# Implementing CBC

Petr Ročkai

## Overview

- we will use the AES block encryption function
- to implement the CBC mode

- this is just an exercise
- to understand how CBC works
- you should not do this in real projects

## Cipher Modes

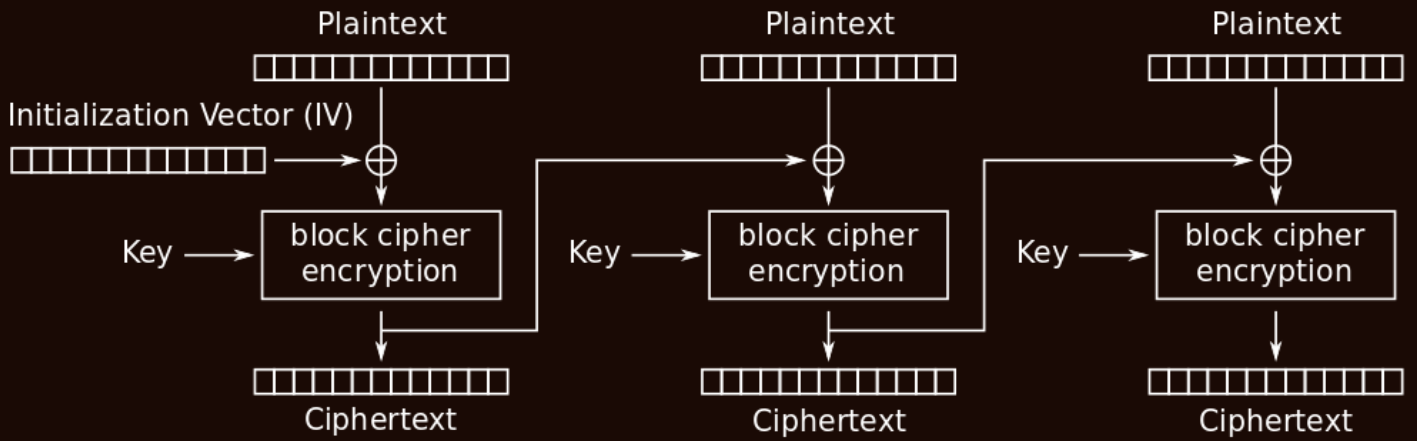
- a block cipher can only encrypt one block at a time
- typically same size as the key
- plaintext length must be divisible by block length → padding

## ECB

- split the message into block-sized chunks
- encrypt each block separately
- insecure

# Can we do better? CBC

- XOR previous ciphertext into current plaintext



## CBC Properties

- error-resistant (self-synchronising)
- parallel decryption is possible
- can't encrypt in parallel

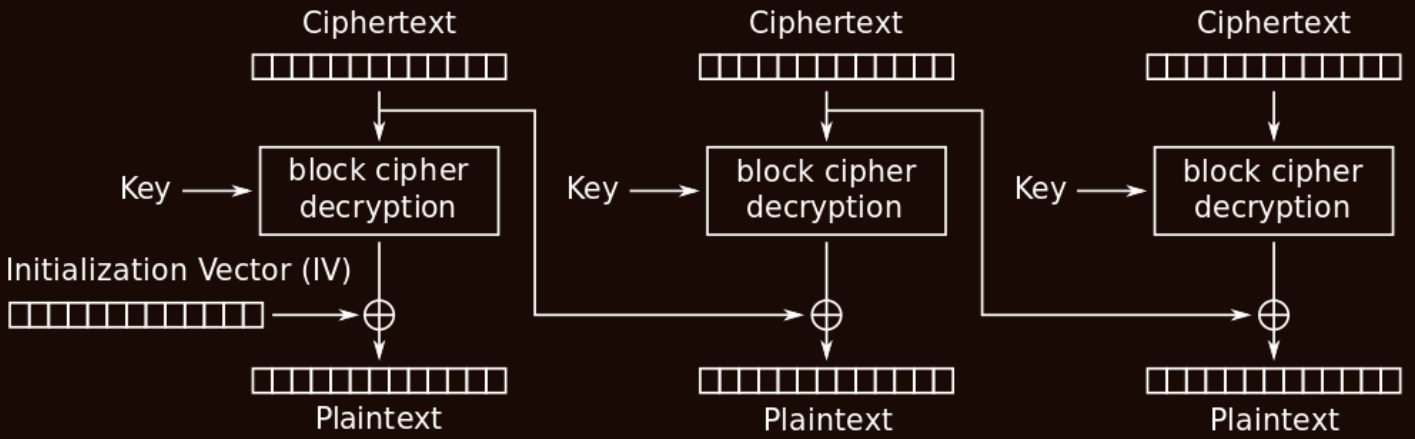
## How to Pad

- let $n$ be the number of missing bytes
- clearly $0 < n < 255$, so it fits in a byte
- PKCS7: each padding byte just repeats $n$
- example: ???? ???? ??? → ???? ???? ???5 5555

## Padding Oracles

- possible if a server indicates a padding error
- an apparently minor info leak compromises the cipher
- CBC with PKCS7 padding is vulnerable

# CBC Decryption



## A Padding Oracle

- assume 128b AES: 1 block = 16 bytes
- consider a ciphertext (IV, $C_1$, $C_2$) that decrypts to ($P_1$, $P_2$)
- consider (hex) $P_2$ = ???? ???? ???? ??01 → OK
- what about $P_2$ = ???? ???? ???? ??12 → ERROR

## Recovering the Last Byte

- set $C_1'[15] = C_1[15] \oplus X \oplus 0x01$
- send (IV, $C_1'$, $C_2$) to the oracle
- if we get OK, it's likely that $P_2[15] = X$

## Correctness

- $(I_1, I_2)$ are the intermediate results from AES block decrypt
- $C_1[15] \oplus X \oplus 0x01 \oplus I_2[15] = 0x01$ $/ \oplus I_2[15]$
- $C_1[15] \oplus X \oplus 0x01 = 0x01 \oplus I_2[15]$ $/ \oplus 0x01$
- $C_1[15] \oplus X = I_2[15]$ $/ \oplus X \oplus I_2[15]$
- $C_1[15] \oplus I_2[15] = X$

## Getting More Bytes

- if we already know $X = P_2[15]$
- we can set $C_1'[15]$ to $C_1[15] \oplus X \oplus 0x02$
- and $C_1'[14]$ to $C_1[14] \oplus Y \oplus 0x02$
- and guess again until we hit the right Y

## This Lab

- download and compile the skeleton from study materials
- implement `my_encrypt_cbc` only using `aes_crypt_ecb`
- create a new file, eg. `cbc.c` with a new `main()` function
- start working on your assignment (next slide)

# Assignment 3

- implement the padding oracle attack
  - recovery of the last byte (1pt)
  - recovery of an entire block (1pt)
- is a specific error code/message required?
  - what other info could the attacker use? explain (1pt)
- what could you do to defend against the attack? (1pt)
  - take previous into account
  - describe at least 2 modes of defence
- implement the better of those 2 defences (1pt)

## Assignment 3 (cont'd)

- pick any block you like for your attack
- the function `performServerDecrypt` is your oracle
- do not modify this function
- make a copy for implementing your defence
- mention your sources
- the deadline is Thu 26th at midnight