DTKI

A new formalised PKI with no trusted parties

Vincent Cheval, Mark Ryan, Jiangshan Yu

UNIVERSITY OF BIRMINGHAM

2 April, LORIA, Nancy

Context

Most communications take place over a public network

It is important to ensure their security

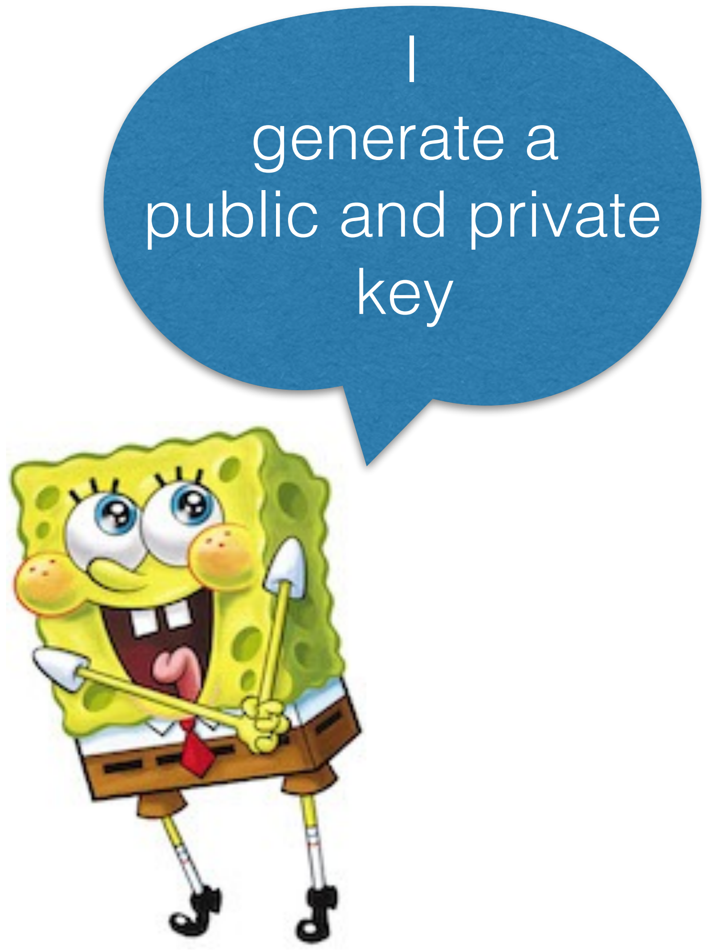
Context

Asymmetric encryption

M

Context

Asymmetric encryption



M

sk, $\mathbf{pk}(sk)$

Context

Asymmetric encryption

$\mathbf{pk}(sk)$

M

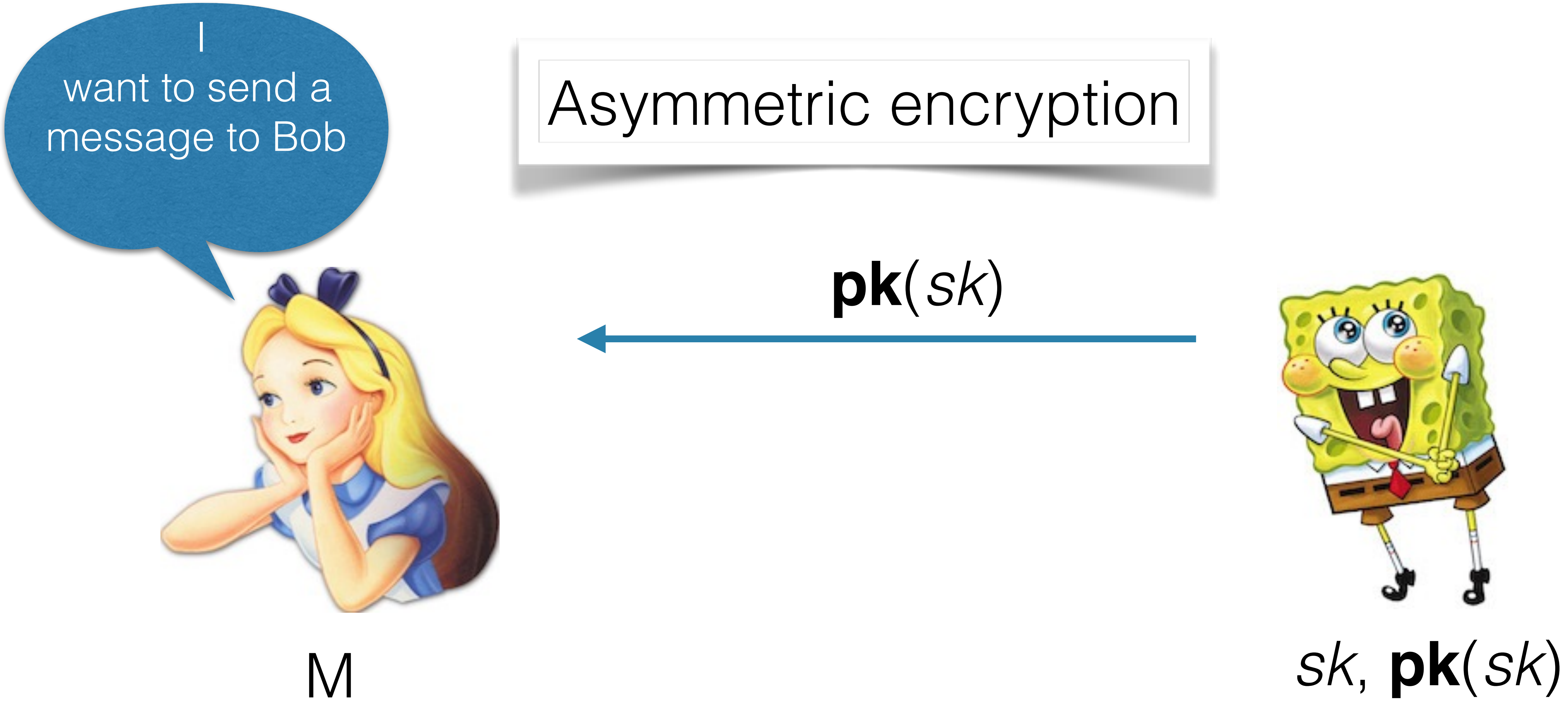
sk, $\mathbf{pk}(sk)$

I generate a public and private key

I distribute my public key

Context

Asymmetric encryption



Context

I want to send a message to Bob

Asymmetric encryption

$$\begin{tikzcd}
M & & sk, \mathbf{pk}(sk) \arrow[ll, "\mathbf{pk}(sk)"'] \\
M \arrow[rr, "\mathbf{enc}(M{,}\mathbf{pk}(sk))"] & & sk, \mathbf{pk}(sk)
\end{tikzcd}$$

M

sk, $\mathbf{pk}(sk)$

I encrypt it with the public key of Bob and send it

Context

Asymmetric encryption

M

pk(*sk*)

$\longleftarrow$

enc(M,**pk**(*sk*))

$\longrightarrow$

sk, **pk**(*sk*)

Context

Asymmetric encryption

$\mathbf{pk}(sk)$ ←

$\mathbf{enc}(M,\mathbf{pk}(sk))$ →

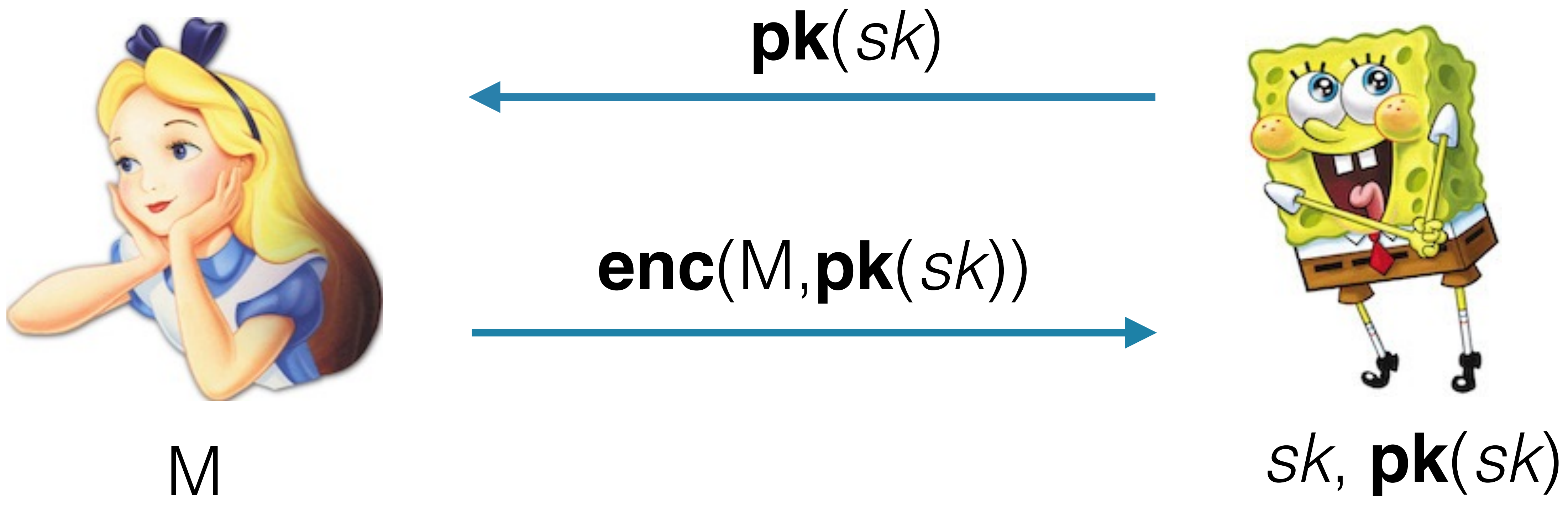
M

sk, $\mathbf{pk}(sk)$

SSL / TLS protocol

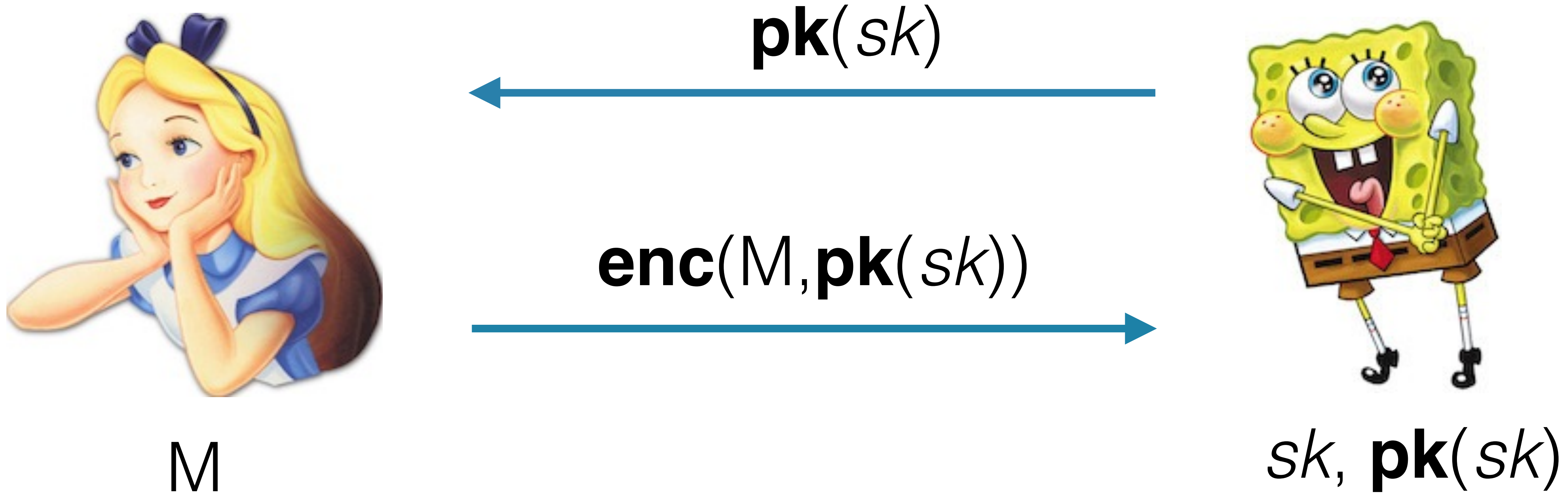
HTTPS connection

Distribution of the public key



Distribution of the public key

Authenticity of pk(sk) ?



Distribution of the public key

Authenticity of pk(sk) ?

M

sk, $\mathbf{pk}(sk)$

Distribution of the public key

Authenticity of pk(sk) ?

M

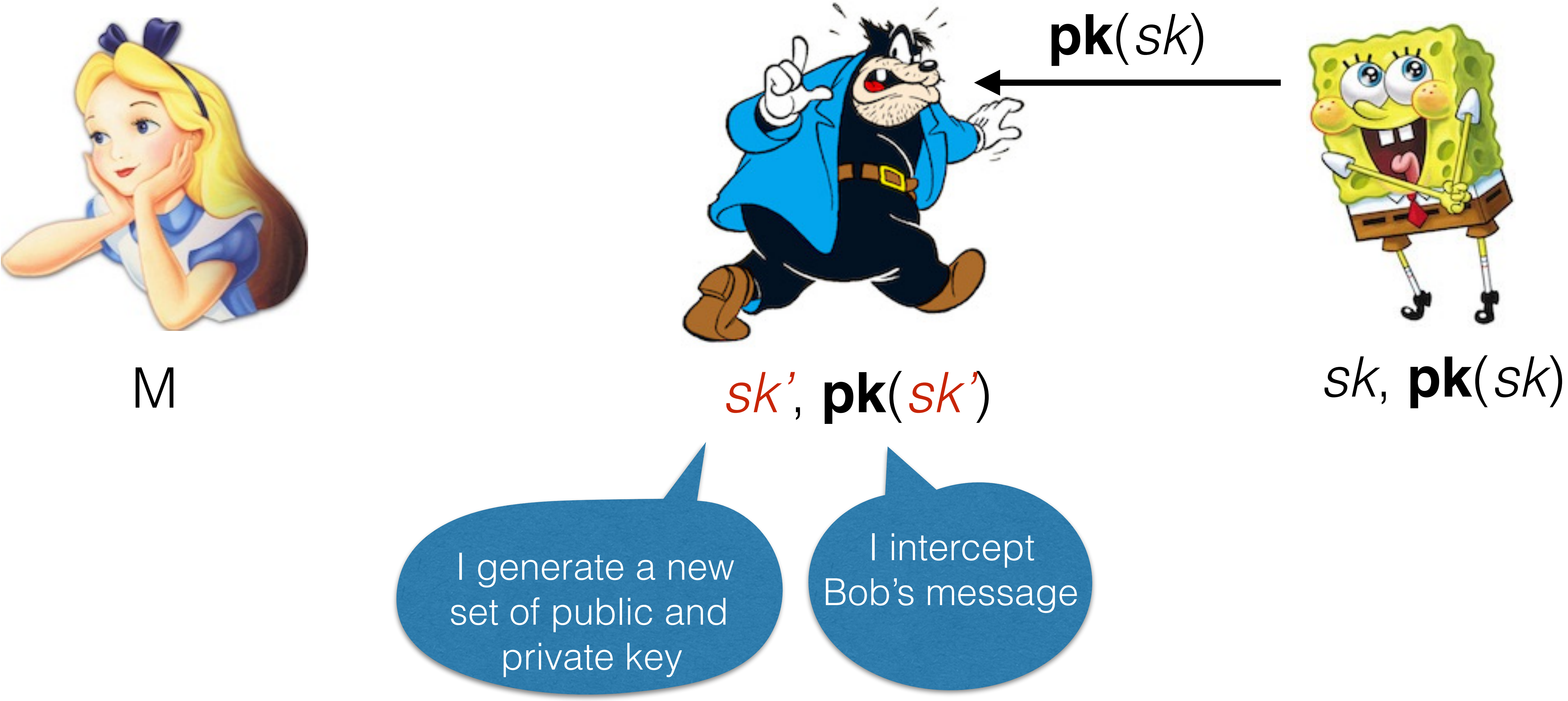
$\xleftarrow{\mathbf{pk}(sk)}$

sk, $\mathbf{pk}(sk)$

I intercept Bob's message

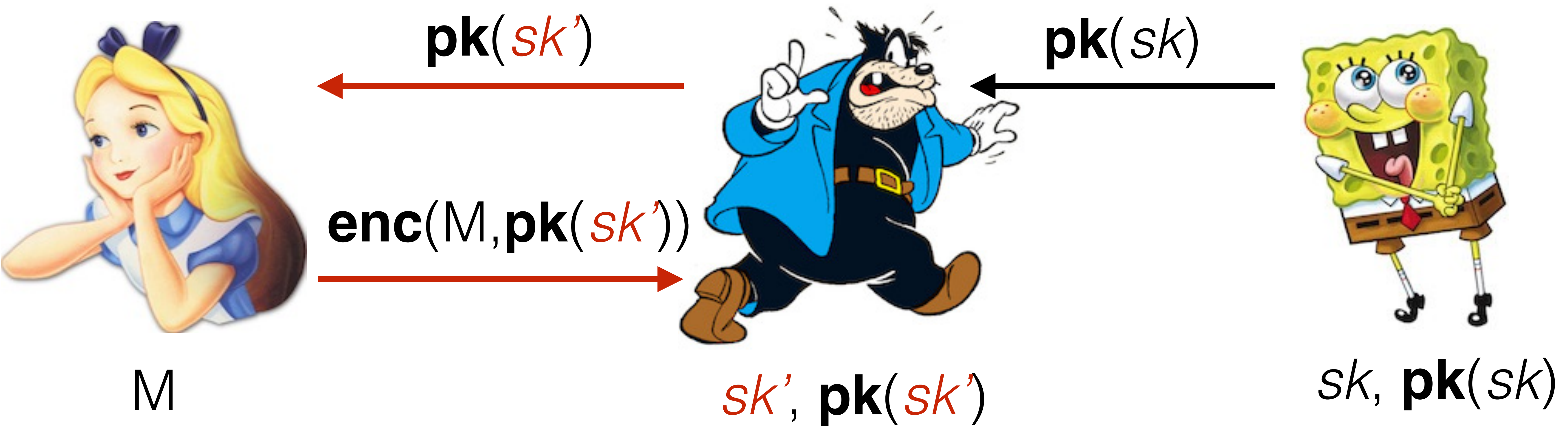
Distribution of the public key

Authenticity of pk(sk) ?


pk(*sk*)
M
sk', **pk**(*sk'*)
sk, **pk**(*sk*)
I generate a new set of public and private key
I intercept Bob's message

Distribution of the public key

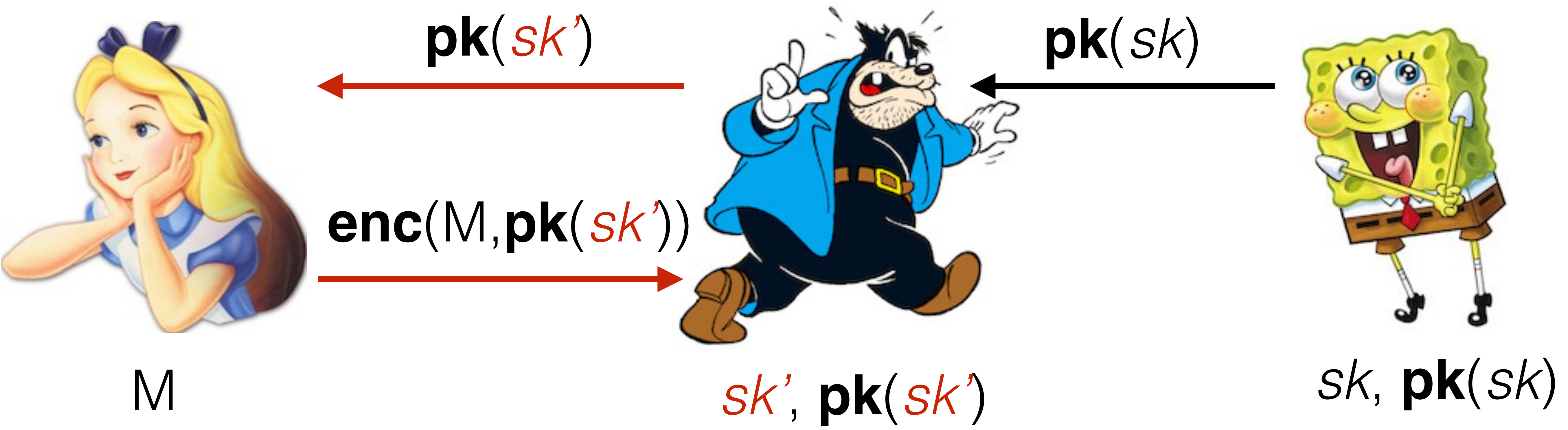
Authenticity of pk(sk) ?



I send the fake public key to Alice

Distribution of the public key

Authenticity of pk(sk) ?



We need a reliable Public Key Infrastructure (PKI)

Existing solution

Public key certificate: digital identity (standard X.509)

Certificate authority: VeriSign, Comodo, Go Daddy…

Existing solution

Public key certificate: digital identity (standard X.509)

Certificate authority: VeriSign, Comodo, Go Daddy…

sk_{CA}, $\mathbf{pk}(sk_{CA})$

sk, $\mathbf{pk}(sk)$

Existing solution

Public key certificate: digital identity (standard X.509)

Certificate authority: VeriSign, Comodo, Go Daddy…

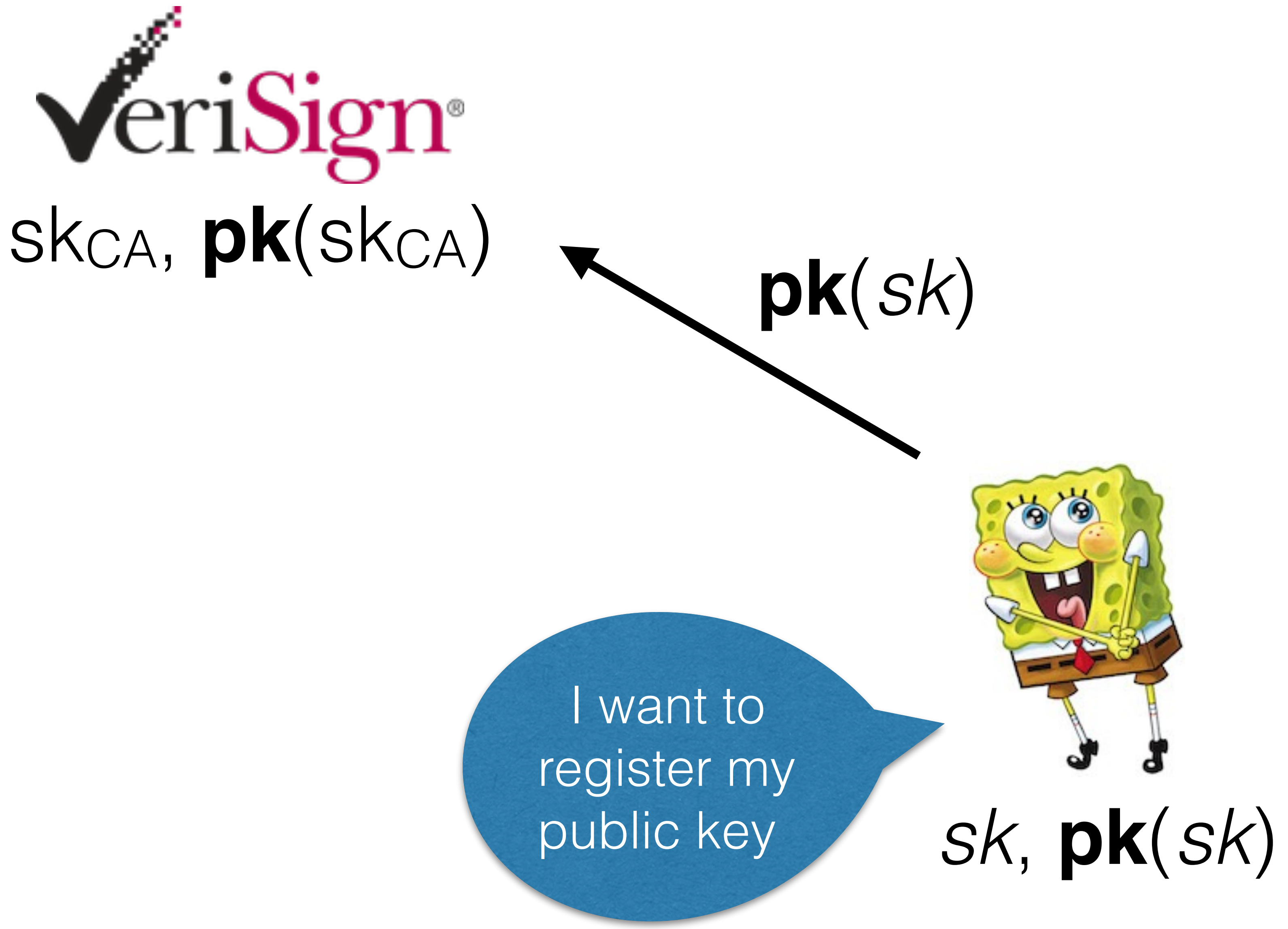
sk_{CA}, $\mathbf{pk}(sk_{CA})$

sk, $\mathbf{pk}(sk)$

Existing solution

Public key certificate: digital identity (standard X.509)

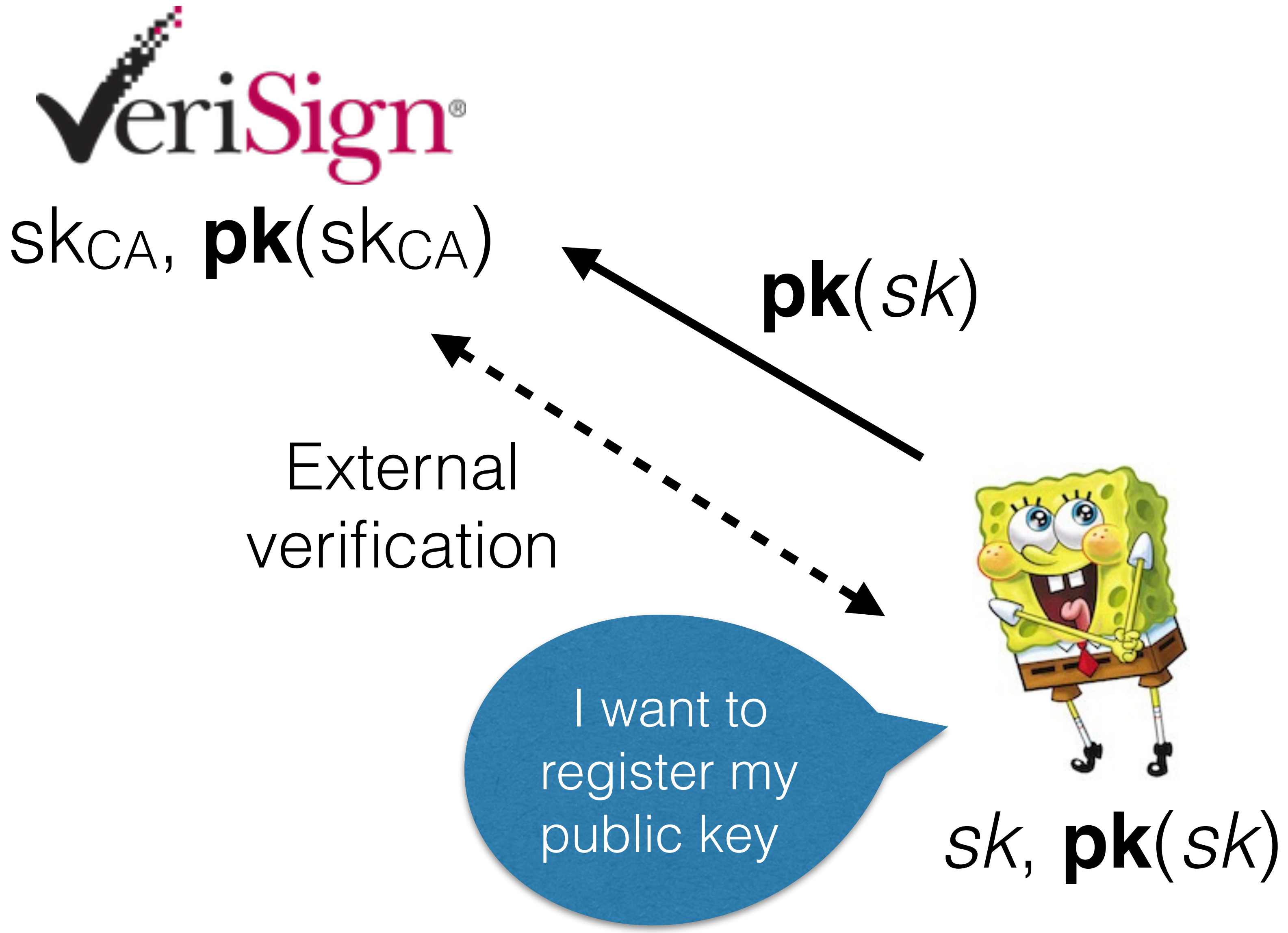
Certificate authority: VeriSign, Comodo, Go Daddy…


VeriSign
sk_{CA}, $\mathbf{pk}(sk_{CA})$
$\mathbf{pk}(sk)$
I want to register my public key
sk, $\mathbf{pk}(sk)$

Existing solution

Public key certificate: digital identity (standard X.509)

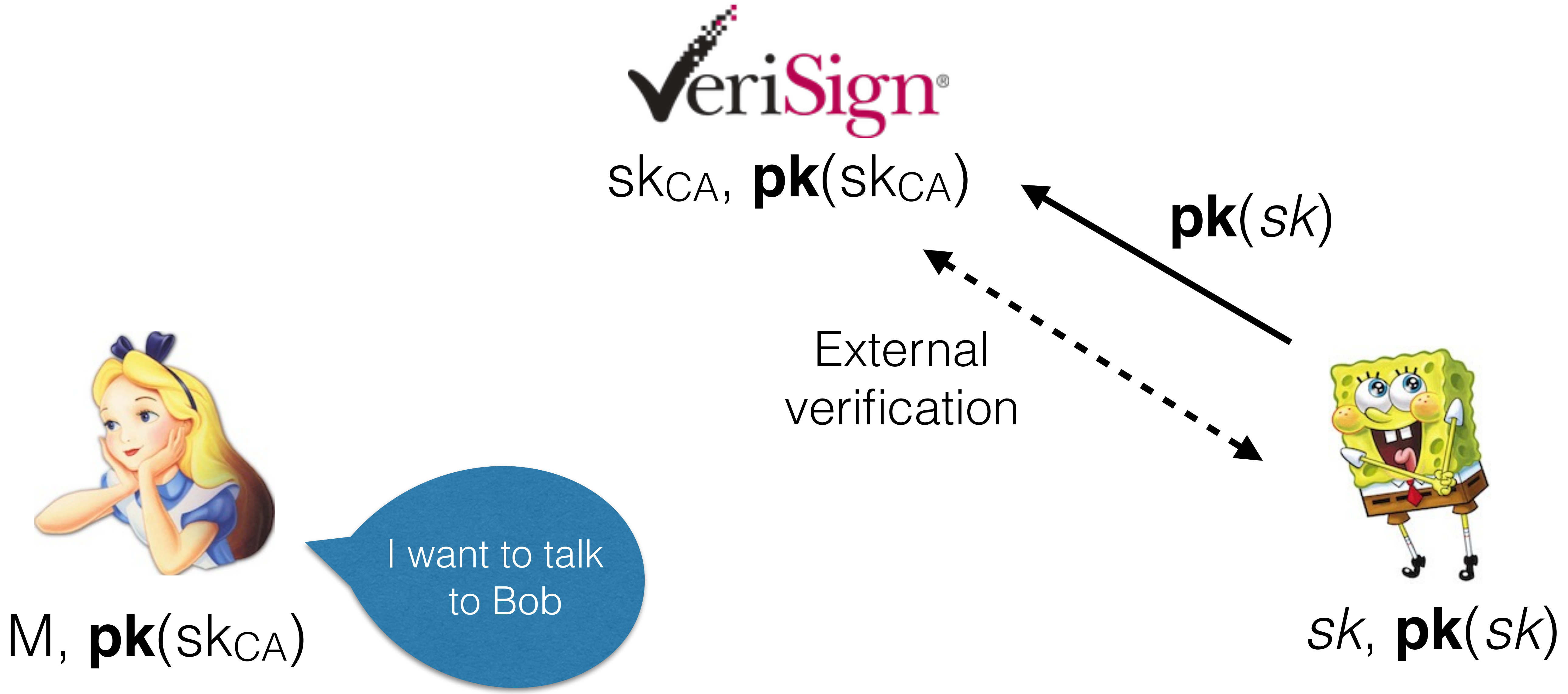
Certificate authority: VeriSign, Comodo, Go Daddy…



Existing solution

Public key certificate: digital identity (standard X.509)

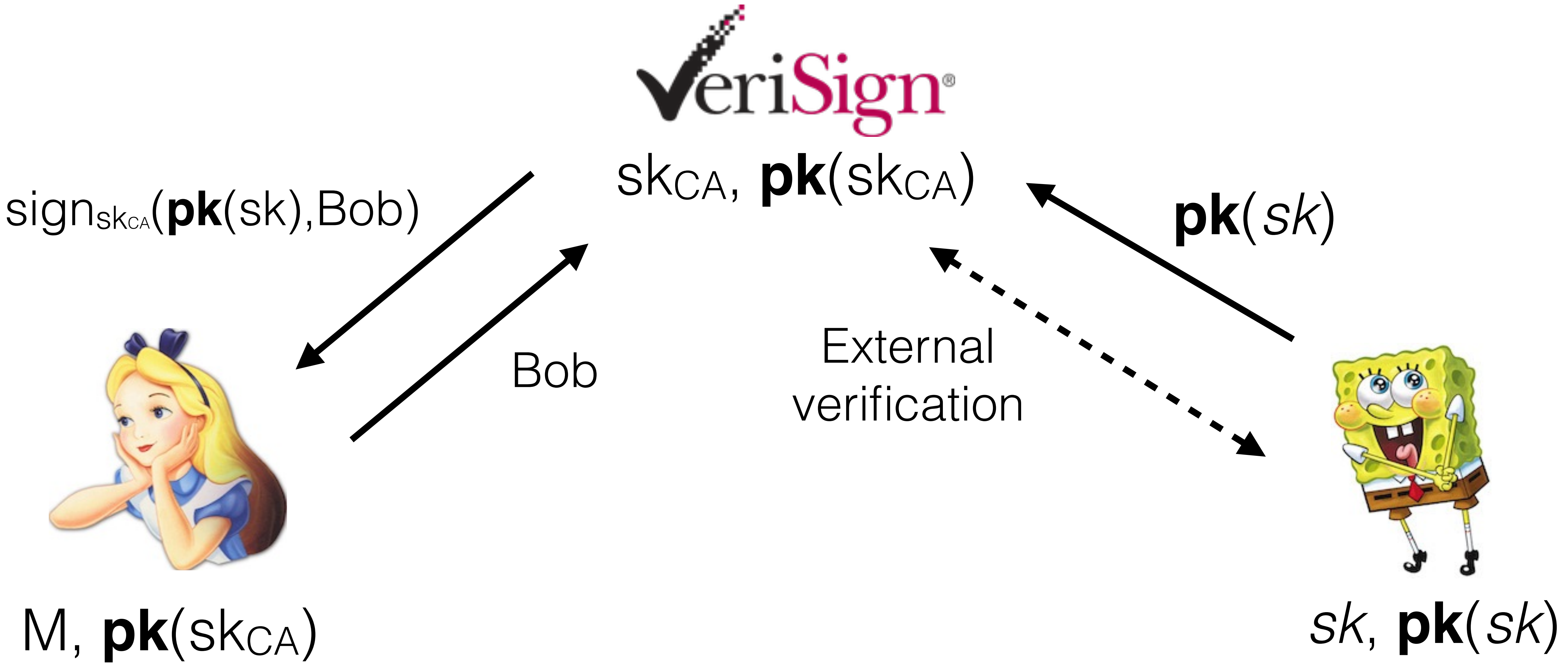
Certificate authority: VeriSign, Comodo, Go Daddy…



Existing solution

Public key certificate: digital identity (standard X.509)

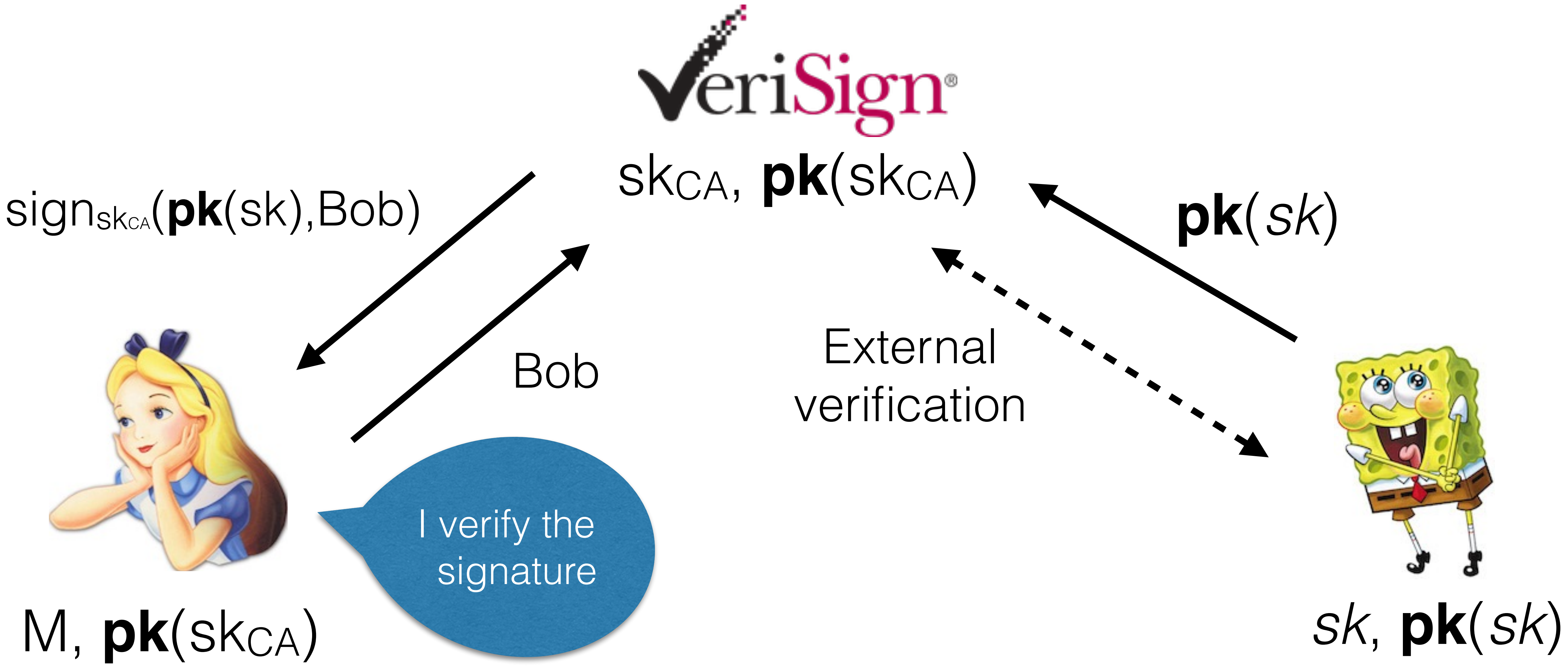
Certificate authority: VeriSign, Comodo, Go Daddy…


sk_{CA}, $\mathbf{pk}(\mathrm{sk}_{CA})$
$\mathrm{sign}_{\mathrm{sk}_{CA}}(\mathbf{pk}(\mathrm{sk}),\mathrm{Bob})$
$\mathbf{pk}(sk)$
Bob
External verification
M, $\mathbf{pk}(\mathrm{sk}_{CA})$
sk, $\mathbf{pk}(sk)$

Existing solution

Public key certificate: digital identity (standard X.509)

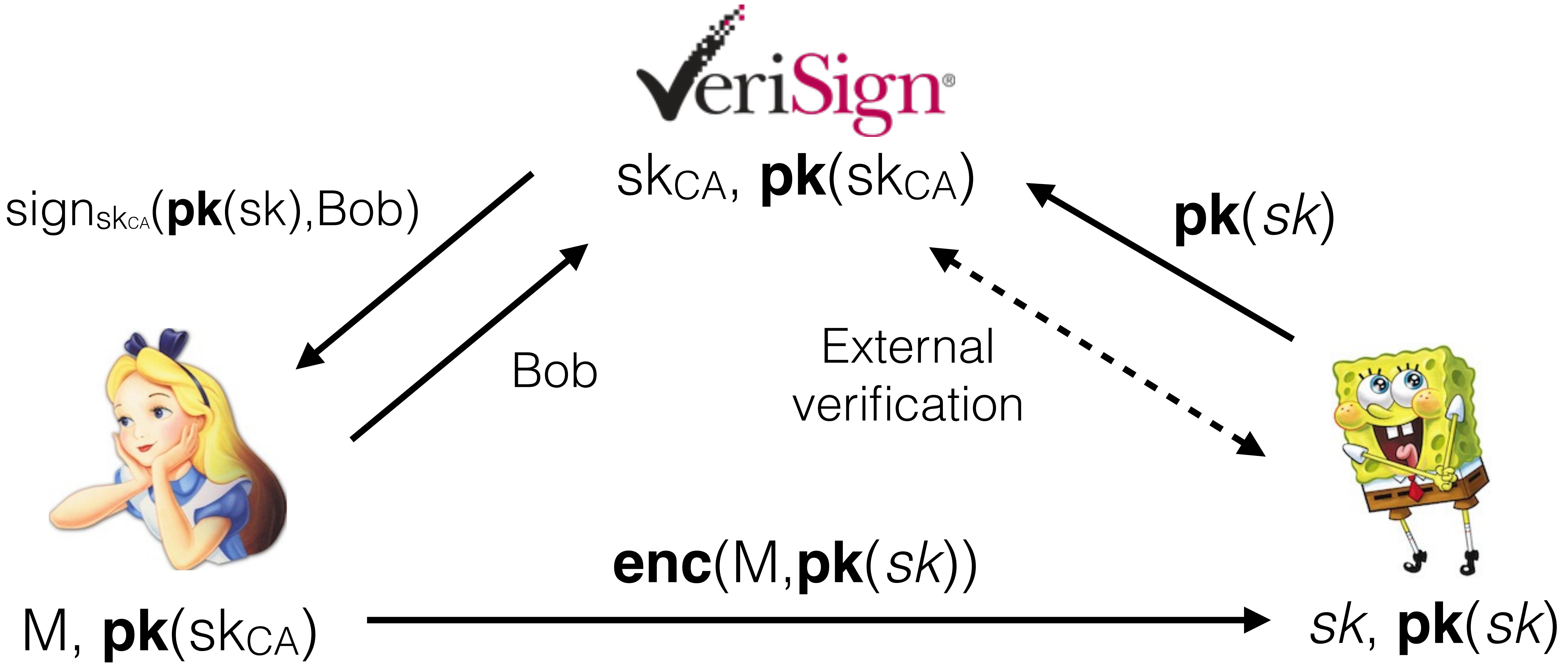
Certificate authority: VeriSign, Comodo, Go Daddy…


sk_{CA}, $\mathbf{pk}(sk_{CA})$
$\text{sign}_{sk_{CA}}(\mathbf{pk}(sk),\text{Bob})$
Bob
External verification
$\mathbf{pk}(sk)$
I verify the signature
M, $\mathbf{pk}(sk_{CA})$
sk, $\mathbf{pk}(sk)$

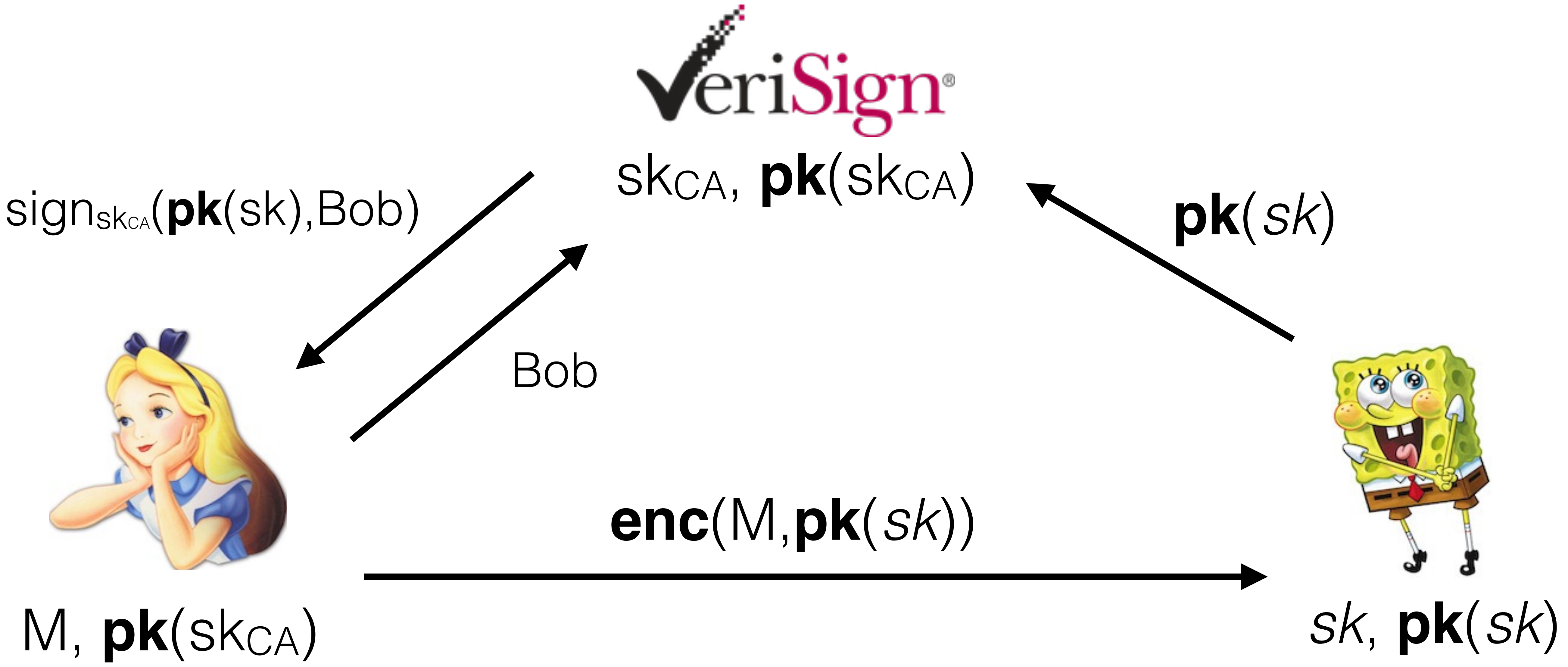
Existing solution

Public key certificate: digital identity (standard X.509)

Certificate authority: VeriSign, Comodo, Go Daddy…



Problems with existing solution



Problems with existing solution

sk_{CA}, $\mathbf{pk}(\text{sk}_{CA})$

control

M, $\mathbf{pk}(\text{sk}_{CA})$

sk, $\mathbf{pk}(sk)$

Problems with existing solution

VeriSign®

sk_{CA}, $\mathbf{pk}(sk_{CA})$

control

M, $\mathbf{pk}(sk_{CA})$

sk, $\mathbf{pk}(sk)$

Problems with existing solution

sk_{CA}, $\mathbf{pk}(sk_{CA})$

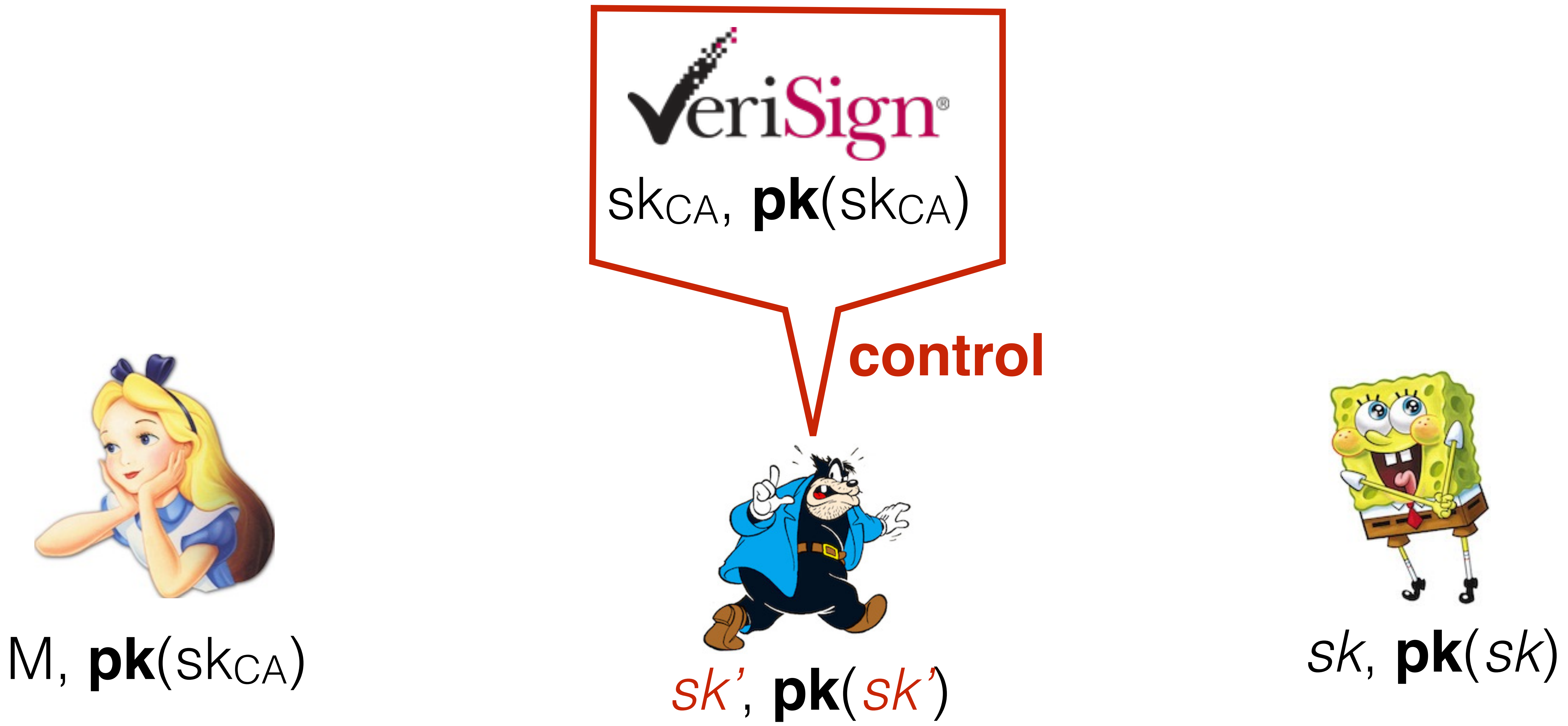
control

M, $\mathbf{pk}(sk_{CA})$

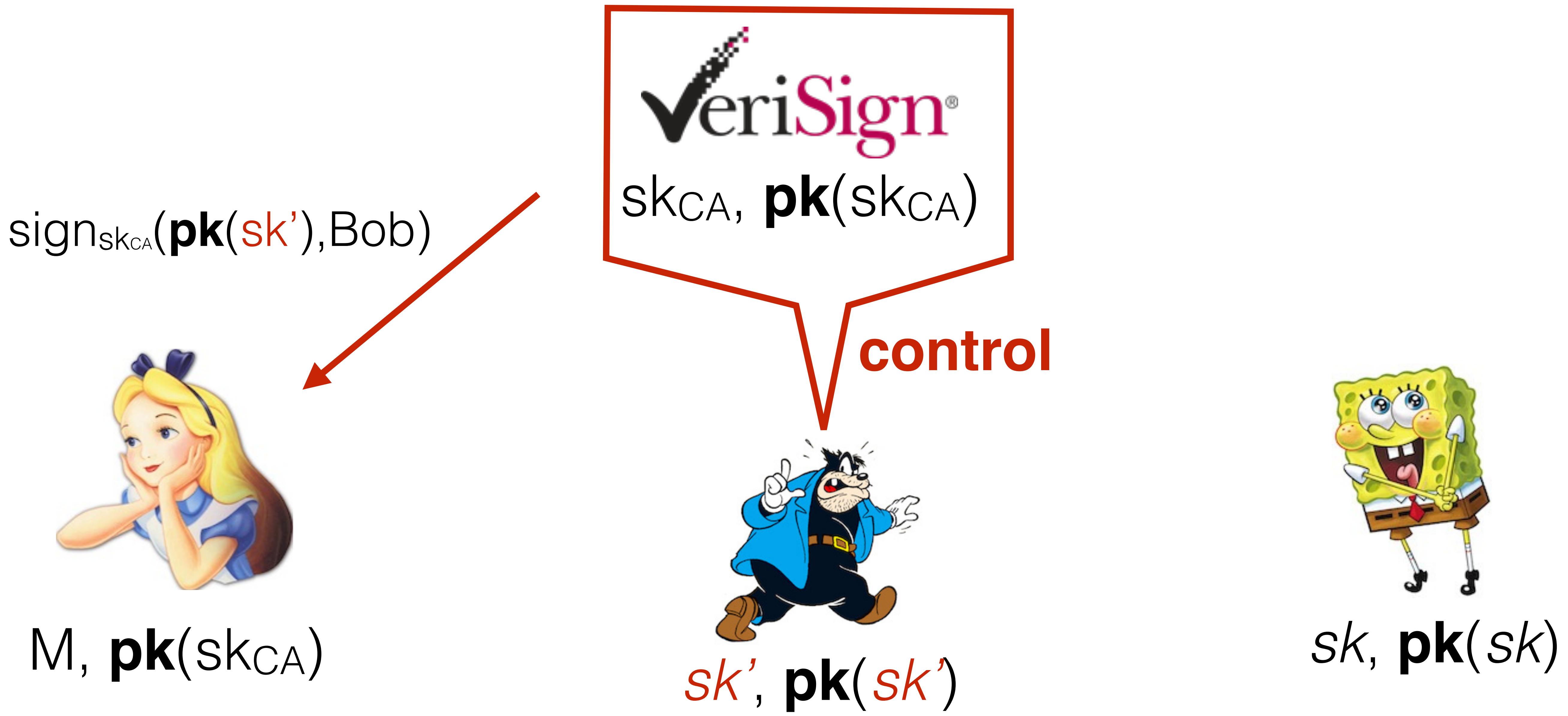
sk', $\mathbf{pk}(sk')$

sk, $\mathbf{pk}(sk)$

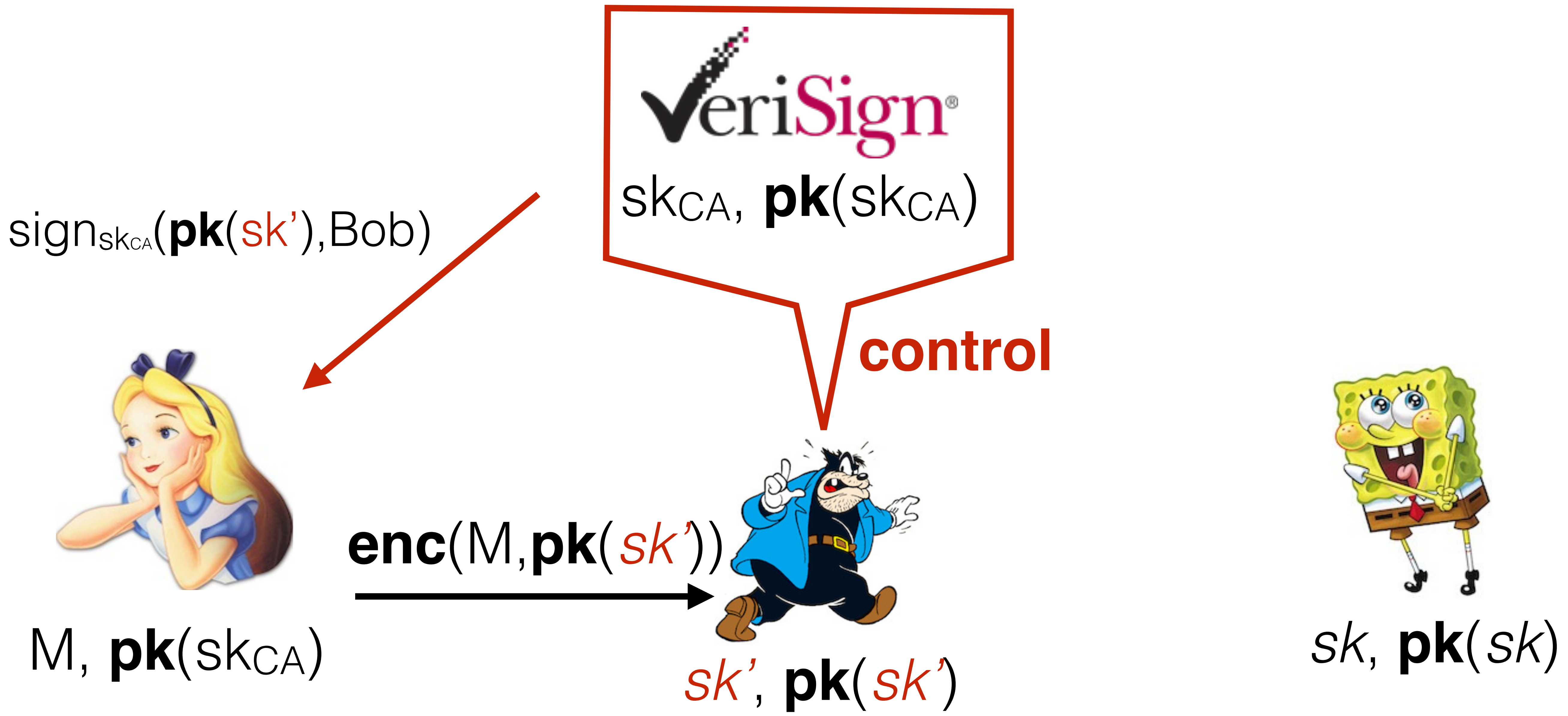
Problems with existing solution



Problems with existing solution



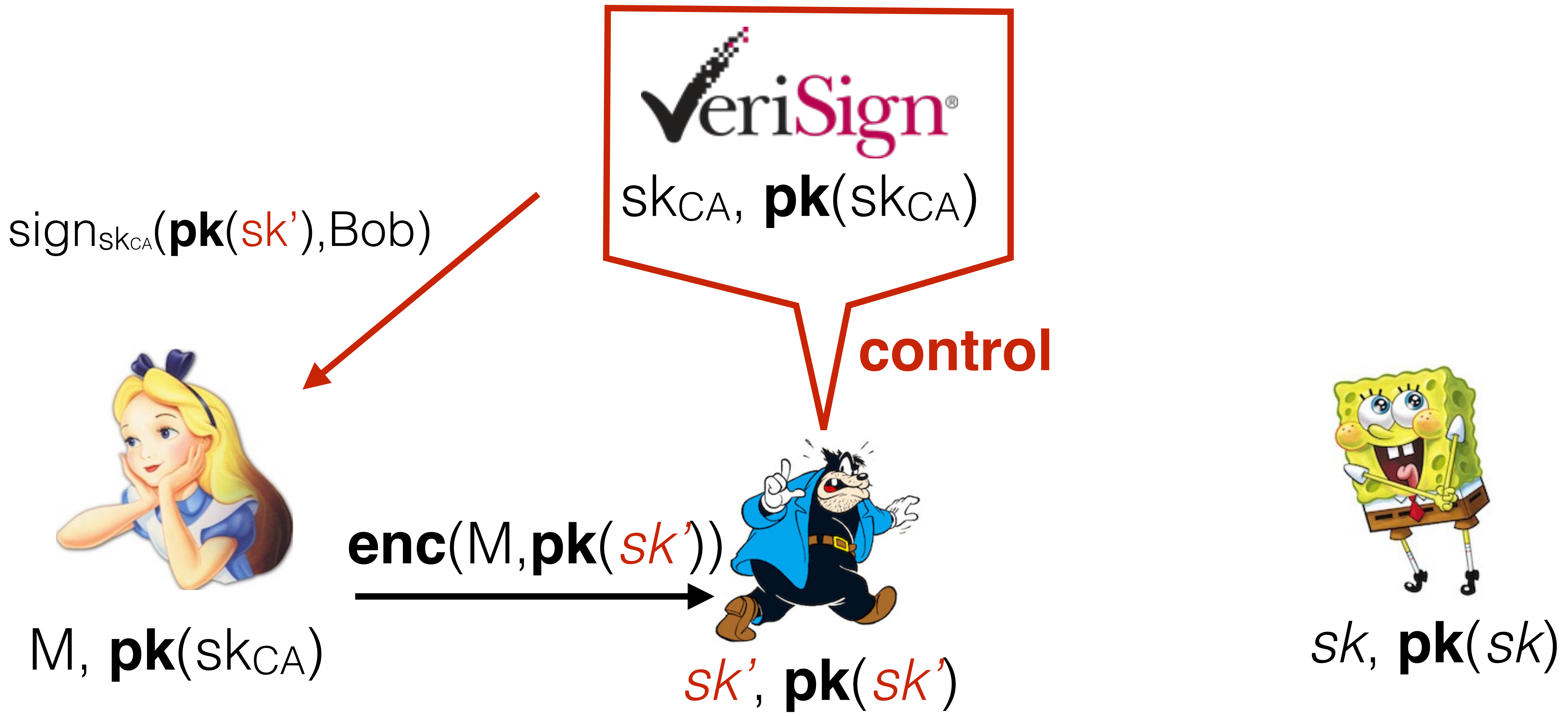
Problems with existing solution



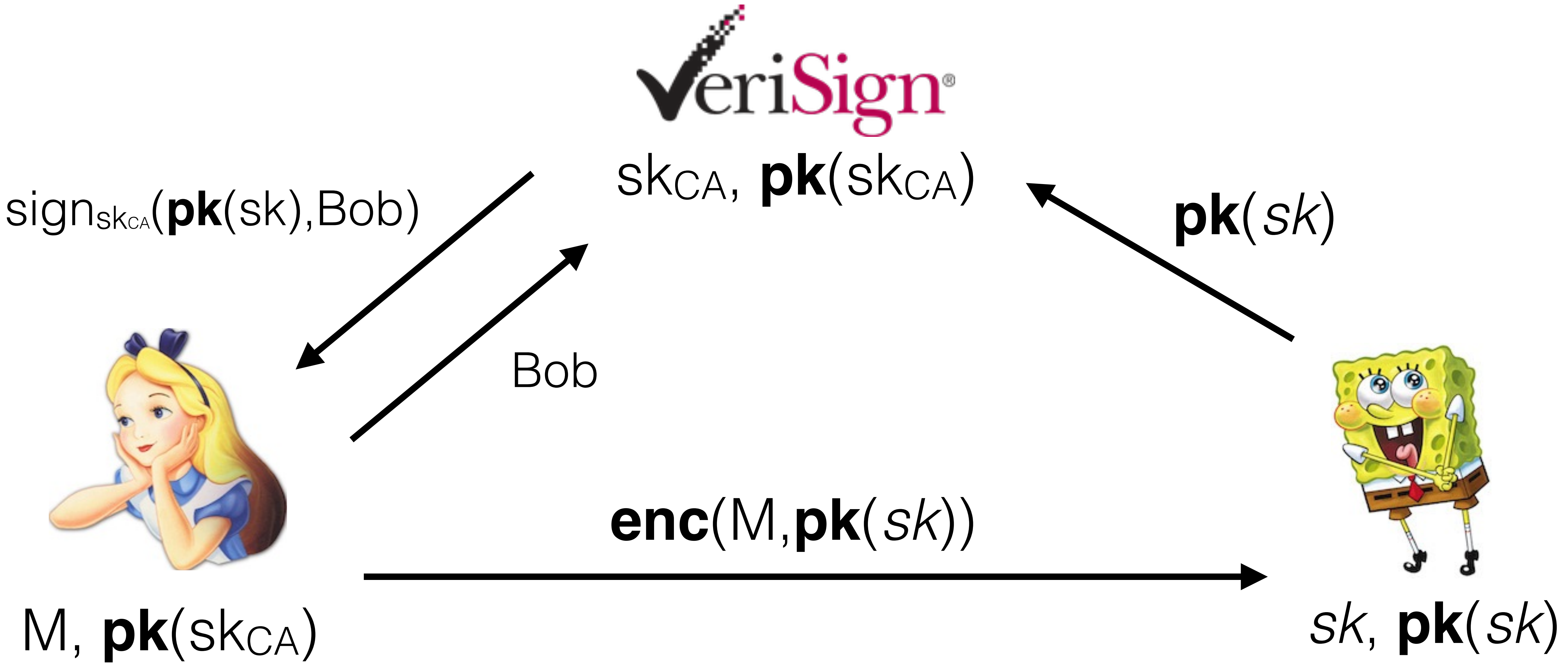
Problems with existing solution

Problem 1: Trust given to the Certificate Authority

Real attacks reported: Comodo, DigiNotar, ANSSI


VeriSign
sk_{CA}, $\mathbf{pk}(sk_{CA})$
$\text{sign}_{sk_{CA}}(\mathbf{pk}(sk'),\text{Bob})$
control
$\mathbf{enc}(M,\mathbf{pk}(sk'))$
M, $\mathbf{pk}(sk_{CA})$
sk', $\mathbf{pk}(sk')$
sk, $\mathbf{pk}(sk)$

Problems with existing solution



Problems with existing solution

sk_{CA}, $\mathbf{pk}(\mathrm{sk}_{CA})$

M, $\boxed{\mathbf{pk}(\mathrm{sk}_{CA})}$ $\longleftarrow$ **Embedded in browser**

sk, $\mathbf{pk}(sk)$

Problems with existing solution

COMODO Creating Trust Online™ GlobalSign. VeriSign®

sk_{CA}, $\mathbf{pk}(sk_{CA})$

> 100 in Firefox

M, $\boxed{\mathbf{pk}(sk_{CA})}$ ← **Embedded in browser** sk, $\mathbf{pk}(sk)$

Problems with existing solution

Problem 2: Monopoly of the certificate authority

COMODO Creating Trust Online™ GlobalSign. VeriSign®

> 100 in Firefox

sk_{CA}, **pk**(sk_{CA})

M, $\boxed{\textbf{pk}(sk_{CA})}$ $\longleftarrow$ **Embedded in browser** *sk*, **pk**(*sk*)

Problems with existing solution

Problem 2: Monopoly of the certificate authority

COMODO Creating Trust Online™ GlobalSign. VeriSign® > 100 in Firefox

sk$_{CA}$, **pk**(sk$_{CA}$)

No link between CAs

M, $\boxed{\textbf{pk}(sk_{CA})}$ ← **Embedded in browser** *sk*, **pk**(*sk*)

Problems with existing solution

Problem 2: Monopoly of the certificate authority

Problem 3: Coarse grain security

COMODO
Creating Trust Online™

GlobalSign.

VeriSign®

$\mathrm{sk}_{\mathrm{CA}}$, $\mathbf{pk}(\mathrm{sk}_{\mathrm{CA}})$

> 100 in Firefox

No link between CAs

M, $\boxed{\mathbf{pk}(\mathrm{sk}_{\mathrm{CA}})}$ ← **Embedded in browser** $\;$ sk, $\mathbf{pk}(sk)$

State of the art

Several proposals:

- Crowd-sourcing (Perspectives, DoubleCheck)
- Pinning (TACK)
- Public Log (Certificate Transparency, AKI, Sovereign Key)

↰ Log accessible to anyone, verifiable proof

Issues with public log proposals:

- Relies on trusted parties (monitors, validator, mirror)
- Single log
- No revocation
- Monopoly

Our proposal

DTKI: Distributed Transparent Key Infrastructure

- No trusted party
- Fully transparent
- Secure for multiple public log of certificates
- Revocation

Our proposal

DTKI: Distributed Transparent Key Infrastructure

- No trusted party
- Fully transparent
- Secure for multiple public log of certificates
- Revocation

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

pk(sk_{log}),d_{log}

sk_{log}, **pk**(sk_{log})

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

$$\xrightarrow{\text{Bob}, d_{log}}$$

pk$(sk_{log}), d_{log}$ sk_{log}, **pk**(sk_{log})

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

$$\xrightarrow{\text{Bob}, d_{log}}$$

$$\xleftarrow{\text{cert}_{\text{Bob},} d_{log}'}$$

$\mathbf{pk}(sk_{log}), d_{log}$

$sk_{log}, \mathbf{pk}(sk_{log})$

Structure of Public log

Data structure

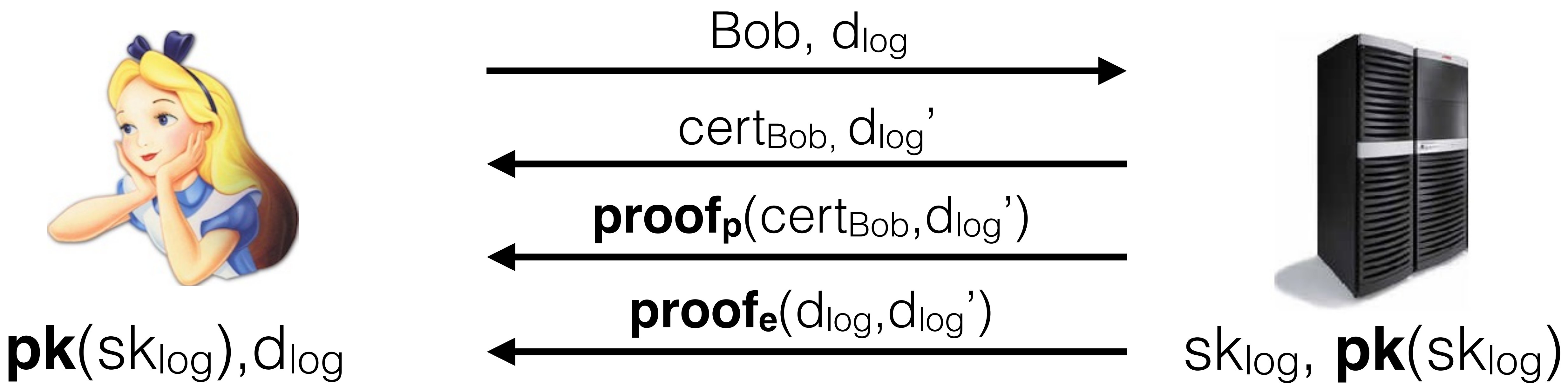
- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

$$\begin{tikzcd}
\mathbf{pk}(sk_{log}), d_{log} \arrow[r, bend left, "\text{Bob}, d_{log}"] & sk_{log}, \mathbf{pk}(sk_{log}) \arrow[l, "\text{cert}_{\text{Bob}}, d_{log}'"'] \arrow[l, bend left, "\mathbf{proof_p}(\text{cert}_{\text{Bob}}, d_{log}')"]
\end{tikzcd}$$

Structure of Public log

Data structure

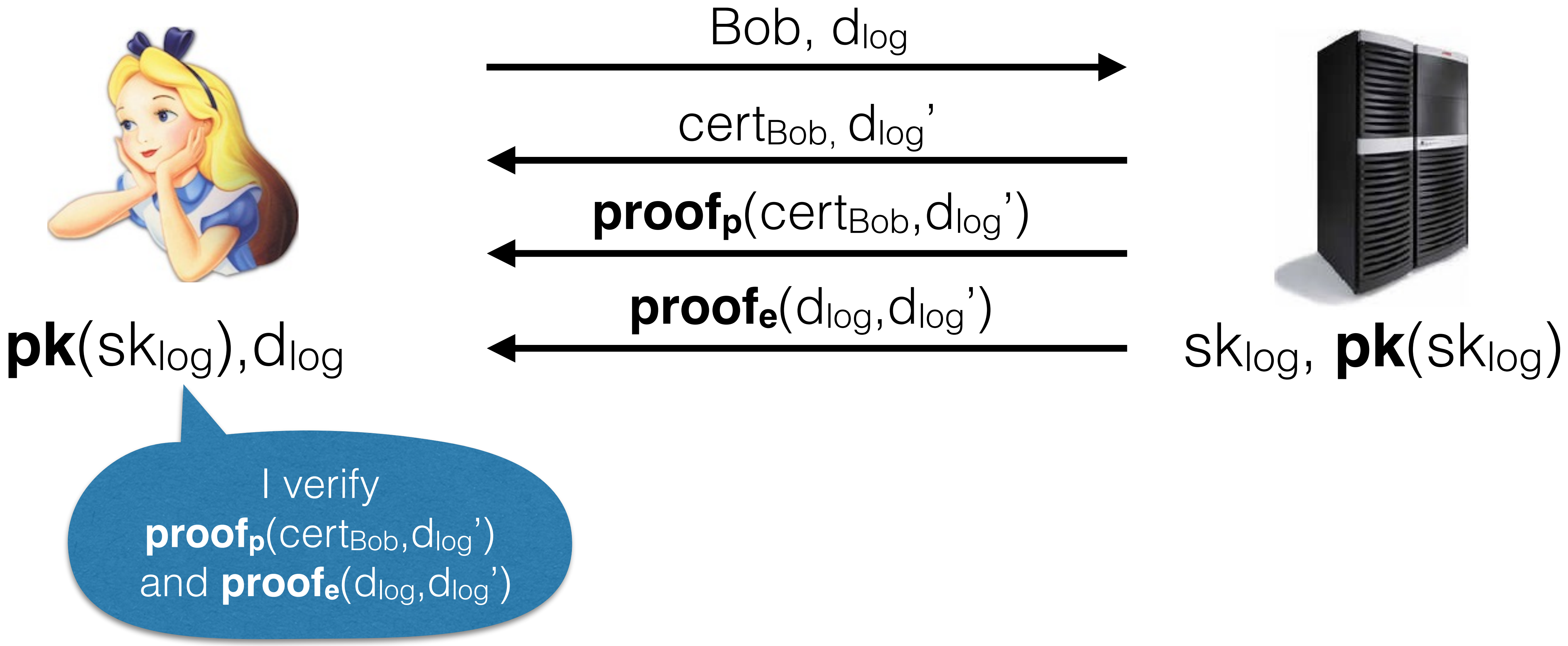
- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc


Bob, d_{log}
$\mathrm{cert}_{Bob}, d_{log}'$
$\mathbf{proof_p}(\mathrm{cert}_{Bob}, d_{log}')$
$\mathbf{proof_e}(d_{log}, d_{log}')$
$\mathbf{pk}(sk_{log}), d_{log}$
$sk_{log}, \mathbf{pk}(sk_{log})$

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc


Bob, d_{log}
$cert_{Bob}$, d_{log}'
$\mathbf{proof_p}(cert_{Bob}, d_{log}')$
$\mathbf{proof_e}(d_{log}, d_{log}')$
$\mathbf{pk}(sk_{log}), d_{log}$
sk_{log}, $\mathbf{pk}(sk_{log})$
I verify $\mathbf{proof_p}(cert_{Bob}, d_{log}')$ and $\mathbf{proof_e}(d_{log}, d_{log}')$

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc

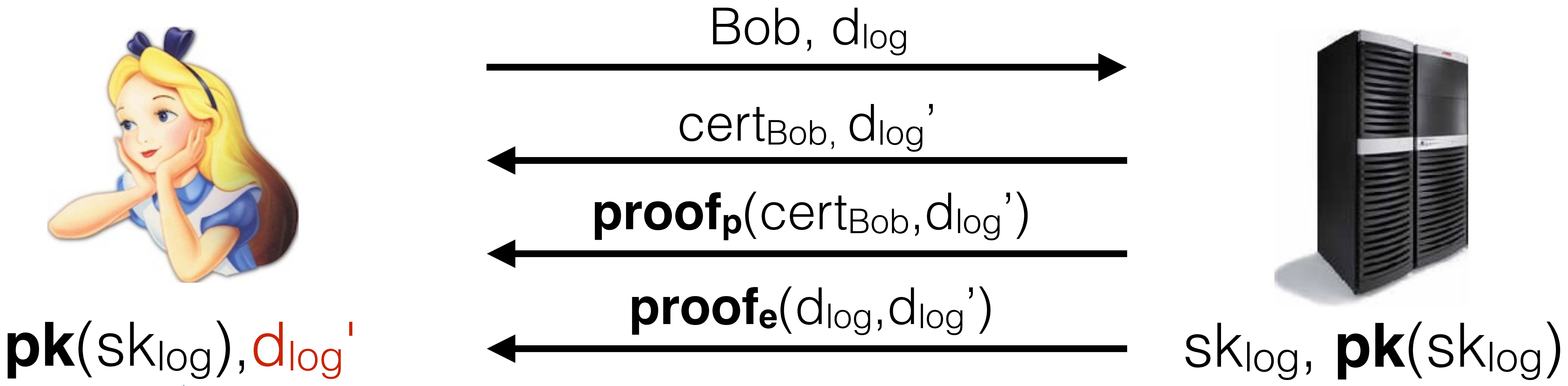
$$\begin{tikzcd}[column sep=huge] \mathbf{pk}(sk_{log}), d_{log}' \arrow[r, shift left=6, "{Bob, d_{log}}"] & sk_{log}, \mathbf{pk}(sk_{log}) \arrow[l, shift right=2, "{cert_{Bob}, d_{log}'}"'] \arrow[l, shift left=2, "{\mathbf{proof_p}(cert_{Bob}, d_{log}')}"] \arrow[l, shift left=6, "{\mathbf{proof_e}(d_{log}, d_{log}')}"] \end{tikzcd}$$

I verify
$\mathbf{proof_p}(cert_{Bob}, d_{log}')$
and $\mathbf{proof_e}(d_{log}, d_{log}')$

Structure of Public log

Data structure

- Digest of the log
- Action: Addition, deletion, modification, search, etc
- Proofs of any action, presence, absence, extension, etc


Bob, d_{log}
$cert_{Bob}$, d_{log}'
$\textbf{proof}_{\textbf{p}}(cert_{Bob}, d_{log}')$
$\textbf{proof}_{\textbf{e}}(d_{log}, d_{log}')$
$\textbf{pk}(sk_{log}), d_{log}'$
sk_{log}, $\textbf{pk}(sk_{log})$

I verify $\textbf{proof}_{\textbf{p}}(cert_{Bob}, d_{log}')$ and $\textbf{proof}_{\textbf{e}}(d_{log}, d_{log}')$

Size and verification time of proofs must be O(log(n))

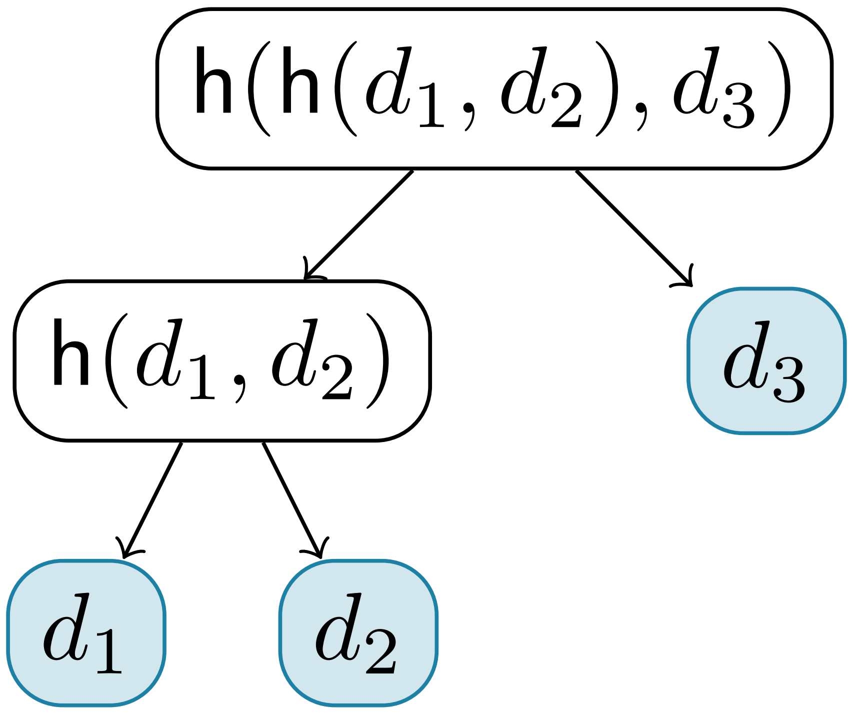
ChronTree

- Based on a binary hash tree

$$\begin{tikzcd}
 & & \mathsf{h}(\mathsf{h}(d_1, d_2), d_3) \arrow[dl] \arrow[dr] & \\
 & \mathsf{h}(d_1, d_2) \arrow[dl] \arrow[dr] & & d_3 \\
d_1 & & d_2 &
\end{tikzcd}$$

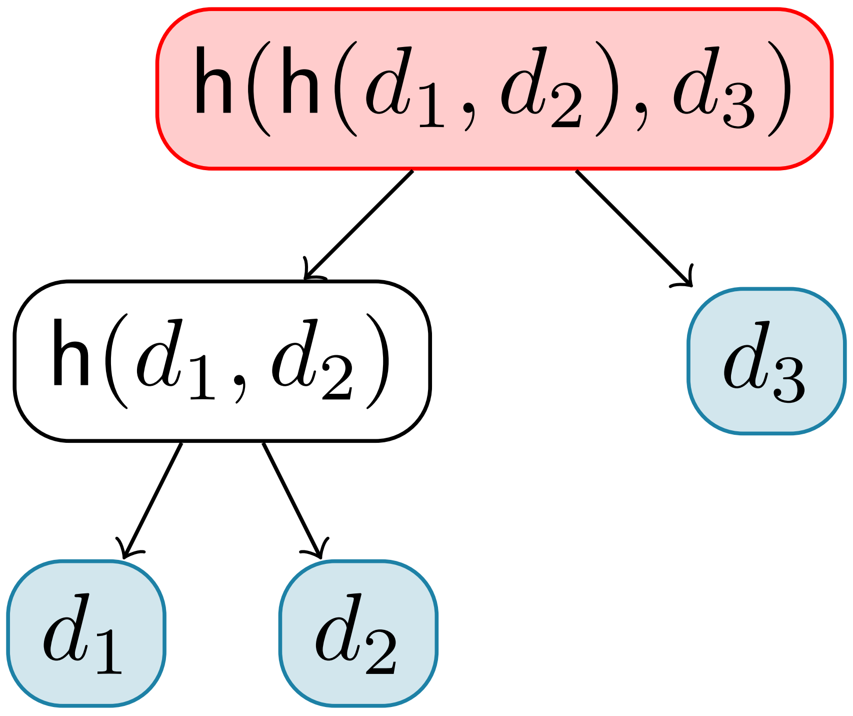
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves



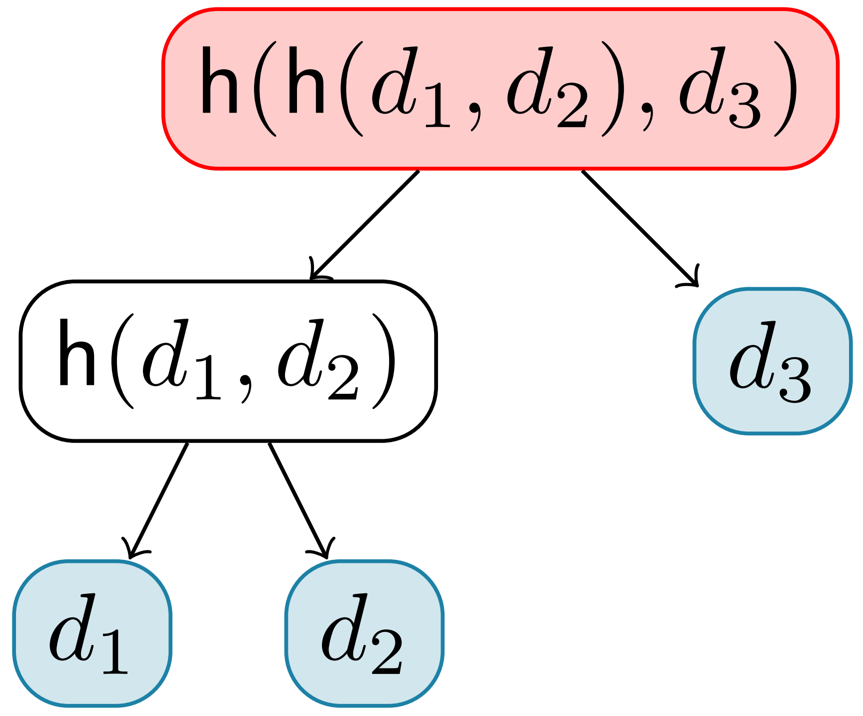
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root



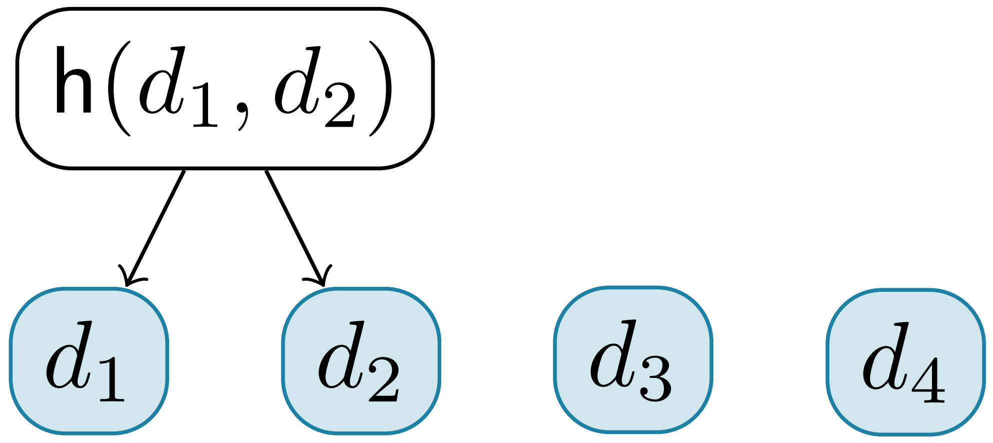
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*



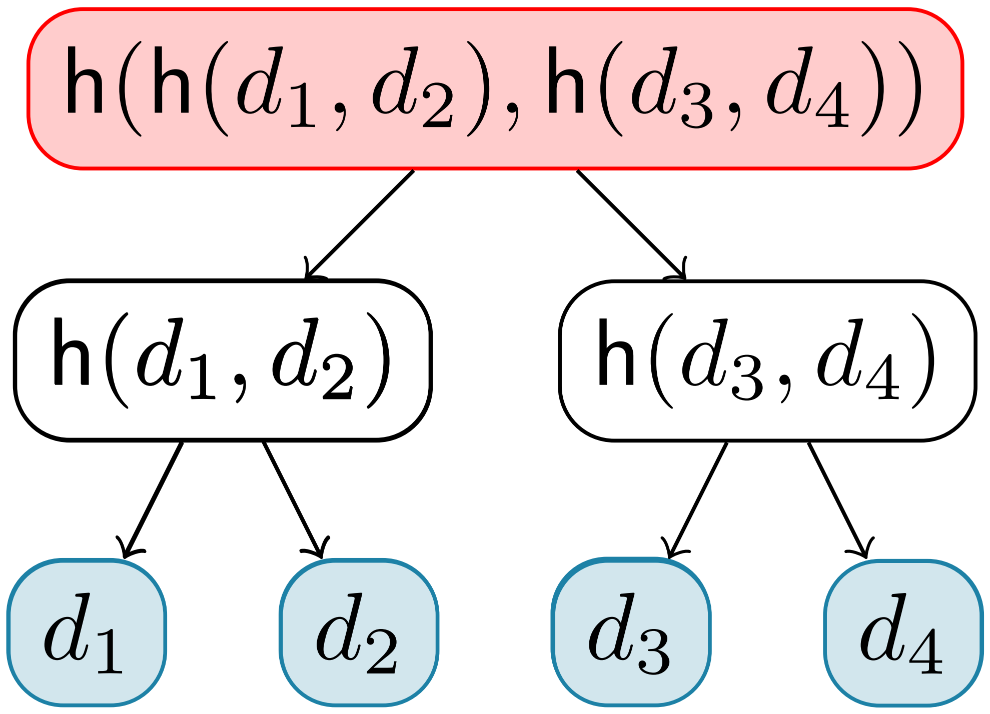
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*



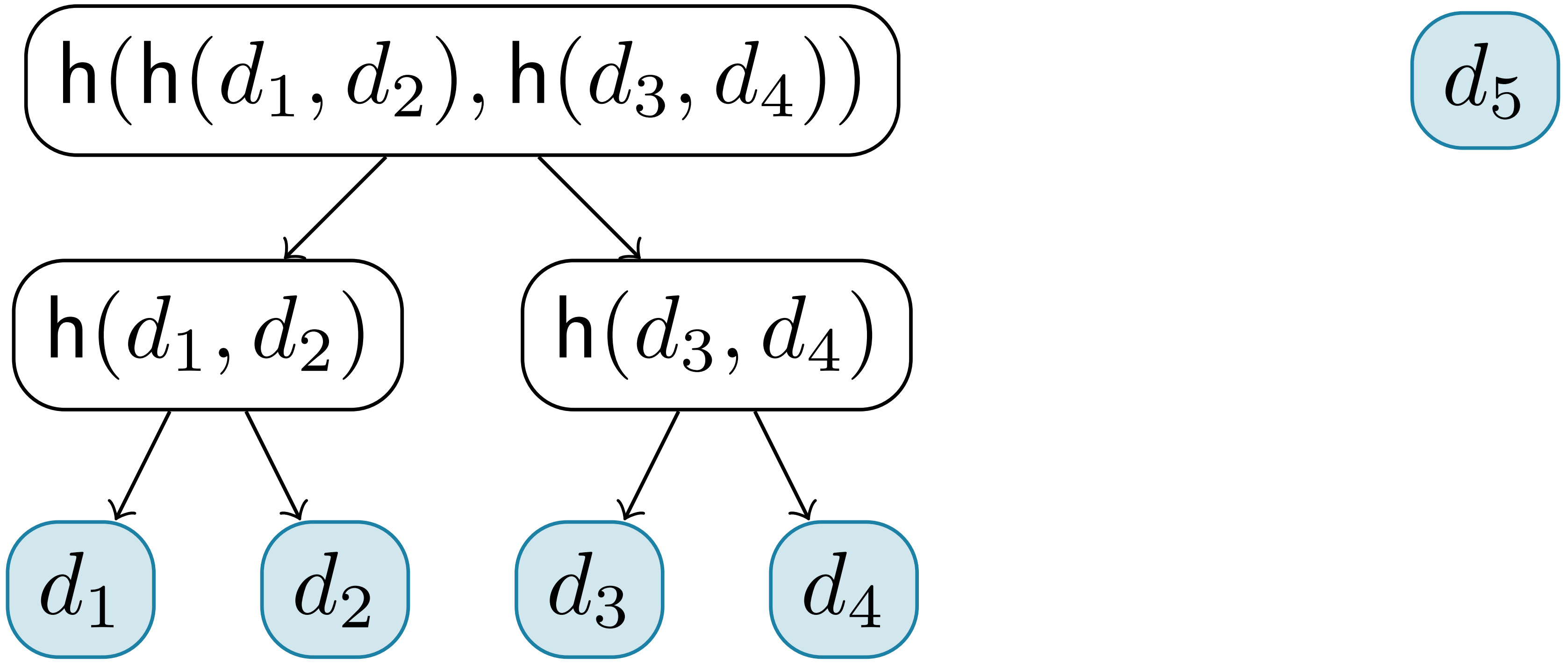
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*



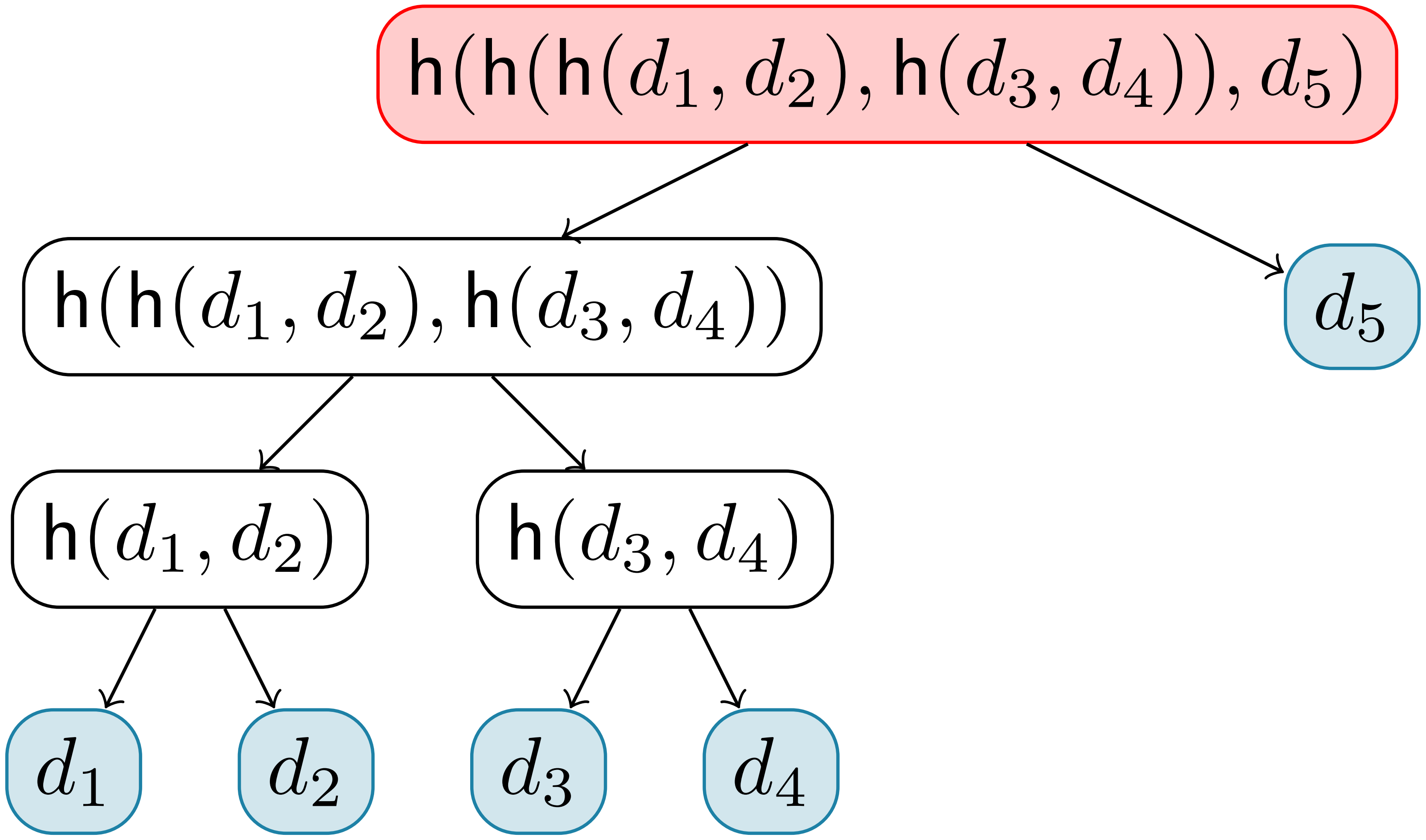
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*



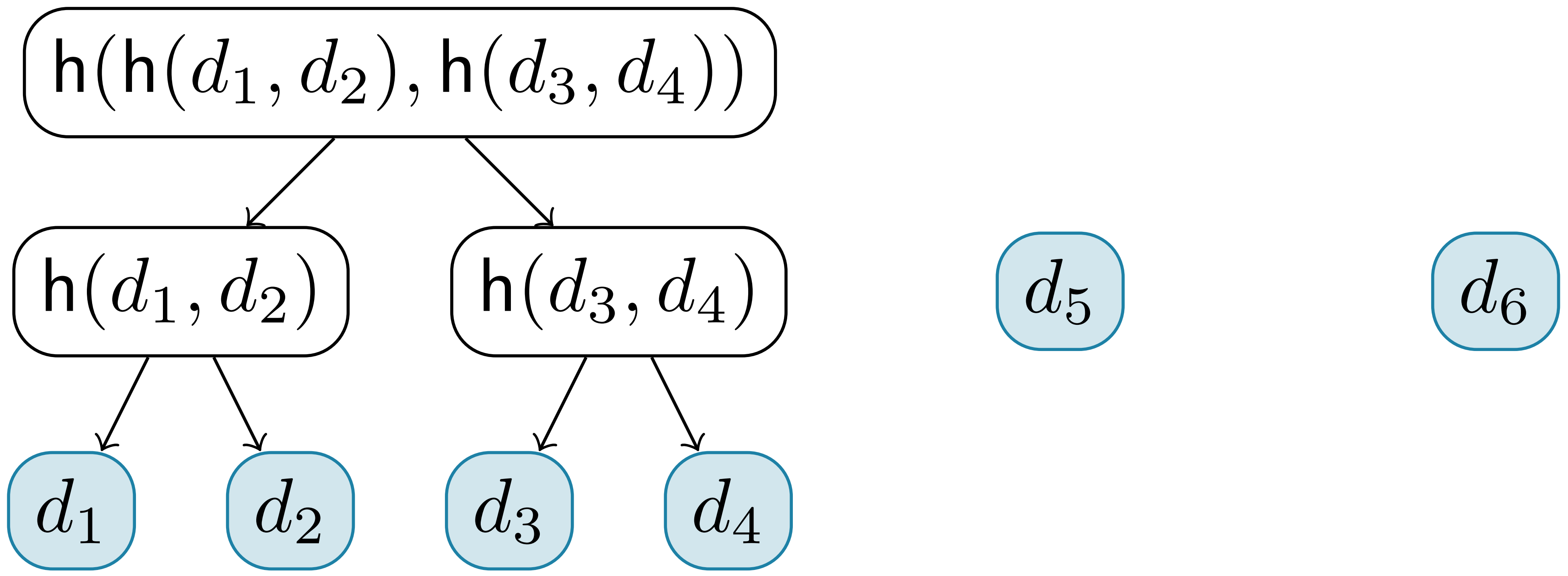
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*


$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), d_5)$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
d_5
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_1
d_2
d_3
d_4

ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*



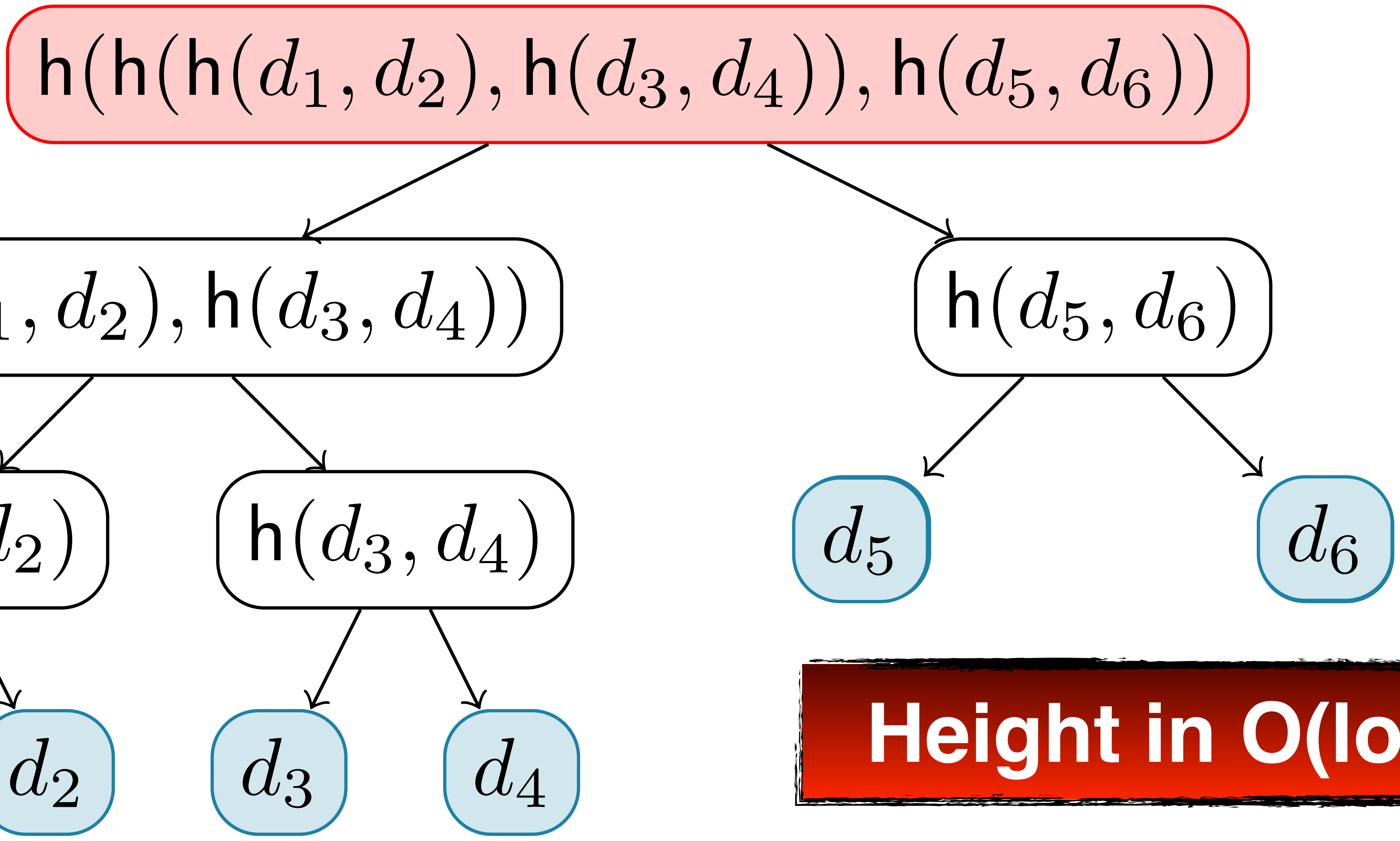
ChronTree

- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*

$$\begin{tikzcd}
 & & & & \mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6)) \arrow[dlll] \arrow[drr] & & & \\
 & \mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)) \arrow[dl] \arrow[dr] & & & & & \mathsf{h}(d_5, d_6) \arrow[dl] \arrow[dr] & \\
\mathsf{h}(d_1, d_2) \arrow[d] \arrow[dr] & & \mathsf{h}(d_3, d_4) \arrow[d] \arrow[dr] & & & d_5 & & d_6 \\
d_1 & d_2 & d_3 & d_4 & & & &
\end{tikzcd}$$

ChronTree

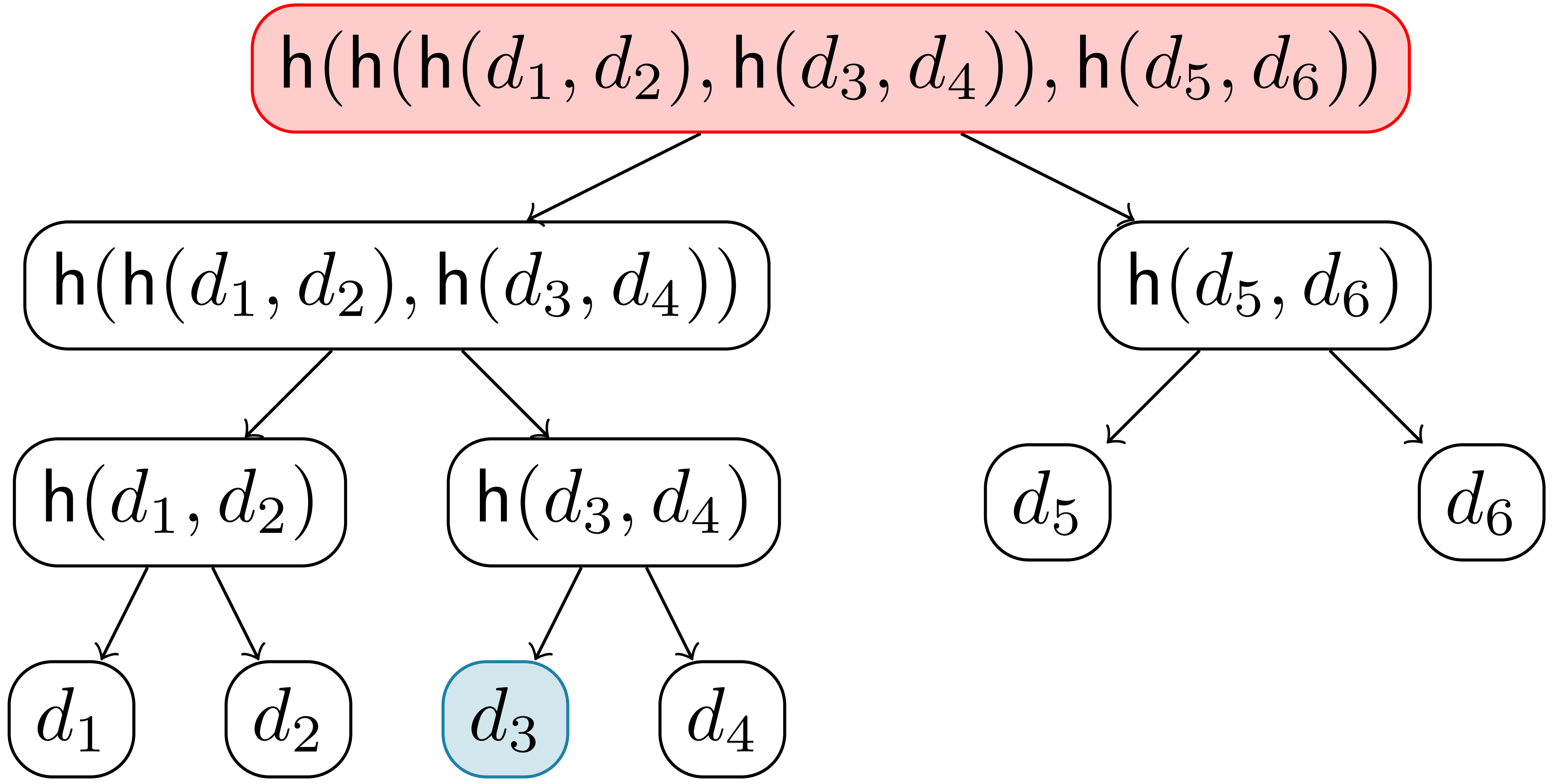
- Based on a binary hash tree
- Data are stored on the leaves
- The digest of the data is the label of the root
- Addition *on the right*


$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Height in O(log(n))

ChronTree

Proof of presence of some data in the digest


$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

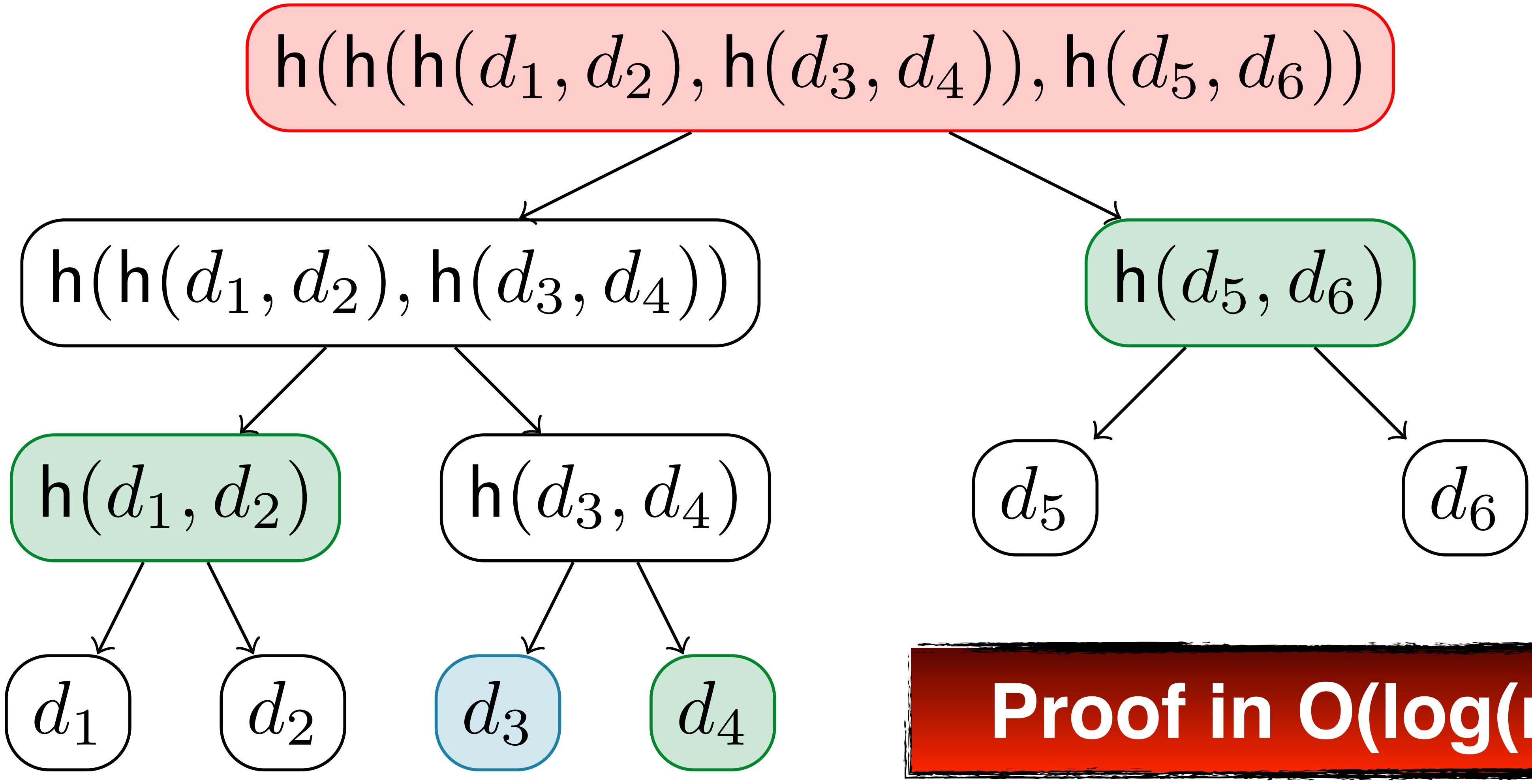
ChronTree

Proof of presence of some data in the digest

$$\begin{tikzcd}[column sep=tiny]
 & & & & & & & \mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6)) \arrow[dllll] \arrow[drr] & & & \\
 & & & \mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)) \arrow[dll] \arrow[drr] & & & & & & \mathsf{h}(d_5, d_6) \arrow[dl] \arrow[dr] & \\
 & \mathsf{h}(d_1, d_2) \arrow[dl] \arrow[dr] & & & & \mathsf{h}(d_3, d_4) \arrow[dl] \arrow[dr] & & & d_5 & & d_6 \\
d_1 & & d_2 & & d_3 & & d_4 & & & &
\end{tikzcd}$$

ChronTree

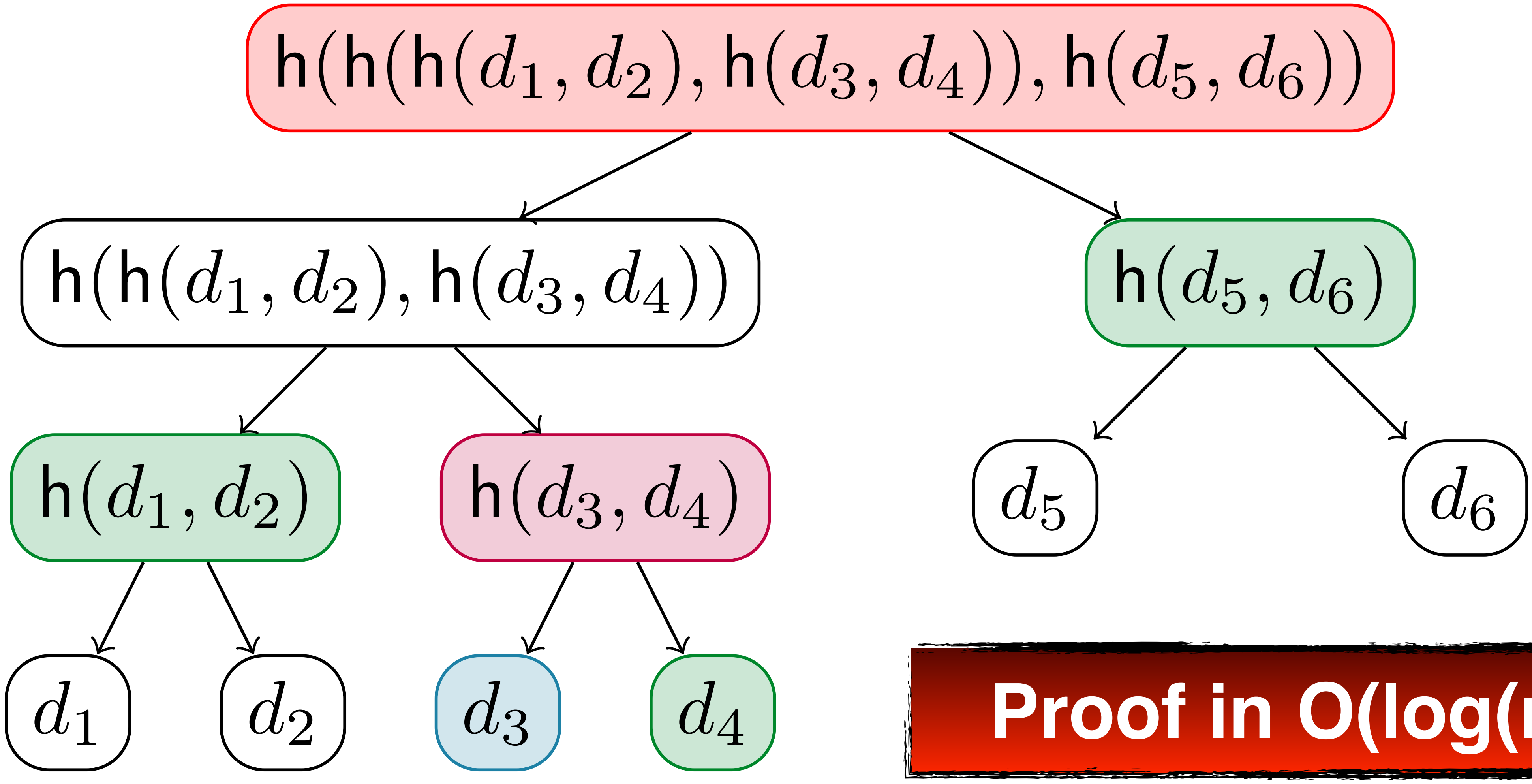
Proof of presence of some data in the digest


$$\begin{tikzcd}
 & & & \mathsf{h}(\mathsf{h}(\mathsf{h}(d_1,d_2),\mathsf{h}(d_3,d_4)),\mathsf{h}(d_5,d_6)) \arrow[dl] \arrow[dr] & \\
 & \mathsf{h}(\mathsf{h}(d_1,d_2),\mathsf{h}(d_3,d_4)) \arrow[dl] \arrow[dr] & & & \mathsf{h}(d_5,d_6) \arrow[dl] \arrow[dr] \\
\mathsf{h}(d_1,d_2) \arrow[d] \arrow[dr] & & \mathsf{h}(d_3,d_4) \arrow[d] \arrow[dr] & d_5 & & d_6 \\
d_1 & d_2 & d_3 & d_4 & &
\end{tikzcd}$$

Proof in O(log(n))

ChronTree

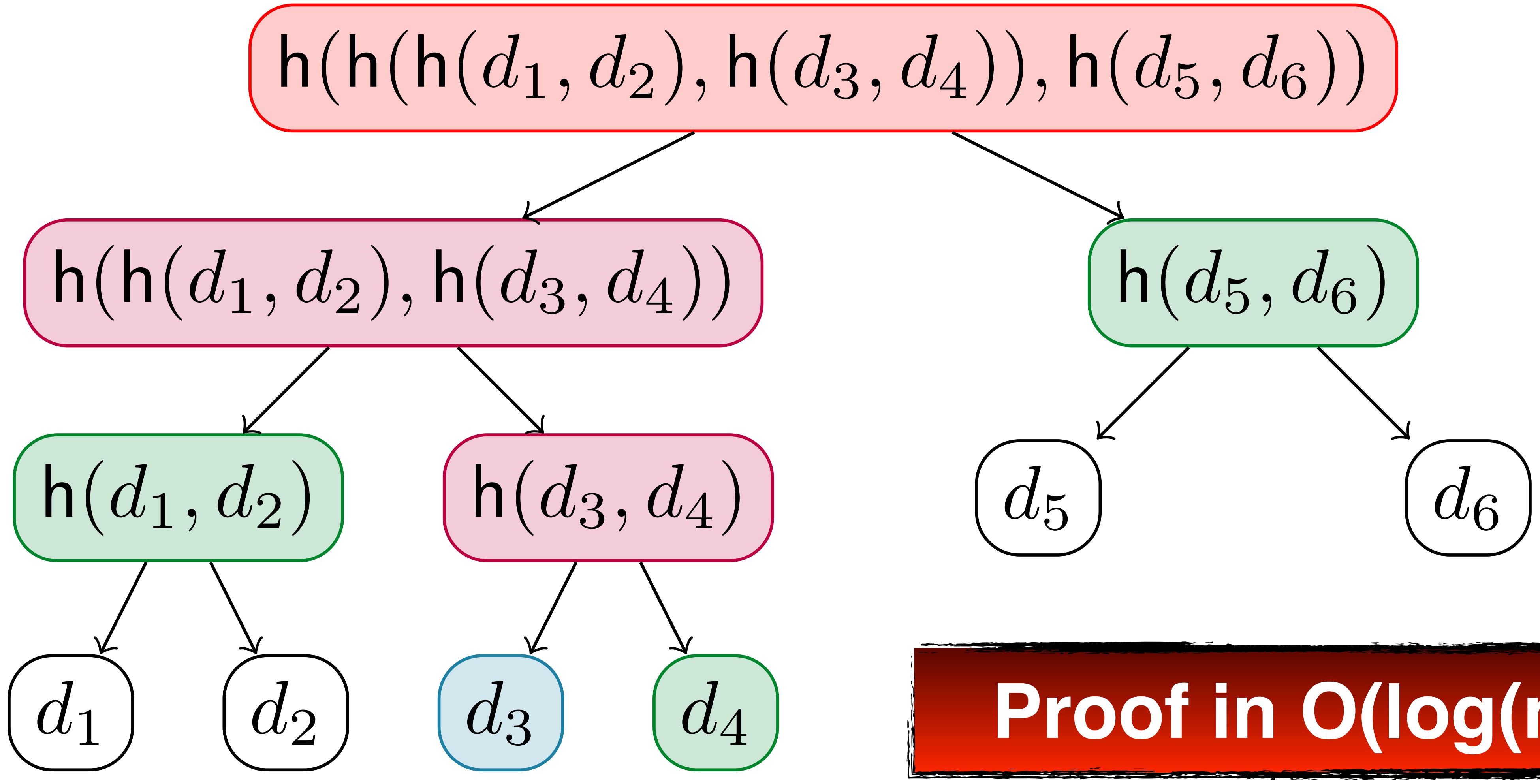
Proof of presence of some data in the digest


$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

ChronTree

Proof of presence of some data in the digest

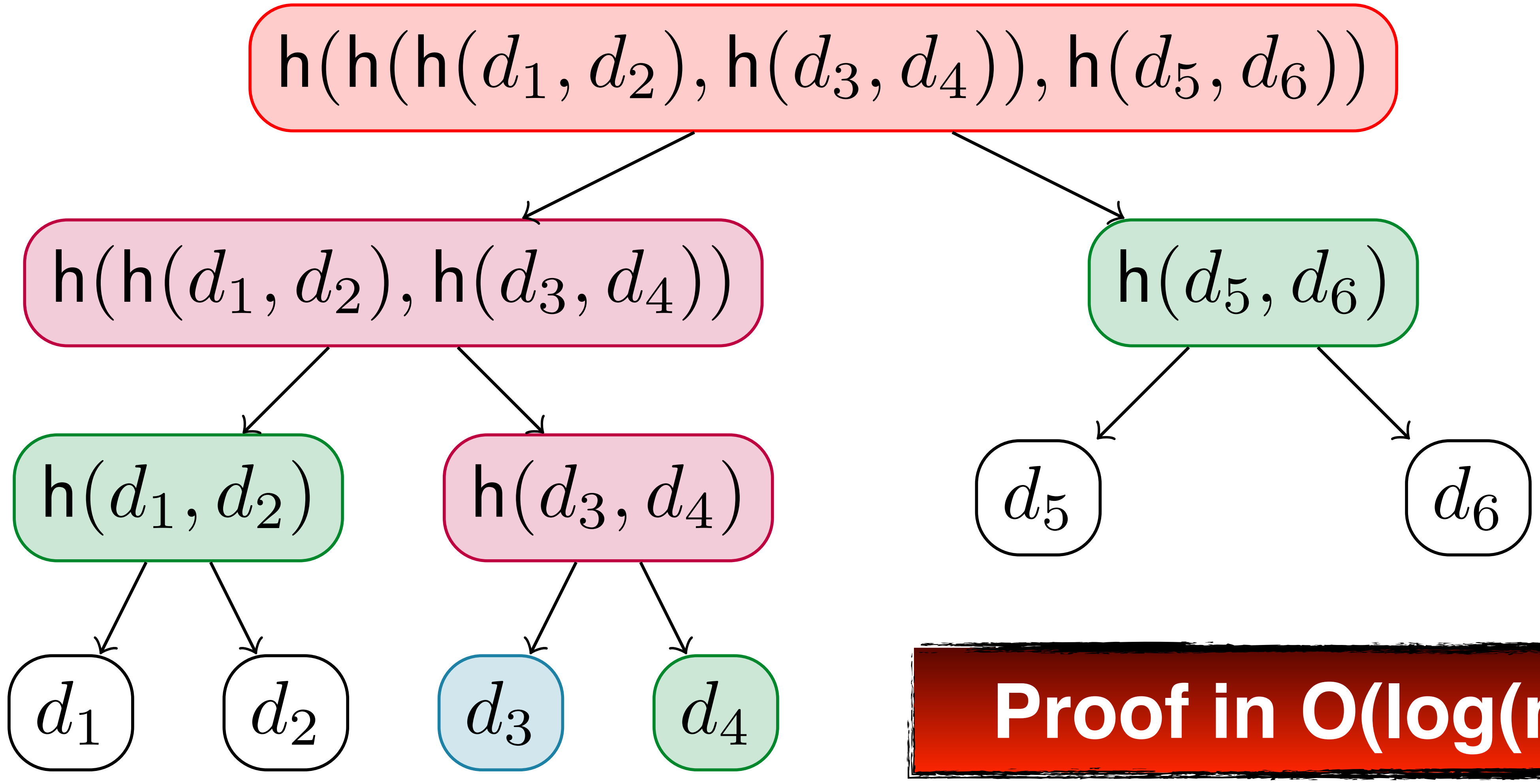


$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

ChronTree

Proof of presence of some data in the digest

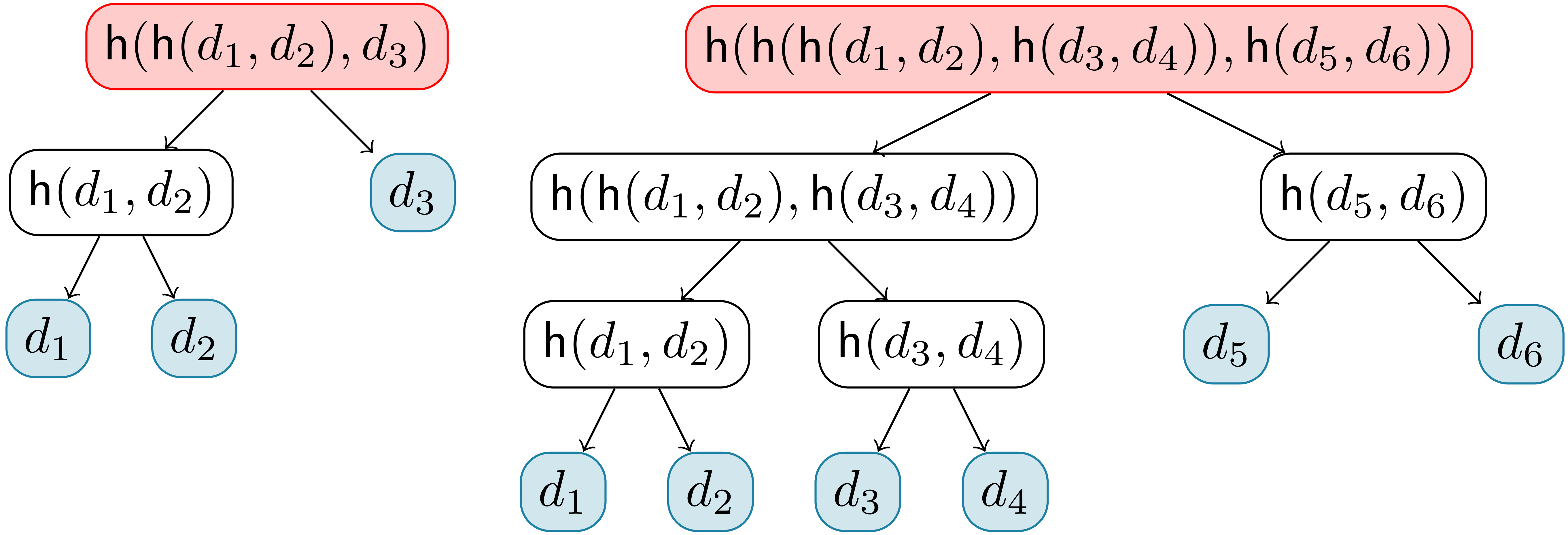

$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

Verification of the proof in O(log(n))

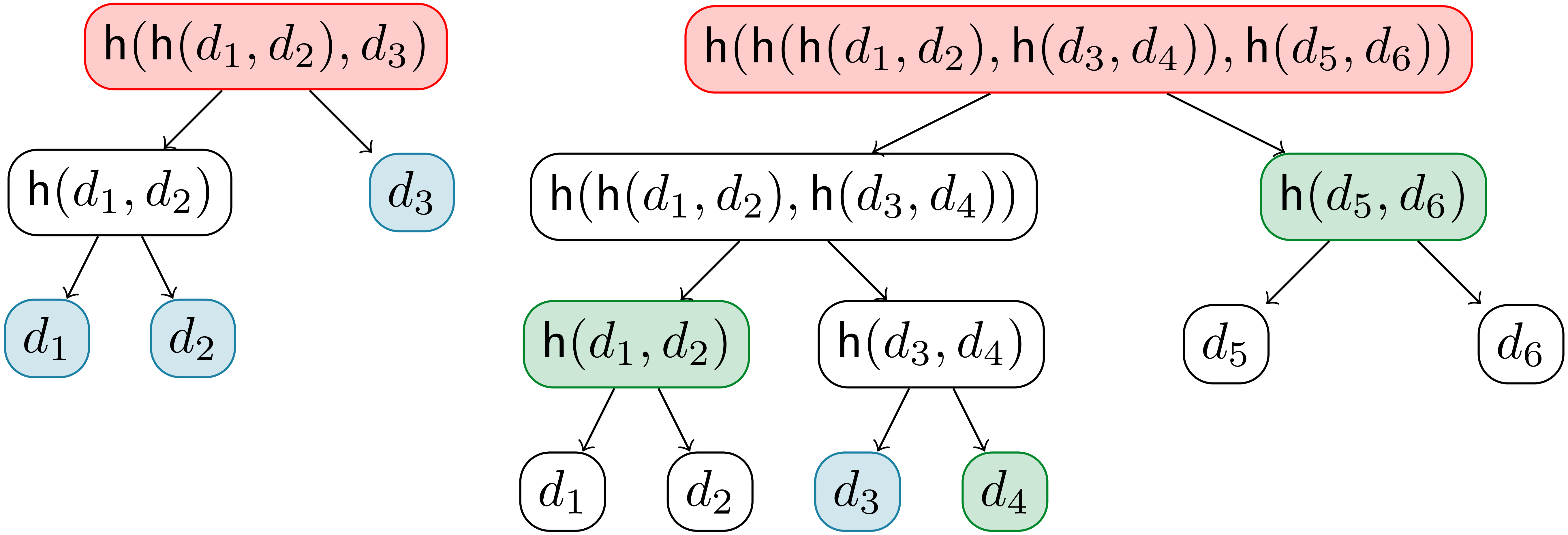
ChronTree

Proof of extension between two digests


$\mathsf{h}(\mathsf{h}(d_1, d_2), d_3)$
$\mathsf{h}(d_1, d_2)$
d_3
d_1
d_2
$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

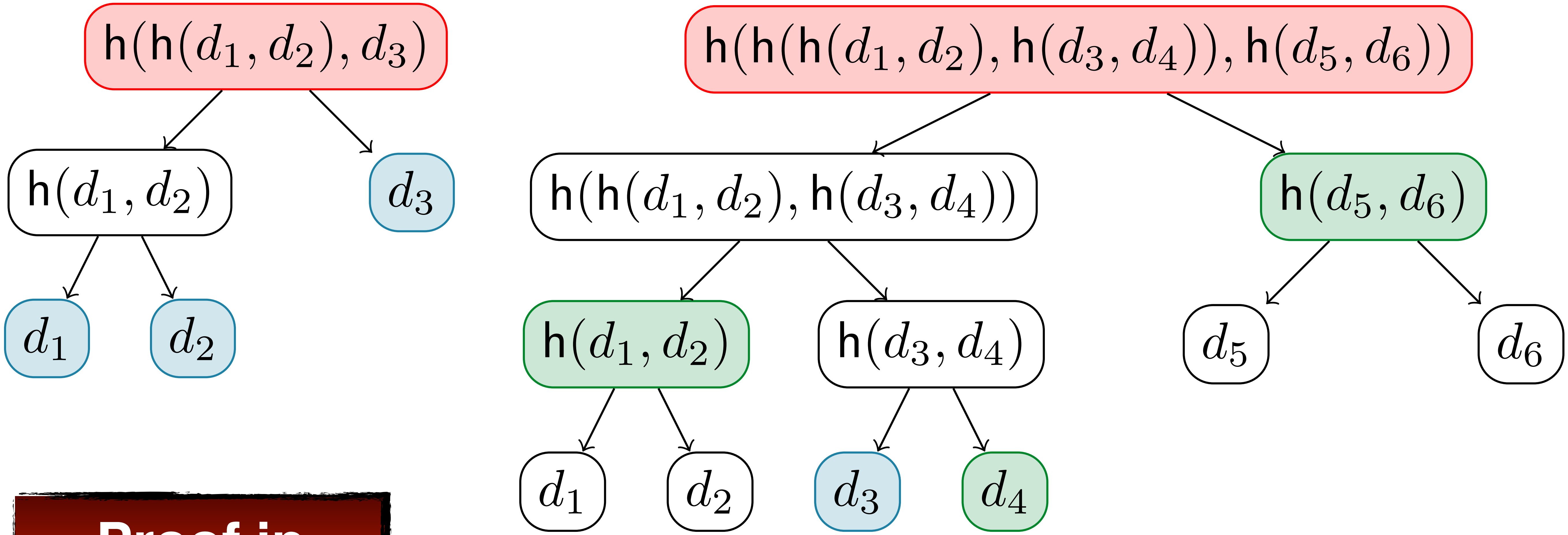
ChronTree

Proof of extension between two digests


$\mathsf{h}(\mathsf{h}(d_1, d_2), d_3)$
$\mathsf{h}(d_1, d_2)$
d_3
d_1
d_2
$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

ChronTree

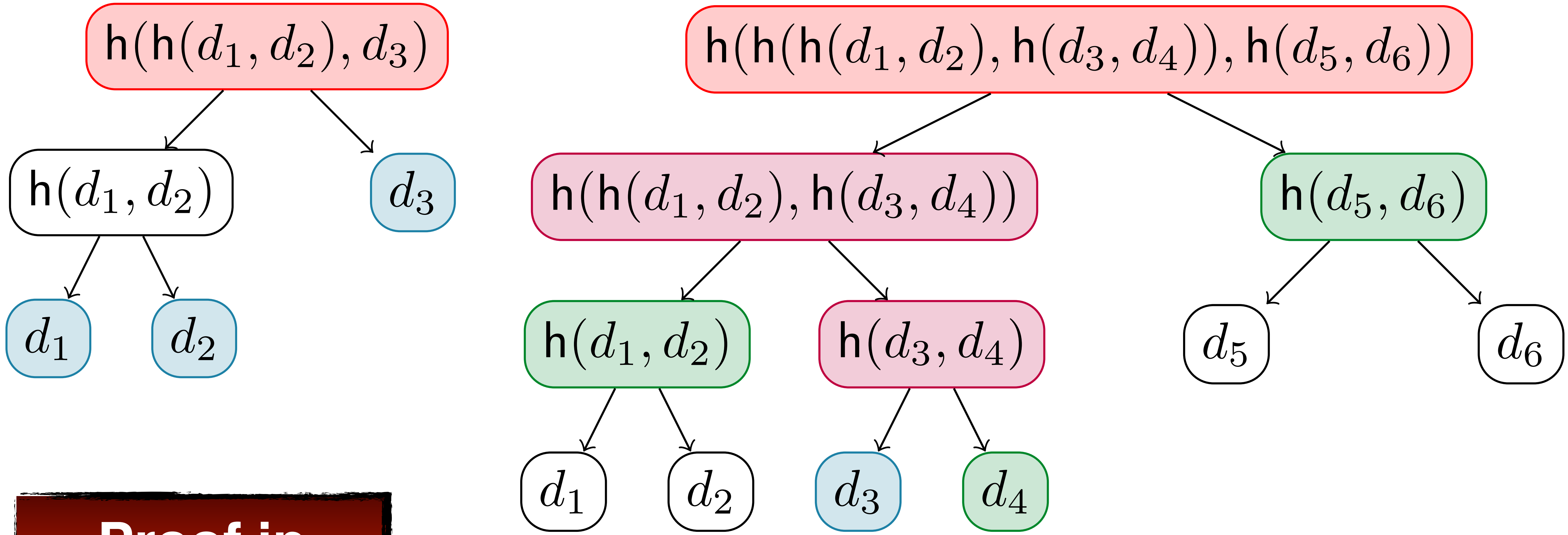
Proof of extension between two digests


$\mathsf{h}(\mathsf{h}(d_1, d_2), d_3)$
$\mathsf{h}(d_1, d_2)$
d_3
d_1
d_2
$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

ChronTree

Proof of extension between two digests

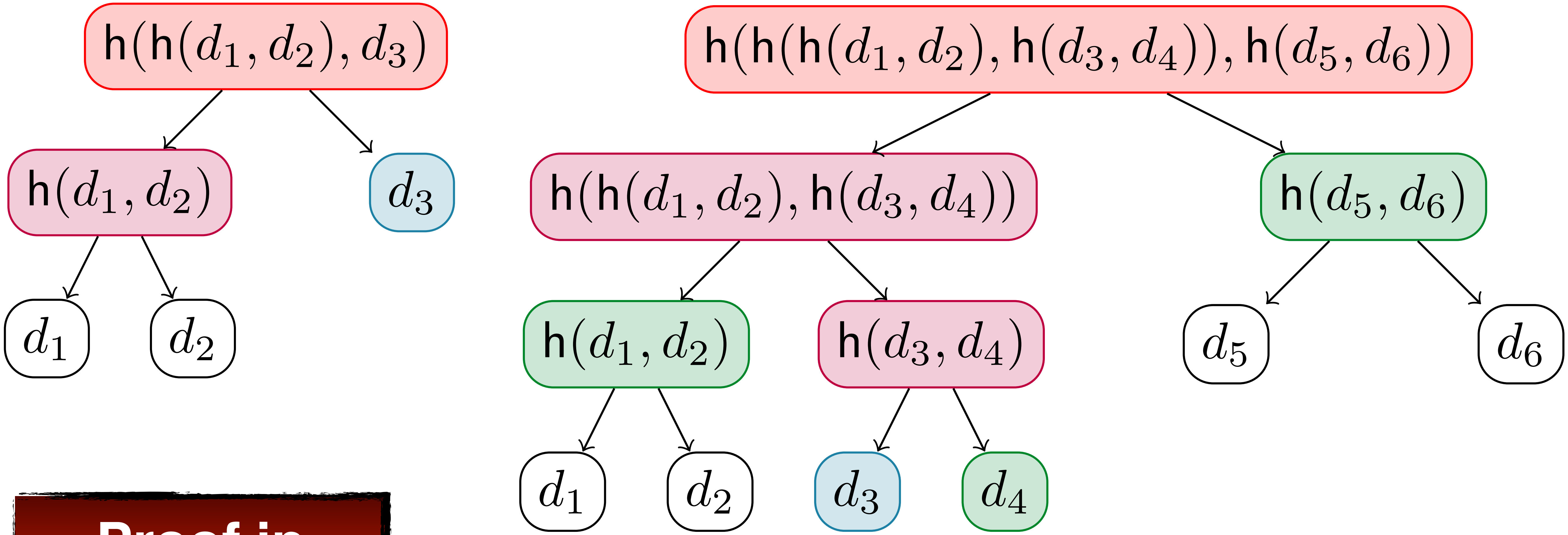


$\mathsf{h}(\mathsf{h}(d_1, d_2), d_3)$
$\mathsf{h}(d_1, d_2)$
d_3
d_1
d_2
$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

ChronTree

Proof of extension between two digests

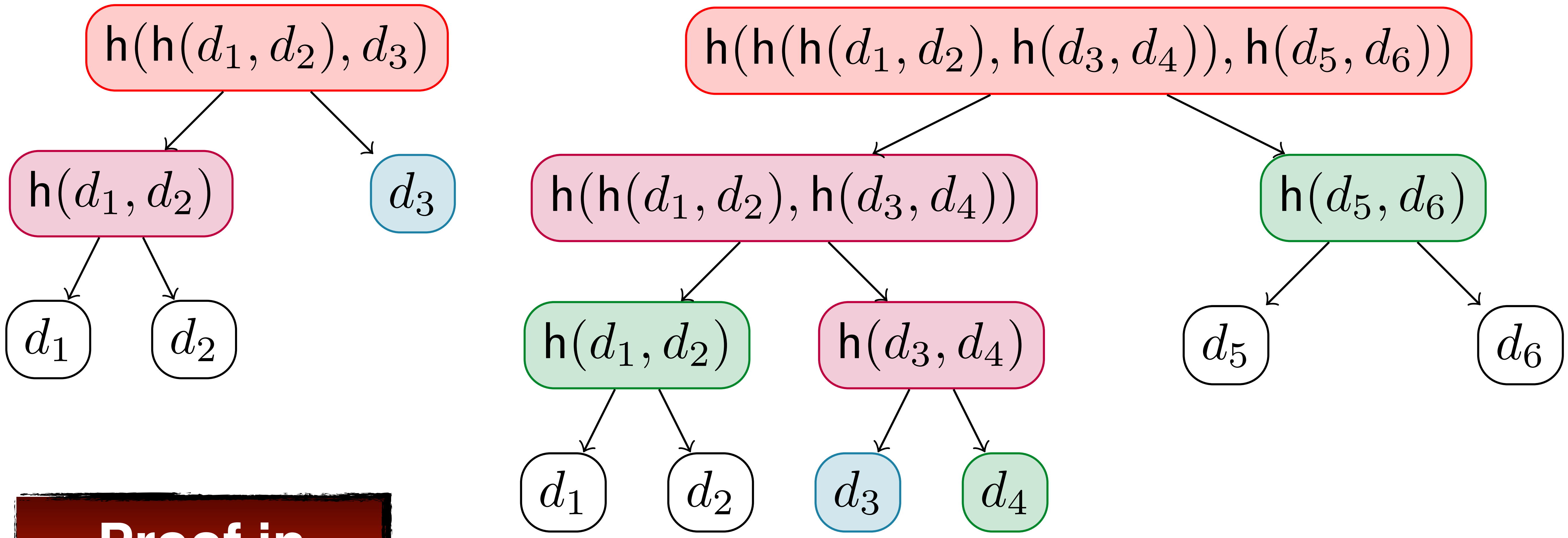


$\mathsf{h}(\mathsf{h}(d_1, d_2), d_3)$
$\mathsf{h}(d_1, d_2)$
d_3
d_1
d_2
$\mathsf{h}(\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4)), \mathsf{h}(d_5, d_6))$
$\mathsf{h}(\mathsf{h}(d_1, d_2), \mathsf{h}(d_3, d_4))$
$\mathsf{h}(d_5, d_6)$
$\mathsf{h}(d_1, d_2)$
$\mathsf{h}(d_3, d_4)$
d_5
d_6
d_1
d_2
d_3
d_4

Proof in O(log(n))

ChronTree

Proof of extension between two digests



Proof in O(log(n))

Verification of the proof in O(log(n))

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

ChronTree

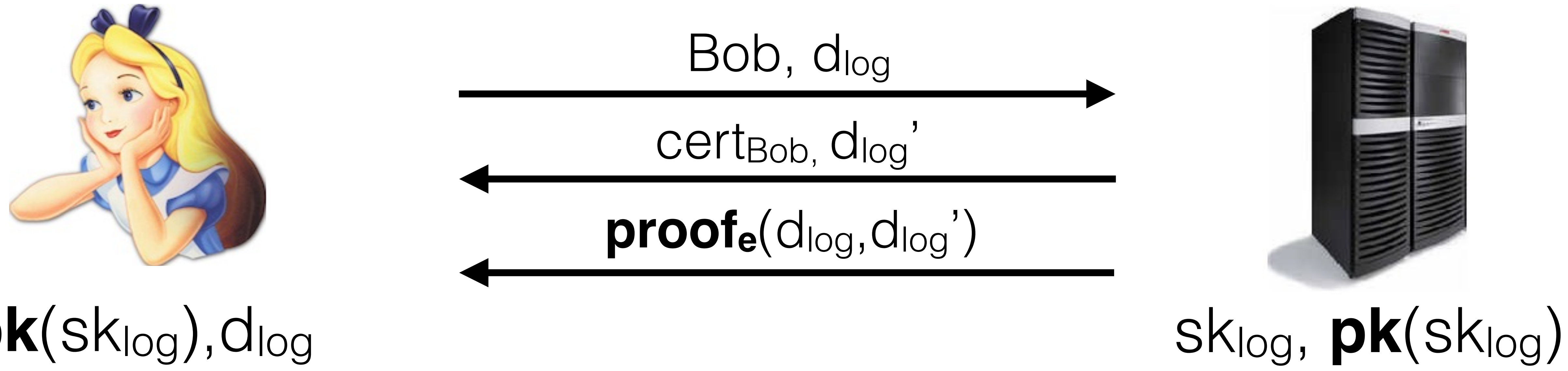
- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Why proof of extension and not addition ?

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Why proof of extension and not addition ?


Bob, d_{log}
$cert_{Bob}$, d_{log}'
$\textbf{proof}_{\textbf{e}}(d_{log}, d_{log}')$
$\textbf{pk}(sk_{log}), d_{log}$
sk_{log}, $\textbf{pk}(sk_{log})$

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Why proof of extension and not addition ?

Bob, d_{log} $\longrightarrow$

$\longleftarrow$ $\text{cert}_{Bob}, d_{log}'$

$\longleftarrow$ $\textbf{proof}_{\textbf{e}}(d_{log}, d_{log}')$

$\textbf{pk}(sk_{log}), d_{log}$

$sk_{log}, \textbf{pk}(sk_{log})$

Unbounded number of certificates added between d_{log} and d_{log}'

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Issues with Chrontree

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Issues with Chrontree

- Deletion and modification of data not possible
  - No revocation
- No efficient proof of absence
  - Possible stripping attack
  - Possibility of adding fake certificate

ChronTree

- Digest in constant size (size of the hash)
- Action: addition
- Proofs of presence and extension

Issues with Chrontree

- Deletion and modification of data not possible

  No revocation
- No efficient proof of absence

  Possible stripping attack

  Possibility of adding fake certificate

Introduction of a new data structure: AVL hash tree

AVL hash tree

Data structure:

- Digest in constant size
- Action: addition, deletion, modification, search
- Proofs of addition, deletion,modification
- Proofs of presence, absence

AVL hash tree

Data structure:

- Digest in constant size
- Action: addition, deletion, modification, search
- Proofs of addition, deletion,modification
- Proofs of presence, absence
- No proof of extension

AVL hash tree

Data structure:

- Digest in constant size
- Action: addition, deletion, modification, search
- Proofs of addition, deletion,modification
- Proofs of presence, absence
- No proof of extension

Combination of ChronTree and AVL hash tree

AVL hash tree

Data structure:

- Digest in constant size
- Action: addition, deletion, modification, search
- Proofs of addition, deletion,modification
- Proofs of presence, absence
- No proof of extension

Combination of ChronTree and AVL hash tree

- **AVL hash tree stores the current state**

  Data: certificates
- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

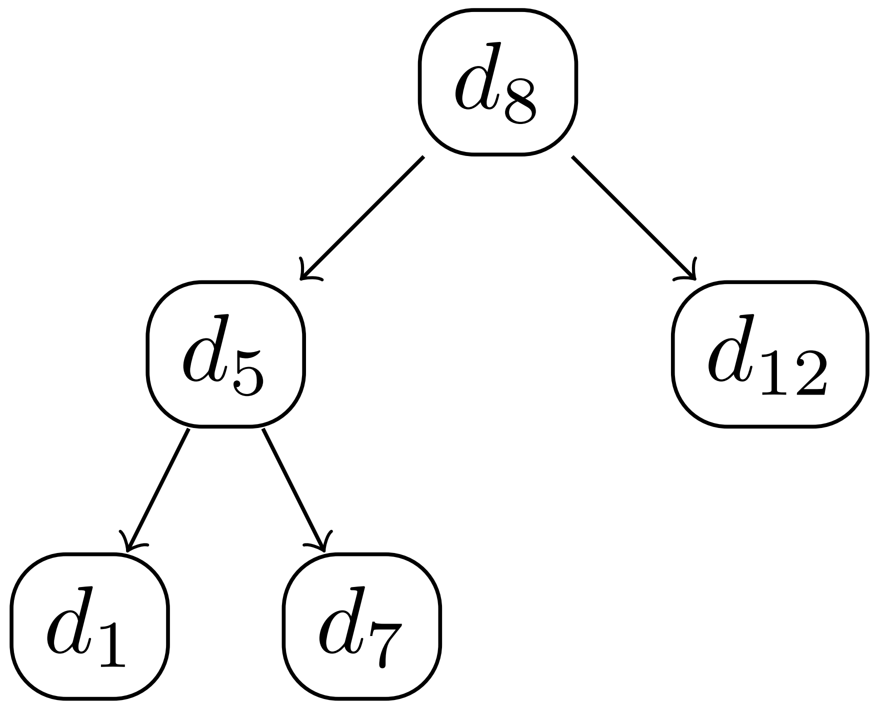
AVL hash tree

- Based on binary search tree

AVL hash tree

- Based on binary search tree

Order $\mathcal{R}$ on data

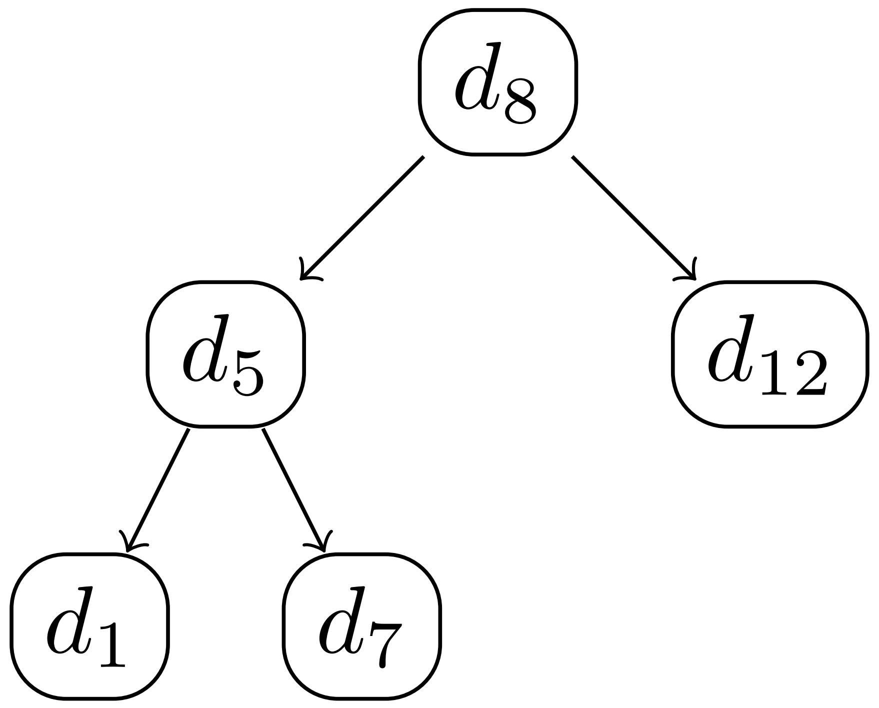


$$\begin{tikzcd}
 & & d_8 \arrow[dl] \arrow[dr] & \\
 & d_5 \arrow[dl] \arrow[d] & & d_{12} \\
d_1 & d_7 & &
\end{tikzcd}$$

AVL hash tree

- Based on binary search tree
- Satisfies the AVL property

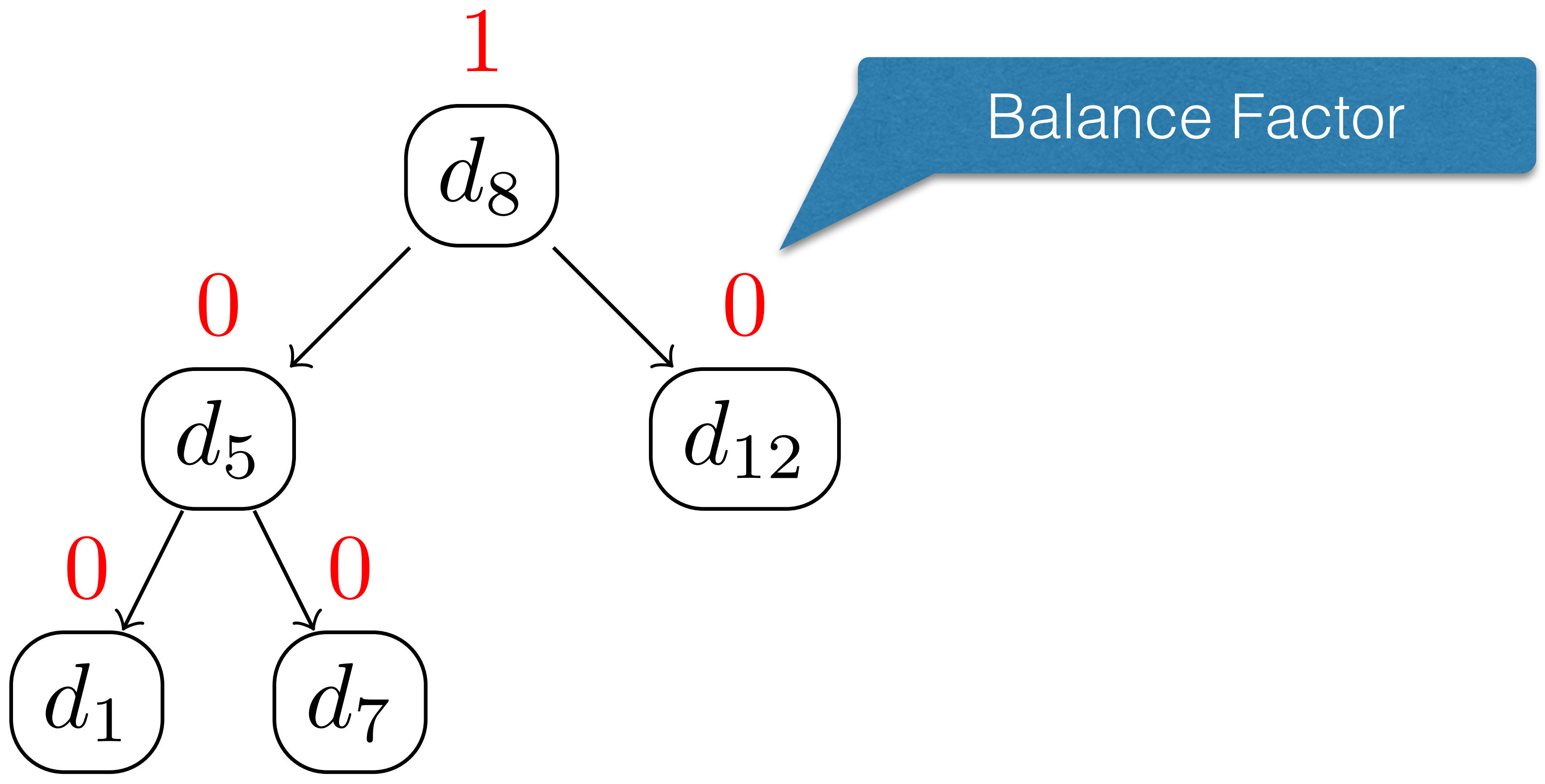
Order $\mathcal{R}$ on data



AVL hash tree

Order $\mathcal{R}$ on data

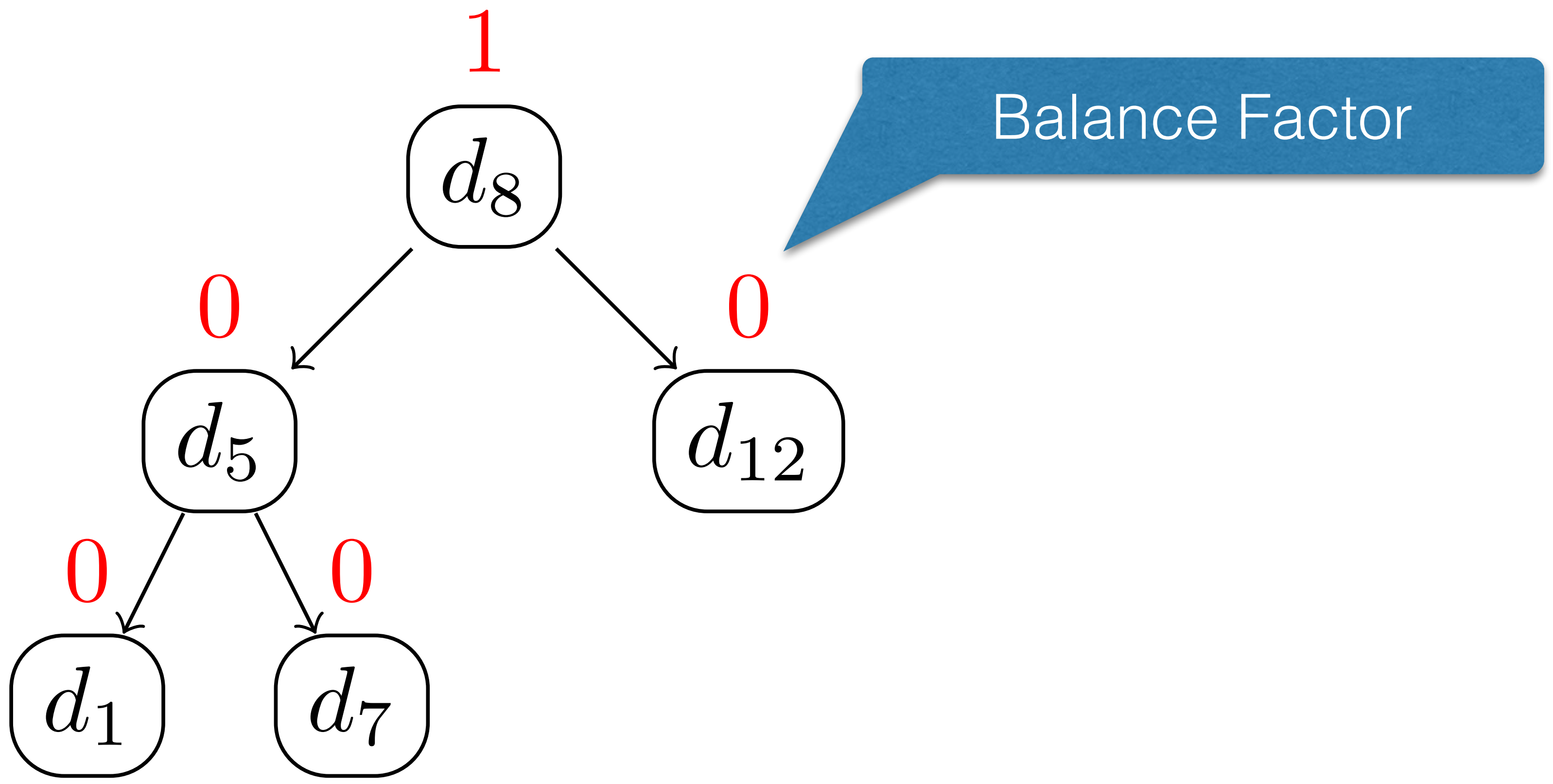
- Based on binary search tree
- Satisfies the AVL property



AVL hash tree

Order $\mathcal{R}$ on data

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree



AVL hash tree

Order $\mathcal{R}$ on data

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

$$\begin{tikzcd}
 & & d_8, 1 \arrow[dl] \arrow[dr] & & \\
 & d_5, 0 \arrow[dl] \arrow[dr] & & d_{12}, 0 & \\
d_1, 0 & & d_7, 0 & &
\end{tikzcd}$$

AVL hash tree

Order $\mathcal{R}$ on data

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

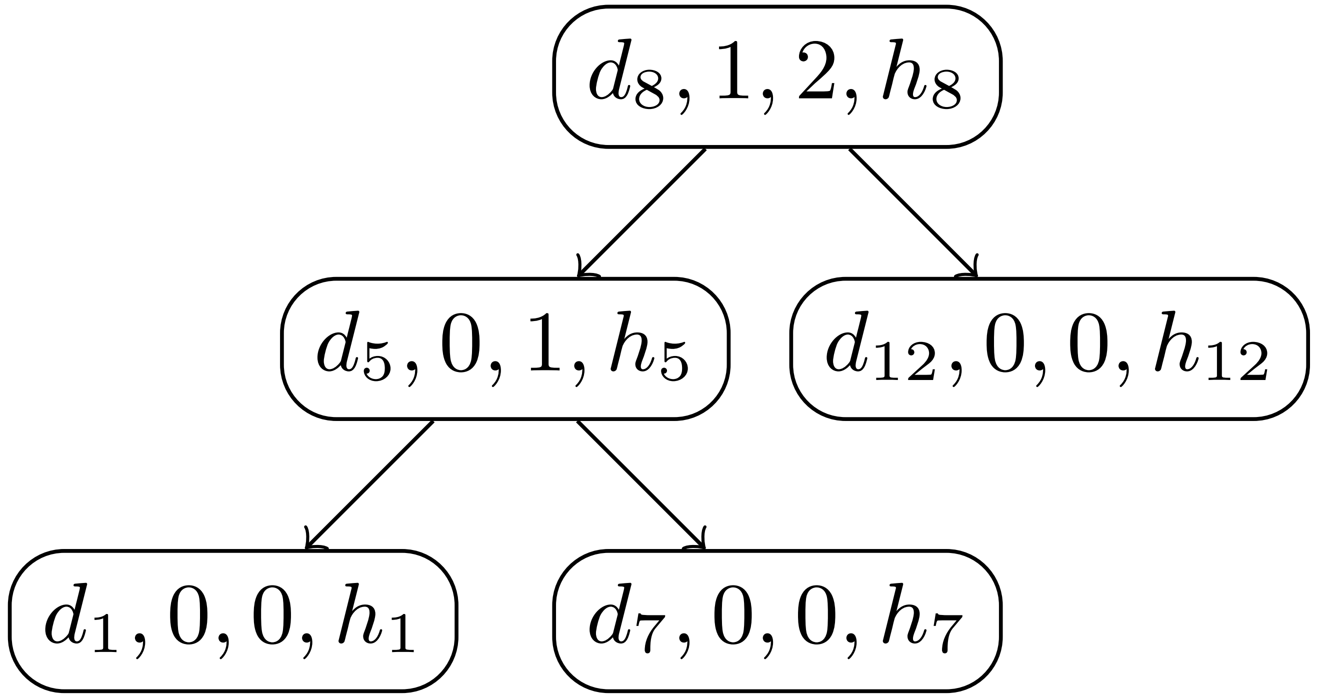
$$\begin{tikzcd}
& & d_8, 1, 2 \arrow[dl] \arrow[dr] & \\
& d_5, 0, 1 \arrow[dl] \arrow[dr] & & d_{12}, 0, 0 \\
d_1, 0, 0 & & d_7, 0, 0 &
\end{tikzcd}$$

Height

AVL hash tree

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

Order $\mathcal{R}$ on data



$$\begin{tikzcd}
 & d_8, 1, 2, h_8 \arrow[dl] \arrow[dr] & \\
d_5, 0, 1, h_5 \arrow[d] \arrow[dr] & & d_{12}, 0, 0, h_{12} \\
d_1, 0, 0, h_1 & d_7, 0, 0, h_7 &
\end{tikzcd}$$

AVL hash tree

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

Order $\mathcal{R}$ on data

$$\begin{tikzcd}
 & & d_8, 1, 2, h_8 \arrow[dl] \arrow[dr] & \\
 & d_5, 0, 1, h_5 \arrow[dl] \arrow[dr] & & d_{12}, 0, 0, h_{12} \\
d_1, 0, 0, h_1 & & d_7, 0, 0, h_7 &
\end{tikzcd}$$

h$(d_1, 0, 0)$

AVL hash tree

Order $\mathcal{R}$ on data

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

$$\begin{tikzcd}
 & & d_8, 1, 2, h_8 \arrow[dl] \arrow[dr] & \\
 & d_5, 0, 1, h_5 \arrow[dl] \arrow[dr] & & d_{12}, 0, 0, h_{12} \\
d_1, 0, 0, h_1 & & d_7, 0, 0, h_7 &
\end{tikzcd}$$

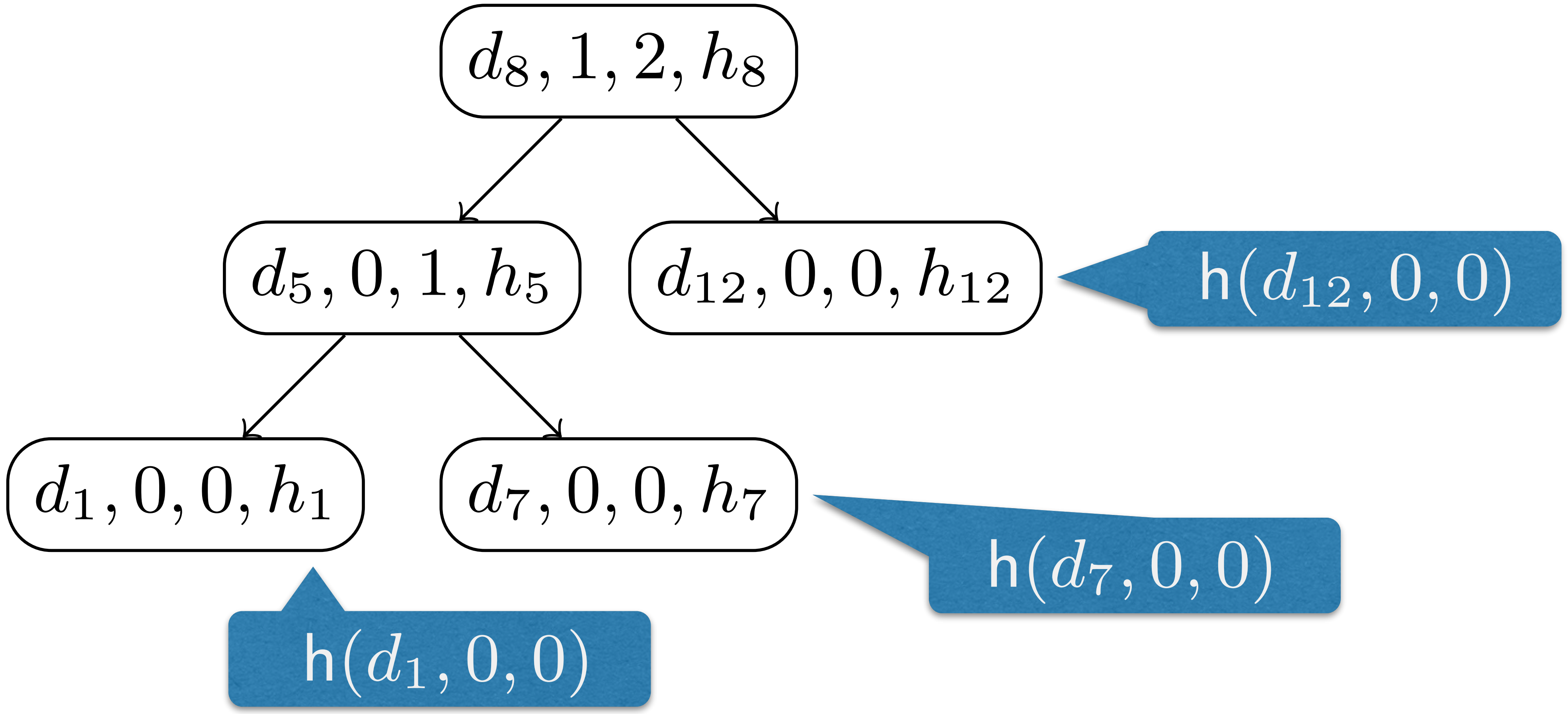
h$(d_7, 0, 0)$

h$(d_1, 0, 0)$

AVL hash tree

Order $\mathcal{R}$ on data

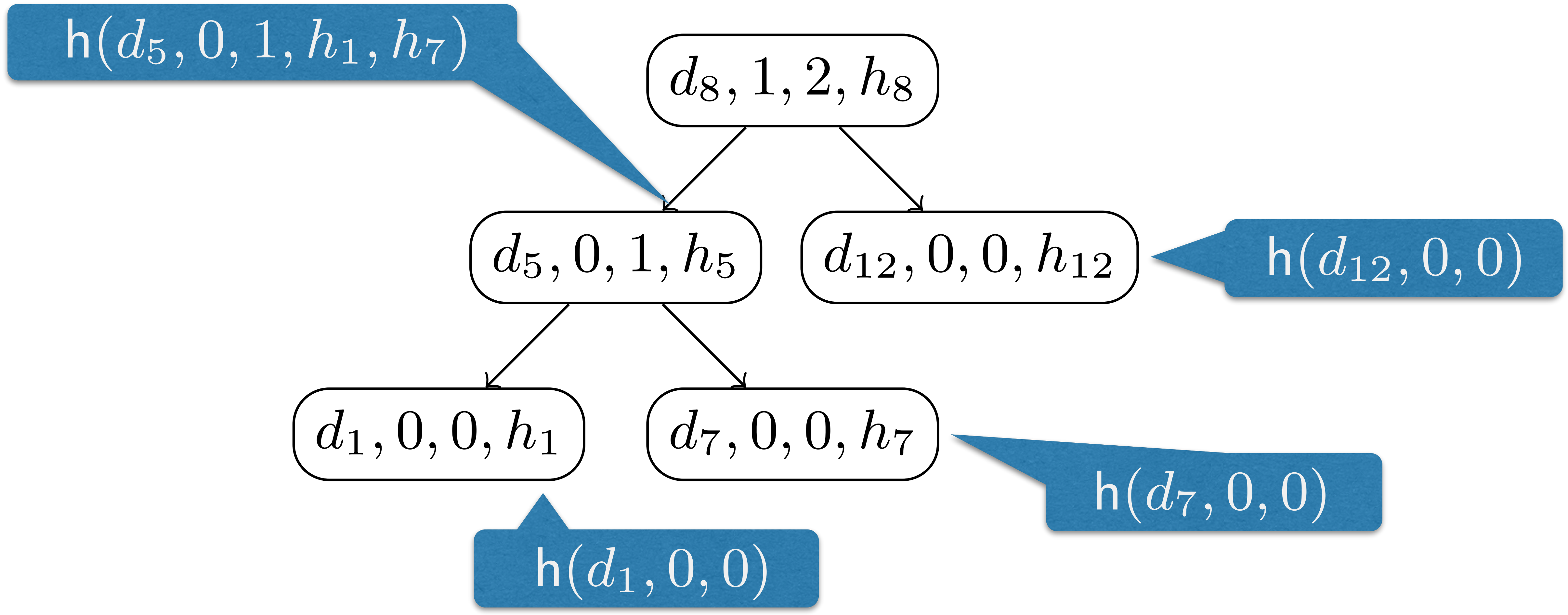
- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree



AVL hash tree

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

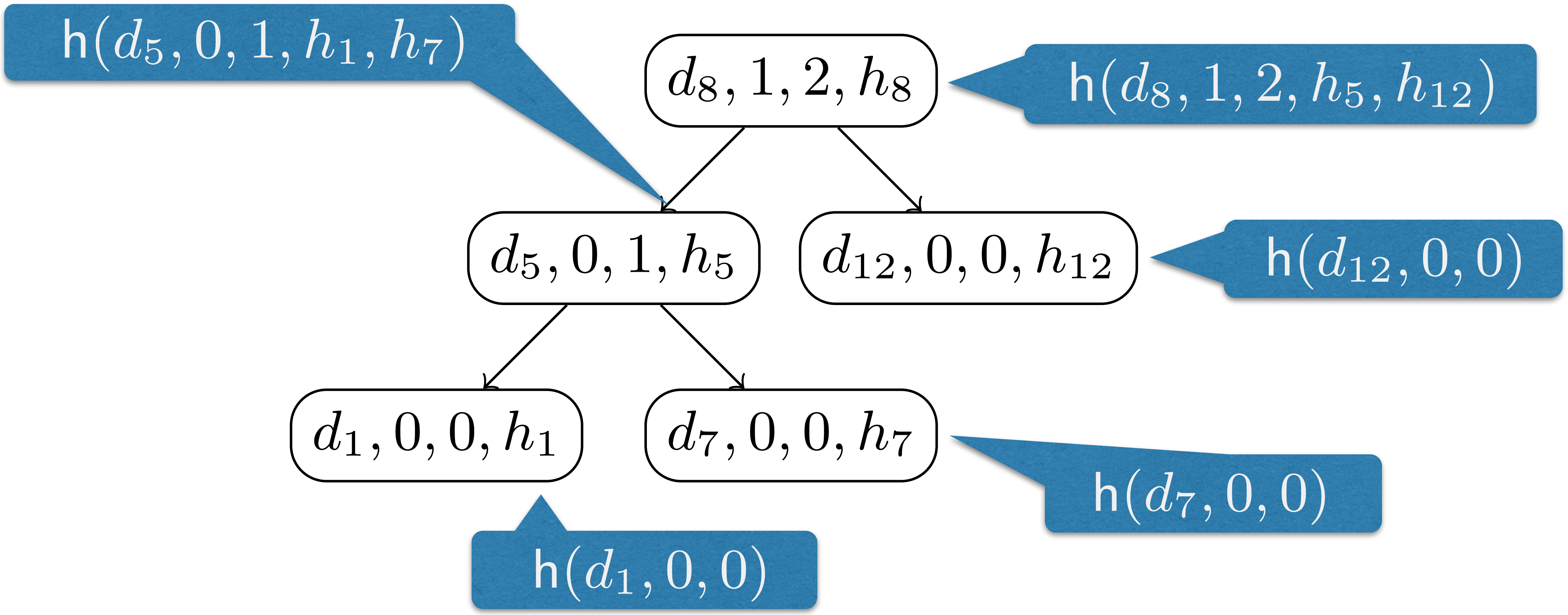
Order $\mathcal{R}$ on data



AVL hash tree

- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree

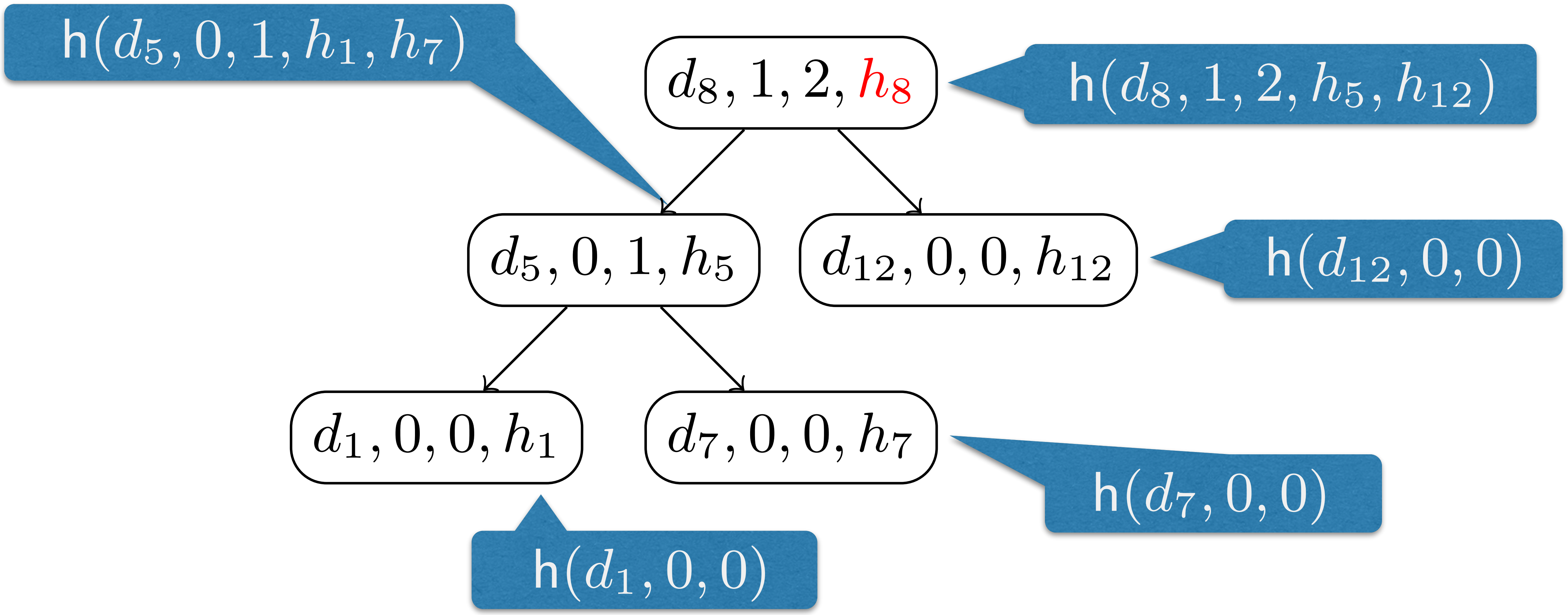
Order $\mathcal{R}$ on data



AVL hash tree

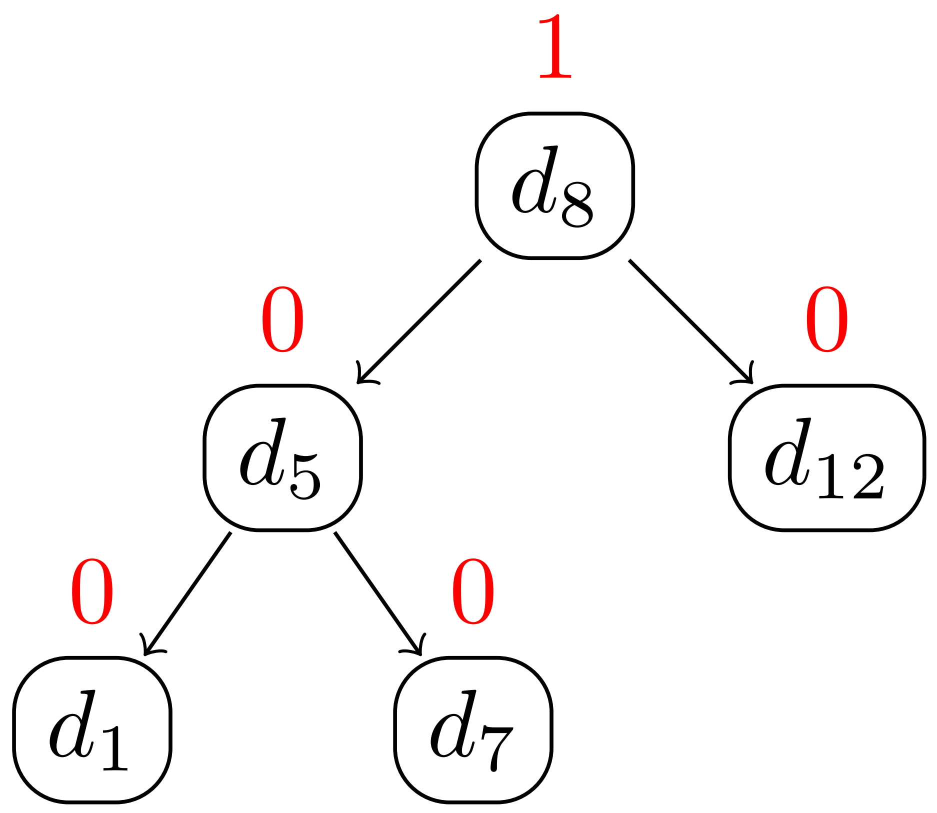
- Based on binary search tree
- Satisfies the AVL property
- Also based on hash tree
- The digest of the data is the hash value of the root

Order $\mathcal{R}$ on data



AVL hash tree

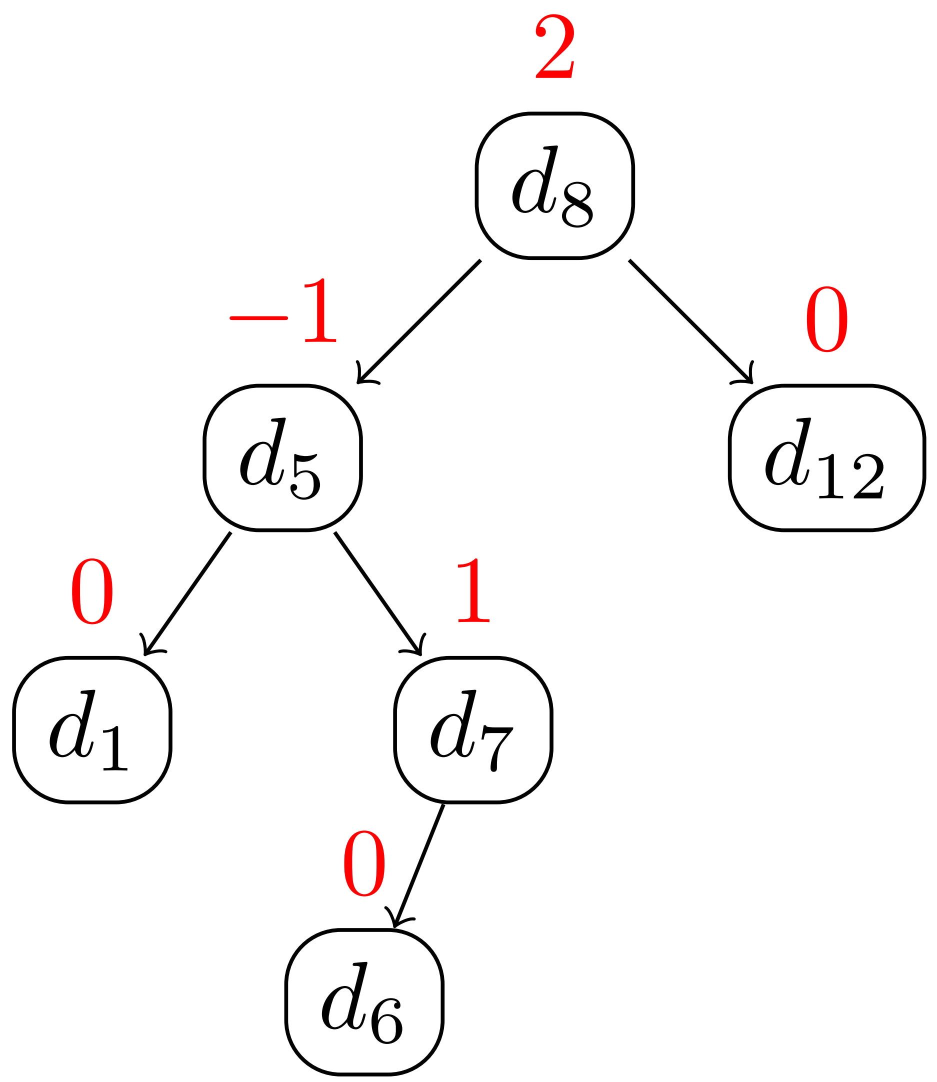
- Addition and deletion similar to AVL tree



AVL hash tree

- Addition and deletion similar to AVL tree

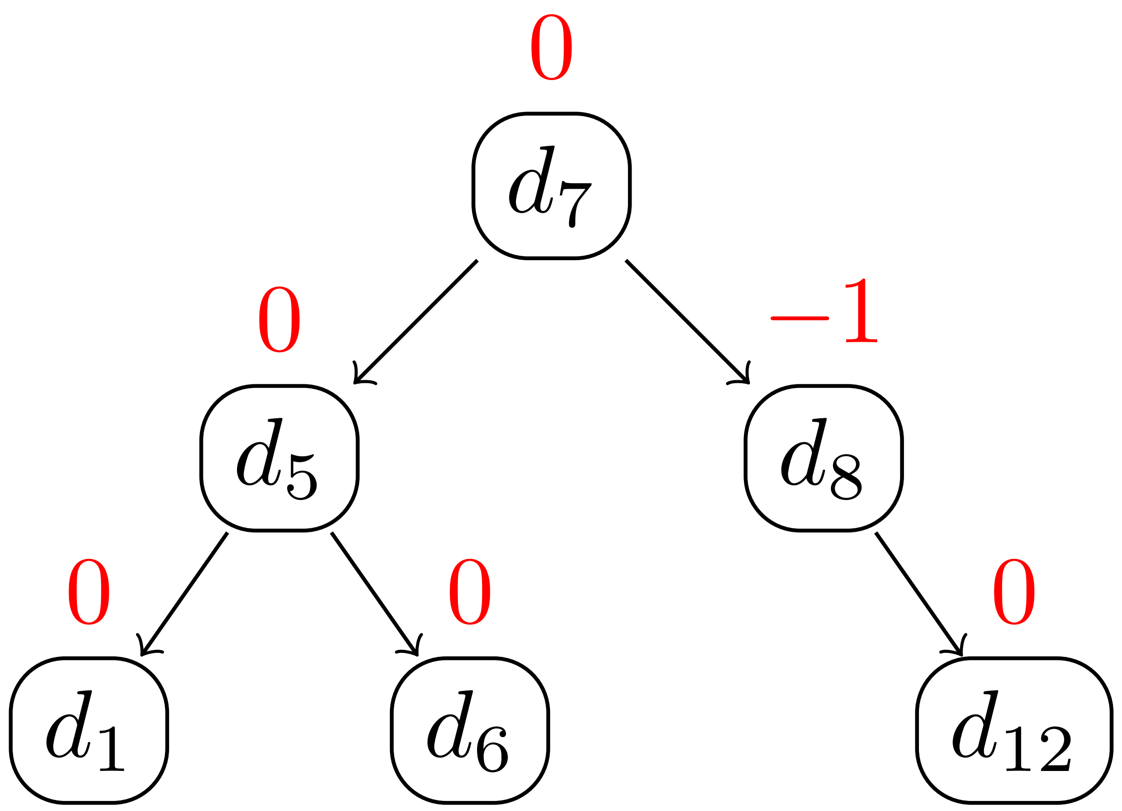
Not AVL



AVL hash tree

- Addition and deletion similar to AVL tree

AVL

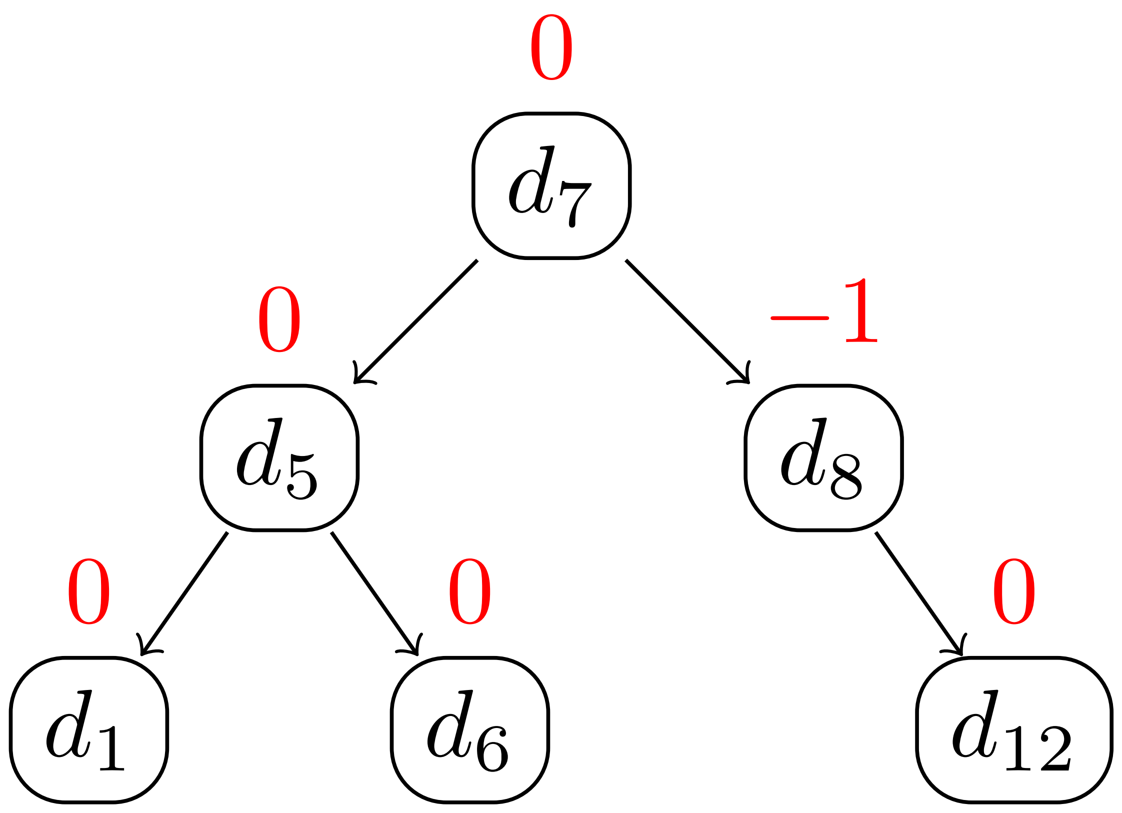


Rotations of subtrees

AVL hash tree

- Addition and deletion similar to AVL tree
- Self balancing tree

AVL

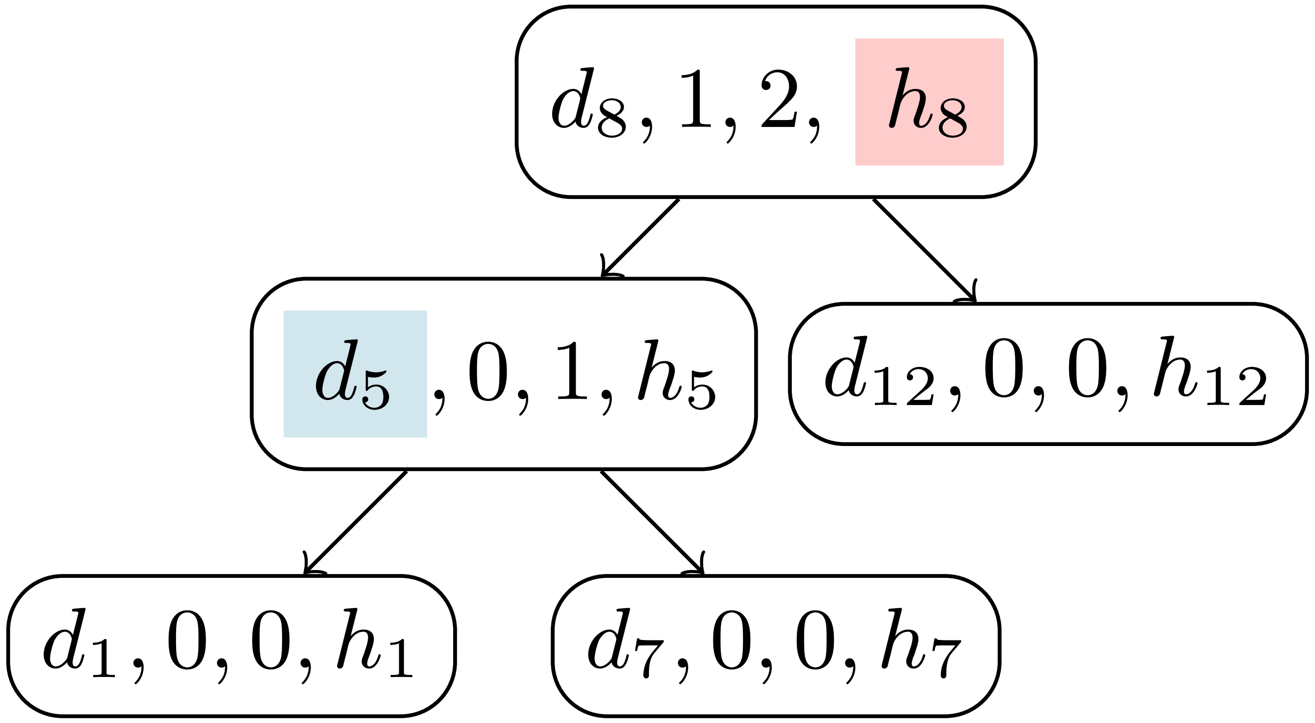


Rotations of subtrees

AVL hash tree

Proof of presence and absence

Proof contains:

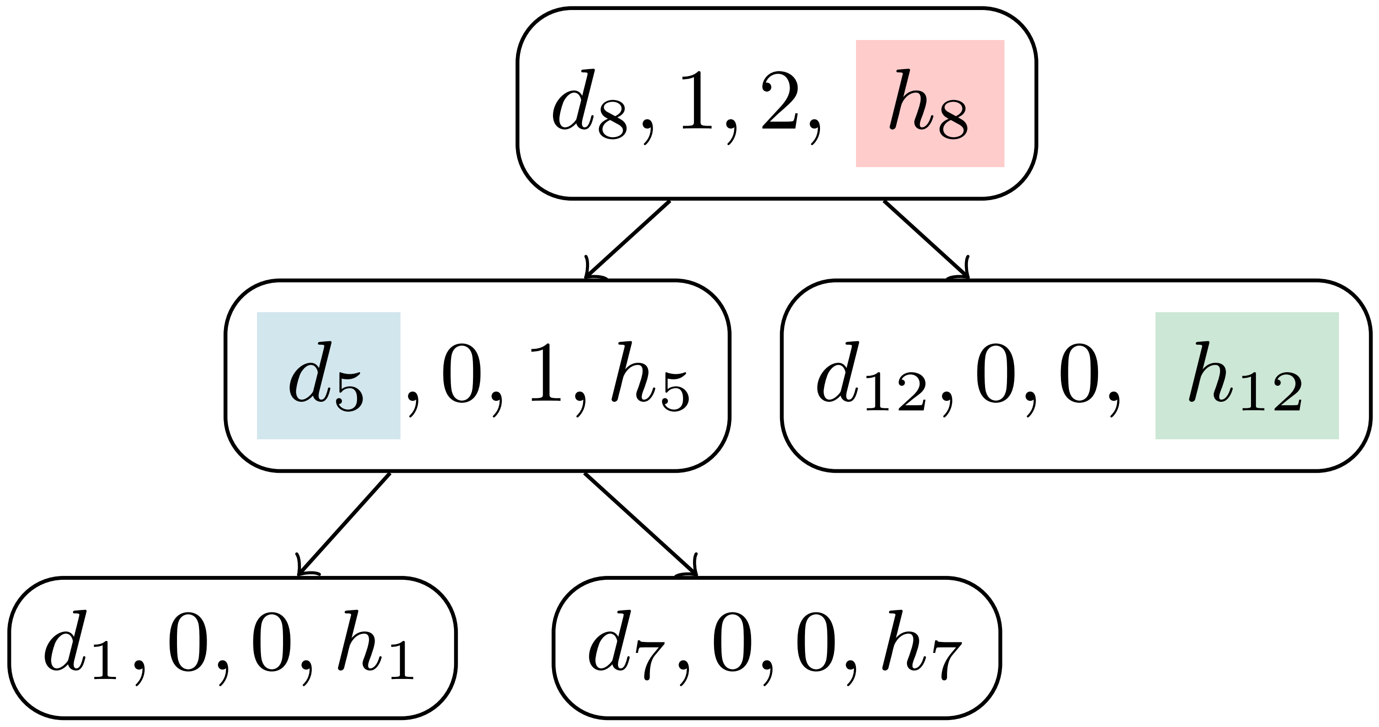


AVL hash tree

Proof of presence and absence

Proof contains:

- hash value of sibling in path

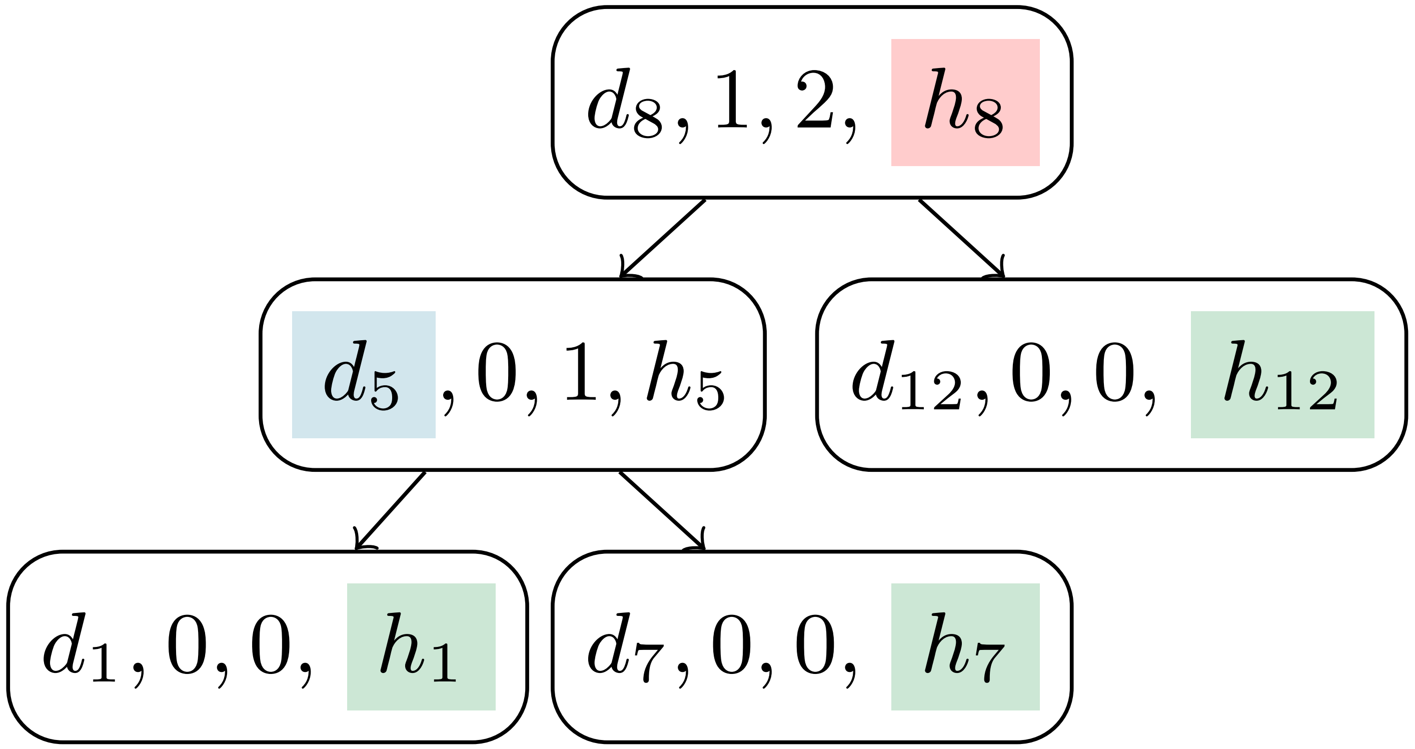


AVL hash tree

Proof of presence and absence

Proof contains:

- hash value of sibling in path
- hash values of children

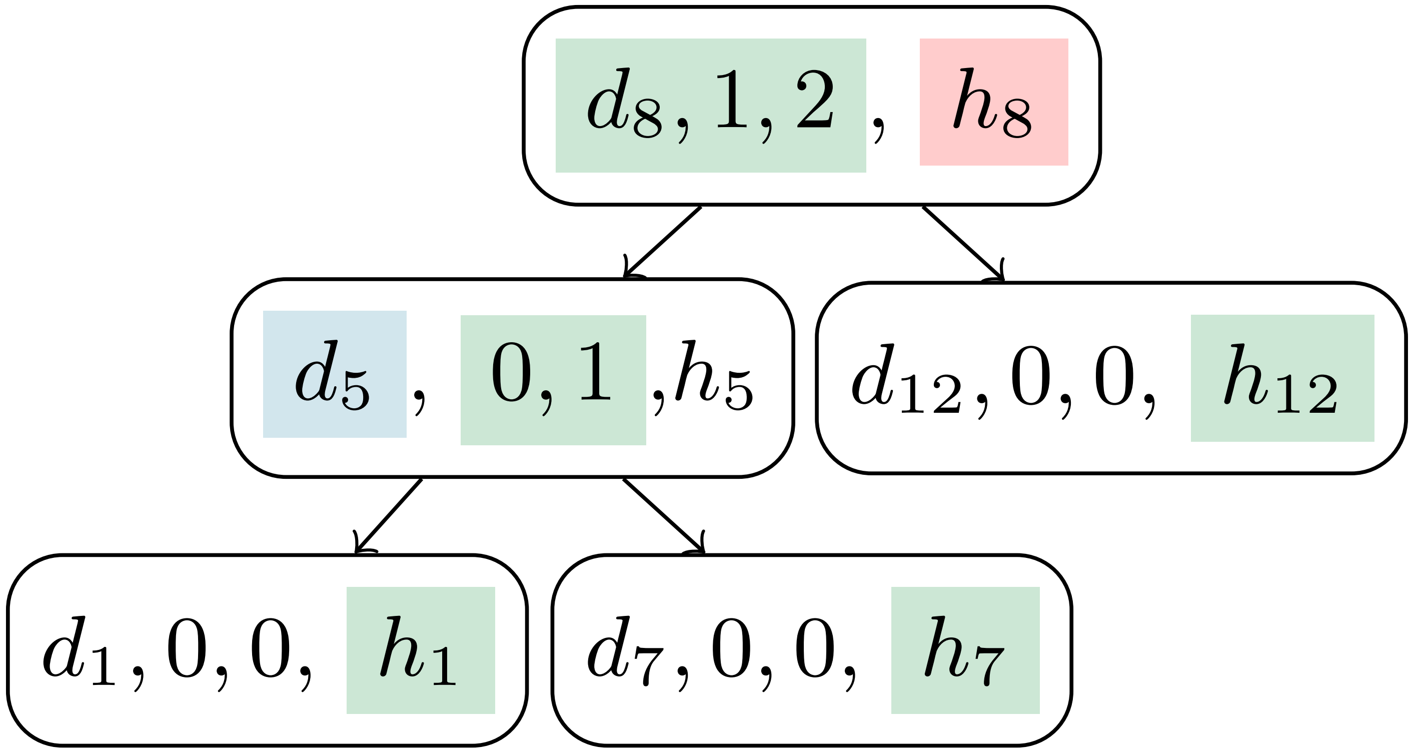


AVL hash tree

Proof of presence and absence

Proof contains:

- hash value of sibling in path
- hash values of children
- data on the path



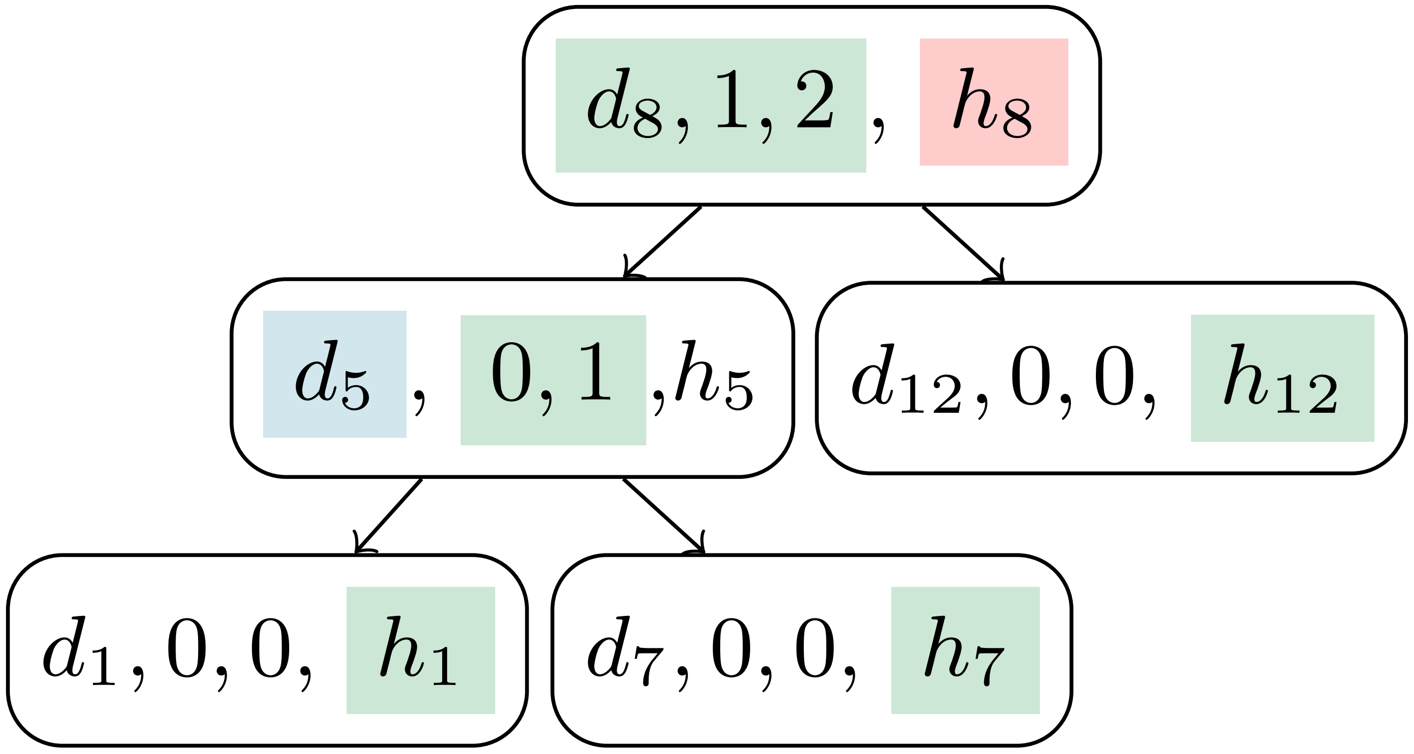
AVL hash tree

Proof of presence and absence

Proof contains:

- hash value of sibling in path
- hash values of children
- data on the path

Proof in O(log(n))



AVL hash tree

Proof of presence and absence

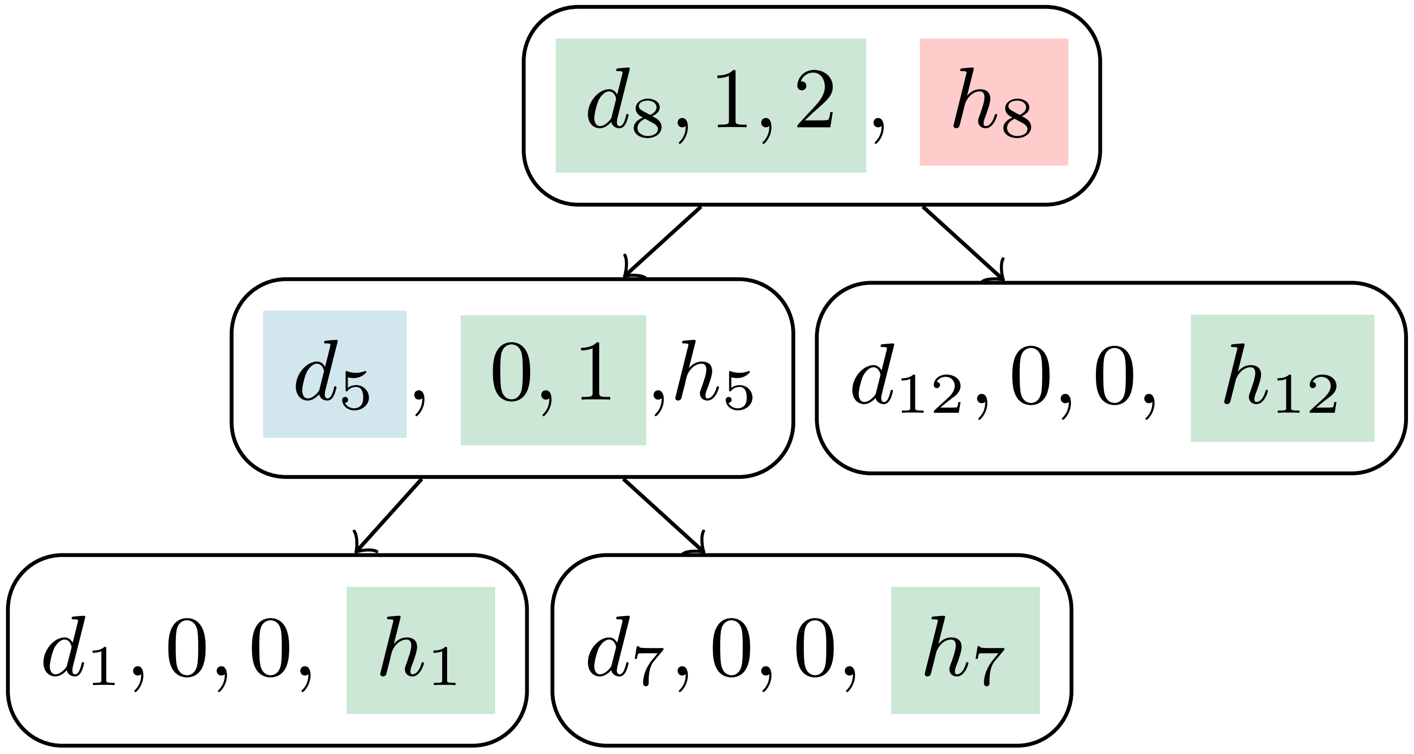
Proof contains:

- hash value of sibling in path
- hash values of children
- data on the path

Proof in O(log(n))

Verification:

- compute hashes
- verify order on data



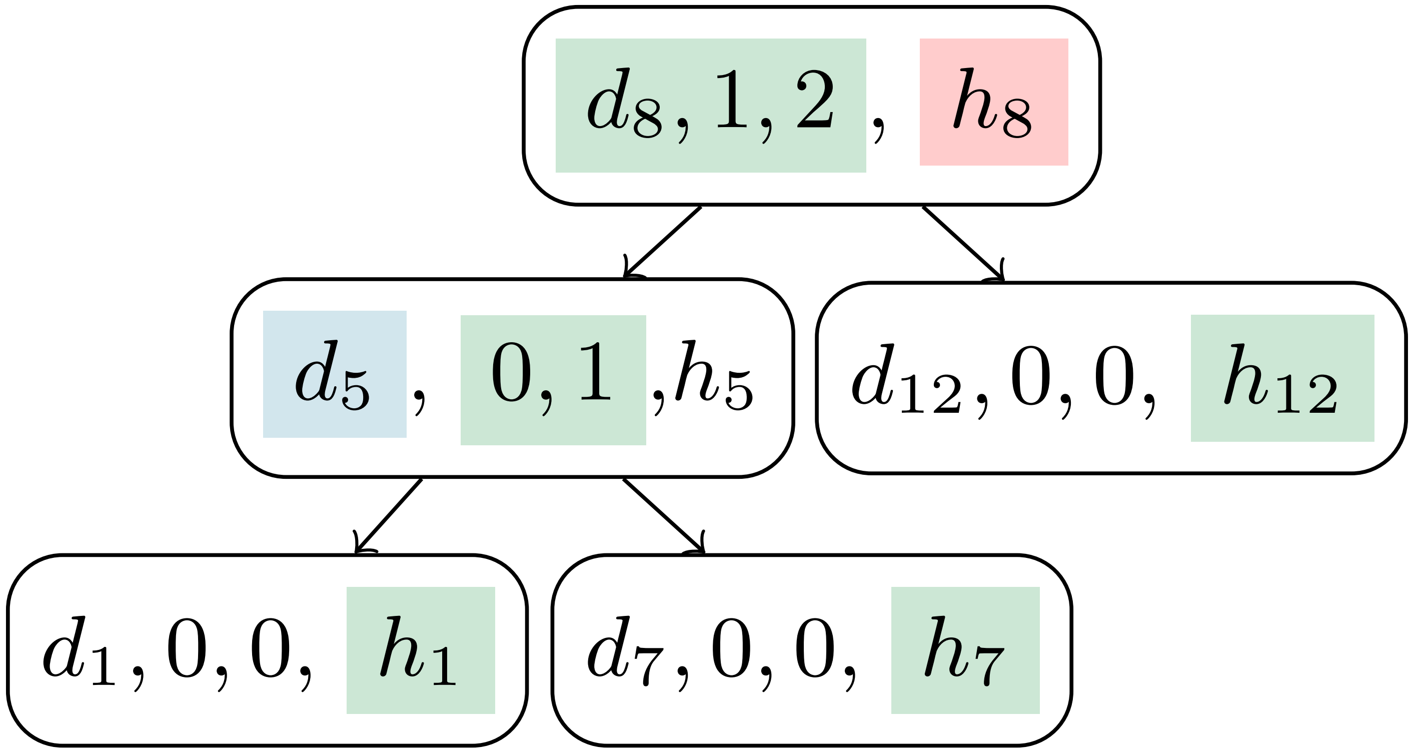
AVL hash tree

Proof of presence and absence

Proof contains:

- hash value of sibling in path
- hash values of children
- data on the path

Proof in O(log(n))



Verification:

- compute hashes
- verify order on data

Verification of the proof in O(log(n))

Certificate log

- **AVL hash tree stores the current state**

  Data: id + certificate

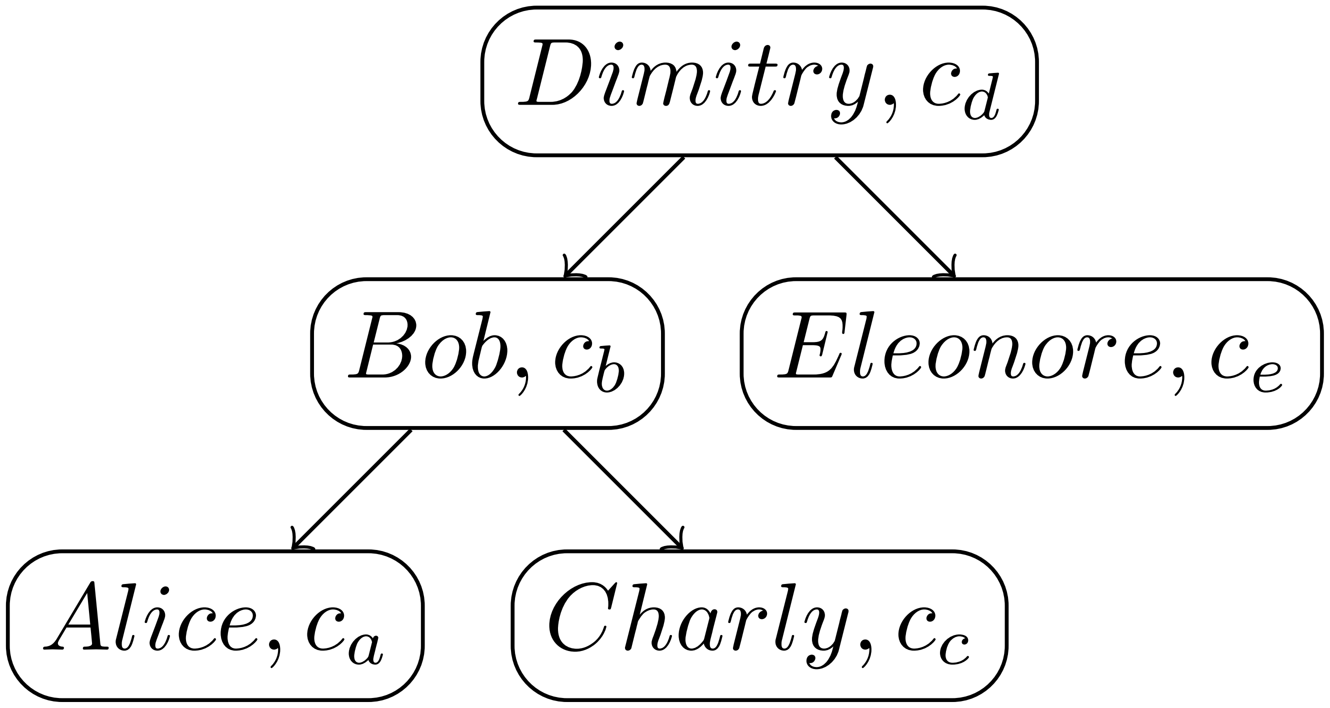
  $\mathcal{R}$: **Total order on id, ignore certificate**

Certificate log

- **AVL hash tree stores the current state**

  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**



$$\begin{tikzcd}
 & & Dimitry, c_d \arrow[dl] \arrow[dr] & \\
 & Bob, c_b \arrow[dl] \arrow[dr] & & Eleonore, c_e \\
Alice, c_a & & Charly, c_c &
\end{tikzcd}$$

Certificate log

- **AVL hash tree stores the current state**

  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

Certificate log

- **AVL hash tree stores the current state**

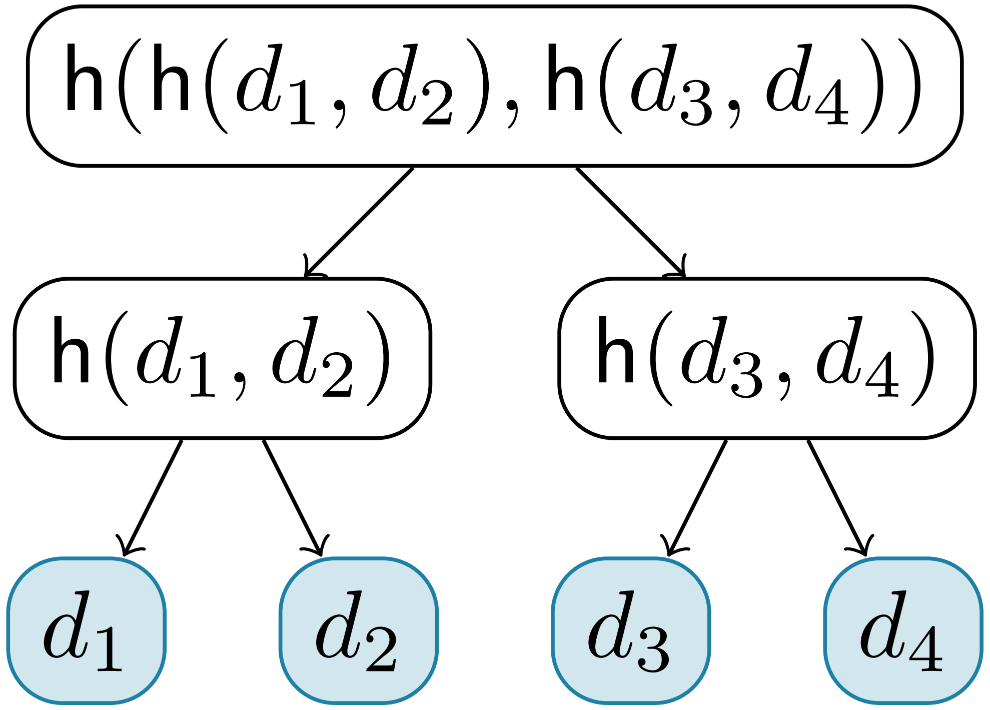
  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$



Certificate log

- **AVL hash tree stores the current state**

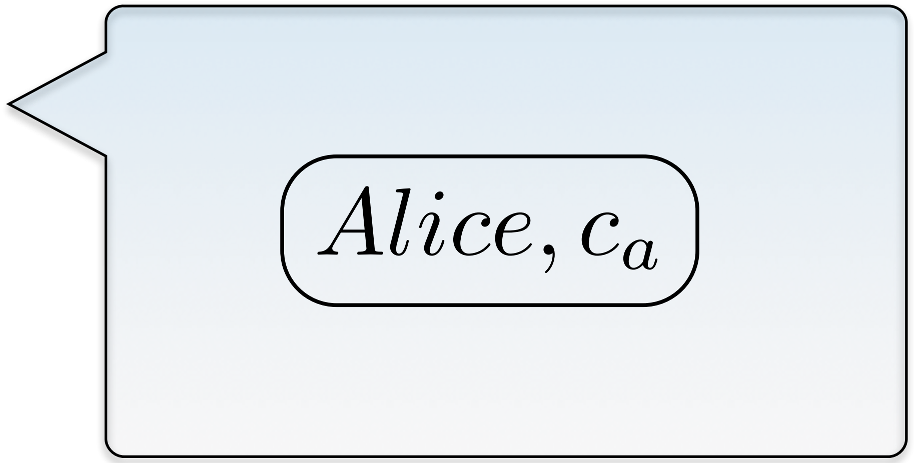
  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$



Certificate log

- **AVL hash tree stores the current state**

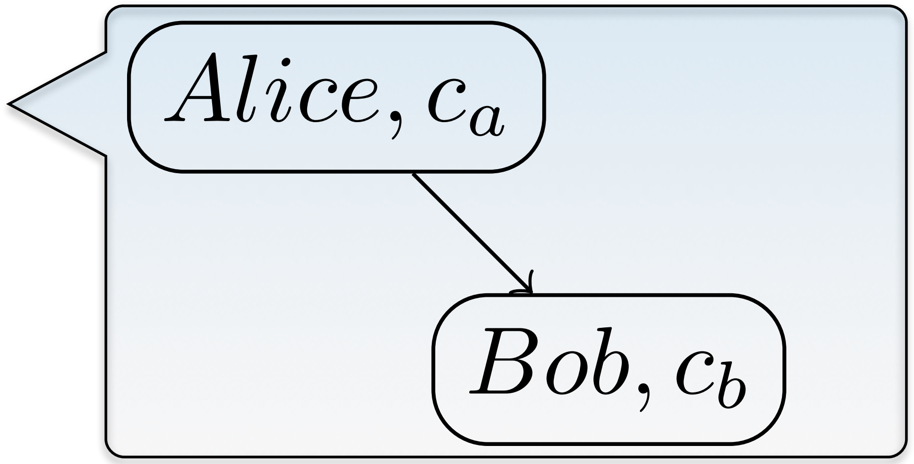
  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$



Certificate log

- **AVL hash tree stores the current state**

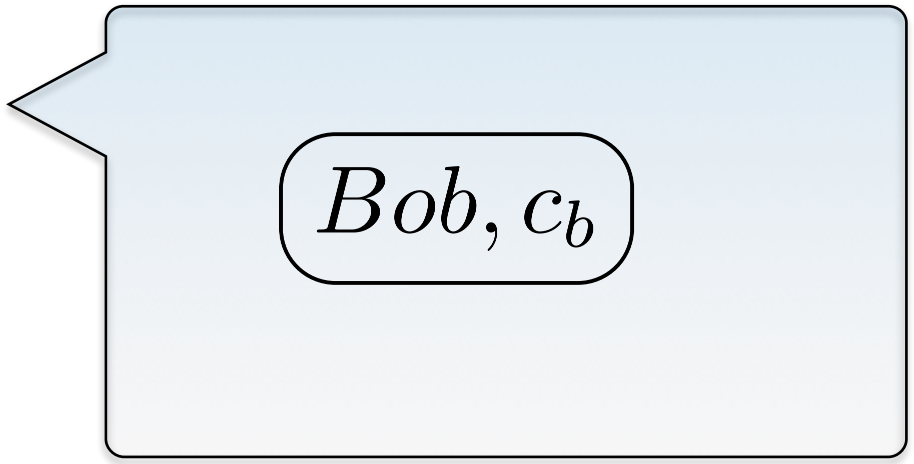
  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

$$
\begin{aligned}
d_1 &= add(Alice, c_a), h_1 \\
d_2 &= add(Bob, c_b), h_2 \\
d_3 &= del(Alice), h_3 \\
d_4 &= add(Alice, c'_a), h_4
\end{aligned}
$$



Certificate log

- **AVL hash tree stores the current state**

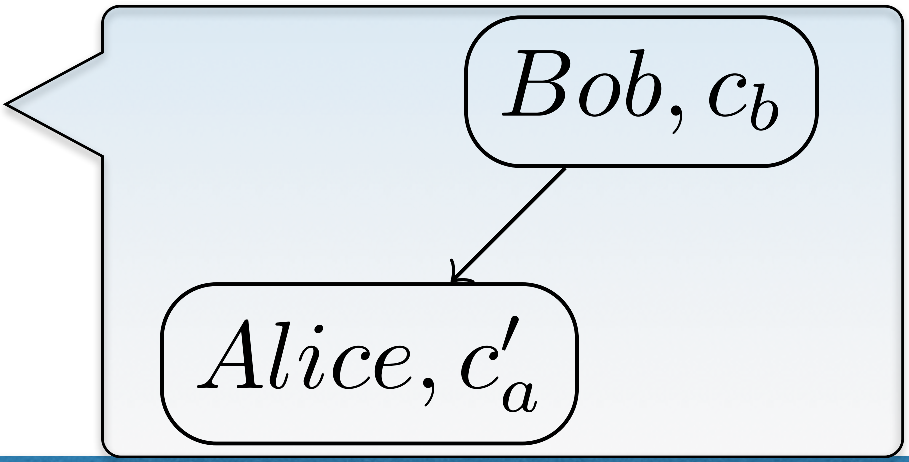
  Data: id + certificate

  $\mathcal{R}$: **Total order on id, ignore certificate**

- **ChronTree stores requests**

  Data: *add(cert), rev(cert)* + digest of AVL hash tree

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$


$$\begin{tikzcd} Bob, c_b \arrow[d] \\ Alice, c'_a \end{tikzcd}$$

Certificate log

Random verification

Certificate log

Random verification

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c_a'), h_4$$

1. Randomly select i
2. Proof of presence of di and d_{i+1}
3. Proof of addition / deletion from the digest of d_i to d_{i+1} depending on the request

Certificate log

Random verification

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$

1. Randomly select i
2. Proof of presence of di and d_{i+1}
3. Proof of addition / deletion from the digest of d_i to d_{i+1} depending on the request

Individual verification is O(log(n)) in time and size

Certificate log

Random verification

$$d_1 = add(Alice, c_a), h_1$$
$$d_2 = add(Bob, c_b), h_2$$
$$d_3 = del(Alice), h_3$$
$$d_4 = add(Alice, c'_a), h_4$$

1. Randomly select i
2. Proof of presence of di and d_{i+1}
3. Proof of addition / deletion from the digest of d_i to d_{i+1} depending on the request

Individual verification is O(log(n)) in time and size

Complete verification is O(n · log(n)) in time and size

Conclusion

DTKI: Distributed Transparent Key Infrastructure

- No trusted party
- Fully transparent
- Secure for multiple public log of certificates
- Revocation