



Authenticated Encryption

Active attacks on
CPA-secure encryption

Recap: the story so far

Confidentiality: semantic security against a CPA attack

- Encryption secure against **eavesdropping only**

Integrity:

- Existential unforgeability under a chosen message attack
- CBC-MAC, HMAC, PMAC, CW-MAC

This module: encryption secure against **tampering** (*active adversary*)

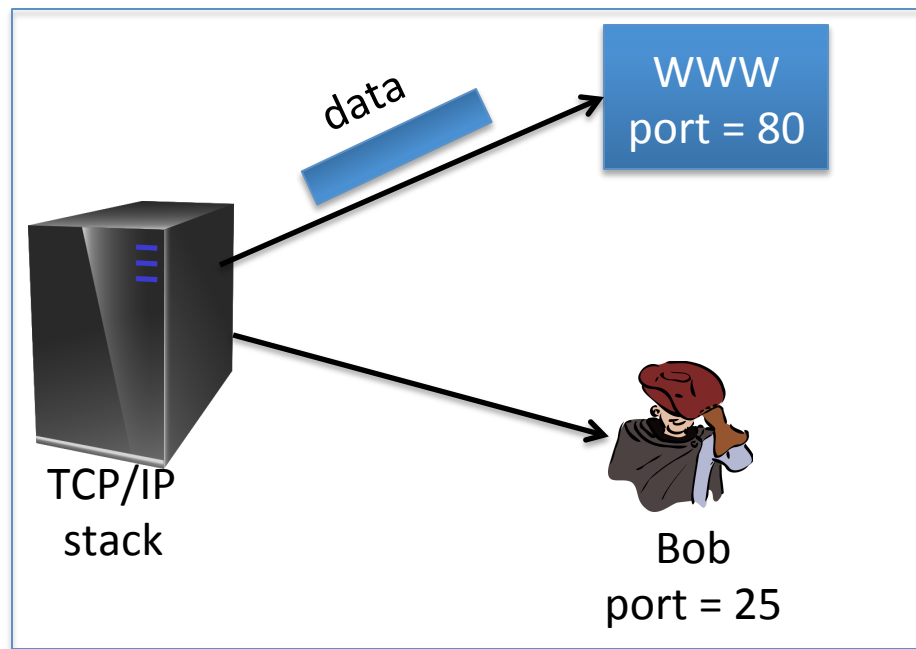
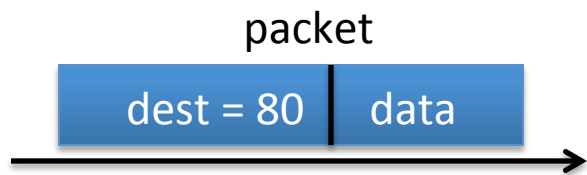
- Ensuring both confidentiality and integrity

Sample tampering attacks

TCP/IP: (highly abstracted)



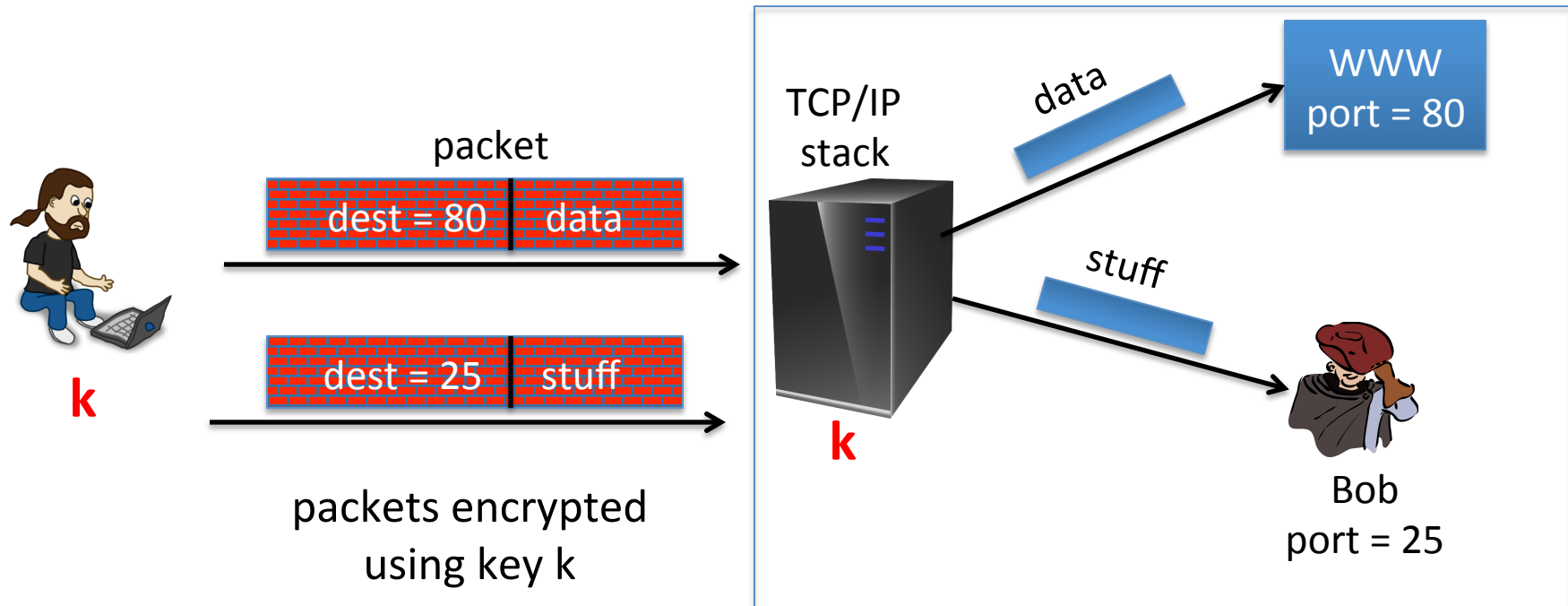
source machine



destination machine

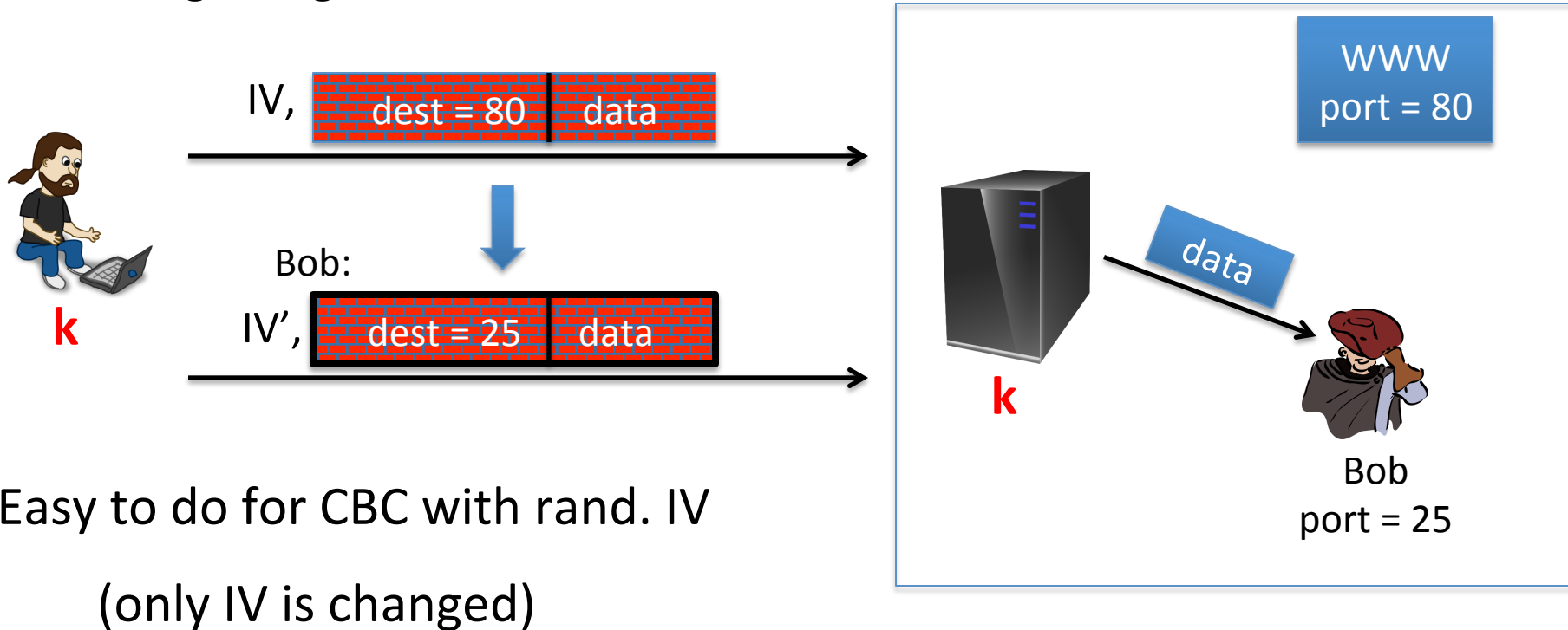
Sample tampering attacks

IPsec: (highly abstracted)



Reading someone else's data

Note: attacker obtains decryption of any ciphertext beginning with "dest=25"





Encryption is done with CBC with a random IV.

What should IV' be?

$$m[0] = D(k, c[0]) \oplus IV = \text{"dest=80..."}$$

$IV' = IV \oplus (...25...)$

$IV' = IV \oplus (...80...)$

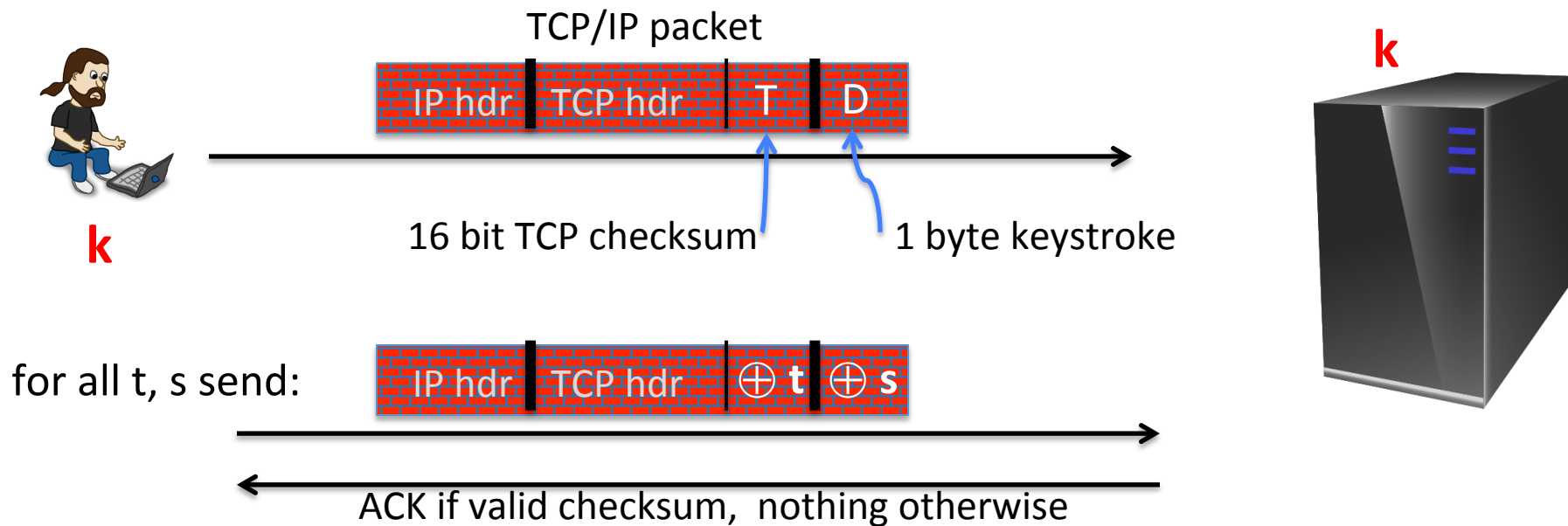
$IV' = IV \oplus (...80...) \oplus (...25...)$ ←

It can't be done

$$\begin{aligned}
 \underline{D(k, c[0]) \oplus IV'} &= \overbrace{D(k, c[0]) \oplus IV}^{...80...} \oplus \cancel{80} \oplus \cancel{25} \\
 &= ...25...
 \end{aligned}$$

An attack using only network access

Remote terminal app.: each keystroke encrypted with CTR mode



$$\{ \text{checksum}(\text{hdr}, D) = t \oplus \text{checksum}(\text{hdr}, D \oplus s) \} \Rightarrow \text{can find } D$$

The lesson

CPA security cannot guarantee secrecy under active attacks.

Only use one of two modes:

- If message needs integrity but no confidentiality:
use a **MAC**
- If message needs both integrity and confidentiality:
use **authenticated encryption** modes (this module)

End of Segment



Authenticated Encryption

Definitions

Goals


An **authenticated encryption** system (E,D) is a cipher where

As usual: $E: K \times M \times N \rightarrow C$

but $D: K \times C \times N \rightarrow M \cup \{\perp\}$

Security: the system must provide

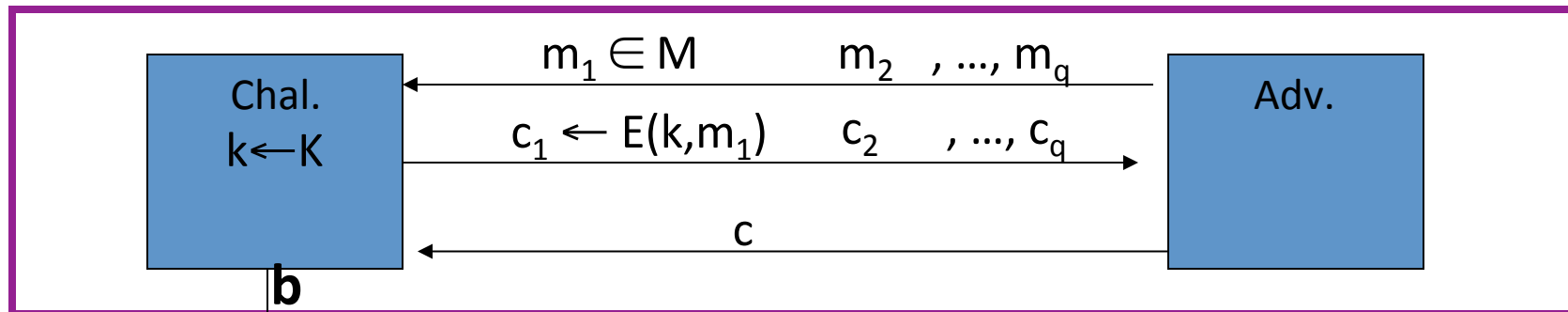
- sem. security under a CPA attack, and
- **ciphertext integrity**:
attacker cannot create new ciphertexts that decrypt properly



ciphertext
is rejected

Ciphertext integrity

Let (E,D) be a cipher with message space M .



$$\begin{cases} \mathbf{b}=1 & \text{if } D(k,c) \neq \perp \text{ and } c \notin \{c_1, \dots, c_q\} \\ \mathbf{b}=0 & \text{otherwise} \end{cases}$$

Def: (E,D) has **ciphertext integrity** if for all “efficient” A :

$$\text{Adv}_{\text{CI}}[A,E] = \Pr[\text{Chal. outputs } 1] \text{ is “negligible.”}$$

Authenticated encryption

Def: cipher (E,D) provides authenticated encryption (AE) if it is

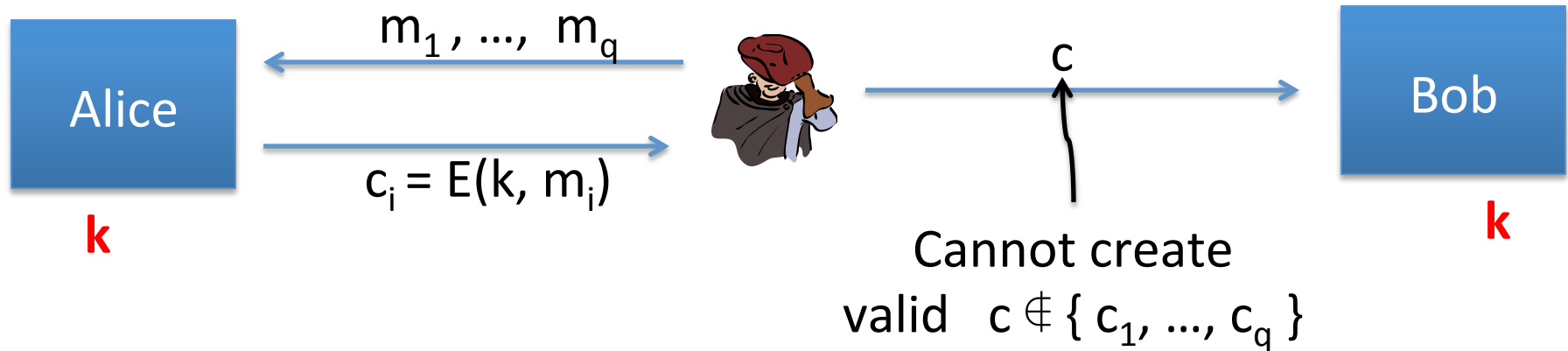
- (1) semantically secure under CPA, and
- (2) has ciphertext integrity

Bad example: CBC with rand. IV does not provide AE

- $D(k, \cdot)$ never outputs \perp , hence adv. easily wins CI game

Implication 1: authenticity

Attacker cannot fool Bob into thinking a message was sent from Alice



\Rightarrow if $D(k, c) \neq \perp$ Bob knows message is from someone who knows k
(but message could be a replay)

Implication 2

Authenticated encryption \Rightarrow

Security against **chosen ciphertext attacks**
(next segment)

End of Segment



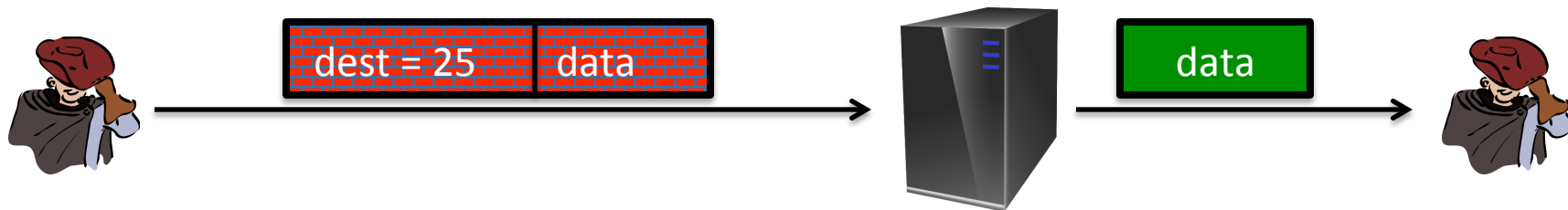
Authenticated Encryption

Chosen ciphertext
attacks

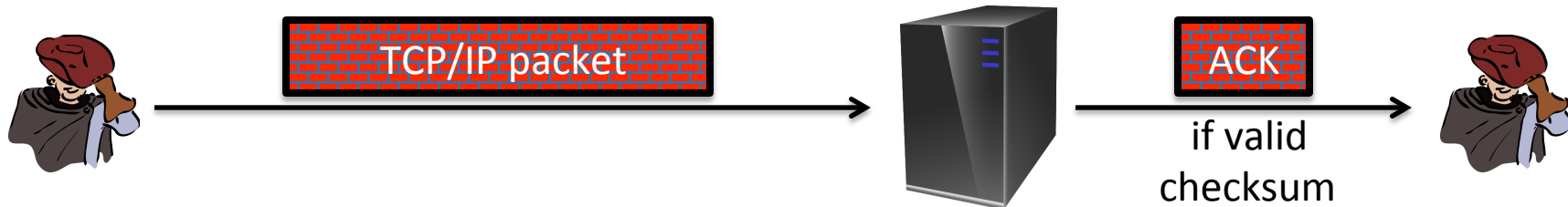
Example chosen ciphertext attacks

Adversary has ciphertext c that it wants to decrypt

- Often, adv. can fool server into decrypting **certain** ciphertexts (not c)



- Often, adversary can learn partial information about plaintext



Chosen ciphertext security

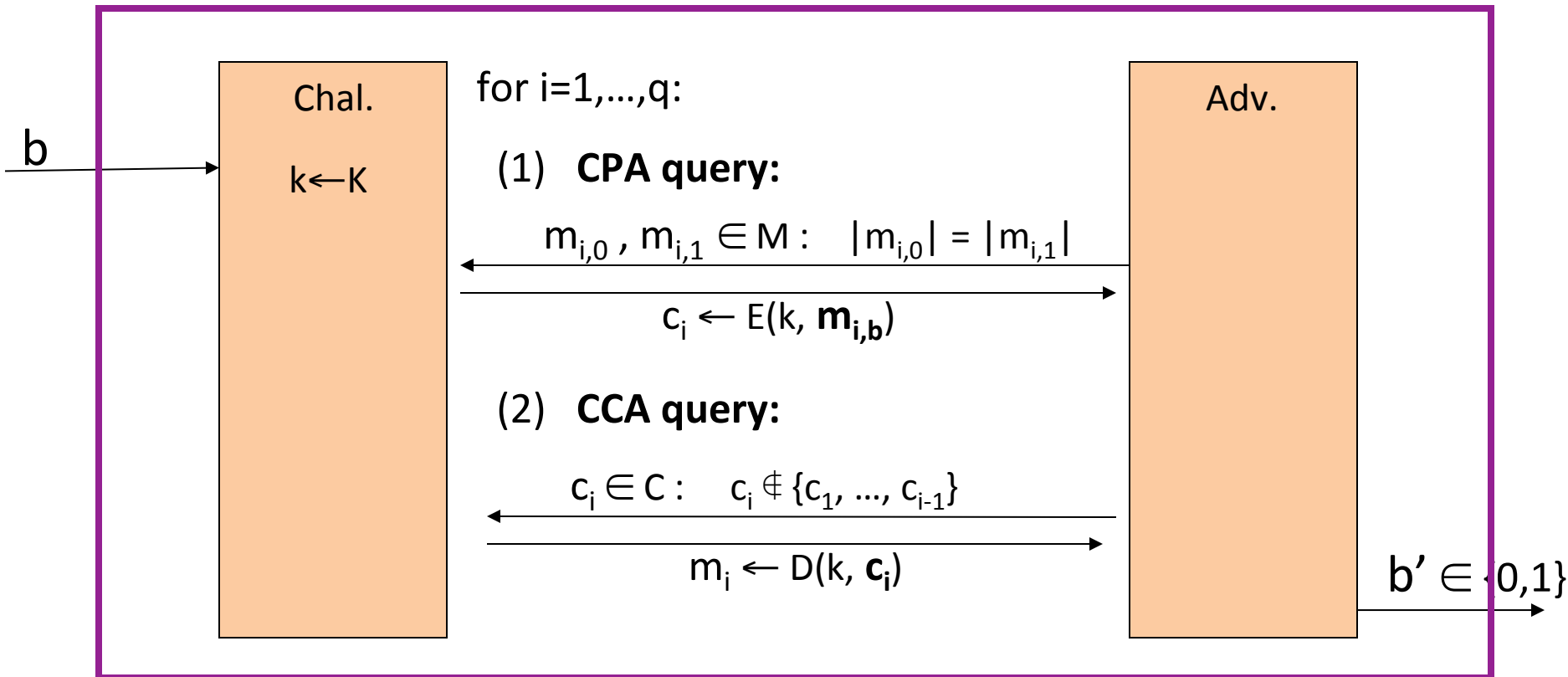
Adversary's power: both CPA and CCA

- Can obtain the encryption of arbitrary messages of his choice
- Can decrypt any ciphertext of his choice, other than challenge
(conservative modeling of real life)

Adversary's goal: Break semantic security

Chosen ciphertext security: definition

$E = (E, D)$ cipher defined over (K, M, C) . For $b=0,1$ define $\text{EXP}(b)$:

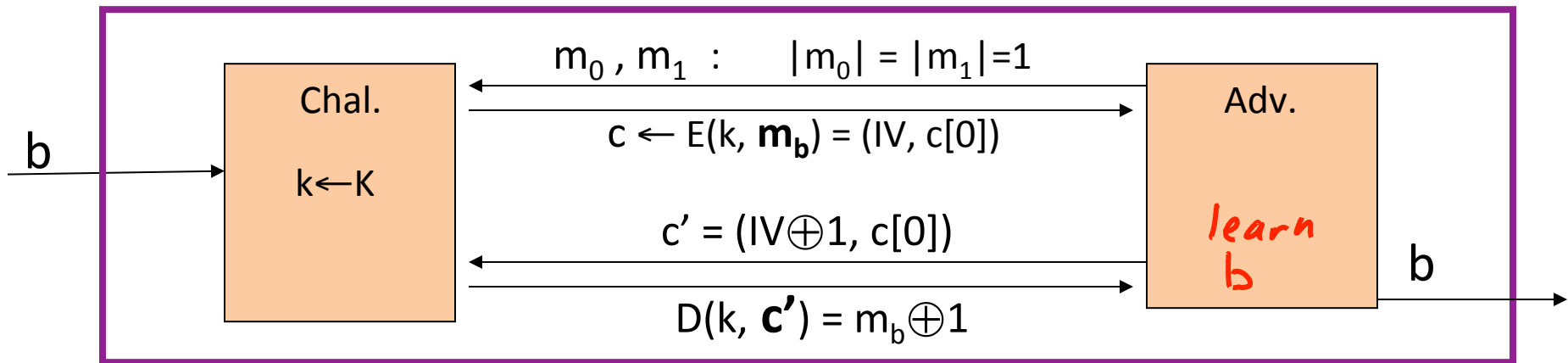


Chosen ciphertext security: definition

E is CCA secure if for all “efficient” A:

$$\text{Adv}_{\text{CCA}} [A, E] = \left| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \right| \text{ is “negligible.”}$$

Example: CBC with rand. IV is not CCA-secure



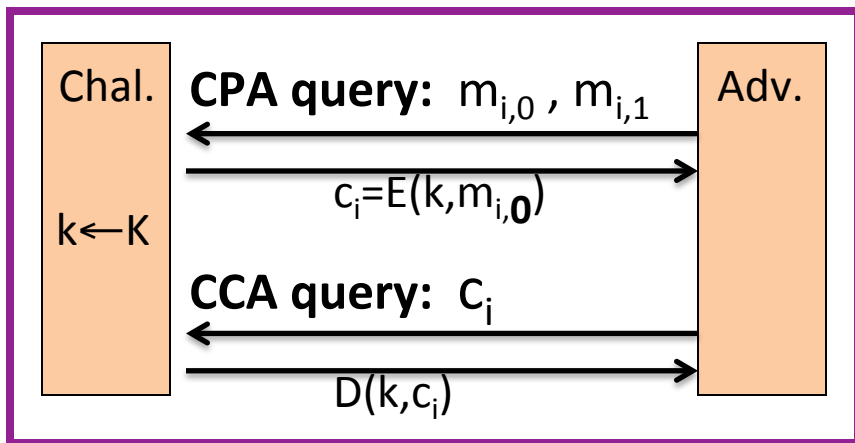
Authenticated enc. \Rightarrow CCA security

Thm: Let (E,D) be a cipher that provides AE.
Then (E,D) is CCA secure !

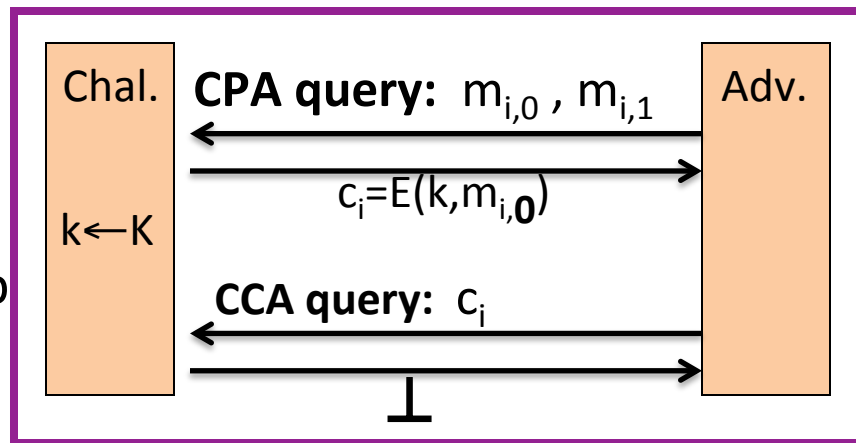
In particular, for any q -query eff. A there exist eff. B_1, B_2 s.t.

$$\text{Adv}_{\text{CCA}}[A,E] \leq 2q \cdot \text{Adv}_{\text{CI}}[B_1,E] + \text{Adv}_{\text{CPA}}[B_2,E]$$

Proof by pictures

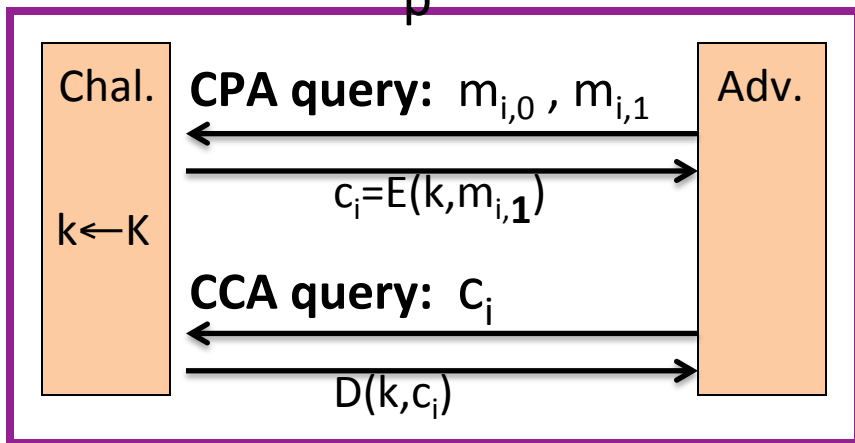


\approx_p

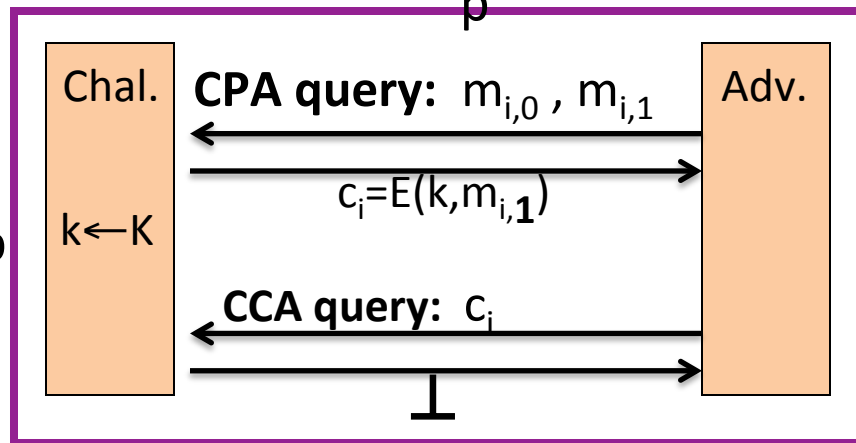


\approx_p

\approx_p



\approx_p



So what?

Authenticated encryption:

- ensures confidentiality against an active adversary that can decrypt some ciphertexts

Limitations:

- does not prevent replay attacks
- does not account for side channels (timing)

End of Segment



Authenticated Encryption

Constructions from
ciphers and MACs

... but first, some history

Authenticated Encryption (AE): introduced in 2000 [KY'00, BN'00]

Crypto APIs before then: (e.g. MS-CAPI) *crypto API*

- Provide API for CPA-secure encryption (e.g. CBC with rand. IV)
- Provide API for MAC (e.g. HMAC)

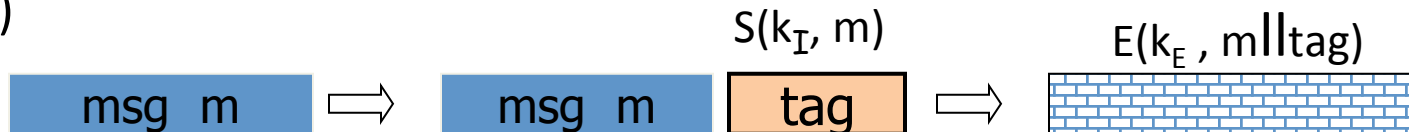
Every project had to combine the two itself without a well defined goal

- Not all combinations provide AE ...

Combining MAC and ENC (CCA)

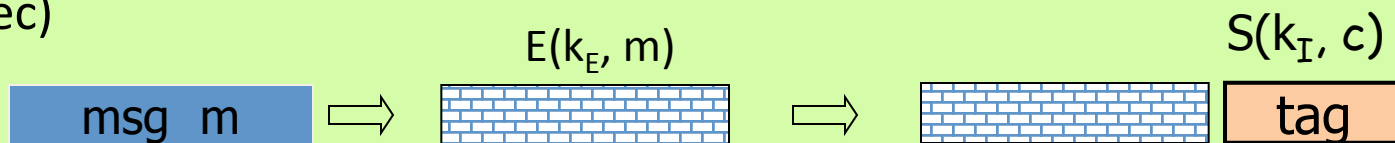
Encryption key k_E . MAC key = k_I

Option 1: (SSL)

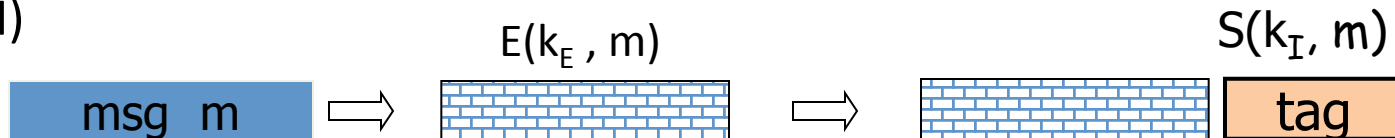


Option 2: (IPsec)

**always
correct**



Option 3: (SSH)



A.E. Theorems

Let (E,D) be CPA secure cipher and (S,V) secure MAC. Then:

1. **Encrypt-then-MAC:** always provides A.E.

2. **MAC-then-encrypt:** may be insecure against CCA attacks

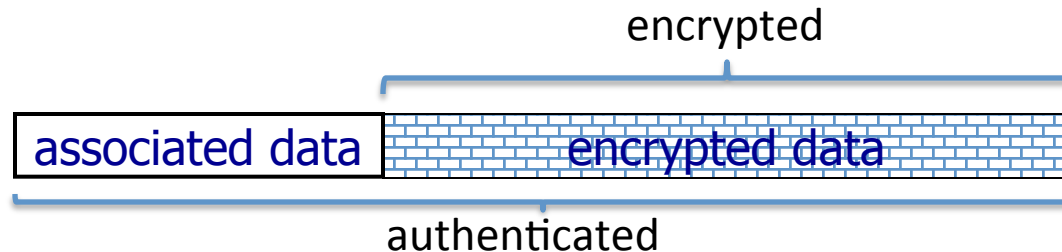
however: when (E,D) is rand-CTR mode or rand-CBC
M-then-E provides A.E.

for rand-CTR mode, one-time MAC is sufficient

Standards (at a high level)

- **GCM:** CTR mode encryption then CW-MAC
(accelerated via Intel's PCLMULQDQ instruction)
- **CCM:** CBC-MAC then CTR mode encryption (802.11i)
- **EAX:** CTR mode encryption then CMAC

All support AEAD: (auth. enc. with associated data). All are nonce-based.



An example API (OpenSSL)

```
int AES_GCM_Init(AES_GCM_CTX *ain,  
    unsigned char *nonce, unsigned long noncelen,  
    unsigned char *key, unsigned int klen )
```

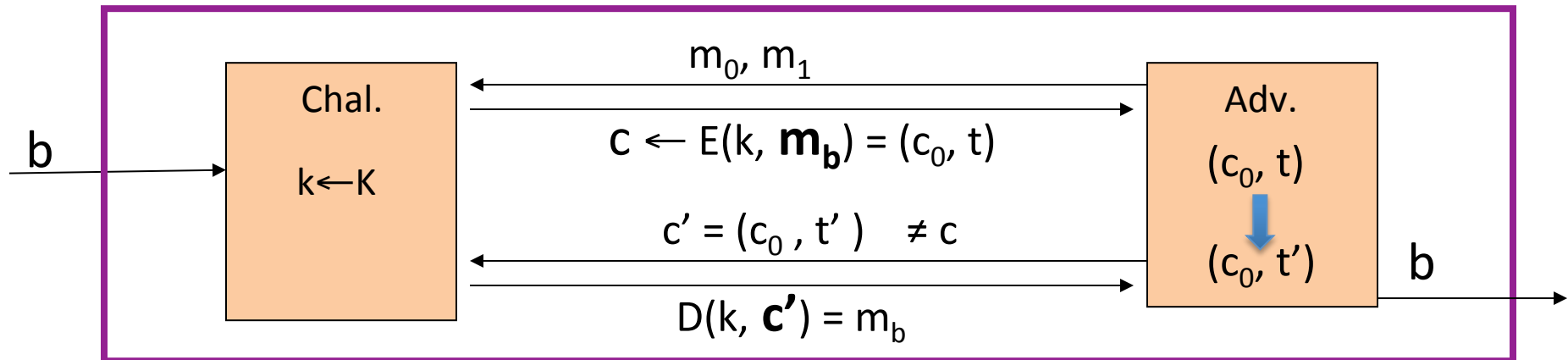
```
int AES_GCM_EncryptUpdate(AES_GCM_CTX *a,  
    unsigned char *aad, unsigned long aadlen,  
    unsigned char *data, unsigned long datalen,  
    unsigned char *out, unsigned long *outlen)
```

MAC Security -- an explanation

Recall: MAC security implies $(m, t) \not\Rightarrow (m, t')$

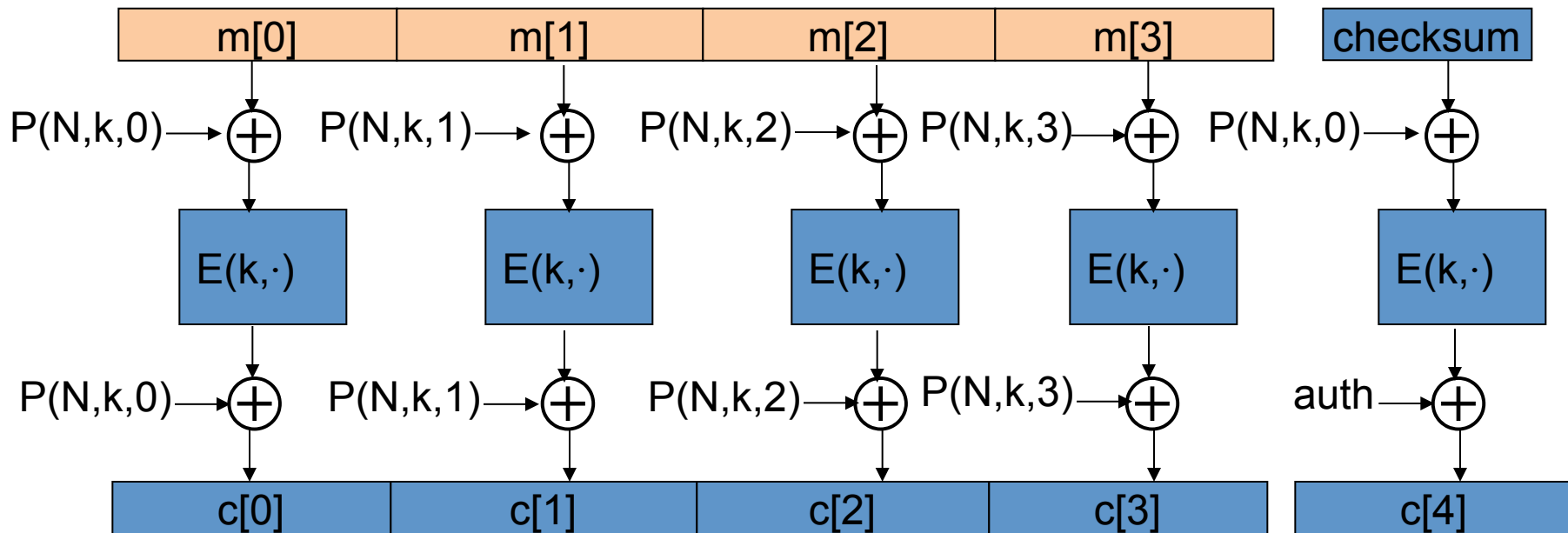
Why? Suppose not: $(m, t) \rightarrow (m, t')$

Then Encrypt-then-MAC would not have Ciphertext Integrity !!



OCB: a direct construction from a PRP

More efficient authenticated encryption: one $E()$ op. per block.



Performance:

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

<u>Cipher</u>	<u>code size</u>	<u>Speed (MB/sec)</u>		
AES/GCM	large**	108	AES/CTR	139
AES/CCM	smaller	61	AES/CBC	109
AES/EAX	smaller	61		
			AES/CMAC	109
AES/OCB		129*	HMAC/SHA1	147

* extrapolated from Ted Kravitz's results

** non-Intel machines

End of Segment



Authenticated Encryption

Case study: TLS

The TLS Record Protocol (TLS 1.2)



Unidirectional keys: $k_{b \rightarrow s}$ and $k_{s \rightarrow b}$

Stateful encryption:

- Each side maintains two 64-bit counters: $ctr_{b \rightarrow s}$, $ctr_{s \rightarrow b}$
- Init. to 0 when session started. $ctr++$ for every record.
- Purpose: replay defense

TLS record: encryption (CBC AES-128, HMAC-SHA1)

$$k_{b \rightarrow s} = (k_{\text{mac}}, k_{\text{enc}})$$



Browser side $\text{enc}(k_{b \rightarrow s}, \text{data}, \text{ctr}_{b \rightarrow s})$: *not transmitted in packet*

step 1: $\text{tag} \leftarrow S(k_{\text{mac}}, [++\text{ctr}_{b \rightarrow s} \parallel \text{header} \parallel \text{data}])$

step 2: pad [header || data || tag] to AES block size

step 3: CBC encrypt with k_{enc} and new random IV

step 4: prepend header

TLS record: decryption (CBC AES-128, HMAC-SHA1)

Server side $\text{dec}(k_{b \rightarrow s}, \text{record}, \text{ctr}_{b \rightarrow s})$:

step 1: CBC decrypt record using k_{enc}

step 2: check pad format: send **bad_record_mac** if invalid

step 3: check tag on $[++\text{ctr}_{b \rightarrow s} \parallel \text{header} \parallel \text{data}]$
send **bad_record_mac** if invalid

Provides authenticated encryption

(provided no other info. is leaked during decryption)

Bugs in older versions (prior to TLS 1.1)

IV for CBC is predictable: (chained IV)

IV for next record is last ciphertext block of current record.

Not CPA secure. (a practical exploit: BEAST attack)

Padding oracle: during decryption

if pad is invalid send **decryption failed** alert

if mac is invalid send **bad_record_mac** alert

⇒ attacker learns info. about plaintext (attack in next segment)

Lesson: when decryption fails, do not explain why

Leaking the length

The TLS header leaks the length of TLS records

- Lengths can also be inferred by observing network traffic

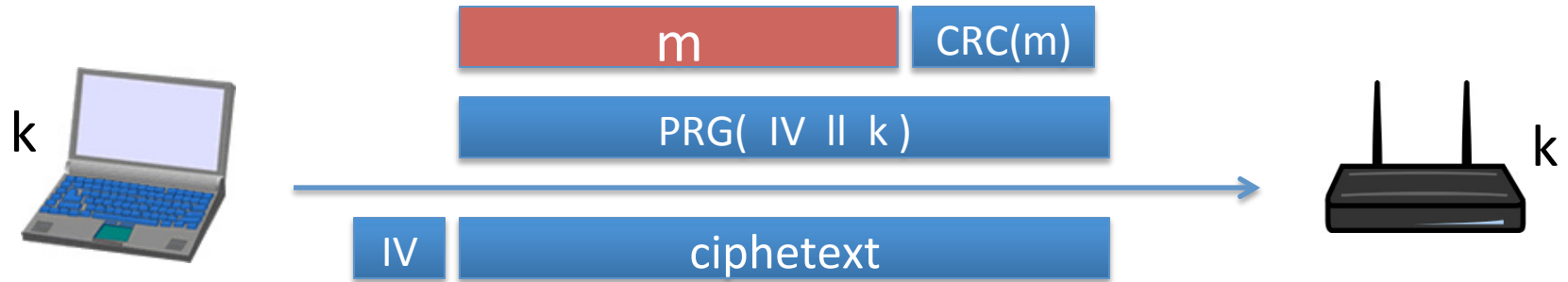
For many web applications, leaking lengths reveals sensitive info:

- In tax preparation sites, lengths indicate the type of return being filed which leaks information about the user's income
- In healthcare sites, lengths leaks what page the user is viewing
- In Google maps, lengths leaks the location being requested

No easy solution

802.11b WEP: how not to do it

802.11b WEP:



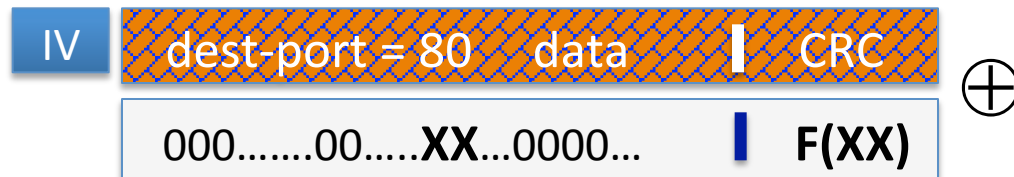
Previously discussed problems:

two time pad and related PRG seeds

Active attacks

Fact: CRC is linear, i.e. $\forall m, p: \text{CRC}(m \oplus p) = \text{CRC}(m) \oplus F(p)$

WEP ciphertext:



$XX = 25 \oplus 80$



Upon decryption: CRC is valid, but ciphertext is changed !!

End of Segment



Authenticated Encryption

CBC paddings attacks

Recap

Authenticated encryption: CPA security + ciphertext integrity

- Confidentiality in presence of **active** adversary
- Prevents chosen-ciphertext attacks

Limitation: cannot help bad implementations ... (this segment)

Authenticated encryption modes:

- Standards: GCM, CCM, EAX
- General construction: encrypt-then-MAC

The TLS record protocol (CBC encryption)

Decryption: $\text{dec}(k_{b \rightarrow s}, \text{record}, \text{ctr}_{b \rightarrow s})$:

step 1: CBC decrypt record using k_{enc}

step 2: check pad format: abort if invalid

step 3: check tag on $[++\text{ctr}_{b \rightarrow s} \parallel \text{header} \parallel \text{data}]$
abort if invalid

Two types of error:

- padding error
- MAC error



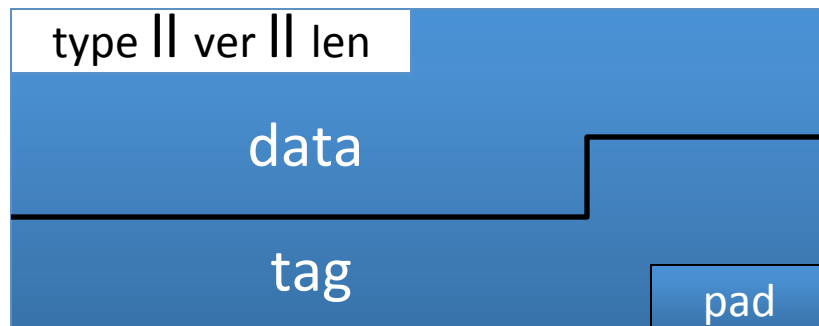
Padding oracle

Suppose attacker can differentiate the two errors
(pad error, MAC error):

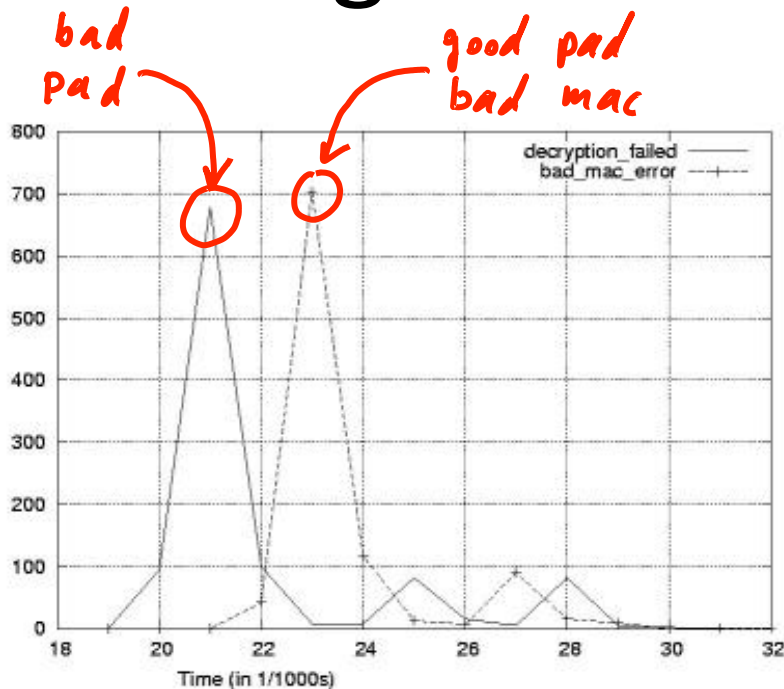
⇒ **Padding oracle:**

attacker submits ciphertext and learns if
last bytes of plaintext are a valid pad

Nice example of a
chosen ciphertext attack



Padding oracle via timing OpenSSL



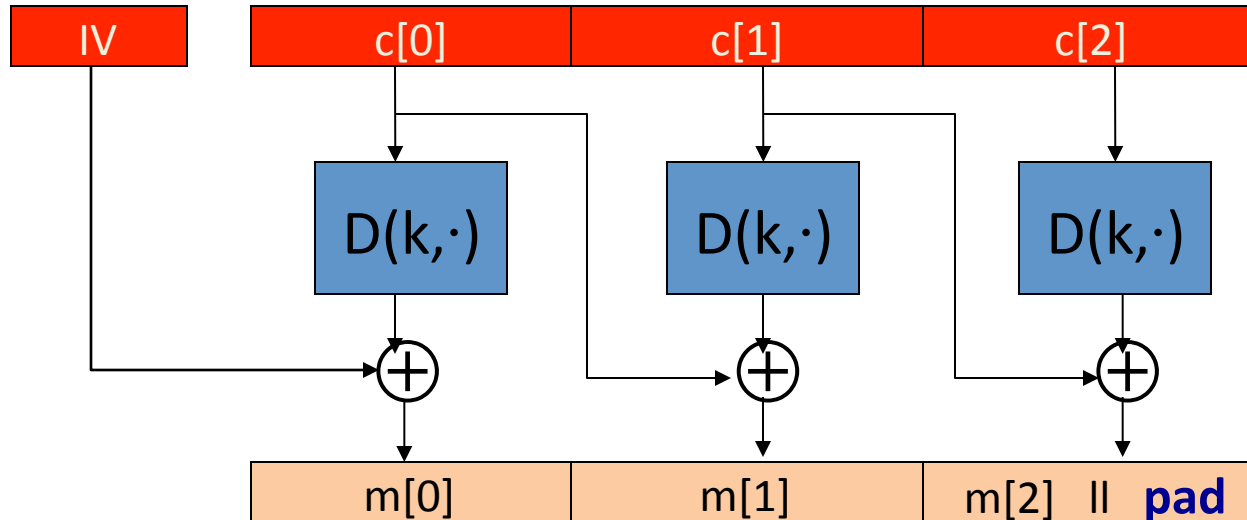
Credit: Brice Canvel

(fixed in OpenSSL 0.9.7a)

In older TLS 1.0: padding oracle due to different alert messages.

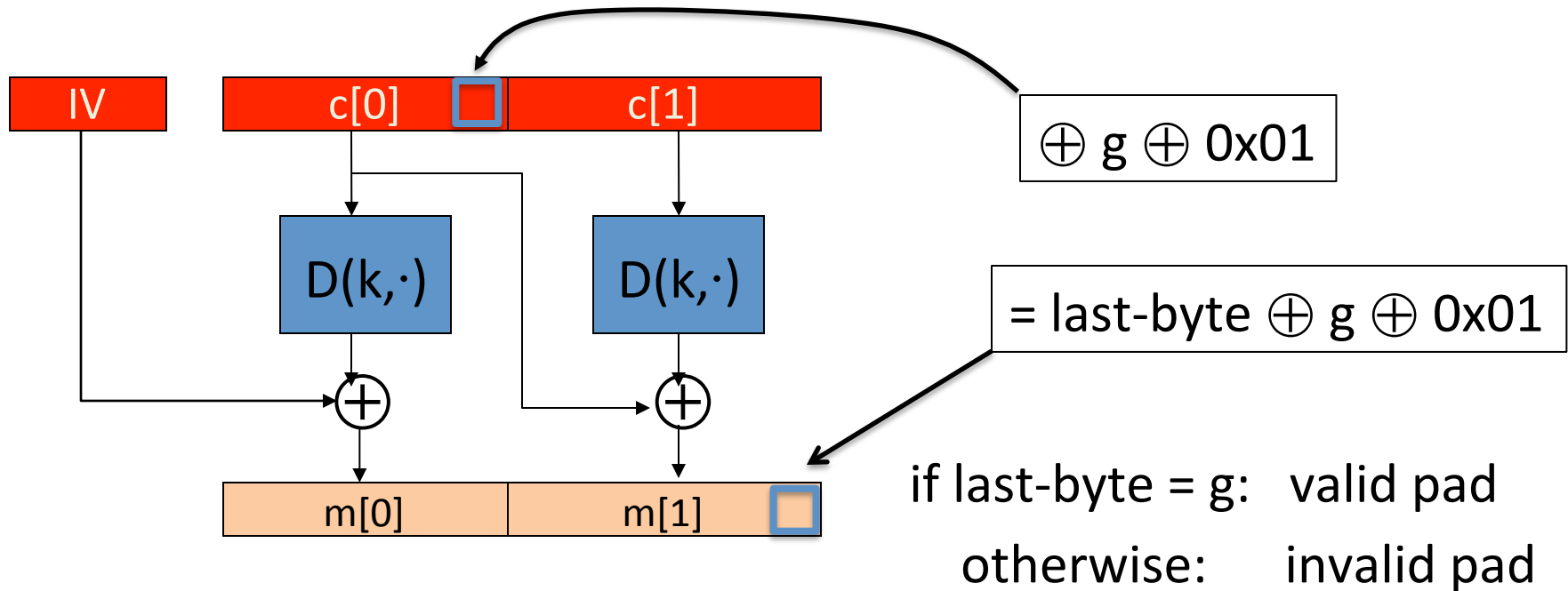
Using a padding oracle (CBC encryption)

Attacker has ciphertext $c = (c[0], c[1], c[2])$ and it wants $m[1]$



Using a padding oracle (CBC encryption)

step 1: let g be a guess for the last byte of $m[1]$



Using a padding oracle (CBC encryption)

Attack: submit $(IV, c'[0], c[1])$ to padding oracle

\Rightarrow attacker learns if last-byte = g

Repeat with $g = 0, 1, \dots, 255$ to learn last byte of $m[1]$

Then use a $(02, 02)$ pad to learn the next byte and so on ...

IMAP over TLS

Problem: TLS renegotiates key when an invalid record is received

Enter IMAP over TLS: (protocol for reading email)

- Every five minutes client sends login message to server:
LOGIN "username" "password"
- Exact same attack works, despite new keys
⇒ recovers password in a few hours.

Lesson


1. Encrypt-then-MAC would completely avoid this problem:

MAC is checked first and ciphertext discarded if invalid

2. MAC-then-CBC provides A.E., but padding oracle destroys it

Will this attack work if TLS used counter mode instead of CBC?

(i.e. use MAC-then-CTR)

- Yes, padding oracles affect all encryption schemes
- It depends on what block cipher is used
- No, counter mode need not use padding 
-

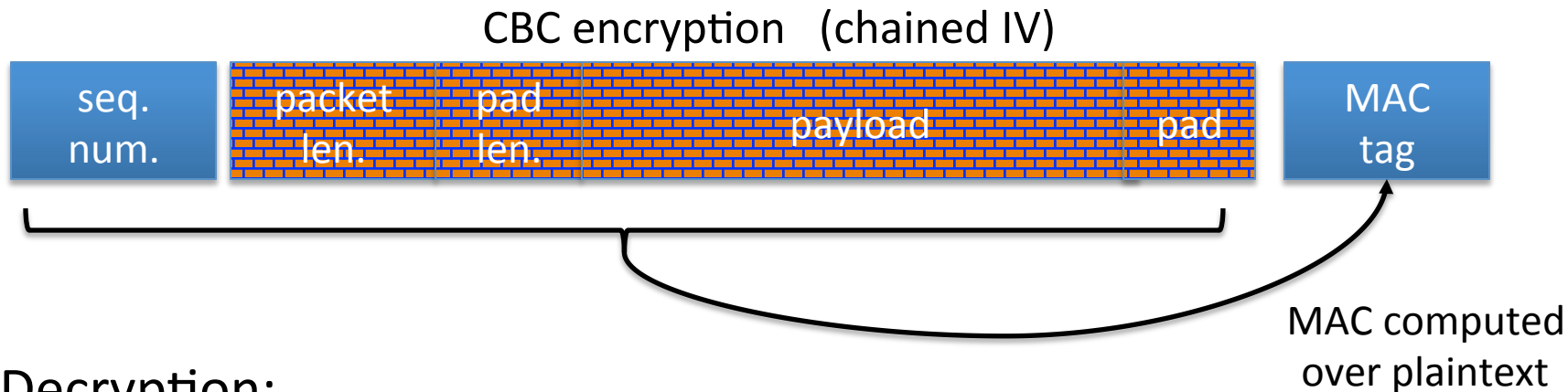
End of Segment



Authenticated Encryption

Attacking non-atomic
decryption

SSH Binary Packet Protocol

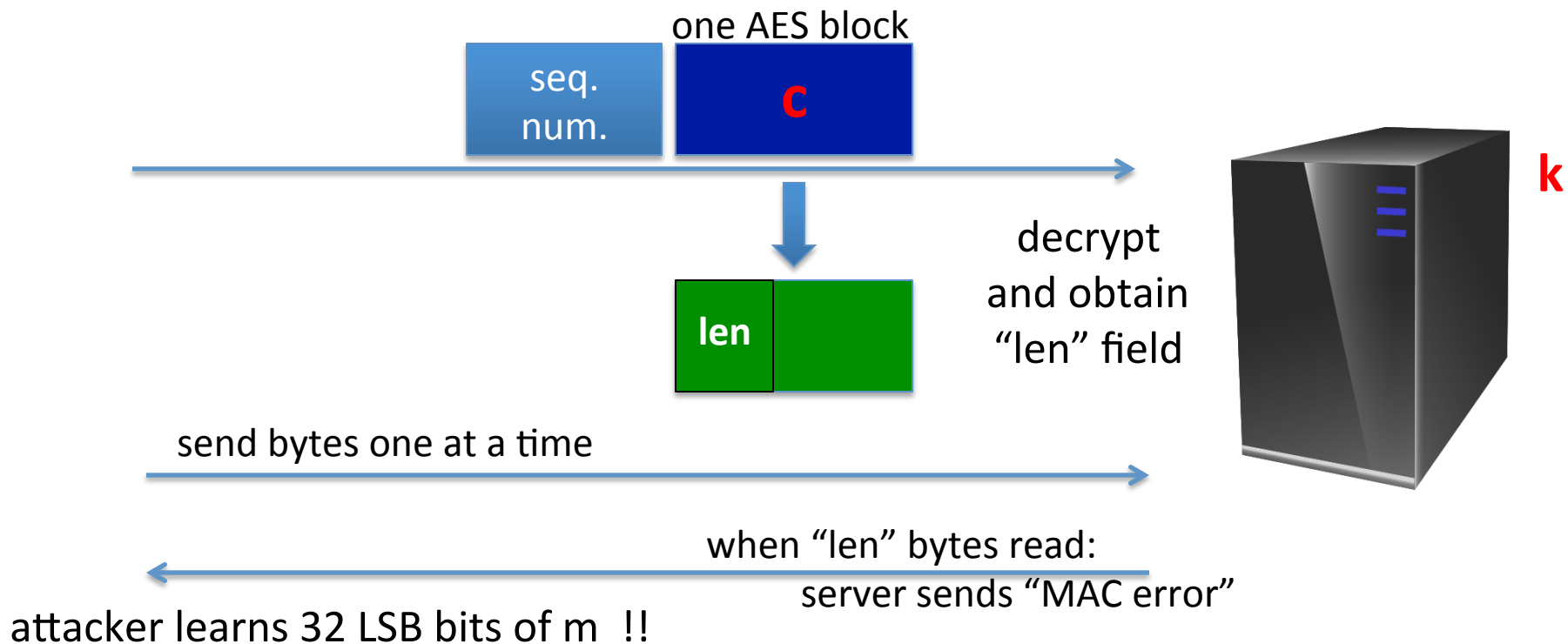


Decryption:

- step 1: decrypt packet length field only (!)
- step 2: read as many packets as length specifies
- step 3: decrypt remaining ciphertext blocks
- step 4: check MAC tag and send error response if invalid

An attack on the enc. length field (simplified)

Attacker has one ciphertext block $c = \text{AES}(k, m)$ and it wants m



Lesson

The problem: (1) non-atomic decrypt

(2) len field decrypted and used before it is authenticated

How would you redesign SSH to resist this attack?

- ⇒ ○ Send the length field unencrypted (but MAC-ed)
- Replace encrypt-and-MAC by encrypt-then-MAC
- ⇒ ○ Add a MAC of (seq-num, length) right after the len field
- Remove the length field and identify packet boundary by verifying the MAC after every received byte

Further reading

- The Order of Encryption and Authentication for Protecting Communications, H. Krawczyk, Crypto 2001.
- Authenticated-Encryption with Associated-Data, P. Rogaway, Proc. of CCS 2002.
- Password Interception in a SSL/TLS Channel, B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux, Crypto 2003.
- Plaintext Recovery Attacks Against SSH, M. Albrecht, K. Paterson and G. Watson, IEEE S&P 2009
- Problem areas for the IP security protocols, S. Bellare, Usenix Security 1996.

End of Segment