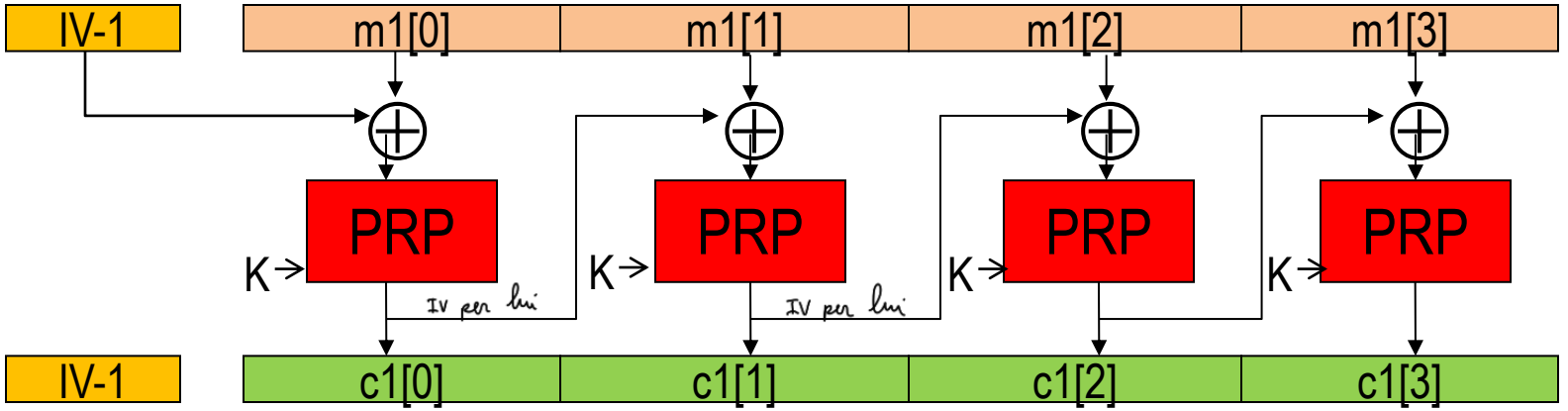
BEAST attack: exploits predictable CBC IV in TLS v1.0

IV nuovo ∀ CHOP, ma se devo fare encrypt di più msg?

(original TLS)

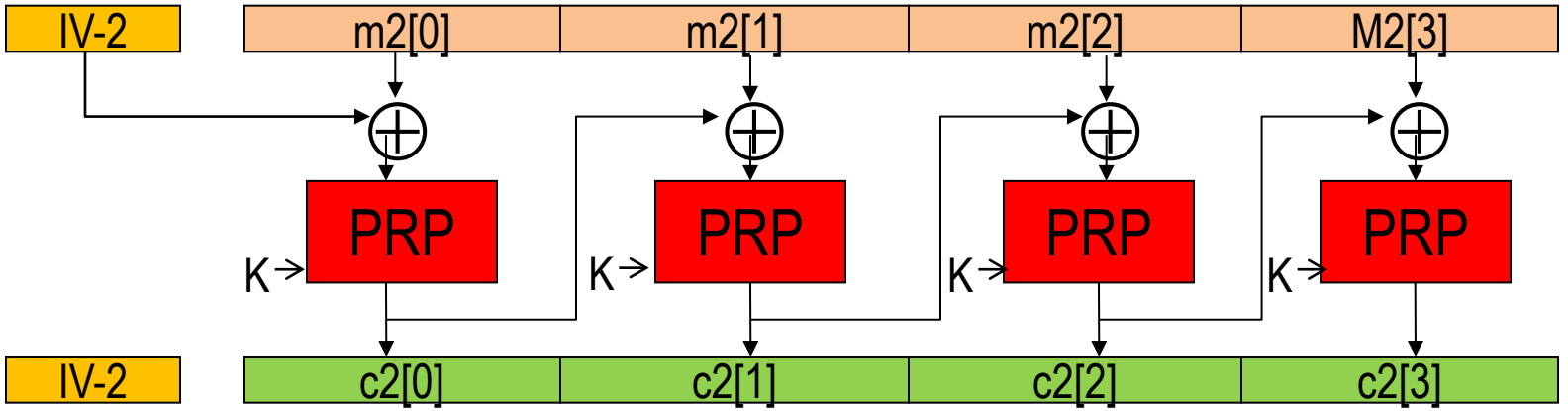
CBC for multiple msg (same K)



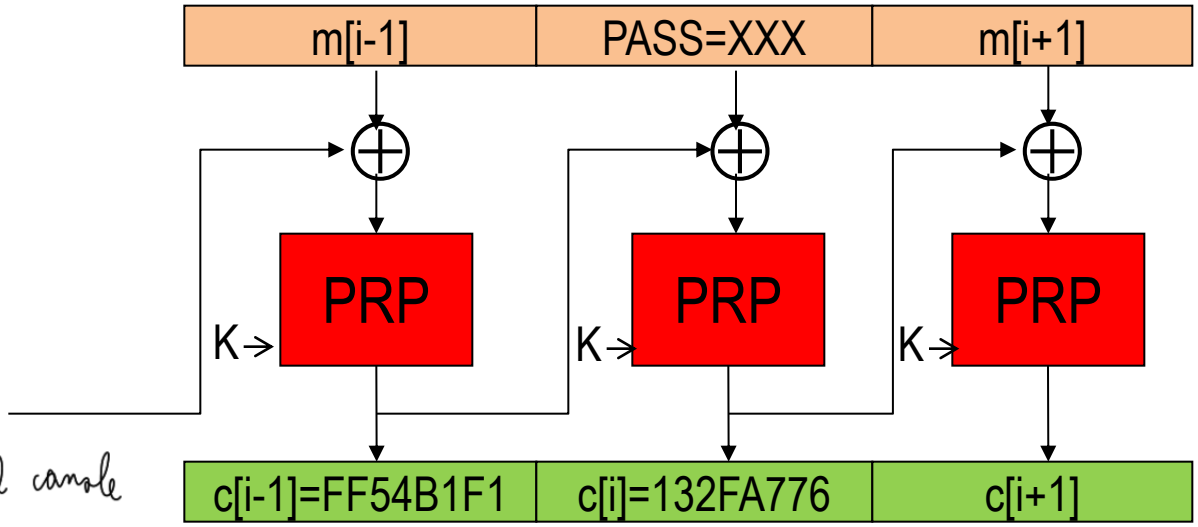
Se c1[3] lo uso come IV-2 per nuovo blocco msg `m2`?

Necessary condition: **IV must differ**, otherwise insecure, again! (same msg → same ciphertext)

But different IV is NOT sufficient: **IV must also be UNPREDICTABLE**

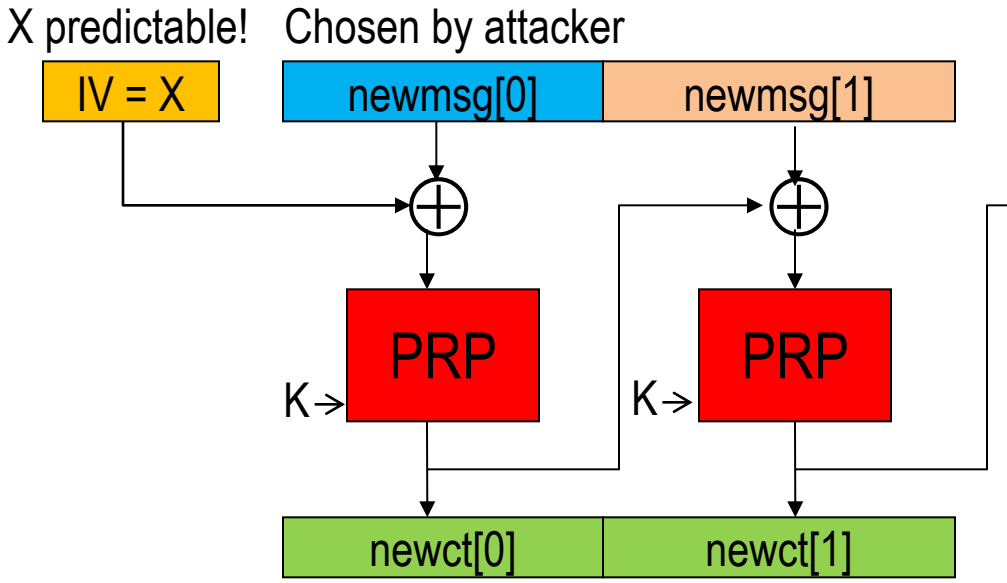


What if predictable IV?



- **Attacker knows m[i] contains password**
  - ⇨ And guess this is either UGO or JOE, but does not know which
- **Attacker sees relevant ciphertext**
  - ⇨ But cannot learn which among the two passwd is encrypted (semantic security)
- **BUT attacker can convict the implementation to encrypt some data of his choice (Chosen plaintext attack): can he learn which passwd was used among the two choices?**

What if predictable IV?



SOLUZIONE: (CPA)
conoscendo c[i-1], messa in ⊕ con pw e in ⊕ con X (1° blocco), se la pw è OK (provo) allora il risultato deve essere c[i]

➔**Attacker predicts X**

➔**Sets newmsg[0] = X⊕c[i-1]⊕PWGUESS**

⇨Example: X⊕"FF54B1F1"⊕"PASS=UGO"

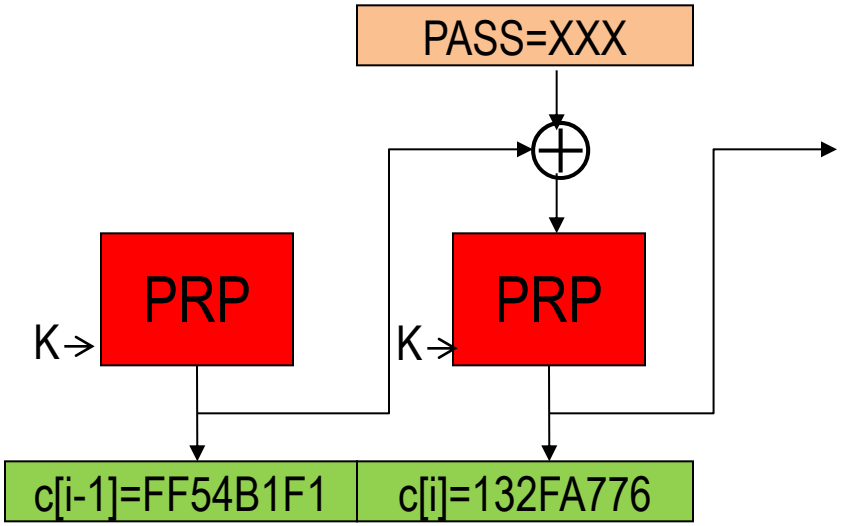
➔**If newct[0] == c[i] then guess OK!!**

Aftermath: predictable IV → can mount a dictionary attack (initially prevented by semantic security)!

HO rotto S.S., eseguendo dictionary attack.

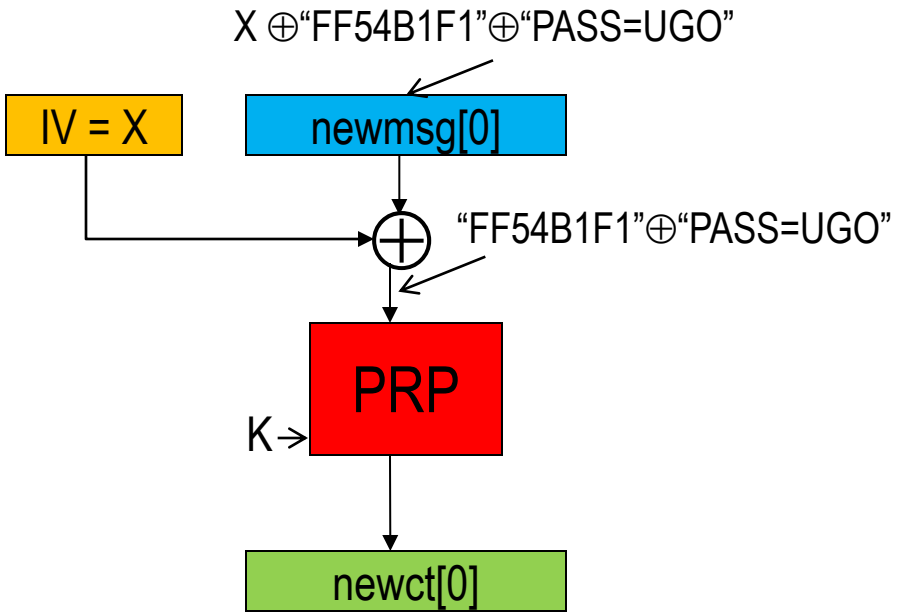
What if predictable IV?

Collected data, including target password



C[i]=ENC(K, “FF54B1F1”⊕“PASS=XXX”)

Chosen plaintext attack



newct[0]=ENC(K, “FF54B1F1”⊕“PASS=UGO”)

If equal to c[i], then guess right!!
If different from c[i], then other pw right!!

anche se le condizioni sono particolari, è successo!

Exploited in TLS!

- **Predictable IV: well known vulnerability**
  - Rogaway 1999 (on IPsec issues)
  - Moller 2004 (explicitly addressing TLS) scopri che
  - ⇨ TLS 1.0: IV of new msg = last c[i] of previous message
- **Corrected in TLS v1.1+**
  - ⇨ Explicit IV per new msg
  - ⇨ Note that this is mandatory for DTLS
    - reception of previous msg not guaranteed
- **Not practical (?)**
  - ⇨ Limited interest so far in deploying TLS1.1+
  - ⇨ Vulnerability theoretical, attack apparently hard to mount
    - **…until september 2011:**
      BEAST (T. Duong & J. Rizzo)

Why attack believed to be unpractical

- **Software issues**
  - Chosen Plaintext Attach is active
    - Active agent code running on user browser
    - Injection of chosen plaintext from attacker
  - Found feasible with modern web technologies
    - Websockets, HTML5 (third party server, potrebbero convincere l'implementazione ad eseguire msg dell'attacker)

come convinco PC a encrypt dato scelto da attacker?
Dovrei fare inject data nel PC e farli encryptare, ma PC è collegato direct col server, non ho altre connessioni.

Why attack believed to be unpractical

Se ciò descritto fosse possibile avrei comunque i seguenti problemi:

➔**Complexity issues**

- ⇨Brute force attack: must provide a guess for all possible content of a block
  - ➔AES: 128 bits block ➔ 2^{128} guesses ➔ no way
- ⇨Authentication cookies may be long!!
  - ➔Complexity of guess exponential with authentication cookie size
  - ➔Frequently auth cookies look very much «entropic», hard to dictionary attack

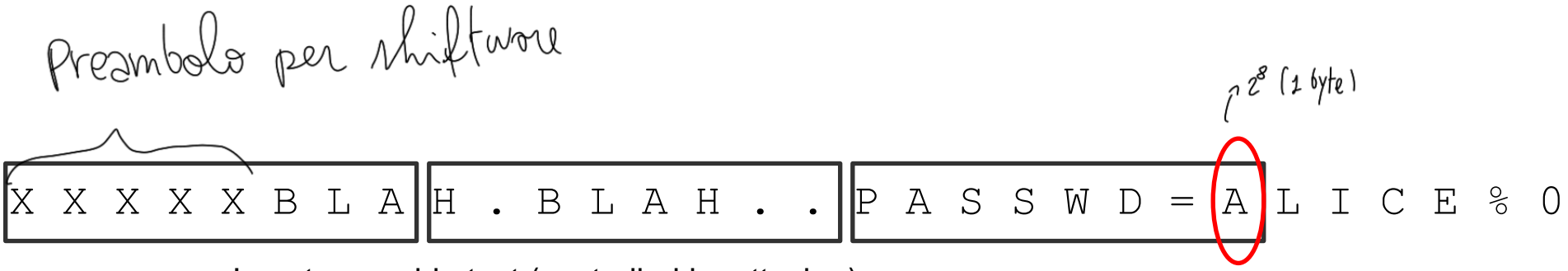
➔**Turned out NOT to be the case!**

- ➔Breakthrough enabler: «chosen boundary attack»!!
- ➔Attack LINEAR (!) with authentication cookie size!!

Chosen Boundary attack

AES Block «border» not aligned to our field of interest

And in general authentication token not inside a single block

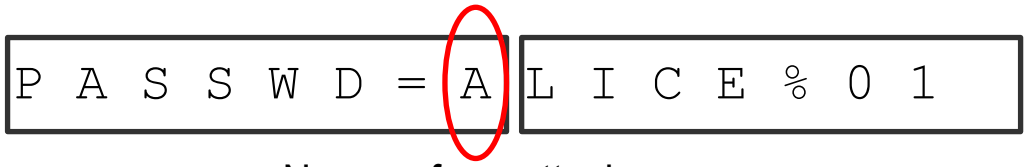


Insert preamble text (controlled by attacker)

Align authentication token so that **ONLY FIRST CHARACTER is in block!!**

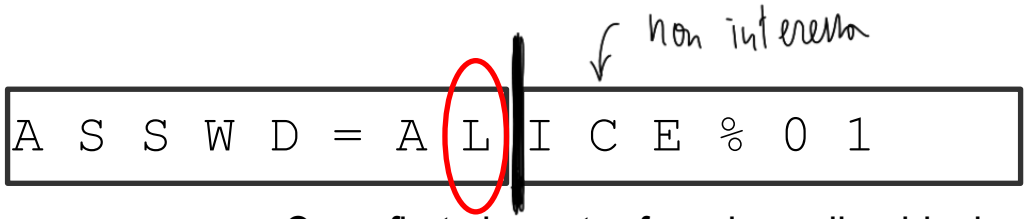
And read relevant ciphertext

Chosen Boundary attack



Now perform attack:
256 guesses at most!
Less if base64 or if letters/numbers used, only

sto attaccando il singolo blocco, con trovo una lettera alla volta



Once first character found, re-aling block and perform attack:
256 guesses at most, again!

Complexity for N bytes cookie = 256*N
Versus direct guess = 256^N

e.g. 8 bytes passwd: $256*8 = 2^{11} = 2048$
e.g. 8 bytes passwd: $256^8 = 2^{64} =$
$= 18.446.744.073.709.551.616$

See Duong-Rizzo demo on Youtube

http://www.youtube.com/watch?v=BTqAIDVUvrU

CRIME attack: compression leaks (USABLE) information!

TLS Compression is harmful

Source: T. Duong, J. Rizzo

- **Compression disabled in Chrome after september 2012 CRIME attack**
- **Actually, more general than TSL: similar attack in other settings**
  - TIME – Timing Info-leaks Made Easy
  - BREACH - Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext
    - Extends attack to HTTP compression
- **Not a problem of TLS, but a more general problem!**
  - Actually, potential vulnerability known since 2002, but disregared for 10 years
    - Kelsey, Compression and Information Leakage of Plaintext

Compression leaks size information

Size = 4 bytes

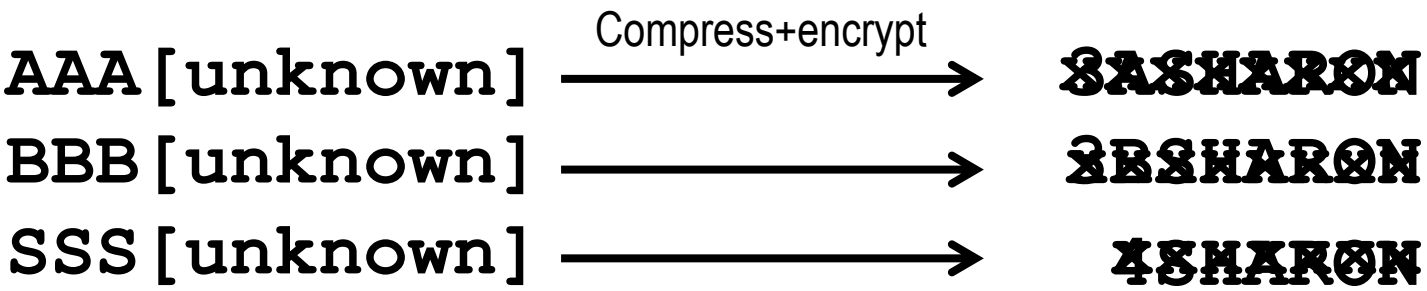
Size = 6 bytes

Combining compression with Plain text injection – concept idea

Your data, unknown to attacker **SHARON**

But attacker can see size of Compressed+encrypted data **SHARON**

Attacker can repeatedly inject chosen plaintext prior to your data, and SEE size of encrypted + compressed result



What's the first letter of your data? Pretty easy, isn't it? ☺

Attack made practical

- **CRIME = Compression Ratio Info-leak Made Easy**
  - Duong and Rizzo, again, september 2012
  - Works IRRESPECTIVE of cipher!!

- **Practical enablers**
  - Injection of chosen text prior to user data
    - Reusing BEAST software tools
  - Suitably chosen injected text
    - Relates to compression method details
      - » Practically done for DEFLATE, in principle can be extended to other compression methods
  - Some extra tech details (skipped)

Attack details (simplified)

- **Attack relies on DEFLATE details**
- **DEFLATE (jointly) uses two sub-algorithms**
  - Huffman coding
    - we don't discuss it, to keep simple
  - LZ77: replaces repeated 3+ char strings with (offset,size) pointers

```
GIUSEPPE BIANCHI AND MARCO BIANCHINI      36B

GIUSEPPE BIANCHI AND MARCO (-18,7)NI      31B
```

Attacker injects chosen text

- **Legitimate user text:**
  - known header + unknown secret
  - e.g. authentication token: passwd=alice%01
- **Message crafted by attacker**
  - **Att_input** + [header + secret]
  - Repeater many times
- **Attacker sees**
  - Len(encrypt(compress(att_input+header+secret)))

How to chose input

```
GET /comment:twid=a HTTP/1.1 Cookie: twid=flavia\r\n
GET /comment:twid=a HTTP/1.1 Cookie: (-24,5)flavia\r\n
```

Repeat guesses… until

```
GET /comment:twid=f HTTP/1.1 Cookie: twid=flavia\r\n
GET /comment:twid=f HTTP/1.1 Cookie: (-24,6)lavia\r\n
```

1 byte shorter!!

And now start guessing second character as well!!

```
GET /comment:twid=fa HTTP/1.1 Cookie: twid=flavia\r\n
GET /comment:twid=fa HTTP/1.1 Cookie: (-24,6)lavia\r\n

GET /comment:twid=fl HTTP/1.1 Cookie: twid=flavia\r\n
GET /comment:twid=fl HTTP/1.1 Cookie: (-24,7)avia\r\n
```

And so on, linearly with secret size (again!)

A bit more complex than this, but… enough to sketch the idea

- **Compression works in bits, not bytes**
  - With huffman coding as well
  - Must be careful to get at least a byte difference
- **Done on two protocols**
  - TLS
  - SDPY (http2.0 candidate)
- **Some technical differences**
  - TLS: includes a further chosen boundary attack
    - Very different boundary: TLS fragment (versus AES block in crime)
  - SPDY: must handle compression state over multiple requests
    - Uses compression dictionary window
- **Several optimizations to make the attack faster**
  - Less guesses
  - compression size separately computed by attacker

More details @ http://www.ekoparty.org//archive/2012/CRIME_ekoparty2012.pdf