How to Make Operating Systems for Smart Cards Open

Przemysław Błaśkiewicz, Przemysław Kubiak, Mirosław Kutyłowski
Wrocław University of Technology

Bulcrypt 2012, Sofia

Smart card as a secure $* * *$ device

Motivation for smart cards and similar embedded systems

1. controlling security design of smart cards is not as hopeless as in case of complex devices
2. cheap hardware
3. can be devoted to single tasks - solving concrete critical problems

Deploying Applications on a Smart Card

Typical strategies

Native system

everything is done by the card manufacturer

advantages
- it is easier to control security if everything is in hands of a single party
- no need for complex management of application on card

disadvantages
- closed systems tend to be outdated, obscure, and do not profit from diversity of ideas
- adding anything requires restarting security analysis

Java Card idea

advantages
- flexibility, open for third party designers

disadvantages
- the Java concept in principle should provide secure environment, but . . .
- no control over what is really deployed in concrete cards.

Idea

Operating system

1. publish all details of operating system, libraries, etc. necessary to develop applications for a smart card
2. simplify OS - eliminate implementation of Java and its mechanisms
3. but **add a strict control on what can be uploaded on a smart card**

Idea

Operating system

1. publish all details of operating system, libraries, etc. necessary to develop applications for a smart card
2. simplify OS - eliminate implementation of Java and its mechanisms
3. but **add a strict control on what can be uploaded on a smart card**

Problem

how to build a lightweight system that enables to control uploading process effectively?

Protection by signatures

Signed software

only a code signed with the private key of the system provider can be accepted by a smart card

Consequences

- ⇒ the corresponding public key must be stored on the smart card
- ⇒ **status of the public key has to be checked**

Checking status of public key

OCSP

- heavy,
- even in case of web services not frequently used – the status not checked at all! . . .

Checking status of public key

OCSP

- heavy,
- even in case of web services not frequently used – the status not checked at all! . . .

CVC mechanism

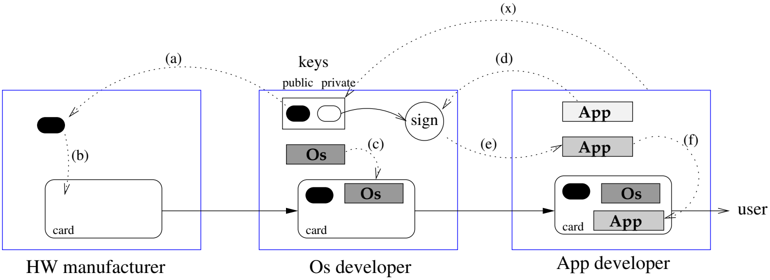
used for German identity documents:

- public keys replaced periodically
- chain principle – *trust points principle*

disadvantages:

- danger of splitting a chain
- asymmetric operations - crypto processor needed

Flow of events for application development



- **a,b** public key embossed in card's ROM
- **c** OS is implemented on the card
- **d,e** request for approving an application, signature issued
- **f** smart card accepts application after verifying the signature

Requirements

Public key on smart card

1. not changed (in ROM)
2. status cannot be checked

Signing

1. possibility of compromise of the private key
2. a distributed control - no **single** point of failure

Błaśkiewicz,
Kubiak,
Kutyłowski

Tree of hashes

1. 2^h public keys for one-time signatures:

$$Y_0, Y_1, \ldots, Y_{2^h-1}$$

2. a binary tree with 2^h leaves
3. labels:
   - leaves with labels $Y_0, Y_1, \ldots, Y_{2^h-1}$
   - a node with children nodes holding labels B and C gets the label $\tilde{H}(B, C)$

the label of the root represents all public keys of the leaves

Signature in Merkle Tree

Form of a signature

- one-time signature using a leaf key
- a path leading to the root from this leaf with all hashes at sibling nodes

one-time signature

- also based on hashes
- simplest signature of a bit b:
  - private key X_0, X_1, public key: Y_0, Y_1 where $Y_i = H(X_i)$
  - signature for b: X_i

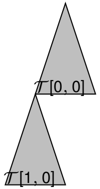
Mediated version

Idea of a mediated version

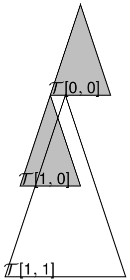
1. if X is a secret value and $Y = H(X)$ is required for a signature, then replace X by k shares and put $Y = H(X_1, \ldots, X_k)$
2. separate the shares:
   - the first shares on HSM1
   - the second shares on HSM2
   - ...
   - the kth shares on HSMk

creating a signature requires cooperation of **all** HSM's

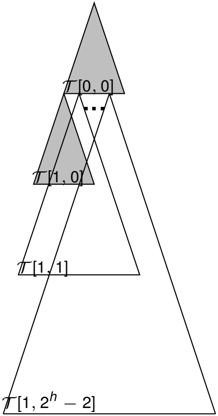
Lazy creation of two-level Merkle tree



Lazy creation of two-level Merkle tree



Lazy creation of two-level Merkle tree



Lazy creation of two-level Merkle tree

Some details

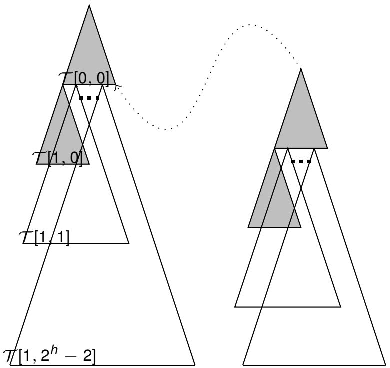
1. construction of the next subtrees can be performed on-the-fly
   – when using one leaf create 2 new ones
2. each new subtree contains one more layer

Revocation

What happens when one HSM leaks keys

1. start a new tree
2. sign the new root with the last public key from the compromised tree
3. destroy the previous keys in the HSM's (also the honest ones)

Changing Merkle tree



Consequences

What happens after revocation

1. smart card does not need to change the public key stored in ROM
2. problems with **one** HSM do not endanger old signatures: they have been created after software inspection
3. as honest HSM cleared, the keys from corrupted HSM have no siblings and cannot be used

Conclusions

1. without PKI: long term control over software for smart card
2. compromising keys does not lead to smart cards replacement

Thanks for your attention!

Contact data

1. `Miroslaw.Kutylowski@pwr.wroc.pl`, `Przemyslaw.Kubiak@pwr.wroc.pl`
2. `http://kutylowski.im.pwr.wroc.pl`
3. +48 71 3202109, +48 71 3202105
   fax: +48 71 3202105