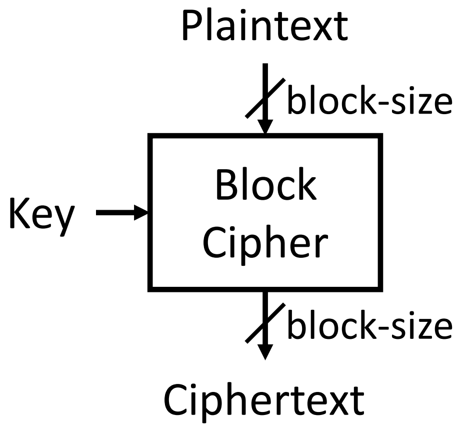
Block Cipher

Elements of Applied Data Security

Alex Marchioni

Block Ciphers

Unlike Stream Ciphers that encrypt one bit at a time, Block Ciphers encrypt a **block of text**. For example, AES encrypts 128 bit blocks.



Since a block cipher is suitable only for the encryption of a single block under a fixed key, a multitude of **modes of operation** have been designed to allow their repeated use in a secure way.

Moreover, block ciphers may also feature as **building blocks** in other cryptographic protocols, such as universal hash functions and pseudo-random number generators.

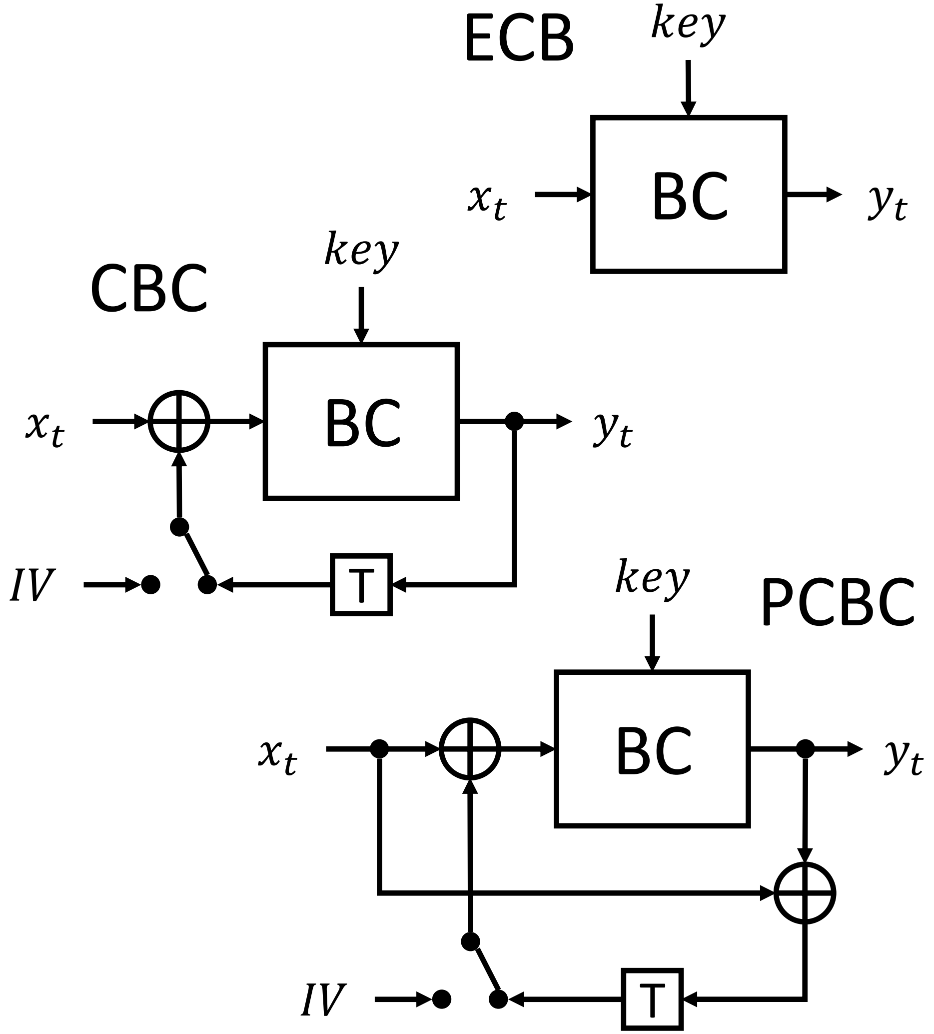
Block Ciphers

Most block cipher algorithms are obtained by iterating an invertible transformation. Each iteration is called **round** and the repeated transformation is known as **round function**.

Usually, the round function takes different **round keys**, which are derived by expanding the original key.

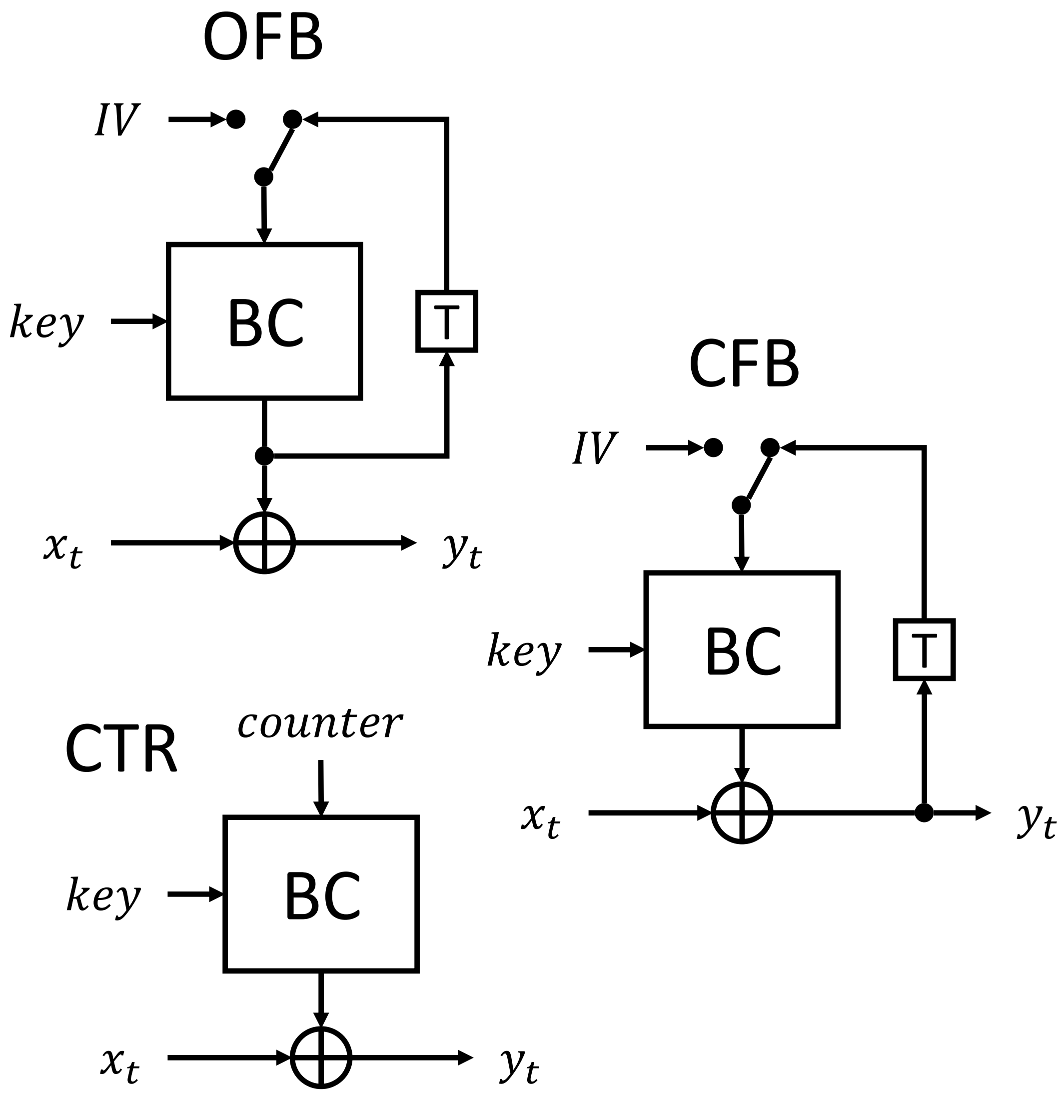
Modes of Operation

- **Electronic codebook (ECB)**: the simplest of the encryption modes where the message is divided into blocks, and each block is encrypted separately.
- **Cipher block chaining (CBC)**: each block of plaintext is XORed with the previous ciphertext block before being encrypted.
- **Propagating cipher block chaining (PCBC)**: each block of plaintext is XORed with both the previous plaintext block and the previous ciphertext block before being encrypted.



Modes of Operation

- **Output feedback (OFB)**: it generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.
- **Cipher feedback (CFB)**: similar to OFB, but to generate the new keystream block, it employs the previous ciphertext instead of the previous keystream block .
- **Counter (CTR)**: the keystream block is generated from the value of a counter that is incremented at each new block.



Advanced Encryption Standard (AES)

AES is a specification for the symmetric-key encryption established by the NIST in 2001 and then adopted by the U.S. government.

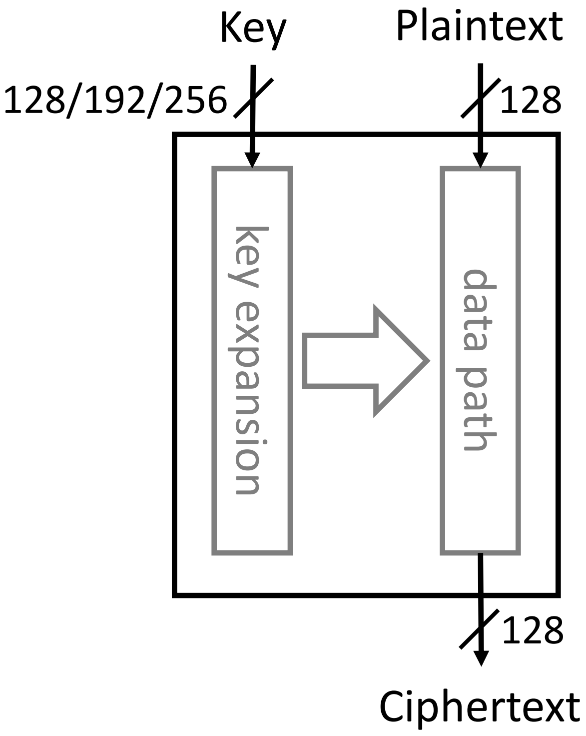
The standard comprises three block ciphers from a larger collection originally published as **Rijndael.** Each of these ciphers has a 128-bit block size, with key sizes of 128, 192 and 256 bits.

[1] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.

[2] Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.)

The Rijndael Block Cipher

Rijndael is designed to resist against all known attacks and to be fast and compact when implemented in most platforms.



Rijndael is composed of a **key expansion** block and a **data path** that can be viewed as an iterated block cipher, where each iteration is called **round**. The number of rounds depends on the block (for AES fixed to 128bit) and key length.

key length	128	192	256
# rounds N_r	10	12	14

Python Packages for Cryptography

Pycryptodome: self-contained Python package of low-level cryptographic primitives. It is a fork of PyCrypto that has been enhanced to add more implementations and fixes to the original library.

PyNaCl: Python binding to libsodium, which is a fork of the Networking and Cryptography library. These libraries have a stated goal of improving usability, security and speed.

Cryptography: cryptography is a package which provides cryptographic recipes and primitives to Python developers. It includes both high level recipes and low level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions.

Pycryptodome

PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It is organized in sub-packages dedicated to solving a specific class of problems:

- `Crypto.Cipher`: Modules for protecting **confidentiality** that is, for encrypting and decrypting data (example: AES).
- `Crypto.Signature`: Modules for assuring **authenticity**, that is, for creating and verifying digital signatures of messages (example: PKCS#1)
- `Crypto.Hash`: Modules for creating cryptographic **digests** (example: SHA-256).
- `Crypto.PublicKey`: Modules for generating, exporting or importing public keys (example: RSA or ECC).

`Crypto.Cipher` subpackage

The base API of a cipher is fairly simple:

- You instantiate a cipher object by calling the `new()` function from the relevant cipher module. The first parameter is always the cryptographic **key**. You can (and sometimes must) pass additional cipher- or mode-specific parameters such as a nonce or a mode of operation.

```
from Cryptodome.Cipher import AES

key = b'0123456701234567'
cipher = AES.new(key, AES.MODE_ECB)
```

`Crypto.Cipher` subpackage

- For encrypting data, you call the `encrypt()` method of the cipher object with the plaintext. The method returns the piece of ciphertext. For most algorithms, you may call encrypt() multiple times (i.e. once for each piece of plaintext).
- For decrypting data, you call the `decrypt()` method of the cipher object with the ciphertext. The method returns the piece of plaintext.

```
plaintextA = b'this is a secret'
ciphertext = cipher.encrypt(plaintextA) # b'\x8dk\x84\xcey*h\xach\x9b\xd0[\xb6pR\x95'
plaintextB = cipher.decrypt(ciphertext) # b'this is a secret'
```

Task

1. Get familiarity with MonteCarlo Simulation by estimating the value of π.
2. Apply MonteCarlo Simulation to AES with the aim of observing the effect of diffusion and confusion.

Task 1: Monte Carlo Simulations

Monte Carlo Simulations

A Monte Carlo simulation is a statistical technique that allows for the modelling of complex situations where many random variables are involved. It uses the process of repeated random sampling to model stochastic systems and determine the odds for a variety of outcomes.

The idea comes from the Law of Large Numbers that states: «*the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed.*»

Roughly speaking, If you do not know some parameters of your sistem, you can make several trials and then take the average.

Estimating π with MCS

A classical example is the estimation of π.

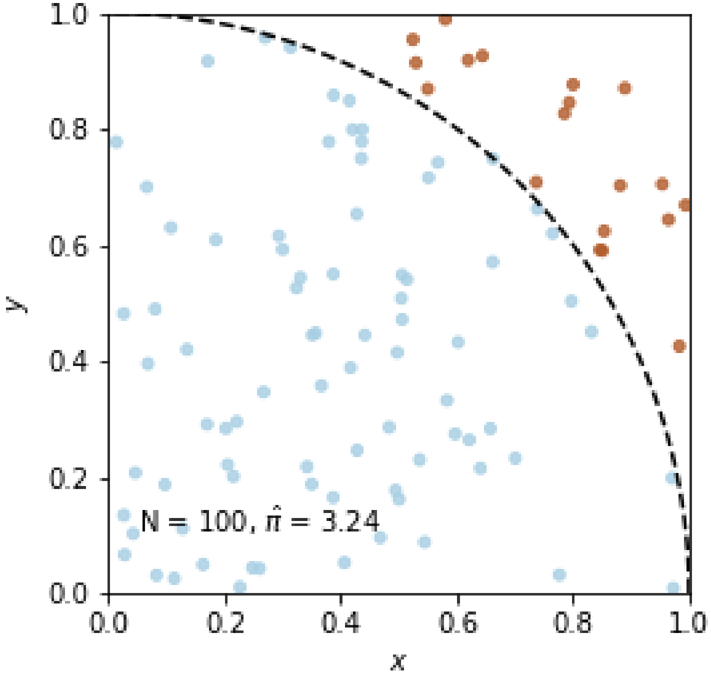
We know that πr^2 corresponds to the area of a circle with radius r, so we can estimate the value of π as four times the ratio between the area of a square and the inscribed circle.

$$4\frac{A_{\circ}}{A_{\blacksquare}} = 4\frac{\pi r^2}{(2r)^2} = 4\frac{\pi}{4} = \pi$$

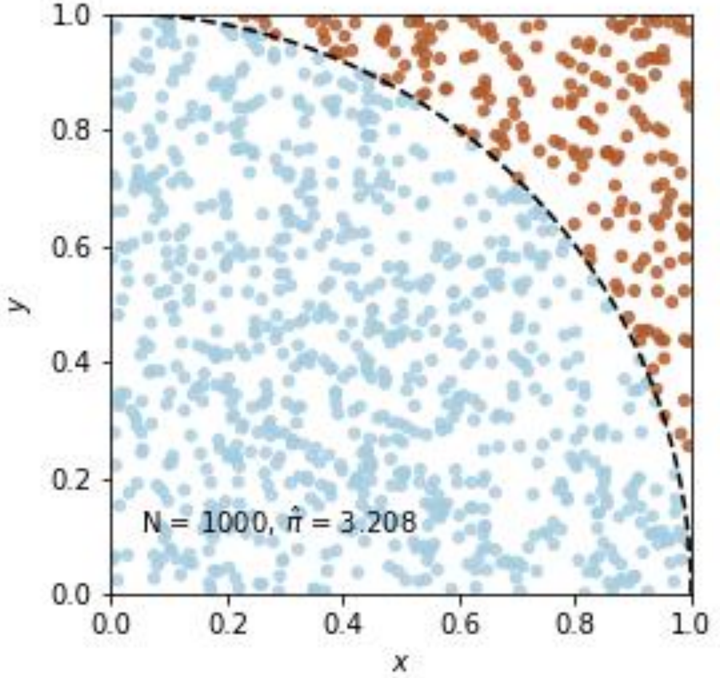
The ratio between the two areas can be estimate by drawing N coordinates (x, y) as instances of uniform random variables $x, y \in U(0,1)$ and count how many fall into the circle with respect to the total.

Estimating π with MCS

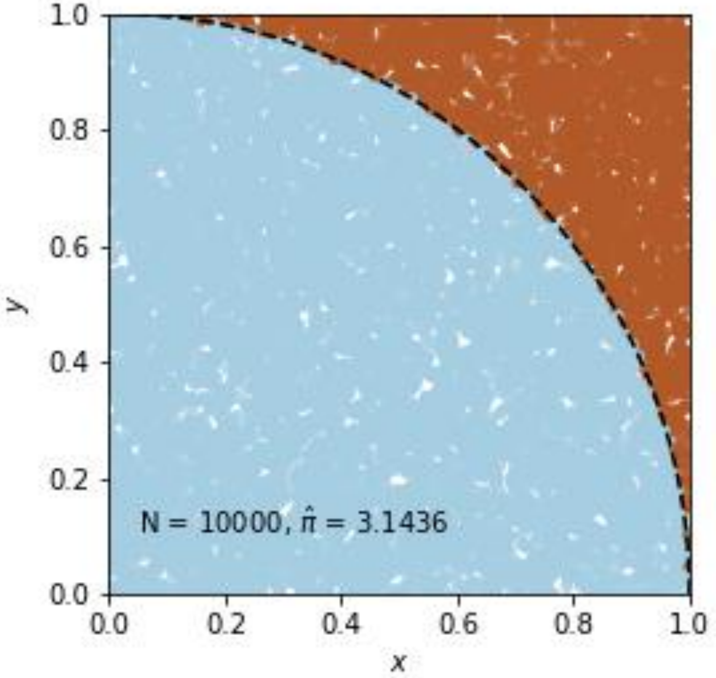
$N = 100, \hat{\pi} = 3.24$



$N = 1000, \hat{\pi} = 3.208$



$N = 10000, \hat{\pi} = 3.1436$



Task 1

- Estimate the value of π by means of MonteCarlo Simulations as explained in the previous slides.
- Discuss the choice of the number of trials to run to be quite confident that the estimate is accurate enough for your application.
  - How does the estimate converge to the actual value?
  - If I did not know the actual value of π, how can one decide when to stop?

Task 2: AES Diffusion and Confusion

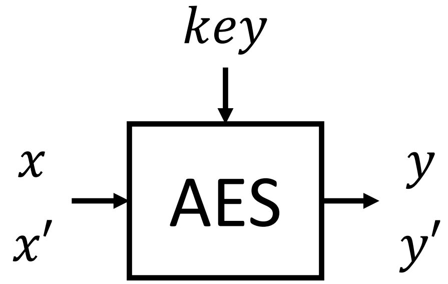
Diffusion with MCS

Diffusion: *if we change a single bit of the plaintext, then (statistically) half of the bits in the ciphertext should change*.

We want to test qualitatively if AES provides diffusion.

To do so, we can randomly draw a plaintext and a key, change a bit in the plaintext and then observe how this change affects the ciphertext.

Plaintext x' differs only 1bit from plaintext x.

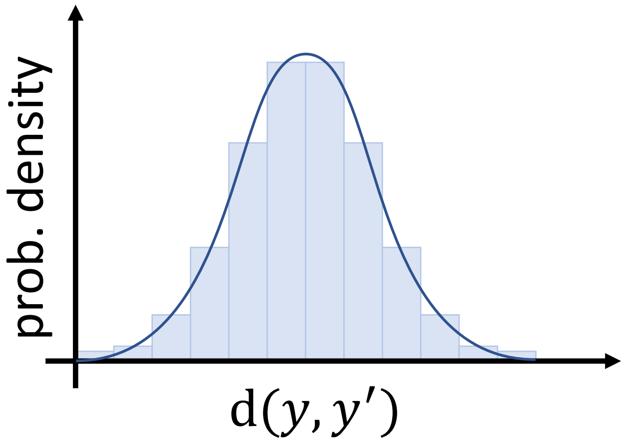


How many bits does the ciphertext y' differ from the ciphertext y?

Diffusion with MCS

We can define $\mathrm{d}(y, y')$ the distance between y and y' as the number of bit y' differs from y. This distance is the quantity we want to estimate by means of Monte Carlo simulation and more than its average we are interested in its **distribution**.

We do not expect to find that $\mathrm{d}(y, y')$ is always $n/2$ (where n is the number of bits in the ciphertext) but we expect to find a distribution that is concentrated around $n/2$.

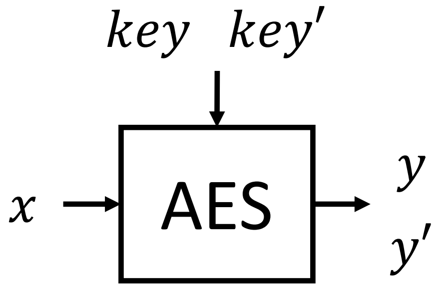


Confusion with MCS

Confusion: *the relationship between the key and the ciphertext must be very complicated and impossible (hard) to invert*

At bit level, it means that each bit of the ciphertext depends on several parts of the key. As a result, if a single bit is changed in the key, many bits in the ciphertext change.

Again, we want to assess qualitatively confusion of AES qualitatively.



The keys key and key' differ only by 1bit.
What is the distribution of the distance between the ciphertext y and y' obtained by encripting the same plaintext x with the two different keys?

Task 2

- What kind of Mode of Operation should I use to perform this to best characterize AES block cipher?
- Discuss about the trade-off between number of trials and accuracy.
- How the distribution changes by varying the length of the key?
- What are the differences between diffusion and confusion?