Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack

Gildas Avoine[1,2] **Loïc Ferreira**[3,1]

Univ Rennes, INSA Rennes, CNRS, IRISA, France

Institut Universitaire de France

Orange Labs, Applied Cryptography Group, Caen, France

September 12, 2018

1. Description of SCP02
2. Padding oracle attack
3. Experimental results
4. Conclusion

Context

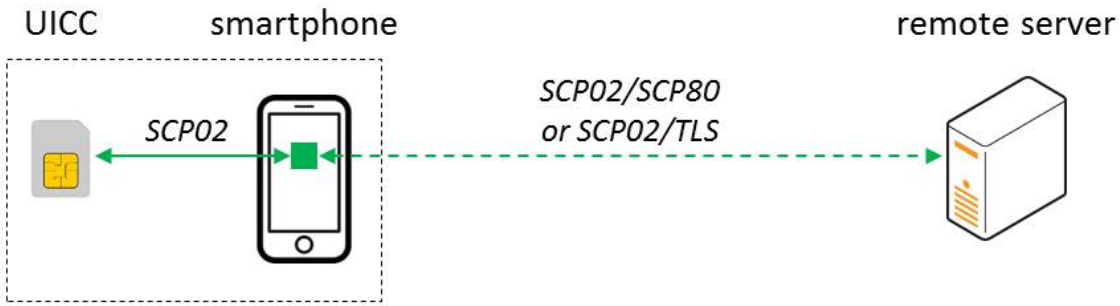
- Security protocol promoted by GlobalPlatform (association that aims at promoting standard, interoperable mechanisms related to the chip technology)
- Element of a set of security protocols: SCP03, SCP80, SCP81, etc.
- Likely the most widely used SCP protocol

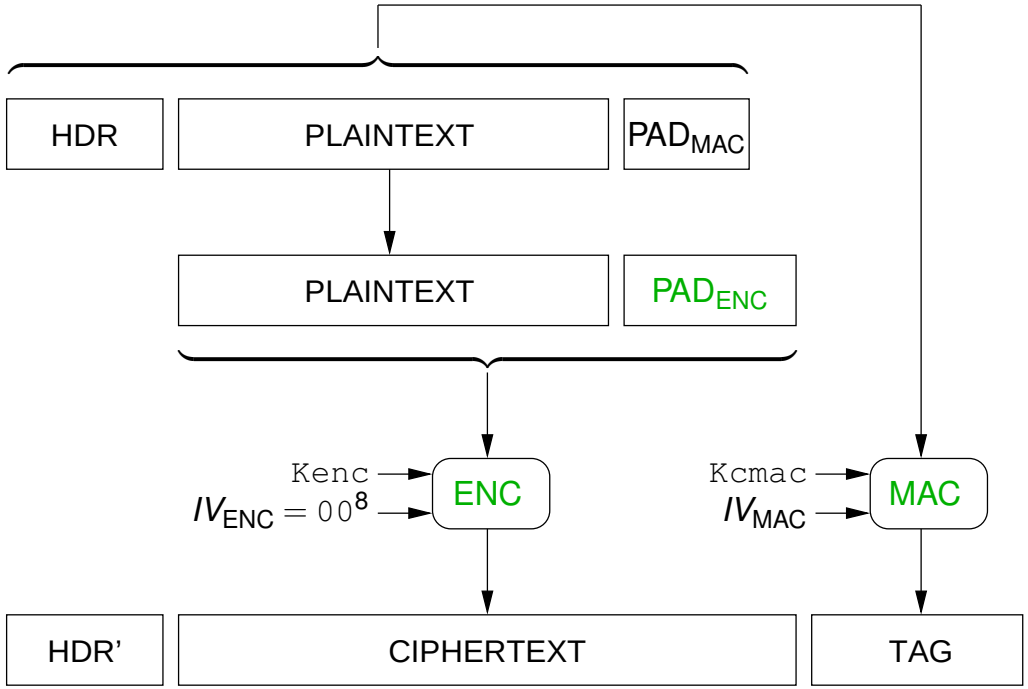
Cryptographic functions

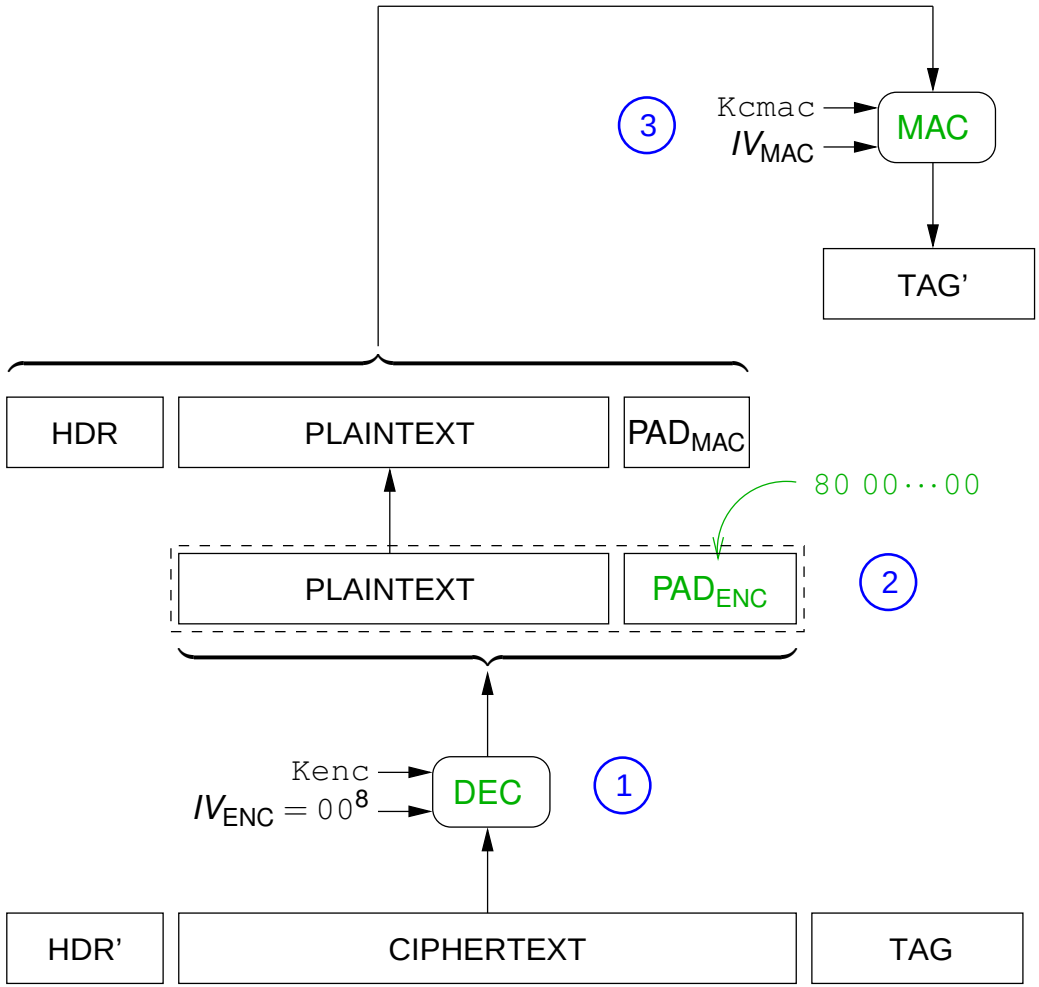
- Based on DES/3DES (encryption and MAC; cf. [ISO9797-1] and [ISO10116])

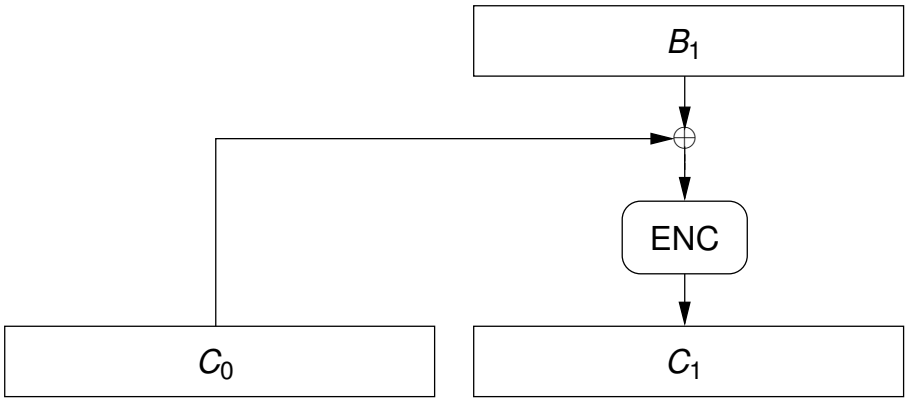
Purpose

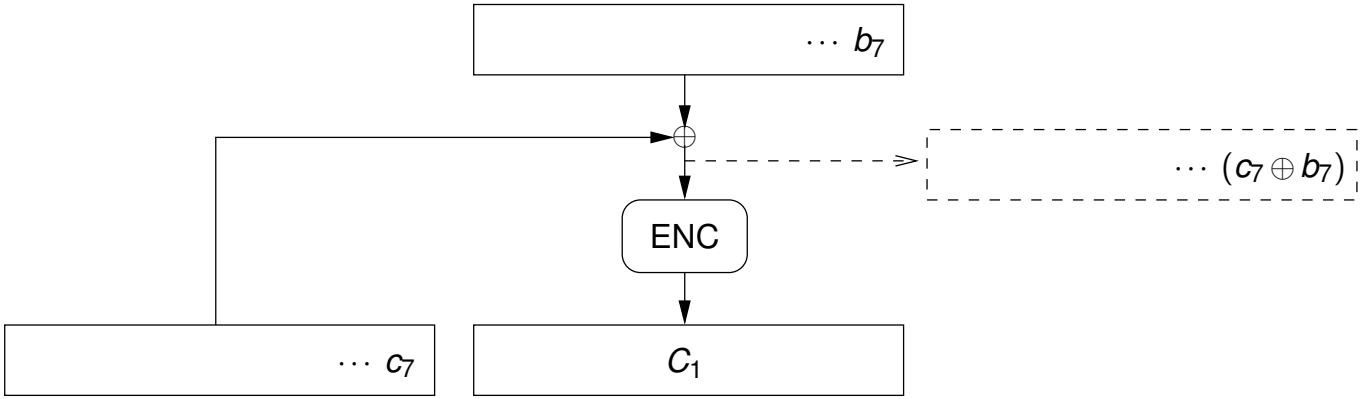
- Secure channel between an "*off card entity*" and a card
- Different security levels: integrity, confidentiality, both
- Remote card management (e.g., applet upload into an UICC/SIM card)

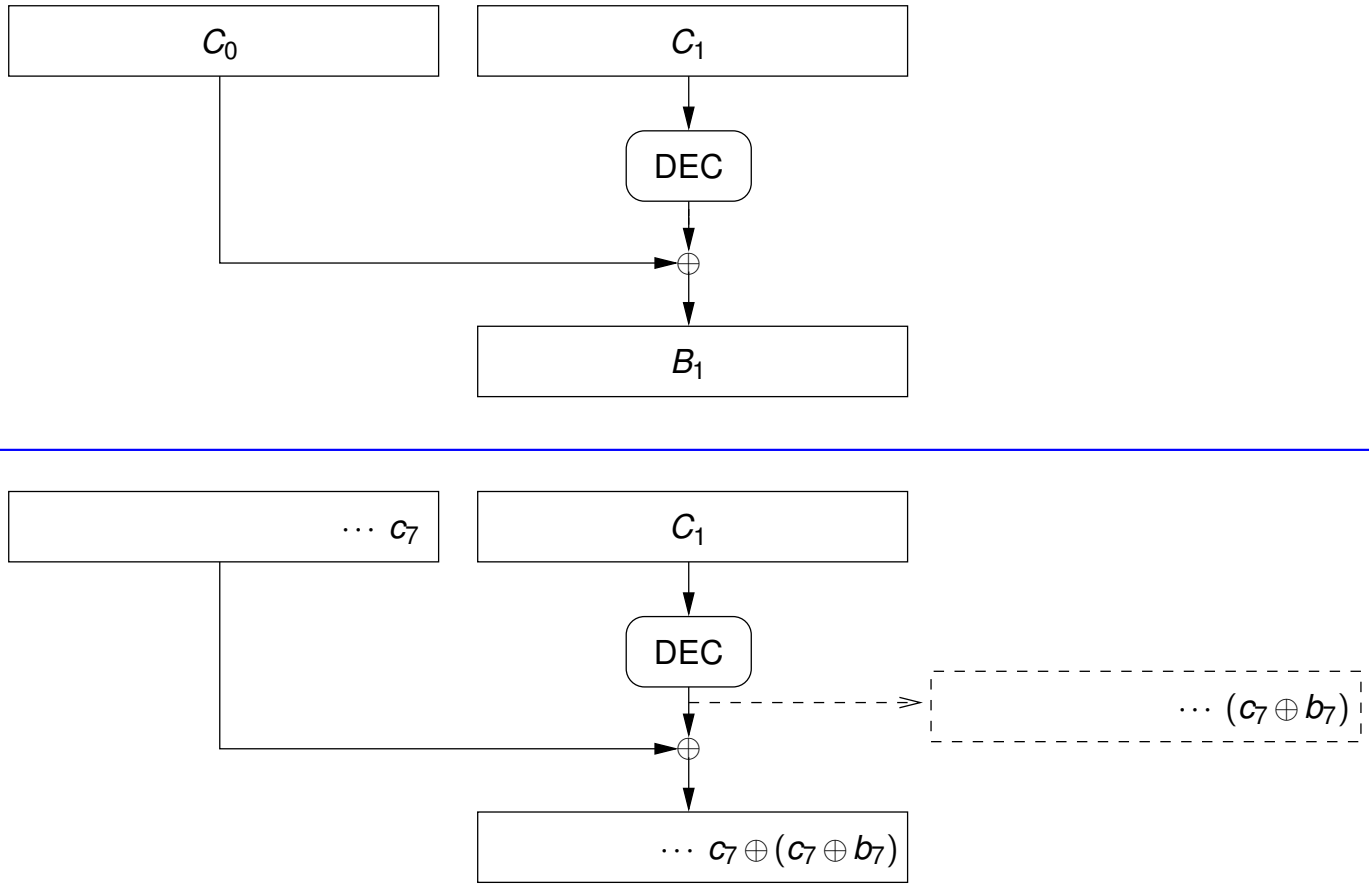


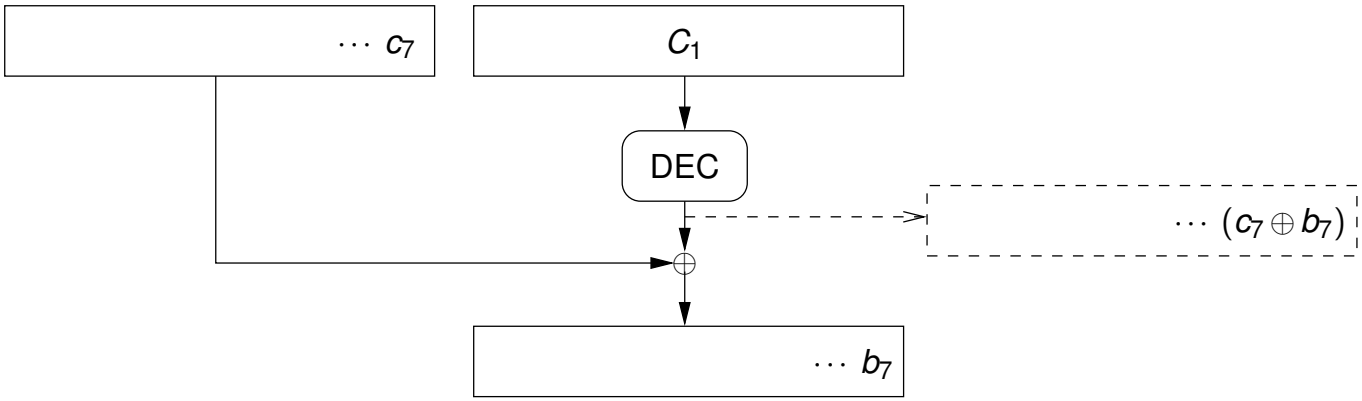

HDR
PLAINTEXT
PAD_MAC
PLAINTEXT
PAD_ENC
Kenc
IV_ENC = 00^8
ENC
Kcmac
IV_MAC
MAC
HDR'
CIPHERTEXT
TAG

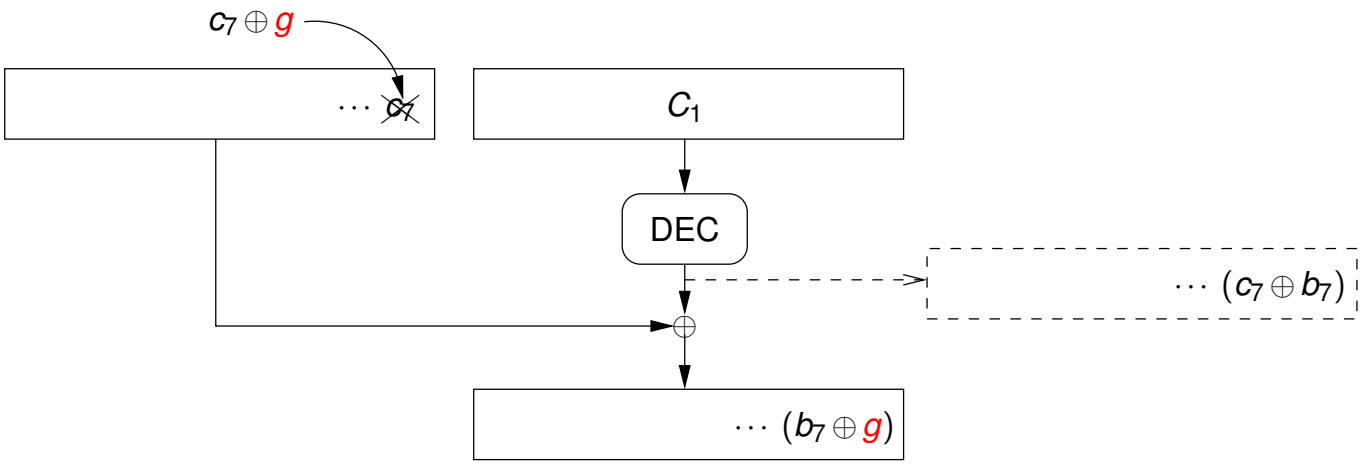

3
Kcmac
IV_{MAC}
MAC
TAG'
HDR
PLAINTEXT
PAD_{MAC}
80 00···00
PLAINTEXT
PAD_{ENC}
2
Kenc
$IV_{ENC} = 00^8$
DEC
1
HDR'
CIPHERTEXT
TAG

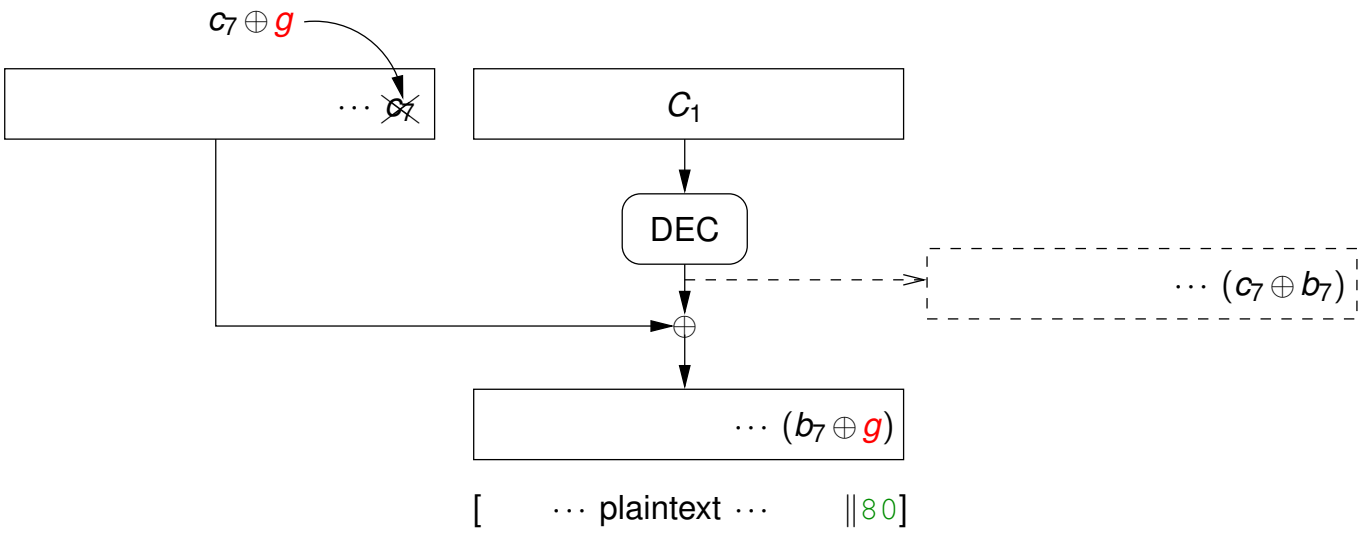

B_1
ENC
C_0
C_1

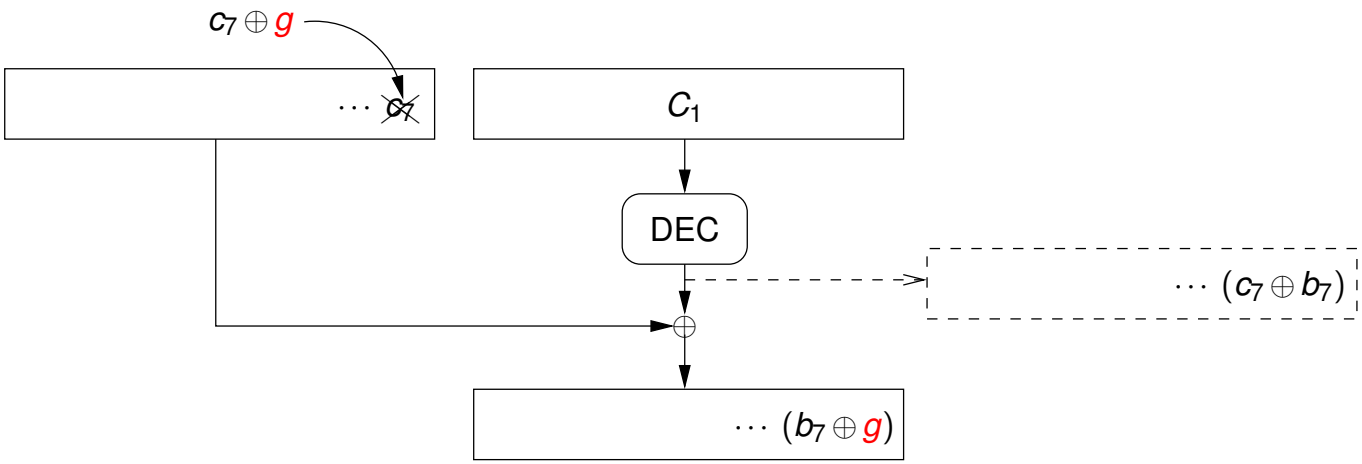

$\cdots b_7$
$\cdots (c_7 \oplus b_7)$
ENC
$\cdots c_7$
C_1


C_0
C_1
DEC
B_1
$\cdots c_7$
C_1
DEC
$\cdots (c_7 \oplus b_7)$
$\cdots c_7 \oplus (c_7 \oplus b_7)$

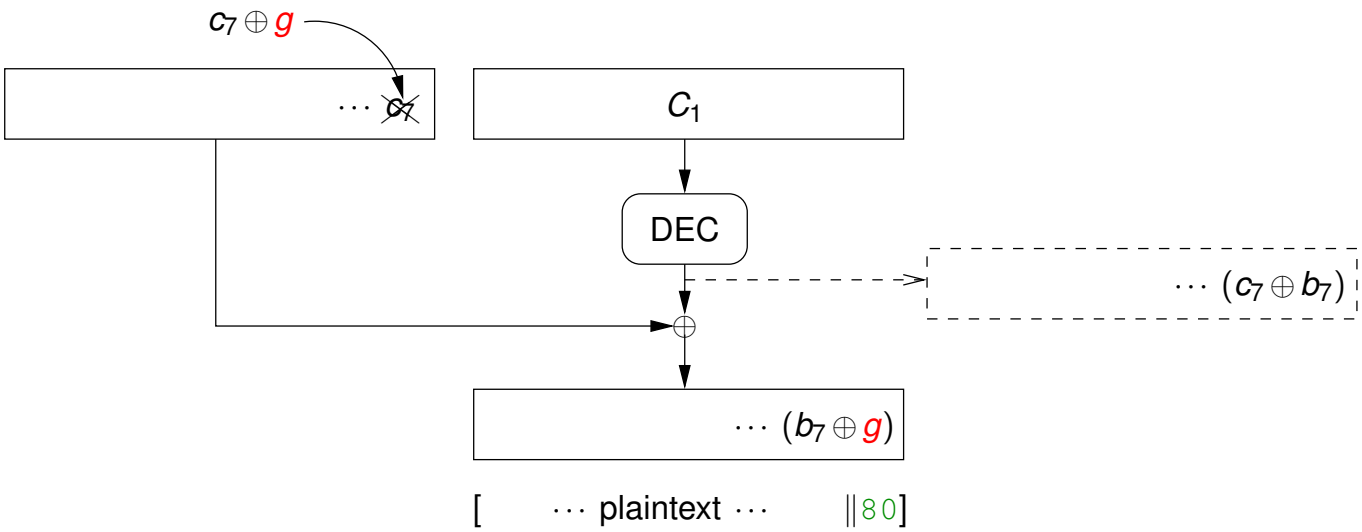

$\cdots c_7$
C_1
DEC
$\cdots (c_7 \oplus b_7)$
$\cdots b_7$


$c_7 \oplus g$
$\cdots$ c_7
C_1
DEC
$\cdots$ $(c_7 \oplus b_7)$
$\cdots$ $(b_7 \oplus g)$

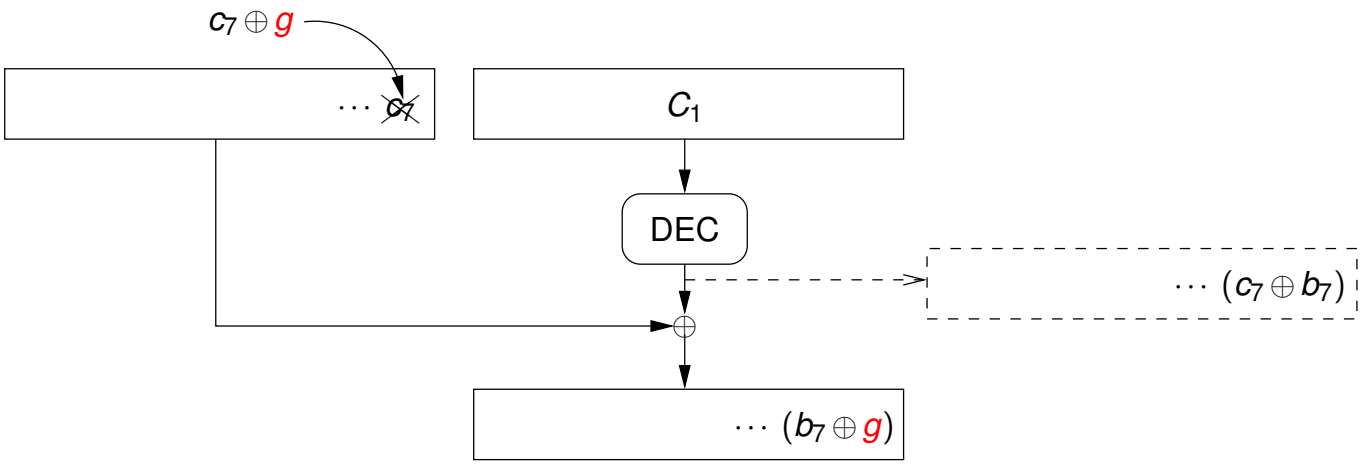

$c_7 \oplus g$
$\cdots$ c_7
C_1
DEC
$\cdots$ $(c_7 \oplus b_7)$
$\cdots$ $(b_7 \oplus g)$
[$\cdots$ plaintext $\cdots$ ‖80]



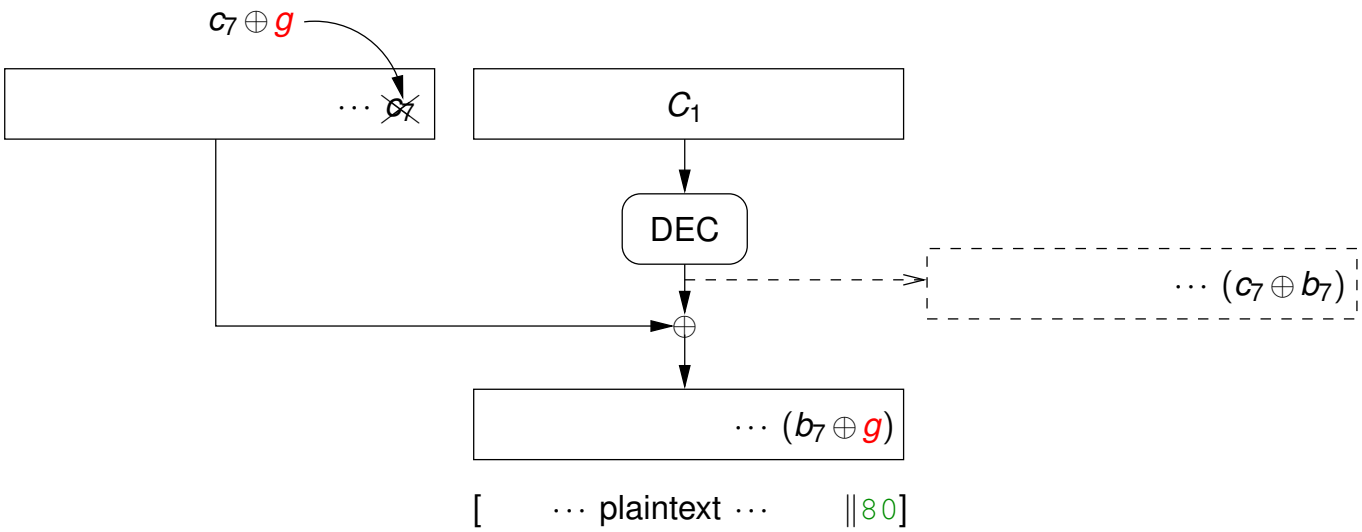
$g = 13$ $\Rightarrow$ $\Rightarrow$ invalid padding



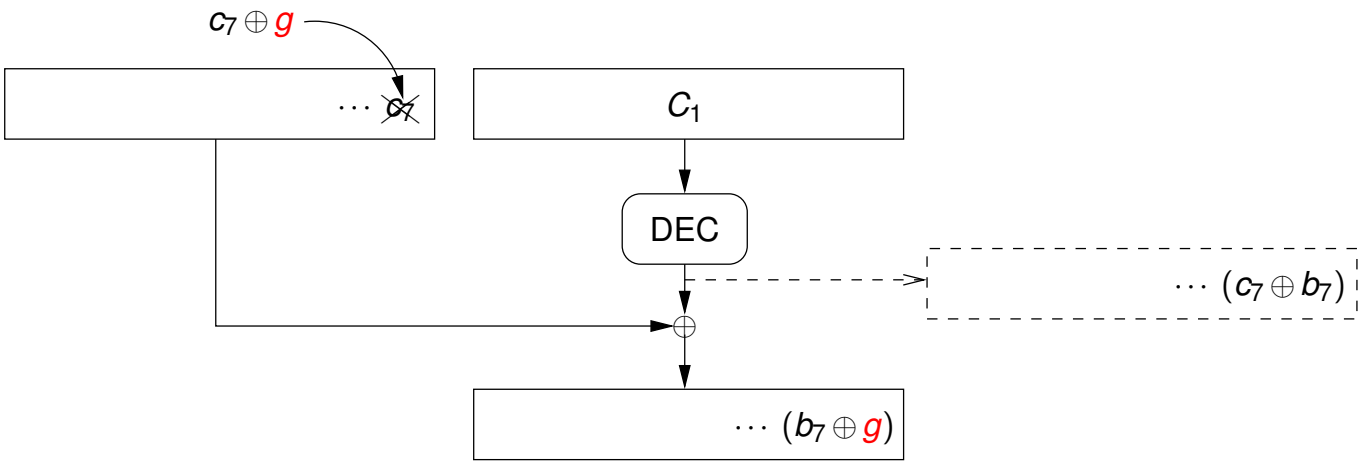
$g = 13$ $\Rightarrow$ $\Rightarrow$ invalid padding
$g = 14$ $\Rightarrow$ $\Rightarrow$ invalid padding



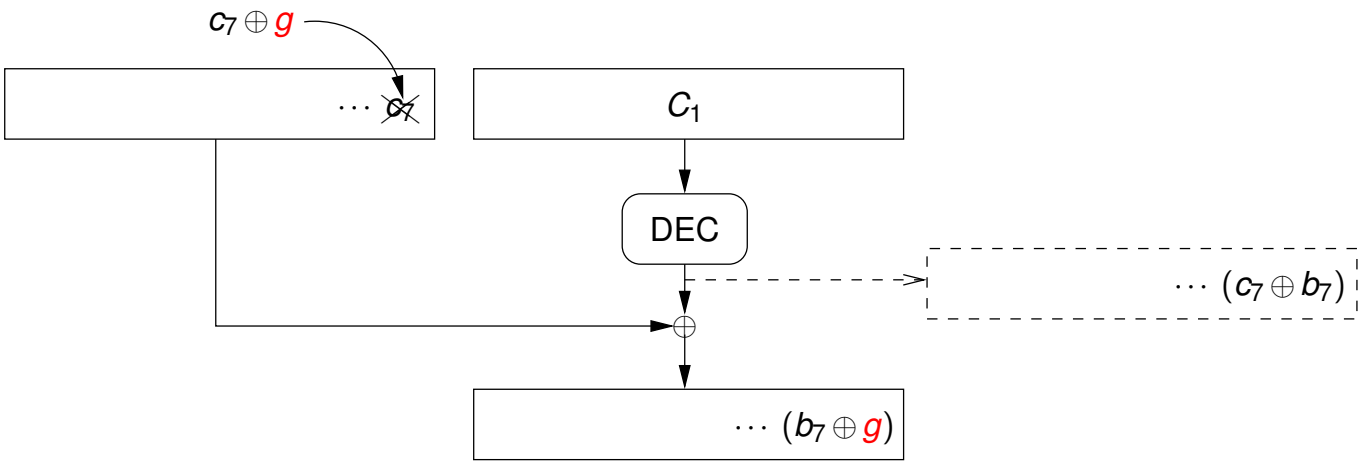
$g = 13$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 14$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 15$	$\Rightarrow$		$\Rightarrow$	invalid padding



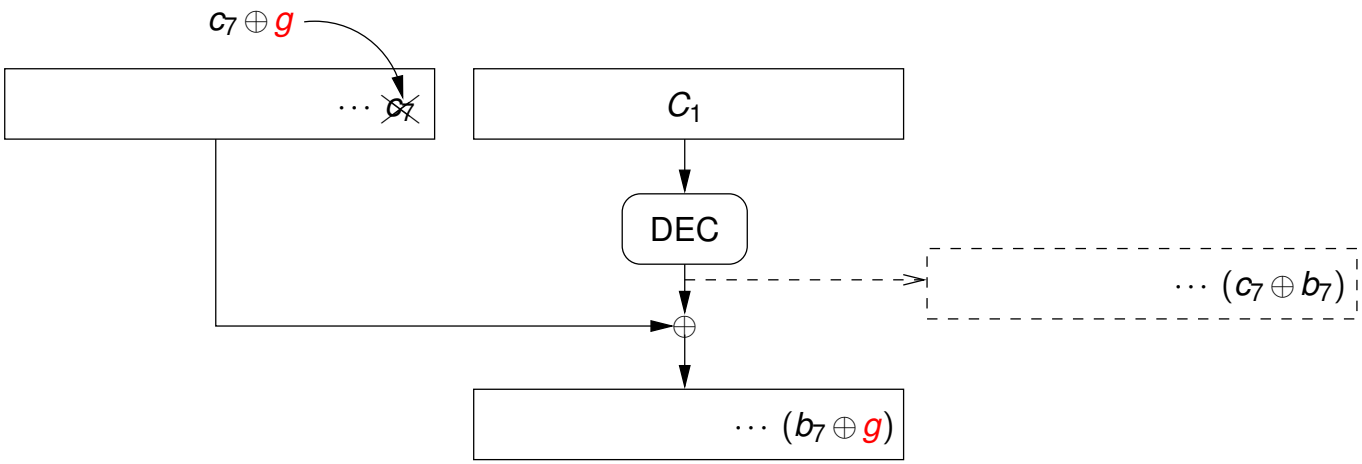
$g = 13$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 14$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 15$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 16$	$\Rightarrow$		$\Rightarrow$	invalid padding



$g = 13$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 14$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 15$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 16$	$\Rightarrow$		$\Rightarrow$	invalid padding
$g = 17$	$\Rightarrow$		$\Rightarrow$	valid padding



$g = 13$	$\Rightarrow$	$b_7 \oplus g = 76$	$\Rightarrow$	invalid padding		
$g = 14$	$\Rightarrow$	$b_7 \oplus g = 77$	$\Rightarrow$	invalid padding		
$g = 15$	$\Rightarrow$	$b_7 \oplus g = 78$	$\Rightarrow$	invalid padding		
$g = 16$	$\Rightarrow$	$b_7 \oplus g = 79$	$\Rightarrow$	invalid padding		
$g = 17$	$\Rightarrow$	$b_7 \oplus g = 80$	$\Rightarrow$	valid padding	$\Rightarrow$	$b_7 = g \oplus 80 = 97$



$g = 13$	$\Rightarrow$	$b_7 \oplus g = 76$	$\Rightarrow$	invalid padding		
$g = 14$	$\Rightarrow$	$b_7 \oplus g = 77$	$\Rightarrow$	invalid padding		
$g = 15$	$\Rightarrow$	$b_7 \oplus g = 78$	$\Rightarrow$	invalid padding		
$g = 16$	$\Rightarrow$	$b_7 \oplus g = 79$	$\Rightarrow$	invalid padding		
$g = 17$	$\Rightarrow$	$b_7 \oplus g = 80$	$\Rightarrow$	valid padding	$\Rightarrow$	$b_7 = g \oplus 80 = 97$

- The validity of padding data indicates whether b_7 can be found or not.
- Technique called “padding oracle attack” due to Vaudenay in 2002 [V02].

- How to know if the padding data is valid or invalid (after decryption)?

	1. decryption 2. padding data: invalid 3. ~~MAC~~	1. decryption 2. padding data: valid 3. MAC

- How to know if the padding data is valid or invalid (after decryption)?

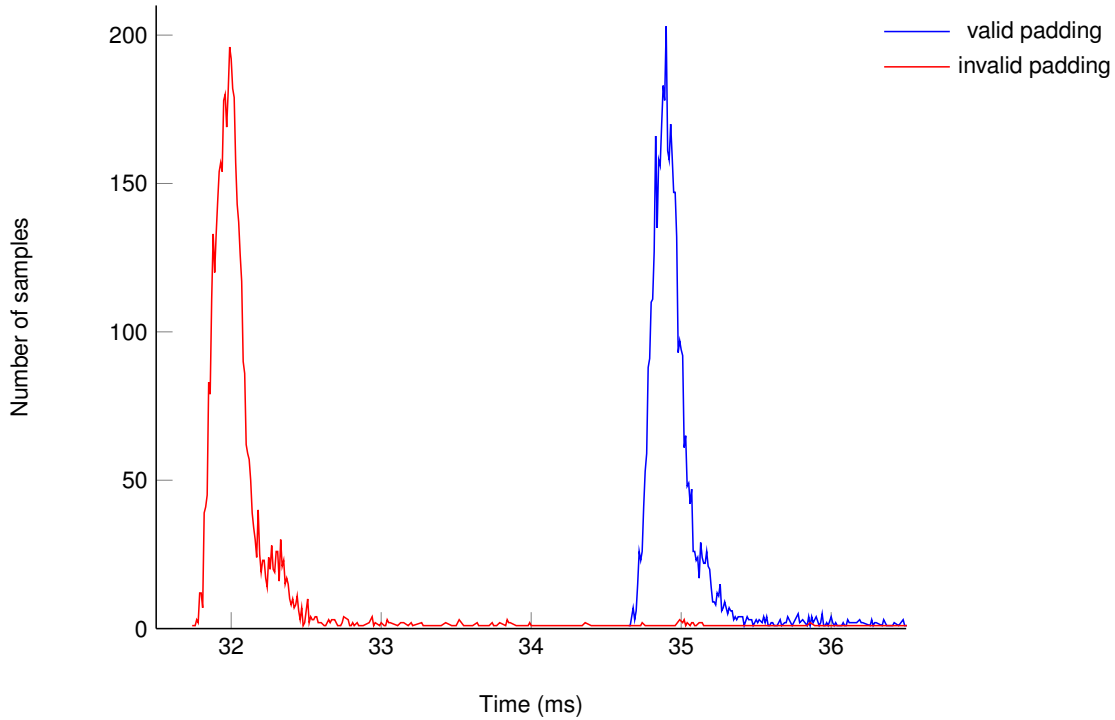
	1. decryption 2. padding data: invalid 3. ~~MAC~~	1. decryption 2. padding data: valid 3. MAC
Error message (e.g., WTLS [V02])	ERR_DEC	ERR_MAC

- How to know if the padding data is valid or invalid (after decryption)?

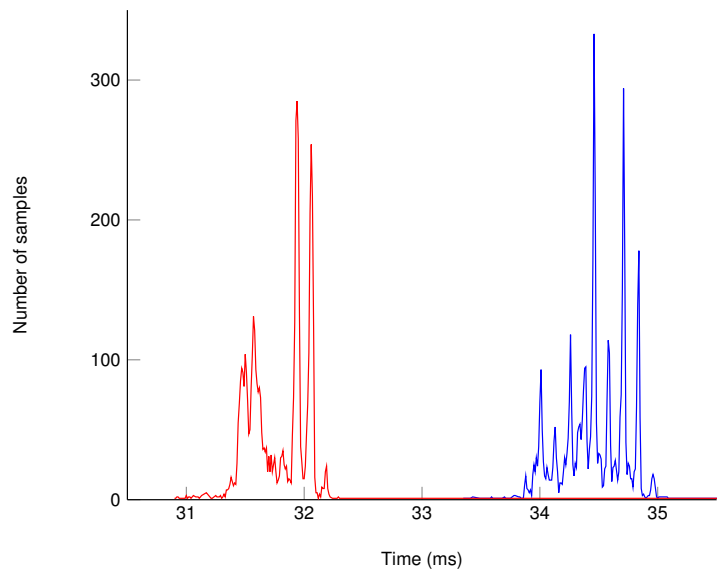
	1. decryption 2. padding data: invalid 3. ~~MAC~~	1. decryption 2. padding data: valid 3. MAC
Error message (e.g., WTLS [V02])	ERR_DEC	ERR_MAC
Computation time (e.g., TLS 1.0 [CHVV03])	time ↘	time ↗

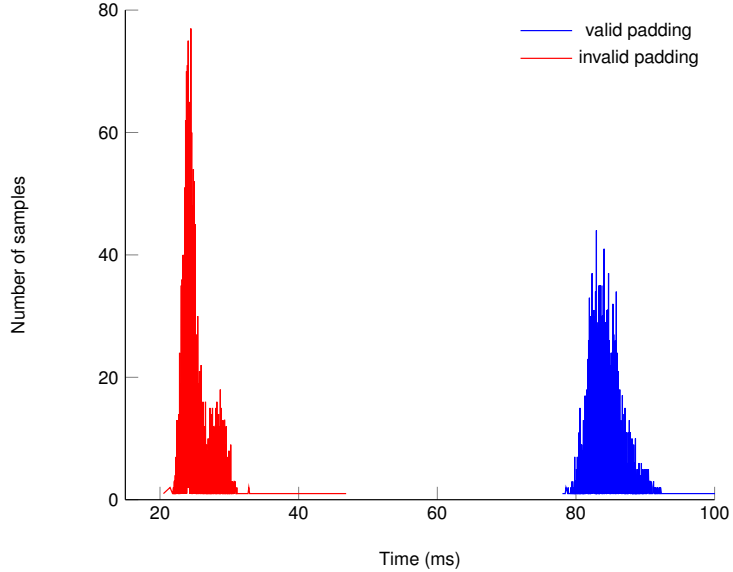
- The smart card sends *always* a response (status word).
- Invalid padding data or invalid MAC $\Rightarrow$ same error code

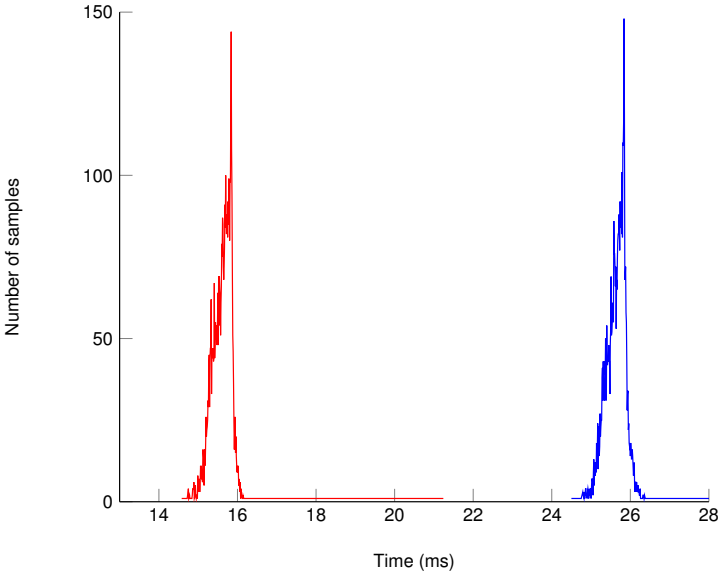
- The smart card sends *always* a response (status word).
- Invalid padding data or invalid MAC ⇒ same error code

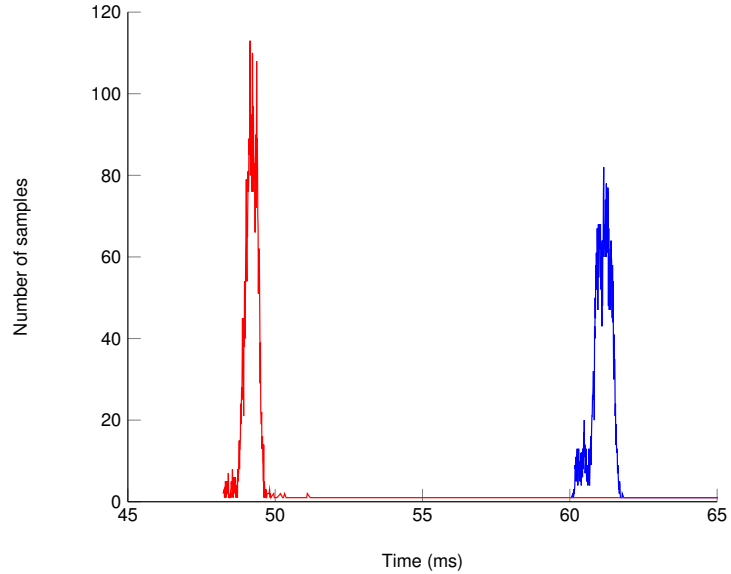


- The card response time reflects the card computation time ⇒ suitable padding oracle


Number of samples
300
200
100
0
31
32
33
34
35
Time (ms)


valid padding
invalid padding
Number of samples
80
60
40
20
0
20
40
60
80
100
Time (ms)


Number of samples
150
100
50
0
14
16
18
20
22
24
26
28
Time (ms)


Number of samples
120
100
80
60
40
20
0
45
50
55
60
65
Time (ms)

- Experimental setting: card connected to a card reader (4 card readers, wired and wireless)
- 10 smart cards from 6 card manufacturers
- SIM cards, generic Java cards

- Experiment: find a 16-byte secret key sent to the smart card in an encrypted SCP02 command
- 300 experiments/card $\Rightarrow$ 100 % success
- Practical complexity $\in [127.75, 133.38]$ close to best average case (128)
- Time to find 16 bytes: 2.7 mn to 11.4 mn (variable response time from the smart card)

- Experimental setting: card connected to a card reader (4 card readers, wired and wireless)
- 10 smart cards from 6 card manufacturers
- SIM cards, generic Java cards

- Experiment: find a 16-byte secret key sent to the smart card in an encrypted SCP02 command
- 300 experiments/card $\Rightarrow$ 100 % success
- Practical complexity $\in [127.75, 133.38]$ close to best average case (128)
- Time to find 16 bytes: 2.7 mn to 11.4 mn (variable response time from the smart card)

$\Rightarrow$ Padding oracle attack is applicable against SCP02.

- Experimental setting: card connected to a card reader (4 card readers, wired and wireless)
- 10 smart cards from 6 card manufacturers
- SIM cards, generic Java cards

- Experiment: find a 16-byte secret key sent to the smart card in an encrypted SCP02 command
- 300 experiments/card $\Rightarrow$ 100 % success
- Practical complexity $\in [127.75, 133.38]$ close to best average case (128)
- Time to find 16 bytes: 2.7 mn to 11.4 mn (variable response time from the smart card)

$\Rightarrow$ Padding oracle attack is applicable against SCP02.

$\Rightarrow$ Among all the deployed smart cards (including 6 billion SIM cards), how many may be impacted?

- Possible real-life scenario: upload of an applet embedding a secret key (e.g., transportation, banking) into the UICC/SIM card.

1. The victim downloads from a popular store an infected application into his smartphone. The application embeds a Trojan ✹ (e.g., Tordow [K16], Dvmap [U17]).
2. The Trojan gets access to the memory space of the legitimate application ■ (through privileges escalation).
3. The Trojan can apply the attack: it reads, and modifies the encrypted SCP02 commands received by the legitimate application.
4. The Trojan repeatedly triggers the installation/deinstallation of the applet $\Rightarrow$ the secret key is repeatedly sent through (new) SCP02 channels.

- Correct implementation (not possible for deployed cards)
- Use additional security mechanisms (if such mechanisms are available)
- Use `PUT KEY` command to send sensitive data (dictionary attack applicable [ST16])
- Do not send too many times the same data (server side)

- The padding oracle attack against SCP02-compliant smart cards is possible because of
  - a theoretical flaw lying in the SCP02 protocol (Encrypt-and-MAC scheme),
  - exploited by means of a timing side-channel provided by the smart cards (implementation).
- Several requirements to be fulfilled in order for the attack to be successful.
- Practical attack
  - Experimental setting: 10 smart cards from 6 manufacturers.
  - How many smart cards impacted in real life?
- Responsible disclosure (October 2017-April 2018): card manufacturers, GlobalPlatform.
- SCP02 is now deprecated (March 2018): use SCP03 instead.

Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack

Gildas Avoine[1,2] Loïc Ferreira[3,1]

Univ Rennes, INSA Rennes, CNRS, IRISA, France

Institut Universitaire de France

Orange Labs, Applied Cryptography Group, Caen, France

September 12, 2018

[SCP02] GlobalPlatform. *GlobalPlatform – Card Specification*, version 2.3.1, ref. GPC_SPE_034, March 2018.

[SCP03] GlobalPlatform. *GlobalPlatform Card Technology – Secure Channel Protocol '03' – Card Specification v2.2 – Amendment D*, version 1.1, ref. GPC_SPE_014, July 2014.

[ISO9797-1] ISO/IEC JTC 1/SC 27. *ISO/IEC 9797-1:2011 – Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*, 2011.

[ISO10116] ISO/IEC JTC 1/SC 27. *ISO/IEC 10116:2017 – Information technology – Security techniques – Modes of operation for an n-bit block cipher*, 2017.

[ISO7816-4] ISO/IEC JTC 1/SC 17. *ISO/IEC 7816-4:2013 – Information technology – Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*, 2013.

[V02] S. Vaudenay. *Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS...* In L. R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*. LNCS, vol. 2332, pp. 534-545. Springer, 2002.

[CHVV03] B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux. *Password interception in a SSL/TLS channel*. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*. LNCS, vol. 2729, pp. 583-599. Springer, 2003.

[ST16] M. Sabt, J. Traoré. *Cryptanalysis of GlobalPlatform Secure Channel Protocols*. In L. Chen, D. McGrew, C. Mitchell, editors, *Security Standardisation Research – SSR 2016*. LNCS, vol. 10074, pp. 62-91. Springer, 2016.

[K16] A. Kivva. *The banker that can steal anything*, 20/09/2016. Available via `https://securelist.com/the-banker-that-can-steal-anything/76101/`.

[U17] R. Unuchek. *Dvmap: the first Android malware with code injection*, 08/06/2017. Available via `https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/`.

M	C	μ_W (ms)	μ_R (ms)	t_{min} (ms)	m	τ_+ (%)	K_W	K_R	Z	Z/n
1	A	39.60	42.59	41.00	28	0.16	1	3	2055.71	128.48
	B	40.19	43.94	42.00	28	0.44	1	3	2077.78	129.86
2	C	25.17	84.34	75.00	0	0.00	1	2	2043.95	127.75
	D	26.64	34.36	32.00	0	0.00	1	2	2066.54	129.16
3	E	15.61	25.65	23.00	0	0.00	1	2	2134.03	133.38
4	F	31.81	34.48	33.00	28	0.48	1	3	2109.71	131.86
	G	15.64	18.53	17.00	0	0.28	1	3	2103.62	131.48
5	H	25.18	84.86	72.00	0	0.00	1	2	2048.34	128.02
6	I	25.90	35.85	32.00	0	0.06	1	3	2108.60	131.79
	J	14.32	19.92	17.50	0	0.10	1	2	2094.85	130.93

1. The attacker sits between the remote server and the card at a point where she can directly eavesdrop on SCP02 encrypted commands and send modified commands to the card.
2. The attacker is able to discriminate response times corresponding to a valid and an invalid padding.
3. The remote server repeatedly sets up a (new) secure channel with the card.
4. The same secret information is sent through each such secure channel.
5. The secret information is sent at a predictable position.

NB: req. 4 $\Rightarrow$ req. 3 (and 5)