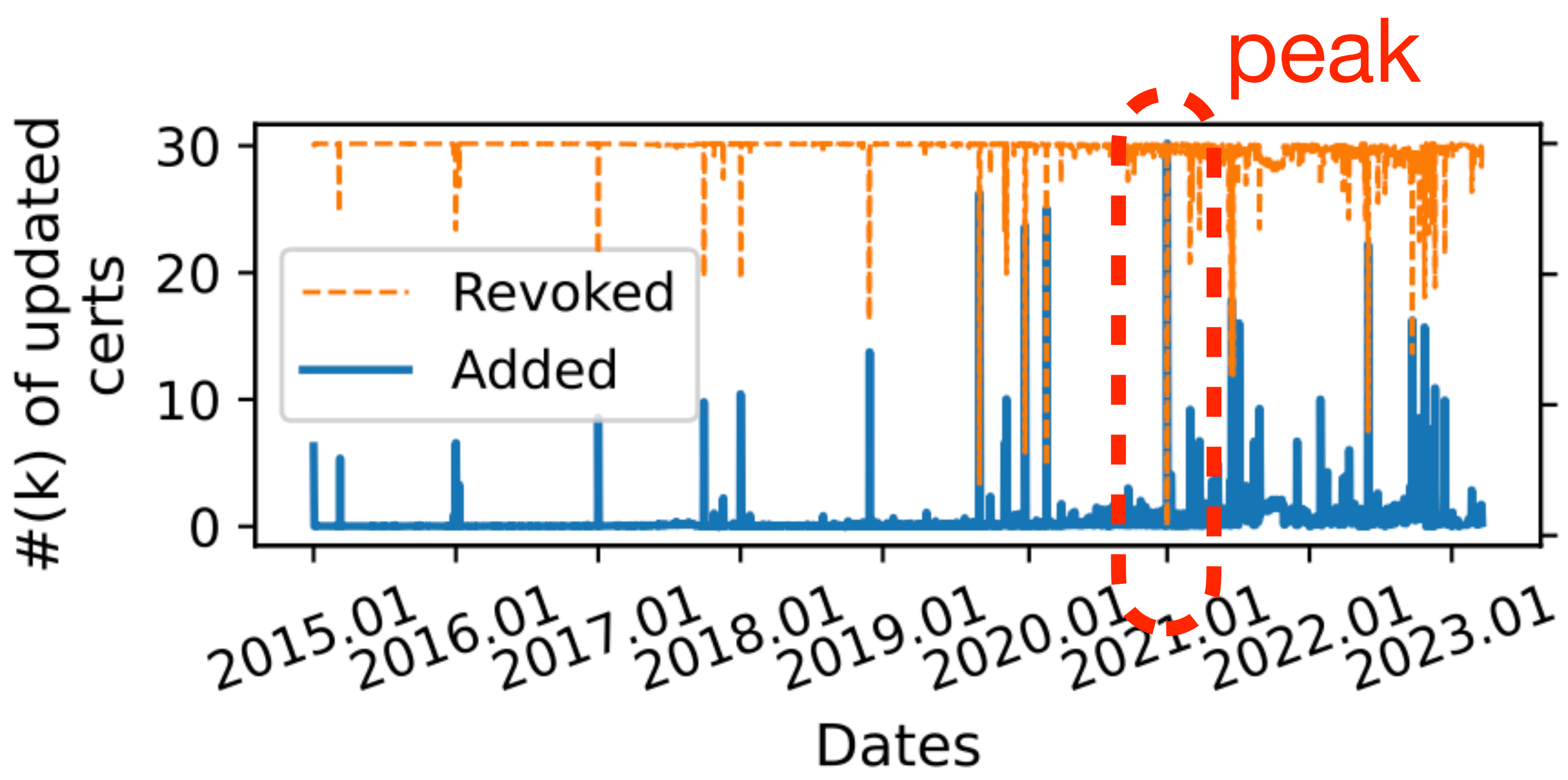


Fig. 9: The number of added and revoked certificates per day from Jan 1, 2015 to Apr 1, 2023.

Evaluating CS Federation

- **Results**
  - throughput ≈ **310/s** (i.e., 26.78 M/day) which is **450 times faster than that of the baseline**
  - **latency < 2s**

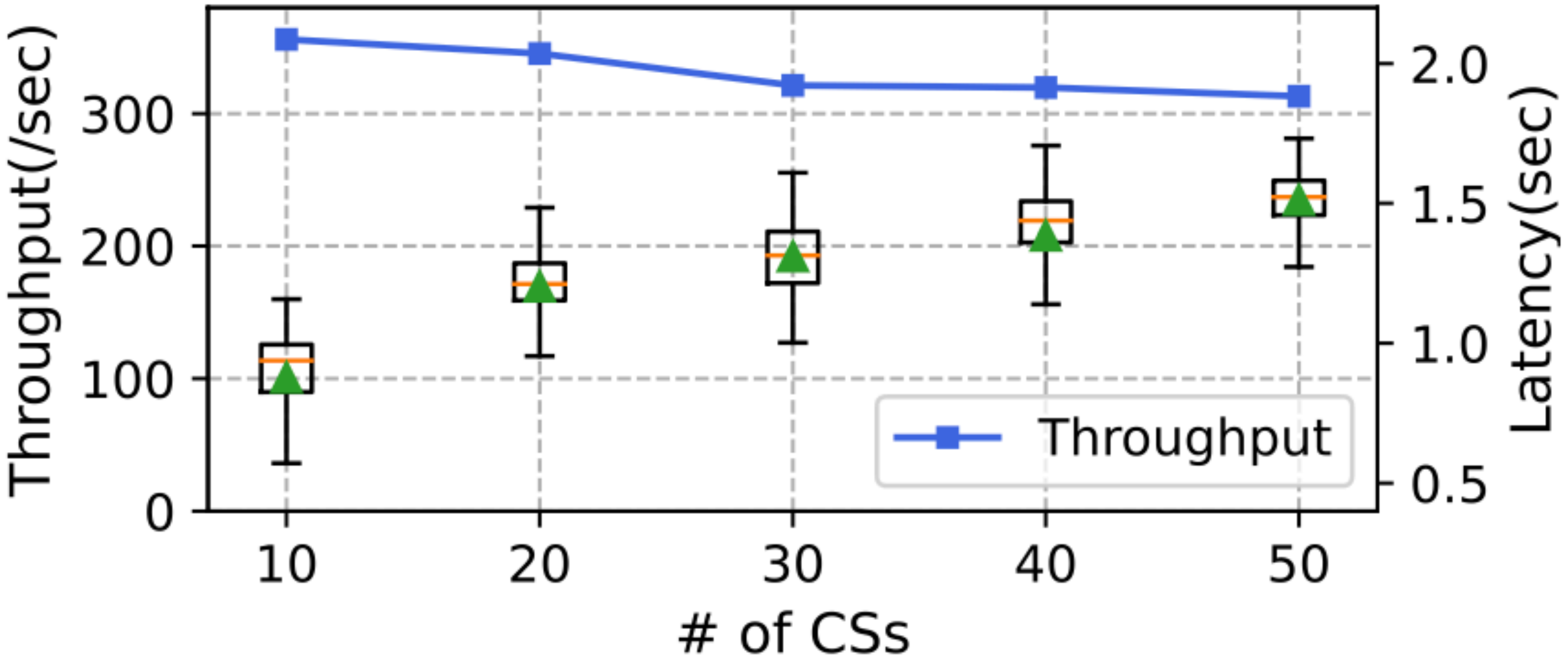


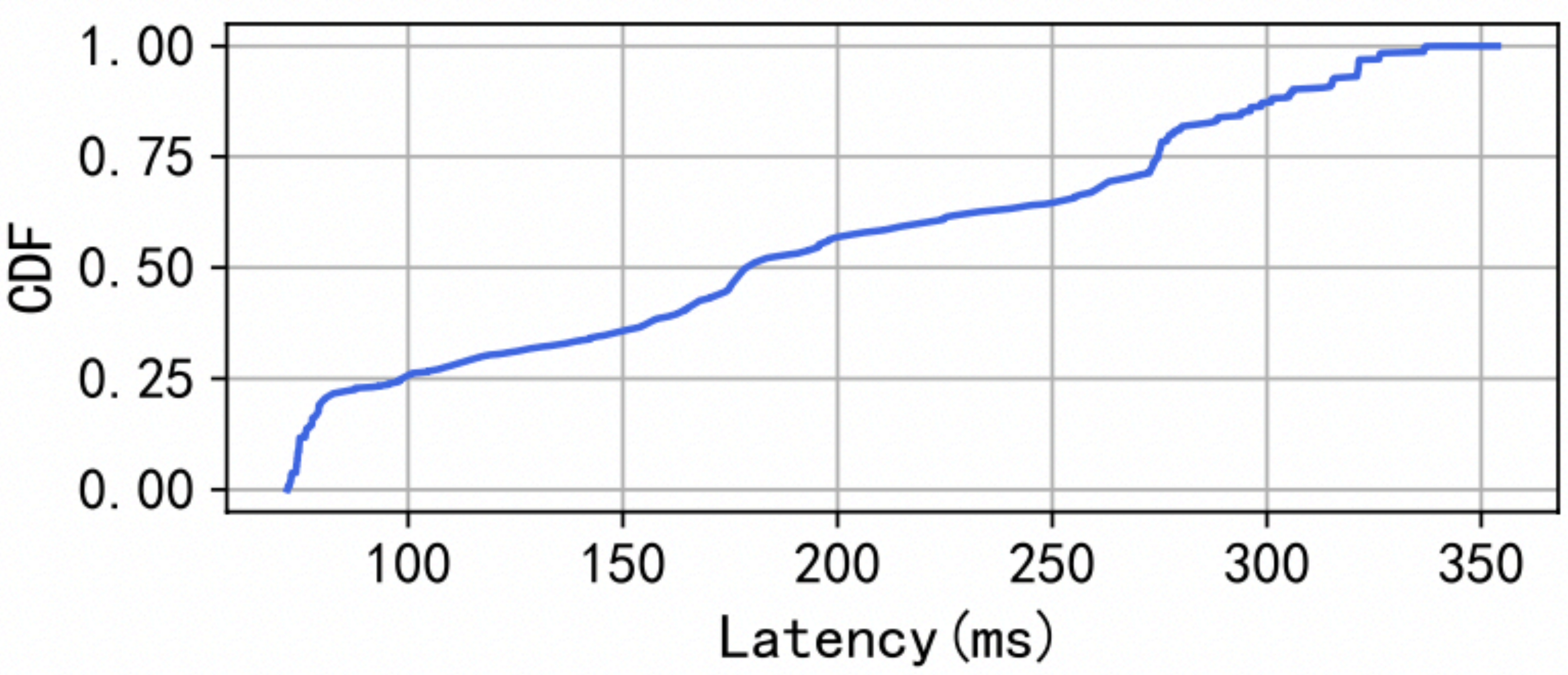
Fig. 12: The maximum throughput and the corresponding latency distribution of the system under different CS scales. Candlesticks show the maximum and minimum latency and the average latency (green triangle).

Evaluating Monitors

- **Latency**
  - from sending a block by a CS to a Monitor completing the M-Tree update about the block

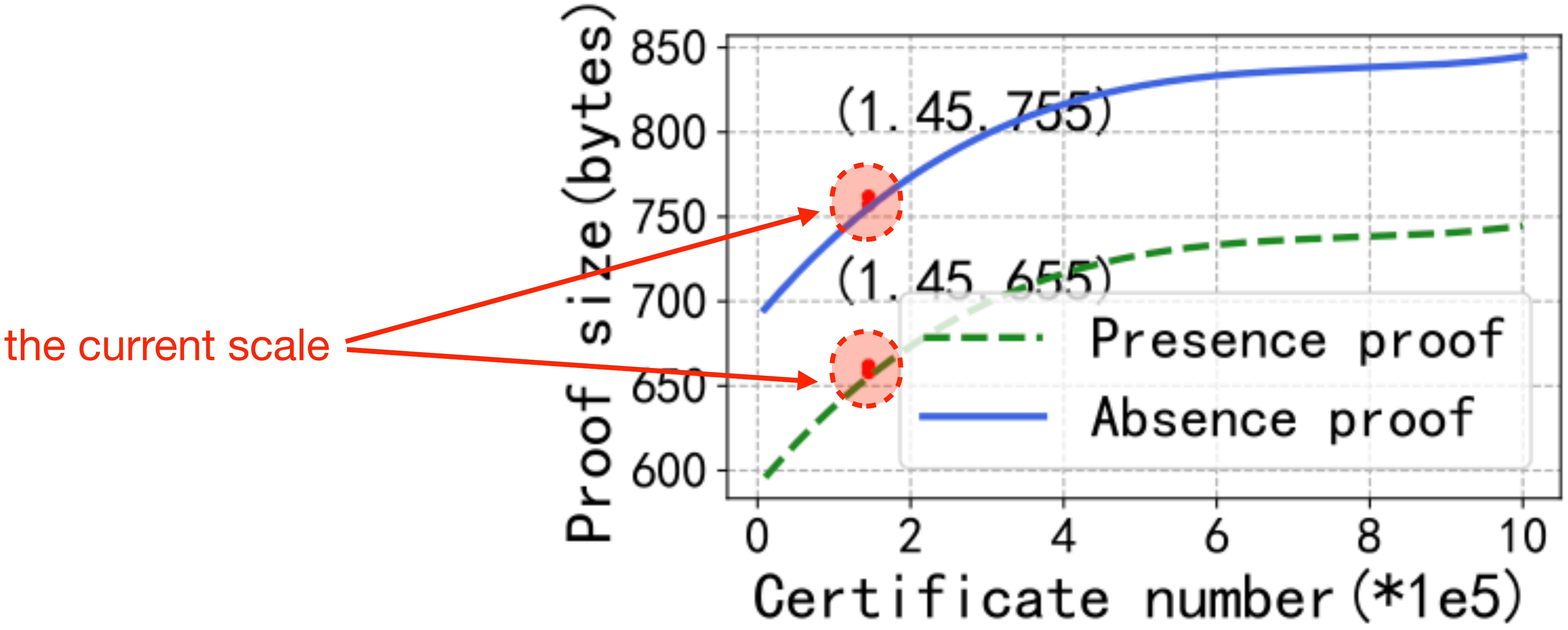
- **Setup**
  - a CS server in Silicon Valley serves 50 Monitors distributed in 15 countries (regions)
  - the CS continuously pushes 10,000 new blocks to 50 Monitors

- **Result**
  - the Monitor can **complete the update** of its M-Tree for this block **within 500ms**



Evaluating Monitors

- ***The size of proofs***
  - grows logarithmically with the number of total certificates
  - at the current RPKI certificate scale, both presence and absence proof sizes are within 1 KB



Summary

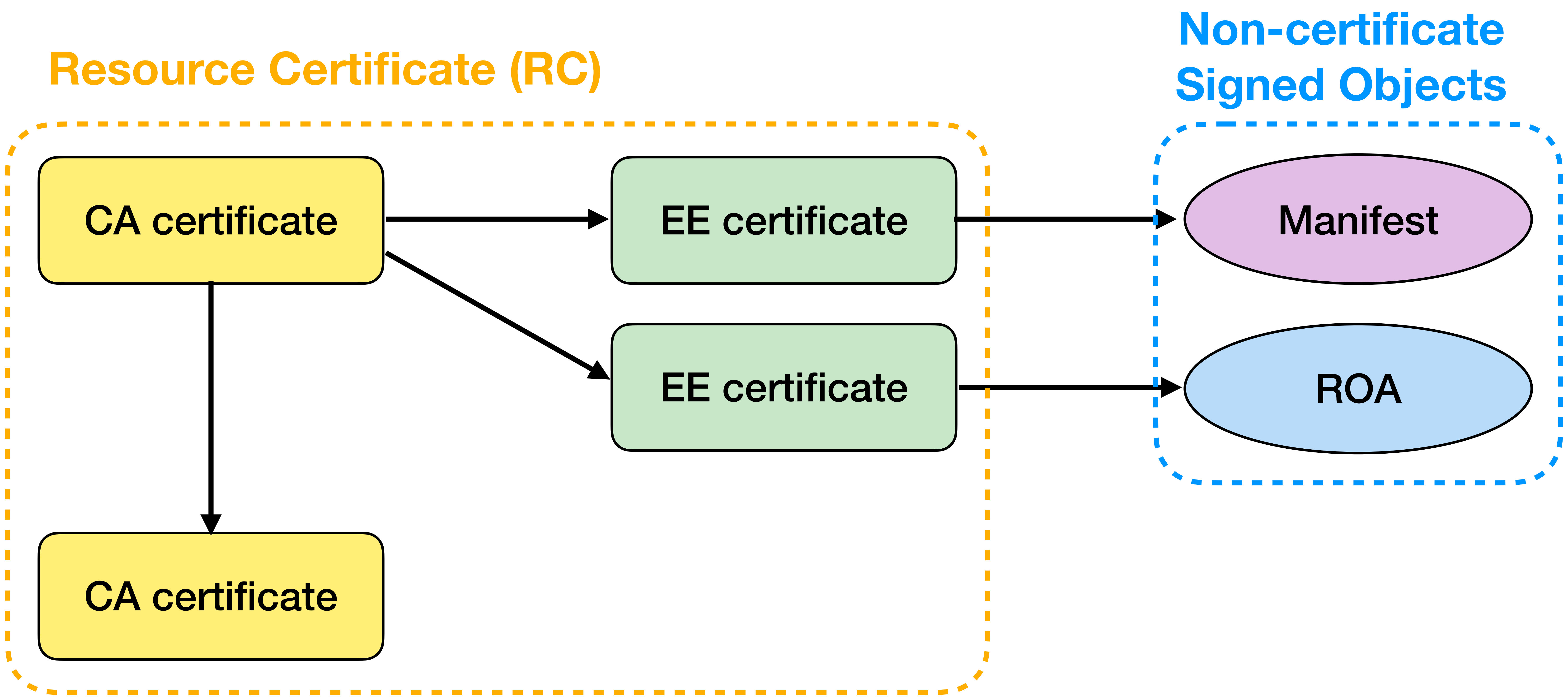
- Conduct the first data-driven RPKI-threat analysis
  - uncover three key problems of the current RPKI repository
- Propose dRR to tackle the problems of the RPKI repository
  - design an RPKI-compatible architecture to enhance security, robustness, and scalability of the RPKI Repository
- Implement a prototype of dRR and evaluate it on a global testbed with 100 nodes
  - show that the new security features of dRR introduce minimal overhead

Q&A

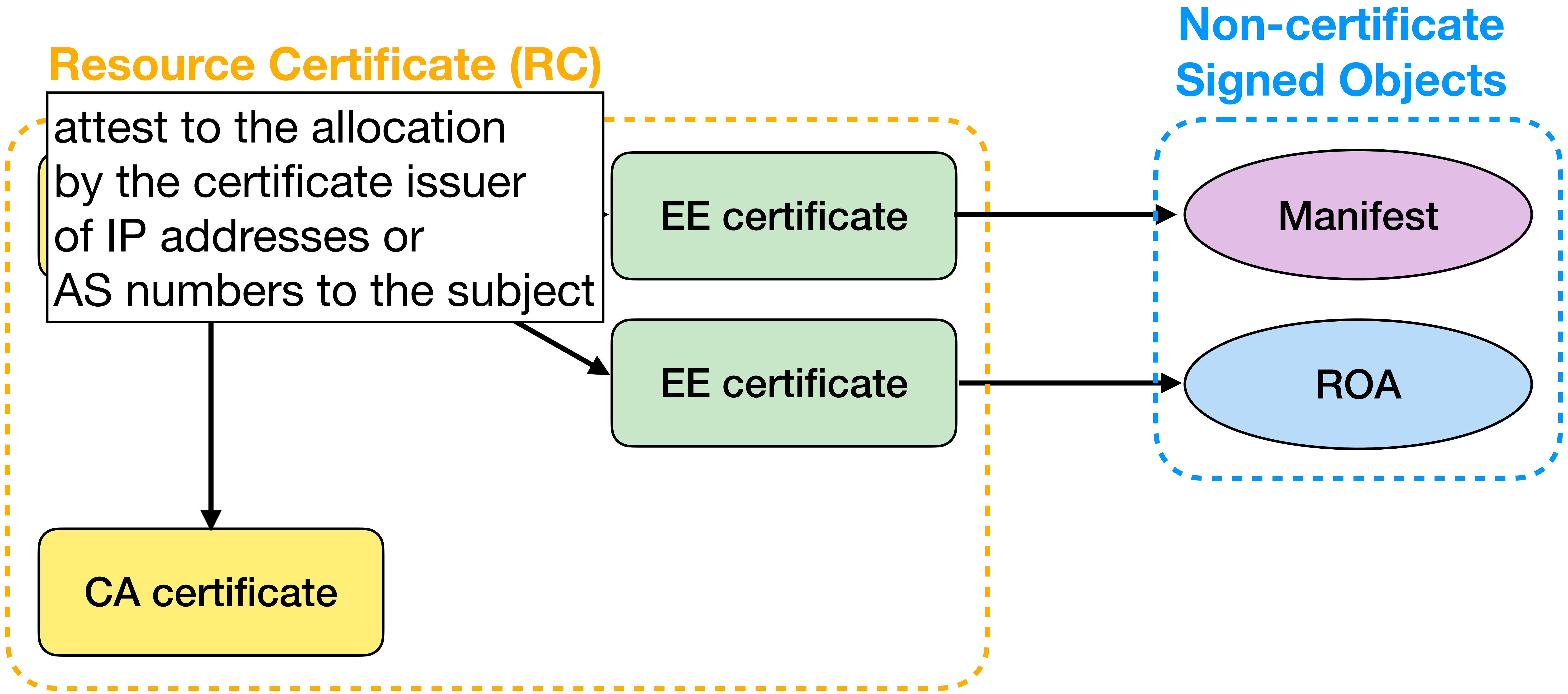
Backup

RPKI

RPKI Objects

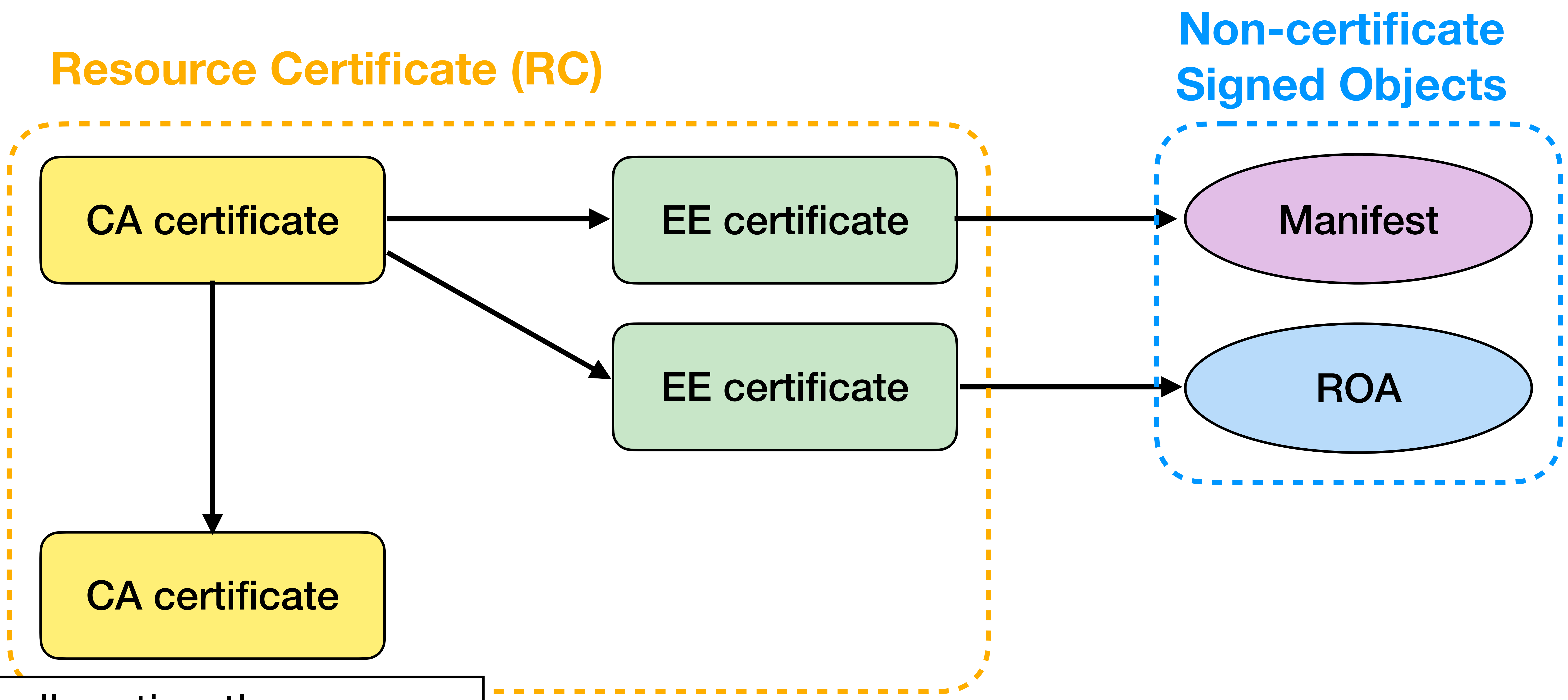


RPKI Objects



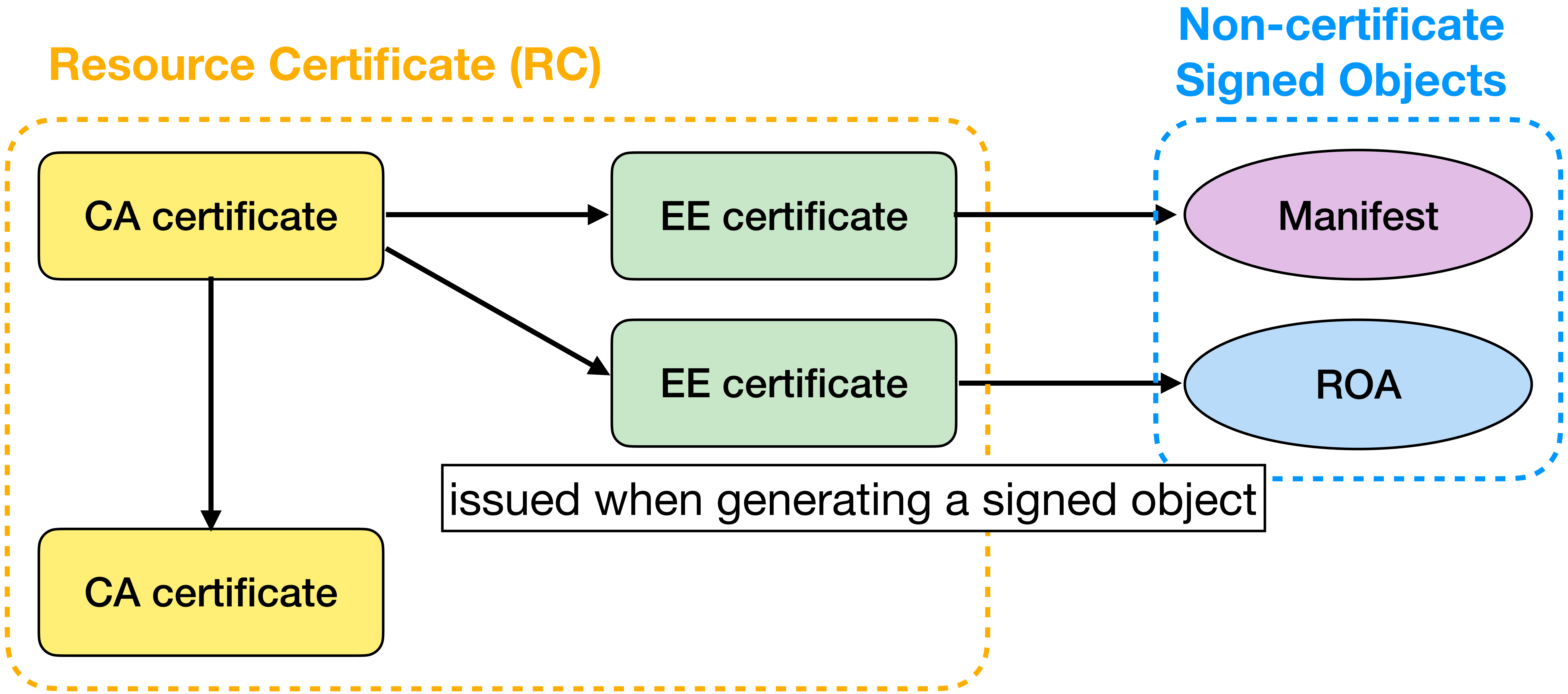
attest to the allocation by the certificate issuer of IP addresses or AS numbers to the subject

RPKI Objects

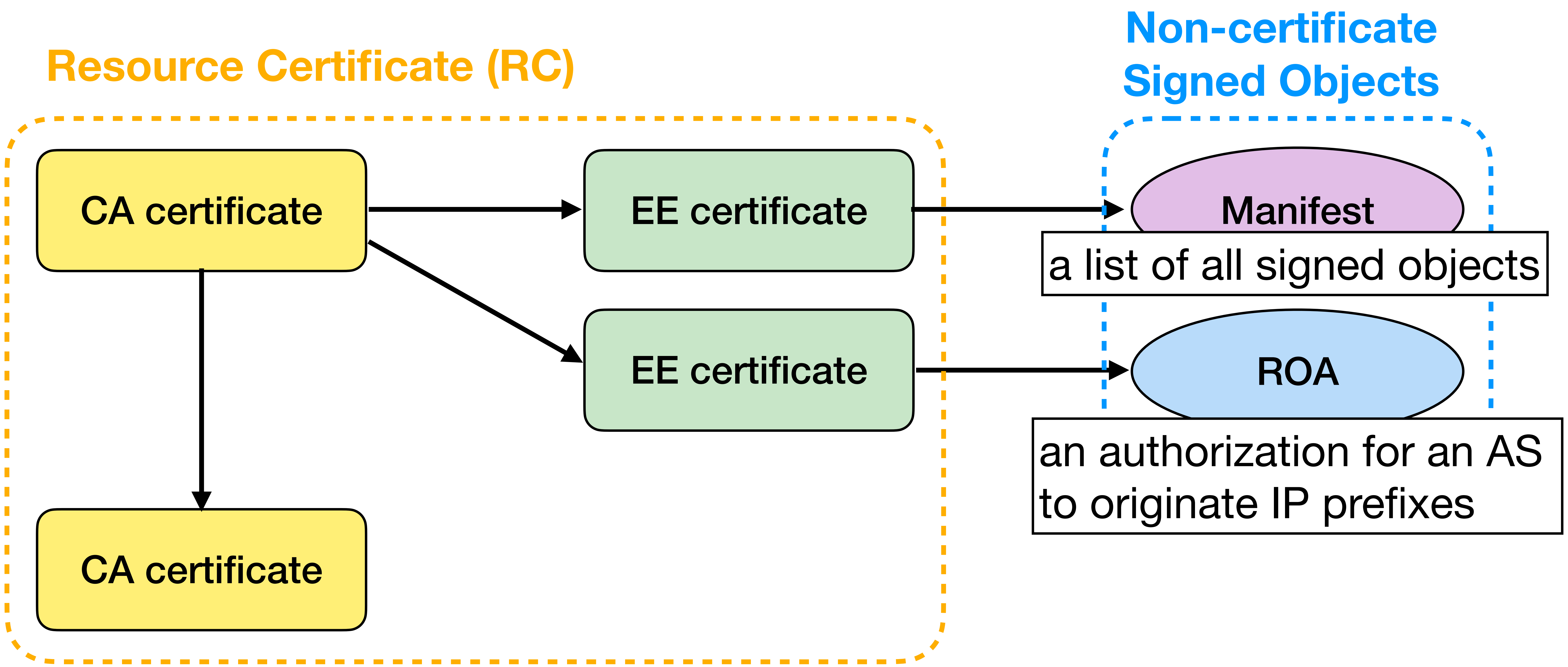


issued when sub-allocating the resource

RPKI Objects



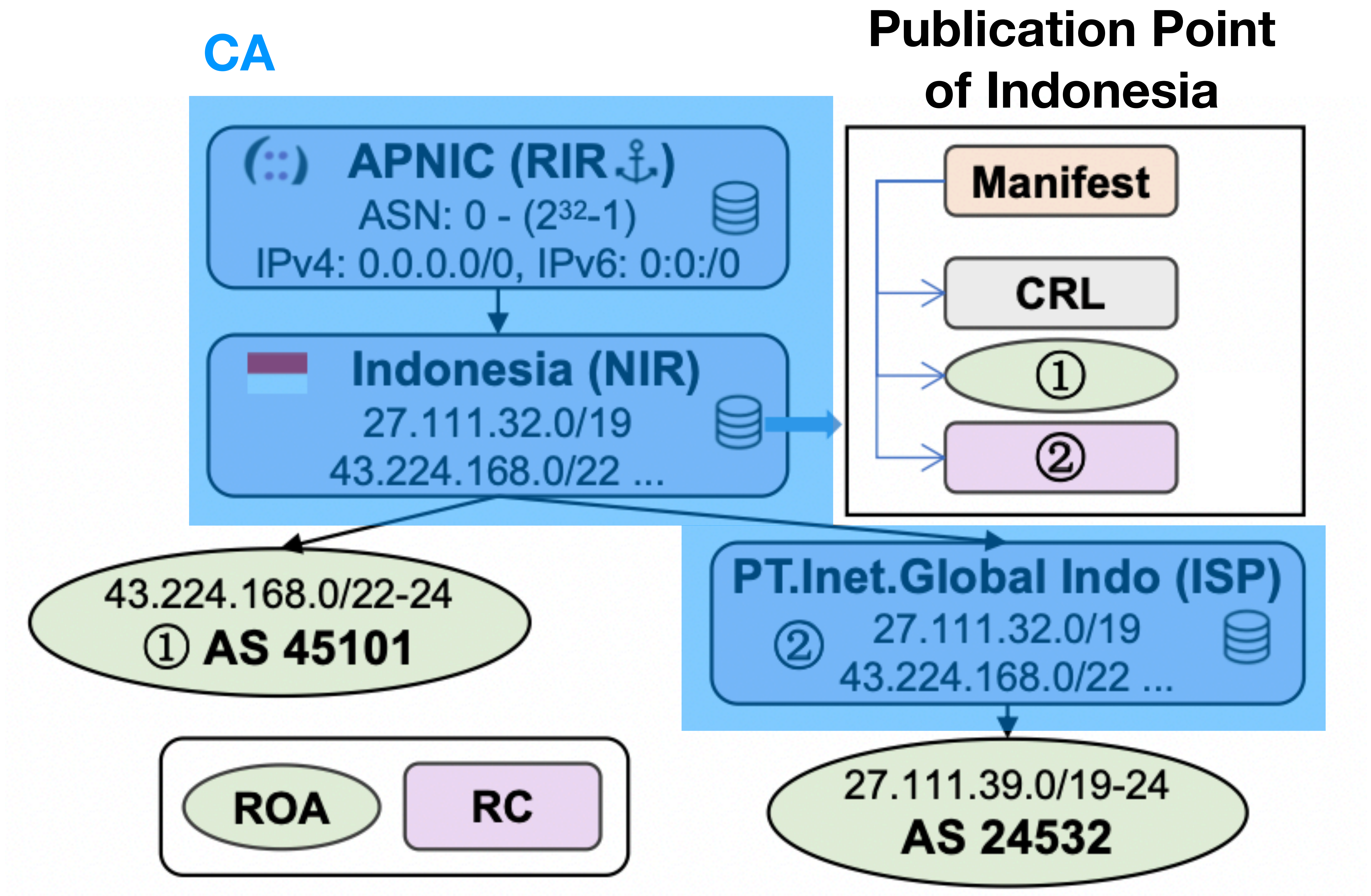
RPKI Objects



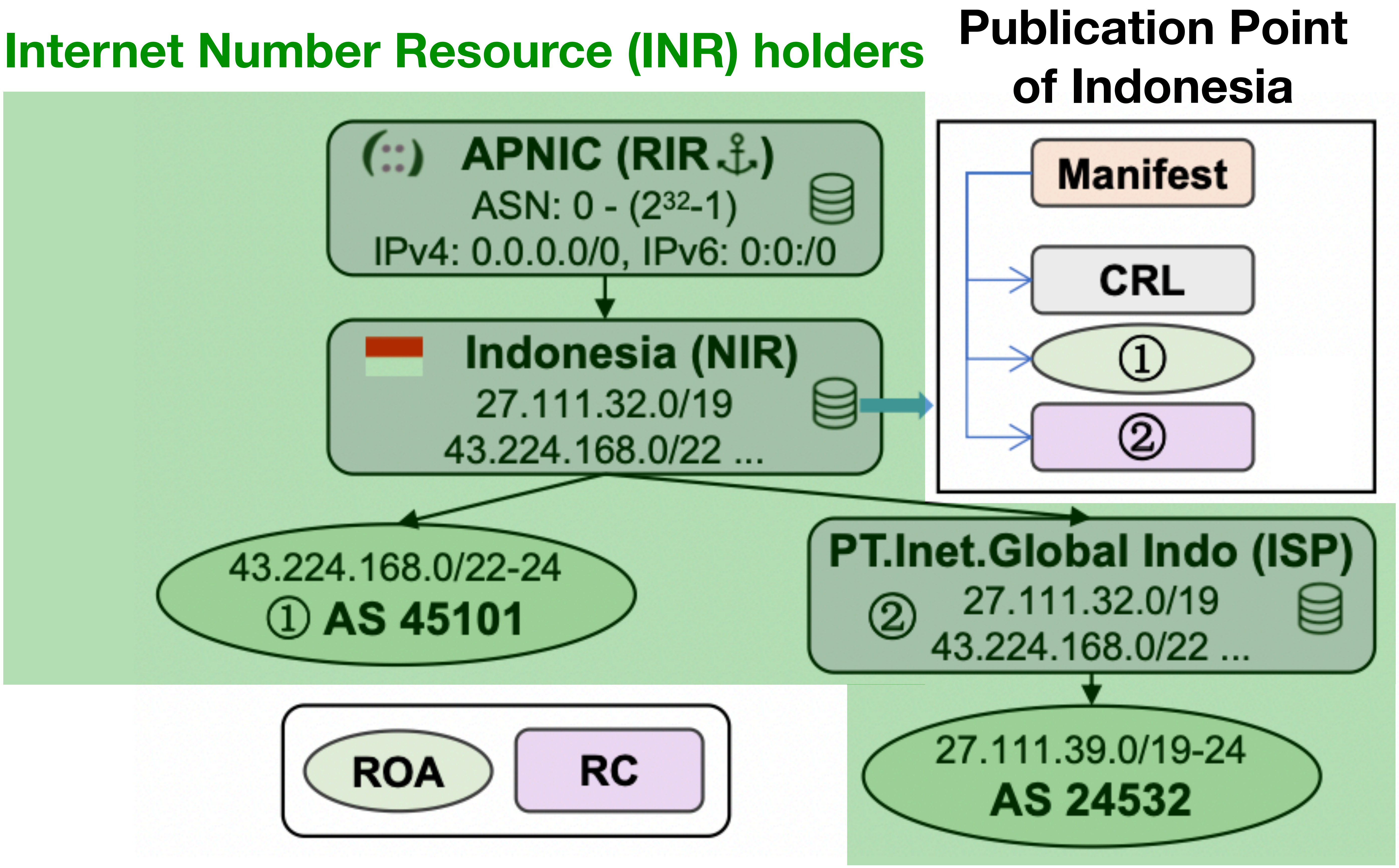
a list of all signed objects

an authorization for an AS
to originate IP prefixes

Example of RPKI hierarchical structure



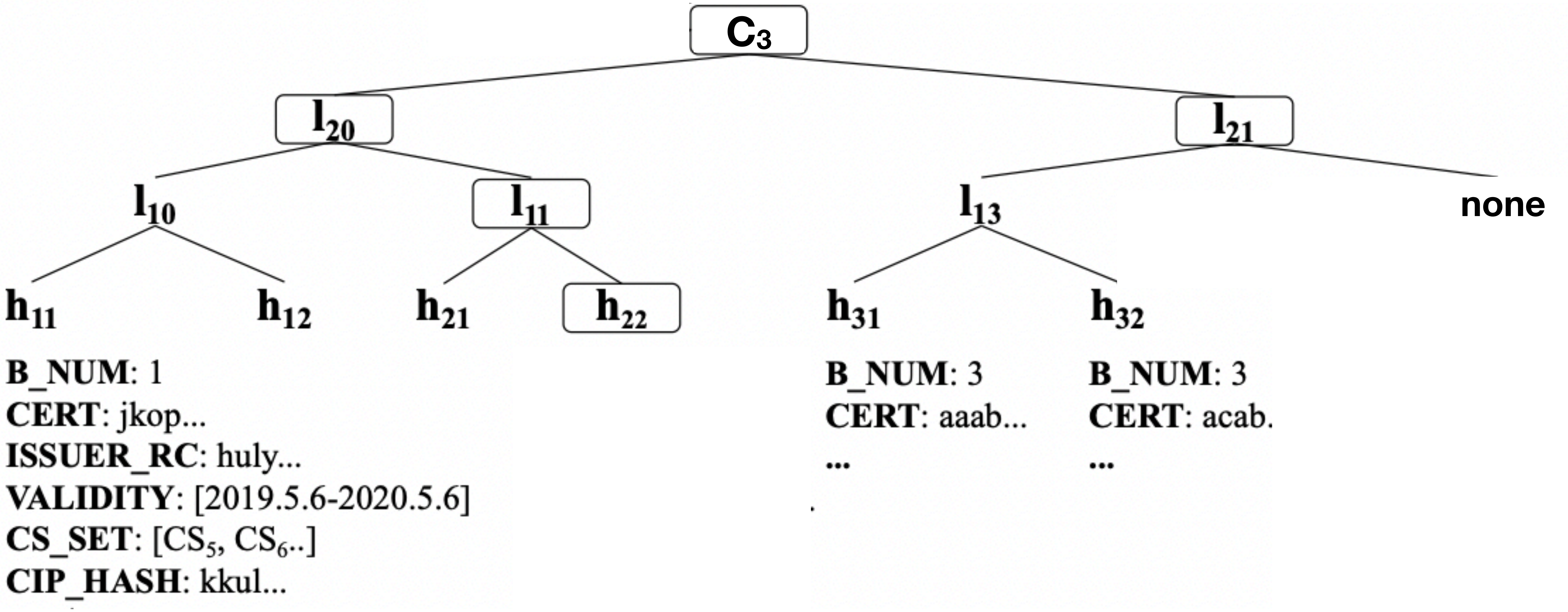
Example of RPKI hierarchical structure

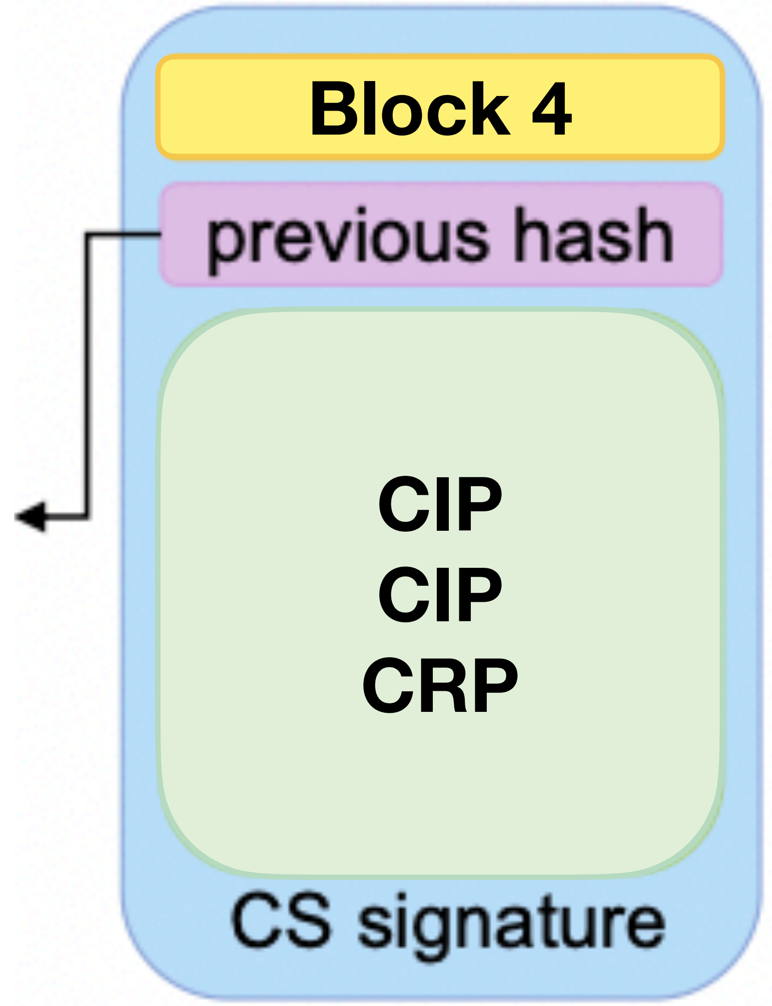


Internet Number Resource (INR) holders

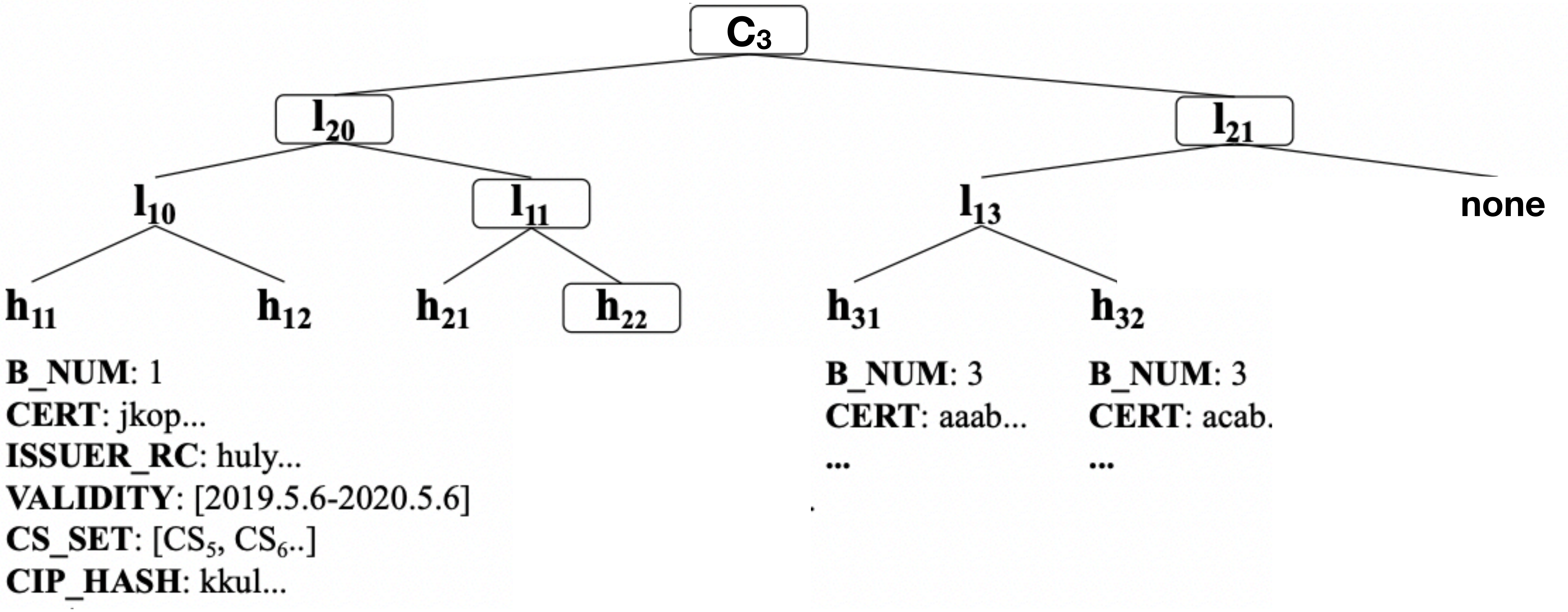
Monitor: Insert a block

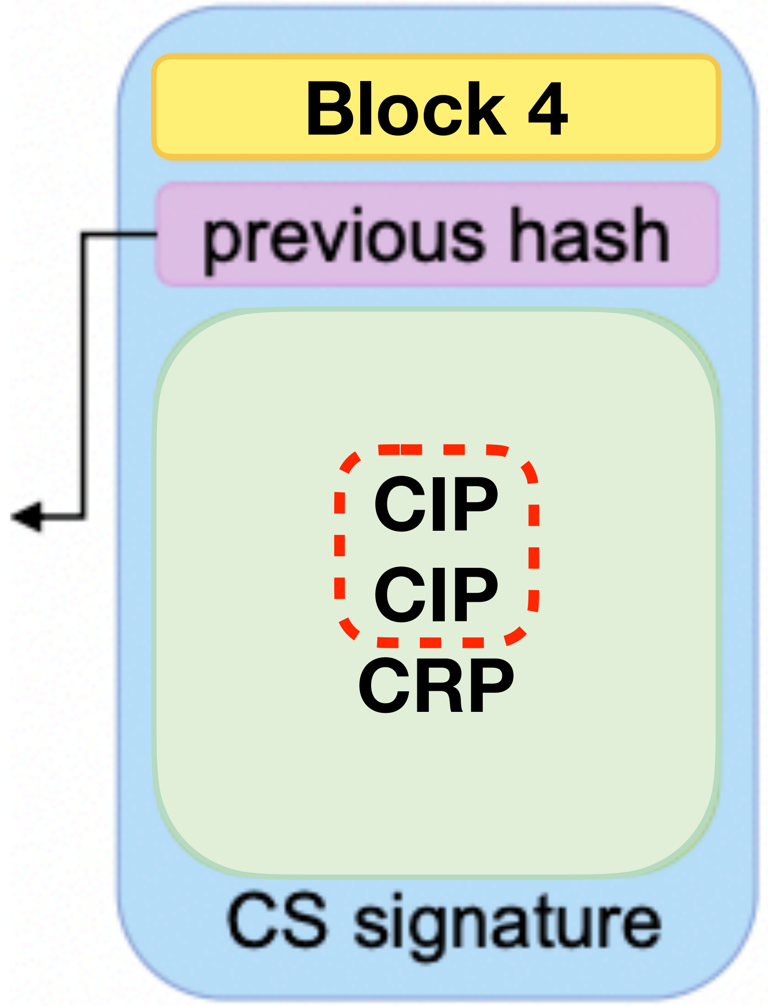
Monitor: CIPs → insert entries



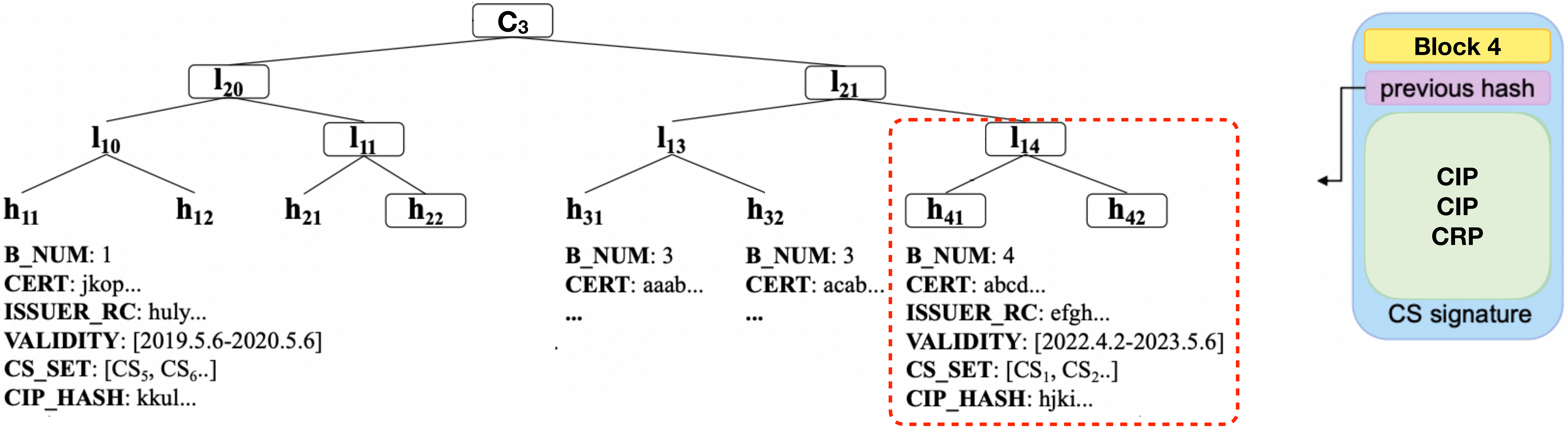


Monitor: CIPs → insert entries

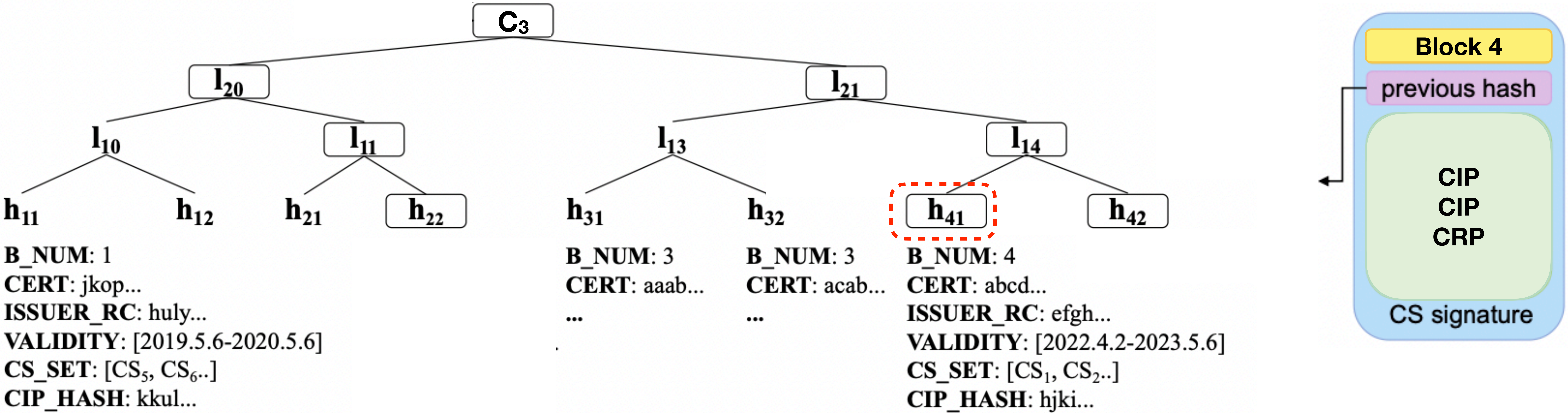




Monitor: CIPs → insert entries



Monitor: CIPs → insert entries



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS_5, CS_6..]
CIP_HASH: kkul...

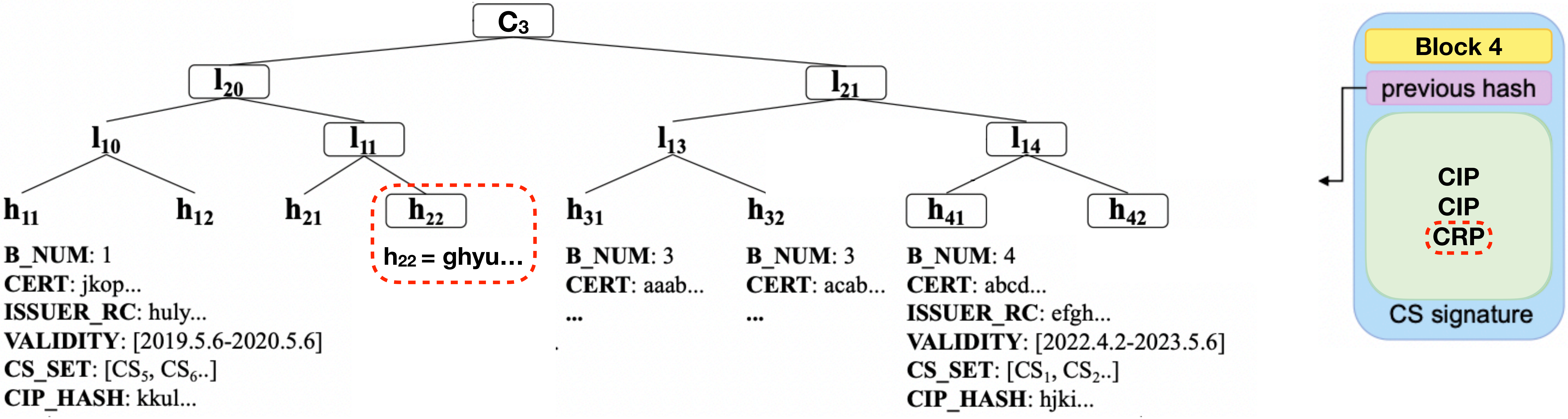
B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS_1, CS_2..]
CIP_HASH: hjki...

$$h_{41} = HASH(\texttt{B\_NUM} \mathbin{||} \texttt{ISSUER\_RC} \mathbin{||} \texttt{CERT} \mathbin{||} \texttt{VALIDITY} \mathbin{||} \texttt{CS\_SET} \mathbin{||} \texttt{CIP\_HASH})$$

Monitor: CRPs → update entries



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS$_5$, CS$_6$..]
CIP_HASH: kkul...

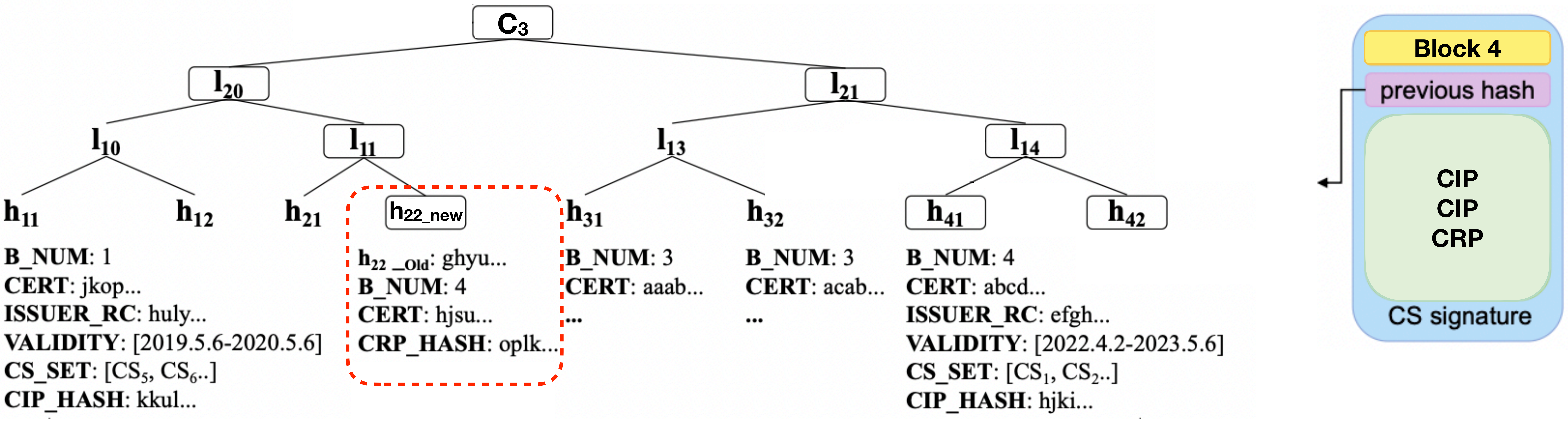
h_{22} = ghyu…

B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS$_1$, CS$_2$..]
CIP_HASH: hjki...

Monitor: CRPs → update entries



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS_5, CS_6..]
CIP_HASH: kkul...

h_{22} _Old: ghyu...
B_NUM: 4
CERT: hjsu...
CRP_HASH: oplk...

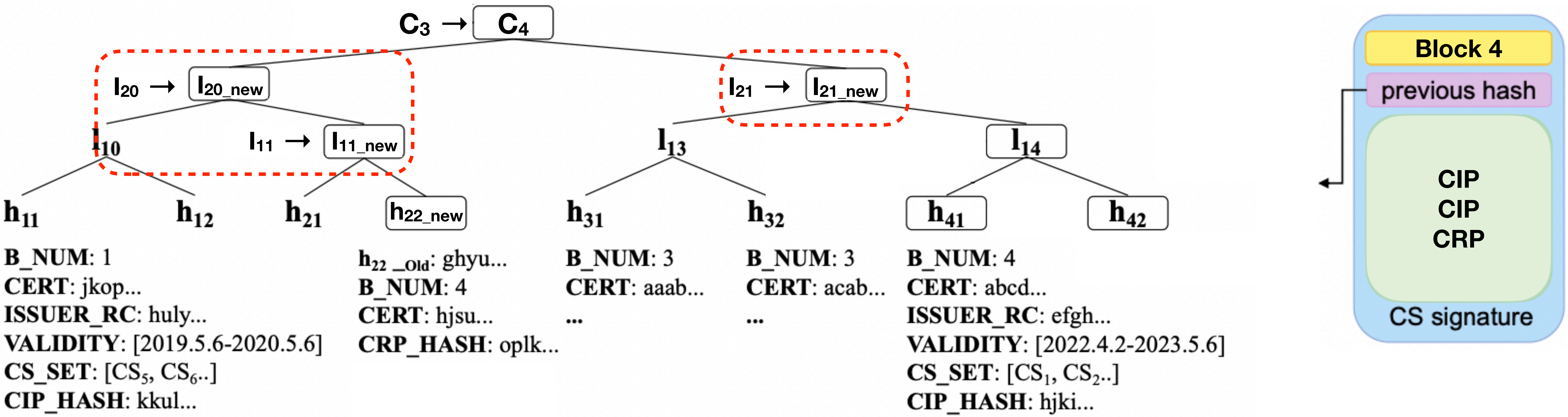
B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS_1, CS_2..]
CIP_HASH: hjki...

$$\boxed{h_{22\_new} = HASH(h_{22\_old} \mid\mid \texttt{B\_NUM} \mid\mid \texttt{CERT} \mid\mid \texttt{CRP\_HASH})}$$

Monitor: generate commitment



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS_5, CS_6..]
CIP_HASH: kkul...

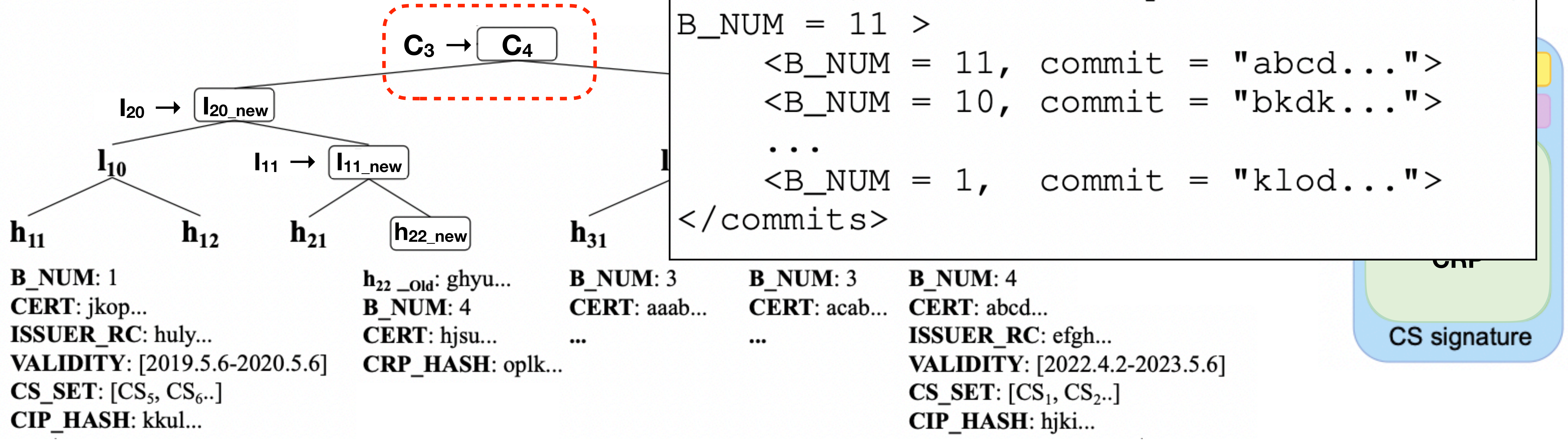
h_{22_Old}: ghyu...
B_NUM: 4
CERT: hjsu...
CRP_HASH: oplk...

B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS_1, CS_2..]
CIP_HASH: hjki...

Monitor: generate commitment



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS_5, CS_6..]
CIP_HASH: kkul...

h_{22} **_Old**: ghyu...
B_NUM: 4
CERT: hjsu...
CRP_HASH: oplk...

B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS_1, CS_2..]
CIP_HASH: hjki...

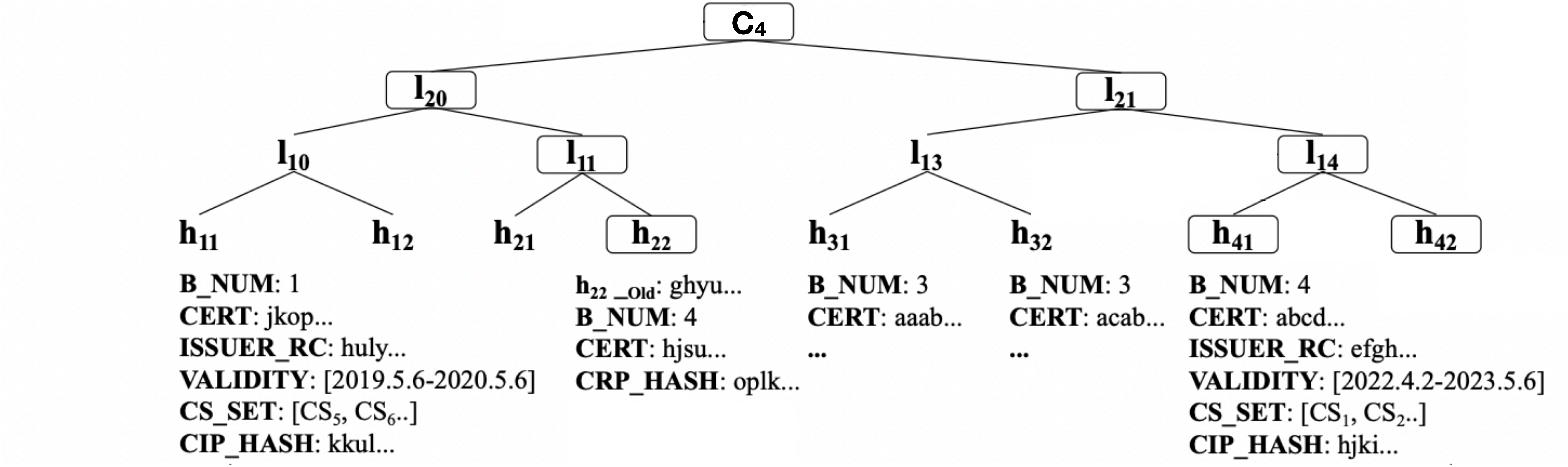
Commitment Update File

```
<commits, xmlns = "https://.../monitor",
B_NUM = 11 >
    <B_NUM = 11, commit = "abcd...">
    <B_NUM = 10, commit = "bkdk...">
    ...
    <B_NUM = 1,  commit = "klod...">
</commits>
```

Monitor: proof of absence

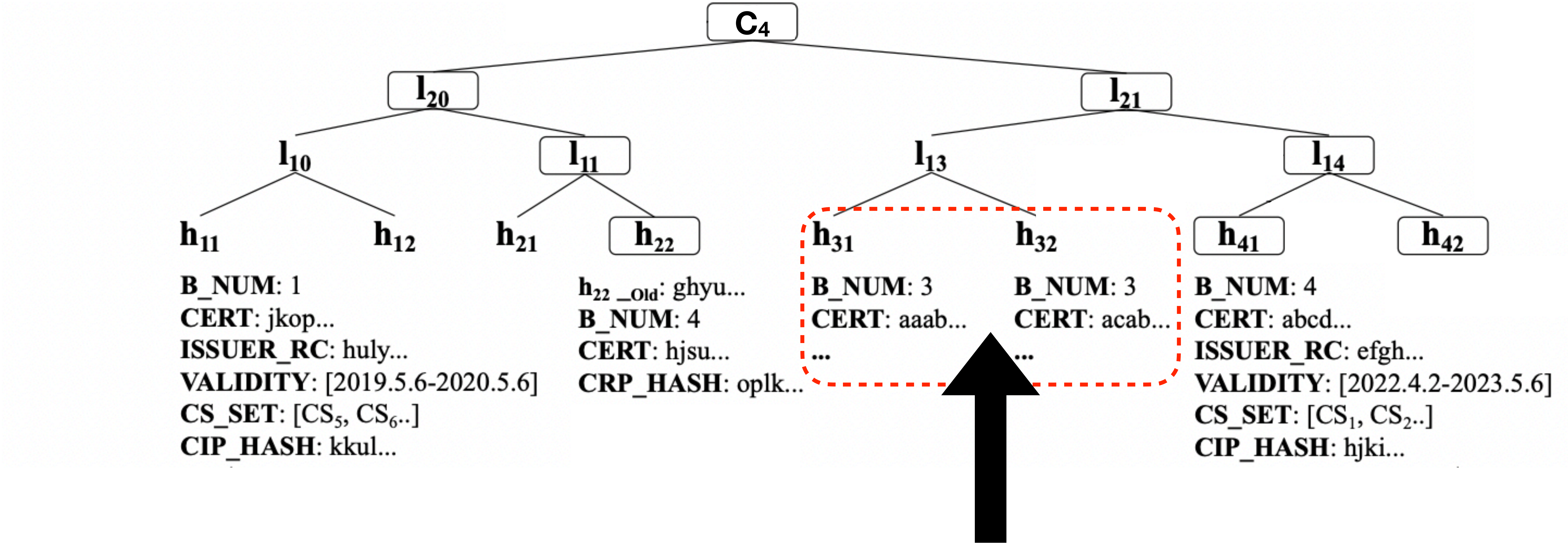
Proof of Absence: example

- INR holder asks whether the certificate in⟨B_NUM = 3,CERT = abab…⟩ exists



Proof of Absence: example

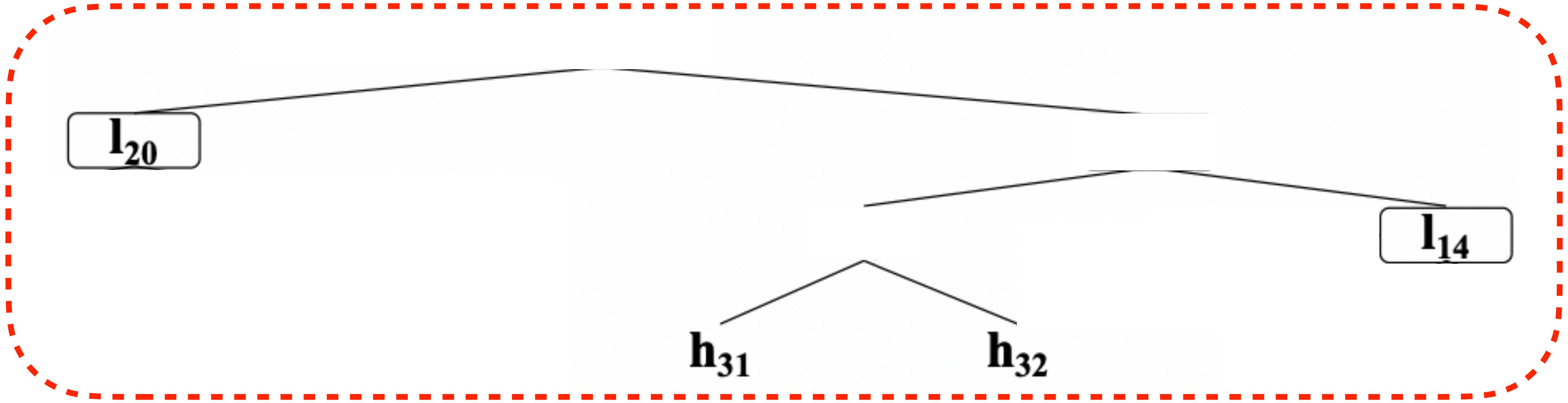
- INR holder asks whether the certificate in⟨B_NUM = 3,CERT = abab…⟩ exists



<B_NUM=3, CERT=abab> should located here
since monitors insert CIPs of one block with lexicographic order

Proof of Absence: example

- The Monitor will return a pruned tree that contains the entry of h_{32} and the hash of h_{32}, intermediate nodes l_{14} and l_{20} the INR holder

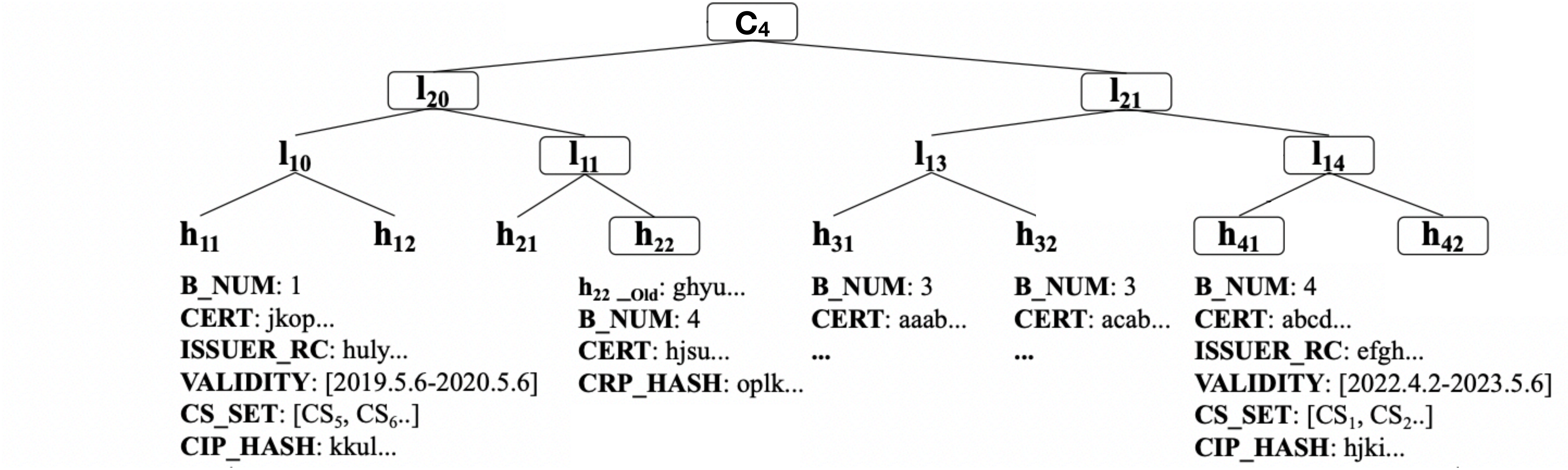


proof of absence

Monitor: proof of consistency

Proof of Consistency: example

- The RP has completed the synchronization of the first three blocks and now needs to synchronize the *block 4* incrementally
  - A Relying Party (RP) submits <B_num=3, c=C_3> to the monitor



B_NUM: 1
CERT: jkop...
ISSUER_RC: huly...
VALIDITY: [2019.5.6-2020.5.6]
CS_SET: [CS_5, CS_6..]
CIP_HASH: kkul...

h_{22} _Old: ghyu...
B_NUM: 4
CERT: hjsu...
CRP_HASH: oplk...

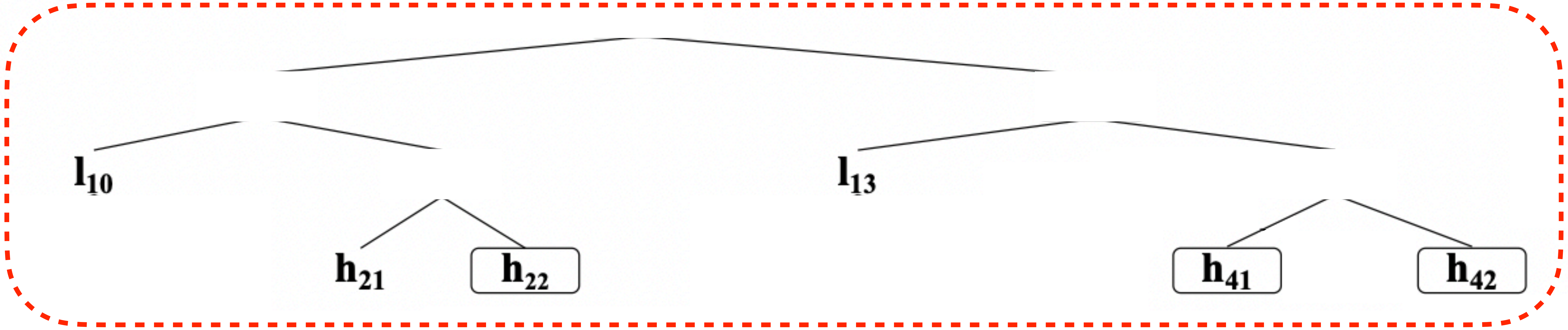
B_NUM: 3
CERT: aaab...
...

B_NUM: 3
CERT: acab...
...

B_NUM: 4
CERT: abcd...
ISSUER_RC: efgh...
VALIDITY: [2022.4.2-2023.5.6]
CS_SET: [CS_1, CS_2..]
CIP_HASH: hjki...

Proof of Consistency: example

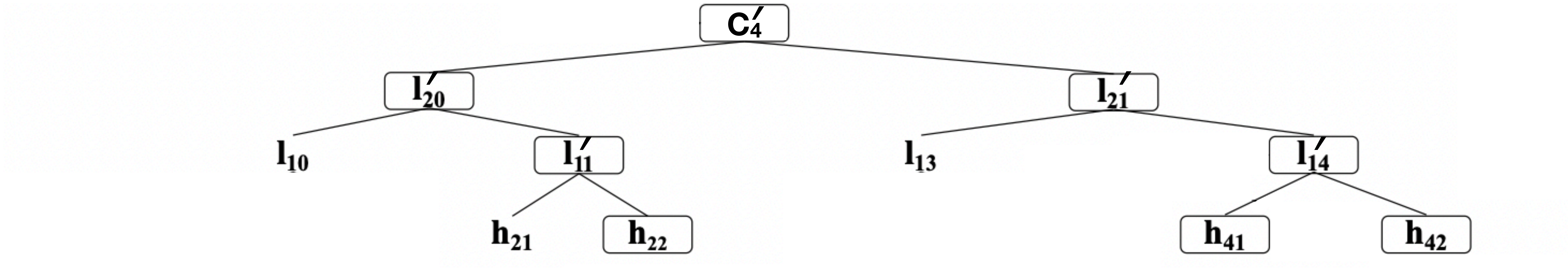
- The Monitor will return a proof of consistency



proof of consistency

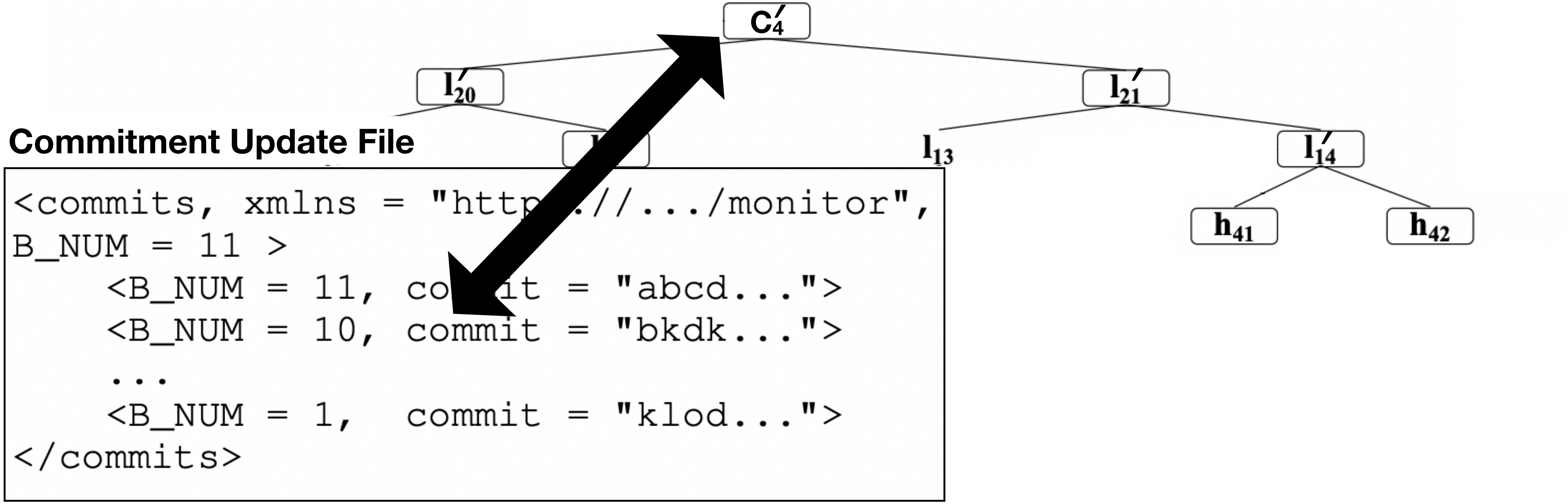
Proof of Consistency: example

- The RP reconstructs C₄′ and verify it



Proof of Consistency: example

- The RP reconstructs C_4' and verify it

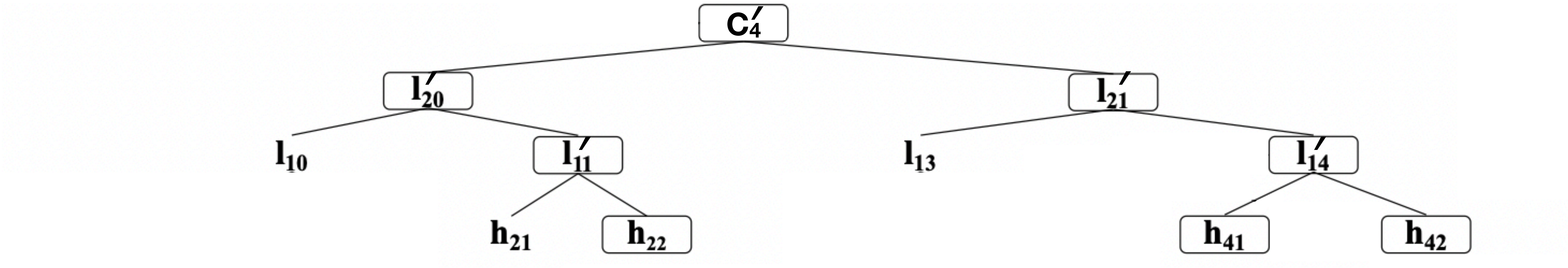


Commitment Update File

```
<commits, xmlns = "http .//.../monitor",
B_NUM = 11 >
    <B_NUM = 11, co   it = "abcd...">
    <B_NUM = 10, commit = "bkdk...">
    ...
    <B_NUM = 1,  commit = "klod...">
</commits>
```

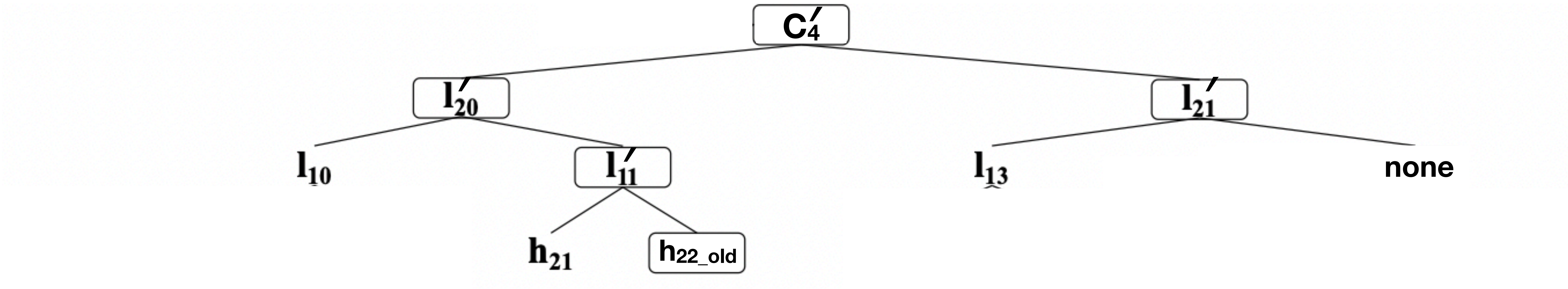
Proof of Consistency: example

- The RP Verify that C_3 can be deduced from C_4'



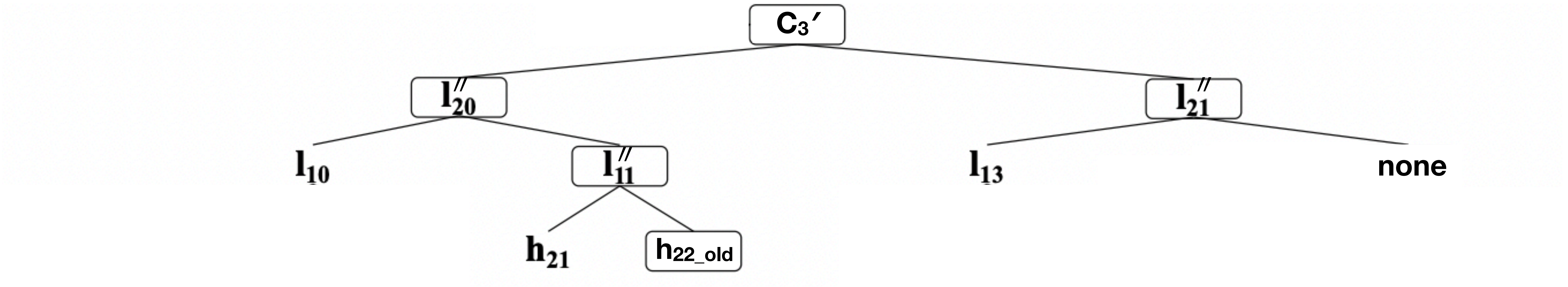
Proof of Consistency: example

- The RP Verify that C_3 can be deduced from C_4'
  - by replacing h_{22} with h_{22_old} and delete h_{41} and h_{42}



Proof of Consistency: example

- The RP Verify that *C_3* can be deduced from C_4'
    - by replacing h_{22} with h_{22_old} and delete h_{41} and h_{42}
    - reconstruct C_3' and check whether it is equal to C_3



Evaluating dRR Parameter

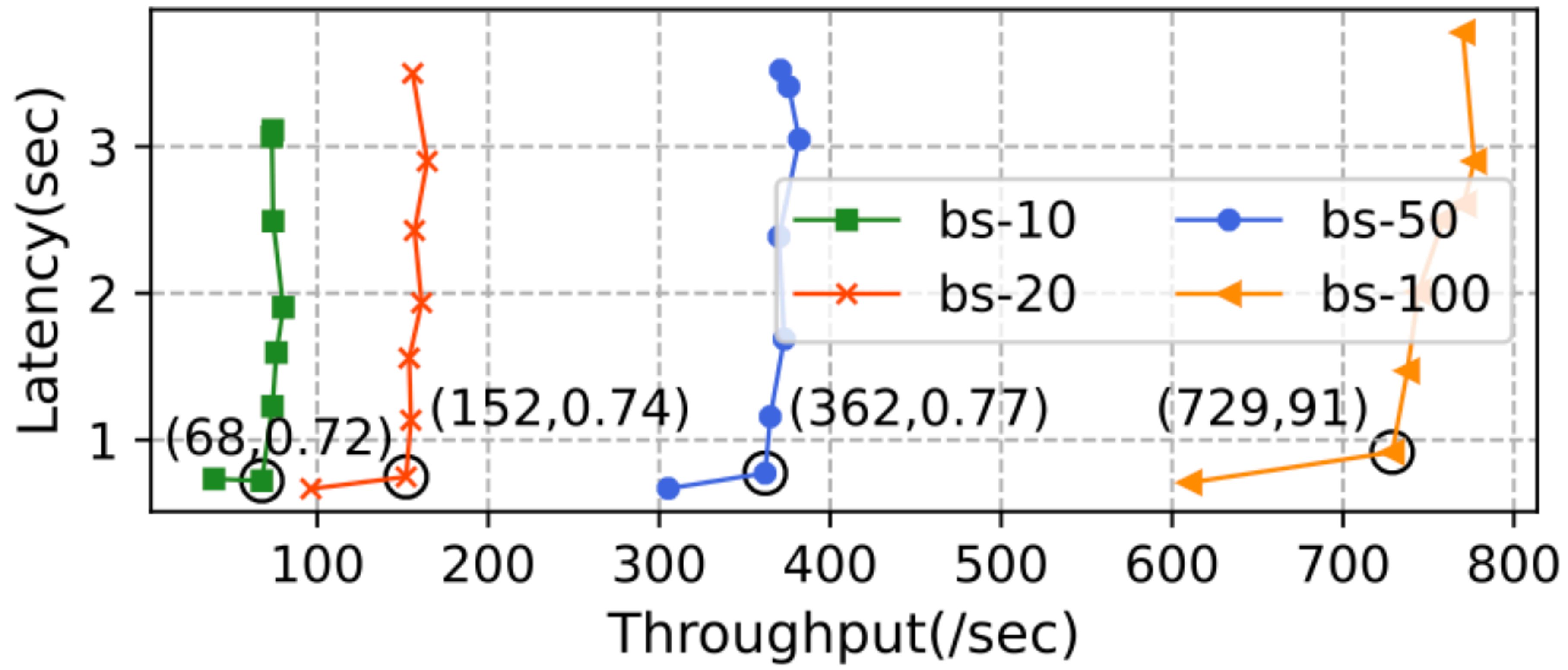


Fig. 10: The throughput and average latency under different batch sizes. Data in the circle represents the maximum throughput and the corresponding average latency.

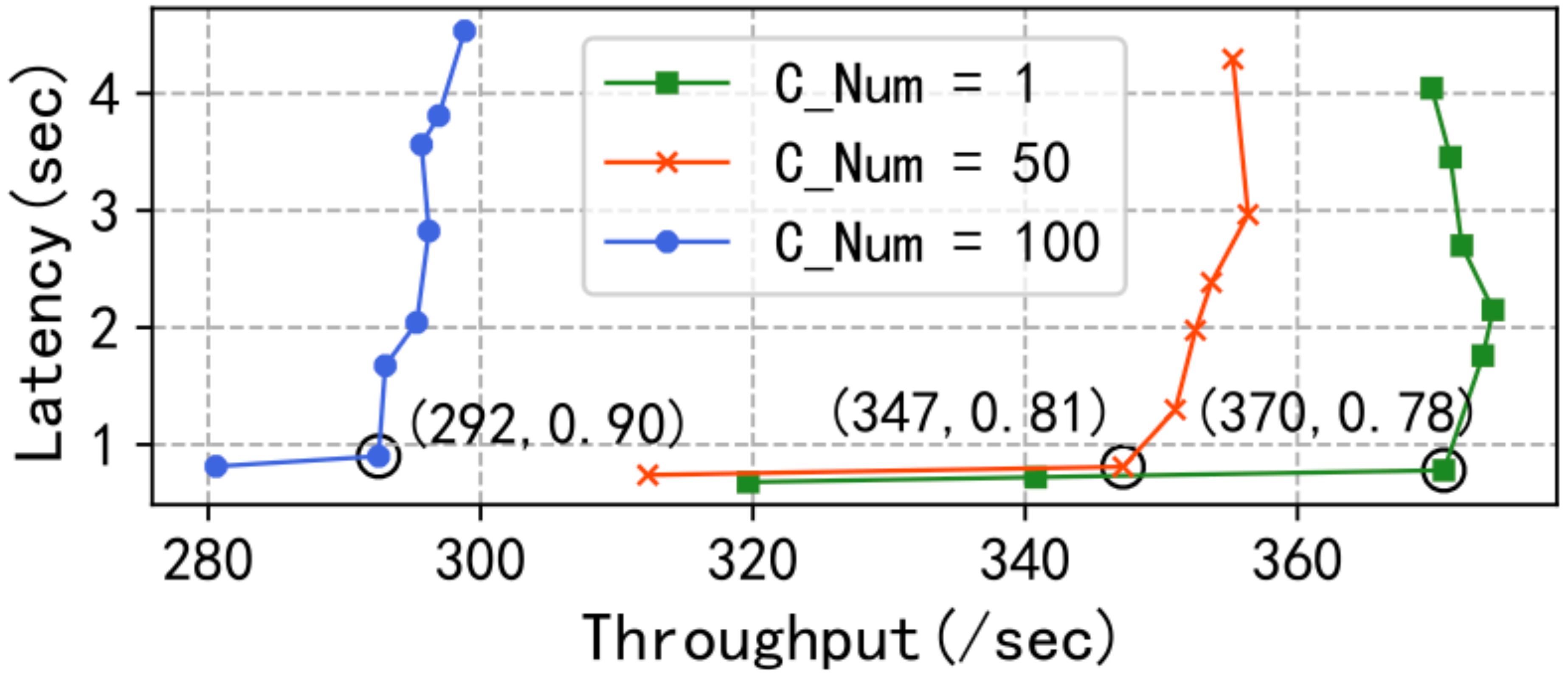
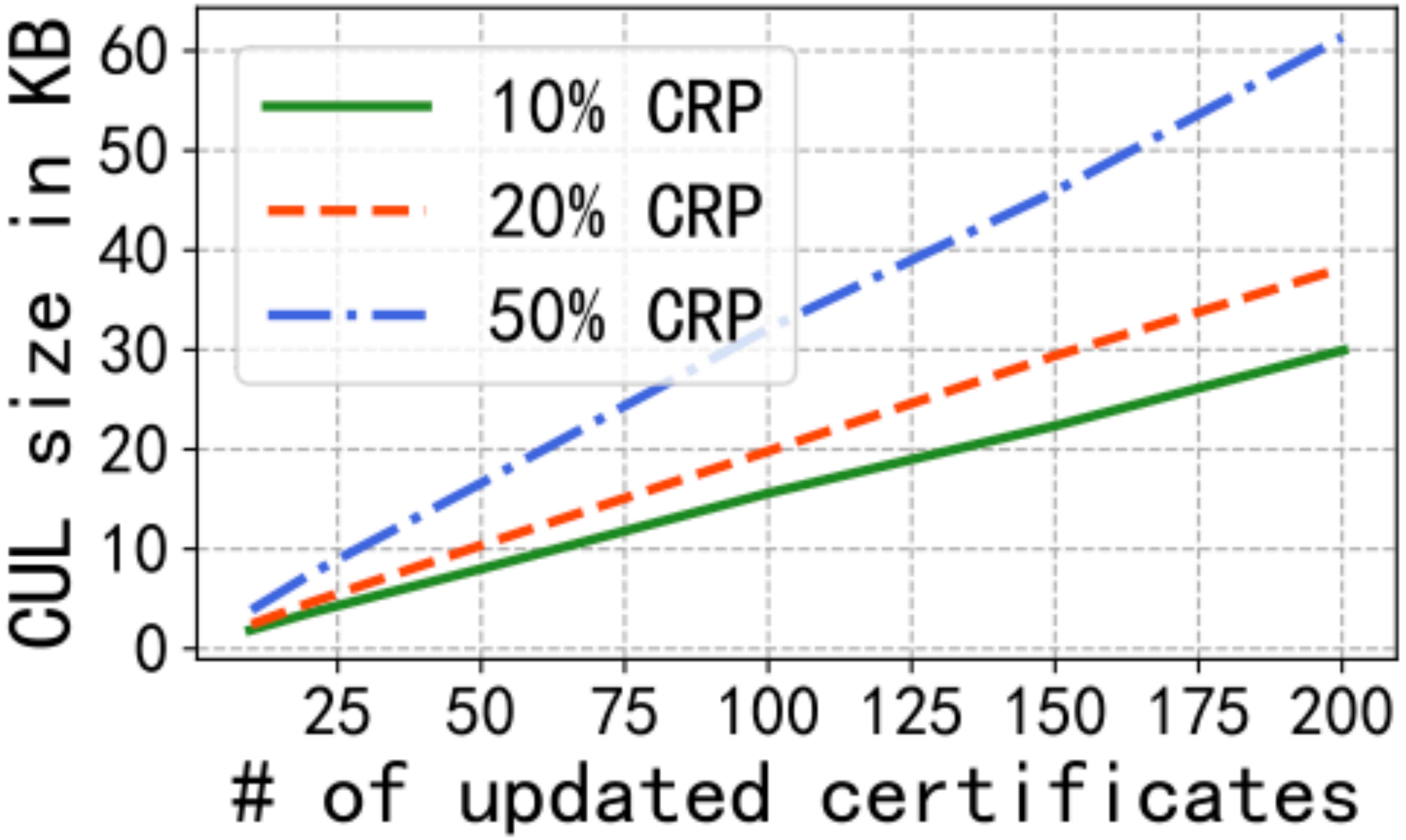


Fig. 11: The throughput and average latency under different numbers of revoked certificates in one CRP.

Evaluating Monitors

- **The size of CULs**
  - the size of the CUL is positively correlated with the number of updated certificates


CUL size in KB
# of updated certificates
10% CRP
20% CRP
50% CRP