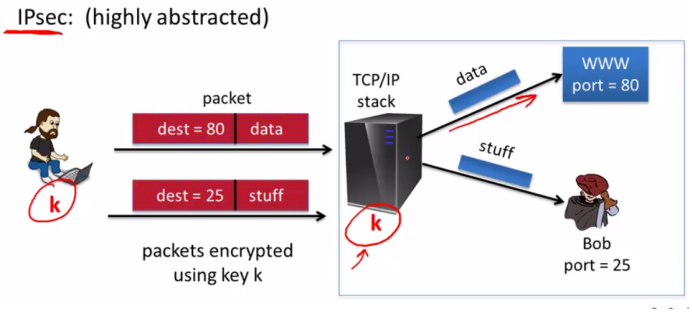
- **Authenticated Encryption 1: why is it so important?**
- **Active Attacks on CPA-Secure Encryption**
- Recap: the story so far

Confidentiality: semantic security against a CPA attack
- Encryption secure against **eavesdropping only**

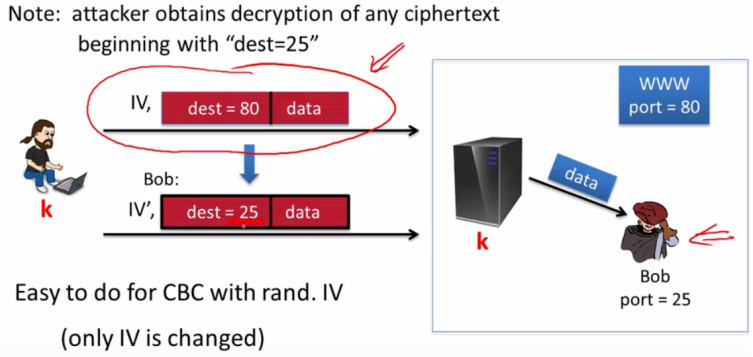
Integrity:
- Existential unforgeability under a chosen message attack
- CBC-MAC, HMAC, PMAC, CW-MAC

This module: encryption secure against **tampering**
- Ensuring both confidentiality and integrity

- Sample tampering attacks
  - TCP/IP: (highly abstracted)
    - send the data to the destination port. There are two people listening to the TCP/IP stack.
  - IPsec: (highly abstacted)
    - encrypts the packets between the sender and the recipient
    - packets encrypted using key k
    - TCP/IP stack decrypts the packet and sends it



- Reading someone else's data



- Example

Encryption is done with CBC with a random IV.

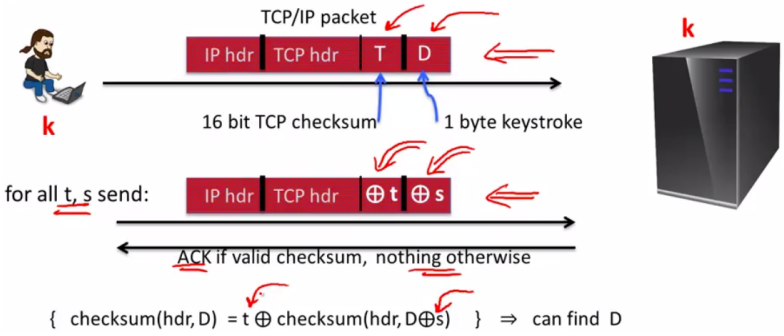
What should IV' be? $m[0] = D(k, c[0]) \oplus IV$ = "dest=80..."

dest=25

- ○ $IV' = IV \oplus (\ldots 25 \ldots)$
- ○ $IV' = IV \oplus (\ldots 80 \ldots)$
- ○ $IV' = IV \oplus (\ldots 80 \ldots) \oplus (\ldots 25 \ldots)$
- ○ It can't be done

- An attack using only network access

Remote terminal app.: each keystroke encrypted with CTR mode



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

- The lesson
  - active attacks attack modifying information in route
  - 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)
- **Definitions**
- **Authenticated Encryption**
- Goals

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

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

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

$\perp \notin M$

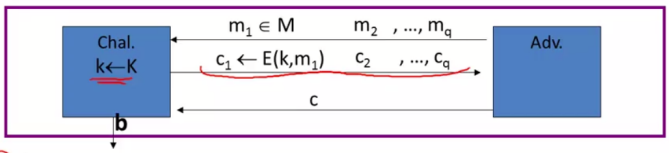
ciphertext is rejected

Security: the system must provide

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

- Ciphertext integrity

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



b=1 if $D(k,c) \neq \perp$ and $c \notin \{c_1, \ldots, c_q\}$
b=0 otherwise

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

$Adv_{CI}[A,E]$ = Pr[Chal. outputs 1] is "negligible."

Dan

- Authenticated encryption

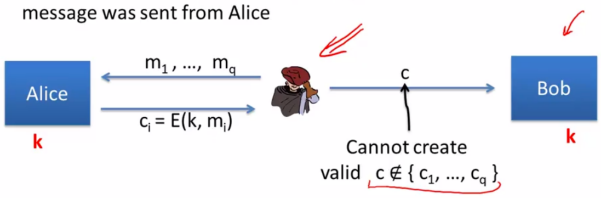
Def: cipher (E,D) provides **authenticated encryption (AE)** if it is
(1) semantically secure under CPA, and
(2) has ciphertext integrity

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

- D(k,·) never outputs ⊥, hence adv. easily wins CI game

- Implication 1: authenticity

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



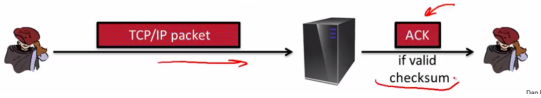
⇒ if $D(k,c) \neq \perp$ Bob knows message is from someone who knows k

- (but message could be a replay)
- Implication 2
  - Authenticated encryption => Security against chosen cipher text attacks
- **Chosen Ciphertext attacks**
- Example chosen cipher text attacks

Adversary has ciphertext c that it wants to decrypt

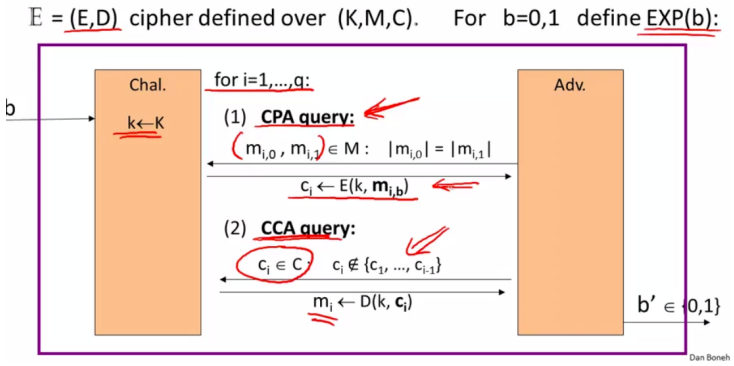
- Often, adv. can fool server into decrypting **certain** ciphertexts (not c)

- Often, adversary can learn partial information about plaintext



Dan Boneh

- Chosen cipher text security
  - Adversary's power: both CPA and CCA
    - can obtain the encryption of arbitrary messages of his choice
    - can decrypt any cipher text of his choice, other than challenge (conservative modeling of real life)
  - Adversary's goal: Break sematic security
- Chosen cipher text security: definition

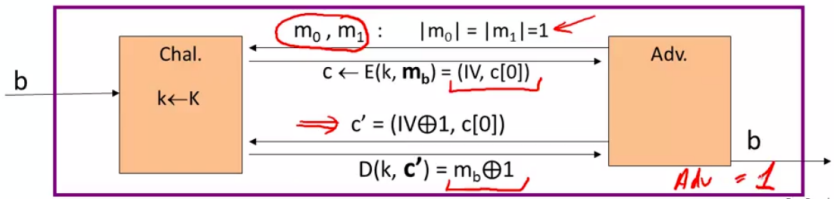


- Chosen cipher text security: definition

E is CCA secure if for all "efficient" A:

$Adv_{CCA}[A,E] = |Pr[EXP(0)=1] - Pr[EXP(1)=1]|$ is "negligible."

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



- Authenticated enc. => CCA security

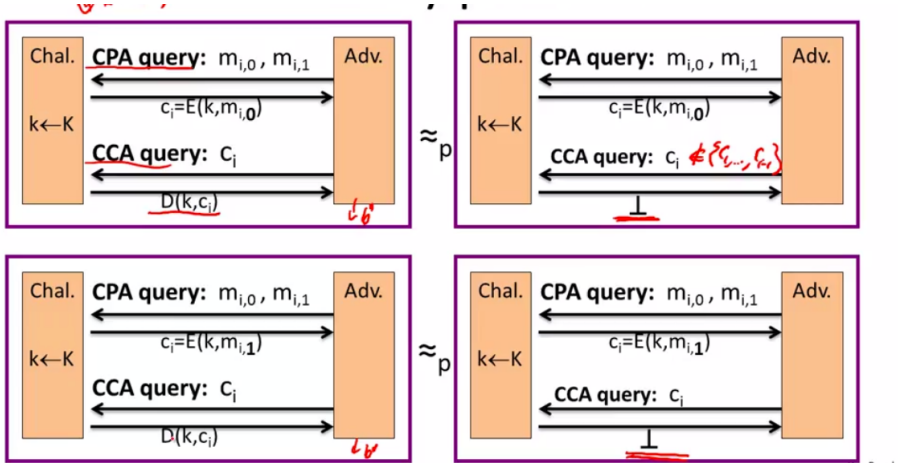
Thm: Let (E,D) be a cipher that provides AE.
Then (E,D) is CCA secure !

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

$Adv_{CCA}[A,E] \le 2q \cdot Adv_{CI}[B_1,E] + Adv_{CPA}[B_2,E]$

neg neg neg

- Proof by pictures
  - CCA secure



  - remove queries since they don't provide any information since they always respond bottom.
- So what?
  - Authenticated encryption:
    - ensures confidentiality against an active adversary that can decrypt some cipher texts
  - Limitations:
    - does not prevent replay attacks
    - does not account for side channels (timing)
- **Authenticated Encryption 2: standard constructions**
- **Constructions from Ciphers and MACs**
- … but first, some history

Authenticated Encryption (AE): introduced in 2000

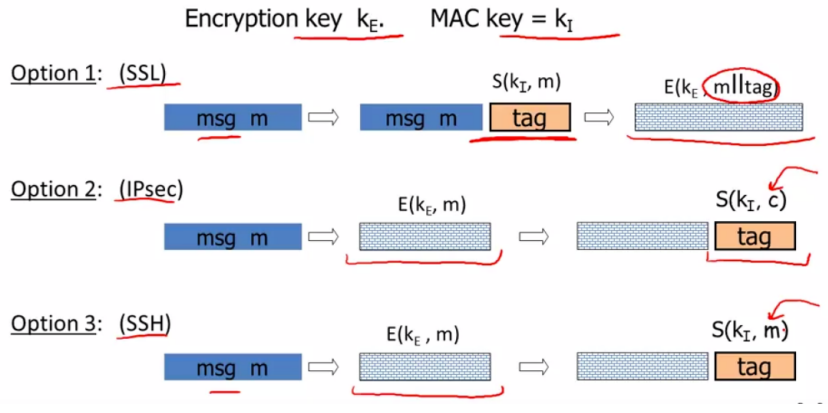
Crypto APIs before then: (e.g. MS-CAPI)

- 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)



- MACs are not designed for confidentiality only designed for integrity
- SSH (encrypt and mac)
  - tag concatenated after the message is encrypted
  - breaks cpa security
  - bits of the message are leaked in the cipher text
  - output of the mac signing algorithm might leak bits of the message
- SSL (mac then encrypt)
  - Bad interaction between the encryption scheme and mac algorithm
- **IPsec** (encrypt then mac)
  - always correct
  - message is encrypted and contents hidden in cipher text
  - compute tag of cipher text locks tag of cipher text and no one can produce a tag of the cipher text that will look valid

- A.E. Theorems

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

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

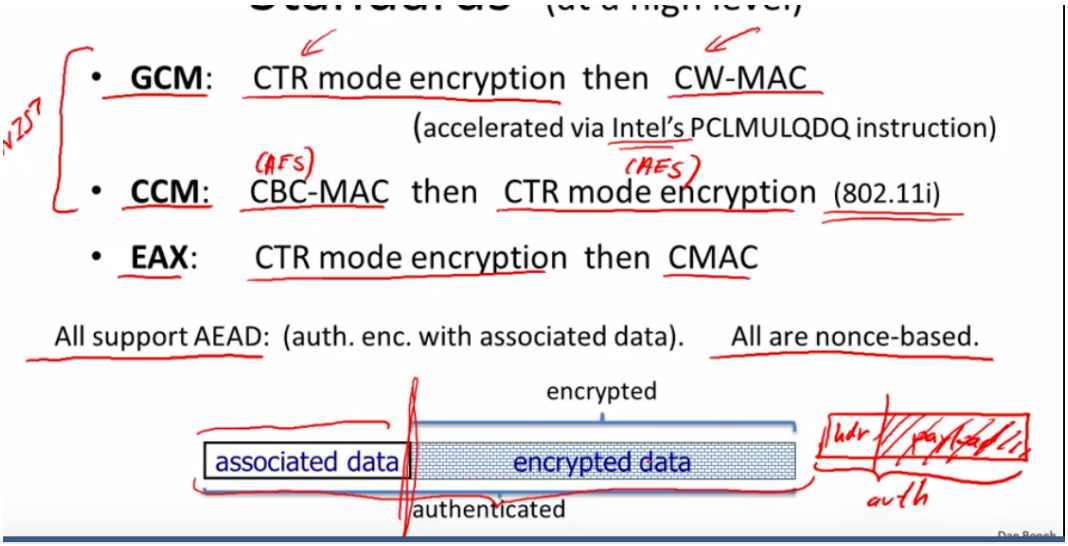
2. **MAC-then-encrypt**: may be insecure against CCA attacks

   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

Dan Bon

- Standards (at a high level)



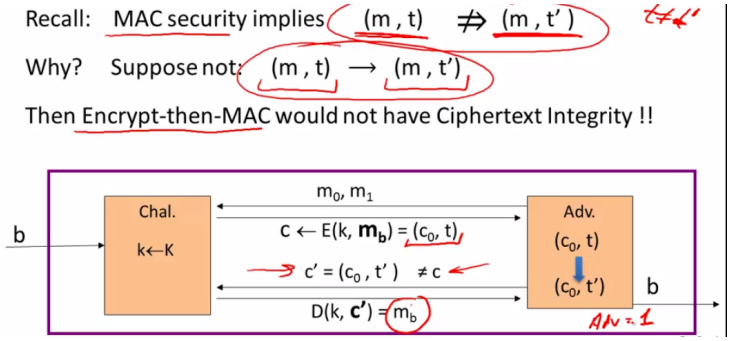
- An example API (OpenSSL)

AEAD

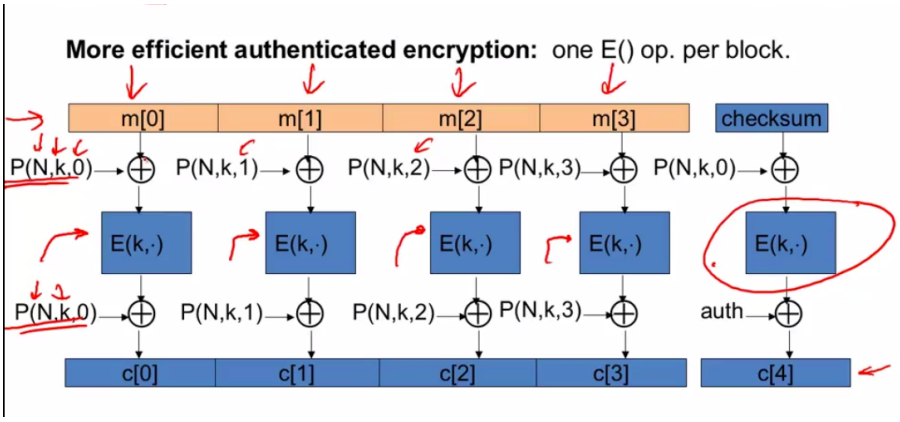
```
int AES_GCM_Init(AES_GCM_CTX *ain,
        unsigned char *nonce,  unsigned long noncelen,
        unsigned char *key,  unsigned int klen )

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



- OCB: a direct construction from a PRP



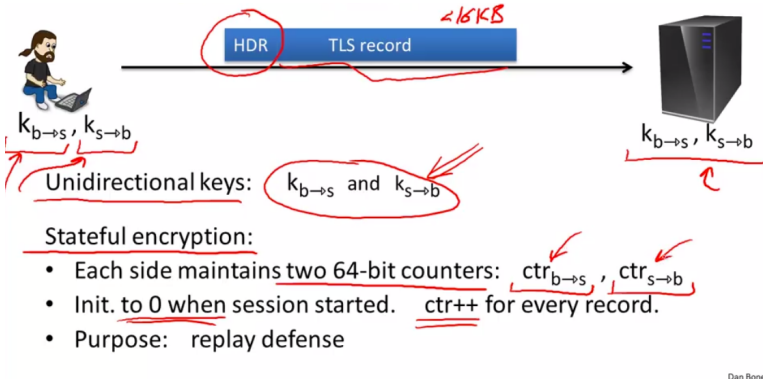
- Performance

AMD Opteron, 2.2 GHz (Linux)

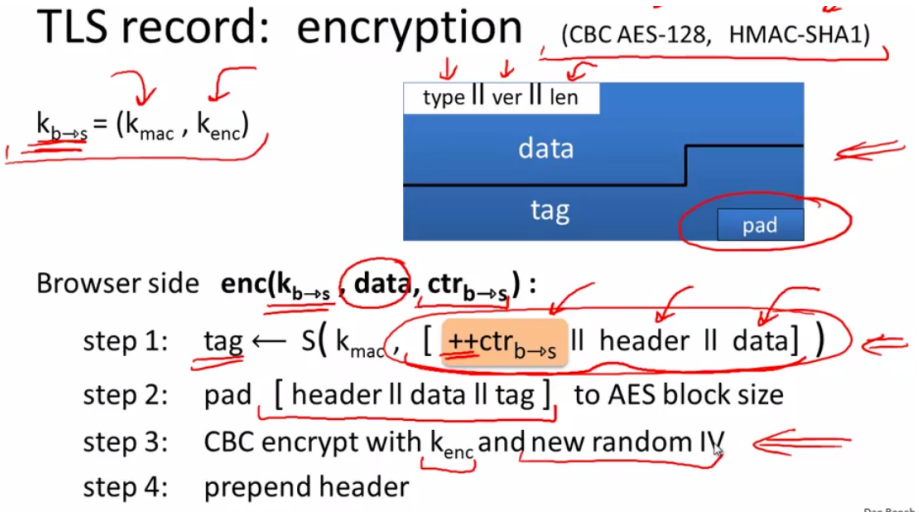
	Cipher	code size	Speed (MB/sec)		
auth. enc.	AES/GCM	large**	108	~~AES/CTR~~	139
	AES/CCM	smaller	61	~~AES/CBC~~	109
	AES/EAX	smaller	61		
				AES/CMAC	109
	AES/OCB		129*	HMAC/SHA1	147

* extrapolated from Ted Krovitz's results ** non-Intel machines

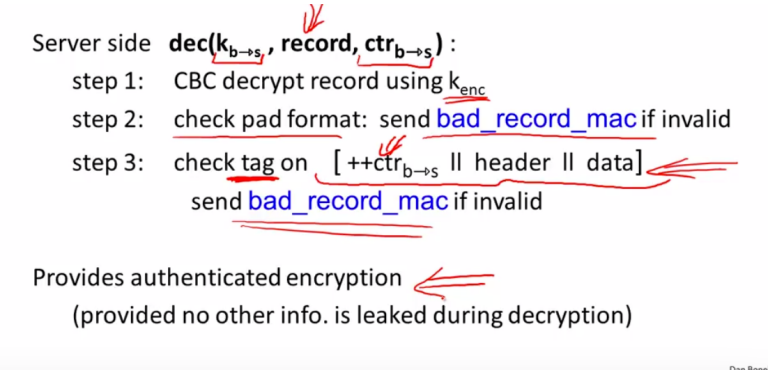
- **Authenticated Encryption 3: pitfalls**
- **Case study: TLS 1.2**
- The TLS Record Protocol



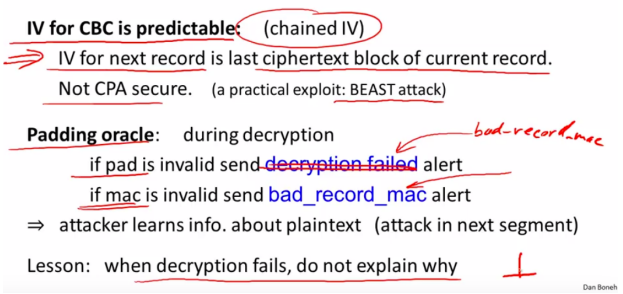
- TLS record: encryption



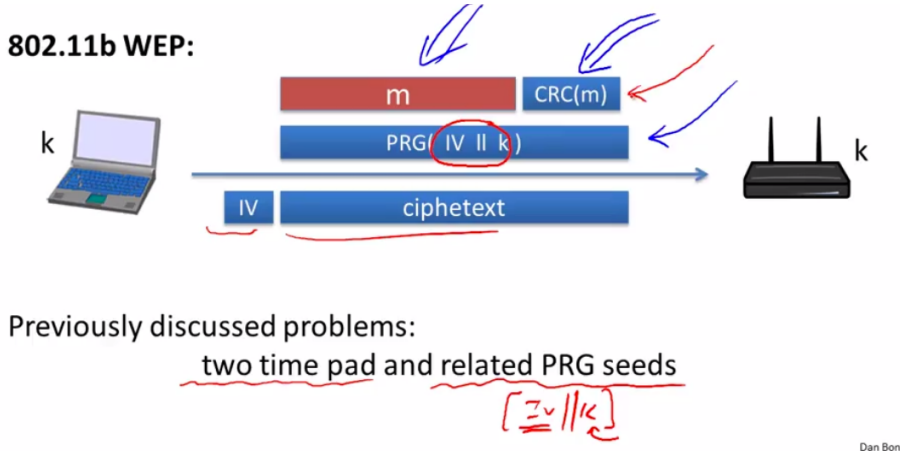
- TLS record: decryption



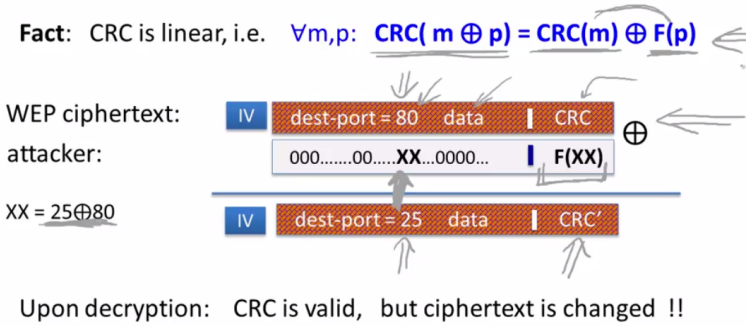
- Bugs in older versions



- 802.11b WEP:



- Active attacks



  - CRC check sum provides no integrity at all against active attacks

- **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($k_{b\to s}$, record, $ctr_{b\to s}$)** :

step 1: CBC decrypt record using k_{enc}

step 2: check pad format: abort if invalid

step 3: check tag on [++$ctr_{b\to s}$ || header || data]
abort if invalid

Two types of error:

- **padding error**
- **MAC error**

Dan Boneh

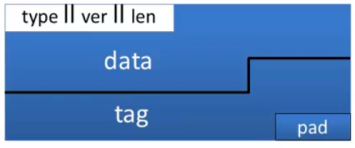
- important that the advisory is no told that a pad or mac error occurred

- Padding oracle

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

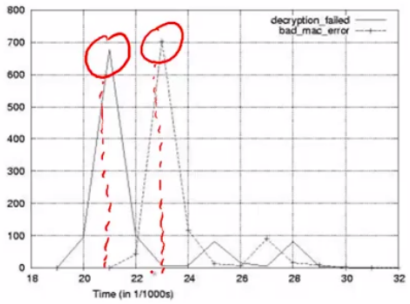
⇒ **Padding oracle**:
attacker submits ciphertext and learns if
last bytes of plaintext are a valid pad

Nice example of a
chosen ciphertext attack



- advisory is allowed to learn something about the resulting plaintext

- Padding oracle via timing OpenSSL



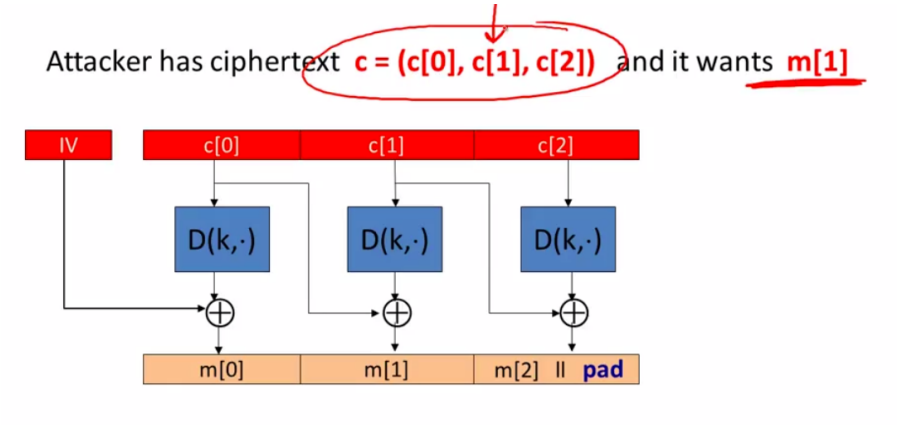
Credit: Brice Canvel

(fixed in OpenSSL 0.9.7a)

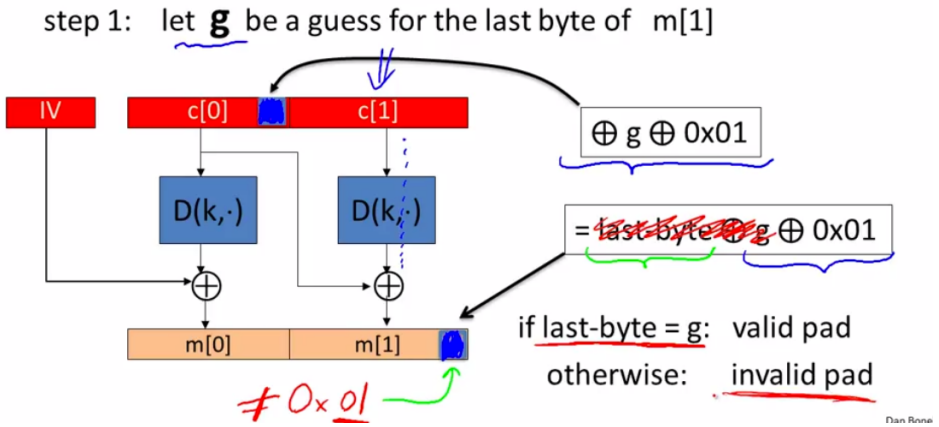
In older TLS 1.0: padding oracle due to different alert messages.

- if the time was short knows that the pad was invalid
- if the time was long knows that the pad was valid but the mac was invalid
- still has a padding oracle to tell if the pad was valid or invalid

- Using a padding oracle (CBC encryption)

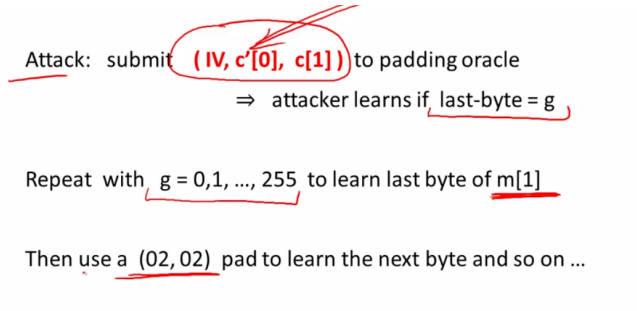


- Using a padding oracle (CBC encryption)



- if our guess for the last pad is correct we learn that our guess is correct
- if our guess for the last byte is not correct we learn that our guess is not correct

- Using a padding oracle (CBC encryption)



- 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.

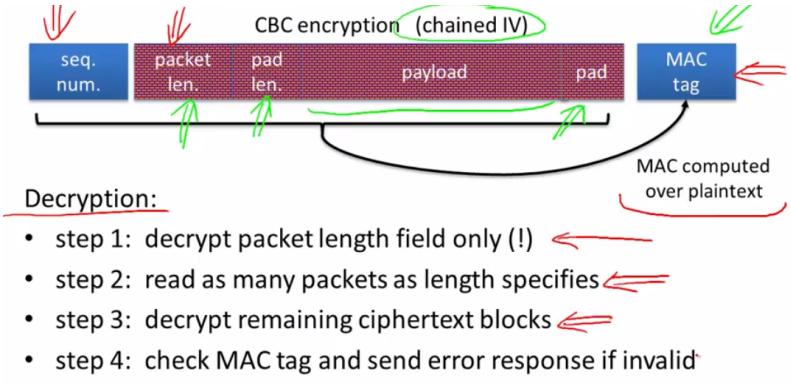
- Lesson
  - 1. Encrypt-then-MAC would completely avoid this problem:
    - MAC is checked first and cipher text discarded if invalid
  - 2. MAC-then-CBC provides A.E., but padding oracle destroys it
- Example

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

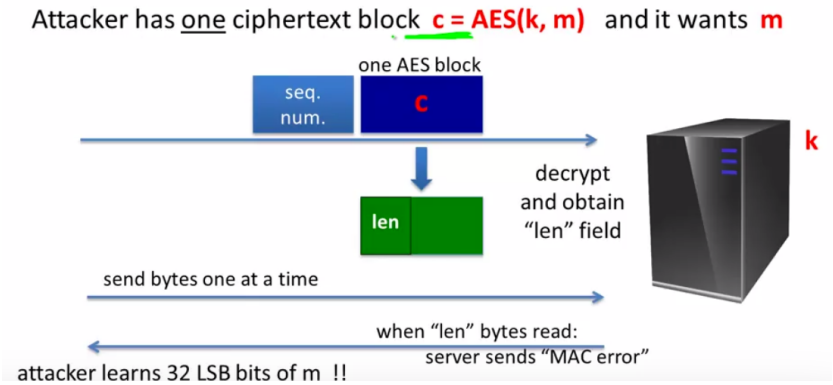
(i.e. use MAC-then-CTR)

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

- **Attacking Non-Atomic Decryption**
- SSH Binary Packet Protocol



- An attack on the ecn. length field



  - send packet to the server
  - server decrypts the first few bytes as the length field of the packet
  - server excepts this many bytes before it checks that this is valid
  - attack feeds server one byte at a time
  - server reads as many bytes as the length field specifies
  - the attack was counting the number of bytes
  - learns the length
- Lesson

The problem: (1) non-atomic decrypt
(2) len field decrypted and used it 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

- **Key Derivation**
- Deriving many keys from one

Typical scenario. a single source key (SK) is sampled from:

- Hardware random number generator
- A key exchange protocol (discussed later)

Need many keys to secure session:

- unidirectional keys; multiple keys for nonce-based CBC.

Goal: generate many keys from this one source key

- When source key is uniform

F: a PRF with key space K and outputs in $\{0,1\}^n$

Suppose source key SK is uniform in K

- Define Key Derivation Function (KDF) as:

$$\mathbf{KDF}(SK, CTX, L) := F(SK, (\mathbf{CTX} \,\|\, \mathbf{0})) \,\|\, F(SK, (\mathbf{CTX} \,\|\, \mathbf{1})) \,\|\, \cdots \,\|\, F(SK, (\mathbf{CTX} \,\|\, \mathbf{L}))$$

  - CTX: a string that uniquely identifies the application

- Example

$$\mathbf{KDF}(SK, CTX, L) := F(SK, (\mathbf{CTX} \,\|\, \mathbf{0})) \,\|\, F(SK, (\mathbf{CTX} \,\|\, \mathbf{1})) \,\|\, \cdots \,\|\, F(SK, (\mathbf{CTX} \,\|\, \mathbf{L}))$$

What is the purpose of CTX?

- ○ Even if two apps sample same SK they get indep. keys
- ○ It's good practice to label strings with the app. name
- ○ It serves no purpose
- ○

- What if source key is not uniform?

Recall: PRFs are pseudo random only when key is uniform in K

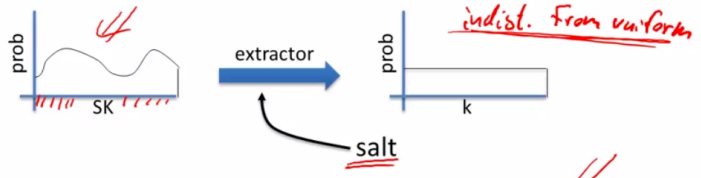
- SK not uniform ⇒ PRF output may not look random

Source key often not uniformly random:

- Key exchange protocol: key uniform in some subset of K
- Hardware RNG: may produce biased output

- Extract-then-Expand paradigm

Step 1: **extract** pseudo-random key k from source key SK



salt: a fixed non-secret string chosen at random

step 2: **expand** k by using it as a PRF key as before

- HKDF: a KDF from HMAC

Implements the extract-then-expand paradigm:

- extract: use $k \leftarrow HMAC(salt, SK)$
- Then expand using HMAC as a PRF with key **k**

- Password-Based KDF (PBKDF)

Deriving keys from passwords:

- Do not use HKDF: passwords have insufficient entropy
- Derived keys will be vulnerable to dictionary attacks

(more on this later)

PBKDF defenses: **salt** and a **slow hash function**

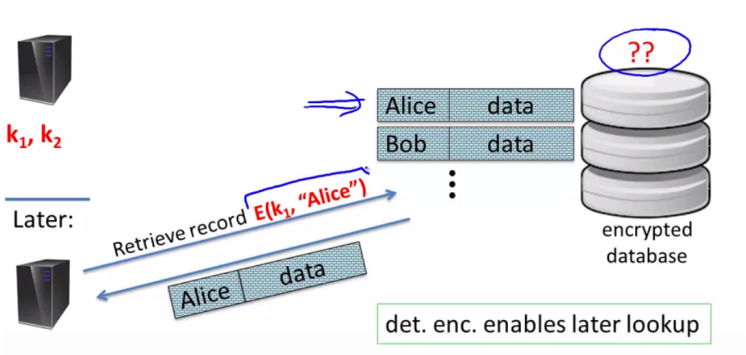
Standard approach: **PKCS#5** (PBKDF1)

$H^{(c)}$**(pwd || salt)**: iterate hash function c times

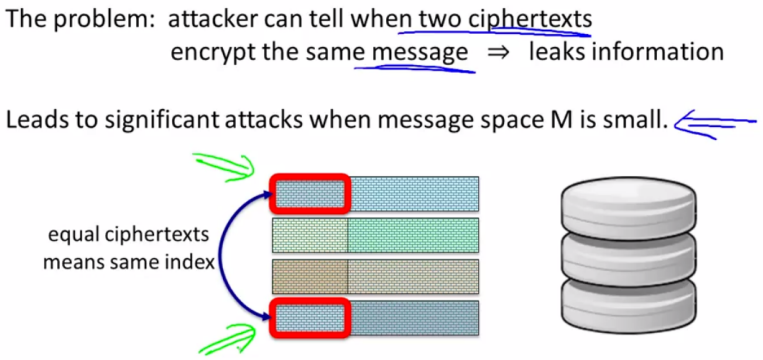
- slow hash function is taking a hash function and iterate the function many many times

- **Odds and Ends 2: searching on encrypted data**
- **Deterministic Encryption**

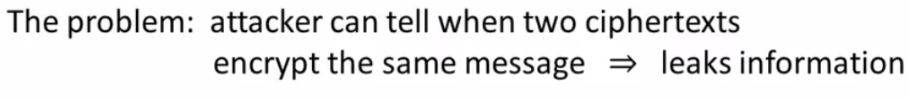
- The need for deterministic Encryption (no nonce)

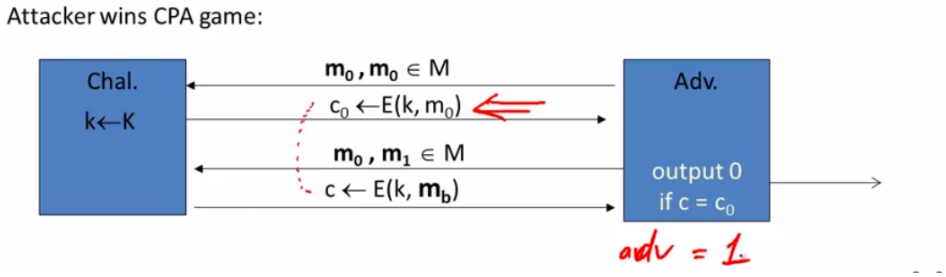


  - encrypts the the index with k1 and encrypts the data with k2 the database has no knowledge of the data that is stored in it
- Problem: deterministic encryption cannot be CPA secure



- Problem: deterministic encryption cannot be CPA secure

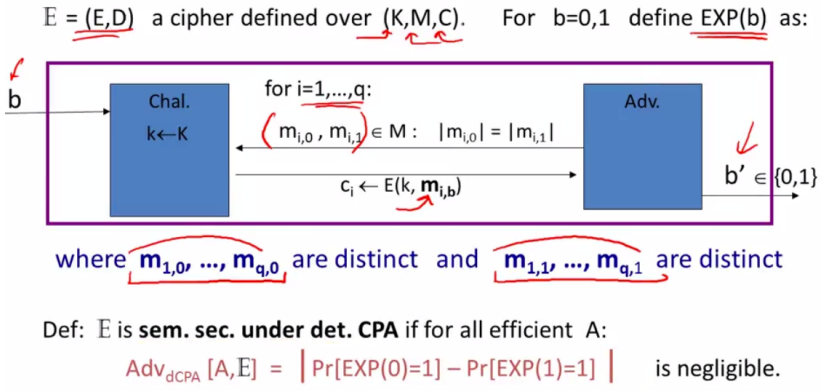




- A solution: the case of unique messages
  - Suppose encryptor never encrypts same message twice: the pair (k,m) never repeats

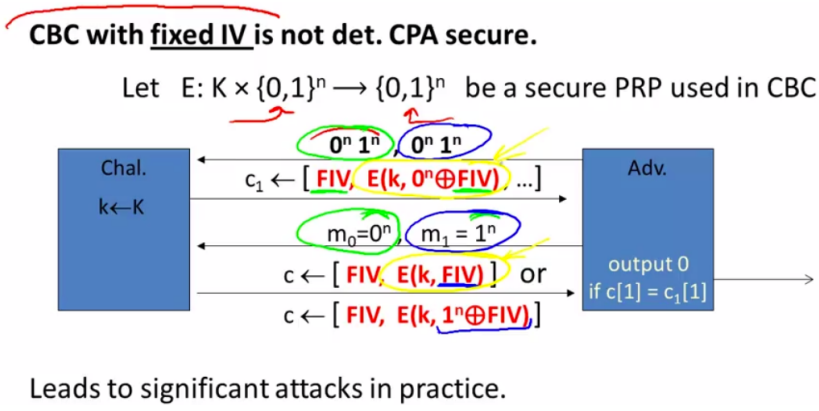
- This happens when encryptor:
  - chooses messages at random from large message space (e.g. keys)
  - message structure ensures uniqueness (e.g. unique user ID)

- Deterministic CPA security

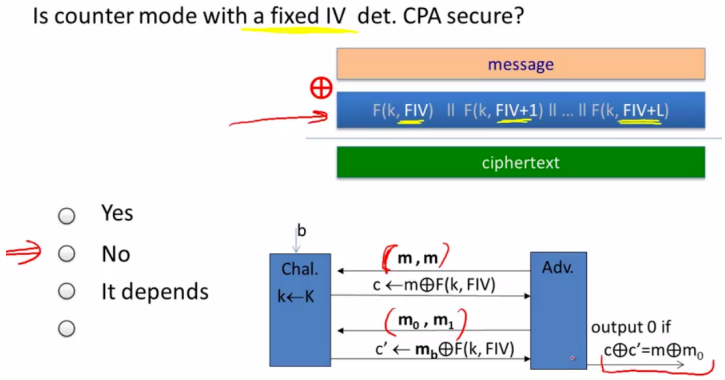


- If the two probabilities are close then the advantage is negligible

- A Common mistake



- Example
  - two time pad attack



- **Deterministic Encryption: SIV and Wide PRP**
- Deterministic Encryption
  - Needed for maintaining an encrypted database index
    - lookup records by encrypted index
  - Deterministic CPA security
    - Security if never encrypt same message twice using same key:
      - the pair (key,msg) is unique
  - Formally: we defined deterministic CPA security game
- Construction 1: Synthetic IV (SIV)
  - CPA that doesn't use nonces has to be randomized

Let (E, D) be a CPA-secure encryption. $E(k, m ; r) \longrightarrow c$

Let $F: K \times M \longrightarrow R$ be a secure PRF

Define: $E_{det}((k_1,k_2), m) =$

$r \leftarrow F(k_1, m)$

$c \leftarrow E(k_2, m ; r)$

output c

Thm: E_{det} is sem. sec. under det. CPA .

Proof sketch: distinct msgs. ⇒ all r's are indist. from random

Well suited for messages longer than one AES block (16 bytes)

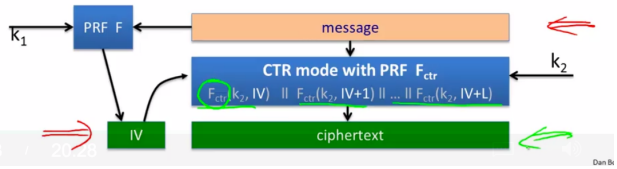
- Ensuring ciphertext integrity
  - ciphertext integrity - attack gets to ask for encryptions of messages of his choice and shouldn't be able to produce another ciphertext that decrypts to a valid message

Goal: det. CPA security and ciphertext integrity

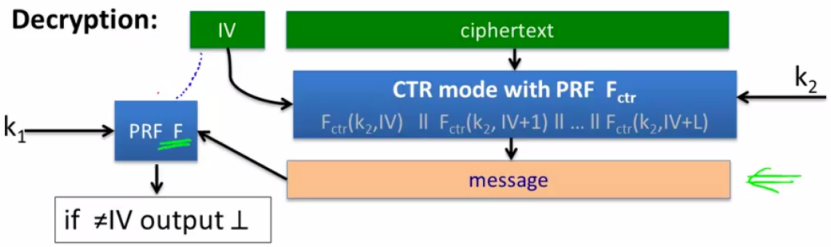
⇒ **DAE: deterministic authenticated encryption**

Consider a SIV special case: SIV-CTR

SIV where cipher is counter mode with rand. IV



- Det. Auth. Enc. (DAE) for free



Thm: if F is a secure PRF and CTR from F_{ctr} is CPA-secure

then SIV-CTR from F, F_{ctr} provides DAE

- Construction 2: just use a PRP

Let (E, D) be a secure PRP. $E: K \times X \longrightarrow X$

Thm: (E,D) is sem. sec. under det. CPA .

Proof sketch: let $f: X \longrightarrow X$ be a truly random invertible func.

in EXP(0) adv. sees: $f(m_{1,0}), \ldots, f(m_{q,0})$ > q random values in X

in EXP(1) adv. sees: $f(m_{1,1}), \ldots, f(m_{q,1})$

Using AES: Det. CPA secure encryption for 16 byte messages.

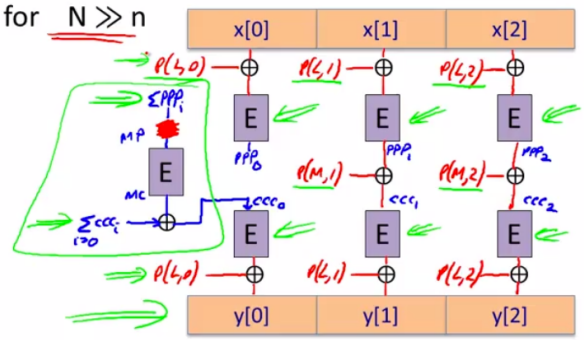
  - if longer then 16 byte messages use SIV
  - can we construct PRP that have message spaces that are larger then 16 bytes
- EME: constructing a wide block PRP
  - function P() padding function
  - EME construction very parallel

Let (E, D) be a secure PRP. $E: K \times \{0,1\}^n \longrightarrow \{0,1\}^n$

EME: a PRP on $\{0,1\}^N$ for $N \gg n$

Key = (K,L)

M ← MP@MC



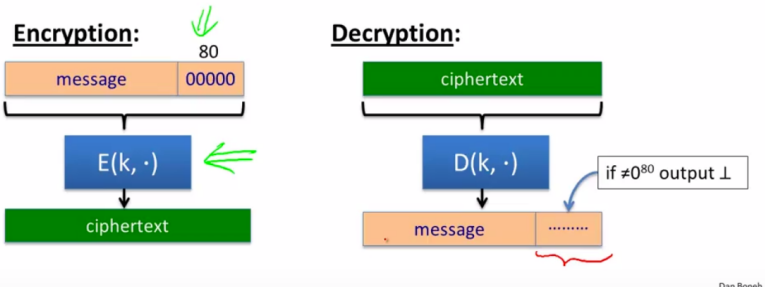
Performance:

- can be 2x slower then SIV

- PRP-based Deterministic Authenticated Encryption

Goal: det. CPA security and ciphertext integrity

⇒ **DAE: deterministic authenticated encryption**

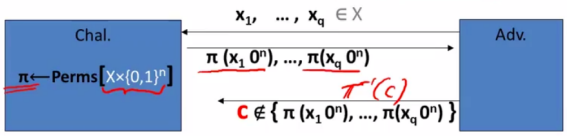


- PRP-based Deterministic Authenticated Encryption

Let (E, D) be a secure PRP. E: K × (X×{0,1}^n) ⟶ X×{0,1}^n

Thm: 1/2^n is negligible ⇒ PRP-based enc. provides DAE

Proof sketch: suffices to prove ciphertext integrity



But then $\Pr[\, LSB_n(\pi^{-1}(c)) = 0^n \,] \le 1/2^n$

- SIV
  - encrypt index in database if large
- PRP
  - encrypt index in database if small
  - append zeros which will be used as an integrity check
  - (wide block PRP from narrow PRP called EME)
- **Odds and Ends 3: disk encryption and credit card encryption**
- **Tweak able Encryption**
- Disk encryption: no expansion

Sectors on disk are fixed size (e.g. 4KB)

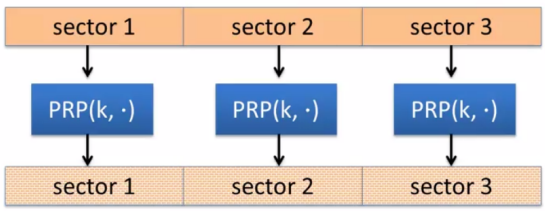
⇒ encryption cannot expand plaintext (i.e. M = C)

⇒ must use deterministic encryption, no integrity

Lemma: if (E, D) is a det. CPA secure cipher with M=C then (E, D) is a PRP.

⇒ every sector will need to be encrypted with a PRP

- encryption using a PRP

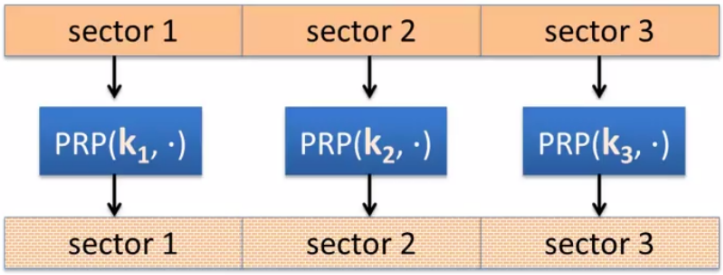


Problem: sector 1 and sector 3 may have same content

- Leaks same information as ECB mode

Can we do better?

- can we do better



Avoids previous leakage problem

- … but attacker can tell if a sector is changed and then reverted

Managing keys: the trivial construction $k_t = PRF(k, t)$, t=1,…,L

- Teakable block ciphers

Goal: construct **many** PRPs from a key $k \in K$.

Syntax: $E, D: K \times T \times X \to X$

for every $t \in T$ and $k \leftarrow K$:

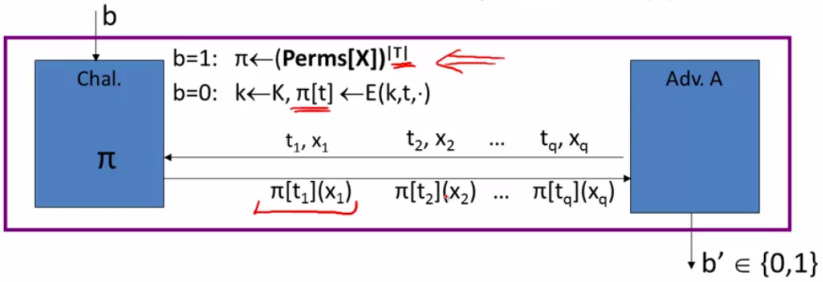
$E(k, t, \cdot)$ is an invertible func. on X, indist. from random

Application: use sector number as the tweak

⇒ every sector gets its own independent PRP

- Secure tweakable block ciphers
  - did he interact with truly random functions or pseudo random functions

$E, D: K \times T \times X \to X$. For b=0,1 define experiment EXP(b) as:



- Def: E is a secure tweakable PRP if for all efficient A:

$Adv_{tPRP}[A,E] = |Pr[EXP(0)=1] - Pr[EXP(1)=1]|$ is negligible.

Dan Bone

- Example 1: the trivial construction

Let (E,D) be a secure PRP, E: $K \times X \longrightarrow X$.

- The trivial tweakable construction: (suppose K = X)

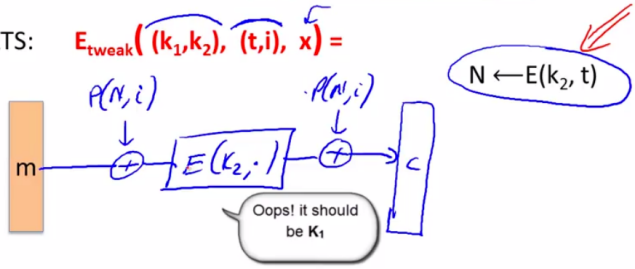
$$E_{tweak}(k, t, x) = E(E(k, t), x)$$

⇒ to encrypt n blocks need 2n evals of E(.,.)

- 2. the XTS tweak able block cipher

Let (E,D) be a secure PRP, E: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$.

- XTS: $E_{tweak}((k_1,k_2), (t,i), x) =$

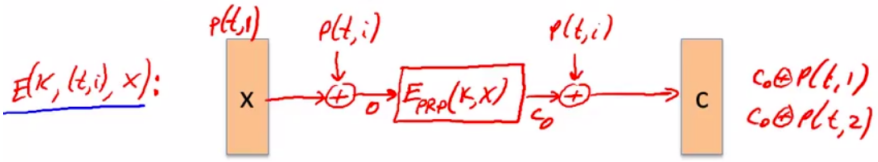


⇒ to encrypt n blocks need n+1 evals of E(.,.)

- Example

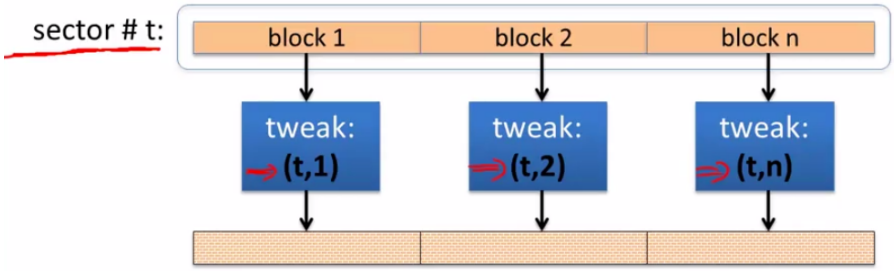
Is it necessary to encrypt the tweak before using it?

That is, is the following a secure tweakable PRP?



- ○ Yes, it is secure
- ○ No: $E(k, (t,1), P(t,2)) \oplus E(k, (t,2), P(t,1)) = P(t,1) \oplus P(t,2)$
- ⇒ ○ No: $E(k, (t,1), P(t,1)) \oplus E(k, (t,2), P(t,2)) = P(t,1) \oplus P(t,2)$
- ○ No: $E(k, (t,1), P(t,1)) \oplus E(k, (t,2), P(t,2)) = 0$

- Disk encryption using XTS



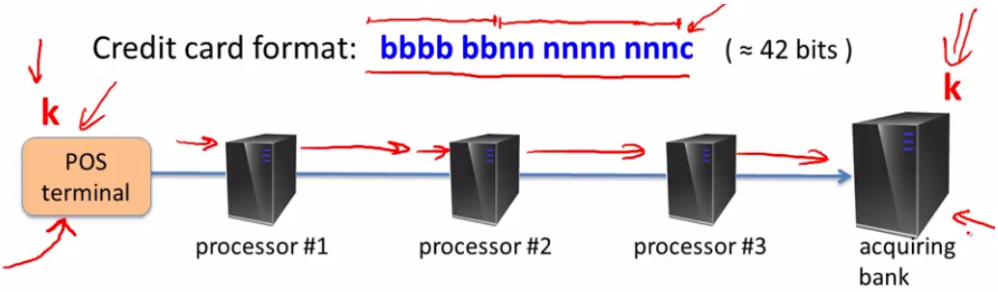
- note: block-level PRP, not sector-level PRP.
- Popular in disk encryption products:

  Mac OS X-Lion, TrueCrypt, BestCrypt, …

- Summary

- Use tweakable encryption when you need many independent PRPs from one key
- XTS is more efficient than the trivial construction
  - Both are narrow block: 16 bytes for AES
- EME (previous segment) is a tweakable mode for wide block
  - 2x slower than XTS

- **Format Preserving Encryption**
- Encrypting credit card numbers



Goal: end-to-end encryption

- intermediate processors expect to see a credit card number => encrypted credit card should look like a credit card

- Format preserving encryption (FPE)

This segment: given $0 < s \le 2^n$, build a PRP on $\{0,\ldots,s-1\}$

from a secure PRF $F: K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ (e.g. AES)

Then to encrypt a credit card number: (s = total # credit cards)

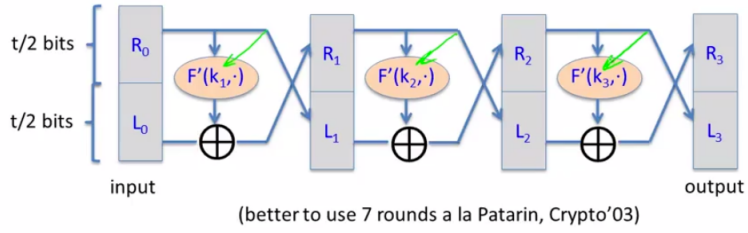
1. map given CC# to $\{0,\ldots,s-1\}$
2. apply PRP to get an output in $\{0,\ldots,s-1\}$
3. map output back a to CC#

- Step 1

Step 1: from $\{0,1\}^n$ to $\{0,1\}^t$ (t<n)

Want PRP on $\{0,\ldots,s-1\}$. Let t be such that $2^{t-1} < s \le 2^t$.

Method: Luby-Rackoff with $F': K \times \{0,1\}^{t/2} \longrightarrow \{0,1\}^{t/2}$ (truncate F)



(better to use 7 rounds a la Patarin, Crypto'03)

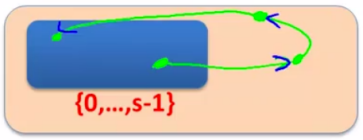
- Step 2

Given PRP $(E,D): K \times \{0,1\}^t \longrightarrow \{0,1\}^t$

we build $(E',D'): K \times \{0,\ldots,s-1\} \longrightarrow \{0,\ldots,s-1\}$

E'(k, x): on input $x \in \{0,\ldots,s-1\}$ do:

y←x; do { y ← E(k, y) } until $y \in \{0,\ldots,s-1\}$; output y



Expected # iterations: 2

- Security

Step 2 is tight: $\forall A \ \exists B: \ PRP_{adv}[A,E] = PRP_{adv}[B,E']$

Intuition: $\forall$sets $Y \subseteq X$, applying the transformation to a random perm. $\pi: X \to X$ gives a random perm. $\pi': Y \to Y$

Step 1: same security as Luby-Rackoff construction

(actually using analysis of Patarin, Crypto'03)

note: no integrity

-