Supervised Usage of Signature Creation Devices

Przemysław Kubiak, Mirosław Kutyłowski

Wrocław University of Technology
Wrocław, Poland

INSCRYPYPT 2013, Guangzhou
Electronic signatures
security – not only cryptographic schemes

what we need to intrust electronic signatures?

1. a good cryptographic scheme
2. a secure device “implementing” secret signing key
3. effective control over this device by the signatory
Electronic signatures
security – not only cryptographic schemes

what we need to intrust electronic signatures?

1. a good cryptographic scheme
2. a secure device “implementing” secret signing key
3. effective control over this device by the signatory
Secure Signature Creation Device (SSCD)

Properties of SSCD

1. implementation: the signing key can be used for creating signatures

2. no other possibilities – including in particular exporting the key outside the SSCD

3. logical and physical protection of the signing key

4. SSCD must be activated by the user
Problems with SSCD

Major issues with SSCD

- physical protection is a “mouse and a cat game”, the protection effective today might fail tomorrow
- usually: activation = entering a PIN number

Threats related to PIN

1. the PC learns the PIN, unless the keyboard with the reader
2. learning the PIN by observing the signatory
Further Problems with SSCD

How do you know how the SSCD is used?

- maybe there is a clone of the SSCD
- maybe there is a second PIN that is used as a back-door

You trust the issuer and you cannot prove anything in a case of a fraud.
Electrical signatures and the ENRON-like cases

Completeness of documentation

In many cases it would save us a lot of problems to know that the list of signatures created with a SSCD is the complete list of the signatures created with this SSCD.

Examples

- financial bookkeeping
  *think about the ENRON case*
- signatures by the legal representatives of legal persons
- electronic document flow in corporations
- detecting clones of a SSCD
Design Goals

Inspection procedure
A Third Trusted Party would be able to check whether
■ the list of signatures presented is the complete list of signatures created with a given SSCD
■ ... without seeing the documents signed

Signature format
preferably, the signatures should be verified with exactly the same way as standard signatures
Counter in the signature

an extra field with the serial number

Problems:

- serial numbers in plaintext – the signature recipients gets more information than intended
- encrypted serial numbers – signing random values? Potentially dangerous.
- legal problems: e.g. European Directive prohibits changing the document to be signed by the SSCD
Mediated signatures would not help in case of ENRON (if the mediator run by the company)

Stamp&Extend (INTRUST 2012) cannot be run by a smart card alone, a solution for a server
Signatures suited for CTRL-Sign

CTRL-Sign reuses $r = g^k$ the signature component $r = g^k$, where $k$ chosen at random

$r$ might be given explicitly (ElGamal) or can be reconstructed (in particular: Schnorr, DSA, ECDSA, Guillou-Quisquater, Nyberg-Rueppel)

For example Schnorr:

1. choose $k \in [1, q - 1]$ uniformly at random,
2. $r := g^k$,
3. $e := \text{Hash}(M, r)$,
4. $s := (k - x \cdot e) \mod q$.
5. output signature $(e, s)$.

Note that $g^s \cdot y^e = r$, where $y = g^x$
Autors of CTRL-Sign

Protocol participants

**Card Issuer**: produces SSCD and installs system-wide parameters,

**Certification Authority (CA)**: issue certificates for public keys of the signatories

**Signatories**: hold SSCDs and create signatures

**Verifiers**: verify signatures in the standard way

**Inspection Authority (IA)**: check whether the presented lists of signatures created by SSCD are complete
Setup

Keys

**Inspection Authority:** secret key $k_{master}$. For a user $U$:
- the control key $c_U := Hash_1(U, k_{master})$,
- the private inspection key $i_U = Hash_2(U, k_{master})$,
- the public inspection key $I_U = g^{i_U}$.

**Card Issuer:** for a user $U$, installs $c_U$ and $I_U$ in the SSCD issued for $U$.

**Signatory $U$:**
- the preinstalled keys $c_U$, $I_U$,
- the private signing key $x_U$,
- the public key $X_U = g^{x_U}$.

**Certification Authority:** as usual for X.509 framework.
CTRL-Sign – Signature Creation

Signature creation procedure

the original scheme with slight changes concerning the choice of $k$

1. generate $k$ at random,
2. check the hidden footprint of $k$, if it is incorrect return to step 1,
3. proceed $\text{Sign}$ for the parameter $k$ chosen.

Hidden footprint

- input: $I_U, k$
- $\text{footprint} := \text{Hash}_3(I_U^k)$
- output $d$ least significant bits of $\text{footprint}$
Hidden footprints

For the SSCD of user $U$

- the key $c_U$ shared by SSCD and the Inspection Authority used to derive a control sequence
  \[ \text{RAND}_U := \text{PRNG}(c_U) \]

- $\text{RAND}_U$ determines footprint values:
  \[ \text{RAND}_U = \rho_{U1}^1 \rho_{U2}^2 \ldots \]

where each $\rho_{Ui}^i$ is a $d$-bit string

- $\rho_{Ui}^i$ is the footprint of the $i$th signature created by the SSCD of user $U$
CTRL-Sign – Signature Verification

According to the standard procedure:

- the CTRL-Sign does not contain any extra fields to be checked
- only the parameter \( r = g^k \) is chosen in a special way ...
- ... but neither a verifier nor a signatory can check it (this is a security requirement)
the way used during signature creation cannot be used as the parameter $k$ cannot be revealed to IA. But:

$$I_U^k = g^{iuk} = (g^k)_{iU}$$

(borrowed from kleptography by Young&Yung)

**Checking footprints by IA**

- Recompute $\rho^1_U \rho^2_U \ldots$ using the shared key $c_U$
- For the $i$th signature with parameter $r_i$
  - Compute $\text{Hash}(r_i^{iU})$
  - Check if its last $d$ bits are equal to $\rho^i_U$
Manipulations and the Inspection Outcome

Asking for signatures in the court

The judge can request to use the SSCD in the court. Then the list of signatures appended with the requested number of last signatures

Probability of an unnoticed replacement

The signatory cannot compute the footprint, so has to try blindly. It succeeds with probability $2^{-d}$ for a single signature.
Manipulations and the Inspection Outcome

Assume a single signature was removed from the list

If signature \( t \) removed:

- the footprints originally \( \rho_1, \ldots, \rho_{t-1}, \rho_t, \rho_{t+1}, \ldots, \rho_N \)
- after removing: \( \rho_1, \ldots, \rho_{t-1}, \rho_{t+1}, \ldots, \rho_N \)

it follows that:

\[
\rho_t = \rho_{t+1}, \rho_{t+1} = \rho_{t+2}, \ldots, \rho_{N-1} = \rho_N
\]

unlikely, situation unknown to the adversary

Removing \( k \) signatures

One can observe that then there are

\[
N - t + k
\]

equalities for the values \( \rho^j_U \) to be satisfied, where \( t \) is the position of the first omitted signature and \( N \) is the index of the last signature created
Key Privacy

Problem
The Inspection Authority has some additional knowledge about the signatures: for each signature with $r = g^k$, it knows $d$ bits of $\text{Hash}_3(I_k^U)$.

Result
Showing that if there is an attack with the keys held by IA, then there is an attack on the background signature scheme in the ordinary setting (only more signatures are needed).
Secrecy of Footprints

Problem

The footprints are secure, if the CTRL-Sign lists of signatures are indistinguishable from the lists generated in the ordinary way.

Proof

In order to get a rigorous formal proof we have to modify slightly the scheme - Enhanced CTRL-Sign.
Enhanced CTRL-Sign

input: $t$, message $M$, secret keys $b_U$, $c_U$, $x_U$, public key $I_U$

1. $\text{RAND}_U := \text{PRNG}(c_U)$ and extract $\rho^t_U$ from $\text{RAND}_U$
2. $k^{(t)}_2 := \text{Hash}_5(b_U, i)$
3. choose $k_1$ at random
4. $z := I^k_U$
5. while $\rho^t_U \neq$ the last $d$ bits of $\text{Hash}_3(z)$ do
   6. $k_1 := k_1 + 1$
   7. $z := z \cdot I_U$
8. $k := k_1 + k^{(t)}_2$
9. $r := g^k$
10. generate a signature $S = (r, s)$ for the message $M$ using $k$ and $r$
note that IA can compute \( \left( \frac{r}{g^{k_2(t)}} \right)^{i_U} \) and that
\[
\left( \frac{r}{g^{k_2^{(i)}}} \right)^{i_U} = \left( g^{k_2^{(i)} - k_2^{(i)}} \right)^{i_U} = \left( g^{k_1} \right)^{i_U} = I_{k_1}^A
\]

For security:
now \( r \) has the uniform probability distribution
Implementation Issues

Computation time

- in order to find a proper exponent one has to try different choices of $k$
- we cannot make more than 1 exponentiation – otherwise the smart card implementation could be too slow
- since the footprint values are taken from $Hash_3(I_U^k)$ even for related values of $k$ we get “random independent” footprints
Efficient Search for $k$

**Procedure**

\[
R := I_U^k \\
\text{while } Hash_3(R) \neq \rho_i U \text{ do} \\
\quad k := k + 1; \\
\quad R := R \cdot I_U \\
\text{end} \\
r := g^k
\]

Time complexity, extra operations:
- one extra exponentiation
- a few multiplications and hash evaluations (negligible time)
### Experimental results

<table>
<thead>
<tr>
<th>operation</th>
<th>MultiApp ID</th>
<th></th>
<th>MultiApp ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dual Citizen</td>
<td>144K CC v2.0 Infineon</td>
<td>144K ECC v2.1 NXP</td>
</tr>
<tr>
<td>scalar multiplication</td>
<td>186 ms</td>
<td>104 ms</td>
<td></td>
</tr>
<tr>
<td>ECDSA signature with SHA1</td>
<td>191 ms</td>
<td>111 ms</td>
<td></td>
</tr>
<tr>
<td>ECDSA signature with SHA256</td>
<td>194 ms</td>
<td>112 ms</td>
<td></td>
</tr>
<tr>
<td>verification of ECDSA+SHA1</td>
<td>140 ms</td>
<td>112 ms</td>
<td></td>
</tr>
<tr>
<td>verification of ECDSA +SHA256</td>
<td>141 ms</td>
<td>115 ms</td>
<td></td>
</tr>
<tr>
<td>SHA-1 computation</td>
<td>4 ms</td>
<td>4 ms</td>
<td></td>
</tr>
<tr>
<td>SHA-256 computation</td>
<td>8.6 ms</td>
<td>6.4 ms</td>
<td></td>
</tr>
</tbody>
</table>

Experimental results, Gemalto Java cards and 256-bit elliptic curve.
Problem with Multiplication /Point Addition

**Problem: multiplication is not as a primitive to be used as a call to secure co-processor**

- Java implementation - too slow
- tricks - via exponentiation – the resulting cost of an exponentiation
- an internal implementation is necessary - only the manufacturer has the right to do it
Hiding Extra Operations

Saving time

- computing $g^k$ and $I_u^k$ at the same time
- to prevent leaking via computation time:
  - unnecessary multiplications performed while performing exponentiations
    (a squaring per bit of the exponent, multiplications corresponding to 1’s in the binary representation of the exponent
  - this time can be used instead for the extra operations
Thanks for your attention!

Contact data

1. Miroslaw.Kutylowski@pwr.wroc.pl
3. +48 71 3202109, +48 71 3202105
   fax: +48 71 3202105