Stream Ciphers

Gaspare FERRARO

CybersecNatLab

Matteo ROSSI

Politecnico di Torino

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal

- Present some issues of the previously seen block ciphers
- Introduce stream ciphers as a way to handle messages of non-fixed sizes
- Present some of the most common modes of operation and their vulnerabilities
- Introduce an example of a native stream cipher and its possible attacks

Prerequisites

- Lecture:
  - *CR_1.3 – Block Ciphers*

Recap

- Remaining problems from block ciphers:
  - How can we deal with non-fixed input sizes?
  - How can we exchange keys?
  - How can we provide authentication?
- In this lecture we address the first of these three problems

Outline

- Introduction
- Modes of operation and vulnerabilities
- CTR mode and native stream ciphers
- Attacks on native stream ciphers

Outline

- Introduction
- Modes of operation and vulnerabilities
- CTR mode and native stream ciphers
- Attacks on native stream ciphers

Introduction

- A *stream cipher* is a symmetric-key encryption algorithm that encrypts a stream of bits of *any* (finite) *length*
- Real-world stream ciphers have limits on the maximum length, but they are normally sufficiently large not to pose a practical problem

Outline

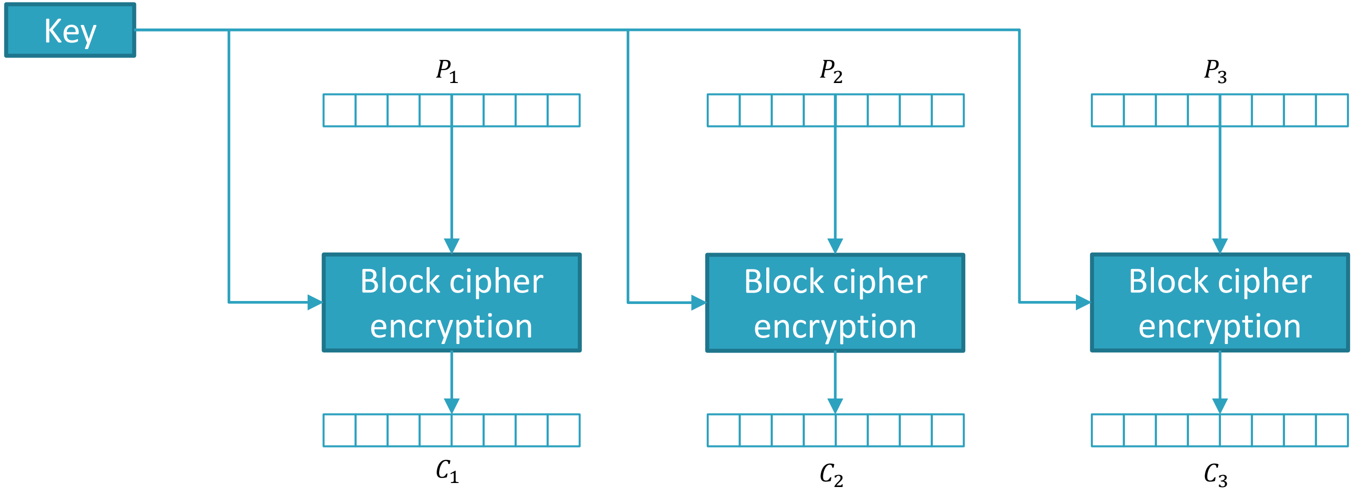
- Introduction
- Modes of operation and vulnerabilities
- CTR mode and native stream ciphers
- Attacks on native stream ciphers

A first naïve attempt

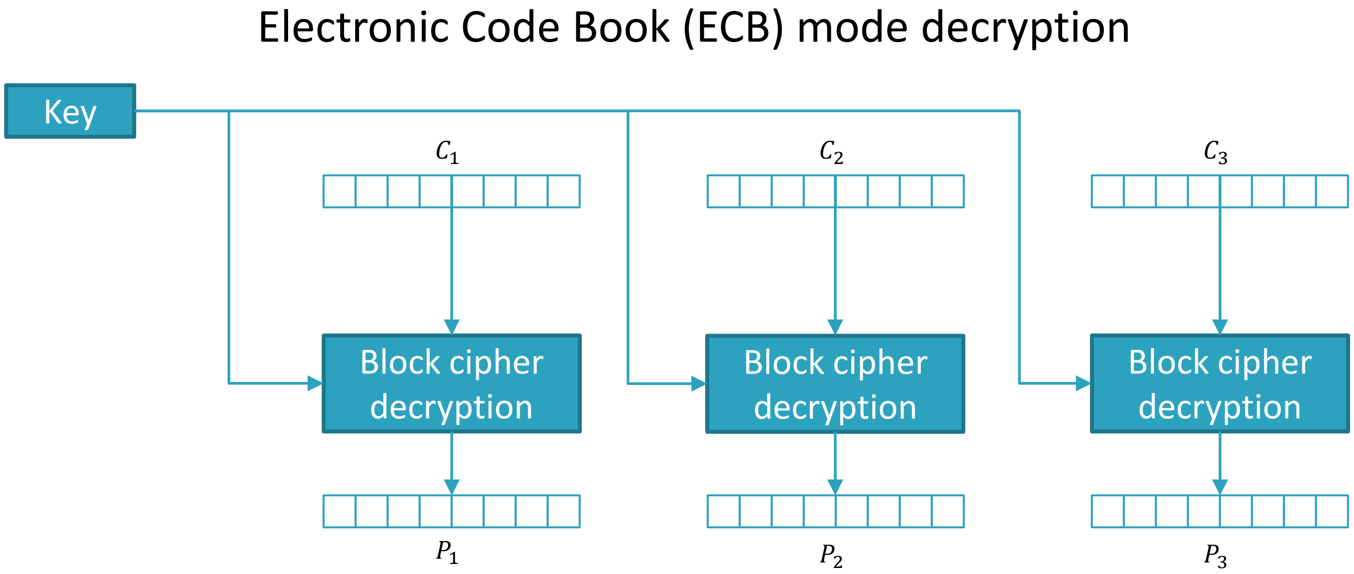
- Let’s try to use what we already have:
  - Suppose that the length n of the message to encrypt is a multiple of b, for a certain b
  - Suppose that we have a block cipher with blocks of size b
  - Split the messages in n/b parts $p_1, p_2, \ldots$ and encrypt every part with the same key to $c_1, c_2, \ldots$
  - This is called *Electronic Code Book Mode* (ECB Mode)

ECB Mode of Operation - Encryption

Electronic Code Book (ECB) mode encryption



ECB Mode of Operation - Decryption



ECB Mode – Issues

- Issues:
  - The multiple of b assumption is too restrictive (more on this later)
  - Equal blocks will give equal ciphertexts
  - The global structure of the encrypted message is preserved

ECB Mode – Example

Image before ECB Encryption

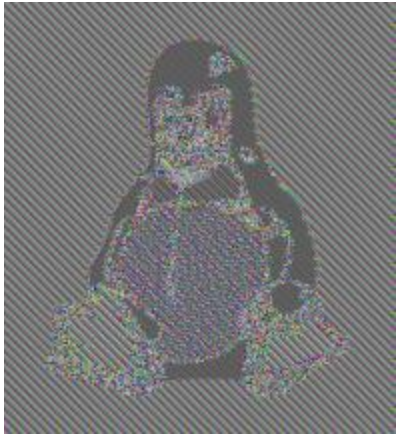


Image after ECB Encryption

Images from https://commons.wikimedia.org/

Stream Ciphers – Encryption Oracle

For the remaining part of this section, we call an encryption oracle a service that, given a plaintext message P, returns the corresponding ciphertext C using always the same key

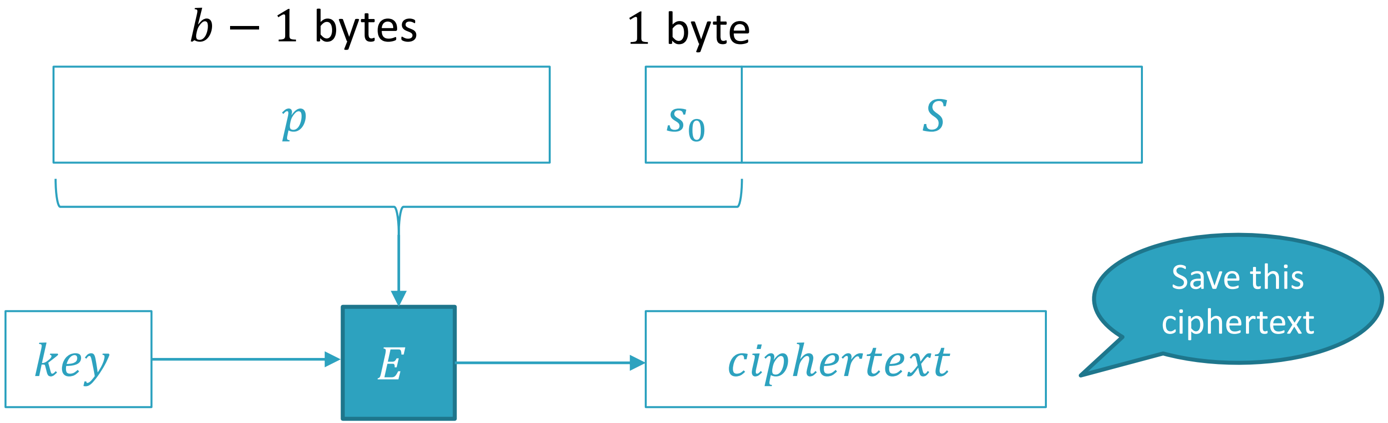
ECB Oracle Attack

- We show that, if misimplemented, ECB can be completely broken
- Scenario: an oracle that returns $C = ECB(key, P||S)$, where:
  - P is a chosen plaintext
  - S is a secret string
  - || is the string concatenation operator
- In this scenario, we can recover S regardless the used block cipher

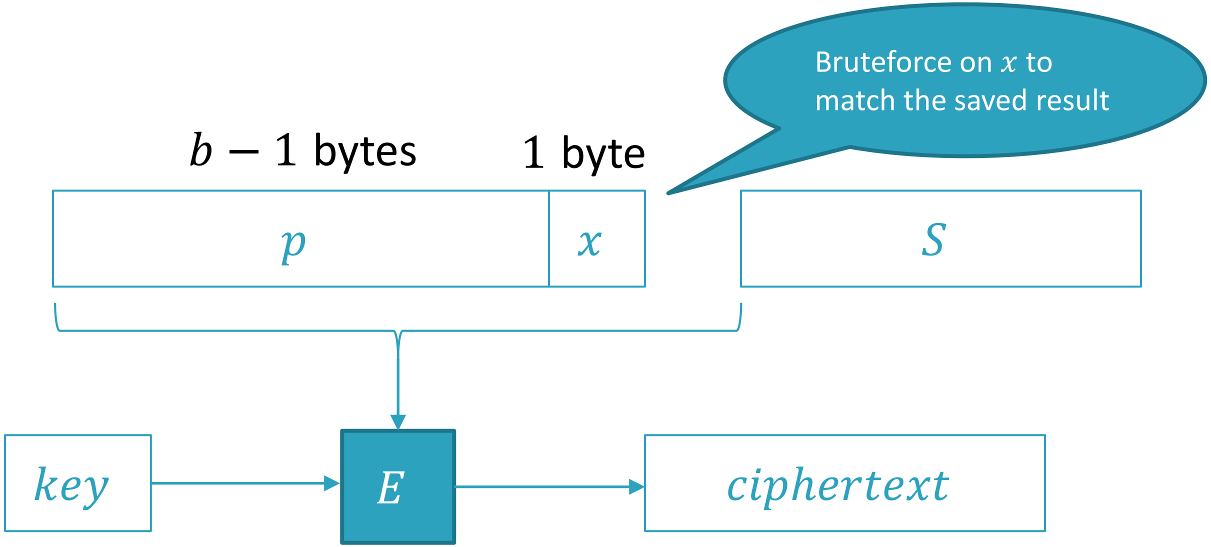
ECB Oracle Attack

- Strategy:
  - We send a message that is 1 byte shorter than the block size and we save the result
  - We bruteforce the last byte until we find the same ciphertext
  - We proceed like this, bruteforcing one byte at a time

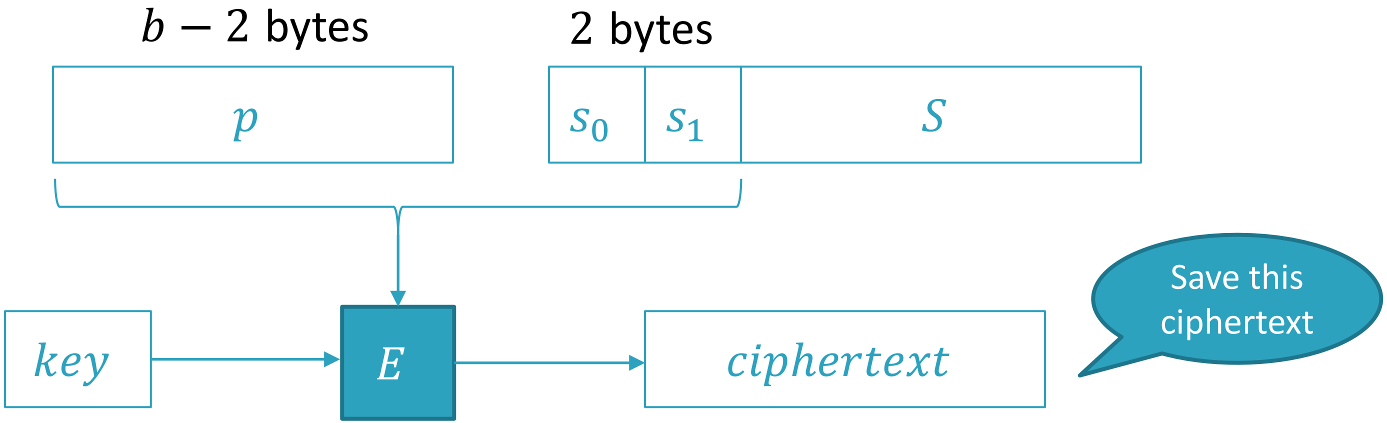
ECB Oracle Attack – step 1



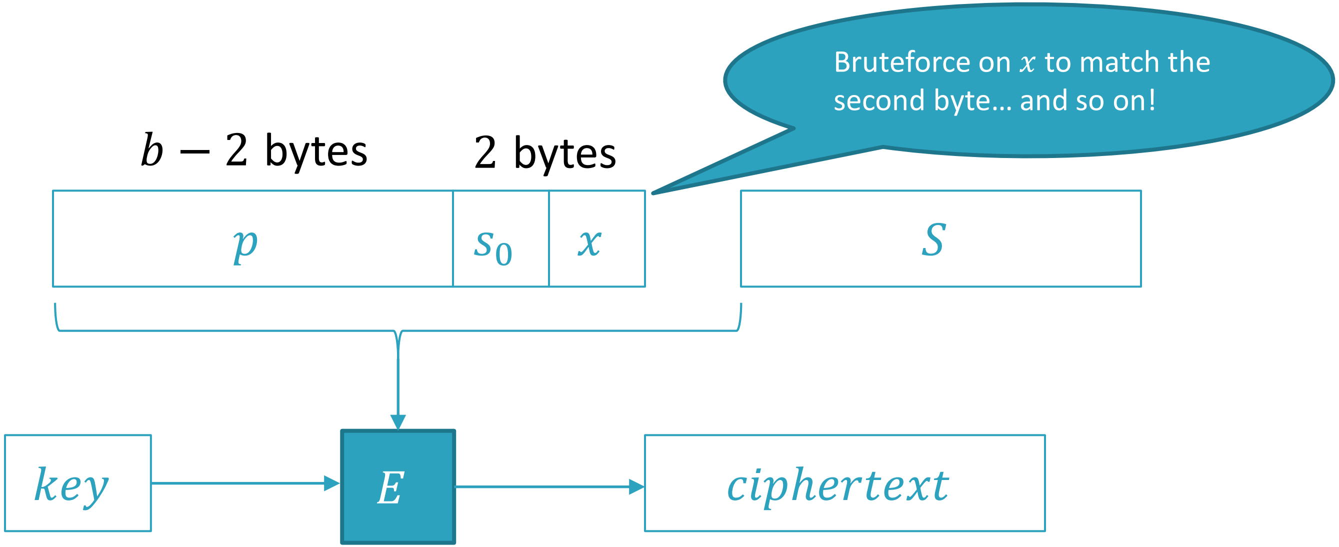
ECB Oracle Attack – step 2



ECB Oracle Attack – step 3



ECB Oracle Attack – step 4



ECB Oracle Attack – Performance

- With AES-128 we have that:
  - Bruteforcing the key takes $2^{128} = 256^{16}$ tries
  - ECB Oracle takes only $256 * 16$ tries!

Stream Ciphers – Modes of Operation

- ECB is in general very ineffective, but we can stick with the idea of using block ciphers, just in a different configuration.
- A configuration to make a system based on a block cipher behave like a stream cipher is called a *mode of operation*
- Before introducing a new mode of operation, let’s take a step back...

Padding

- We want to drop the assumption that the plaintext length is a multiple of the block length
- We do this simply by *completing* our plaintext to get the desired length. This operation is called *padding*

Padding

- First idea: add null bytes ($0x00$) to the end until we get the correct length
- Issue: we can not remove the padding after decryption!
- Better idea: encode the length of the padding in the padding itself

Padding – PKCS#5/PKCS#7

- Clever idea: the value of each added byte is the number of bytes that are added
- This is defined in the *PKCS#5* and *PKCS#7* standards.
- Example: if 3 bytes are missing the padding is $0x03\ 0x03\ 0x03$
- Note: if the plaintext has already the correct length *a whole new block is added*

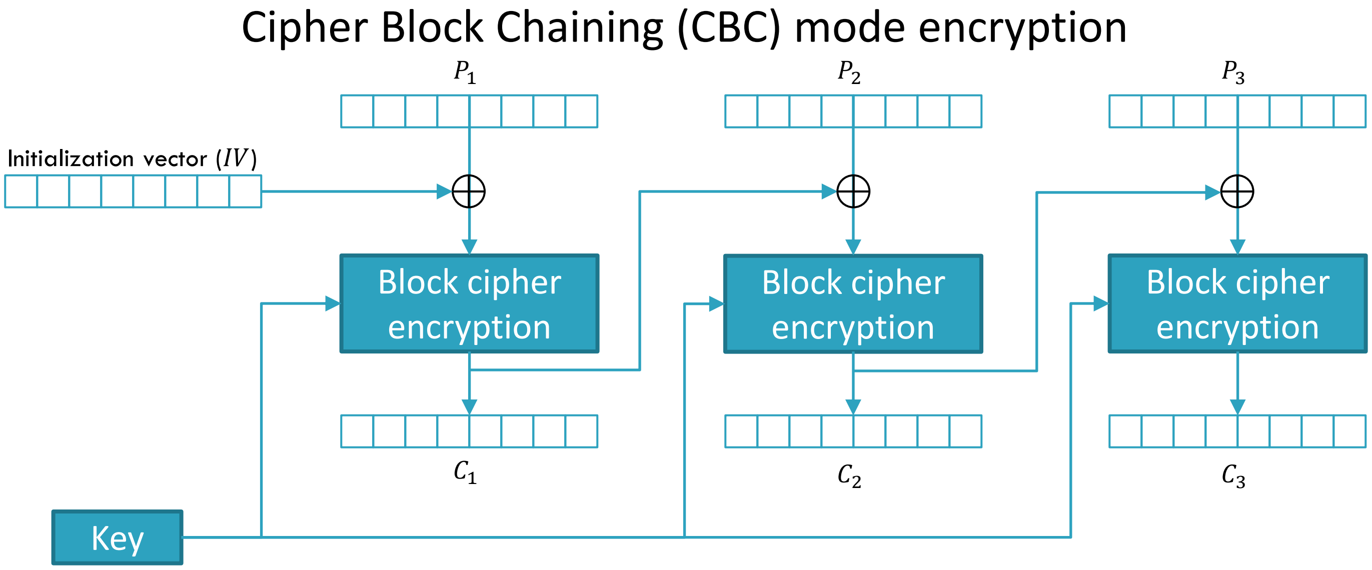
CBC Mode of Operation

- We introduce now a better mode of operation: the *Cipher Block Chaining* (CBC) mode
- The general idea of CBC is to *destroy the plaintext structure* using information from the previous blocks to encrypt

CBC Mode of Operation

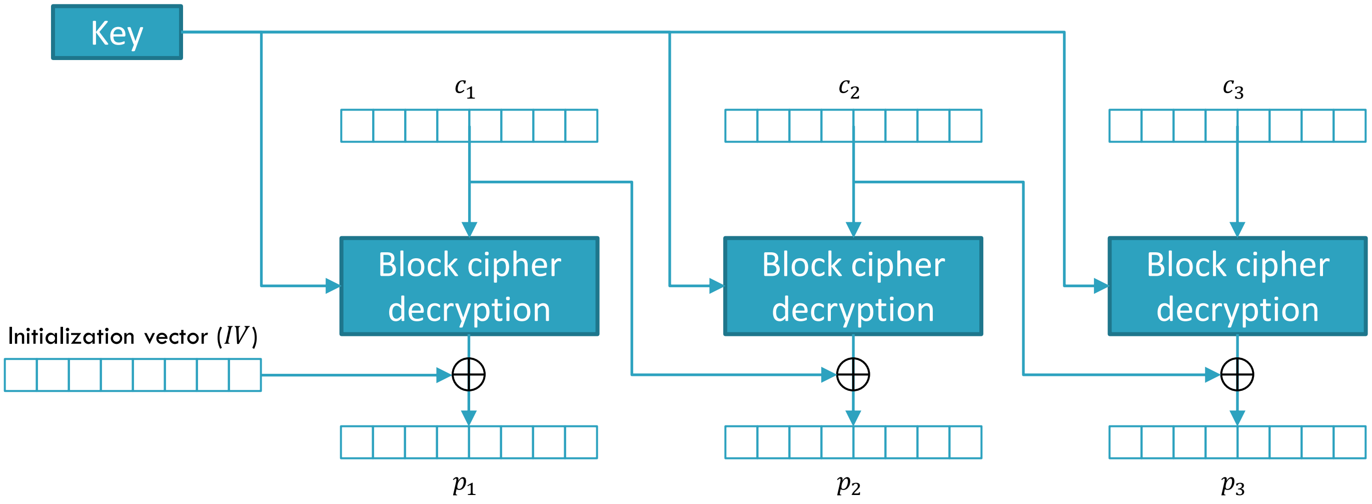
- The general CBC encryption flow is the following:
  - Apply padding to the plaintext and split the plaintext P into blocks $P_1, P_2, P_3, \ldots$
  - Take a key k and an additional random string with the same length of the blocks, called IV (Initialization Vector)
  - For the first block, apply the bitwise XOR operation $\oplus$ between the IV and the first plaintext block P_1 , then encrypt using the key k:
    $$C_1 = E(k, IV \oplus P_1)$$
  - For the next blocks, apply the bitwise XOR operation $\oplus$ between the i^{th} plaintext block P_i and the $(i-1)^{th}$ ciphertext block, then encrypt using the key k:
    $$C_i = E(k, C_{i-1} \oplus P_i)$$

CBC Mode of Operation - Encryption



CBC Mode of Operation - Decryption

Cipher Block Chaining (CBC) mode decryption



CBC vs ECB

- Plaintext structure is no longer maintained
- The same plaintext block repeated gives different encrypted blocks
- The ECB Oracle Attack does not work here because of the IV

CBC – Remarks on the IV

- Randomness in the IV is important: an adversary should not be able to predict an IV before the encryption
- *IV is not a key*: in practice it is shared in plaintext with the encrypted message
- The IV should be *different for every encryption*

CBC Issues

- In the following slides we show the most common problems when using CBC mode, in particular we will show that:
  - The choice of the IV is crucial
  - A small information leakage can lead to a disaster

CBC Issues – key as the IV

- Scenario:
  - A server implements a CBC scheme by using the key (fixed) as the IV (without revealing it)
  - You can ask the server to decrypt a message
  - Can you retrieve the key?

CBC Issues – key as the IV

- Strategy:
  - Send to the server a message with 2 equal blocks BB
  - Obtain $P_1 = D(k, B) \oplus IV$ and $P_2 = D(k, B) \oplus B$
  - Calculate $P_1 \oplus P_2 \oplus B = IV = k$

CBC Issues – Padding Oracle Attack

- Scenario:
  - We have a target ciphertext correctly padded to decrypt
  - We have a *padding oracle*: a server that given a ciphertext simply tells you if the padding is correct (this happens in real life!)

CBC Issues – Padding Oracle Attack

- Outline of the attack (for 1 block ciphertext C):
  - Create a random block R
  - Append the target block obtaining $R||C$
  - Discover the padding length using the oracle
  - Decrypt one byte at a time exploiting it

CBC Issues – Padding Oracle Attack

- Step 1: look for a "correct padding" message
  - Try to decrypt $R||C$
  - With high probability, you will get "wrong padding"
  - Keep changing the last byte of R in order to get "correct padding"
  - Now you know that the decryption of $R||C$ ends in $0x01$ or $0x02\ 0x02$ or $0x03\ 0x03\ 0x03$ or ...

CBC Issues – Padding Oracle Attack

- Step 2: find the length of the padding
  - Let R now be the block that gives "correct padding"
  - Change randomly the first byte of R: if it still gives correct padding, the padding length is $b - 1$ or less
  - Change randomly the second byte of R: if it still gives correct padding, the padding length is $b - 2$ or less, and so on
  - If you reach an "incorrect padding" on the k^{th} byte, you found the padding length!

CBC Issues – Padding Oracle Attack

- Step 3: decrypt the padding bytes
  - Now we discovered (at least) one byte of the plaintext
  - In reality, we discovered n bytes, where n is the padding length
  - In order to get them, just XOR the corresponding bytes of R with the padding bytes

CBC Issues – Padding Oracle Attack

- Step 4: decrypt subsequent bytes
  - To get one more byte, we need to "increase the padding"
  - To do it, XOR the padding bytes with $n \oplus (n+1)$ (this just increase them by 1)
  - Repeat from step 1 using the first non-padding byte instead of the last one!

CBC Issues

- In addition to implementation problems, CBC has some native issues:
  - Data is partially malleable
  - There is no check on data integrity

CBC Issues – Bitflipping Attack

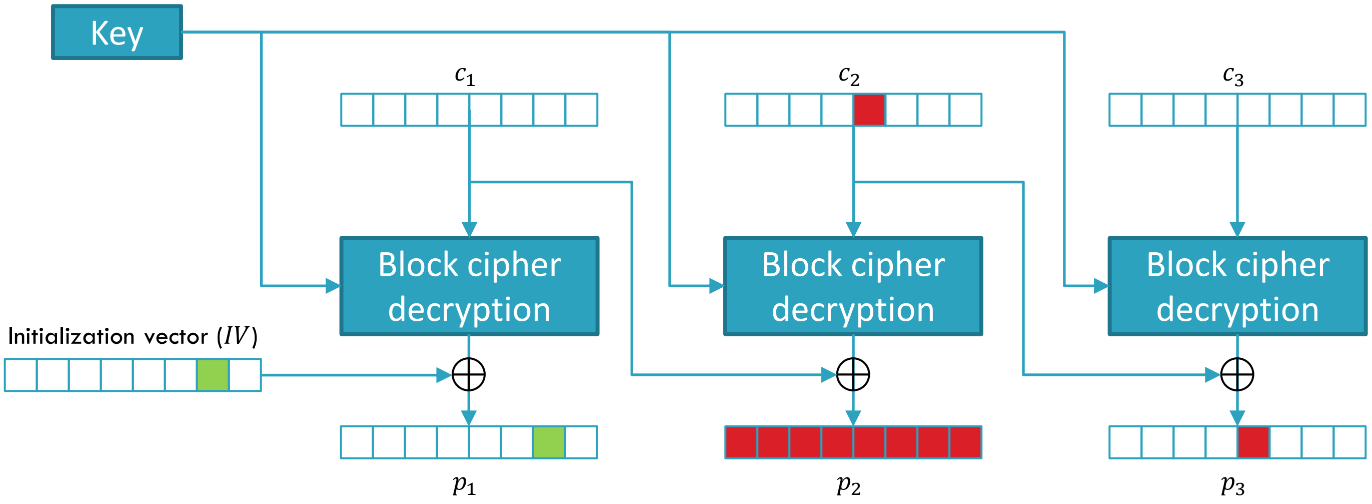
- Scenario:
  - We have a partially controlled CBC-encrypted message, with some secret information inside
  - We show that it is possible to "sacrifice" a piece of plaintext in order to edit the secret part

CBC Issues – Bitflipping Attack

- Attack outline:
  - We reserve an entire block with our controlled data
  - We XOR that block with its plaintext value and the value that we want to put in the secret part
  - Paying the price of destroying our controlled part, we control the secret without controlling the key

CBC Issues – Bitflipping Attack

Cipher Block Chaining (CBC) mode decryption



Outline

- Introduction
- Modes of operation and vulnerabilities
- CTR mode and native stream ciphers
- Attacks on native stream ciphers

Counter Mode & Native Stream Ciphers

- In this last section, we introduce ciphers that don't rely on the concept of "blocks"
- In these ciphers, the plaintext and the ciphertext have the same length
- The structure of block cipher in general remains, but it is used differently!

Counter Mode

- We present here our last mode of operation for block ciphers
- The idea is very simple: we don't use the block cipher as a cipher, but as something that generates a stream to feed a one-time pad
- This is called Counter Mode (CTR)

Counter Mode

- In practice:
  - We generate a random number N, called the *nonce* (number used once)
  - We encrypt strings formed by the nonce concatenated to a counter with the block cipher (and a key k) to generate some bytes
  - We use these bytes as a stream for a one-time pad

Counter Mode – Example

- Here's a toy example with AES-128:
  - Take a random number, for example "12345678"
  - Encrypt 1234567800000000 to generate the first 16 bytes
  - Encrypt 1234567800000001 to generate 16 more bytes
  - Encrypt 1234567800000002 and so on, until you reach the desired number of bytes

Other Modes of Operation

- We have seen ECB, CBC and CTR, but there are a lot of different modes of operation:
  - Cipher FeedBack (CFB)
  - Output FeedBack (OFB)
  - Galois Counter Mode (GCM)
  - … and many more!

Native Stream Ciphers

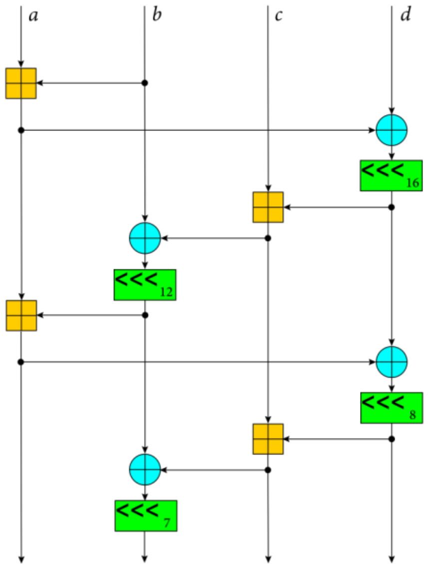
- Some ciphers are built to natively work as the CTR mode: we call these ciphers *native stream ciphers*
- Most of them work on an internal state (like AES) and in practice they generate a block of data, to then cut it to the desired length

Example – ChaCha20

- One of the most used native stream ciphers is *ChaCha20*
- It is a variant of *Salsa20* published in 2008
- It has an *ARX* structure: it uses only (modular) Additions, Rotations and XORs

Example – ChaCha20

- ChaCha20 works on a 4×4 state matrix of 32-bit numbers
- The first row is filled with constants, the second and third one are for the key (up to 256-bit), and the last one behaves like a counter
- For 20 rounds, the function in the picture is applied to the 4 columns and diagonals of the state matrix



Outline

- Introduction
- Modes of operation and vulnerabilities
- CTR mode and native stream ciphers
- Attacks on native stream ciphers

Native Stream Ciphers - Issues

- Stream ciphers can have some vulnerabilities similar to block ciphers, like:
  - On native stream cipher (or CTR mode), bitflipping is easier (you can do it directly!)
  - If nonces are reused, the same stream is generated
  - They don't mask the length of the plaintext (we may leak some information!)

Gaspare FERRARO

CybersecNatLab

Matteo ROSSI

Politecnico di Torino

Stream Ciphers