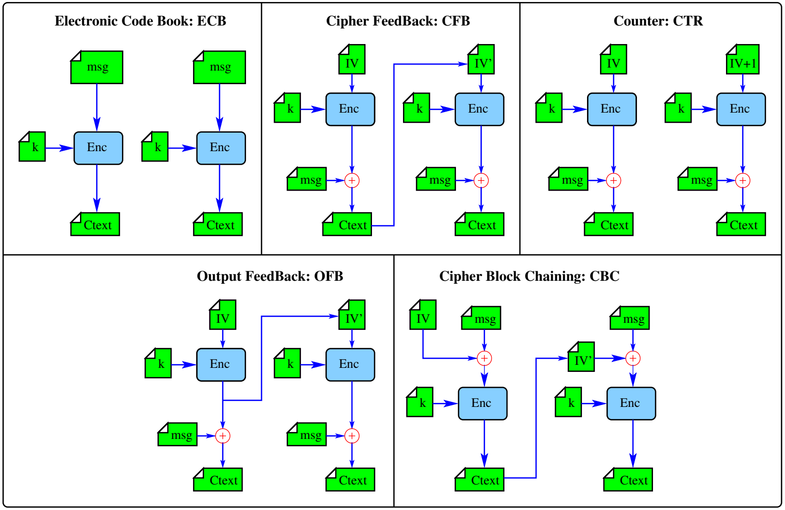
CS4236 Cryptography Theory and Practice Topic 5 - Modes, and CCA: (ECB, CFB, CTR, OFB and CBC modes)

Hugh Anderson

National University of Singapore
School of Computing

September, 2022

Modes - making "stream" ciphers



Outline

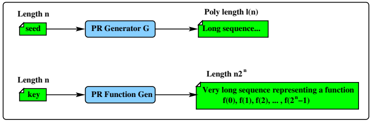
1. **Modes, and CCA**
   - ECB, CFB, CTR, OFB and CBC modes
   - CCA Security

The story so far ... where are we?

The last 4 weeks: weekly steps we have taken

1. We had a historical introduction and introduced a **framework**.
2. We progressed from the traditional view of perfect secrecy, through to a game/experiment view: *perfect* indistinguishability.
3. *Perfectly* indistinguishable was relaxed to give *computationally* indistinguishable
4. Multi-encryption and CPA-secure properties.

What is the difference between PRF and PRG again?



The adversary has access to the PRF and can query f(1), f(2) and so on.

Pseudo-random functions

How to think of these functions (REDO)...

How many different m-bit values are there? 2^m right?

Look at the following way of representing a 3-bit input, 3-bit output example function f:

$$2^3 \text{ different values } \left\{ \begin{array}{llll} f(0) & = & 001 & \text{(i.e.1)} \\ f(1) & = & 110 & \text{(i.e.6)} \\ f(2) & = & 110 & \text{(i.e.6)} \\ f(3) & = & 100 & \text{(i.e.4)} \\ f(4) & = & 001 & \text{(i.e.1)} \\ f(5) & = & 011 & \text{(i.e.3)} \\ f(6) & = & 010 & \text{(i.e.2)} \\ f(7) & = & 101 & \text{(i.e.5)} \end{array} \right.$$

The number of bits in the representation of this example function is 3×2^3, and in general for an n-bit function it would be $n \times 2^n$.

So how many different $n \times 2^n$-bit representations (and consequently functions) are there? As before it must be... $2^{n \times 2^n}$ right?

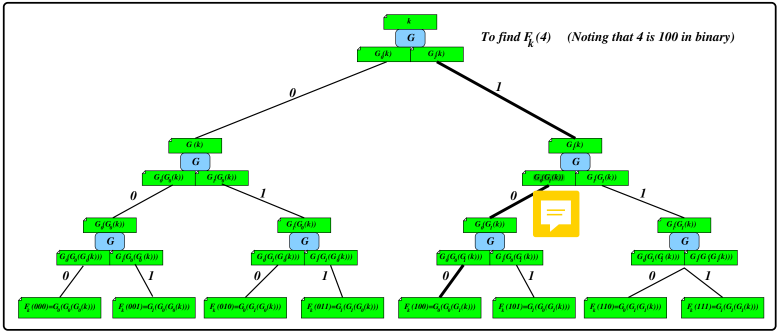
GGM - a PRF $F_k(i)$ from a PRG $G(k)$

Length doubling PRG $G : \{0,1\}^n \mapsto \{0,1\}^{2n}$, and a tree...

Assume we can get bits $0 \dots n-1$, and $n \dots 2n-1$ using functions G_0, G_1:

$$\begin{aligned} G_0(s) &= G(s)_{0\dots n-1} \\ G_1(s) &= G(s)_{n\dots 2n-1} \end{aligned}$$

Given key k we can compute the i^{th} element of the PRF from the bits of i.

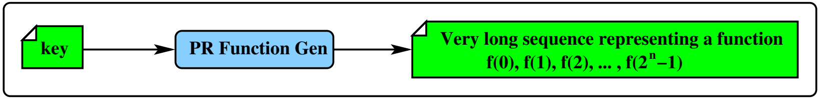


This can be proved to be a PRF, given $G()$ is a PRG, and it is PT!

Pseudorandom permutations

Previously we had pseudorandom functions

A *function* that is indistinguishable from a uniformly chosen function.


key
PR Function Gen
Very long sequence representing a function
f(0), f(1), f(2), ... , f(2ⁿ−1)

From functions to permutations...

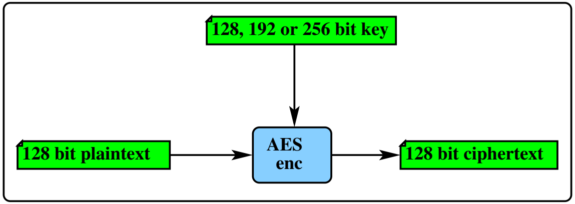
A PRP (pseudorandom permutation) generates a 1-1 and onto function that is indistinguishable from a uniformly chosen 1-1 and onto function - a permutation. Permutations can be used for encryption. They are invertible...

AES block cipher is a keyed permutation

Hopefully it is pseudorandom. There is no proof that it is indeed a pseudorandom permutation, but many people believe that it is so.

$$\mathrm{AES}_{\mathrm{enc}} : \{0,1\}^{128,192 \text{ or } 256} \times \{0,1\}^{128} \longrightarrow \{0,1\}^{128}$$

Modes of operation



From blocks to messages - block/stream ciphers...

We can extend the fixed size AES block cipher to encrypt arbitrary long messages, using the “modes” of operation.

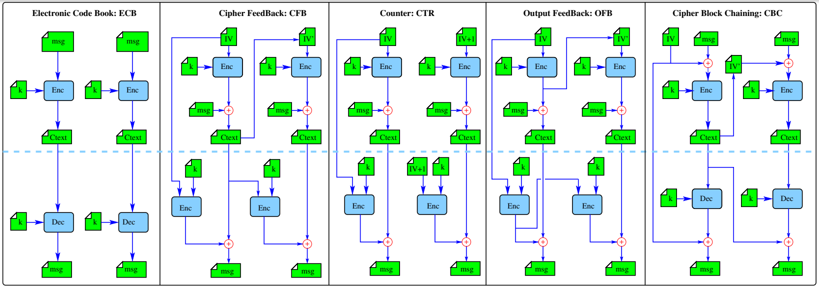
Now, if we assume that AES block cipher is a pseudorandom permutation, is the extended version CPA secure? Or even CCA (to be defined later) secure?

Well known modes...

ECB, CFB, CTR, OFB, CBC.

ECB shouldn’t be used. Nevertheless, some still use it!

Efficiency of ECB, CFB, CTR, OFB, CBC



Remarks

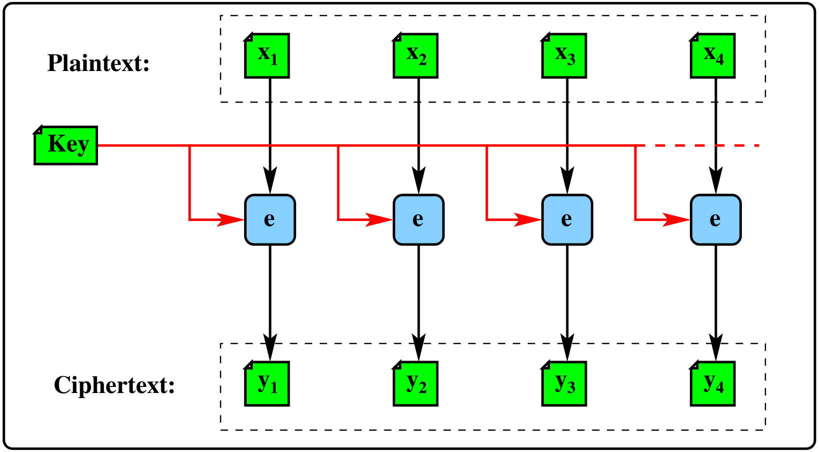
Some modes can be done in parallel, some can have pre-processing, etc. This is important in practice but not discussed here.

You may wonder why bother about modes? Why not just encrypt independently and each block has a randomly chosen IV? This *is* CPA-secure.

...

Unfortunately, this mode generates double-sized ciphertext. It is called *Unsynchronized mode* in the textbook (pg 87).

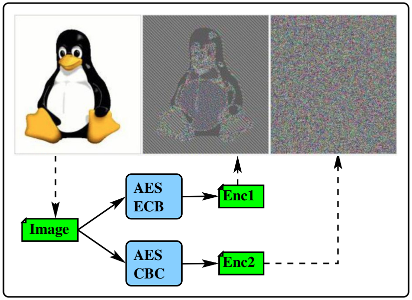
ECB mode



Comments on ECB mode

Each block of the plaintext is encrypted with the same key. Encryption and decryption can be done in parallel.
We could make this CPA-secure, by adding a random initial value (IV) for each block - but the resultant encryption would be large.

Security of ECB



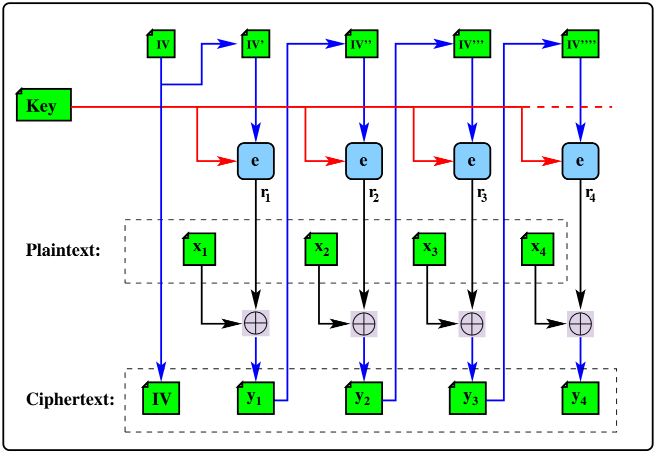
Remark

Clearly not CPA-secure. In this example, the image is divided into blocks, and (the middle image) encrypted with some deterministic encryption scheme using the same key. Since it is deterministic, any two blocks that are exactly the same will be encrypted to the same ciphertext.

Example of misuse

Should (must) avoid ECB. Surprisingly, there are popular systems that adopted ECB, in particular Zoom. . .

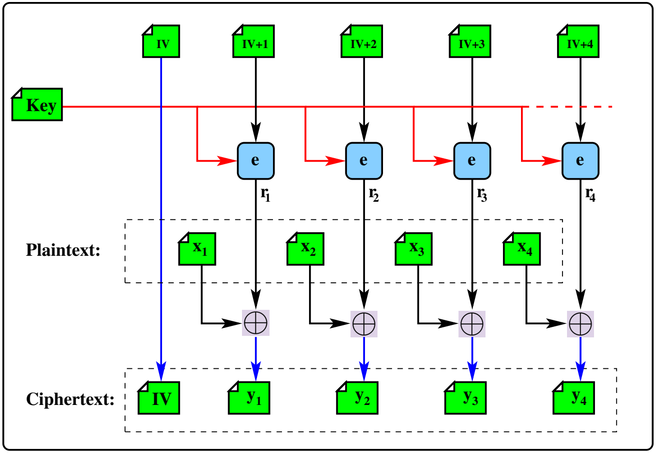
CFB mode



Comments on CFB mode

Each IV is encrypted with the same key, and blocks use the previous ciphertext block as the next IV. Decryption can be done in parallel, but not encryption.
CPA-secure, and incorrect blocks affect all following blocks.

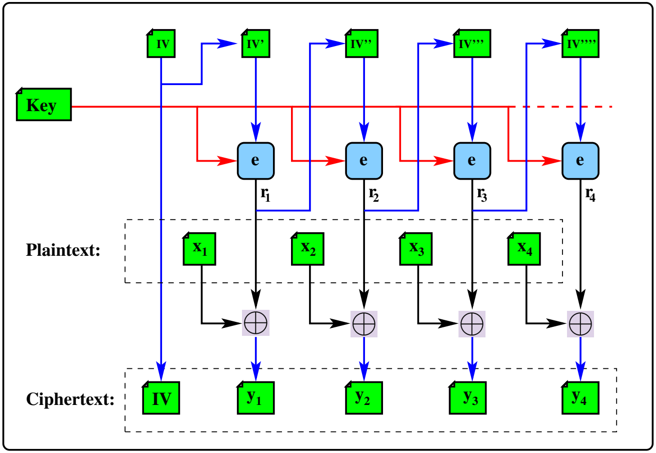
CTR Counter mode



Remark

IV for each block is predictable - encryption and decryption can be done in parallel. Bad blocks only affect current block.

OFB mode



Remark

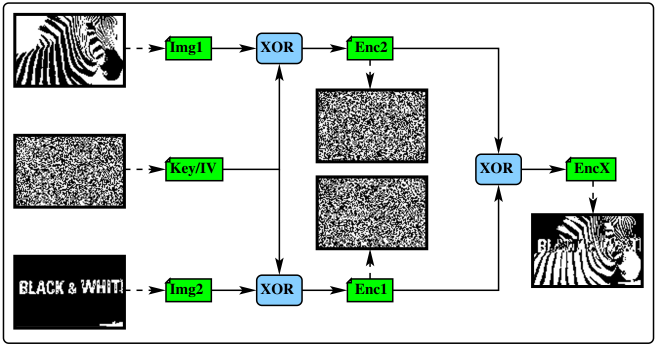
IV for each block is less predictable, encryption and decryption cannot be done in parallel. Bad blocks only affect current block.

Security of CFB, CTR, OFB modes

All can be shown to be CPA-secure

permutation

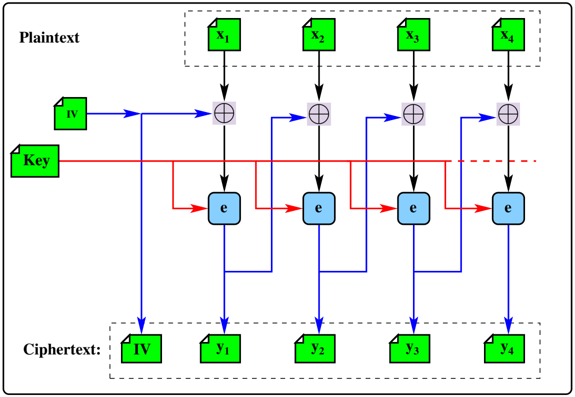
... given that the block cipher is a pseudorandom [redacted]



OFB re-use of IVs? No!

The diagram shows the consequence of IV reuse.
Also, in CTR mode, if the difference of two IVs is small, and the message is long, there could be overlap in the IVs. Note that OFB would not do this.

CBC mode



Comments on the IV

The IV is an arbitrary value chosen before encryption. So, it is different in different encryptions of the same plaintext. Depending on the implementation, an IV can be randomly chosen, obtained from a counter, or from other information (date/location etc). Decryption possible in parallel.

Security of CBC mode

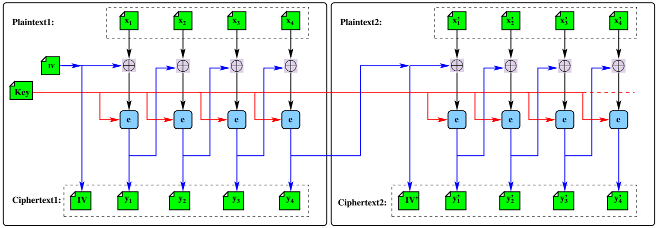
Remarks

CBC can also be shown to be CPA-secure.

Many argue that CBC is more “secure” because modification of a bit in the ciphertext would affect the decryption of subsequent bits, and thus able to more easily detect error.
This is more related to “integrity” and “authentication”. For encryption, those should not be considered. In fact, we shouldn’t rely on CBC mode for integrity of messages as it might not be secure.

For CBC mode, the IV must be “unpredictable”. E.g. Suppose that we use a counter as IV, i.e. for the first message use $x + 1$, the second message uses $x + 2, \ldots$ where x is randomly chosen. This is not secure under CPA.

Security of CBC in TLS 1.0



Context for the attack

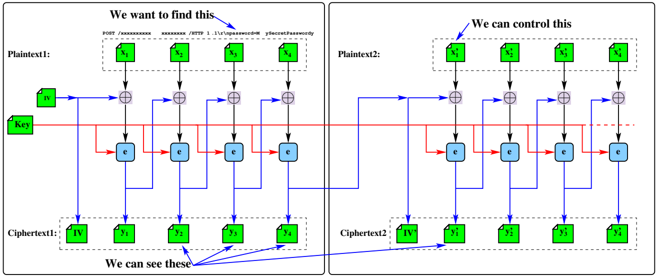
TLS 1.0 (used for https) used CBC to encrypt the messages. To save having to regenerate an IV for a second (multi-block) message, the last encrypted message is used as the next IV.

Leads to an attack: The "BEAST" attack!

This man-in-the-middle attack make use of the fact that the IV is predictable. It is a well-known flaw in TLS:

`https://rdist.root.org/2008/02/05/tlsssl-predictable-iv-flaw/`

Rizzo/Duong Attack (Beast Attack) on TLS


We want to find this
POST /xxxxxxxxxx xxxxxxxx /HTTP 1 .1\r\npassword=M ySecretPasswordy
Plaintext1:
x_1 x_2 x_3 x_4
IV
Key
e
Ciphertext1:
IV y_1 y_2 y_3 y_4
We can see these
We can control this
Plaintext2:
x'_1 x'_2 x'_3 x'_4
Ciphertext2
IV' y'_1 y'_2 y'_3 y'_4

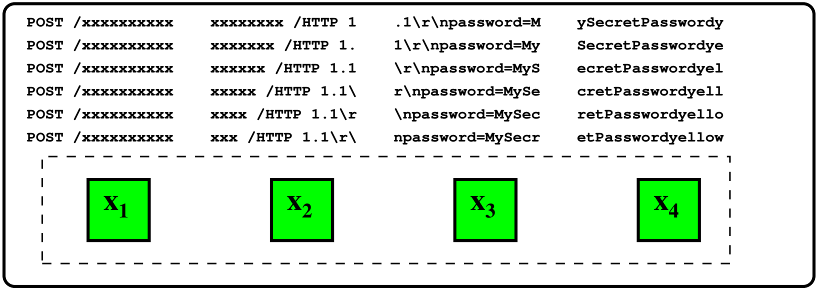
BEAST: Browser Exploit Against SSL/TLS

The attack is exploited by Rizzo & Duong on TLS (BEAST).
A malicious site with some javascript/java may convince a browser to send crafted messages (A CPA), and by observing the encrypted messages, perhaps discover an important cookie or token.

Description

https://www.exploit-db.com/docs/english/15137-practical-padding-oracle-attacks.pdf

Attack model



```
POST /xxxxxxxxxx    xxxxxxxx /HTTP 1    .1\r\npassword=M    ySecretPasswordy
POST /xxxxxxxxxx    xxxxxxx /HTTP 1.    1\r\npassword=My    SecretPasswordye
POST /xxxxxxxxxx    xxxxxx /HTTP 1.1    \r\npassword=MyS    ecretPasswordyel
POST /xxxxxxxxxx    xxxxx /HTTP 1.1\    r\npassword=MySe    cretPasswordyell
POST /xxxxxxxxxx    xxxx /HTTP 1.1\r    \npassword=MySec    retPasswordyello
POST /xxxxxxxxxx    xxx /HTTP 1.1\r\    npassword=MySecr    etPasswordyellow
```

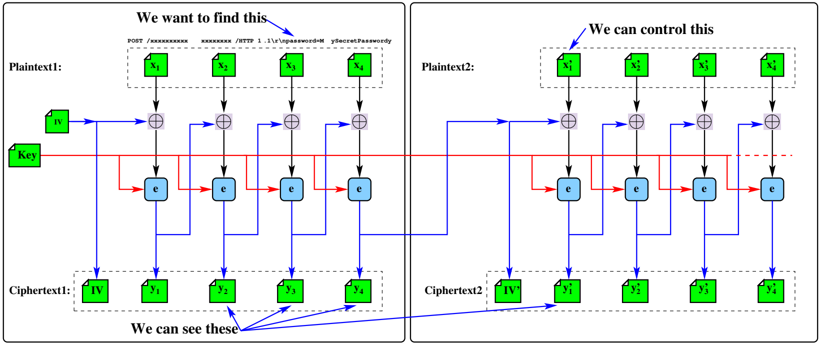
Encoding

$\mathrm{Enc}_k(\mathrm{IV}, m)$ is encryption with the key k, the plaintext m, and initial value IV. The attacker does not know a message m and the secret key k. The attacker knows the value of IV before feeding the message to the oracle.

Attacker has access to restricted "encryption" oracle

By feeding the oracle any message m_0, the oracle returns the ciphertext (encrypted using k) of a message $m_0 \mathbin{+\!\!+} m$, where $\mathbin{+\!\!+}$ is concatenation. The attacker's goal is to find m.

The attack... Bytewise? Bitwise?

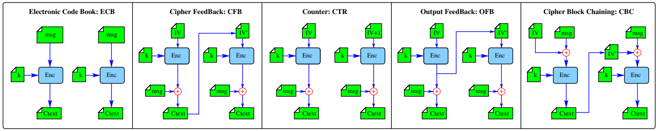


Algorithm to find first character of password string

Note that $x_3 = '.1\backslash r\backslash npassword = M'$, and we know y_2, y_3, y_4. Set m_0 to $'.1\backslash r\backslash npassword ='$,and for each guess g_*: $g_a = m_0 + 'a', g_b = m_0 + 'b', \ldots$

- Set $x'_1 = y_2 \oplus y_4 \oplus g_*$ and (oracle) finds $y'_1 = \mathrm{Enc}_k(\mathrm{IV}, x'_i)$
- If $g_* = x_3$ then y'_1 will be the same as y_3
- Repeat (until $g_* = g_M$ - the correct character guess).

Comparing security of OFB, CBC, CTR, OFB



Remarks, particularly wrt IVs

OFB, CBC, CTR, OFB can be proven to be CPA-secure. They are not CCA-secure.

1. CBC's IV cannot be predictable. Otherwise, vulnerable to CPA attack.
2. IV must not be reused. In the unfortunate situation that it is being reused, the differences of the messages will be leaked. CTR, OFB will leak more than CBC. This is because in CBC, from the 2nd block onward, the chained IV in the two encryptions likely would be different.

If implemented correctly, the probability that two encryptions use the same IV is extremely low. Unfortunately, as demonstrated in many past incidents, many applications do not implement or adopt crypto constructions correctly, leading to reduced sets of IVs, with consequent issues.

CCA (Chosen Ciphertext Attack)

Game definition

Under CPA-security, the adversary can access an encryption oracle. Now, what about “decryption” oracle? Can we formulate CCA-security?

The formulation of CCA-security is similar to CPA-security, except:

1. The adversary has additional access to a restricted decryption oracle, as well as the encryption oracle.
2. The decryption oracle refuses to take in the challenge as input. That is, it will decrypt anything sent by the adversary, except the challenge c.

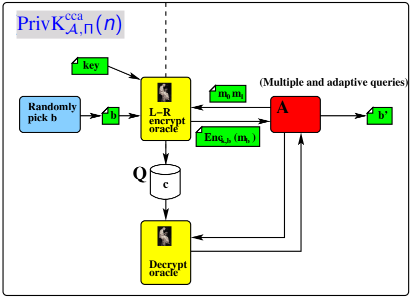
We will not give the full description of the experiment $\mathrm{PrivK}^{\mathrm{cca}}_{\mathcal{A},\Pi}$: see pg 96.

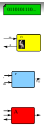
Definition 3.33

A private-key encryption scheme Π has indistinguishable encryptions under a CCA attack, or is CCA-secure, if for any PPT $\mathcal{A}$, there is a negl s.t.

$$\Pr[\mathrm{PrivK}^{\mathrm{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

Is CCA practical?


$\mathrm{PrivK}^{\mathrm{cca}}_{\mathcal{A},\Pi}(n)$
key
Randomly pick b
b
L–R encrypt oracle
$\mathrm{m_0\ m_1}$
(Multiple and adaptive queries)
A
b'
$\mathrm{Enc_{k,b}\ (m_b\ )}$
Q
c
Decrypt oracle



CCA commentary

CBC, CTR, OFB mode are not CCA secure. It is difficult to achieve CCA-security. Try constructing a simple counter example on CTR.

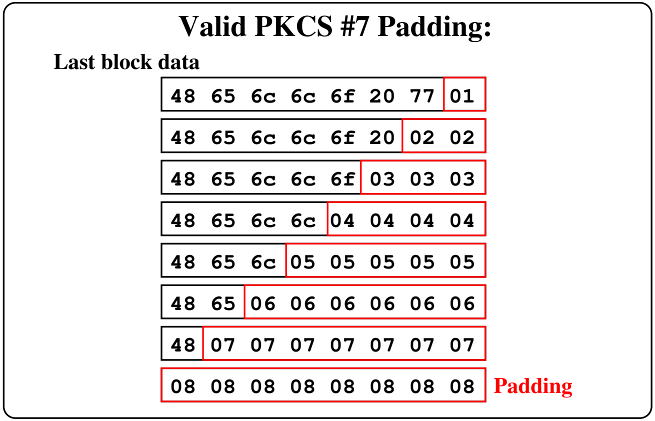
Some argue that CCA-security is important. The Padding oracle attack is a weaker (more difficult for attacker) form of CCA attack, where an oracle reveals only if the message is in the correct padding format.

You may have already seen padding oracles in other modules, but we will briefly repeat it here. We will re-visit CCA-security in the next topic.

Bit and byte padding for block ciphers...

PKCS #7 - a byte-wise padding technique...

One technique is to add a single 1 bit, followed by enough 0 bits to fill out the block. Here we have the valid PKCS #7 paddings for a final 8 byte (64 bit) block. Each byte is given as 2 hex digits:



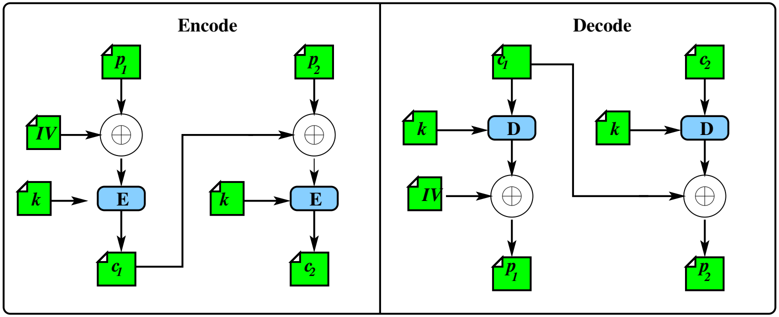
Valid PKCS #7 Padding:

Last block data

48	65	6c	6c	6f	20	77	01	
48	65	6c	6c	6f	20	02	02	
48	65	6c	6c	6f	03	03	03	
48	65	6c	6c	04	04	04	04	
48	65	6c	05	05	05	05	05	
48	65	06	06	06	06	06	06	
48	07	07	07	07	07	07	07	
08	08	08	08	08	08	08	08	Padding

CBC: encoding and decoding...

Encode is XOR, then encrypt...

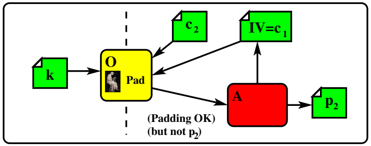


(...and of course, decode is decrypt, then XOR).

Note that when you are decoding, the final plaintext p_2 is the decrypted ciphertext exclusive-ored with c_1.

If there was only one block, the final plaintext p_1 is the decrypted ciphertext exclusive-ored with IV

Padding oracle...



What is a padding oracle?

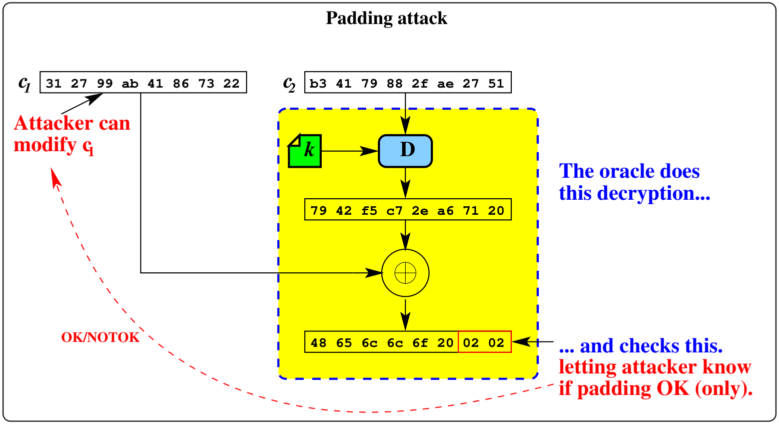
Imagine that you have a bit of software, that, given c_1 and c_2, will tell you if the padding is correct in p_2. If the message was only one block long, given IV and c_1, it will tell you if the padding is correct in p_1. In general, given c_{n-1} and c_n, it will tell you if the padding is correct. We call this a padding oracle.

Is this realistic? Well, yes it is.
When you send a padded, encrypted message to another system, after decrypting, the other system will check the padding, and if it is not valid, will fail immediately. Otherwise it will fail (or succeed) later.
In other words, normal receiving software may leak the correctness of the padding. Normal receiving software may be a padding oracle.

Note also that an attacker can directly change each bit/byte of the final plaintext for p_n, because the attacker has control over the input c_{n-1}.

Using the oracle in an attack...

The attacker does not get to see the decrypted data...

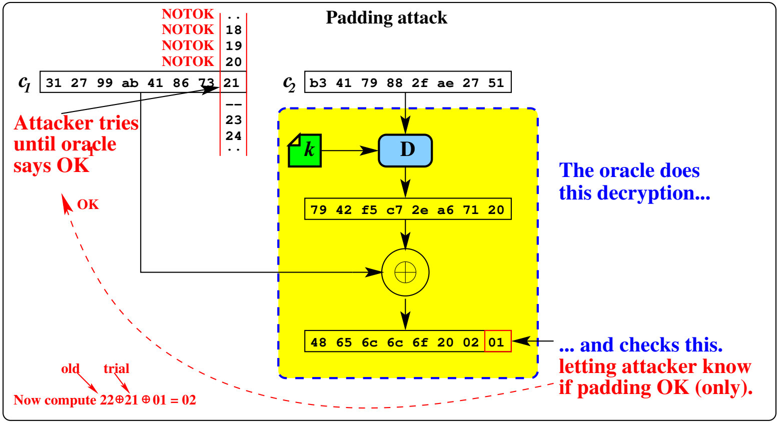


The oracle only learns if the last bytes are correct - they could be `01`, `0202`, `030303`, and so on.

The attacker starts by trying a series of values for the last byte of c_1:

Using the oracle in an attack...

Attacker discovers last byte...



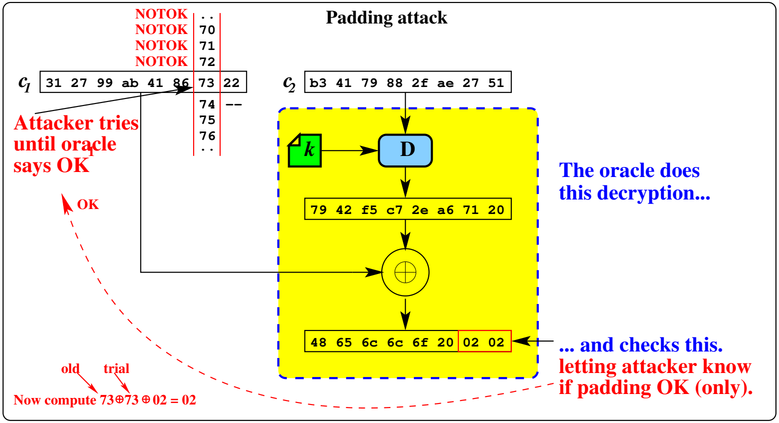
Padding attack
NOTOK ..
NOTOK 18
NOTOK 19
NOTOK 20
c1 31 27 99 ab 41 86 73 21
--
23
24
..
c2 b3 41 79 88 2f ae 27 51
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 6f 20 02 01
Attacker tries until oracle says OK
OK
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 22⊕21 ⊕ 01 = 02

The attacker is trying values in the last byte of c_1 until **01** is the last byte. The oracle will return padding OK...

Attacker now sets the last byte to **02**, and tries for the second-last byte of c_1:

Using the oracle in an attack...

Attacker discovers second-to-last byte...

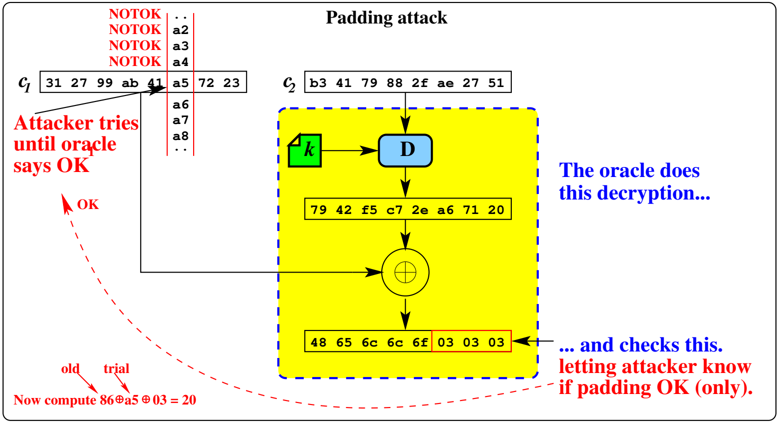


Since we know (now) the last byte (**02**), and the original last byte of c_1, we can set the last byte to anything we want.

We try all the second-to-last values in c_1, and uncover a second byte.

Using the oracle in an attack...

Attacker discovers third-to-last byte...



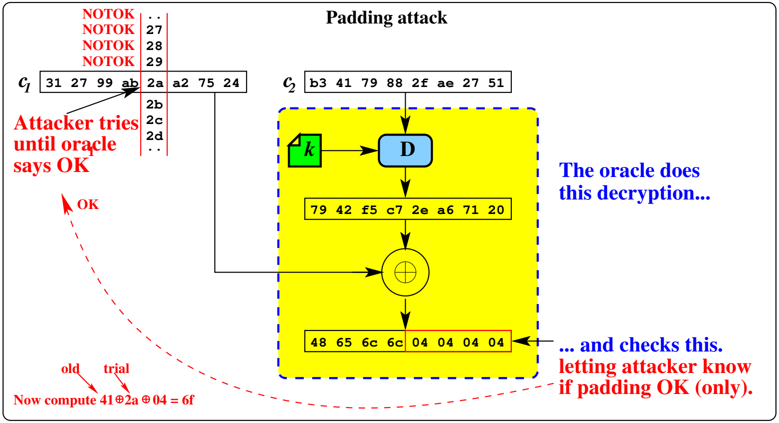
Padding attack
NOTOK ..
NOTOK a2
NOTOK a3
NOTOK a4
c_1 31 27 99 ab 41 a5 72 23
a6
a7
a8
..
c_2 b3 41 79 88 2f ae 27 51
Attacker tries until oracle says OK
OK
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 6f 03 03 03
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 86⊕a5 ⊕ 03 = 20

Since we know (now) the last two bytes (02 02), we can set the last two bytes to 03 03.

Now we try all the third-to-last values in c_1 and uncover 20.

Using the oracle in an attack...

Attacker discovers fourth-to-last byte...


Padding attack
NOTOK ..
NOTOK 27
NOTOK 28
NOTOK 29
c_1 31 27 99 ab 2a a2 75 24
2b
2c
2d
..
c_2 b3 41 79 88 2f ae 27 51
Attacker tries until oracle says OK
OK
k
D
79 42 f5 c7 2e a6 71 20
48 65 6c 6c 04 04 04 04
The oracle does this decryption...
... and checks this.
letting attacker know if padding OK (only).
old
trial
Now compute 41⊕2a ⊕ 04 = 6f

Since we know (now) the last three bytes (**20 02 02**), we can set the last three bytes to **04 04 04**.

Now we try all the fourth-to-last values in c_1 and uncover **6f**.

Using the oracle in an attack...

Attacker discovers all bytes...

The attacker keeps moving through the block, and has to try a maximum of 255 trials for each byte recovered, so this is an $\mathcal{O}(n)$ attack, where n is the number of bytes in the block.

This attack can be avoided if...

- There is no oracle (i.e. the receiving code does not give any indication if the padding is correct or not)
- CFB is used instead.
- A different padding scheme is used...

That is it - a little aside into cracking that makes use of much of what we have learned.