Comparison of symmetric ciphers in OpenSSL
HW1 - CNS Sapienza

Edoardo Puglisi 1649359

04/11/2019

Contents

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 chipers 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 fized-length plaintext and transforms it into a ciphertex of same size using a serie of complicated operations also with the usage of key to costumize 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 $03x^3 + 01x^2 + 01x + 02(mod(x^4 + 1))$
     - AddRoundKey
  4. Final round - same steps as before but without Multiplication

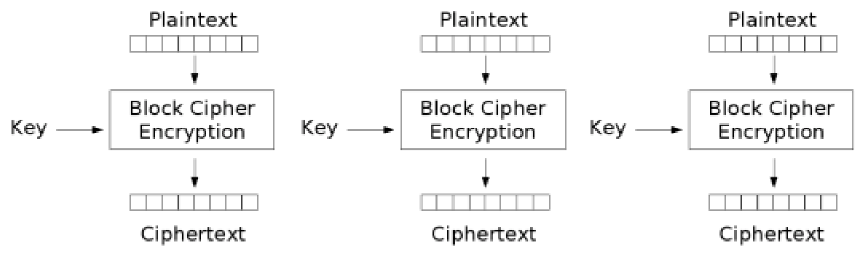
- **Camellia:** algorithm developed by Mitsubishi Electric and NNT of Japan. It is comparable to AES in term of security and processing abilities. It was designed to be suitable for both software and hardware implementations. Composed by 18/24 rounds, every six rounds it apply a FL function (or its inverse) 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

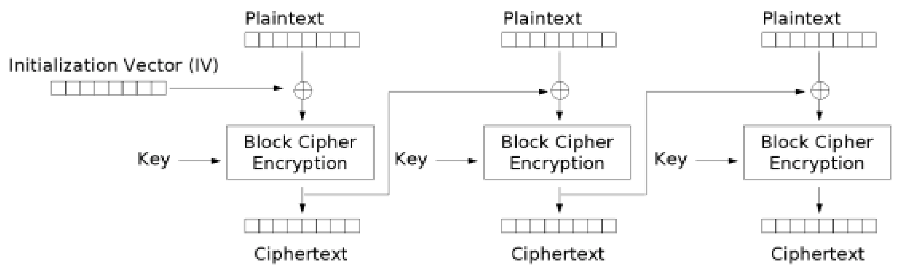
Short description of operating modes used by all ciphers.

- **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

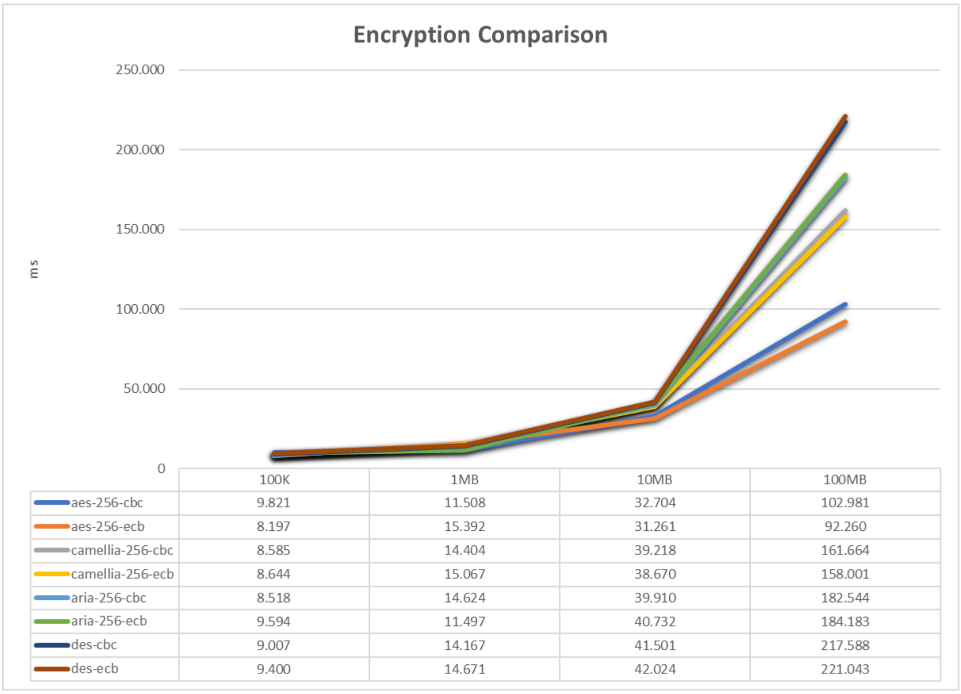
Code description.

```
#!/bin/bash

PASSWORD="passwordmoltosicura"
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 "
               $size-hw1-1649359.file" -out "$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
```

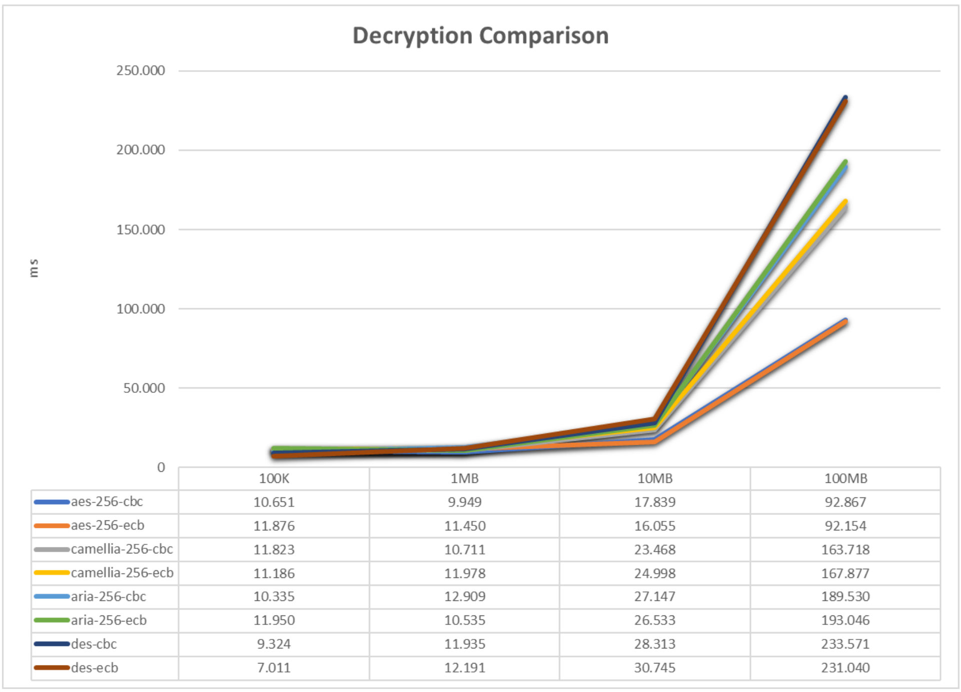
The script used for the experiment consists of 10 iterations of encryption and 10 iterations of decryption for every cipher on files of different sizes. In addition for every cipher two different operating mode have been used (ECB and CBC). Encryption and decryption times are written on a csv file used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

5 Results

Here the resulting graphs.

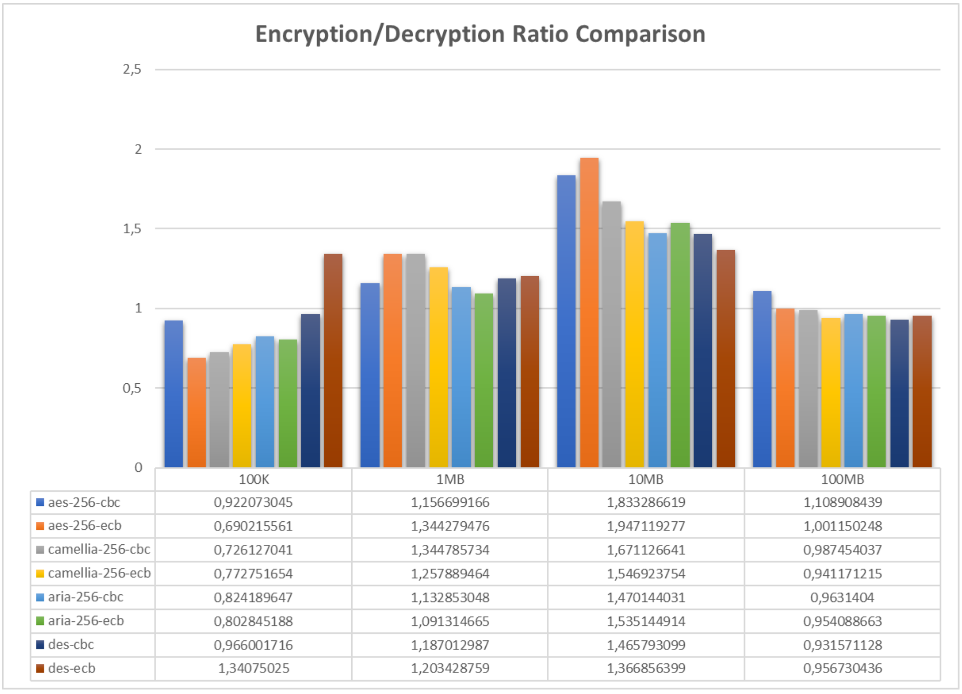


	100K	1MB	10MB	100MB
aes-256-cbc	9.821	11.508	32.704	102.981
aes-256-ecb	8.197	15.392	31.261	92.260
camellia-256-cbc	8.585	14.404	39.218	161.664
camellia-256-ecb	8.644	15.067	38.670	158.001
aria-256-cbc	8.518	14.624	39.910	182.544
aria-256-ecb	9.594	11.497	40.732	184.183
des-cbc	9.007	14.167	41.501	217.588
des-ecb	9.400	14.671	42.024	221.043



	100K	1MB	10MB	100MB
aes-256-cbc	10.651	9.949	17.839	92.867
aes-256-ecb	11.876	11.450	16.055	92.154
camellia-256-cbc	11.823	10.711	23.468	163.718
camellia-256-ecb	11.186	11.978	24.998	167.877
aria-256-cbc	10.335	12.909	27.147	189.530
aria-256-ecb	11.950	10.535	26.533	193.046
des-cbc	9.324	11.935	28.313	233.571
des-ecb	7.011	12.191	30.745	231.040

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.



	100K	1MB	10MB	100MB
🟦 aes-256-cbc	0,922073045	1,156699166	1,833286619	1,108908439
🟧 aes-256-ecb	0,690215561	1,344279476	1,947119277	1,001150248
⬜ camellia-256-cbc	0,726127041	1,344785734	1,671126641	0,987454037
🟨 camellia-256-ecb	0,772751654	1,257889464	1,546923754	0,941171215
🟦 aria-256-cbc	0,824189647	1,132853048	1,470144031	0,9631404
🟩 aria-256-ecb	0,802845188	1,091314665	1,535144914	0,954088663
🟦 des-cbc	0,966001716	1,187012987	1,465793099	0,931571128
🟫 des-ecb	1,34075025	1,203428759	1,366856399	0,956730436

About the encryption/decryption ratio we can see that the main differences can be found on small size file where operating mode influences a lot the ratio: for example AES CBC has a ratio much higher than AES EBC on 100K files. We have an huge increment of ratio in all ciphers and operating modes for 10MB files that decreases again on 100MB files where becomes almost equal for all ciphers.