The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10

Daniel De Almeida Braga
Pierre-Alain Fouque
Mohamed Sabt

April, 9th 2020
1 Context

2 Notation & Reminders

3 Deterministic RSA Padding

4 Padding Oracle on Key Transport

5 Key Reuse

6 Secure Implementation

7 Conclusion
The smart card world
The smart card world
SCP (Secure Communication Protocol)

Establish a secure session between a card and an Off-Card Entity

2-steps protocol: Key Exchange + Communication

SCP10 relies on a Public Key Infrastructure:
- Both the card and off-card entity have a key pair
- They use each other public key to encrypt/verify messages
SCP (Secure Communication Protocol)
SCP (Secure Communication Protocol)

Establish a secure session between a card and an Off-Card Entity using a 2-step protocol: Key Exchange + Communication. SCP10 relies on a Public Key Infrastructure, where both the card and off-card entity have a key pair and use each other's public keys to encrypt/verify messages.
SCP (Secure Communication Protocol)

- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication
SCP (Secure Communication Protocol)

- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication
- SCP10 relies on a Public Key Infrastructure:
  - Both the card and off-card entity have a key pair
  - They use each other public key to encrypt/verify messages
Key Exchange Modes

(a) Key Transport mode
Key Exchange Modes

(a) Key Transport mode

(b) Key Agreement mode
Our contributions:

1. Abuse blurs and flaws in the RSA encryption in Key Transport
2. Recovered session keys by two independent means
   - In less than a second with the first attack
   - In an average of 2h30 for the second
3. Exploit a design flaw in the specification to forge a valid certificate, signed by the card (allowing impersonation)
4. Implement a (semi-)compliant version of SCP10 as an applet
5. Propose a secure implementation, with an estimation of the corresponding overhead
Our contributions:

1. Abuse blurs and flaws in the RSA encryption in Key Transport
2. Recovered session keys by two independent means
   - In less than a second with the first attack
   - In an average of 2h30 for the second
3. Exploit a design flaw in the specification to forge a valid certificate, signed by the card (allowing impersonation)
4. Implement a (semi-)compliant version of SCP10 as an applet
5. Propose a secure implementation, with an estimation of the corresponding overhead

However, we did not:

- Attack real cards (no implementation in the wild)
- Try to exploit weakness in the symmetric encryption
Our Threat Model

Our attacker can:

✓ Initiate an SCP10 session with a card
✓ Intercept, read and modify plaintext message transmitted between a legitimate Off-Card Entity and the card
✓ Measure the time needed by the card to respond

She cannot:

× Have physical access to the card
× Break the cryptographic primitives
Notation & Reminders
Acronyms

- APDU: Application Protocol Data Unit
  Message format of request send to the card
- TLV: Tag Length Value
  Data structure used to ease parsing
- CRT: Control Reference Template
  Data structure defining a symmetric key and its usage
- IV: Initialization Vector
  Initialisation vector used to initialize symmetric encryption
RSA and padding

**RSA:**

\[ pub = (n, e) \]
\[ priv = (n, d) \]

*Encryption:*  \[ c = m^e \mod n \]

*Decryption:*  \[ m = c^d \mod n \]

*Signature:*  \[ s = RSA_{sign}(m, priv) \]

*Verification:*  \[ m == RSA_{ver}(m, pub) \]
RSA and padding

RSA:
\[ \text{pub} = (n, e) \]
\[ \text{priv} = (n, d) \]

Encryption: \[ c = m^e \mod n, \]
Decryption: \[ m = c^d \mod n. \]

Signature: \[ s = \text{RSA}_\text{sign}(m, \text{priv}), \]
Verification: \[ m == \text{RSA}_\text{ver}(m, \text{pub})? \]

PKCS#1v1.5 padding:

\[ \text{Enc: EME-PKCS1-v1_5}(m) = 0x00 \ || \ 0x02 \ || \ \text{PS} \ || \ 0x00 \ || \ m \]
\[ \text{Sig: EMSA-PKCS1-v1_5}(m) = 0x00 \ || \ 0x01 \ || \ 0xFF..FF \ || \ 0x00 \ || \ m \]
Deterministic RSA Padding
Perform Security Operation APDU:

\[ M: \text{params} \ || \ CRT \ [\ || \ CRT] \]
**Perform Security Operation**

APDU:

\[ M: \text{params} \ || \ CRT \ [\ || \ CRT] \xrightarrow{\text{padding}} \ EM \]

\[ EM: 0002 \ || \ FF..FF \ || \ 00 \ || \ \text{params} \ || \ CRT \ [\ || \ CRT \ldots] \]

\[ 128 - \text{len}(CRTs) - 3 \text{ bytes} \quad 3 \text{ bytes} \quad [22,42] \text{ bytes} \]

→ Hybrid between EME and EMSA
**Perform Security Operation**

**APDU:**

\[ M: \text{params} || \text{CRT} [|| \text{CRT}] \overset{\text{padding}}{\longrightarrow} \text{EM} \]

\[ \text{EM: } 0002 || FF..FF || 00 || \text{params} || \text{CRT} [|| \text{CRT} ...] \]

\[ 128 - \text{len(CRTs)} - 3 \text{ bytes} \]
\[ 3 \text{ bytes} \]
\[ [22,42] \text{ bytes} \]

→ Hybrid between EME and EMSA

**CRT:**

\[ \text{header} || \text{key} [|| 91 08 \text{ iv}] \]

\[ [6,8] \text{ fixed bytes} \]
\[ [16,24] \text{ bytes} \]
\[ 8 \text{ bytes} \]
Perform Security Operation APDU:

\[
M: \text{params} \ || \ CRT \ [\ | | \ CRT] \xrightarrow{\text{padding}} EM
\]

\[
EM: 0002 \ || \ FF..FF \ || \ 00 \ || \ \text{params} \ || \ CRT \ [\ | | \ CRT \ldots]
\]

\[
\begin{align*}
128 - \text{len}(CRTs) - 3 \text{ bytes} & \quad 3 \text{ bytes} & \quad [22,42] \text{ bytes}
\end{align*}
\]

→ Hybrid between EME and EMSA

\[
\text{CRT: } \text{header} \ || \ \text{key} \ [\ | | \ 91 \ 08 \ \text{iv} ]
\]

\[
\begin{align*}
[6,8] \text{ fixed bytes} & \quad [16,24] \text{ bytes} & \quad 8 \text{ bytes}
\end{align*}
\]

⇒ Only few unknown bytes (compared to the modulus size)
Coppersmith’s Low Exponent Attack

Coppersmith attack:\(^1\)
Recover the message if the unknown part is small enough: we need
\[ x \leq n^{\frac{1}{e}} \]


\[^2\text{European Payments Council. Guidelines on cryptographic algorithms usage and key management. epc342-08, 2018}\]
Coppersmith’s Low Exponent Attack

Coppersmith attack:¹
Recover the message if the unknown part is small enough: we need
\[ x \leq n^{\frac{1}{e}} \]

Assuming the card is using:

- A 1024 bits modulus (RSA-2048 would make it easier)
- A small public exponent² (\( e = 3 \))


²European Payments Council. Guidelines on cryptographic algorithms usage and key management. epc342-08, 2018
Coppersmith’s Low Exponent Attack

Coppersmith attack:\(^1\)
Recover the message if the unknown part is small enough: we need \(x \leq n^{\frac{1}{e}}\)

Assuming the card is using:
- A 1024 bits modulus (RSA-2048 would make it easier)
- A small public exponent\(^2\) \((e = 3)\)

We can recover up to \(\left\lceil \log_2(n^{\frac{1}{3}}) \right\rceil = 341\) bits (\(\approx 42\) bytes)
- An encryption key: 16-24 unknown bytes
- An integrity key (with IV): 26-34 unknown bytes


\(^2\)European Payments Council. Guidelines on cryptographic algorithms usage and key management. epc342-08, 2018
In practice...

- Recover the message in 0.35s on average for a 128 bits key
  \(\Rightarrow\) on-the-fly attack possible
- Passive interception only
- Only works for Key Transport
In practice...

- Recover the message in 0.35s on average for a 128 bits key
  ⇒ on-the-fly attack possible
- Passive interception only
- Only works for Key Transport

⇒ Need a sufficiently big enough public exponent, or random padding
In practice...

- Recover the message in 0.35s on average for a 128 bits key ⇒ on-the-fly attack possible
- Passive interception only
- Only works for Key Transport

⇒ Need a sufficiently big enough public exponent, or random padding

⚠ Bigger RSA modulus is not enough (makes the attack easier)
⚠ "Classic" PKCS#1v1.5 padding may not be a valid solution...
Padding Oracle on Key Transport
Bleichenbacher’s attack

**Abusing** **Perform Security Operation**:  
- Anybody can send this APDU (no authentication before)  
- 3 steps on card: decryption $\rightarrow$ verification $\rightarrow$ TLV parsing  
- Unique error code but no mention of constant time  
- Constant time verification is hard, even harder with TLV parsing
In practice...

- Attack possible with some additional analysis

- Large number of query needed
  - On average 28000 queries $\rightarrow \approx 2h30$
  - Significant communication overhead
  - Can be reduced by increasing brute force
- No on-the-fly attack: message collection for future decryption
In practice...

- Attack possible with some additional analysis

![Graph showing statistical distribution]

- Large number of query needed
  - On average 28000 queries $\rightarrow \approx 2h30$
  - Significant communication overhead
  - Can be reduced by increasing brute force

- No on-the-fly attack: message collection for future decryption

$\Rightarrow$ Need robust RSA padding (OAEP would solve both problems)
In practice...

- Attack possible with some additional analysis

![Graph showing a peak at around 300 and another at 325 with a decrease towards 400]

- Large number of query needed
  - On average 28000 queries $\rightarrow \approx 2h30$
  - Significant communication overhead
  - Can be reduced by increasing brute force

- No on-the-fly attack: message collection for future decryption

$\Rightarrow$ Need robust RSA padding (OAEP would solve both problems)

$\Rightarrow$ Bigger RSA modulus is not enough (makes the attack easier)
Key Reuse
RSA Key Reuse

**Design flaw:**

- Same RSA key for Key Transport and Key Agreement
- Same RSA key for confidentiality and authentication

⇒ Less storage, processing and complexity but no key isolation
RSA Key Reuse

Design flaw:
- Same RSA key for Key Transport and Key Agreement
- Same RSA key for confidentiality and authentication
⇒ Less storage, processing and complexity but no key isolation

Consequences:
- Valid signature forgery using Bleichenbacher’s attack
  - On average 74838 queries → ≈ 7h
- Certificate forgery, signed by the card ⇒ card impersonation in all future sessions
- In case of shared CA, a single forgery may allow impersonating on a large scale
RSA Key Reuse

Design flaw:
- Same RSA key for Key Transport and Key Agreement
- Same RSA key for confidentiality and authentication
⇒ Less storage, processing and complexity but no key isolation

Consequences:
- Valid signature forgery using Bleichenbacher’s attack
  - On average 74838 queries → ≈ 7h
- Certificate forgery, signed by the card ⇒ card impersonation in all future sessions
- In case of shared CA, a single forgery may allow impersonating on a large scale

⇒ Key isolation, at least between confidentiality and authentication
Secure Implementation
Major countermeasures

- Key isolation
  - Significant overhead during certificate verification
  - No need to repeat it at each session
- RSA-OAEP
  - Negligible overhead ($\approx 0.01s$)
- Enforce public exponent $e = 65537$
  - Negligible overhead
  - Not mandatory when using OAEP
- Switching from null to random IV for CBC encryption
  - Negligible overhead
Global Overhead

<table>
<thead>
<tr>
<th>Key Transport, (mutual authentication)</th>
<th>Original</th>
<th>Secure</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cert. verification (card)</td>
<td>0.92</td>
<td>2.06</td>
<td>+124%</td>
</tr>
<tr>
<td>Cert. verification (OCE)</td>
<td>0.15</td>
<td>0.24</td>
<td>+60%</td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>0.68</td>
<td>0.8</td>
<td>+18%</td>
</tr>
<tr>
<td>Internal authentication</td>
<td>0.73</td>
<td>0.71</td>
<td>-3%</td>
</tr>
<tr>
<td>Total</td>
<td>2.76</td>
<td>4.11</td>
<td>+49%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Transport, (external authentication only)</th>
<th>Original</th>
<th>Secure</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cert. verification (card)</td>
<td>1.13</td>
<td>2.44</td>
<td>+116%</td>
</tr>
<tr>
<td>Cert. verification (OCE)</td>
<td>0.15</td>
<td>0.24</td>
<td>+60%</td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>0.72</td>
<td>0.82</td>
<td>+14%</td>
</tr>
<tr>
<td>Total</td>
<td>2.31</td>
<td>3.81</td>
<td>+65%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Agreement</th>
<th>Original</th>
<th>Secure</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cert. verification (card)</td>
<td>1.18</td>
<td>2.12</td>
<td>+80%</td>
</tr>
<tr>
<td>Cert. verification (OCE)</td>
<td>0.15</td>
<td>0.24</td>
<td>+60%</td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>1.61</td>
<td>1.43</td>
<td>-11%</td>
</tr>
<tr>
<td>Internal authentication</td>
<td>0.85</td>
<td>0.80</td>
<td>-6%</td>
</tr>
<tr>
<td>Total</td>
<td>4.09</td>
<td>4.90</td>
<td>+20%</td>
</tr>
</tbody>
</table>

---

Measure done on a NXP J3H145 JCOP3 JavaCard 3.0.4
## Global Overhead

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Secure</th>
<th>Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Transport, (mutual authentication)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>0.68</td>
<td>0.8</td>
<td>+18%</td>
</tr>
<tr>
<td>Internal authentication</td>
<td>0.73</td>
<td>0.71</td>
<td>-3%</td>
</tr>
<tr>
<td>Total</td>
<td>1.56</td>
<td>1.67</td>
<td>+7%</td>
</tr>
<tr>
<td><strong>Key Transport, (external authentication only)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>0.72</td>
<td>0.82</td>
<td>+14%</td>
</tr>
<tr>
<td>Total</td>
<td>0.87</td>
<td>0.98</td>
<td>+13%</td>
</tr>
<tr>
<td><strong>Key Agreement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSO (decipher)</td>
<td>0.15</td>
<td>0.16</td>
<td>+6%</td>
</tr>
<tr>
<td>External authentication</td>
<td>1.61</td>
<td>1.43</td>
<td>-11%</td>
</tr>
<tr>
<td>Internal authentication</td>
<td>0.85</td>
<td>0.80</td>
<td>-6%</td>
</tr>
<tr>
<td>Total</td>
<td>2.61</td>
<td>2.39</td>
<td>-10%</td>
</tr>
</tbody>
</table>

1Measure done on a NXP J3H145 JCOP3 JavaCard 3.0.4
Conclusion
We tried to apply well known attack to the smart cards world

Successfully performed two attacks speculating on the implementation
  - We believe our assumption to be reasonable giving past attacks
  - Key isolation is not implementation dependent

Suggest mitigations:
  - Easy to add in the specification
  - Reasonable overhead

GlobalPlatform is taking our recommendations into account
Thank you for your attention!

1. Context
2. Notation & Reminders
3. Deterministic RSA Padding
4. Padding Oracle on Key Transport
5. Key Reuse
6. Secure Implementation
7. Conclusion