The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10

Daniel De Almeida Braga
Pierre-Alain Fouque
Mohamed Sabt
TCHES 2020
Overview

- Context
- Deterministic RSA Padding
- Padding Oracle
- Key Reuse
- Secure Implementation
- Conclusion
Context
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
• SCP 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
- 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

Off-Card Entity (OCE)
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

- Certificate exchange
- Manage Security Environment
- Applet Selection
Key Exchange Modes

(a) Key Transport mode

OCE

Card

- Applet Selection
- Manage Security Environment
- Certificate exchange
- Perform Security Operation (dec)
Key Exchange Modes

(a) Key Transport mode
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 to forge a certificate, signed by the card
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 contributions:

1. Abuse blur 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 to forge a certificate, signed by the card
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 attackers 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

They cannot:

× Have physical access to the card
× Break the cryptographic primitives
Deterministic RSA Padding
Perform Security Operation APDU:

\[ M: \text{params} \, || \, \text{CRT} \, \, [|| \text{CRT} \, \ldots ] \]

3 bytes \quad [22,42] \text{ bytes}
Perform Security Operation APDU:

M: \text{params} \parallel \text{CRT} \parallel \ldots
\begin{align*}
\text{3 bytes} & \quad [22,42] \text{ bytes} \\
\end{align*}

\text{CRT}: \quad \text{header} \parallel \text{key} \parallel 91 \ 08 \ \text{iv}
\begin{align*}
[6,8] \text{ fixed bytes} & \quad [16,24] \text{ bytes} & \quad 8 \text{ bytes} \\
\end{align*}
Perform Security Operation APDU:

\[ M: \text{params} \ |\ | \text{CRT} \ [\ |\ | \text{CRT} \ldots] \]

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

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

→ Hybrid padding (mixing EME and EMSA)
Perform Security Operation APDU:

\[
M: \text{params} || \text{CRT} [|| \text{CRT} \ldots] \\
3 \text{ bytes} \quad [22,42] \text{ bytes}
\]

\[
\text{CRT: header} || \text{key} [|| 91 \ 08 \ \text{iv}] \\
[6,8] \text{ fixed bytes} \quad [16,24] \text{ bytes} \quad 8 \text{ bytes}
\]

\[
\text{EM: 0002 || FF..FF || 00 || M} \\
128-\text{len}(M)-3 \text{ bytes}
\]

→ Hybrid padding (mixing EME and EMSA)

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

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

---

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

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
- A small public exponent$^1$ ($e = 3$)

---

$^1$European Payments Council. Guidelines on cryptographic algorithms usage and key management. epc342-08, 2018
Coppersmith’s Low Exponent Attack

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
- A small public exponent\(^1\) ($e = 3$)

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

\(^1\)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 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 big enough public exponent, or random padding

⚠️ Bigger RSA modulus makes the attack easier

⚠️ "Classic" PKCS#1v1.5 padding may not be a valid solution...
Padding Oracle
Bleichenbacher’s attack

Abusing **Perform Security Operation**:

- Anybody can send this APDU (no authentication before)
Bleichenbacher’s attack

Abusing **Perform Security Operation**:

- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → TLV parsing
Bleichenbacher’s attack

Abusing **PERFORM SECURITY OPERATION**:

- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → TLV parsing
- Unique error code but no mention of constant time
Bleichenbacher’s attack

Abusing **Perform Security Operation**:

- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → 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
  - Average: 28000 queries ≈ 2h30
  - 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

- Large number of query needed
  - Average: 28000 queries \( \approx 2h30 \)
  - Can be reduced by increasing brute force

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

\[ \Rightarrow \text{Need robust RSA padding (OAEP would solve both problems)} \]
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
  ⇒ Need key isolation
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

⇒ Need key isolation
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
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 released a new standard version based on our recommendations
Thank you for your attention!