DTKI
A new formalised PKI with no trusted parties

Vincent Cheval, Mark Ryan, Jiangshan Yu

2 April, LORIA, Nancy
Most communications take place over a public network.

It is important to ensure their security.
Context

Asymmetric encryption

M
Asymmetric encryption

I generate a public and private key $sk, pk(sk)$
Asymmetric encryption

I generate a public and private key

I distribute my public key

M

\( \text{pk}(sk) \)

\( sk, \text{pk}(sk) \)
I want to send a message to Bob

Asymmetric encryption

$\text{pk}(\text{sk})$

$\text{sk}, \text{pk}(\text{sk})$
Asymmetric encryption

I want to send a message to Bob

\[ \text{enc}(M, \text{pk}(sk)) \]

\[ \text{pk}(sk) \]

\[ sk, \text{ pk}(sk) \]
Asymmetric encryption

\[ \text{pk}(sk) \]

\[ \text{enc}(M, \text{pk}(sk)) \]

I decrypt with my private key

\[ sk, \text{ pk}(sk) \]
Asymmetric encryption

\[ \text{pk}(sk) \]

\[ \text{enc}(M, \text{pk}(sk)) \]

\[ \text{sk, pk}(sk) \]

SSL / TLS protocol
HTTPS connection
Distribution of the public key

\[ \text{M} \rightarrow \text{pk}(\text{sk}) \]

\[ \text{enc}(\text{M}, \text{pk}(\text{sk})) \]

\[ \text{sk, pk}(\text{sk}) \]
Distribution of the public key

Authenticity of pk(sk) ?

M

pk(sk)

enc(M, pk(sk))

sk, pk(sk)
Distribution of the public key

Authenticity of pk(sk) ?

M

sk, pk(sk)
Distribution of the public key

Authenticity of \( pk(sk) \) ?

\[ M \]

I intercept Bob’s message

\[ sk, pk(sk) \]
Distribution of the public key

Authenticity of \( pk(sk) \) ?

I generate a new set of public and private key

I intercept Bob’s message

\( sk, \ pk(sk) \)

\( sk', \ pk(sk') \)
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…

\[ \text{sk}_{CA}, \text{pk}(\text{sk}_{CA}) \]
Existing solution

Public key certificate: digital identity (standard X.509)
Certificate authority: VeriSign, Comodo, GoDaddy…

\[ \text{sk}_{CA}, \text{pk}(\text{sk}_{CA}) \]

I want to register my public key

\[ \text{sk}, \text{pk}(\text{sk}) \]
Existing solution

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

Certificate authority: VeriSign, Comodo, Go Daddy…

I want to register my public key

$sk, pk(sk)$
Existing solution

Public key certificate: digital identity (standard X.509)
Certificate authority: VeriSign, Comodo, Go Daddy…

I want to register my public key
Existing solution

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

Certificate authority: VeriSign, Comodo, Go Daddy…

\[ M, \text{pk}(sk_{CA}) \]

I want to talk to Bob

\[ \text{sk}_{CA}, \text{pk}(\text{sk}_{CA}) \]

pk(sk)

External verification

\[ \text{sk}, \text{pk}(sk) \]
Existing solution

Public key certificate: digital identity (standard X.509)
Certificate authority: VeriSign, Comodo, Go Daddy…

\[
\text{sign}_{sk_{CA}}(pk(sk), Bob)
\]

\[
M, pk(sk_{CA})
\]
Existing solution

Public key certificate: digital identity (standard X.509)
Certificate authority: VeriSign, Comodo, Go Daddy…

\[
s_{sk_{CA}}(pk(sk), Bob)
\]

Bob

External verification

M, pk(sk_{CA})

I verify the signature

sk, pk(sk)
Existing solution

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

Certificate authority: VeriSign, Comodo, Go Daddy…
Problems with existing solution

\[ \text{sign}_{sk_{CA}}(pk(sk), Bob) \]

\[ \text{sk}_{CA}, pk(sk_{CA}) \]

\[ enc(M, pk(sk)) \]

\[ pk(sk) \]

\[ M, pk(sk_{CA}) \]

\[ sk, pk(sk) \]
Problems with existing solution

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

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

control

\( sk, \ pk(\text{sk}) \)
Problems with existing solution

\[ M, \text{pk}(sk_{CA}) \]

\[ \text{control} \]

\[ \text{sk}, \text{pk}(sk) \]
Problems with existing solution

M, pk(sk_{CA})

veriSign

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

control

sk', pk(sk')

sk, pk(sk)
Problems with existing solution

\[ M, \: \text{pk}(\text{sk}_{\text{CA}}) \]

\[ \text{sk}', \: \text{pk}(\text{sk}') \]

\[ \text{control} \]

\[ \text{pk}(\text{sk}) \]
Problems with existing solution

\[ \text{sign}_{sk_{CA}}(pk(sk'), Bob) \]

\[ M, pk(sk_{CA}) \]

\[ \text{sk'}, pk(sk') \]

\[ \text{control} \]

\[ \text{sk}, pk(sk) \]
Problems with existing solution

\[ \text{sign}_{sk_{CA}}(pk(sk'), Bob) \]

\[ \text{enc}(M, pk(sk')) \]

\[ sk', pk(sk') \]

\[ sk_{CA}, pk(sk_{CA}) \]
Problems with existing solution

Problem 1: Trust given to the Certificate Authority

Real attacks reported: Comodo, DigiNotar, ANSSI
Problems with existing solution

\[ \text{sign}_{sk_{CA}}(pk(sk), Bob) \]

\[ sk_{CA}, pk(sk_{CA}) \]

\[ \text{Bob} \]

\[ \text{enc}(M, pk(sk)) \]

\[ sk, pk(sk) \]
Problems with existing solution

\[ \text{M, } \mathbf{pk}(\mathbf{sk}_\text{CA}) \rightarrow \mathbf{sk}, \mathbf{pk}(\mathbf{sk}) \]

\[ \mathbf{sk}_\text{CA}, \mathbf{pk}(\mathbf{sk}_\text{CA}) \]

Embedded in browser
Problems with existing solution

$M, \text{pk}(sk_{CA}) \quad \text{Embedded in browser} \quad sk, \text{pk}(sk)$
Problems with existing solution

Problem 2: Monopoly of the certificate authority

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

$M, \ pk(sk_{CA})$   Embedded in browser   $sk, \ pk(sk)$
Problems with existing solution

Problem 2: Monopoly of the certificate authority

M, \textbf{pk}(\textbf{sk}_{CA}) \quad \textbf{pk}(\textbf{sk}_{CA}) \quad \textbf{sk}, \textbf{pk}(\textbf{sk})

Embedded in browser

No link between CAs

> 100 in Firefox
Problems with existing solution

Problem 2: Monopoly of the certificate authority

Problem 3: Coarse grain security

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

No link between CAs

$M, pk(sk_{CA})$  Embedded in browser  $sk, pk(sk)$
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

\[ pk(s_{\text{log}}), d_{\text{log}} \]

\[ s_{\text{log}}, pk(s_{\text{log}}) \]
Structure of Public log

Data structure

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

\[ \text{pk}(\text{sk}_{\log}), d_{\log} \]

\[ \text{sk}_{\log}, \text{pk}(\text{sk}_{\log}) \]
Structure of Public log

Data structure

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

\[ pk(\text{sk}_{log}), d_{log} \]

\[ \text{Bob, } d_{log} \]

\[ \text{cert}_{Bob}, d'_{log} \]

\[ \text{sk}_{log}, pk(\text{sk}_{log}) \]
Structure of Public log

Data structure

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

\[ \text{pk}(\text{sk}_{\text{log}}), d_{\text{log}} \]

\[ \text{Bob}, d_{\text{log}} \]

\[ \text{cert}_{\text{Bob}}, d_{\log}' \]

\[ \text{proof}_p(\text{cert}_{\text{Bob}}, d_{\log}') \]

\[ \text{sk}_{\text{log}}, \text{pk}(\text{sk}_{\text{log}}) \]
Structure of Public log

Data structure

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

\[ \text{pk}(sk_{\text{log}}), d_{\text{log}} \]

\[ \text{Bob, } d_{\text{log}} \]

\[ \text{cert}_{\text{Bob}}, d_{\text{log}'} \]

\[ \text{proof}_p(\text{cert}_{\text{Bob}}, d_{\text{log}'} ) \]

\[ \text{proof}_e(d_{\text{log}}, d_{\text{log}'} ) \]

\[ \text{sk}_{\text{log}}, \text{pk}(sk_{\text{log}}) \]
Structure of Public log

Data structure

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

\[ \text{pk}(\text{sk}_{\text{log}}), d_{\text{log}} \]

Bob, \( d_{\text{log}} \)

\[ \text{cert}_{\text{Bob}}, d_{\text{log}}' \]

\[ \text{proof}_p(\text{cert}_{\text{Bob}}, d_{\text{log}}') \]

\[ \text{proof}_e(d_{\text{log}}, d_{\text{log}}') \]

\[ \text{sk}_{\text{log}}, \text{pk}(\text{sk}_{\text{log}}) \]

I verify \( \text{proof}_p(\text{cert}_{\text{Bob}}, d_{\text{log}}') \) and \( \text{proof}_e(d_{\text{log}}, d_{\text{log}}') \)
Structure of Public log

Data structure

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

I verify \( \text{proof}_p(\text{cert}_{\text{Bob}}, \text{d}_{\log'}) \) and \( \text{proof}_e(\text{d}_{\log}, \text{d}_{\log'}) \)
Structure of Public log

Data structure

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

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

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

$$\text{proof}_p(\text{cert}_{\text{Bob}}, d_{\log})$$

$$\text{proof}_e(d_{\log}, d_{\log}’)$$

$$\text{sk}_{\log}, \text{pk}(\text{sk}_{\log})$$

I verify $$\text{proof}_p(\text{cert}_{\text{Bob}}, d_{\log})$$ and $$\text{proof}_e(d_{\log}, d_{\log})$$

Size and verification time of proofs must be $$O(\log(n))$$
• Based on a binary hash tree
ChronTree

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

\[ h(h(d_1, d_2), d_3) \]

\[ h(d_1, d_2) \]

\[ d_1 \]

\[ d_2 \]

\[ d_3 \]
ChronTree

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

\[
    h(h(d_1, d_2), d_3)
\]
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

\[ h(h(d_1, d_2), d_3) \]

\[ h(d_1, d_2) \]

\[ d_1 \]

\[ d_2 \]

\[ d_3 \]
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*

\[
h(d_1, d_2)
\]

\[
d_1 \quad d_2 \quad d_3 \quad d_4
\]
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*

$$d_1, d_2, d_3, d_4, d_5$$

$$\begin{align*}
    h(d_1, d_2) & \quad h(d_3, d_4) \\
    h(h(d_1, d_2), h(d_3, d_4)) & \\
    d_5 &
\end{align*}$$
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*

\[
\text{h(h(h(d_1, d_2), h(d_3, d_4)), d_5)}
\]
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*

\[
\begin{align*}
    h(h(d_1, d_2), h(d_3, d_4)) \\
    h(d_1, d_2) & \quad h(d_3, d_4) \\
    d_1 & \quad d_2 & \quad d_3 & \quad d_4 \\
    d_5 & \quad d_6
\end{align*}
\]
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*

```
h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6))
```

```
\text{h}(\text{h}(d_1, d_2), \text{h}(d_3, d_4))
```

```
\text{h}(d_5, d_6)
```

```
d_1 \quad d_2 \\
\downarrow \quad \downarrow \\
d_3 \quad d_4 \\
```

```
d_5 \quad d_6
```

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*

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

Height in $O(\log(n))$
ChronTree

Proof of presence of some data in the digest

$h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6))$
ChronTree

Proof of presence of some data in the digest

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

- \( h(h(d_1, d_2), h(d_3, d_4)) \)
  - \( h(d_1, d_2) \)
  - \( d_1 \)
  - \( d_2 \)
  - \( h(d_3, d_4) \)
  - \( d_3 \)
  - \( d_4 \)
- \( h(d_5, d_6) \)
  - \( d_5 \)
  - \( d_6 \)
ChronTree

Proof of presence of some data in the digest

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

Proof in \( O(\log(n)) \)
Proof of presence of some data in the digest

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

Proof in $O(\log(n))$
ChronTree

Proof of presence of some data in the digest

Proof in $O(\log(n))$
Proof of presence of some data in the digest

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

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

Proof in \(O(\log(n))\)
Proof of extension between two digests
Proof of extension between two digests

\[ h(h(d_1, d_2), d_3) \]

\[ h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]
Proof of extension between two digests

Proof in $O(\log(n))$
ChronTree

Proof of extension between two digests

Proof in $O(\log(n))$
Proof of extension between two digests

\[ h(h(d_1, d_2), d_3), h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6)) \]

Proof in \( O(\log(n)) \)
ChronTree

Proof of extension between two digests

$h(h(h(d_1, d_2), d_3), h(d_5, d_6))$

$h(h(h(d_1, d_2), h(d_3, d_4)), h(d_5, d_6))$

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?

$$\text{pk}(\text{sk}_{\text{log}}, d_{\text{log}}) \xrightarrow{\text{Bob, } d_{\text{log}}} \text{cert}_{\text{Bob, } d_{\text{log}'}} \xleftarrow{\text{proof}_{e}(d_{\text{log}}, d_{\text{log}'})} \text{sk}_{\text{log}}, \text{pk}(\text{sk}_{\text{log}})$$
ChronTree

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

Why proof of extension and not addition?

\[ \text{Unbounded number of certificates added between } d_{\log} \text{ and } d_{\log}' \]

\[ \text{proof}_{e}(d_{\log}, d_{\log}') \]

\[ \text{cert}_{\text{Bob}}, d_{\log}' \]

\[ \text{Bob, } d_{\log} \]

\[ \text{pk}(\text{sk}_{\log}), d_{\log} \]

\[ \text{sk}_{\log}, \text{pk}(\text{sk}_{\log}) \]
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
AVL hash tree

- Based on binary search tree
- Satisfies the AVL property

Order $\mathcal{R}$ on data
AVL hash tree

- Based on binary search tree
- Satisfies the AVL property

Order $\mathcal{R}$ on data

Balance Factor
AVL hash tree

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

Order $\mathcal{R}$ on data

Balance Factor
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
AVL hash tree

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

Order $\mathcal{R}$ on data

```
\begin{array}{c}
\text{d_8, 1, 2, h_8} \\
\text{d_5, 0, 1, h_5} & \text{d_12, 0, 0, h_12} \\
\text{d_1, 0, 0, h_1} & \text{d_7, 0, 0, h_7} \\
\end{array}
```
AVL hash tree

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

Order $\mathcal{R}$ on data

$$h(d_1, 0, 0)$$
AVL hash tree

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

Order $\mathcal{R}$ on data

$h(d_1, 0, 0)$

$h(d_7, 0, 0)$
AVL hash tree

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

Order $\mathcal{R}$ on data

```
<table>
<thead>
<tr>
<th>Data (d, l, r, h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d8, 1, 2, h8</td>
</tr>
<tr>
<td>d5, 0, 1, h5</td>
</tr>
<tr>
<td>d1, 0, 0, h1</td>
</tr>
<tr>
<td>d7, 0, 0, h7</td>
</tr>
<tr>
<td>d12, 0, 0, h12</td>
</tr>
</tbody>
</table>
```

- $h(d_{12}, 0, 0)$
- $h(d_{7}, 0, 0)$
- $h(d_{1}, 0, 0)$
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

$h(d_5, 0, 1, h_1, h_7)$

$h(d_8, 1, 2, h_8)$

$h(d_1, 0, 0, h_1)$

$h(d_7, 0, 0, h_7)$

$h(d_{12}, 0, 0, h_{12})$

$h(d_{12}, 0, 0)$

$h(d_1, 0, 0)$

$h(d_7, 0, 0)$
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

Rotations of subtrees
AVL hash tree

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

Rotations of subtrees
Proof of presence and absence

Proof contains:

- $d_8, 1, 2, h_8$
- $d_5, 0, 1, h_5$
- $d_{12}, 0, 0, h_{12}$
- $d_1, 0, 0, h_1$
- $d_7, 0, 0, h_7$
Proof of presence and absence

Proof contains:
• hash value of sibling in path
Proof of presence and absence

Proof contains:
- hash value of sibling in path
- hash values of children

AVL hash tree diagram:

- $d_{8}, 1, 2, h_{8}$
- $d_{5}, 0, 1, h_{5}$
- $d_{12}, 0, 0, h_{12}$
- $d_{1}, 0, 0, h_{1}$
- $d_{7}, 0, 0, h_{7}$
Proof of presence and absence

Proof contains:

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

- $d_8, 1, 2, h_8$
- $d_5, 0, 1, h_5$
- $d_{12}, 0, 0, h_{12}$
- $d_1, 0, 0, h_1$
- $d_7, 0, 0, h_7$
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
Proof of presence and absence

Proof contains:
• hash value of sibling in path
• hash values of children
• data on the path

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
  \( R: \text{Total order on id, ignore certificate} \)
Certificate log

- AVL hash tree stores the current state

Data: id + certificate

**R:** Total order on id, ignore certificate

![AVL hash tree diagram]

- **Dimitry, c_d**
- **Bob, c_b**
- **Eleonore, c_e**
- **Alice, c_a**
- **Charly, c_c**
Certificate log

• AVL hash tree stores the current state
  Data: id + certificate
  \[R: \text{Total order on id, ignore certificate}\]

• 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
  R: Total order on id, ignore certificate

• ChronTree stores requests
  Data: add(cert), rev(cert) + digest of AVL hash tree

\[
\begin{align*}
  d_1 &= \text{add}(Alice, c_a), h_1 \\
  d_2 &= \text{add}(Bob, c_b), h_2 \\
  d_3 &= \text{del}(Alice), h_3 \\
  d_4 &= \text{add}(Alice, c'_a), h_4
\end{align*}
\]
Certificate log

• **AVL hash tree stores the current state**
  
  Data: id + certificate
  
  $R$: **Total order on id, ignore certificate**

• **ChronTree stores requests**

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

\[
\begin{align*}
  d_1 &= add(Alice, c_a), h_1 \\
  d_2 &= add(Bob, c_b), h_2 \\
  d_3 &= del(Alice), h_3 \\
  d_4 &= add(Alice, c'_a), h_4
\end{align*}
\]
Certificate log

• AVL hash tree stores the current state
  Data: id + certificate
  \( R: \text{Total order on id, ignore certificate} \)

• ChronTree stores requests
  Data: \( \text{add(cert)}, \text{rev(cert)} \) + digest of AVL hash tree

\[
\begin{align*}
  d_1 &= \text{add}(Alice, c_a), h_1 \\
  d_2 &= \text{add}(Bob, c_b), h_2 \\
  d_3 &= \text{del}(Alice), h_3 \\
  d_4 &= \text{add}(Alice, c'_a), h_4
\end{align*}
\]
Certificate log

- **AVL hash tree stores the current state**
  
  Data: id + certificate
  
  \( R: \text{Total order on id, ignore certificate} \)

- **ChronTree stores requests**
  
  Data: \( add(cert) \), \( rev(cert) \) + digest of AVL hash tree

\[
\begin{align*}
  d_1 &= add(Alice, c_a), h_1 \\
  d_2 &= add(Bob, c_b), h_2 \\
  d_3 &= del(Alice), h_3 \\
  d_4 &= add(Alice, c'_a), h_4
\end{align*}
\]
Certificate log

• **AVL hash tree stores the current state**
  
  Data: id + certificate

  \[ R: \text{Total order on id, ignore certificate} \]

• **ChronTree stores requests**
  
  Data: \text{add(cert), rev(cert)} + digest of AVL hash tree

\[
\begin{align*}
  d_1 &= \text{add}(Alice, c_a), h_1 \\
  d_2 &= \text{add}(Bob, c_b), h_2 \\
  d_3 &= \text{del}(Alice), h_3 \\
  d_4 &= \text{add}(Alice, c'_a), h_4
\end{align*}
\]
Certificate log

Random verification
Certificate log

Random verification

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

\[ d_1 = \text{add}(Alice, c_a), h_1 \]
\[ d_2 = \text{add}(Bob, c_b), h_2 \]
\[ d_3 = \text{del}(Alice), h_3 \]
\[ d_4 = \text{add}(Alice, c'_a), h_4 \]
Certificate log

Random verification

\[ d_1 = \text{add}(\text{Alice}, c_a), h_1 \]
\[ d_2 = \text{add}(\text{Bob}, c_b), h_2 \]
\[ d_3 = \text{del}(\text{Alice}), h_3 \]
\[ d_4 = \text{add}(\text{Alice}, c'_a), h_4 \]

1. Randomly select \( i \)
2. Proof of presence of \( d_i \) and \( d_{i+1} \)
3. Proof of addition / deletion from the digest of \( d_i \) to \( d_{i+1} \)

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

Random verification

\[ d_1 = \text{add}(Alice, c_a), h_1 \]
\[ d_2 = \text{add}(Bob, c_b), h_2 \]
\[ d_3 = \text{del}(Alice), h_3 \]
\[ d_4 = \text{add}(Alice, c'_a), h_4 \]

1. Randomly select \( i \)
2. Proof of presence of \( d_i \) 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 \( O(n \cdot \log(n)) \) in time and size
Conclusion

DTKI: Distributed Transparent Key Infrastructure

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