The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10

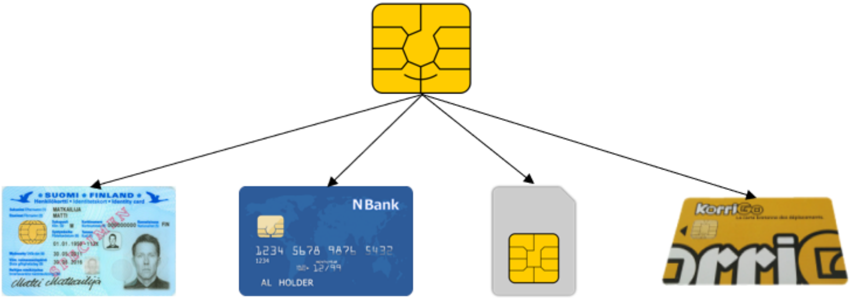
Daniel De Almeida Braga
Pierre-Alain Fouque
Mohamed Sabt

April, 9th 2020

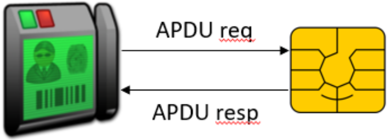
Context

The smart card world

The smart card world

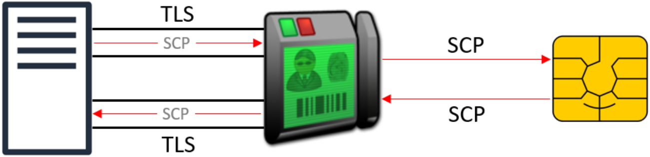


SCP (Secure Communication Protocol)

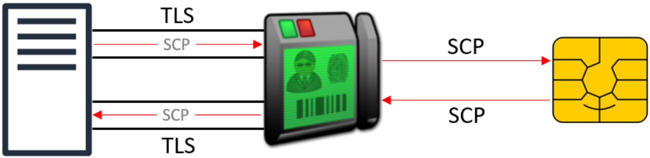


SCP (Secure Communication Protocol)

SCP (Secure Communication Protocol)

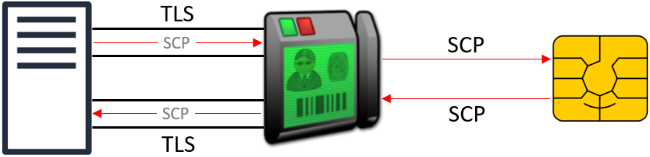


SCP (Secure Communication Protocol)



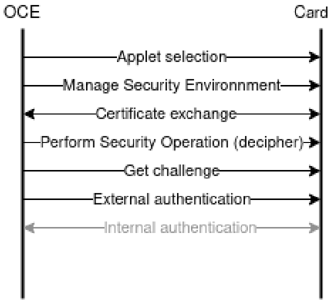
- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication

SCP (Secure Communication Protocol)



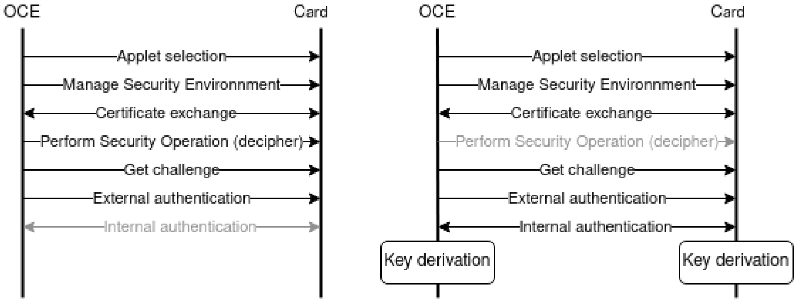
- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication
- SCP10 relies on a Public Key Infrastructure:
  - Both the card and off-card entity have a key pair
  - They use each other public key to encrypt/verify messages

Key Exchange Modes



(a) Key Transport mode

Key Exchange Modes



(a) Key Transport mode (b) Key Agreement mode

Our contributions

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

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:

$pub = (n,\ e)$
$priv = (n,\ d)$

$$\textit{Encryption:}\quad c = m^e \mod n,$$
$$\textit{Decryption:}\quad m = c^d \mod n.$$

$$\textit{Signature:}\quad s = RSA_{sign}(m, priv),$$
$$\textit{Verification:}\quad m == RSA_{ver}(m, pub)\ ?$$

RSA and padding

RSA:

$pub = (n,\ e)$
$priv = (n,\ d)$

$$\textit{Encryption:} \quad c = m^e \mod n, \qquad \textit{Signature:} \quad s = RSA_{sign}(m, priv),$$
$$\textit{Decryption:} \quad m = c^d \mod n. \qquad \textit{Verification:} \quad m == RSA_{ver}(m, pub)\ ?$$

PKCS#1v1.5 padding:

$$\textit{Enc:}\ \texttt{EME-PKCS1-v1\_5(m)} = \texttt{0x00 || 0x02 ||} \underbrace{\texttt{PS}}_{\text{random bytes}} \texttt{|| 0x00 || m}$$

$$\textit{Sig:}\ \texttt{EMSA-PKCS1-v1\_5(m)} = \texttt{0x00 || 0x01 || 0xFF..FF || 0x00 || m}$$

Deterministic RSA Padding

Perform Security Operation

Perform Security Operation APDU:

```
M: params || CRT [|| CRT]
```

Perform Security Operation

Perform Security Operation APDU:

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

$$\texttt{EM: } \underbrace{\texttt{0002 || }\textcolor{orange}{\texttt{FF..FF}}\texttt{ || 00}}_{128-len(CRTs)-3\texttt{ bytes}} \texttt{ || } \underbrace{\texttt{params}}_{\texttt{3 bytes}} \texttt{ || } \underbrace{\texttt{CRT}}_{\texttt{[22,42] bytes}} \texttt{ [|| CRT ...]}$$

$\rightarrow$ Hybrid between EME and EMSA

Perform Security Operation

Perform Security Operation APDU:

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

$$\texttt{EM: } \underbrace{\texttt{0002 || FF..FF || 00}}_{128-len(CRTs)-3\ \texttt{bytes}} \texttt{ || } \underbrace{\texttt{params}}_{\texttt{3 bytes}} \texttt{ || } \underbrace{\texttt{CRT}}_{\texttt{[22,42] bytes}} \texttt{ [|| CRT ...]}$$

→ Hybrid between EME and EMSA

$$\texttt{CRT: } \underbrace{\texttt{header}}_{\texttt{[6,8] fixed bytes}} \texttt{ || } \underbrace{\texttt{key}}_{\texttt{[16,24] bytes}} \texttt{ [|| 91 08 } \underbrace{\texttt{iv}}_{\texttt{8 bytes}} \texttt{]}$$

Perform Security Operation

Perform Security Operation APDU:

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

$$\texttt{EM: } \underbrace{\texttt{0002 || FF..FF || 00}}_{128-len(CRTs)-3\texttt{ bytes}} \texttt{ || } \underbrace{\texttt{params}}_{\texttt{3 bytes}} \texttt{ || } \underbrace{\texttt{CRT}}_{\texttt{[22,42] bytes}} \texttt{ [|| CRT ...]}$$

→ Hybrid between EME and EMSA

$$\texttt{CRT: } \underbrace{\texttt{header}}_{\texttt{[6,8] fixed bytes}} \texttt{ || } \underbrace{\texttt{key}}_{\texttt{[16,24] bytes}} \texttt{ [|| 91 08 } \underbrace{\texttt{iv}}_{\texttt{8 bytes}} \texttt{]}$$

⇒ 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 $x \leq n^{\frac{1}{e}}$

[1] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10(4):233–260, 1997

[2] 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 exponent[2] ($e = 3$)

[1] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10(4):233–260, 1997

[2] 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 exponent[2] ($e = 3$)

We can recover up to $\left\lceil \log_2(n^{\frac{1}{3}}) \right\rceil = 341$ bits (≈ 42 bytes)

- An encryption key: 16-24 unknown bytes
- An integrity key (with IV): 26-34 unknown bytes

[1] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10(4):233–260, 1997

[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
  $\Rightarrow$ on-the-fly attack possible
- Passive interception only
- Only works for Key Transport

$\Rightarrow$ 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
  $\Rightarrow$ on-the-fly attack possible
- Passive interception only
- Only works for Key Transport

$\Rightarrow$ Need a sufficiently big enough public exponent, or random padding

⚠ Bigger RSA modulus is not enough (makes the attack easier)

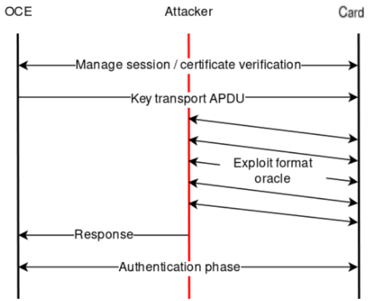
⚠ "Classic" PKCS#1v1.5 padding may not be a valid solution...

Padding Oracle on Key Transport

Bleichenbacher's attack

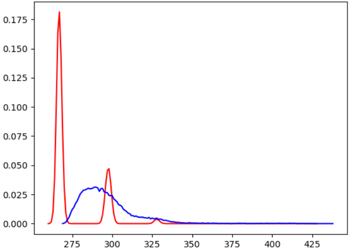
Abusing PERFORM SECURITY OPERATION:

- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → TLV parsing
- Unique error code but no mention of constant time
- Constant time verification is hard, even harder with TLV parsing



In practice...

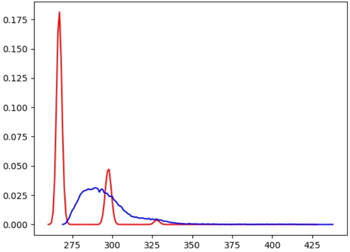
- Attack possible with some additional analysis



- Large number of query needed
  - 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

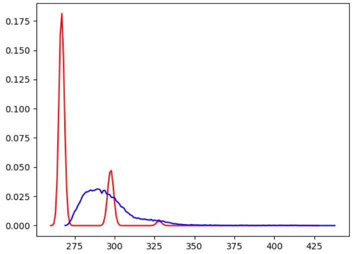


- Large number of query needed
  - On average 28000 queries $\rightarrow \approx$ 2h30
  - Significant communication overhead
  - Can be reduced by increasing brute force
- No on-the-fly attack: message collection for future decryption

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

In practice...

- Attack possible with some additional analysis



- 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)

⚠ 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

$\Rightarrow$ 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

$\Rightarrow$ Less storage, processing and complexity but no key isolation

Consequences:

- Valid signature forgery using Bleichenbacher's attack
  - On average 74838 queries $\rightarrow \approx$ 7h
- Certificate forgery, signed by the card $\Rightarrow$ 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

$\Rightarrow$ Less storage, processing and complexity but no key isolation

Consequences:

- Valid signature forgery using Bleichenbacher's attack
  - On average 74838 queries $\rightarrow \approx$ 7h
- Certificate forgery, signed by the card $\Rightarrow$ card impersonation in all future sessions
- In case of shared CA, a single forgery may allow impersonating on a large scale

$\Rightarrow$ 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[1]

		Original	Secure	Diff.
Key Transport, (mutual authentication)	Cert. verification (card)	0.92	2.06	+124%
	Cert. verification (OCE)	0.15	0.24	+60%
	PSO (decipher)	0.15	0.16	+6%
	External authentication	0.68	0.8	+18%
	Internal authentication	0.73	0.71	-3%
	Total	**2.76**	**4.11**	**+49%**
Key Transport, (external authentication only)	Cert. verification (card)	1.13	2.44	+116%
	Cert. verification (OCE)	0.15	0.24	+60%
	PSO (decipher)	0.15	0.16	+6%
	External authentication	0.72	0.82	+14%
	Total	**2.31**	**3.81**	**+65%**
Key Agreement	Cert. verification (card)	1.18	2.12	+80%
	Cert. verification (OCE)	0.15	0.24	+60%
	PSO (decipher)	0.15	0.16	+6%
	External authentication	1.61	1.43	-11%
	Internal authentication	0.85	0.80	-6%
	Total	**4.09**	**4.90**	**+20%**

[1] Measure done on a NXP J3H145 JCOP3 JavaCard 3.0.4

Global Overhead[1]

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

[1] Measure done on a NXP J3H145 JCOP3 JavaCard 3.0.4

Conclusion

Sum-up

- 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 !