# The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10

**Daniel De Almeida Braga**
Pierre-Alain Fouque
Mohamed Sabt

TCHES 2020

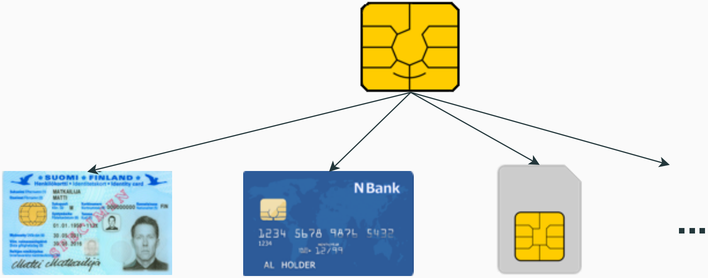
# Overview

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

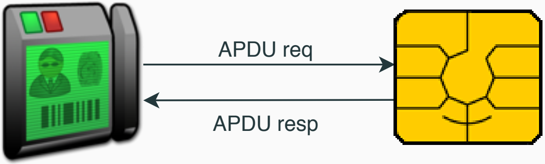
# Context

# The smart card world

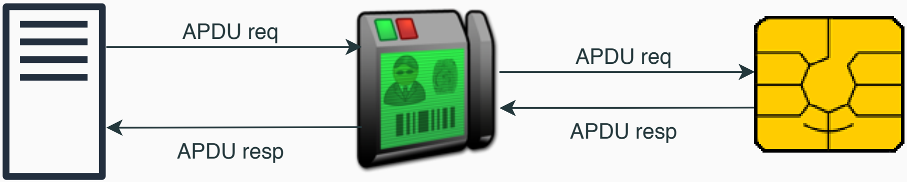
# The smart card world



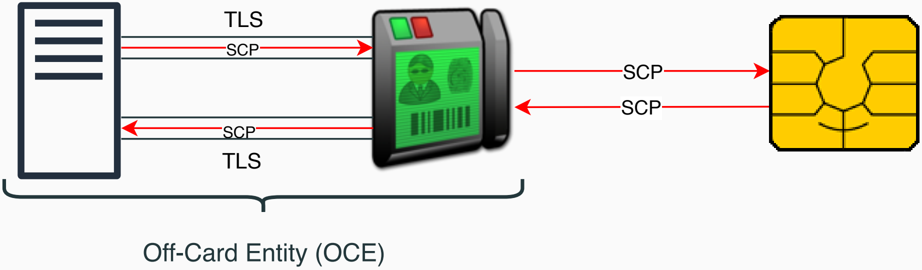
# SCP (Secure Communication Protocol)



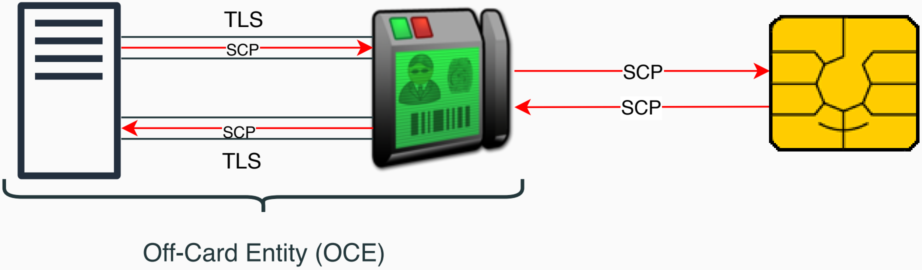
# SCP (Secure Communication Protocol)



# SCP (Secure Communication Protocol)

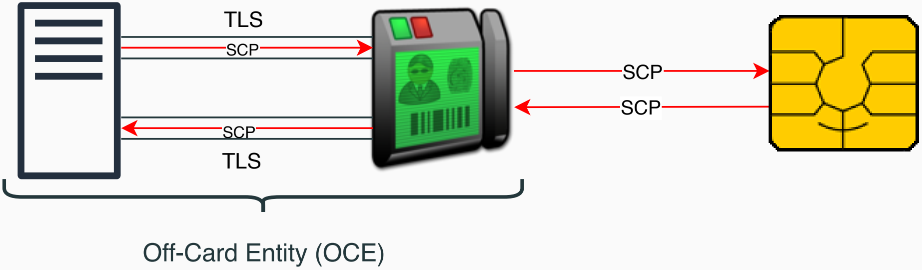


# SCP (Secure Communication Protocol)



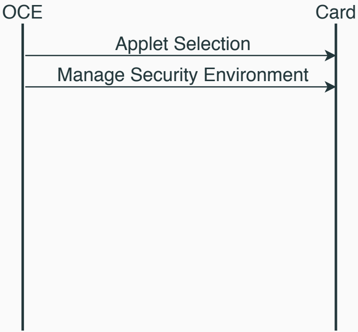

- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication

# SCP (Secure Communication Protocol)



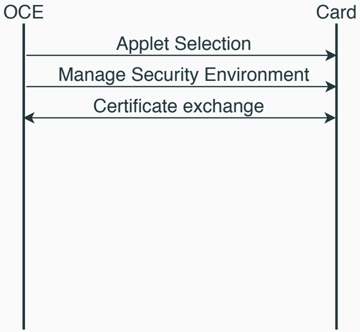

- Establish a secure session between a card and an Off-Card Entity
- 2-steps protocol: Key Exchange + Communication
- SCP10 relies on a Public Key Infrastructure:
  - Both the card and off-card entity have a key pair
  - They use each other public key to encrypt/verify messages

# Key Exchange Modes



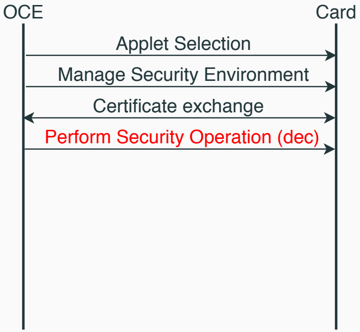

**(a)** Key Transport mode

# Key Exchange Modes



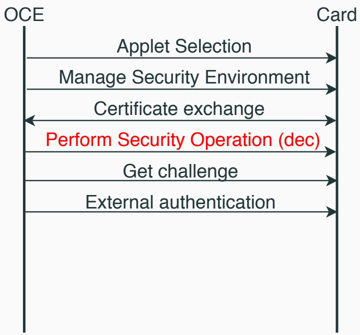

**(a)** Key Transport mode

# Key Exchange Modes



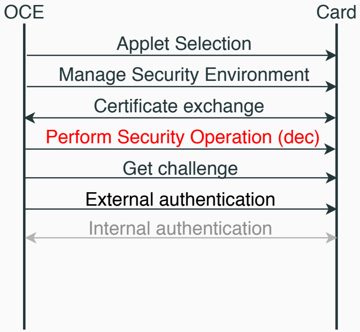

**(a)** Key Transport mode

# Key Exchange Modes



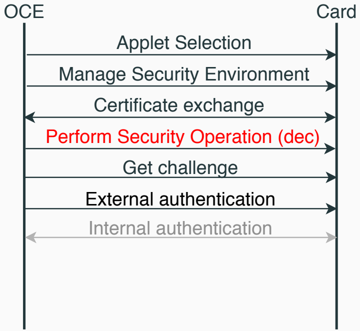

**(a)** Key Transport mode

# Key Exchange Modes

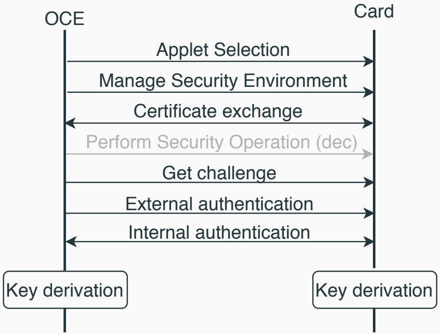



**(a)** Key Transport mode

# Key Exchange Modes




(a) Key Transport mode




(b) Key Agreement mode

# Our contributions

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

# 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 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

PERFORM SECURITY OPERATION APDU:

```
M: params || CRT [|| CRT ...]
   3 bytes   [22,42] bytes
```

# Perform Security Operation

PERFORM SECURITY OPERATION APDU:

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

- params: 3 bytes
- CRT: [22,42] bytes

```
CRT: header || key [|| 91 08 iv ]
```

- header: [6,8] fixed bytes
- key: [16,24] bytes
- iv: 8 bytes

# Perform Security Operation

PERFORM SECURITY OPERATION APDU:

M: $\underbrace{\texttt{params}}_{\texttt{3 bytes}}$ || $\underbrace{\texttt{CRT}}_{\texttt{[22,42] bytes}}$ [|| CRT ...]

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

EM: $\underbrace{\texttt{0002 || FF..FF || 00}}_{128-len(M)-3\ \texttt{bytes}}$ || M

→ Hybrid padding (mixing EME and EMSA)

# Perform Security Operation

PERFORM SECURITY OPERATION APDU:

$$\texttt{M: } \underbrace{\texttt{params}}_{\texttt{3 bytes}} \texttt{ || } \underbrace{\texttt{CRT}}_{\texttt{[22,42] bytes}} \texttt{ [|| CRT ...]}$$

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

$$\texttt{EM: } \underbrace{\texttt{0002 || FF..FF || 00}}_{128-len(M)-3 \texttt{ bytes}} \texttt{ || M}$$

→ 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

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

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

⚠ Bigger RSA modulus makes the attack easier

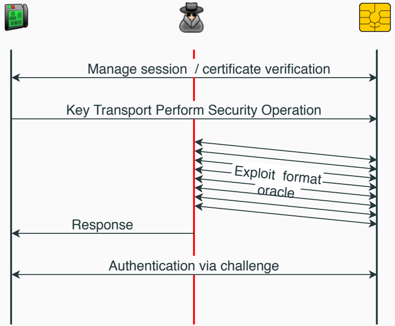
⚠ "Classic" PKCS#1v1.5 padding may not be a valid solution...

# Padding Oracle

# Bleichenbacher's attack

Abusing Perform Security Operation:

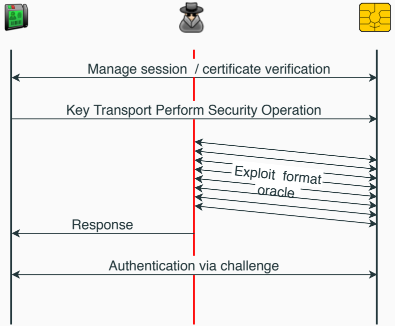
- Anybody can send this APDU (no authentication before)



# Bleichenbacher's attack

Abusing Perform Security Operation:

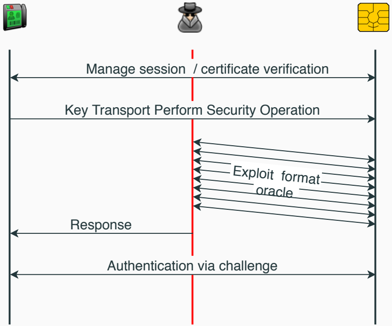
- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → TLV parsing



# Bleichenbacher's attack

Abusing PERFORM SECURITY OPERATION:

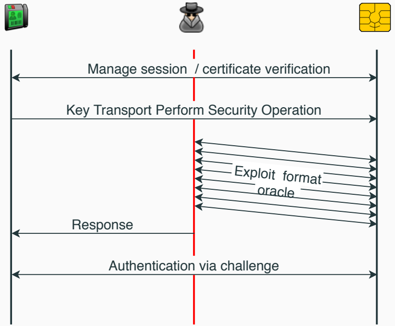
- Anybody can send this APDU (no authentication before)
- 3 steps on card: decryption → verification → TLV parsing
- Unique error code but no mention of constant time



# Bleichenbacher's attack

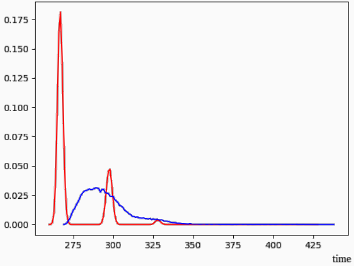
Abusing Perform Security Operation:

- 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...

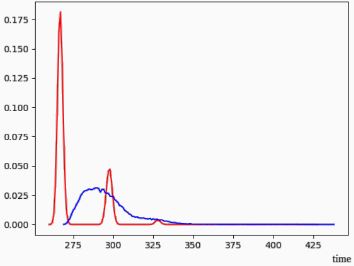
- Attack possible with some additional analysis




- 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

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

# 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 $\approx$ 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 $\approx$ 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

$\Rightarrow$ 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 ($\approx 0.01$s)
- Enforce public exponent $e = 65537$
  - Negligible overhead
  - Not mandatory when using OAEP

# 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 released a new standard version based on our recommendations

**Thank you for your attention !**