Security in Informatics and Communications

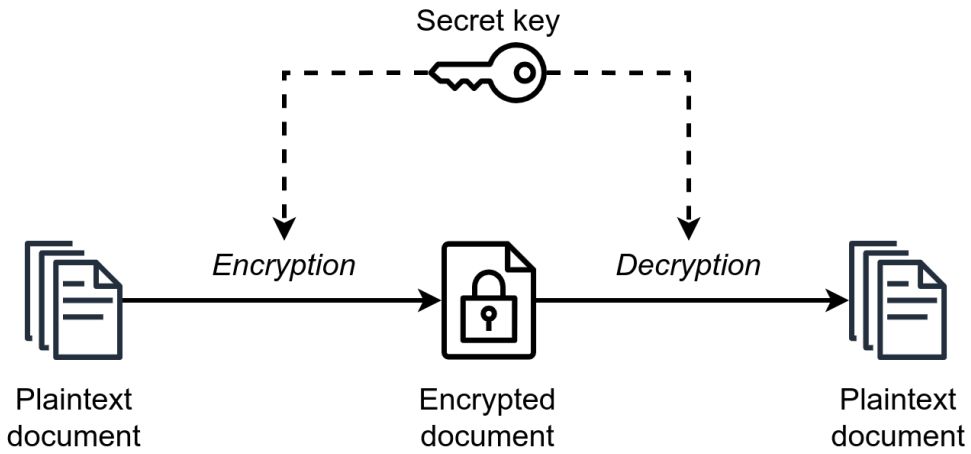
Practical Class #5:

Symmetric Cryptography

Vítor Cunha (vitorcunha@ua.pt)

What is Symmetric Cryptography?

It is the process of using the same shared secret to cipher and decipher data

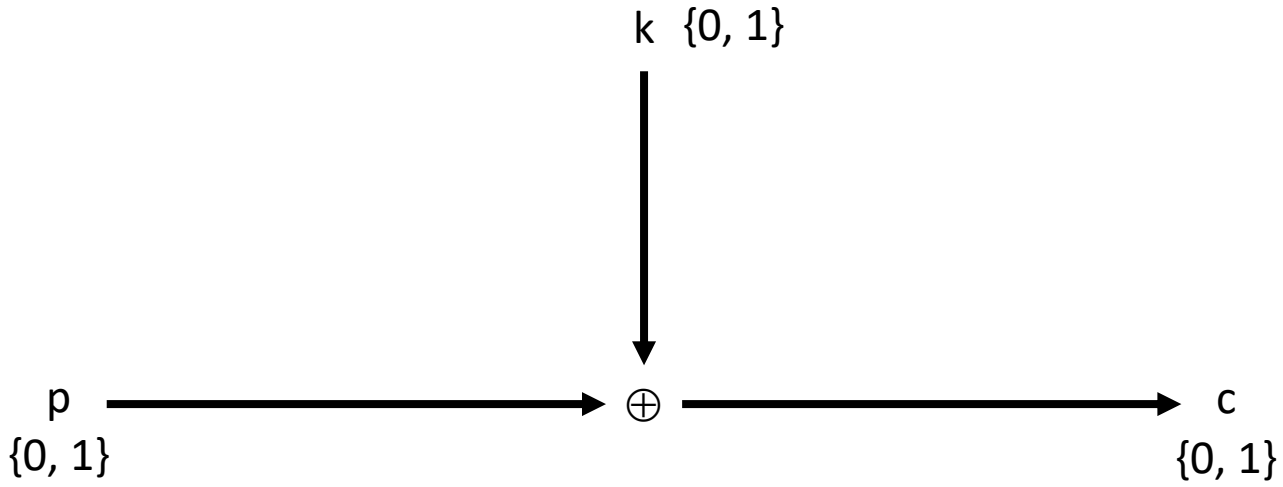


https://en.wikipedia.org/wiki/Symmetric-key_algorithm

Symmetric Cryptography

One Time Pad

(Vernam, as per US Patents Office -- disputed)



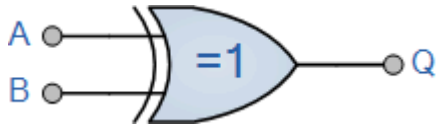
random

k	p	c
0	0	0
0	1	1
1	0	1
1	1	0

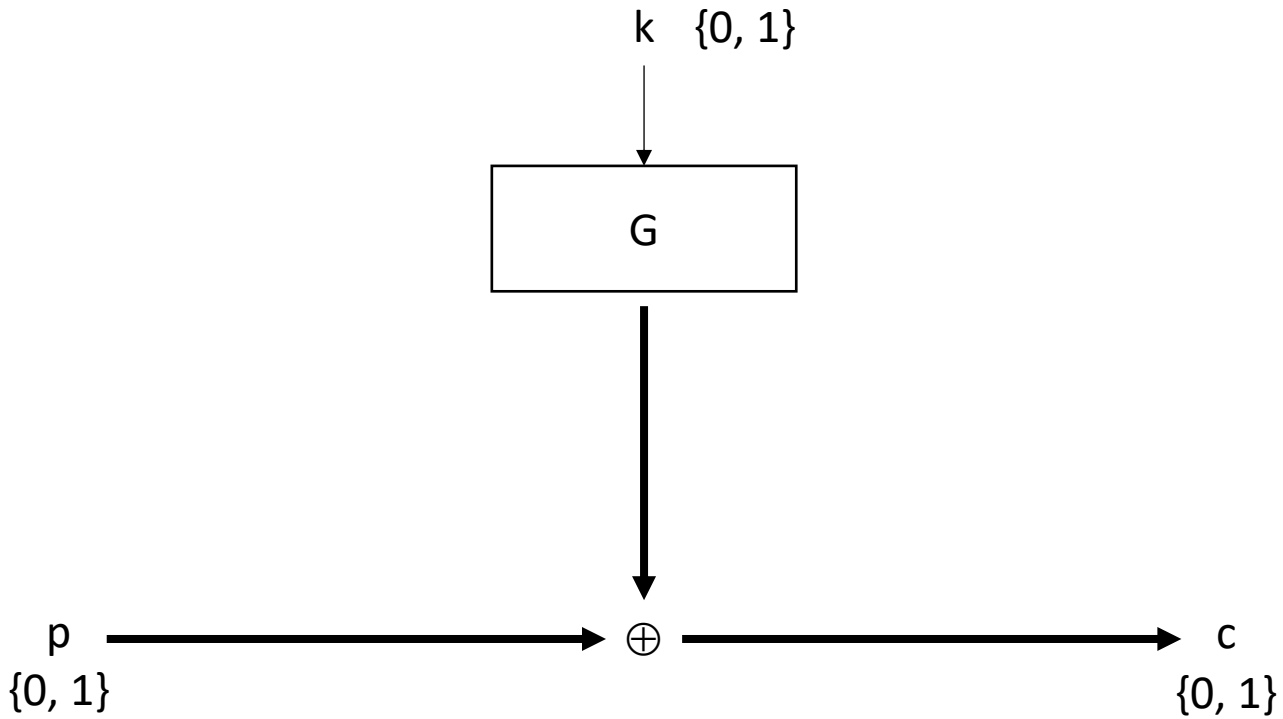
If k is truly random:

$P_c(p) = 1/2$

$P_c(\overline{p}) = 1/2$



Symmetric Cryptography



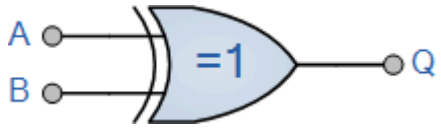
G	p	c
0	0	0
0	1	1
1	0	1
1	1	0

If G is not truly random:

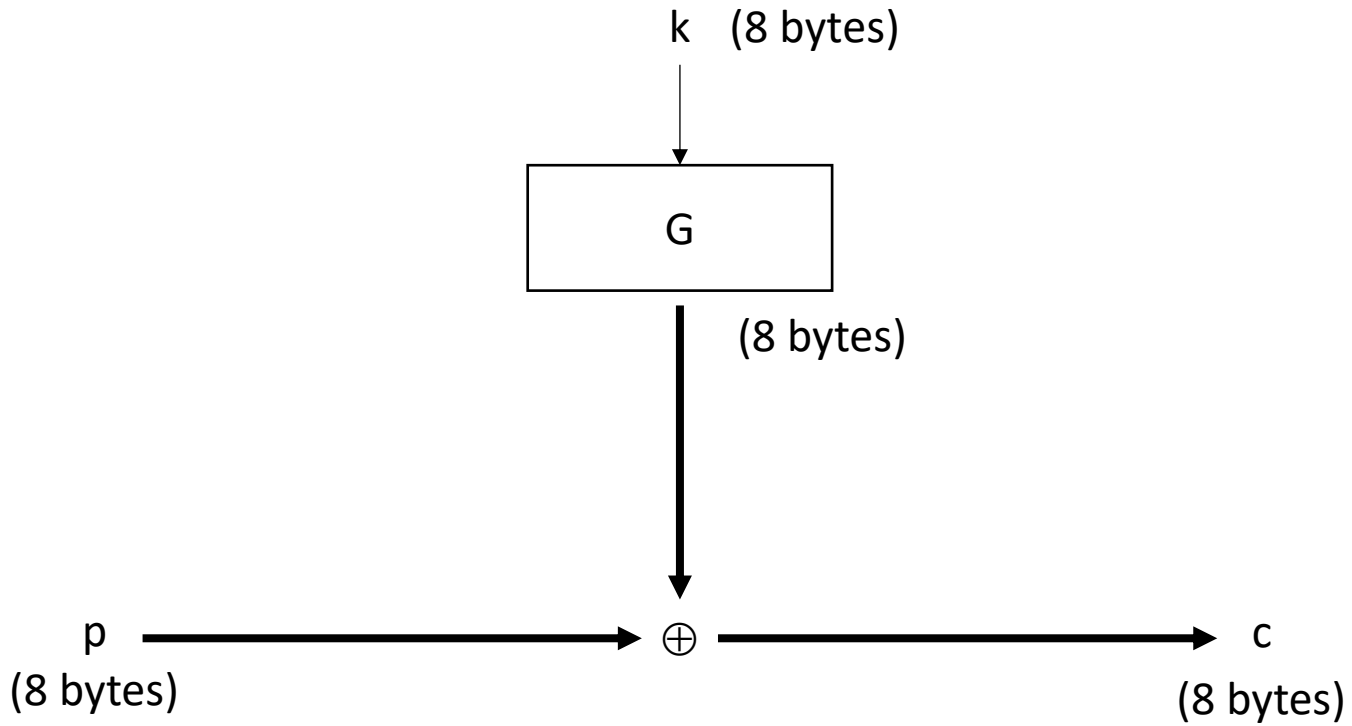
$P_c(p) = ½ + \varepsilon$

$P_c(\bar{p}) = ½ + \varepsilon$

Secure if $\varepsilon < E$

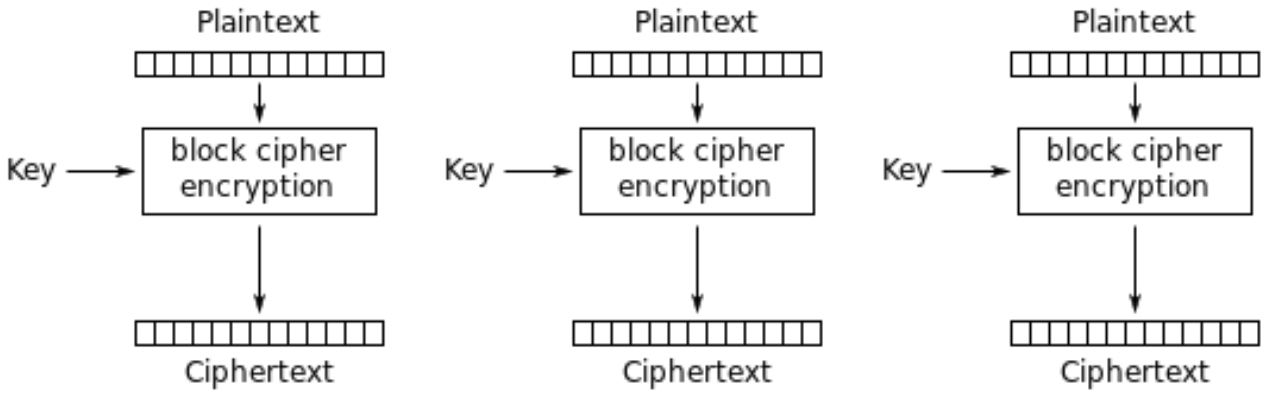


Symmetric Cryptography

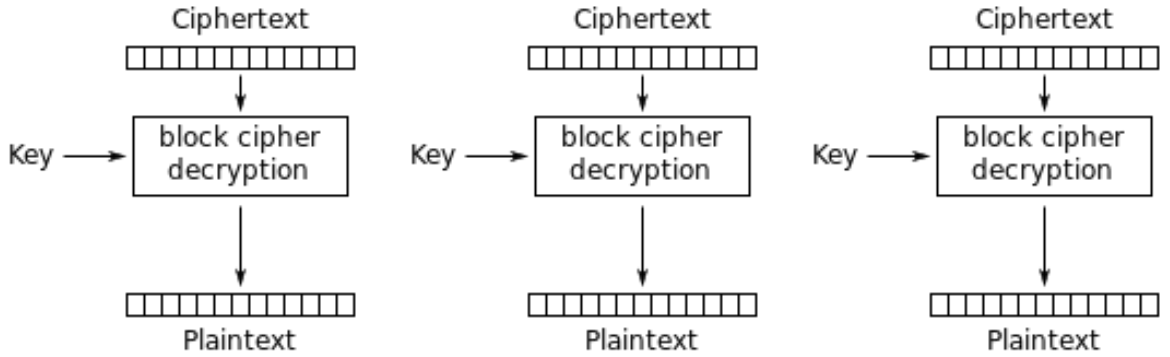


Block ciphers allowed to control $\varepsilon < E$

Symmetric Cryptography (cipher modes)

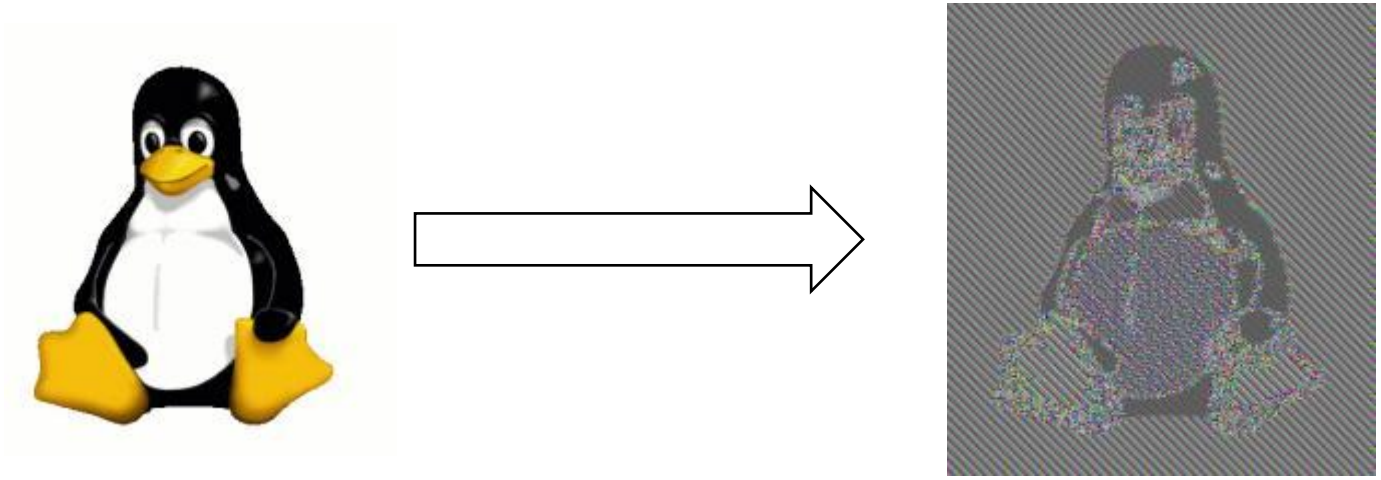


Electronic Codebook (ECB) mode encryption

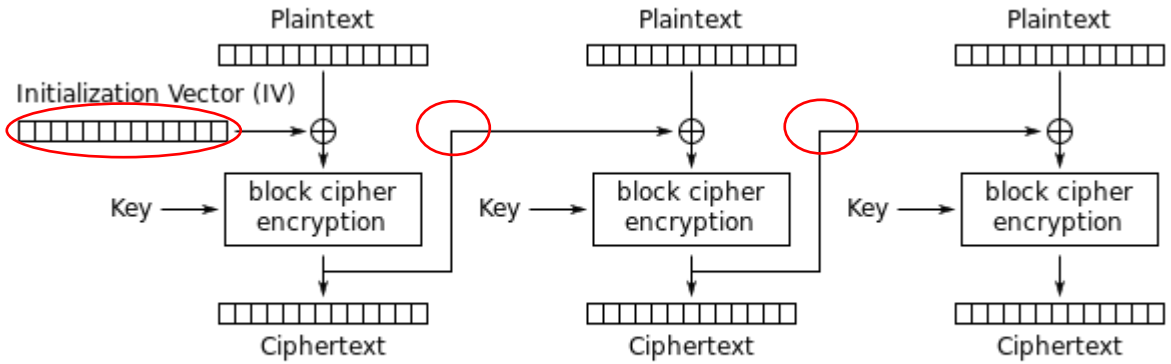


Electronic Codebook (ECB) mode decryption

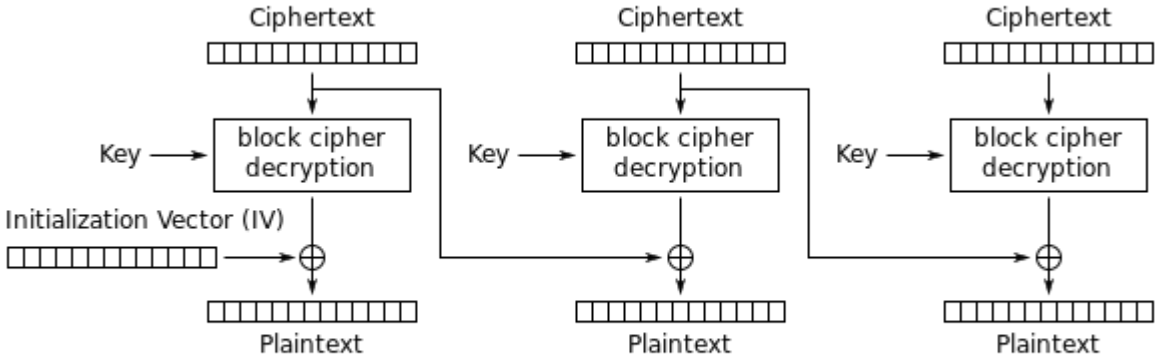
Symmetric Cryptography (ECB)



Symmetric Cryptography (CBC)

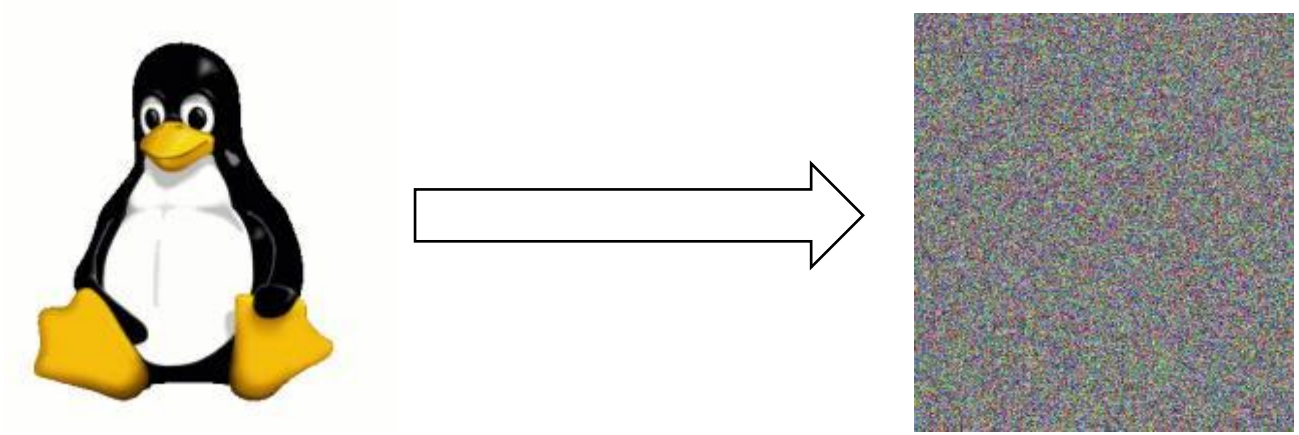


Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Symmetric Cryptography (CBC)



Why Padding (in AES)?

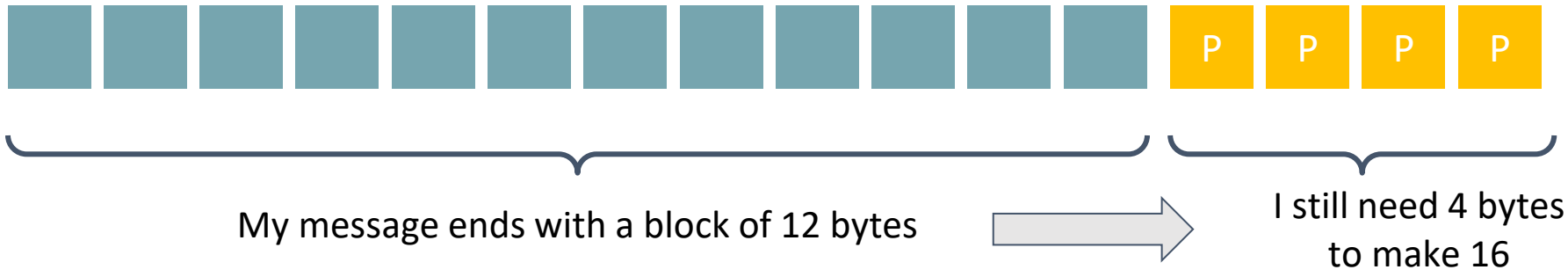
- Blocks need to have a well-known size to be ciphered
- All blocks must be full for the operation to be “defined”
- AES has 128 bit = 16 byte blocks

But what if the message is not a multiple of 128 bits?

Padding: Simple Example

My message ends with a block of 12 bytes

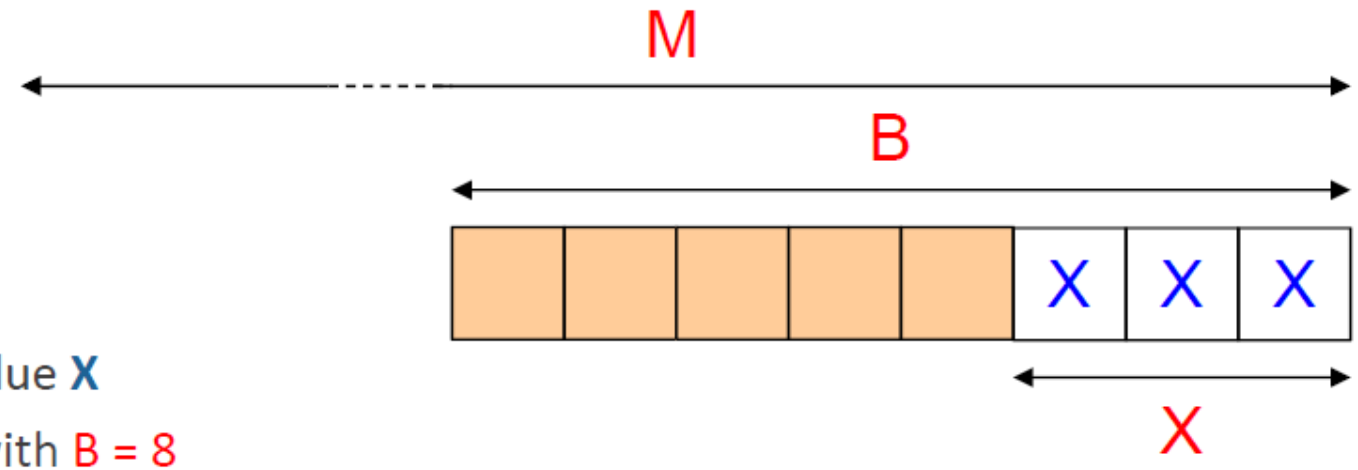
Padding: Simple Example

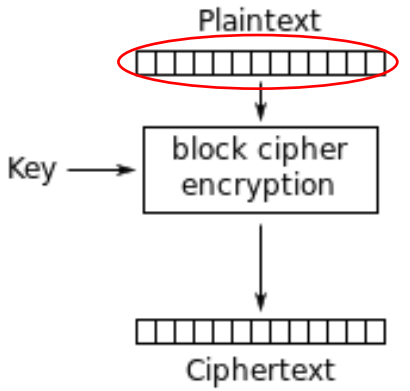


But keep in mind, padding needs to be identifiable!

Symmetric Cryptography (Padding)

- Padding
  - Of last block, identifiable
  - PKCS #7
    - **X = B – (M mod B)**
    - **X** extra bytes, with the value **X**
  - PKCS #5: Equal to PKCS #7 with B = 8
- Different processing for the last block
  - Adds complexity





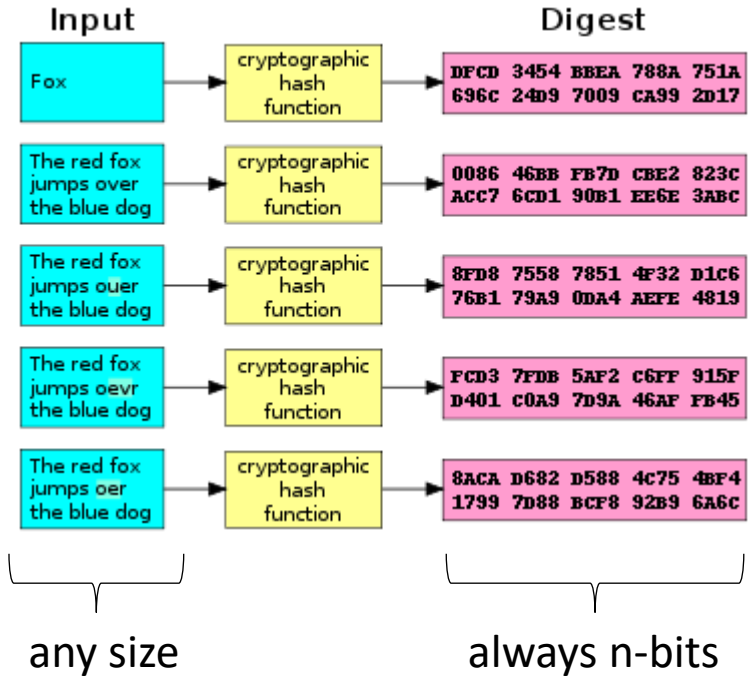
How to derive Passwords?

- We need keys of a given size (e.g., 16 bytes)

- **But,** common practice is to accept arbitrary length passwords.

How can we achieve that?

Hint: Use a Digest Function (aka “Hash”)



> Truncate the output to the desired length (in bytes).

(see PBDKF2 for a better approach!)

Practical Guide

✓ Three fundamental topics:

- Symmetric encryption
- Symmetric Padding
- Key Derivation Functions.

➢ Alternative to C: use a python cryptography library

- cryptography.io module
- “hazardous materials” documentation

Encrypting / Decrypting (Python)

```
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes


cipher = Cipher(algorithms.AES(key), modes.CBC(iv))

pt  = b"a secret message"
encryptor = cipher.encryptor()
ct = encryptor.update(pt) + encryptor.finalize()

decryptor = cipher.decryptor()
dt = decryptor.update(ct) + decryptor.finalize()
```

Padding

```
from cryptography.hazmat.primitives import padding


padder = padding.PKCS7(128).padder()
padded_data = padder.update(b"text")
padded_data += padder.finalize()


unpadder = padding.PKCS7(128).unpadder()
data     = unpadder.update(padded_data)
original = data + unpadder.finalize()
```

Password-Based Key Derivation Function 2 (PBKDF2)

```
import os
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC


salt = os.urandom(16)
kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    length=32,
    salt=salt,
    iterations=480000,
)
key = kdf.derive(b"my password")
```

Any questions?