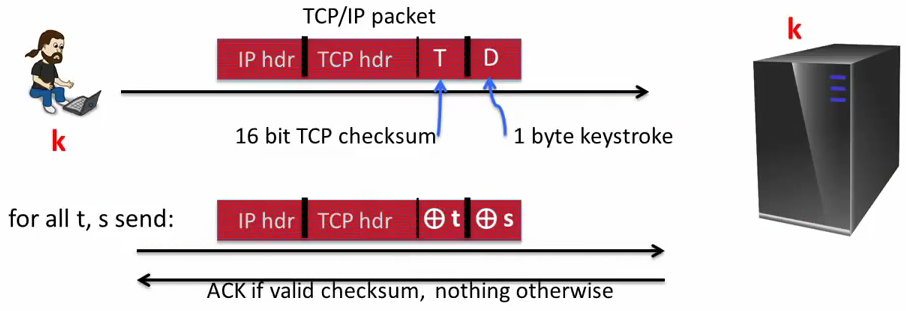
# Active attacks on CPA-secure encryption

This module: encryption secure against **tampering**: ensuring both **confidentiality** **and** **integrity**.

Example:

Remote terminal app: each keystroke encrypted with CTR mode

In CTR cipher if we xor the original message with a value t the resulting ciphered message is the original ciphered message xored with the same t.



If we do this with many t,s we can get many valid combinations (the server computes the checksum and respond with an ACK if valid)

{ checksum(hdr,D) = t xor checksum(hdr, D xor s) } ⇒ can find D

## In resume

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**

# Authenticated Encryption

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

As usual: E: K x M x N → C

but D: K x C x N → M U {bot}

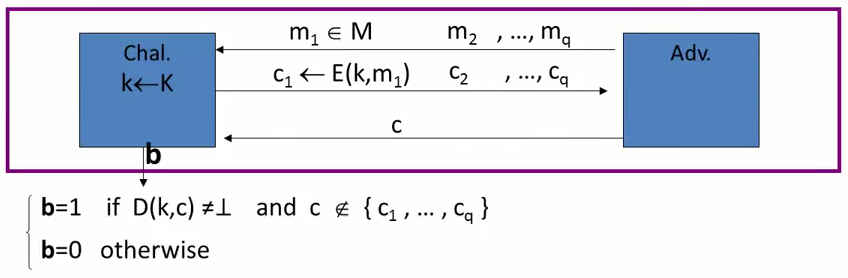
Security: the system must provide

- Semantically security under a CPA attack

- ciphertext integrity: attacker cannot create new ciphertexts that decrypts properly

## Ciphertext integrity

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



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

AdvCI[A,E] = Pr[Chal. outputs 1] is “negligible”

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

1.- semantically secure under CPA,

2.- has ciphertext integrity

Bad example: CBC with random IV does not provide AE

D(k,·) never outputs (bottom), hence adversary easily wins CI game

Implication 1: Authenticity = Attacker cannot fool Bob into thinking a message was sent from Alice. If the receiver gets a valid message then it must come from someone who knows the key.

Implication 2: Security against **chosen ciphertext attacks**

# Chosen ciphertext attacks

Adversary has ciphertext c that it wants to decrypt:

- Often, adversary can fool server into decrypting certain ciphertexts (not c)

- Often, adversary can learn partial information about plaintext

Adversary’s power: both CPA (Chosen Plain text Attack) and CCA (Chosen Cipher text Attack)

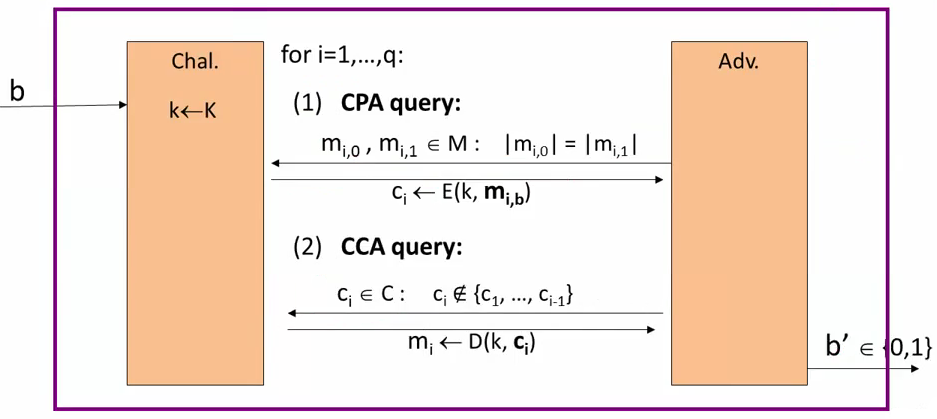
- Can obtain the encryption of arbitrary messages of his choice

- Can decrypt any ciphertext of his choice, other than challenge

Adversary’s goal: Break semantic security

## Chosen ciphertext security: Definition

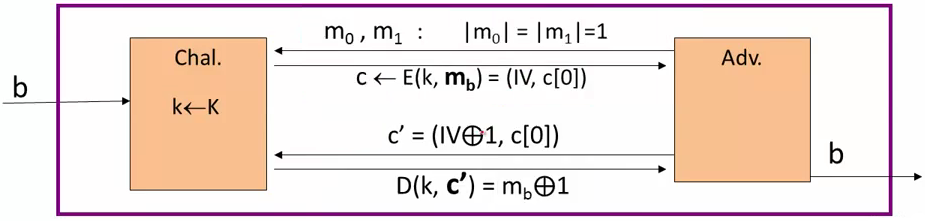
EBIG = (E,D) cipher defined over (K,M,C). For b=0,1 define EXP(b):



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

AdvCCA[A,EBIG] = | Pr[EXP(0)=1] - Pr[EXP(1)=1] | is “negligible”

Example: CBC with random IV is no CCA-secure



Note thar the cipher text c’ the attacker sends to the challenger is different from c → the IV part is changed

The Advantage is 1, the attacker always correcty guess the experiment.

## Authenticated encryption ⇒ CCA security

Theorem: Let (E,D) be a cipher that provides Authenticated Encryption

Then (E,D) is CCA (Chosen Ciphertext Attacks) secure

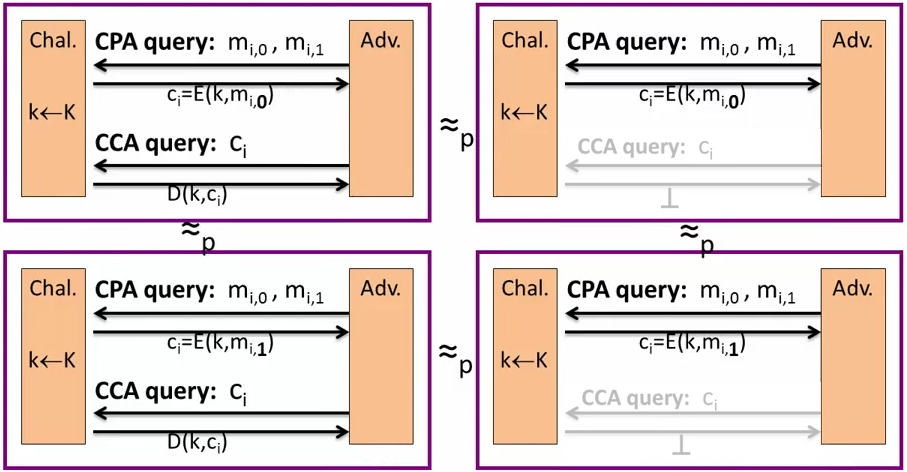
In particular, for any q-query efficient A there exist efficient B1, B2 such that:

AdvCCA[A,E] <= 2q AdvCI[B1,E] + AdvCPA[B2,E]

CI = Ciphertext integrity

## Proof AE ⇒ CCA security

Proof by pictures:



The CCA query responses can be replaced by a bottom response. That’s because the Adversary can’t generate a valid ciphertext, so the response will always be bottom.

If the adversary always receive the bottom result then that step provides no information to him, so we can remove these queries. Without them the game looks exactly the same as the CPA game.

## Cons: Authenticated Security

Limitations:

- does not prevent replay attacks

- does not account for side channels (timing)

# Construction from ciphers and MACs

Authenticated Encryption (AE) introduced in 2000

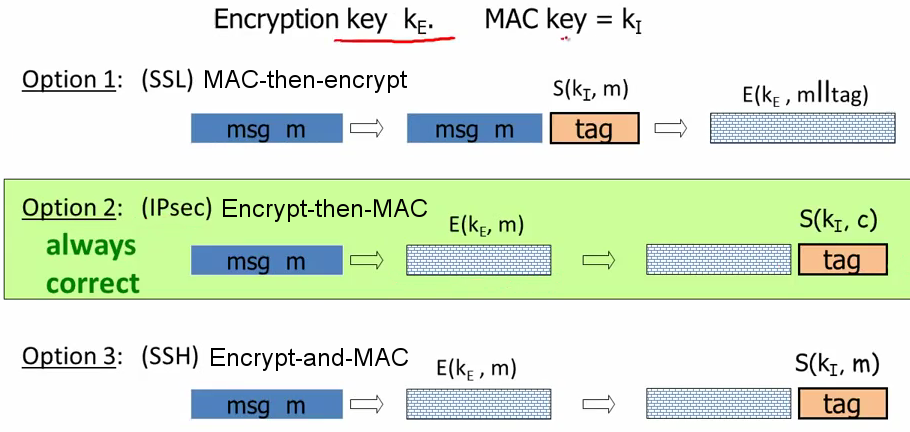
Before that date some crypto APIs provided the two requirements for AE but separately:

- Provide API for CPA-secure encryption (e.g. CBC with random 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 Authenticated Encryption (AE)

The most common mistake was incorrectly combine encryption and integrity mechanisms.



## AE 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 MAC-then-Encrypt provides A.E.

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

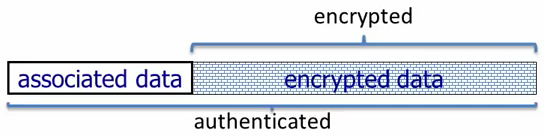
## Standards (at a high level)

**GCM**: CTR mode encryption then CW-MAC (accelerated via Intel’s instruccion)

**CCM**: CBC-MAC then CTR mode encryption (802.11i) → AES in both mechanisms.

**EAX**: CTR mode encryption then CMAC

All support AEAD: (authenticated encryption with associated data). All are nonce-based



Only a part of the message is intended to be encrypted but all of the message is intended to be authenticated.

## 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)

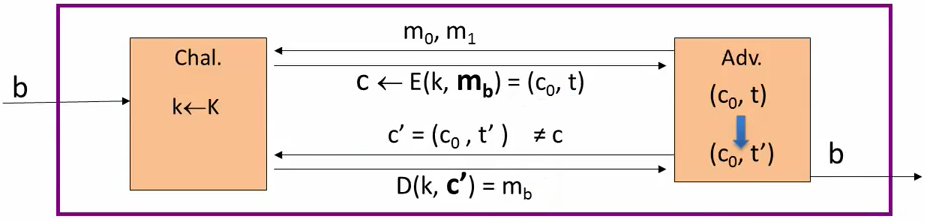
## MAC Security -- an explanation

Recall: MAC security implies (m,t) =/=> (m, t’) t != t’

Why is this?

Suppose not: (m, t) → (m,t’)

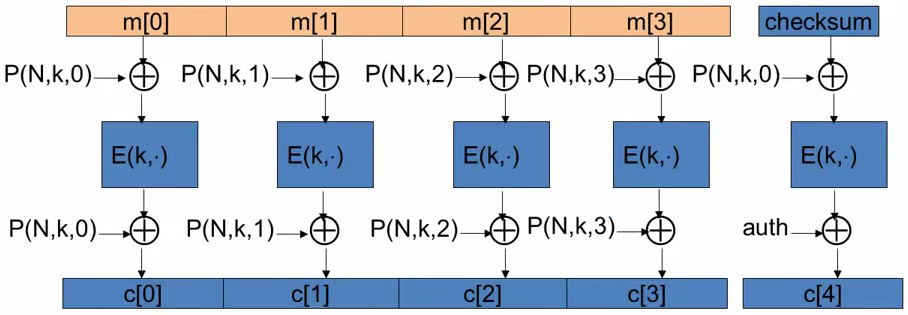
Then Encrypt then MAC would not have Ciphertext Integrity!!



The Adversary can always determine the experiment, so the Adv = 1

## OCB: a direct construction from a PRP

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



## Performance

|  |  |  |
| --- | --- | --- |
| **Cipher** | **Code size** | **Speed (MB/s)** |
| AES/GCM | large\* | 108 |
| AES/CCM | smaller | 61 |
| AES/EAX | smaller | 61 |
| AES/OCB |  | 129 |

Never use:

AES/CTR

AES/CBC

AES/CMAC

HMAC/SHA1

The recommended mode is the first: AES/GCM

# Case study: TLS



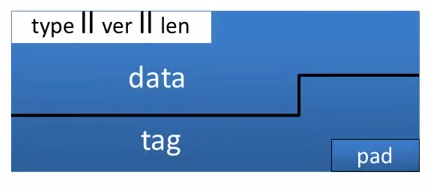
Unidirectional keys: kb→s and ks→b

TLS record < 16KB

Stateful encryption:

* Each side maintains two 64-bit counters: ctrb→s , ctrs→b
* Initialize to 0 when session started. ctr++ for every record
* Purpose: replay defense

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



kb→s=(kmac, kenc)

Browser side **enc (kb→s, data, ctrb→s)**:

step 1: tag ← S( kmac, [++ctrb→s] || header || data] )

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

step 3: CBC encrypt with kenc and new random IV

step 4: prepend header

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

Server side **dec (kb→s, record, ctrb→s)**:

step 1: CBC decrypt record using kenc

step 2: check pad format: send **bad\_record\_mac** if invalid

step 3: check tag on [ ++ctrb→s || header || data ]: send **bad\_record\_mac** if invalid

Provides authenticated encryption. No other information is leaked during decryption.

The bad\_record\_mac is the bottom symbol. The fact that bad\_record\_mac doesn’t reveal the step at which the record was rejected provides AE. If TLS differentiated each reject point then there would be some attacks that an adversary could try.

## Bugs in older version (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 attack: BEAST attack)

**Padding oracle**: during encryption

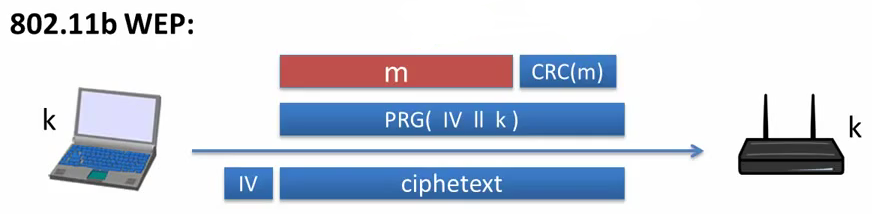
if pad is invalid send decryption failed alert

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

## 802.11b WEP: how not to do it



The IV is repeated and the entry to the PRG is very similar always: [ IV || k ]

Problems: two time pad and related PRG seeds.

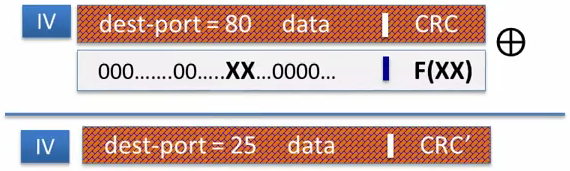
### Active attacks

Fact: CRC is linear, i.e.

F is a well-known public function.

WEP ciphertext: 

The attacker will modify the message and set the destination port to another value. It will be done xor-ing the message and the CRC



Upon decryption: CRC is valid, but ciphertext is changed

Remember never use CRC as an integrity mechanism.

# CBC padding attacks

## Recap

Authenticated Encryption: CPA security + ciphertext integrity

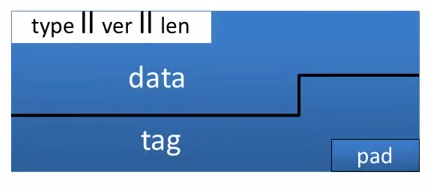
* Confidentiality in presence of active adversary
* Prevents chosen-ciphertext attacks

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: dec(kb→s, record, ctrb→s)

step 1: CBC decrypt record using kenc

step 2: check pad format: abort if invalid

step 3: check tag on [ ++ctrb→s || header || data ]: abort if invalid

Two types of error:

* padding error
* MAC error

It’s very important the attacker doesn’t know which of this two types of errors occurs.

## Padding oracle

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

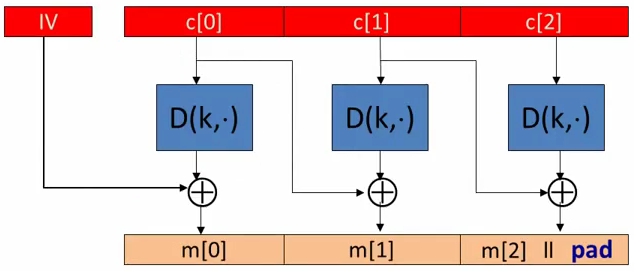
⇒ Padding oracle:

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

Padding oracle via timing OpenSSL

Using a padding oracle (CBC encryption)

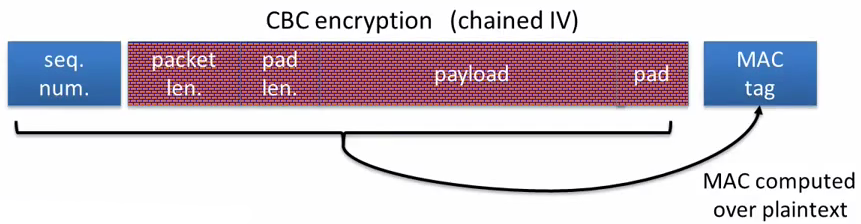
Attacker has ciphertext c = c(c[0], c[1], c[2]) and it wants m[1]



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

# Attacking non-atomic decryption

## SSH Binary Packet Protocol



Decryption:

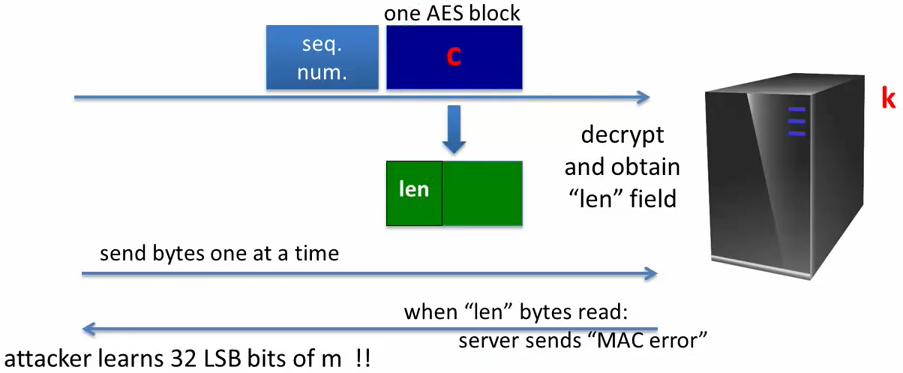
* step 1: decrypt packet length field only (!)
* step 2: read as many packets as length specifies
* step 3: decrypt remaining ciphertext block
* step 4: check MAC tag and send error response if invalid

MAC has no confidentiality

At step 1 it can be seen that the procotol uses the length field before verifying the MAC is valid.

## An attack on the encrypted length field (simplified)

Attacker has one cipfertext block c=AES(k,m) and it wants m



The problem here is:

1.- non-atomic decrypt

2.- length field decrypted and used it before it is authenticated

Possible solutions:

* Send the length field unencrypted (but MAC-ed) → for example as TLS does
* Add a MAC of (seq-num, length) right after the len field

Wrong solutions:

* Replace encrypt-and-MAC by encrypt-then-MAC → the problem will still be there, the length will be used before the ciphertext was verified
* 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 is 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. Bellovin, Usenix Security 1996