Revoke and Let Live

A Secure Key Revocation API for Cryptographic Devices

Véronique Cortier  Graham Steel  Cyrille Wiedling
LORIA-CNRS, Nancy (FR)  INRIA, Paris (FR)  LORIA-CNRS, Nancy (FR)

Séminaire Méthodes Formelles et Sécurité
Rennes, May 24th, 2013

Funded by
Goal: Enforce security of data stored inside the trusted device, even when connected to untrusted host machines.
Applications

• Smartphones,

• Online Banking, Asynchronous Transfer Mode,

• Electronic Ticketing Systems,

• Vehicle-to-vehicle networking.

• ...
How does it work?

<table>
<thead>
<tr>
<th>Host machine</th>
<th>Trusted device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$h_1$

$h_2$
How does it work?

Host machine  Trusted device

export, $h_1$, $h_2$

$h_1$

$h_2$
How does it work?

Host machine

Trusted device

export, $h_1$, $h_2$
How does it work?

Host machine

Export, $h_1, h_2$

Import, $\{ \text{keys} \}, h_2$

Trusted device

$h_1$

$h_2$
How does it work?

Host machine

Trusted device

export, $h_1, h_2$

import, $\{h_1, h_2\}$, $h_2$

$h_1$

$h_2$

$h_3$
How does it work?

Host machine

trusted device

export, $h_1, h_2$

import, $h_3$

$h_3$

$h_1$

$h_2$

$h_3$
Related Work

Many flaws found on PKCS #11 security tokens.

M. Bortolozzo, M. Centenaro, R. Focardi and G. Steel, CSF’10.
Many flaws found on PKCS #11 security tokens.

M. Bortolozzo, M. Centenaro, R. Focardi and G. Steel, CSF’10.

Proposals for key management APIs with security proofs.

J. Courant, J.-F. Monin, WITS’06.
C. Cachin, N. Chandran, CSF’09.
V. Cortier, G. Steel, ESORICS’09...
Related Work

Many flaws found on PKCS #11 security tokens.

M. Bortolozzo, M. Centenaro, R. Focardi and G. Steel, CSF’10.

Proposals for key management APIs with security proofs.

J. Courant, J.-F. Monin, WITS’06.

C. Cachin, N. Chandran, CSF’09.

V. Cortier, G. Steel, ESORICS’09...

Use of long-term keys implying unrecoverable loss of devices if keys are lost
Breaking Keys in a TRD

«Because I’m bad, really really bad !»

There are ways for the attacker to **break some keys** of a Tamper-Resistant Device (TRD):

- Brute forcing,
- Side-channel attack,
- ...

```
(More) Related Work

Proposals for key management APIs with revocation:

L. Eschenauer, V. D. Gligor, CCS’02.
(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC’07.
(Secret sharing scheme)
Proposals for key management APIs with revocation:

L. Eschenauer, V. D. Gligor, CCS’02.
(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC’07.
(Secret sharing scheme)

Still use long-term keys!
Proposals for key management APIs with revocation:

L. Eschenauer, V. D. Gligor, CCS’02.
(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC’07.
(Secret sharing scheme)

Still use long-term keys!

F. E. Kargl, Sevecom, 2009...
(Two root keys)
(More) Related Work

Proposals for key management APIs with revocation:

L. Eschenauer, V. D. Gligor, CCS’02.
(Using a control server)

X. Z. Yong Wan, B. Ramamurthy, ICC’07.
(Secret sharing scheme)

Still use long-term keys!

F. E. Kargl, Sevecom, 2009...
(Two root keys)

Attacked by S. Möderschein & P. Modesti
(solution proposed but no security proof)
Keys must remain **confidential**: Information about key should not be recovered by the intruder.
Ideal Key Revocation API

Keys must remain **confidential**: Information about key should not be recovered by the intruder.

Any key should be **revocable**: The more sensitive a key is, the more an attacker will try to break it.
Ideal Key Revocation API

Keys must remain **confidential**: Information about key should not be recovered by the intruder.

Any key should be **revocable**: The more sensitive a key is, the more an attacker will try to break it.

The device should remain **functional**: A revocation of a key should not prevent the user from using his/her device.
Our Contributions

• **Design** of an API satisfying previous properties with:
  • **update** functionality,
  • **revocation** functionality.

• **A formal proof of security** ensuring three properties:
  • A key remains secret unless it is broken (brute forced),
  • the system is able to **recover itself** from an attack,
  • a revocation immediately secures the device.
Description of the API

Some assumptions on the tamper-resistant devices:
Description of the API

Some **assumptions** on the tamper-resistant devices:

- **TRD**

A **clock** assumed synchronized with a global clock
Description of the API

Some **assumptions** on the tamper-resistant devices:

- A **table** indexed by handles to store keys’ information (level, validity date, value, ...)
- A **clock** assumed synchronized with a global clock
Some **assumptions** on the tamper-resistant devices:

- A **table** indexed by handles to store keys’ information (level, validity date, value, ...)
- A **blacklist** of elements of the form \((l, t)\)
- A **clock** assumed synchronized with a global clock
We also assume a **hierarchy of levels** for keys:

- with a (partial) order,
- with a maximal and a minimal element.
We also assume a hierarchy of levels for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:
We also assume a hierarchy of levels for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:
We also assume a hierarchy of levels for keys:

- with a (partial) order,
- with a maximal and a minimal element.

Example:
We have a set of **basic commands**.
We have a set of basic commands.

Running example:

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$h'_1$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$h'_2$</td>
</tr>
<tr>
<td>$h_3$</td>
<td>$h'_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>${l, v, m}$</th>
<th>${l, v, m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We have a set of **basic commands**.

Alice

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$\hat{}, l, v, m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

Bob

<table>
<thead>
<tr>
<th>$h'_1$</th>
<th>$\hat{}, l, v, m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h'_2$</td>
<td></td>
</tr>
<tr>
<td>$h'_3$</td>
<td></td>
</tr>
</tbody>
</table>

Alice and Bob share a key and wish to securely exchange a message.
User's Commands

<table>
<thead>
<tr>
<th>Alice</th>
<th>![Icon]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$\text{, l, v, m}$</td>
</tr>
<tr>
<td>$h_2$</td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>
## User’s Commands

| Alice |  
|-------|---
| $h_1$ | $\text{generateSecret}(l_1, m_1)$  
| $h_2$ |  
| $h_3$ |  

$\text{User’s Commands}$
### User’s Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$l, l_1, v, m$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$l_1, v_1, m_1$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

**Alice**

generateSecret($l_1, m_1$)
User’s Commands

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$l, v, m$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$l_1, v_1, m_1$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

$\text{generateSecret}(l_1, m_1)$
To share the new session key with Bob, Alice needs to « export » the new key.

\[ \text{generateSecret}(l_1, m_1) \]

\[ \text{encrypt}(h_2, h_1) \]
To share the new session key with Bob, Alice needs to « export » the new key.

\[ \text{encrypt}(h_2, h_1) \]

Only works if \( l_1 < l \)
### User’s Commands

<table>
<thead>
<tr>
<th>Alice</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>${l, v, m}$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>${l_1, v_1, m_1}$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

To share the new session key with Bob, Alice needs to « export » the new key.

<table>
<thead>
<tr>
<th>Alice</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>${l, v, m}$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>${l_1, v_1, m_1}$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

- **generateSecret**($l_1, m_1$)
- **encrypt**($h_2, h_1$)

Only works if $l_1 < l$
Alice sends the new session key to Bob which can « import » it in his TRD.
Alice sends the new session key to Bob which can « import » it in his TRD.

\[
\text{decrypt}\left(\left\{ l_1, v_1, m_1 \right\}, h'_1\right)
\]
Alice sends the new session key to Bob which can « import » it in his TRD.

\[\text{decrypt}\left(\left\{\overline{\{l_1, v_1, m_1\}}\right\}, h_1'\right)\]

Only works if tests succeed!
Alice sends the new session key to Bob which can « import » it in his TRD.

\[
\text{decrypt}\left(\left\{\mathcal{K}, l_1, v_1, m_1\right\}, h'_1\right)
\]

Only works if tests succeed!
Alice sends the new session key to Bob which can «import» it in his TRD.

\[
decrypt(\{h_1, l_1, v_1, m_1\}, h'_1) = h'_2
\]

Only works if tests succeed!
Alice can now encrypt the message using the session key.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$l, v, m$</td>
<td></td>
</tr>
<tr>
<td>$h_2$</td>
<td>$l_1, v_1, m_1$</td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Alice can now encrypt the message using the session key.

User’s Commands

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$l, v, m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td>$l_1, v_1, m_1$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

encrypt($l, h_2$)
Alice can now encrypt the message using the session key.

User's Commands

<table>
<thead>
<tr>
<th>Alice</th>
<th>encrypt( \text{message}, h_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$l, v, m$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$l_1, v_1, m_1$</td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
</tr>
</tbody>
</table>

\{ \text{message}, 0, v_0, m_0 \}
And, finally, Alice sends the encrypted message to Bob, which decrypts it.
And, finally, Alice sends the encrypted message to Bob, which decrypts it.

\[
\text{decrypt}\left(\left\{ l, 0, v_0, m_0 \right\}, h_2' \right)
\]
User’s Commands

And, finally, Alice sends the encrypted message to Bob, which decrypts it.

decrypt\left(\left\{\text{(document, 0, } v_0, m_0)\right\}, h'_2\right)

Only works if tests succeed!
And, finally, Alice sends the encrypted message to Bob, which decrypts it.

\[
\text{decrypt(\{file, 0, v_0, m_0\}, h'_2)}
\]

Only works if tests succeed!

Information is public, no need for handles.
User’s Commands

A set of basic commands (summary):

\[
\begin{align*}
generatePublic(m) \quad & \quad \text{Generate a nonce or a key, and store under a handle the information.} \\
generateSecret(l, m) \quad & \quad \text{Decrypt } C \text{ with the key stored under } h \text{ and return a message or a handle.} \\
deencrypt(C, h) \quad & \quad \text{Encrypt the input under the key stored in handle } h.
\end{align*}
\]
Lower Level Keys Management

We also have **admin commands:**

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys,** i.e. keys of level Max.
- Each device has its own set of admin keys.
Lower Level Keys Management

We also have **admin commands**:

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys**, i.e. keys of level Max.
- Each device has its own set of admin keys.

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$h_n$</td>
<td>, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>$l, v, m$</td>
</tr>
</tbody>
</table>

Revocation keys of the device.
We also have **admin commands**:

- Allow to **administrate lower level keys** (i.e. level < Max).
- Need **revocation keys**, i.e. keys of level Max.
- Each device has its own set of admin keys.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>Max, $v_1$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$h_n$</td>
<td>Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>$l, v, m$</td>
</tr>
</tbody>
</table>

**Revocation keys of the device.**

**lower level key**
### Lower Level Keys Management

Update value and attributes of keys that are not admin (level Max) keys.

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h_n$</td>
<td>, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>, $l, v, m$</td>
</tr>
</tbody>
</table>
**Lower Level Keys Management**

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$h_n$</td>
<td>, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>, $l, v, m$</td>
</tr>
</tbody>
</table>

\[ \text{update}(C, h_1, \ldots, h_n) \]

Update value and attributes of keys that are not admin (level Max) keys.

\[ C = \left\{ \text{update}, l', v', m' \right\} \ldots \]
### Lower Level Keys Management

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$\text{Max, } v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h_n$</td>
<td>$\text{Max, } v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>$l, v, m$</td>
</tr>
</tbody>
</table>

**update($C', h_1, \ldots, h_n$)**

Update value and attributes of keys that are not admin (level Max) keys.

$$C = \{ \text{update}, l', v', m' \}$$

---

**How does it work?**
Lower Level Keys Management

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h_n$</td>
<td>, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>, $l, v, m$</td>
</tr>
</tbody>
</table>

update($C, h_1, \ldots, h_n$)

Update value and attributes of keys that are not admin (level Max) keys.

$C = \{\text{update, } l', v', m'\}$

How does it work?

1. Tests on keys stored under $h_1, \ldots, h_n$.

> Are they level Max and valid keys?
Lower Level Keys Management

update\((C, h_1, \ldots, h_n)\)

Update value and attributes of keys that are not admin (level Max) keys.

\[ C = \left\{ \text{update, } l', v', m' \right\} \]

How does it work?

1. Tests on keys stored under \( h_1, \ldots, h_n \).
   
   \( > \) Are they level Max and valid keys?

2. Decryption of \( C \).
   
   \( > \) Obtaining old/new value and new attributes.
Lower Level Keys Management

How does it work?

3. Verify that the old key (🔑) is in the device.
Lower Level Keys Management

How does it work?

3. Verify that the old key (钥匙) is in the device.

4. Tests on the new attributes $l'$, $v'$ of new key (钥匙).

> Are the new level and validity date correct?

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>钥匙, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h_n$</td>
<td>钥匙, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>钥匙, $l, v, m$</td>
</tr>
</tbody>
</table>
Lower Level Keys Management

How does it work?

3. Verify that the old key (กด) is in the device.

4. Tests on the new attributes $l', v'$ of new key (กด).

   > Are the new level and validity date correct?

5. Table update with the new values.

\[
\begin{array}{|c|c|}
\hline
h_1 & \text{Max, } v_1 \\
\hline
\cdots & \cdots \\
\hline
h_n & \text{Max, } v_n \\
\hline
h & l, v, m \\
\hline
\end{array}
\]
Lower Level Keys Management

How does it work?

3. Verify that the old key (キー) is in the device.

4. Tests on the new attributes $l'$, $v'$ of new key (キー).
   > Are the new level and validity date correct?

5. Table update with the new values.

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>, Max, $v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>$h_n$</td>
<td>, Max, $v_n$</td>
</tr>
<tr>
<td>$h$</td>
<td>$l'$, $v'$, $m'$</td>
</tr>
</tbody>
</table>
Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

\[ \text{updateMax}(C, h_1, \ldots, h_n) \]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.
The same scheme applies for revoking revocation keys.

updateMax\((C, h_1, \ldots, h_n)\)

> Require a number \(N_{\text{Max}}\) of valid revocation keys.

\[
\begin{array}{|c|c|}
\hline
h_1 & \text{Max, } v_1 \\
\hline
h_2 & \text{Max, } v_2 \\
\hline
h_3 & \text{Max, } v_3 \\
\hline
\end{array}
\]

\[N_{\text{Max}} = 2\]
The same scheme applies for revoking revocation keys.

\[
\text{updateMax}(C, h_1, \ldots, h_n)
\]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.

\[
\{ \text{UpdateMax, } h_1, h_2, h_3, v_1' \}
\]

| \( h_1 \) | \( , \text{Max, } v_1 \) |
| \( h_2 \) | \( , \text{Max, } v_2 \) |
| \( h_3 \) | \( , \text{Max, } v_3 \) |

\[
N_{\text{Max}} = 2
\]
The same scheme applies for **revoking revocation keys**.

\[
\text{updateMax}(C, h_1, \ldots, h_n)
\]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.

\[
\{ \text{UpdateMax, } h_1, h_2, h_3, v'_1 \}
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1 )</td>
<td>(, \text{ Max, } v'_1 )</td>
<td></td>
</tr>
<tr>
<td>( h_2 )</td>
<td>(, \text{ Max, } v_2 )</td>
<td></td>
</tr>
<tr>
<td>( h_3 )</td>
<td>(, \text{ Max, } v_3 )</td>
<td></td>
</tr>
</tbody>
</table>

\( N_{\text{Max}} = 2 \)
Revocation Keys Management

The same scheme applies for revoking revocation keys.

updateMax\((C, h_1, \ldots, h_n)\)

> Require a number \(N_{\text{Max}}\) of valid revocation keys.

\[
\begin{align*}
\{ \text{UpdateMax, } & h_1, \ldots, h_n, v_1' \} \\
\{ \text{UpdateMax, } & h_1, \ldots, h_n, v_2' \} \\
\end{align*}
\]

| \(h_1\) | , Max, \(v_1'\) |
| \(h_2\) | , Max, \(v_2\) |
| \(h_3\) | , Max, \(v_3\) |

\(N_{\text{Max}} = 2\)
Revocation Keys Management

The same scheme applies for revoking revocation keys.

\[ \text{updateMax}(C, h_1, \ldots, h_n) \]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.

\[
\begin{align*}
\{ \text{UpdateMax}, &\quad,\quad, \quad, \quad, \quad, \quad, v_1' \} \\
\{ \text{UpdateMax}, &\quad,\quad, \quad, \quad, \quad, \quad, v_2' \} \\
\{ \text{Max}, &\quad,\quad, \quad, \quad, \quad, \quad, v_3 \}
\end{align*}
\]

\[ h_1 \]
\[ h_2 \]
\[ h_3 \]

\[ N_{\text{Max}} = 2 \]
Revocation Keys Management

The same scheme applies for **revoking revocation keys**.

\[ \text{updateMax}(C, h_1, \ldots, h_n) \]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.

\[
\begin{align*}
\{ \text{UpdateMax}, v_1' \} \\
\{ \text{UpdateMax}, v_2' \} \\
\{ \text{UpdateMax}, v_3' \}
\end{align*}
\]

<table>
<thead>
<tr>
<th>( h_1 )</th>
<th>Max, ( v_1' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_2 )</td>
<td>Max, ( v_2' )</td>
</tr>
<tr>
<td>( h_3 )</td>
<td>Max, ( v_3 )</td>
</tr>
</tbody>
</table>

\( N_{\text{Max}} = 2 \)
The same scheme applies for revoking revocation keys.

\[ \text{updateMax}(C, h_1, \ldots, h_n) \]

> Require a number \( N_{\text{Max}} \) of valid revocation keys.

\[
\{ \text{UpdateMax}, h_1, \ldots, v' \} \\
\{ \text{UpdateMax}, h_2, \ldots, v'_2 \} \\
\{ \text{UpdateMax}, h_3, \ldots, v'_3 \}
\]

| \( h_1 \) | Max, \( v'_1 \) |
| \( h_2 \) | Max, \( v'_2 \) |
| \( h_3 \) | Max, \( v'_3 \) |

\( N_{\text{Max}} = 2 \)
Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public?

\[
\begin{align*}
\{ \text{UpdateMax}, & \quad , \quad , \quad , \quad , v'_1 \} \\
\{ \text{UpdateMax}, & \quad , \quad , \quad , \quad , v'_2 \} \\
\{ \text{UpdateMax}, & \quad , \quad , \quad , \quad , v'_3 \} \\
\end{align*}
\]
Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public?

\[
\{\text{UpdateMax, } k_1, k_2, v'_1 \}\]
\[
\{\text{UpdateMax, } k_3, k_4, v'_2 \}\]
\[
\{\text{UpdateMax, } k_5, k_6, v'_3 \}\]
Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public?

\[
\{ \text{UpdateMax}, v_1' \} + \{ \text{UpdateMax}, v_2' \} + \{ \text{UpdateMax}, v_3' \}
\]
Revocation Keys Management

What if (old) revocation keys can be lost and if revocation messages are public?

\[
\begin{align*}
\{ \text{UpdateMax}, v_1 \} \\
\{ \text{UpdateMax}, v_2 \} \\
\{ \text{UpdateMax}, v_3 \}
\end{align*}
\]

The intruder can break all the level Max keys! (up to the current ones)
Hypothesis:

Level Max commands are sent over a secure channel.
Hypothesis:

Level Max commands are sent over a secure channel.

This can be achieved by several means:

• The administrator has a physical access to the TRD that needs to be updated,

• The user would connect his/her TRD to a trusted machine, on which a secure channel (e.g. via TLS) is established with the key administrator.
And now, what about Security?
And now, what about Security?
And now, what about Security?
Abstraction

Messages are represented by terms

Nonces, keys :

\[ n, m, \ldots, k_1, k_2, \ldots \]

Primitives :

\[ \{m\}_k, \langle m_1, m_2 \rangle \]

Modeling deduction rules :

\[
\begin{align*}
  & \frac{x \quad y}{\langle x, y \rangle} & \frac{\langle x, y \rangle}{x} & \frac{\langle x, y \rangle}{y} & \frac{x \quad y}{\{x\}_y} & \frac{\{x\}_y \quad y}{x}
\end{align*}
\]
We model the system using **global states**:

$$(P, I, M, N, K, t)$$
We model the system using **global states**: 

\[(P, I, M, N, K, t)\]

\(P\), the set of **TRDs** in use in the system.
Formal Model

We model the system using **global states**:

\[ (\mathcal{P}, I, \mathcal{M}, N, K, t) \]

\(\mathcal{P}\), the set of **TRDs** in use in the system.

\(\mathcal{M}\), the set of **messages** that have been **sent on the network**.

(Represents also the **knowledge of the intruder**.)
We model the system using **global states**:

\[(\mathcal{P}, \mathcal{I}, \mathcal{M}, N, K, t)\]

\(\mathcal{P}\), the set of **TRDs** in use in the system.

\(\mathcal{M}\), the set of **messages** that have been **sent on the network**.

(Represents also the **knowledge of the intruder**.)

\(N\), the set of **nonces** currently in used in the system.
Formal Model

We model the system using **global states**:

\[(\mathcal{P}, \mathcal{I}, \mathcal{M}, N, K, t)\]

\(\mathcal{P}\), the set of **TRDs** in use in the system.

\(\mathcal{M}\), the set of **messages** that have been **sent on the network**.
\(\) (Represents also the **knowledge of the intruder**.)

\(N\), the set of **nonces** currently in used in the system.

\(K\), the set of **keys** currently in used in the system.
We model the system using global states:

\[(\mathcal{P}, \mathcal{I}, \mathcal{M}, N, K, t)\]

\(\mathcal{P}\), the set of TRDs in use in the system.

\(\mathcal{M}\), the set of messages that have been sent on the network. (Represents also the knowledge of the intruder.)

\(N\), the set of nonces currently in used in the system.

\(K\), the set of keys currently in used in the system.

\(t\), represents the current time.
Formal Model

We model the system using **global states**: 

\[(P, I, M, N, K, t)\]

\[I : a \mapsto (\Theta_a, H_a, \mathcal{B}_a, t, N_a, K_a)\]

is a **function** describing the **local state** of TRD. \(a\)
Formal Model

We model the system using **global states**: 

\[(\mathcal{P}, \mathcal{I}, \mathcal{M}, N, K, t)\]

\[\mathcal{I} : a \mapsto (\Theta_a, H_a, \mathcal{B}_a, t, N_a, K_a)\]

is a **function** describing the **local state** of TRD \(a\) \(B_a\), the set of **blacklisted levels**.
We model the system using **global states**: 

\[(P, I, M, N, K, t)\]

\[I : a \mapsto (\Theta_a, H_a, B_a, t, N_a, K_a)\]

is a **function** describing the **local state** of TRD \(a\). 

\(B_a\), the set of **blacklisted levels**. 

\(\Theta_a\), a function representing the **memory** of the TRD.

<table>
<thead>
<tr>
<th>Handle</th>
<th>Value</th>
<th>Level</th>
<th>Validity</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h_1)</td>
<td>![key]</td>
<td>(l_1)</td>
<td>(v_1)</td>
<td>(m_1)</td>
</tr>
<tr>
<td>(h_2)</td>
<td>![key]</td>
<td>(l_2)</td>
<td>(v_2)</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Formal Model

Semantics

consists in several transitions modifying the global state.
Semantics

consists in several transitions modifying the global state.

\[(\text{TIM}) \quad (P, I, M, N, K, t) \rightarrow (P, I, M, N, K, t') \quad (t' > t)\]

models the time passing...
Formal Model

Semantics

consists in several transitions modifying the global state.

\[(TIM) \ (P, I, M, N, K, t) \rightarrow (P, I, M, N, K, t') \quad (t' > t)\]

models the time passing...

\[(DED) \ (P, I, M, N, K, t) \rightarrow (P, I, M \cup \{m\}, N, K, t)\]

models the deduction abilities of the intruder. \((M \vdash m)\)
Formal Model

Semantics

consists in several transitions modifying the global state.

\[(\text{TIM}) \quad (P, I, M, N, K, t) \rightarrow (P, I, M, N, K, t') \quad (t' > t)\]

models the time passing...

\[(\text{DED}) \quad (P, I, M, N, K, t) \rightarrow (P, I, M \cup \{m\}, N, K, t)\]

models the deduction abilities of the intruder. \((M \vdash m)\)

\[(\text{UPD}) \quad (P, I, M, N, K, t) \rightarrow (P, I', M \cup \{m\}, N', K', t)\]

models changes when an update command is performed.

\[m = \left\{ \text{update}, k, k', l', v', m' \right\}_{k_1 \ldots k_n} \]
Knowledge of the Intruder

TRD

API

Internet

TRD

API

TRD

API

TRD
Knowledge of the Intruder

A key in a TRD may be lost and known by the intruder

Gotcha!
**Hypothesis:**

At most a total of $N_{\text{Max}} - 1$ different « current » level Max keys for one TRD can be lost.

A key in a TRD may be lost and known by the intruder.
What about lost keys?
What about lost keys?
What about lost keys?

The intruder has control over whatever is under a level with a lost key.
What about lost keys?

The intruder has control over whatever is under a level with a lost key.

She may use an encrypt command to get a key with a lower level in a TRD containing a lost key.

Ex: Receive \( \{ \text{key}, l_9, v, m \} \) with lost and of level \( l_5 \).
Secrecy Result

Even if the **intruder** may:

- **control the network** and host machines,
- **break some keys** (but not too many revocation keys),

«I keep my secrets secret !»
Secrecy Result

«I keep my secrets secret !»

Even if the **intruder** may:

- **control the network** and host machines,
- **break some keys** (but not too many revocation keys),

We have:

**Theorem 1**

Keys remain secret (not deducible) provided:

A valid expiration date & not « under a lost »
Formally speaking...

**Theorem 1**

Let $E = (P, I, M, N, K, t)$ be a global state, $L_V$ a set of (broken) levels and $k \in K$.

$\forall k \text{ s.t. } \text{Level}(k) \nsubseteq L_V, \quad M \not\models k$
Secrecy Result

«I keep my secrets secret!»

Formally speaking...

**Theorem 1**

Let $E = (\mathcal{P}, \mathcal{I}, \mathcal{M}, N, K, t)$ be a global state, $L_V$ a set of (broken) levels and $k \in K$.

$$\forall k \text{ s.t. } \text{Level}(k) \not\subset L_V, \quad \mathcal{M} \not\models k$$

Proof (sketch of):

> **Find invariant properties** of the system.

> **Prove** them!
Self Repair Property

«It’s just a flesh wound !»
Self Repair Property

«It’s just a flesh wound!»

$\ell_{\text{Max}} \quad \ldots \quad \ell_1 \quad \ldots \quad \ell_2 \quad \ldots$

TRD
Self Repair Property

«It's just a flesh wound!»

$l_{\text{Max}} \quad \ldots \quad l_1 \quad \ldots \quad l_2 \quad \ldots$

TRD
Self Repair Property

«It's just a flesh wound!»

$V_{\text{Max}} \ldots V_1 \ldots V_2$
Self Repair Property

«It’s just a flesh wound!»

\[ l_{\text{Max}} \ldots l_1 \ldots l_2 \]

TRD
Self Repair Property

«It's just a flesh wound !»

$l_{\text{Max}} \quad \ldots \quad l_1 \quad \ldots$

$\ldots \quad \ldots \quad l_2 \quad \ldots$

TRD
Self Repair Property

«It’s just a flesh wound!»

\[ l_{\text{Max}} \rightarrow \ldots \rightarrow l_1 \rightarrow l_2 \rightarrow \ldots \]

TRD
Self Repair Property

«It’s just a flesh wound !»

\[ l_{\text{Max}} \quad \ldots \quad l_1 \quad \ldots \quad l_2 \quad \ldots \]

TRD
Self Repair Property

«It’s just a flesh wound !»

TRD
Self Repair Property

«It’s just a flesh wound !»
Self Repair Property

«It’s just a flesh wound !»

TRD
Self Repair Property

«It’s just a flesh wound!»

$\ell_{\text{Max}}$ $\cdots$ $\ell_1$ $\cdots$ $\ell_2$ $\cdots$

TRD
Self Repair Property

«It's just a flesh wound!»

Can not be compromised using this key

TRD
Self Repair Property

«It's just a flesh wound!»

$l_{\text{Max}} \quad \ldots \quad l_1 \quad \ldots \quad l_2 \quad \ldots$

TRD
Self Repair Property

«It’s just a flesh wound!»

$\ell_{\text{Max}} \rightarrow \ldots \rightarrow \ell_1 \rightarrow \ell_2 \rightarrow \ldots$

TRD
Self Repair Property

It's just a flesh wound!

Can not be compromised using remaining corrupted keys.
Self Repair Property

«It's just a flesh wound!»

Can not be compromised using remaining corrupted keys.

We gain a level!
Self Repair Property

«It's just a flesh wound!»

Can not be compromised using remaining corrupted keys.

We gain a level!

TRD
Self Repair Property

«It’s just a flesh wound !»

$\ell_{\text{Max}}$ \quad $\ldots$ \quad $\ell_1$ \quad $\ldots$

$\ell_2$

$\ell_3$

TRD
Self Repair Property

«It’s just a flesh wound!»

$\ell_{\text{Max}} \quad \ldots \quad \ell_1 \quad \ldots$

$\ell_2$

$\ell_3$

TRD
Self Repair Property

«It’s just a flesh wound!»

\[ l_{\text{Max}} \quad \ldots \quad l_1 \quad \ldots \quad l_2 \quad \ldots \quad l_3 \]

TRD
Then, the story went, until the TRD was fully repaired and it lived happily ever after...
Then, the story went, until the TRD was fully repaired and it lived happily ever after...

**Theorem 2** *(Stated for one level)*

Assume that all keys are secret at time $t$ except those under a level $l$. Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$. 

«It’s just a flesh wound!»
Then, the story went, until the TRD was fully repaired and it lived happily ever after...

**Theorem 2** *(Stated for one level)*

Assume that all keys are secret at time $t$ except those under a level $l$. Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$.

It assumes that, during time $\Delta(l)$, you do not lose a level higher than the one you «try» to repair.
Then, the story went, until the TRD was fully repaired and it lived happily ever after...

**Theorem 2** (Stated for one level)

Assume that all keys are secret at time $t$ except those under a level $l$. Then at time $t + \Delta(l)$, all keys are secret except those under levels $l_1, \ldots, l_n$ such that $l_i < l$.

It assumes that, during time $\Delta(l)$, you do not lose a level higher than the one you «try» to repair.
Blacklist Option

blacklist\((C, h_1, \ldots, h_n)\)

Ex: \(C = \{\langle\text{blacklist}, \langle l_3, t \rangle\rangle\} \ldots \)
Blacklist Option

\[ \text{blacklist}(C, h_1, \ldots, h_n) \]

Ex: \( C = \{\langle\text{blacklist}, \langle l_3, t \rangle\}\} \ldots \)
Blacklist Option

\[ \text{blacklist}(C, h_1, \ldots, h_n) \]

Ex: \( C = \{(\text{blacklist}, (l_3, t))\} \ldots \)

(diagram of keys and TRD)
Blacklist Option

blacklist\((C, h_1, \ldots, h_n)\)

Ex: \(C = \left\{ \langle \text{blacklist}, \langle l_3, t \rangle \rangle \right\} \)

«For those who are in a hurry...»
Blacklist Option

«For those who are in a hurry...»

Theorem 3 (Stated for one level)

Assume that all keys are secret at time $t$ except those under a level $l$.
If we blacklist level $l$ on a TRD, then, immediately, all keys are secret.
**Theorem 3** (Stated for one level)

Assume that all keys are secret at time $t$ except those under a level $l$. If we blacklist level $l$ on a TRD, then, immediately, all keys are secret.

- It **only works** for the blacklisted TRD.
- The time of the blacklist should be long enough.
- It **prevents the attacker** to operate on the TRD.
Future Work

- **Weaken assumptions**, especially on hidden level Max messages (maybe requiring more cryptographic primitives),

- **Extend** revocation to **asymmetric encryption**,

- **Adapt** the result taking account of possible **clock skew**, or replacing the clock by some sort of nonce based freshness test,

- **Implement** the API in order to carry out some performance tests. [Ongoing work in JavaCard]
Thank you for your attention!

Can we implement Clock Stew?  
Maybe...
Was that a Nice Talk?

Command_not_supported

Speaker  
(Host Machine)

Security API

Truth  
(Trusted device)