More insights on encryption + authentication

come combinarli

Encryption vs Integrity

- **Integrity**
  - Prevents message spoofing (injection)
  - Prevents message tampering (modification)

- **Integrity may be the only requirement**
  - ⇨Goal: everybody should "see"
  - ⇨But nobody should be able to change…

- **Encryption may NOT guarantee integrity!**
  - ⇨We have already discussed this quite a lot!!

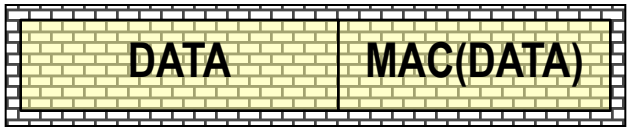
Encrytion does not nearly guarantee integrity!

- **One time pad (one time random key K): perfect secrecy… But:**
  - Encryption: ENC(M) → C = M⊕K
  - But:
    - →C⊕M' = (M⊕K)⊕M' = (M⊕M')⊕K=ENC(M⊕M')
    - →"pay 1000 $" ⊕ "…(1 ⊕ 9)…" → "pay 9000 $"
- **Not the only case**
  - RC4: same as above
  - Homomorphic encryption: modifiable by design!
    - →including RSA
- **In general, don't trust encryption mechanisms for integrity**
  - unless they are explicitly designed ALSO for it (authenticated encryption, e.g. AES-CCM, AES-GCM, etc)
  - TLS 1.3: **ONLY** (!!) AEAD ciphersuites!!
    - →AEAD = Authenticated Encryption with Associated Data see e.g. IETF RFC 5116

How to combine ENC + MAC?

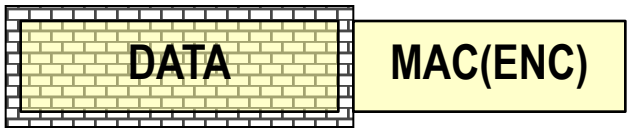
TLS: MAC then ENCRYPT

Data → Hmac → encrypt



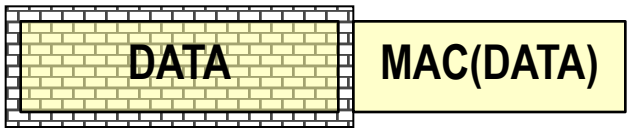
IPsec: ENCRYPT then MAC

Data → encrypt data → MAC(encrypt data)



SSH: ENCRYPT and MAC

data → encryption, HMAC



sbagliato!

Issue: which construction is best, in the assumption of

- GENERAL (semantically secure) Encryption scheme
- GENERAL (unforgeable) MAC scheme

la loro unione perde alcune proprietà

Encrypt and MAC (SSH)

➔**Most insecure!**

⇨Unforgeable MAC construction does NOT guarantee anything about information leakage!

⇨MAC (not encrypted, and applied to plaintext) may reveal something about the message!

➔**Why? ☺**

MAC then Encrypt (TLS)

➔**Not recommended!**

➔**Pathological counterexamples**

➔Krawczyk, Crypto 2001: explicit costruction, active attack against "strange" (but shannon' secure) ENC

➔**Practical attacks found**

➔Degrabriele, Paterson, ACM CCS 2010, on some unusual (but legitimate) IPsec configurations

» AH then ESP with encryption only

➔**Opens the door to Chosed Ciphertext Attacks**

⇨More later: Padding oracle, lucky 13, etc

Encrypt then MAC

- **Provably secure**
  - If semantically secure Encryption scheme
    - Note: ENC not required to be secure against active attacks!
  - And unforgeable MAC scheme
- **Conclusion:**
  - If given the choice, ALWAYS use ENC then MAC, i.e. IPsec-style
  - **EVEN BETTER: avoid the problem with AEAD!**

Interlude: recap on CBC encryption

29/10/21

TLS encryption?

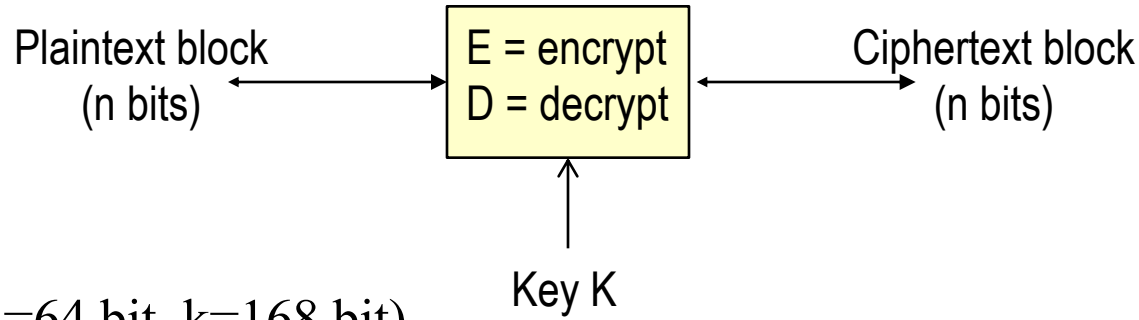
- **Not a single scheme!**
  - TLS supports a number of ciphers!
  - Attacks apply to specific ciphers
- **Attack categories:**
  - Against "bad" ciphers
    - Obvious
  - Against "bad" usage of ciphers made by TLS
    - less obvious!
  - In either cases, practical exploitation of vulnerabilities may not be that obvious

come TLS usa CBC (AES-CBC)

Background: Block ciphers recap

➔Two "ingredients":

- ⇨A "secure" Pseudo Random Permutation
  - ➔Reversible Transformation for a single block



  - » 3DES (n=64 bit, k=168 bit)
  - » AES (n=128 bit, k=128 | 192 | 256 bit)
- ⇨A "secure" way to combine block transformations to encrypt a text of arbitrary length
  - » Cipher Block Chaining - CBC
  - » Counter mode – CTR
  - » …

From PRP to cipher: what NOT to do

➔ **Electronic Code Book**

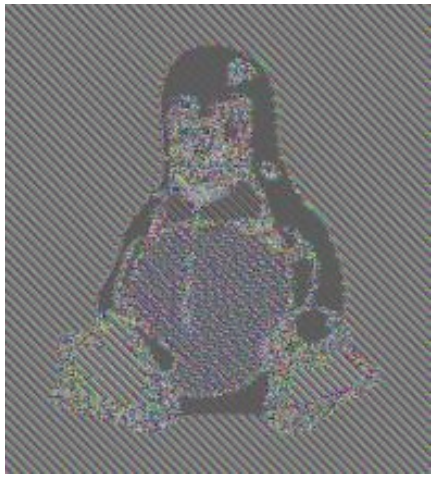
…	CIAO	COME	STAI	COME	VA??	…

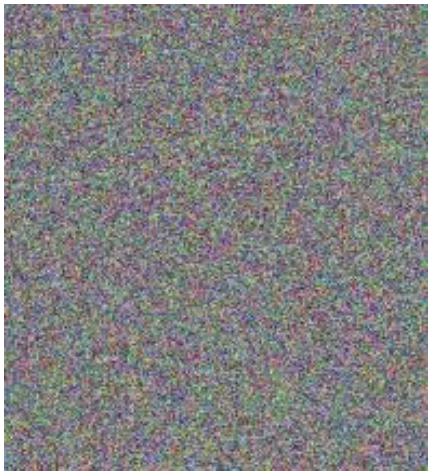
…	A231	3BFA	1221	3BFA	F565	…

➔ **Same PT block ➔ same CT block!**

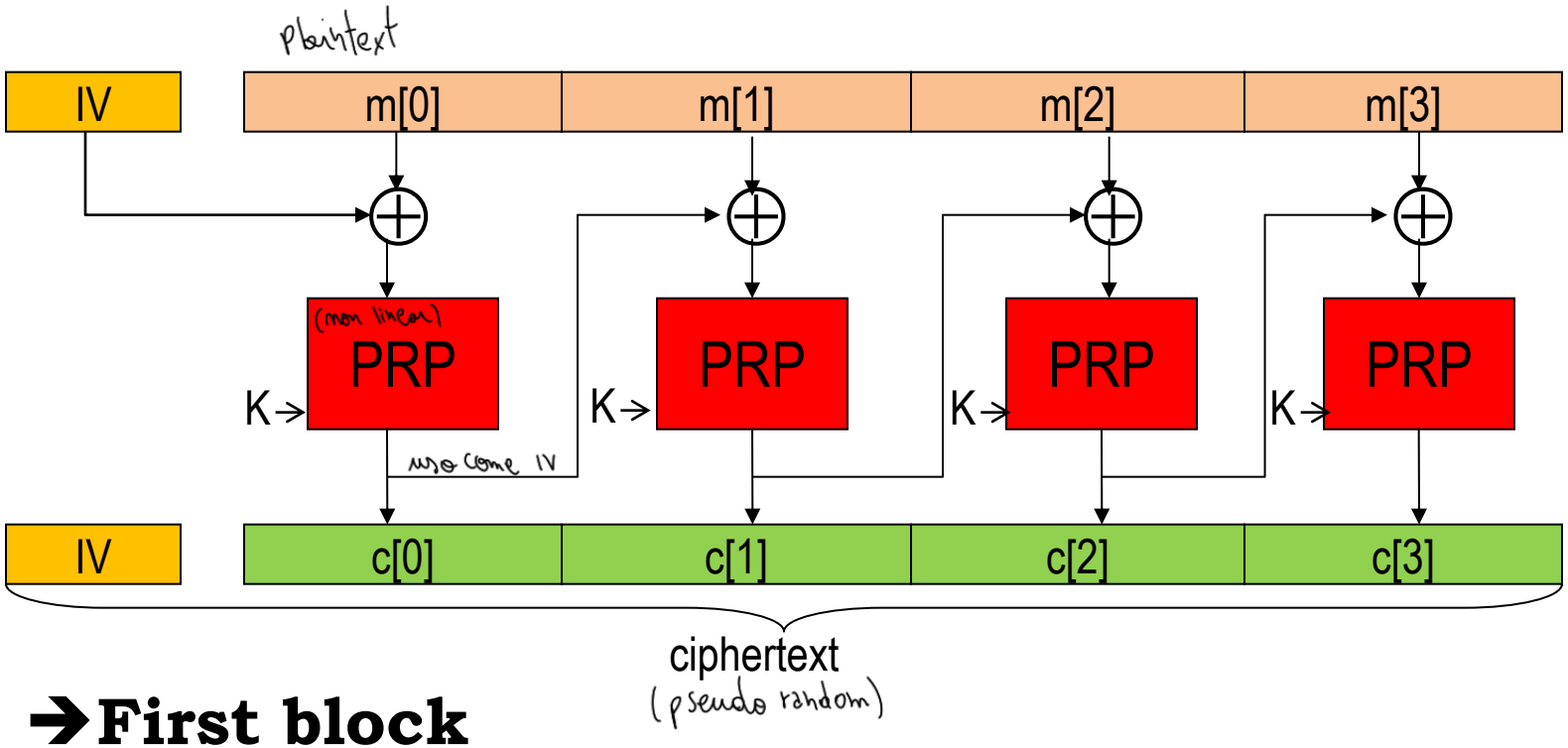
➔Leaks info on data patterns (NO semantic security)

➔Rather, CT should be undistinguishable from random noise





CBC encryption



➔**First block**

⇨XOR message m[0] with random initialization vector IV

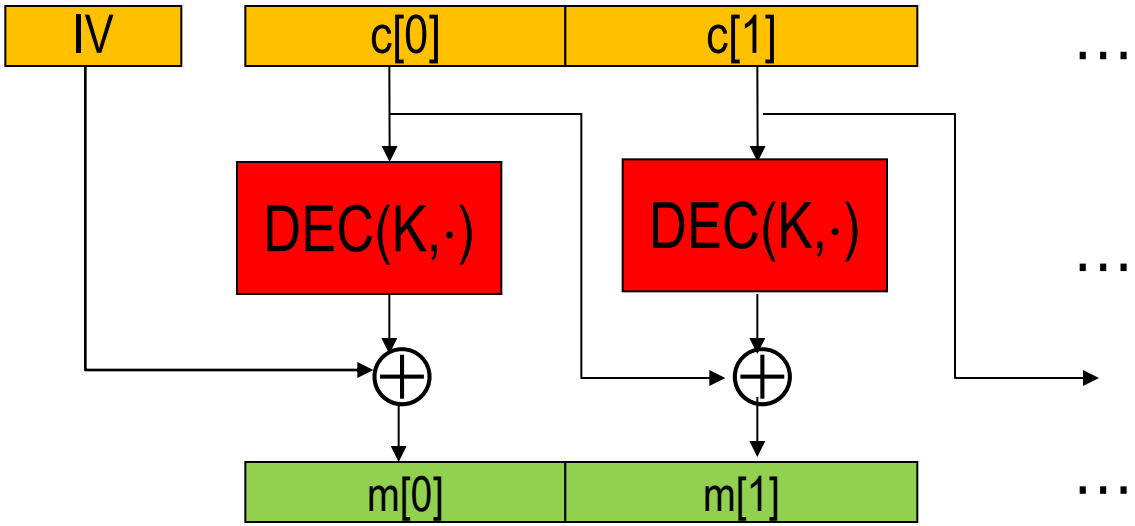
➔**Subsequent blocks**

⇨XOR message m[i] with previous ciphertext block c[i-1]

CBC decryption

→ **c[i] = ENC(K, c[i-1] ⊕ m[i])**

→ **m[i]= c[i-1] ⊕ DEC(K, c[i])**



il pattern "inverte" ciò che facevo nell'encrypt (qui prima decripto, poi ⊕)

Beast attack è un CPA basato sulla predizione dell'IV, dove il TLS della vittima encrypta un msg che voglio.

Back to MAC-then-encrypt: the CBC Padding Oracle attack

exploits poor protocol choice (and bad implementation) in TLS 1.0

ora vediamo altro attacco: dato ciphertext, pomo produrre un cipher text modificato da far decriptare ad un oracolo. Chosen Ciphertext attack CCA

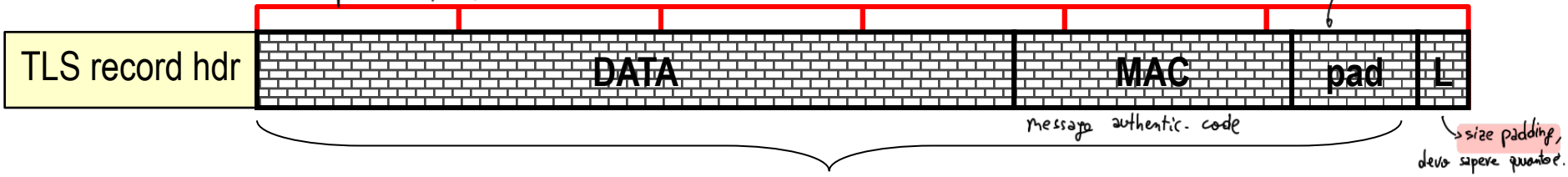
. PKCS #7 è alg. usato da TLS per padding (aggiungere elementi per avere dimensione fissata)

CBC padding

(fondamentale!!)

"chop" data, spezzo in blocchi.

serve a completare il blocco.



Encrypted, padding → fit last block boundary

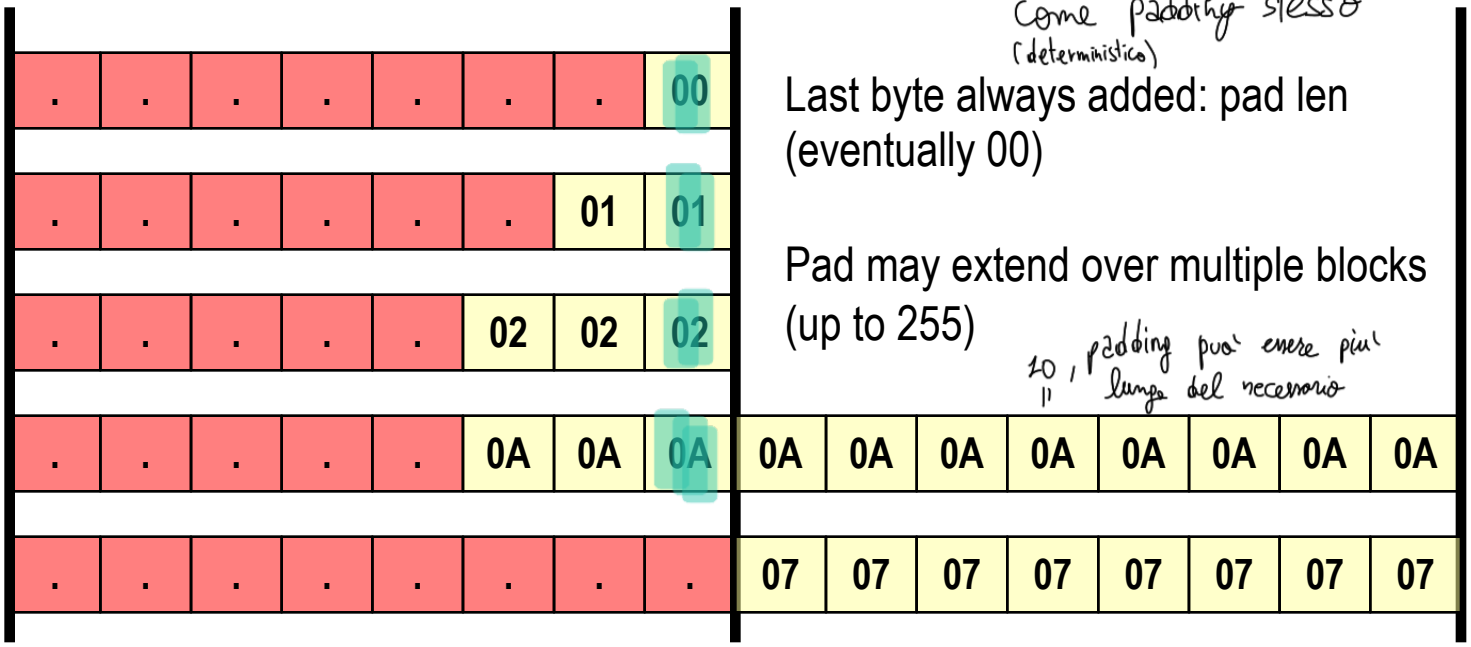
→ TLS padding (say block = 8 B):

size padding, lo uso come padding stesso (deterministico)

è un indicatore: se decripto, mi aspetto alla fine "size L" preceduto da padding con valore 'L'.

... 03 01 02 X

... 02 02 02 ✓, potrei avere errori prima.



Last byte always added: pad len (eventually 00)

Pad may extend over multiple blocks (up to 255)

10, padding può essere più lunga del necessario

ultimo caso: ho un multiplo, quindi metto 'L' nel blocco dopo →

block n

block n+1

CBC decryption steps (TLS 1.0)

- **Decrypt**
  - Unless # bytes NOT multiple of block size - return alert message
    - decryption_failed
- **Remove padding**
  - Read last byte = pad len L
  - Remove remaining L last bytes while checking they are equal to L
  - Invalid pad - return alert message:
    - decryption_failed
- **Check Mac**
  - MAC over decrypted msg (no pad) - if fails return alert message:
    - Bad_record_mac

non spiegare l'errore potrebbe evitare il leak di informazioni? NO, l'alert msg non è l'unico leak, ∃ side channel

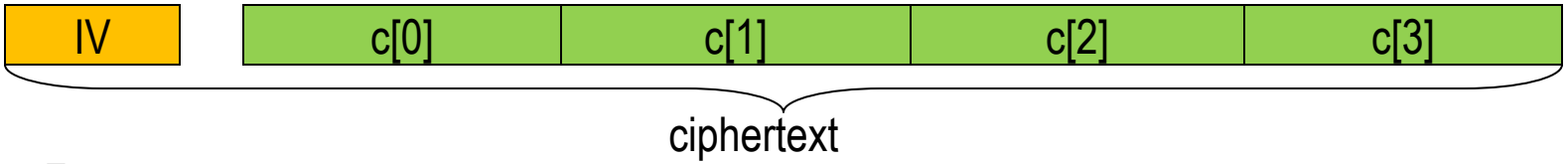
i check sono in sequenza: check pad if NOT OK ... else check mac

ho un "time channel", check mac response più lento del check pad

Typical networking protocols approach: EXPLAIN the error reason

Opposite in crypto: when decryption fails do NOT explain why!!!
(attacker may use such "explanation" to build an attack… see next)

The attack



➔**Goal:**

⇨decryption of a block, say c[1]

➔c[0] needs slightly different approach

➔**Attacker ability: Chosen ciphertext attack**

➔**Attacker is able to submit an(other) ciphertext and get result:**

» Decryption failed

» Bad MAC

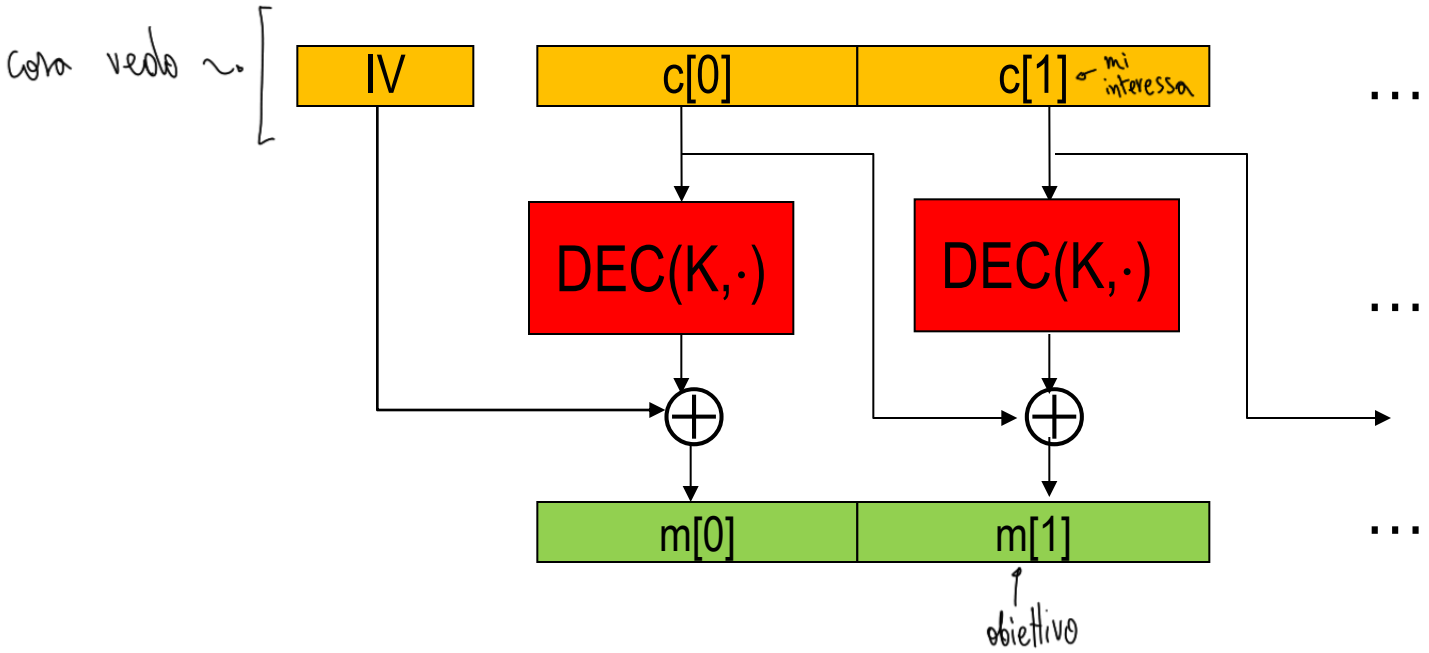
➔**Bad MAC: means padding was found OK! We have a "padding oracle"…**

(errore di design, ho info in più).

Recall CBC decrypt

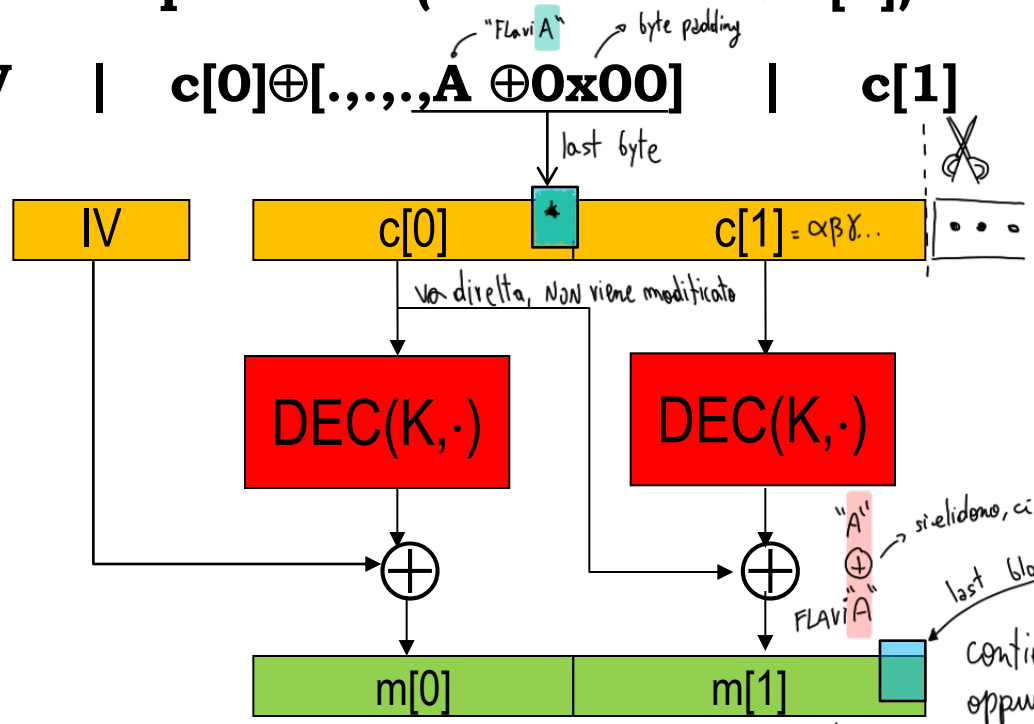
➔c[i] = ENC(K, c[i-1] ⊕ m[i])

➔m[i]= c[i-1] ⊕ DEC(K, c[i])



Start from last byte of m[1]

➔ **Guess: last byte of m[1] = A?**
submit ciphertext (truncated to c[1]):



- Se c'è veramente Flavio stavei facendo C[0] (last byte) ⊕ "A" ⊕ "0x00"
allora m[1] = Flavi 0x00, mi sta dicendo che PADDING è corretto, ok ultimo byte.
- Con enrico e mia scelta flavio avrei avuto ≈ Flavi 0x49, padding NOT OK.

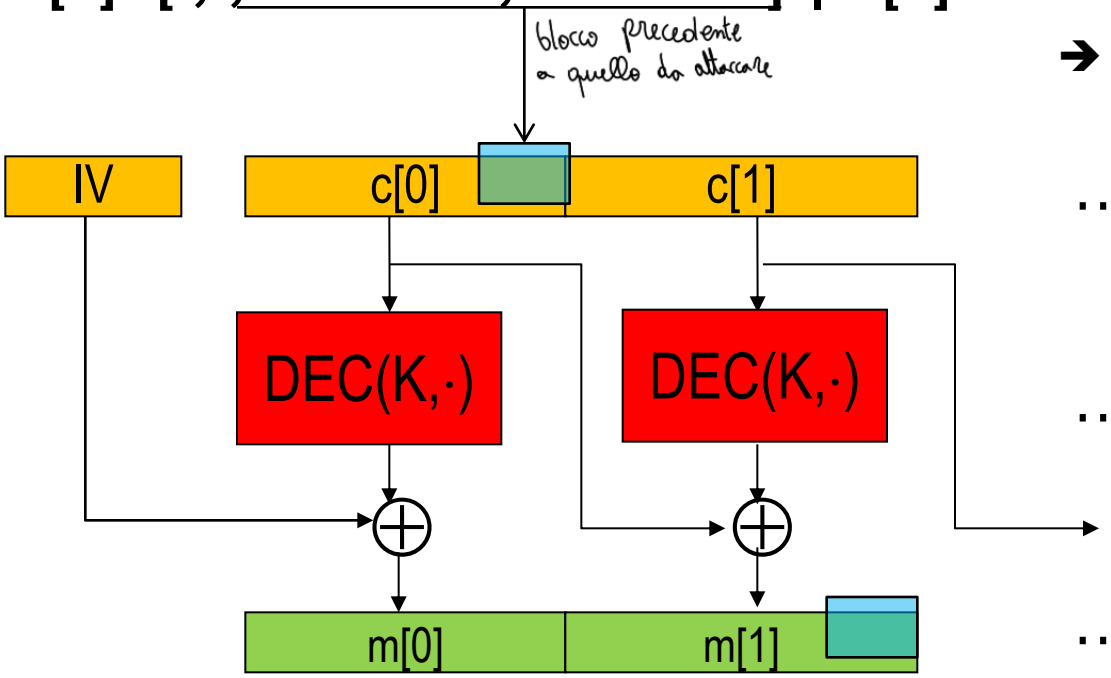
➔ **If guess true, last byte of m[1] = 0x00**
➔ **But then padding would be OK ← bad_record_mac!**
➔ **If guess wrong, ← decryption_failed!!**

Per byte restanti ?

Iterate on remaining bytes

- ➔ **We now know last byte is A**
- ➔ **Next guess: previous byte of m[1] == F?**
- ➔ **submit ciphertext**

IV | c[0]⊕[.,., F ⊕0x01, A ⊕0x01] | c[1]



- ➔ **Example: 8 bytes block:**
  - ⇨ At most 256 guesses for each byte:
    - ➔ $256 \times 8 = 2^{11}$
    - ➔ Fast!

Linear

- ➔ **If true, last 2 bytes of m[1] ➔ 0x01!! Padding check OK, error = bad_mac**

esempio Padding attack

- block size: 4 bytes, CBC block encryption

F1 aa 11 04 | 34 35 F2 20 | 07 07 07 07 | 73 73 73 73 | 65 61 f6 08 | 91 11 5f 20 |

contiene codice segreto

"01 01 01 01" V "02 02 02 02"

Quale chosen plaintext msg attacker deve mandare per determinare il codice?

SVOLGIMENTO

1) chop del msg, termino col msg da attaccare, NON OLTRE

F1 aa 11 04 | 34 35 F2 20 | 07 07 07 07 | 73 73 73 73 |

01 01 01 01
02 02 02 02

2) devo modificare il blocco precedente, come?

provo con ultimo byte "07" ⊕ "01" ⊕ 00, cioè:

- "01" ⊕ 00: provo lui
- 00: padding che vorrei

07
"02" 0 | 0111 ⊕
0 | 0001

(padding non cambia nulla, non considero)

0110

0 | 0110 = 06

3) ho: F1 aa 11 04 | 34 35 F2 20 | 07 07 07 06 | 73 73 73 73

- Se fossero tutti 06 06 06 06 ? Sarebbe comunque valido!

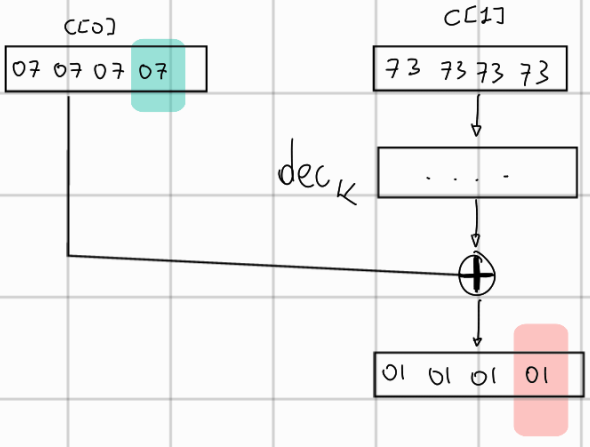
Giusto perché 73 73.. generato da blocco precedente, otterrei padding 00 00 00 00 valido.

- Se avessi sostituito tutti, usando "07" ⊕ "01" ⊕ "03" OK

ottengo 03|03|03|03 (padding | len) — qui

anche 07⊕03⊕00, sostituendo tutti è "OK"

- NB:



mettendo '07 07 07 06' ⊕ dec(K, C[1])

se '01 01 01 01' è corretto, otterrei '01 01 01 00'.

Poiché $dec_K[1] = C[0] \oplus m[1]$ allora

'07' ⊕ '01' ⊕ ⊕ '06' = ∅

Se avessi ipotizzato '01', ma m[1] = 02 avrei avuto:

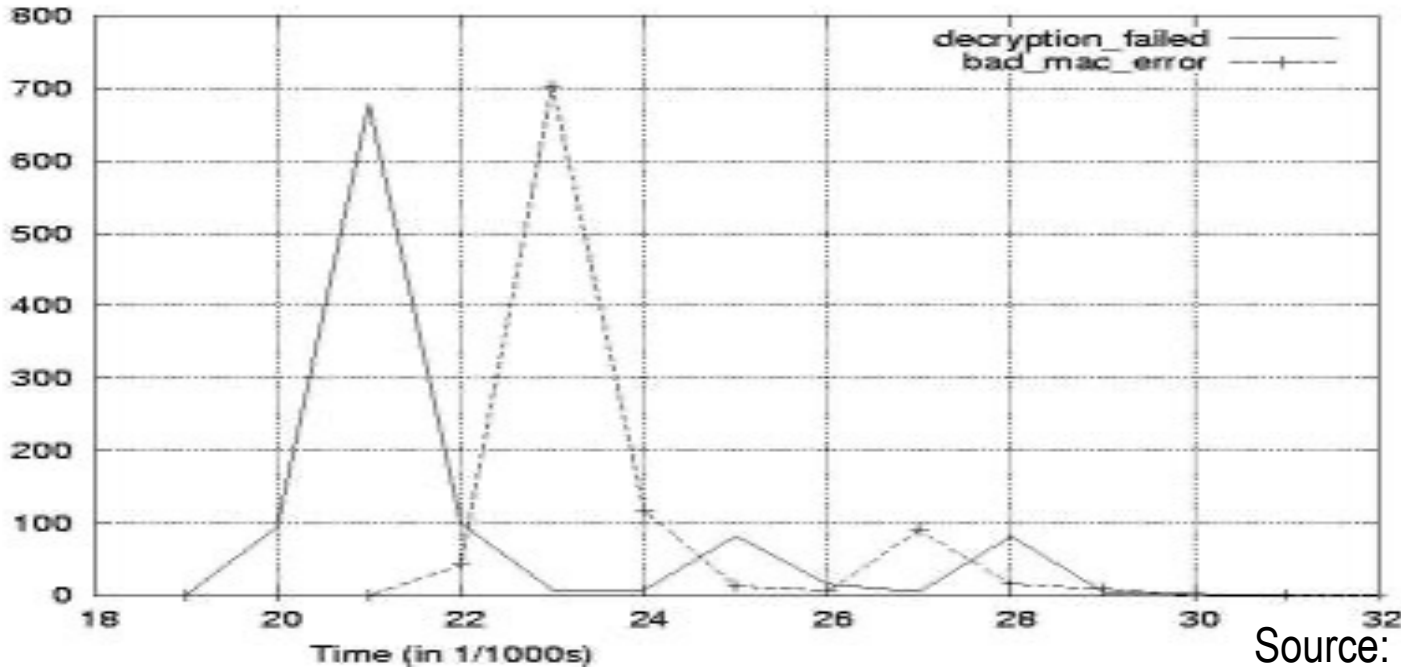
$Dec_K \oplus 06 = [07 \oplus 02 \oplus 06] = 3 \neq \emptyset$

Anche far scoprire un bit può essere TRAGICO!!

Preventing the attack

- **Discovered in 2002 (Vaudenay)**
- **Correct TLS protocol to avoid such "padding oracle"**
  - Most TLS1.0 implementations: respond with same alert (bad mac) in both cases
  - TLS1.1: standardized this
- **But canvel 2003: side channel "padding oracle"!**
  - Must correct also **implementation**!!
  - TLS1.1: always perform MAC even if malformed msg

anche se decryption fail, eseguo mac control, così non so se il padding è corretto, MAI.



Source: canvel 2003

Lessons

➔**Encrypt then MAC would NOT have permitted such attack**

➔**When dealing with crypto, we (networkers) need to be more careful in what we report…**

⇨The least, the better

➔**Crypto implementations are extremely critical**

⇨Side channel attacks

⇨Don't implement crypto yourself!

ultima slide vista ;

Is the attack practical?

- **Apparently no**
  - Bad_mac and decryption failed: fatal alerts
  - TLS connection aborted !!!
  - Next connection will have different key!
- **But… actually performed on IMAP (Vuagnoux & Canvel, 2003)**
  - Email IMAP client periodically sends login/passwd every few minutes (1-5) over TLS
  - Resulting TLS msg plaintext
    - →XXXX LOGIN "username" "password"<cr><nl><MAC><PAD><LEN>
  - C→ S intercepted with MITM attack (DNS spoofing), and CCA attack performed
    - →Connection abort not an issue
    - →Attack (optimized with further dictionary attack strategies) successful in at most few hours

un tentativo ∀ versione TLS

Fixes and follow-ups

- **TLS1.2:**
  - If padding fails, **validate MAC in any case**
- **Devil is in the details**
  - ⇨ But which data is validated? If padding fails, no way to know message size versus padding size
  - ⇨ TLS1.2 solution: use the whole data for validation
    - → But this is **more data** than a correct message!!
    - → May require one extra compression function in the HMAC → extra time
- **2013, Lucky thirteen, kenny Paterson**
- **2014: POODLE**
  - ⇨ combine Padding Oracle with an SSL3.0 downgrade
- **2015, Lucky microseconds, Albrecht & Paterson**
  - ⇨ Attack using this very subtle timing channel!!!
- **2016, CVE-2016-2107 (LuckyNegative20)**
  - ⇨ Attacks the constant-time patch of Lucky13!
  - ⇨ https://blog.cloudflare.com/yet-another-padding-oracle-in-openssl-cbc-ciphersuites/

Fun (?!) Lesson:

➔**Cryptographic Doom Principle**

» © Moxie Marlinspike

⇨if you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom.

MAC deve essere last thing, con disabilito CCA. Se faccio CCA e MAC è prima con che controllo, msg rifiutato sicuramente.
encrypt dopo MAC ✗ | MAC dopo encrypt ✓

➔**Encrypt then MAC would NOT have permitted such a painful Pandora's Box!!**

⇨A 15 years nightmare solved only by… removing the origin problem!

➔AEAD mandatory in TLS1.3