

# Comparison of symmetric ciphers in OpenSSL

## HW1 - CNS Sapienza

Edoardo Puglisi 1649359

04/11/2019

### Contents

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>Ciphers</b>	<b>2</b>
<b>3</b>	<b>Operating modes</b>	<b>3</b>
<b>4</b>	<b>Code</b>	<b>4</b>
<b>5</b>	<b>Results</b>	<b>6</b>

## 1 Overview

The following experiment consists of comparing encryption/decryption speeds of multiple symmetric ciphers with different operating modes. It has the purpose to find correlations between file size and efficiency of different ciphers.

## 2 Ciphers

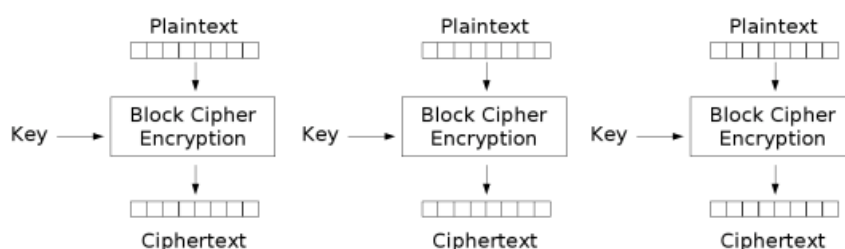
Here the list of ciphers used for the experiment with brief descriptions.

- **DES:** uses a 56-bits key and for this is highly insecure, but it was highly influential for modern cryptography. DES takes a fixed-length plaintext and transforms it into a ciphertext of same size using a series of complicated operations also with the usage of key to customize the transformations and make them more secure.
- **AES:** standing for Advanced Encryption Standard, AES is an algorithm which became standard for the U.S. National Institute of Standards and Technology (NIST). Plaintext is represented as a block matrix  $A$ . Steps:
  1. Key expansion - the cipher key is divided in multiple round key. One round-key block per round + 1
  2. First step - AddRoundKey = bitwise XOR with byte of state and round-key block
  3. Rounds -
    - Substitution =  $A[i, j]$  is swapped with  $A[i, j]^{-1}$
    - RowShift = row  $i$  is shifted of  $i-1$  positions.
    - Multiplication = multiply every column with an invertible polynomial  $03x^3 + 01x^2 + 01x + 02 \pmod{x^4 + 1}$
    - AddRoundKey
  4. Final round - same steps as before but without Multiplication
- **Camellia:** algorithm developed by Mitsubishi Electric and NTT of Japan. It is comparable to AES in term of security and processing abilities. It was designed to be suitable for both software and hardware implementations. Composed by 18/24 rounds, every six rounds it apply a FL function (or its inverse) i.e. a logical transformation layer.
- **Aria:** developed by a group of researchers of South Korea, is based on AES interface. This algorithm uses a substitution-permutation network derived from AES.

### 3 Operating modes

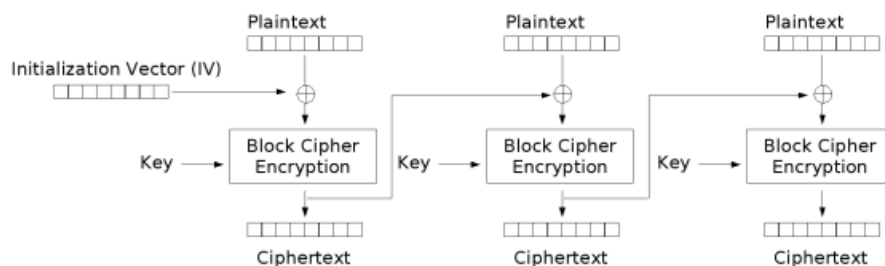
Short description of operating modes used by all ciphers.

- **ECB:** Electronic Code Block, the simplest operating mode. Plaintext managed 64bit at time, each block encrypted with same key. If needed padding bits are added to reach block size. Process can be parallelized and there isn't errors propagation. Data integrity not guaranteed due to the fact that attackers can swap blocks positions and insecure against bruteforce attacks.



Electronic Codebook (ECB) mode encryption

- **CBC:** the basic idea is that a plaintext block if repeated will produce different cipher blocks. To each plaintext block is applied the XOR operand with the previous ciphertext block.



Cipher Block Chaining (CBC) mode encryption

## 4 Code

Code description.

```
#!/bin/bash

PASSWORD="passwordmolto Sicura"
OUTPUT="result.hw1-1649359.csv"
CICLI=10
type nul > $OUTPUT

echo ++++++ HOMEWORK 1 ++++++

for cipher in aes-256-cbc aes-256-ecb camellia-256-
cbc camellia-256-ecb aria-256-cbc aria-256-ecb
des-cbc des-ecb
do
    echo $cipher >> $OUTPUT
    echo +++++ CIPHER $cipher +++++
    for size in 100K 1MB 10MB 100MB
    do
        echo +++++ FILE SIZE $size +++++
        echo ===== ENCRYPTION =====

        runtime=0
        for i in {1..$CICLI}
        do
            start=$(date +%s%N)
            openssl $cipher -a -salt -pbkdf2 -in "
                $size.file" -out "$size.enc" -pass
                pass:$PASSWORD
            runtime="$(( $runtime + $(date +%s%N) -
                $start ))"
        done

        enc_runtime_scaled=$(bc <<< "scale=3;
            $runtime/$CICLI/1000000")
        echo ENC_SPEED $enc_runtime_scaled

        echo ===== DECRYPTION =====

        runtime=0
        for i in {1..$CICLI}
```

```

do
    start=$(date +%s%N)
    openssl $cipher -d -a -salt -pbkdf2 -in
        "$size.enc" -out "$size.new.file" -
        pass pass:$PASSWORD
    runtime="$((($runtime+($(date +%s%N)-
        $start)))"

done

dec_runtime_scaled=$(bc <<< "scale=3;
    $runtime/$CICLI/1000000")
echo DEC_SPEED $dec_runtime_scaled
echo $size >> $OUTPUT
echo ENC";"$enc_runtime_scaled >> $OUTPUT
echo DEC";"$dec_runtime_scaled >> $OUTPUT

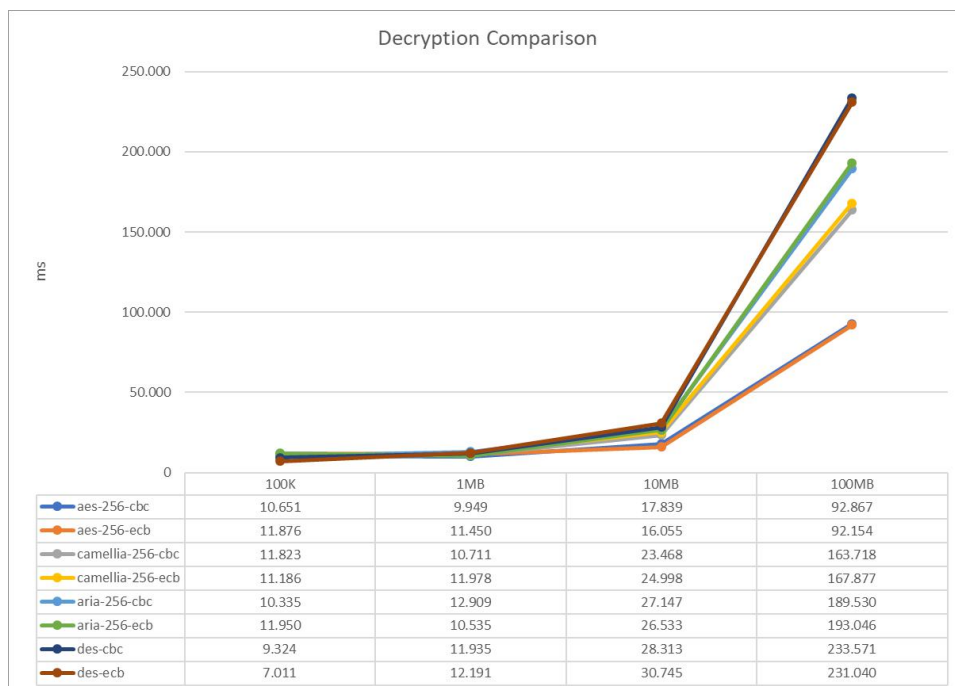
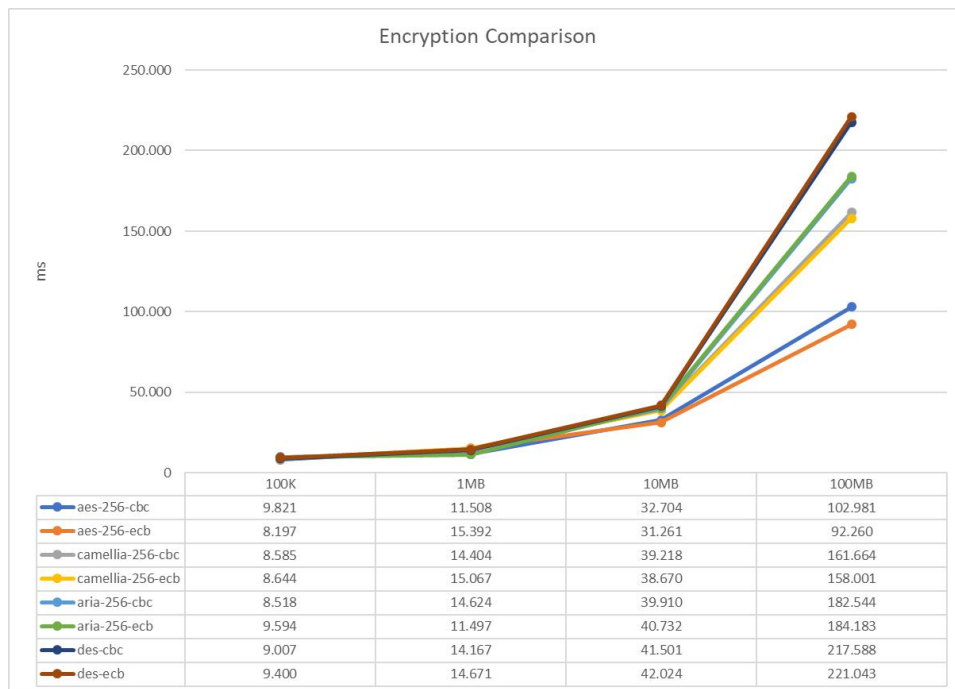
done
done

```

The script used for the experiment consists of 10 iterations of encryption and 10 iterations of decryption for every cipher on files of different sizes. In addition for every cipher two different operating mode have been used (ECB and CBC). Encryption and decryption times are written on a csv file used to draw the graphs.

## 5 Results

Here the resulting graphs.



We can see that AES algorithm is the best in terms of efficiency in both encryption and decryption. The difference between operating modes doesn't influence much computing time and this means that CBC, besides being safer than ECB, doesn't effect encryption and decryption efficiency. Above 10MB, file size starts effecting the computation, showing an huge increment of computing time in all ciphers.