Κρυπτογραφία: POODLE, BREACH

Διδασκαλία: Δ. Ζήνδρος

Στόχοι του σημερινού μαθήματος

- Επιθέσεις MitM
- POODLE
- BREACH
- Πρακτικά chosen plaintext attacks
- Εκμετάλλευση μικρής πιθανότητας αποκάλυψης bits

Υποθέσεις BREACH & POODLE

- Eve βρίσκεται στο μονοπάτι του δικτύου της Alice προς την HTTPS υπηρεσία του Bob
  - Κατά τα γνωστά έχει έλεγχο σύμφωνα με **υπόθεση ανασφαλούς δικτύου**
- Eve αναγκάζει την Alice να επισκευθεί μία κακόβουλη σελίδα evileve.com
  - Κατά τα γνωστά σύμφωνα με το **μοντέλο ασφάλειας του web**

Πρακτική εκτέλεση BREACH & POODLE

- Eve τρέχει Javascript κώδικα **της επιλογής της** στον υπολογιστή της Alice
- Eve μπορεί να κάνει HTTPS requests στη σελίδα του Bob με τα cookies της Alice από τον υπολογιστή της Alice

Αιτήματα HTTPS

- Η Eve επιλέγει για το HTTPS request που θα στείλει η Alice στον Bob:
  1. Ποιο URL θα επισκευθεί
  2. Τι δεδομένα θα σταλούν

```
dionyziz@erdos ~ % nc git-class.gr 80
GET / HTTP/1.1
Host: git-class.gr

HTTP/1.1 200 OK
Server: GitHub.com
Date: Tue, 15 Dec 2015 10:17:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 5142
Last-Modified: Sun, 22 Nov 2015 02:15:06 GMT
Access-Control-Allow-Origin: *
Expires: Tue, 15 Dec 2015 10:27:31 GMT
Cache-Control: max-age=600
Accept-Ranges: bytes
X-GitHub-Request-Id: 554B7F72:35D0:122CC253:566FE8B7

<!DOCTYPE html>
<html>

  <head profile="http://www.w3.org/2005/10/profile">
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="chrom
    <meta name="description" content="Git &amp; GitHu

                <link rel="icon"
      type="image/png"
      href="https://assets-cdn.github.com/favicon.ico
    <link rel="stylesheet" type="text/css" media="scr

    <title>Git &amp; GitHub Class</title>
```

Μορφή ενός GET HTTP request

Επίσκεψη Alice στα Facebook μηνύματά της

```
GET /messages HTTP/1.1
Host: www.facebook.com
Cookie: auth=secret
```

Μορφή ενός GET HTTP response

Επίσκεψη Alice στα Facebook μηνύματά της

```
HTTP/1.1 200 OK
Server: facebook.com

{'messages':
[{'from': 'Bob', 'text': 'Hi'}]}
```

HTTP GET παράδειγμα

Μορφή ενός POST HTTP request

Αποστολή μηνύματος Alice στο Facebook

```
POST /sendmessage HTTP/1.1
Host: www.facebook.com
Cookie: auth=secret

message=Hi&target=Bob
```

Στόχοι της Eve

- Τα **cookies** που στέλνονται στο request που είναι εμπιστευτικά
  - π.χ. authentication cookies
  - Η Eve δεν έχει κανονικά πρόσβαση σε αυτά παρ' όλο που στέλνονται μαζί με το request της
- **Η απάντηση** που λαμβάνεται στο σώμα του response που είναι εμπιστευτική
  - π.χ. προσωπικά μηνύματα, emails
  - Η Eve δεν έχει πρόσβαση λόγω same-origin policy

Στόχοι της Eve: Request cookie

POST /sendmessage HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&**auth=secret**&x=y...

…

message=Hi&target=Bob

Στόχοι της Eve: Response text

```
HTTP/1.1 200 OK
Server: facebook.com
Date: Tue, 15 Dec 2015 10:17:31 GMT

{'messages':
[{'from': 'Bob', 'text': 'Hi'}]}
```

Τι ελέγχει η Eve? URL

GET /**messages** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

Τι ελέγχει η Eve? GET request query

GET /messages?**q=text** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

Τι ελέγχει η Eve? POST body

POST /sendmessage HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob

POODLE

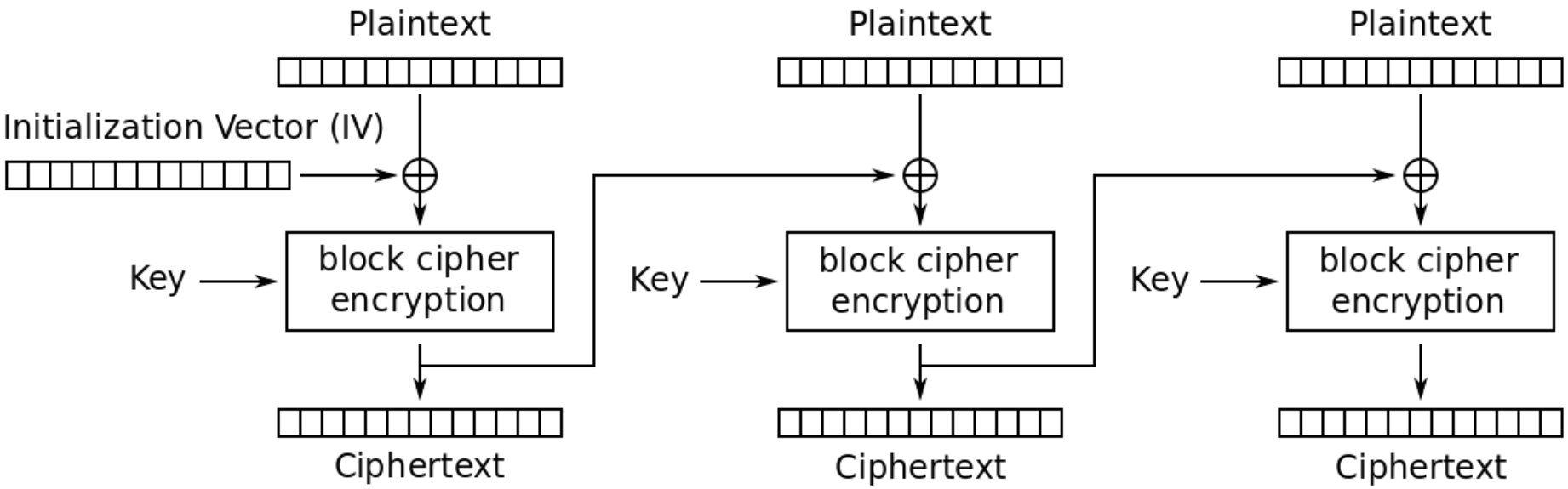
- Thai Duong, Bodo Möller, Krzysztof Kotowicz
  - Google, 2014
- Επίθεση εναντίον του HTTPS
- Πετυχαίνει πλήρη αποκρυπτογράφηση ενός μυστικού, π.χ. πιστωτική κάρτα
- Απαιτεί SSLv3
  - Πλέον δεν μπορεί να εφαρμοστεί
  - Έχει γίνει upgrade σε TLSv1.0, TLSv1.1, TLSv1.2

POODLE

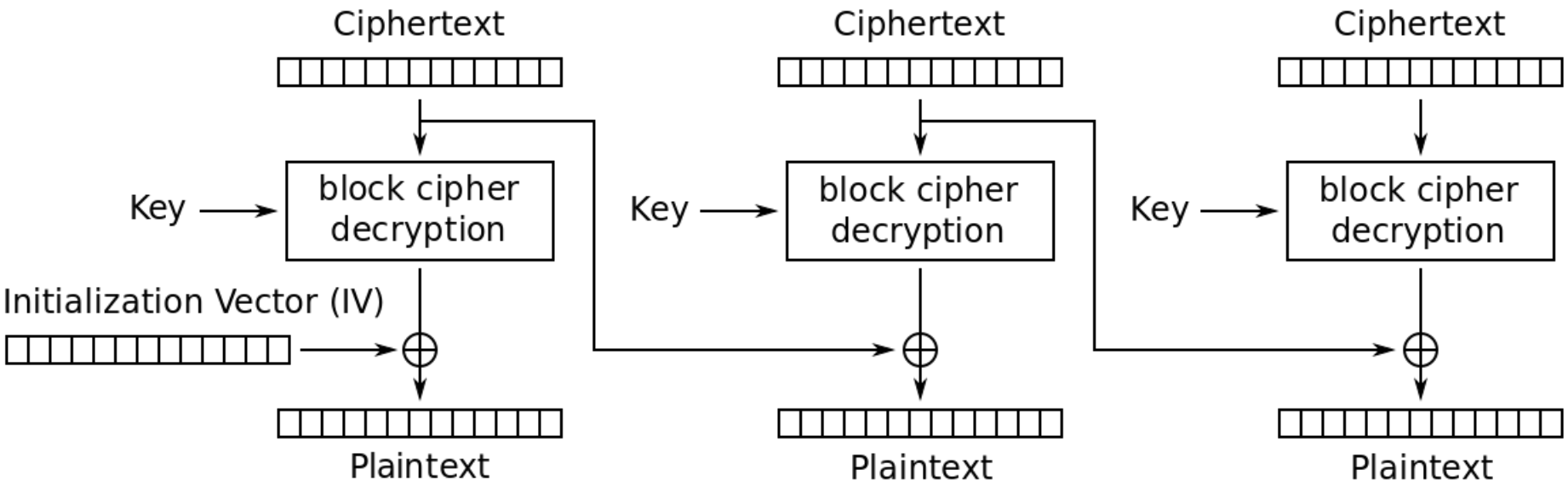
- **Padding Oracle** On Downgraded Legacy Encryption
- Δουλεύει μόνο εναντίον block ciphers
- Σε CBC mode
- Όλο το HTTPS λειτουργεί συνήθως έτσι
- Χρησιμοποιεί ευπάθεια στο padding

Ας θυμηθούμε το CBC

- Cipher Block Chaining
- Μας επιτρέπει να κρυπτογραφούμε με ασφάλεια χρησιμοποιώντας ένα block cipher ως building block



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

CBC AES ορολογία

AES: $\{0, 1\}^{128} \leftrightarrow \{0, 1\}^{128}$

$\mathbf{E_K}$: Συνάρτηση AES encryption με συμμετρικό κλειδί $K \in \{0, 1\}^{128}$

$\mathbf{D_K}$: Συνάρτηση AES decryption με συμμετρικό κλειδί $K \in \{0, 1\}^{128}$

$\mathbf{C_i}$: *i*-οστό ciphertext block (16 bytes) με $i = 1 \ldots n$

$\mathbf{P_i}$: *i*-οστό plaintext (16 bytes)

IV: initialization vector

CBC AES εξισώσεις

Κρυπτογράφηση: Δίνονται P_i

$IV \xleftarrow{\$} \{0, 1\}^{128}$

$C_0 = IV$

$C_i = E_K(P_i \oplus C_{i-1})$

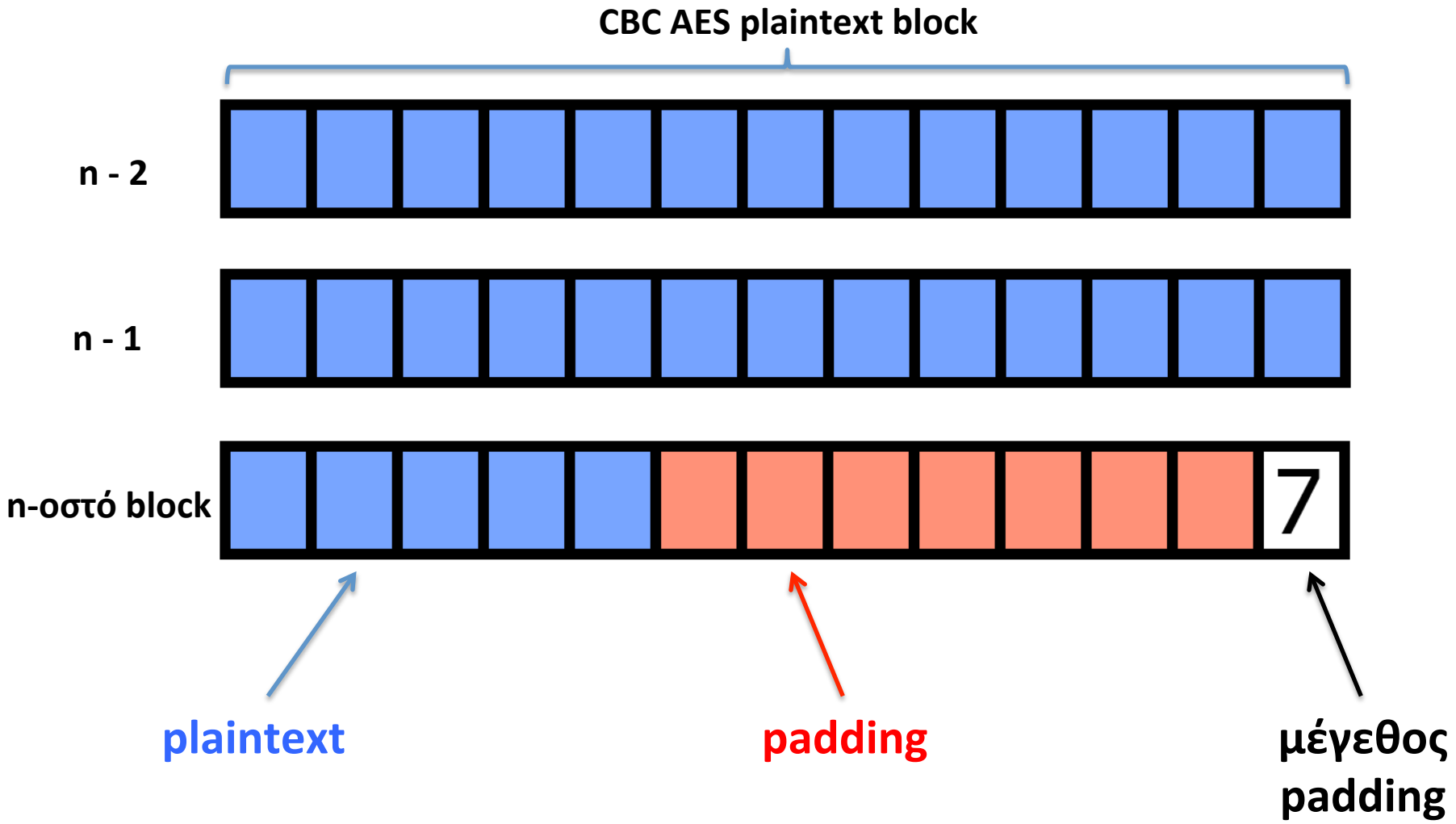
Αποκρυπτογράφηση: Δίνονται C_i

$P_i = D_K(C_i) \oplus C_{i-1}$

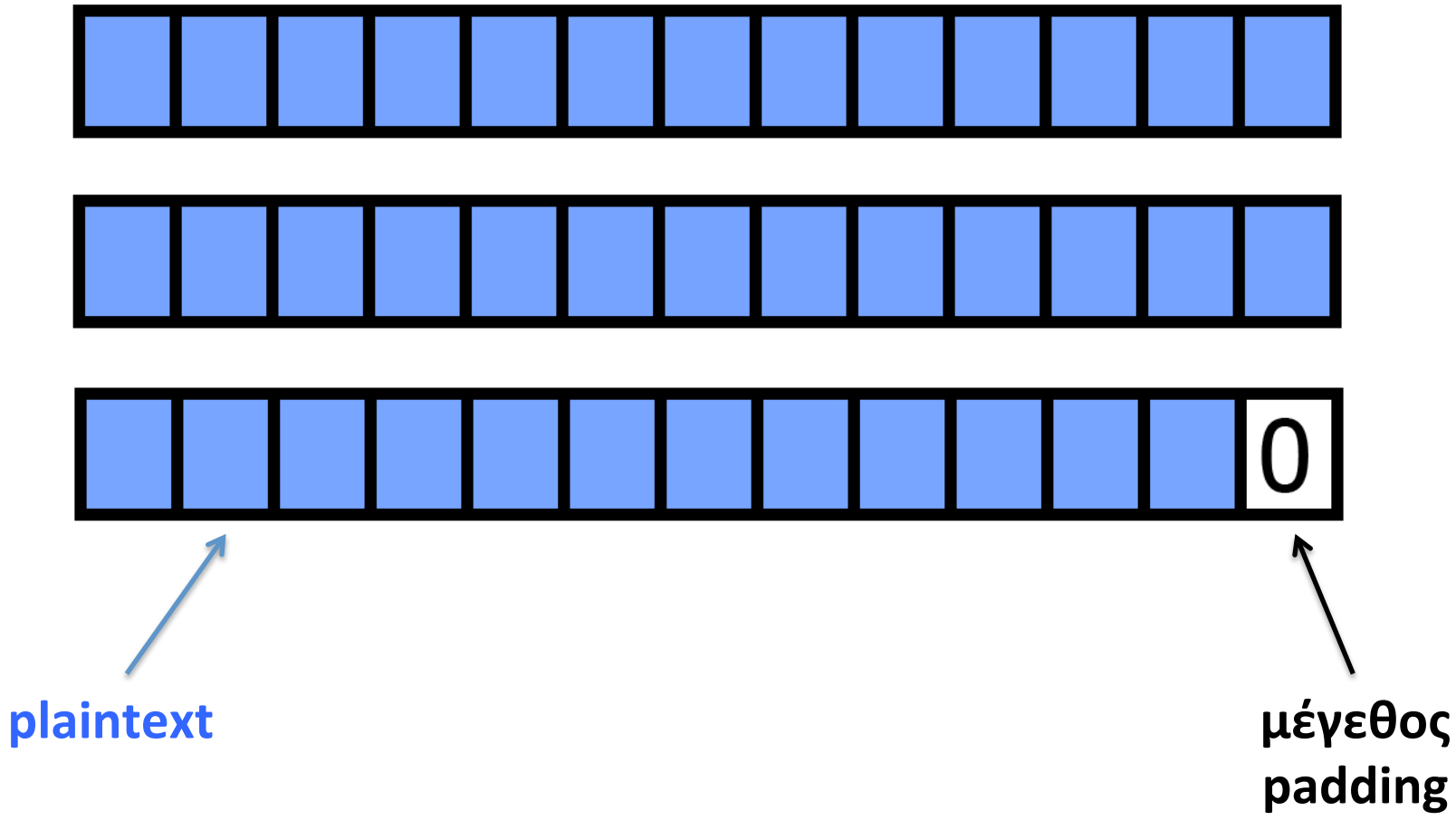
Padding στο SSLv3

- Για να δουλέψει το AES CBC, πρέπει το input στο encryption να είναι πολλαπλάσιο του 128
- Γι' αυτό προσθέτουμε padding:
  - Στο τέλος του plaintext μπορούν να προστεθούν από 0 έως 15 bytes τυχαίων τιμών
  - Μετά από αυτά προστίθεται ένα ακόμη που δείχνει πόσα τυχαία bytes είχαν προστεθεί πιο πριν (με τιμή 0...15)
  - Συνολικά προστίθενται από 1 έως 16 bytes

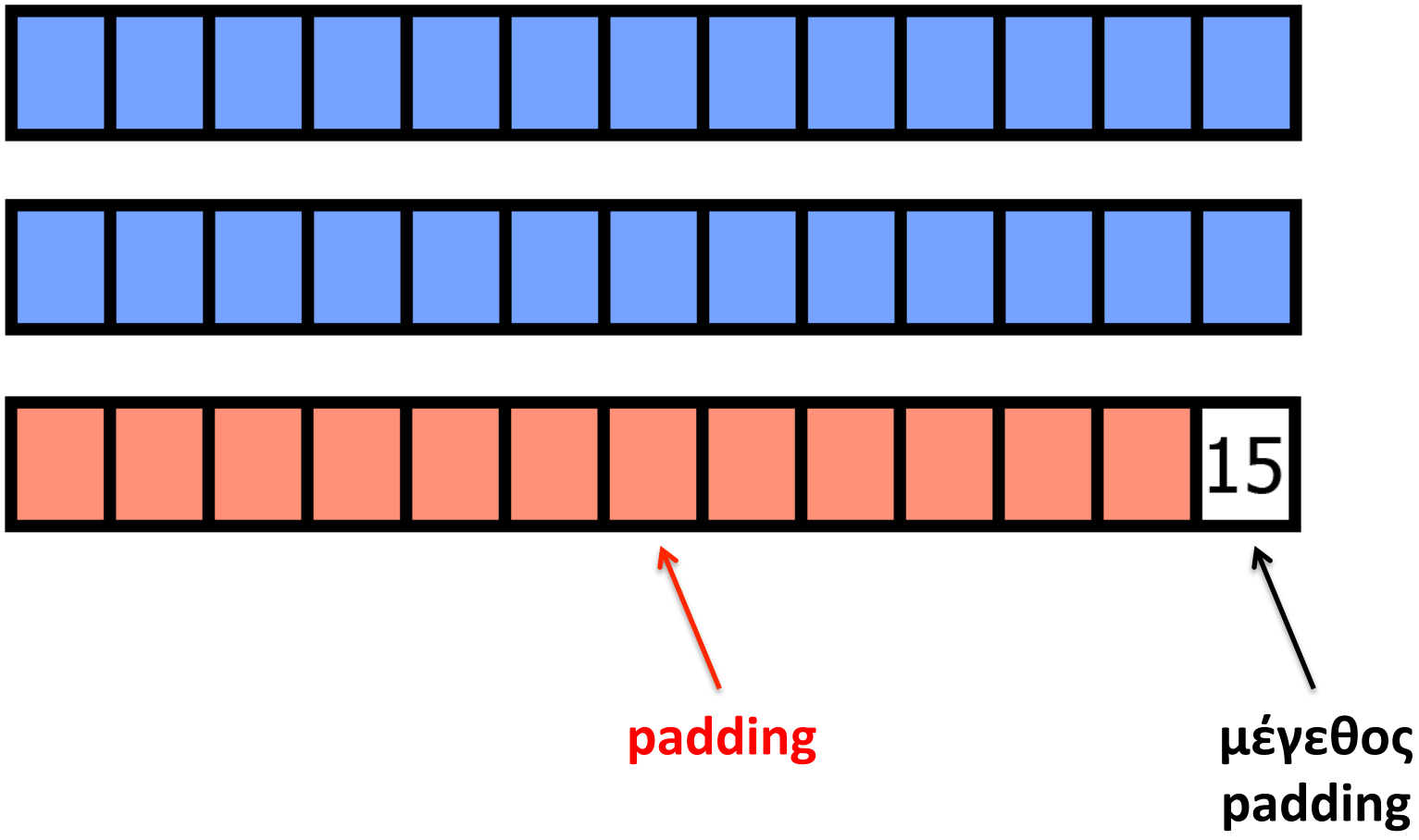
Padding πριν την κρυπτογράφηση



Padding πριν την κρυπτογράφηση



Padding πριν την κρυπτογράφηση



Αφαίρεση του padding

- Ο server κατά τη λήψη του request αφαιρεί το padding ως εξής:
- Αρχικά αποκρυπτογραφεί σύμφωνα με AES CBC
- Κοιτάει το τελευταίο **αποκρυπτογραφημένο** byte του τελευταίου block
  - Αν είναι εκτός [0, 16) απορρίπτει το request
- Αφαιρεί όσα bytes λέει το τελευταίο αποκρυπτογραφημένο byte

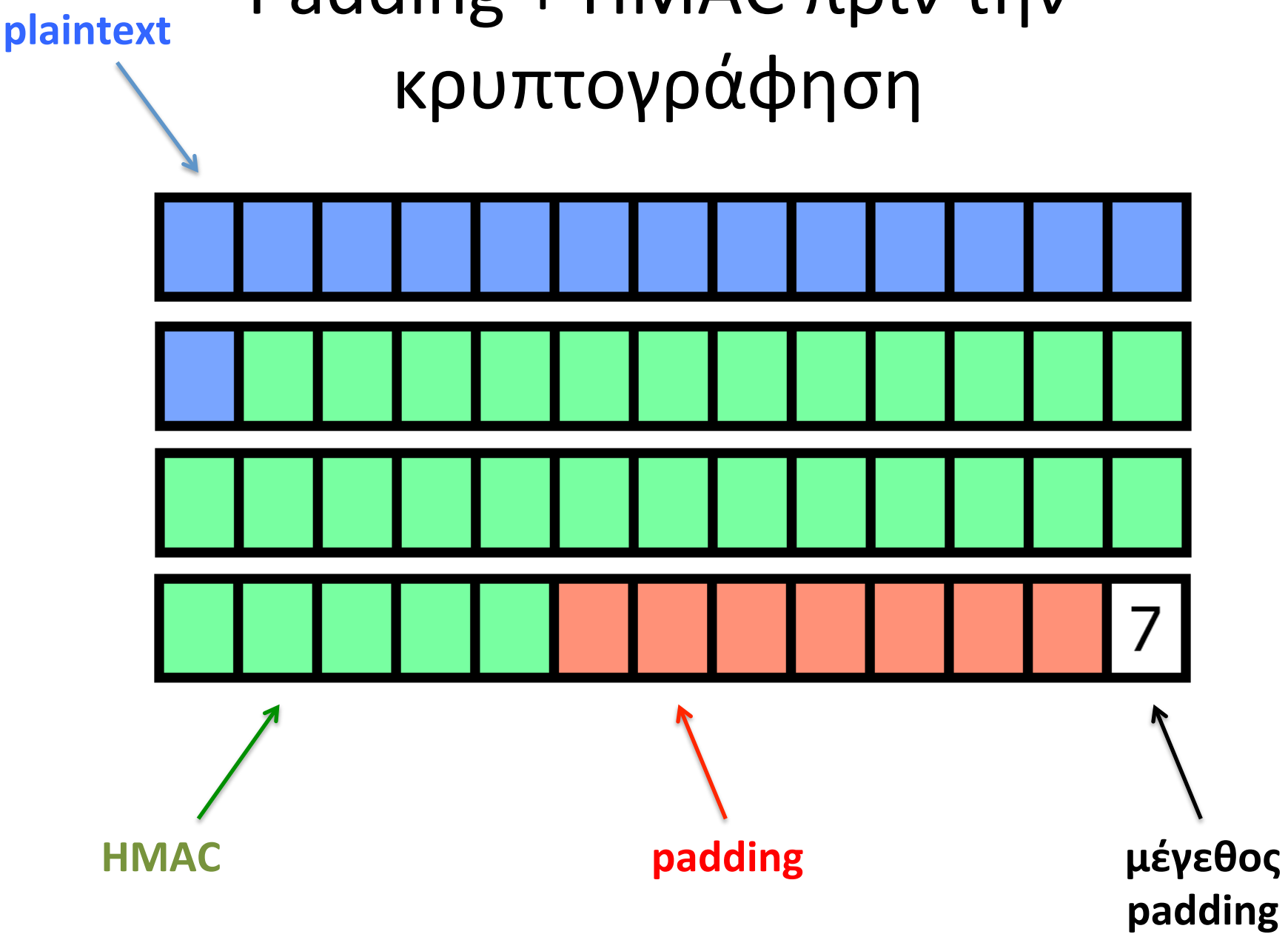
HMAC στο SSLv3

- Στο SSLv3 το plaintext πιστοποιείται με HMAC
- Με AES CBC κρυπτογραφείται το:

`plaintext || HMAC`

- Το HMAC έχει σταθερό μήκος 20 bytes
- Το HMAC προστίθεται μετά το plaintext αλλά πριν το padding
- Το "plaintext || HMAC" αντιμετωπίζεται από το AES CBC ως plaintext

Padding + HMAC πριν την κρυπτογράφηση

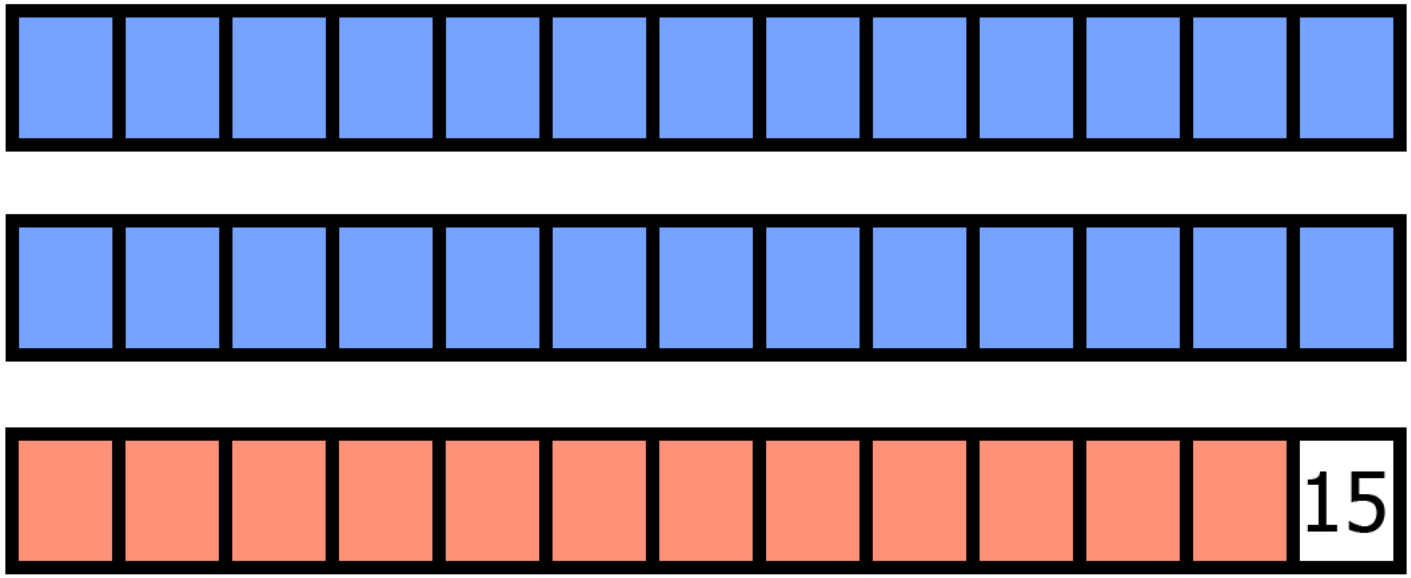


Επιβεβαίωση του HMAC

- Ο Bob (server) σε επίπεδο transport / SSL ελέγχει αν το HMAC είναι σωστό και απαντά **μόνο** αν όντως είναι
- Σε διαφορετική περίπτωση απορρίπτει το request
- Η Eve μπορεί να εντοπίσει αν υπήρξε απάντηση επειδή ελέγχει το δίκτυο

Στοίχιση του padding

- Η Eve για διευκόλυνσή της επιθυμεί το padding στο request που θα στείλει η Alice στον Bob να είναι ακριβώς 15 bytes



Στοίχιση του padding

- Μπορεί να το πετύχει επειδή ελέγχει το σώμα του POST
  - Αυτό είναι **chosen plaintext**!
- Στέλνει πολλά requests από τον υπολογιστή της Alice παρακολουθώντας το δίκτυο
- Αυξάνει το σώμα κατά 1 byte τη φορά
- Μετράει πόσα blocks περνούν από το δίκτυο
- Σταματά όταν αυξηθεί το πλήθος των blocks

Κώδικας στοίχισης Eve

```
URL = 'https://facebook.com/sendmessage';
alignment_amount = 0;
body = 'target=Bob&text=Hi&nonce=A';
make_alice_do_request(URL, body);
prev_num_blocks = read_encrypted_network(…)
do {
    body += 'A';
    ++alignment_amount;
    make_alice_do_request(URL, body);
    num_blocks = read_encrypted_network(…)
} while (num_blocks == prev_num_blocks);
```

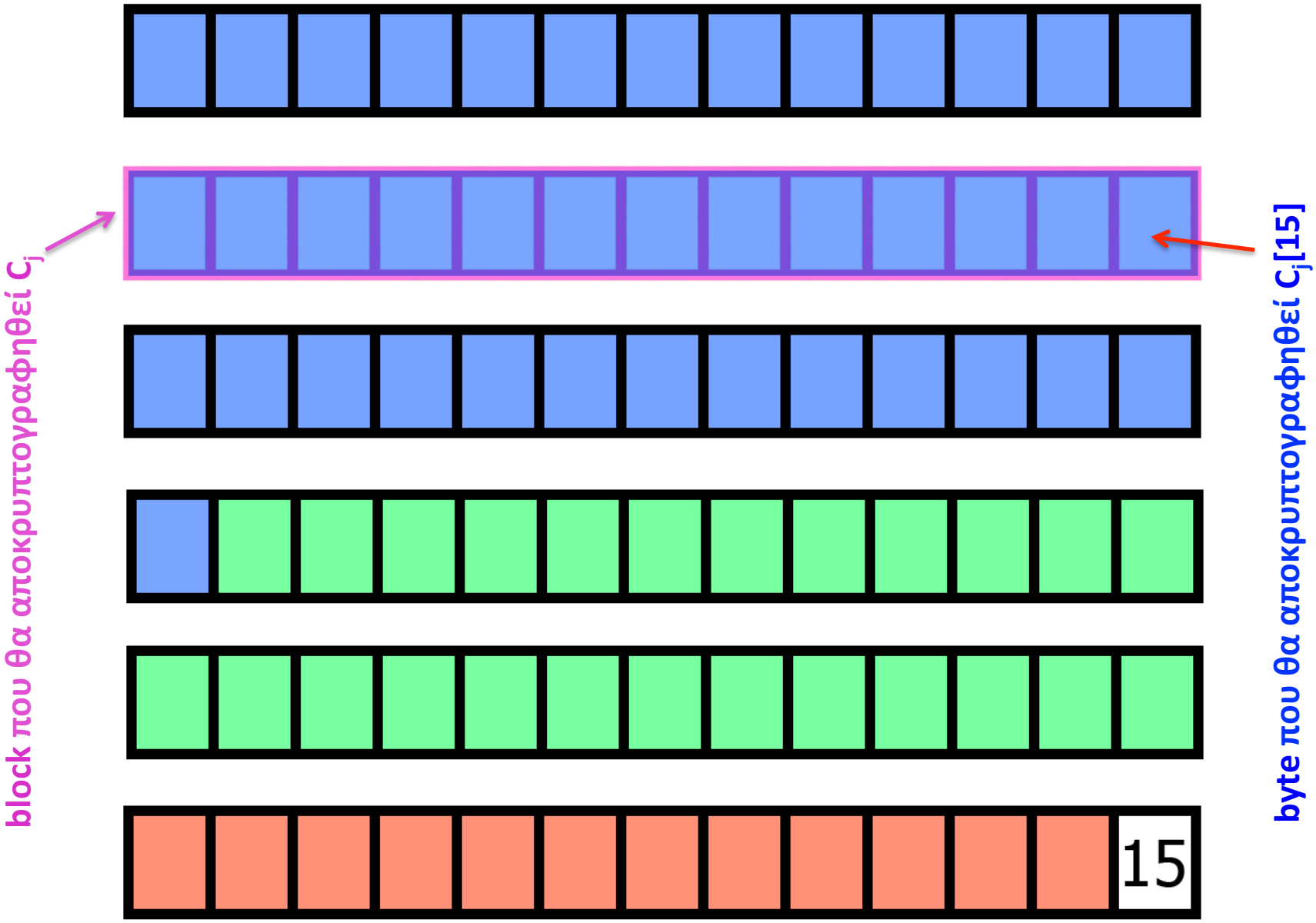
Εύρεση του body που χρειάζεται για τη στοίχιση

Χρόνος εκτέλεσης στοίχισης

- Θα χρειαστούν το πολύ B requests όπου B είναι το μέγεθος ενός block
- Ο χρόνος εκτέλεσης είναι O(B)
- Τρέχει μόνο μία φορά στην αρχή

Αποκρυπτογραφώντας ένα byte

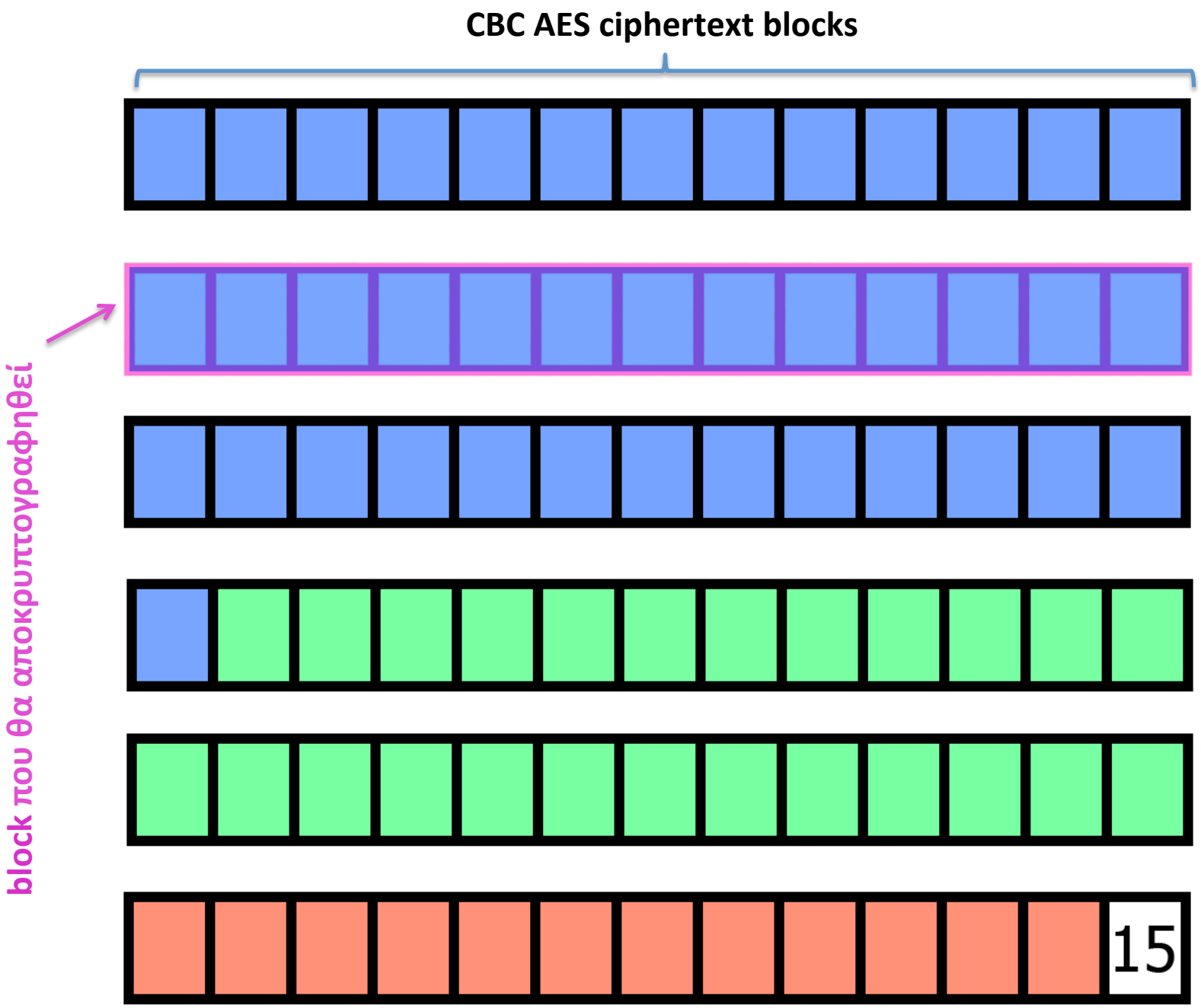
- Πλέον η Eve μπορεί να αποκρυπτογραφήσει το **τελευταίο** byte ενός block της επιλογής της
- Έστω ότι επιθυμεί να αποκρυπτογραφήσει το j-οστό block, δηλαδή το C_j
- Θα μάθει το $P_j[15]$


block που θα αποκρυπτογραφηθεί C_j
byte που θα αποκρυπτογραφηθεί $C_j[15]$
15

Αποκρυπτογραφώντας ένα byte

- Η Eve αντικαθιστά στο request της Alice όπως περνάει από το δίκτυο το τελευταίο κρυπτογραφημένο block **δηλαδή το padding block** με το κρυπτογραφημένο block που επιθυμεί να αποκρυπτογραφήσει

- $C_j \leftrightarrow C_n$


CBC AES ciphertext blocks
block που θα αποκρυπτογραφηθεί
15

Ο server επιβεβαιώνει το HMAC

- Αρχικά, ο server αρχικά αποκρυπτογραφεί όλα τα CBC AES blocks που έλαβε
- Αφαιρεί το padding με βάση την τιμή του **αποκρυπτογραφημένο** τελευταίο byte το οποίο όμως η Eve έχει αντικαταστήσει!

Ο server επιβεβαιώνει το HMAC

- Ποιο είναι το τελευταίο αποκρυπτογραφημένο byte b?
  - Είναι b >= 16? Τότε το request απορρίπτεται ως λάθος
  - Είναι b < 15? Τότε το HMAC δεν θα περάσει!
  - Είναι b = 15? Τότε το HMAC θα περάσει!
- Όμως η Eve γνωρίζει αν το request πέρασε βλέποντας αν υπήρξε απάντηση
- Συνεπώς η Eve συμπεραίνει ότι b = 15

Κώδικας αποκρυπτογράφησης ενός byte που τρέχει η Eve

```
URL = 'https://facebook.com/
sendmessage';
```

σταθερά, έχει υπολογιστεί ήδη για alignment

διαφορετικό κάθε φορά λόγω αλλαγμένου IV

```
do {
    make_alice_do_request(URL, body);
    intercepted = intercept_request();
    swap_blocks(intercepted);
    forward_request(intercepted);
} while (!check_network_response());
```

padding oracle

Παίρνοντας την τιμή ενός byte

$$D_K(C_j)[15] \oplus C_{n-1}[15] = 15$$

$$(P_j \oplus C_{j-1})[15] \oplus C_{n-1}[15] = 15$$

$$P_j[15] \oplus C_{j-1}[15] \oplus C_{n-1}[15] = 15$$

$$P_j[15] = C_{j-1}[15] \oplus C_{n-1}[15] \oplus 15$$

Χρόνος εκτέλεσης για 1 byte

- Παρατηρούμε ότι το C_{j-1} είναι κατανεμημένο περίπου ομοιόμορφα λόγω τυχαίας επιλογής IV
- Αυτή η διαδικασία θα πάρει αναμενόμενο χρόνο $O(|\Sigma|)$

  $\Sigma = [0, 256)$, αλφάβητο για ένα byte
- Η Eve θα χρειαστεί να βάλει την Alice να στείλει αναμενόμενα $|\Sigma|/2$ πλήθος requests

Η Eve ελέγχει τόσο το **URL** όσο και το **σώμα** του POST request:

POST /**sendmessage** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob

Αποκρυπτογραφώντας όλα τα cookies

- Η Eve ελέγχει:
  - Το block alignment με το σώμα του POST
  - Τη θέση του κάθε byte του cookie εντός block μέσω του URL

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?A** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

essage=Hi&target=Bob

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?AA** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

ssage=Hi&target=Bob

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?AAA** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

sage=Hi&target=Bob

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?AAAAAA** HTTP/1.1

Host: www.facebook.com

Ελέγχει ποιο είναι το τελευταίο byte στο block

…

Cookie: boring=boring&auth=secret&x=y...

…

e=Hi&target=Bob

Διατηρεί συνολικό μέγεθος plaintext σταθερό

Σταθερό σημείο για το πλήθος blocks

POST /**sendmessage?** HTTP/1.1

Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob

Μαθαίνοντας όλα τα bytes

POST /**sendmessage?** HTTP/1.1

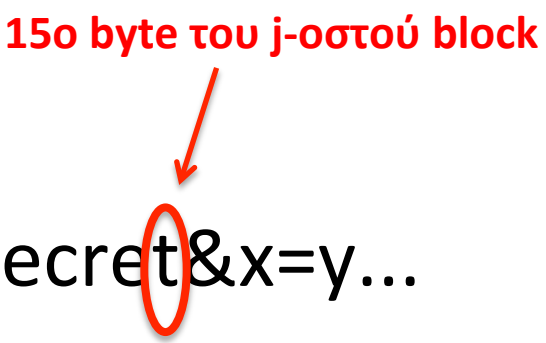
Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

message=Hi&target=Bob



Μαθαίνοντας όλα τα bytes

POST /**sendmessage?A** HTTP/1.1

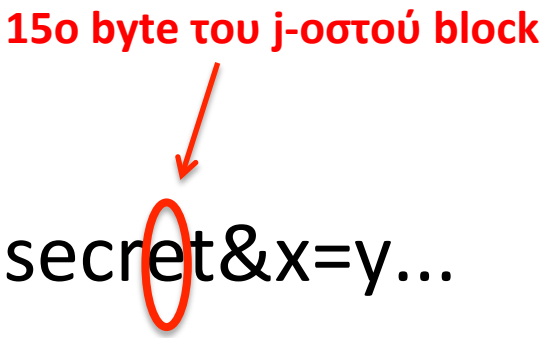
Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

essage=Hi&target=Bob



Μαθαίνοντας όλα τα bytes

POST /**sendmessage?AA** HTTP/1.1

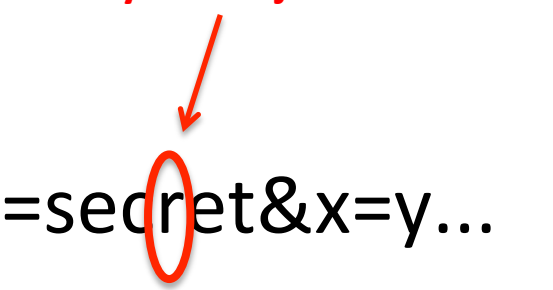
Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

ssage=Hi&target=Bob



Μαθαίνοντας όλα τα bytes

POST /**sendmessage?AAA** HTTP/1.1

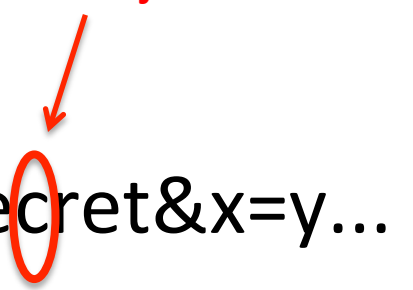
Host: www.facebook.com

…

Cookie: boring=boring&auth=secret&x=y...

…

sage=Hi&target=Bob



Μαθαίνοντας όλα τα bytes

POST /**sendmessage?AAAAAA** HTTP/1.1

Host: www.facebook.co

…

Cookie: boring=boring&auth=secret&x=y...

…

e=Hi&target=Bob

Τελικός κώδικας Eve

```
p = '';
for (i = 0; i < m; ++i) {
    p += get_block_last_byte(
                        URL, body);
    URL += 'A';
    // remove last char
    body = body[1:];
}
return p;
```

Πολυπλοκότητα

- Στοίχηση: O(B)
- Εύρεση ενός byte: O(|Σ|)
- Εύρεση όλων των bytes: O(m|Σ|)
  - m πλήθος bytes που θέλουμε να διαβάσουμε
- Συνολικός χρόνος: O(m|Σ| + B)
- Αναμενόμενο συνολικό πλήθος requests:
- m|Σ|/2 + B/2

Πολυπλοκότητα

Πρακτικά νούμερα:

m = 16 για ένα authentication cookie

$|\Sigma| = 256$

B = 128

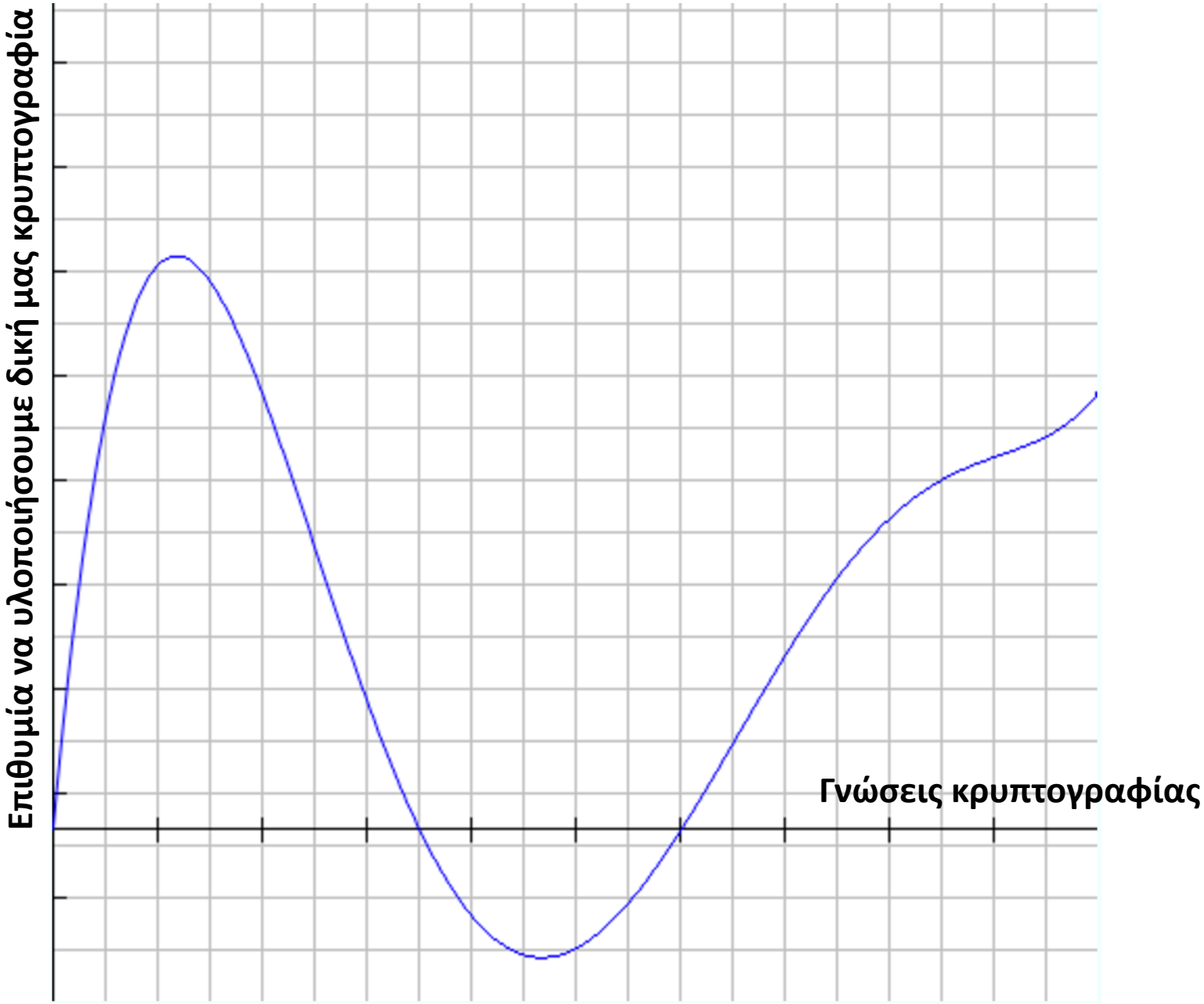
$m|\Sigma|/2 + B/2 = 2112$ requests

~ 30 λεπτά

Ιδέες για αποφυγή;

Αποφυγή POODLE

Encrypt then MAC :)


Επιθυμία να υλοποιήσουμε δική μας κρυπτογραφία
Γνώσεις κρυπτογραφίας

Τι να μας μείνει...

- Η κρυπτογραφία είναι δύσκολη
- Μην φτιάχνετε τους δικούς σας αλγορίθμους για production συστήματα!
  - "Don't roll your own crypto"
- Μην χρησιμοποιείτε απλά έτοιμους αλγορίθμους - χρησιμοποιήστε **έτοιμα συστήματα**
- Αν στον κώδικά σας γράφετε 'A', 'E', 'S', τότε κάνετε λάθος

Μάθαμε

- Επιθέσεις MitM
- POODLE
- BREACH
- Πρακτικά chosen plaintext attacks
- Εκμετάλλευση μικρής πιθανότητας αποκάλυψης bits

Την επόμενη φορά...

Κρυπτογραφικά εργαλεία στην πράξη

- GPG: Κρυπτογραφούμε τα email μας
- OTR: Κρυπτογραφούμε το chat μας
- Tor: Ανώνυμη περιήγηση στο web