Comparison of symmetric ciphers in OpenSSL
HW1 - CNS Sapienza

Lombardo Andrea 1893440

04/11/2019

1 Introduction

Nowadays cyberattacks are the fastest growing crime, we want to know how to protect own information. The main target is to have knowledge on how hiding information to the external world. How we've analyzed during the lesson there're different safer way to exchange information between two people using a encryption and decryption function, which masks the plaintext into a ciphertext. An encryption system scrambles sensitive data using mathematical calculations to turn data into code. The original data can only be revealed with the correct key, allowing it to remain secure from everyone but the authorized parties. Now we're focus only to a block cipher mode that is a particular section of cipher mode with the combination of different algorithm. The main goal of this report is to analyse time performance about combination of different symmetric ciphers with different block cipher modes using **OpenSSL library**. For this reason I decide to prove performance of different configurations like des,aes,blowfish that have different key length, different inner mechanics, different popularity and usability with different block cipher modes like(ecb,cbc,ctr,ofb).Performance metrics are evaluate according to timeval structure of C language. Time differences will change because in the computation with object file we'll give in input four different constraints: cipher mode, cipher algorithm with the particular size of the key used that identify that particular configuration of the algorithm and the file, which is important only for the size.

2 Recap

2.1 AES

AES is fastest cipher used nowadays both ways: hardware and software. It accepts a 128 or 192 or 256-bit key for this reason I decide to test all of them. AES is a symmetric key cipher. This means the same secret key is used for both encryption and decryption, and both the sender and receiver of the data need a copy of the key. I also notice that it has a quick key setup time and small memory requirements . Then we should take care about the size of the block size, because larger is the block size higher is the probability to take computation advantage. Although it depends on which is the cipher between (ofb,ctr,ecb,cbc) linked with the algorithm cipher. It sometimes causes encrypted messages to be slightly longer than they otherwise would be.
The advantage of symmetric systems like AES is their speed. Because a symmetric key algorithm requires less computational power than an asymmetric one. The smallest length of 128 is large enough for brute force attacks due to the computation power today. It require a very high number of attends. AES is resistant to all possible attack tested nowadays. Rijndaesl the best

known of AES uses only operation that are available in every hardware, and this bring the code to be very compact.

2.2 DES

DES is efficient to implement in hardware but very slow if implemented in softwarr. It based on the Feistel network on 16 round and it accepts only a size block of 64 and 56-bit key,which is the real security problem of this symmetric ciphers, due to the fact that nowadays is much too short its size key. It doesn't contain any intrinsic problem in the structure but the real problem is the size key. The way to use it nowadays is through tre-des.

2.3 Blowfish

Blowfish is a algorithm of symmetric block cipher that was designed for software usage. It has a 64bit block size and a variable key length between 32 and 448 bits but I decide to test only 64size key. It has a Feistel network structure with 16 rounds. It uses a lot of memory and has a relatively long key setup time, but it's faster than DES. It is one of the fastest block algorithms in circulation, except when happened the event of changing the key. Each new key requires a processing time near to the 4 kB text encoding, which is a lot compared to other block encryption algorithm.

2.4 Openssl

OpenSSL is a robust open-source software library which is supported by the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is used for applications with the aim of having secure communications over computer networks against eavesdropping or needing to identify the party at the other end . It is also manage to support different cryptographic algorithm and cipher mode.

3 Developing

I decide to implement this homework with the two C files:

Create-binary-file.c : the following file is used to develop a program which helps me to create a binary file with a random content. After computing the compilation with gcc I should give in input the pair name and size I would like of the file.
Implementation:
./Create-binary-file <file-name><size-of-file>

Encrypt.c the following file is used to develop a program which helps me to run a encryption and decryption of the same files giving in input.

The main goal is to analyse performance of pair(Cipher, Operative Modes). After computing the compilation with gcc I should give in input the following data.

Implementation:
./Encrypt <file-input> <file-ouput> <block-cipher-modes> <cryptographic-algorithm><size-key>

Command	Explanation
<fileinput>	It refers to input file which I want to transform
<fileouput>	It refers to output file that is created
<block-cipher-modes>	It could be "cbc","ofb","ctr","ecb"
<cryptographic algorithm>	It could be "aes","des","blowfish"
<size-key >	It could be "128", "192", "256", default("64")

One important issue that I should to mention is that in the file I composed a string with string manipulation for achieving to composed my string with the following instruction:
Example of encryption:
openssl enc -aes-256-cbc -e -in $< file - input >$ **-out** $file - output >$
Example of decryption:
openssl enc -aes-256-cbc -d -in $< file - input >$ **-out** $file - output >$.
The following string is giving in input to the sycall "**system(string)**". I decide to use this approach for running faster without writing

3.1 Time

The <sys/time.h> header defines the timeval structure that includes at least the following members:

time_t tv_sec seconds

suseconds_t tv_usec microseconds

So after defining two variables tv1 and tv2 of timeval structure, I decide to set them according **gettimeoftheday(*timeval,NULL)** which obtains the current time, expressed as seconds and microseconds since 00:0 and set the variable in input with the new time evaluated. So according this script I define a speed function and according the content of the string we refer to encryption or decryption function. The classification of the function depends on the flag that appear in the string "-d" or "-e".

gettimeofday(&tv1,NULL);

system(string);

gettimeofday(&tv2,NULL);

total_time = ((double)(tv2.tv_usec -tv1.tv_usec) /1000000 +(double)(tv2.tv_sec - tv1.t

The total_time variable evaluated is in **microsec**.
After evaluating the two total time I manage to evaluate the speed ratio:

The speed ratio= $\frac{average-speed-of-encryption}{average-speed-of-dencryption}$

4 Result

For the analysis of result I decide to split the collected data according the cryptographic algorithm applied: "aes","des","blowfish". After defining tables where I report all the collected data about encryption,decryption and ratio speed; I decide to represent them into a pareto graph build with excel tool. Therefore all this data prove that AES is the best cipher algorithm in comparison with others especially when the size of the file is higher and des and blowfish need more time. In general what we 've seen that until in the range from 10KB to 100MB size file, not included, the different configurations works more or less in a constant way about performance (decrytion and encryption) and in that range the worst in encryption is **aes-ofb-192** and in decryption is **aes-ecb-128**. When the size of the file is higher this two approach are not the worst but the best how we can see in the picture behind. When the size of the file increase the worst are configuration with des and b. Where des is the worst for timing purpose.

4.1 AES

In AES I should notice the fact that the encryption algorithm changes by the key size which could be:

Speed Encryption	Speed Decryption	Ratio Speed	File-size	block-cipher-modes	key-size
4.359443	2.599262	1.677185	10kB	cbc	256
4.171970	1.301231	3.206172	100kB	cbc	256
3.715735	1.050350	3.537616	1MB	cbc	256
4.659236	1.391298	3.348841	10MB	cbc	256
5.042218	2.073365	2.431901	100MB	cbc	256
30.756625	41.350805	0.743797	1GB	cbc	256
3.689510	1.386400	2.661216	10kB	ofb	256
3.326405	1.186489	2.803570	100kB	ofb	256
2.941245	1.101776	2.669549	1MB	ofb	256
3.004966	1.373692	2.187511	10MB	ofb	256
5.217646	2.632225	1.982219	100MB	ofb	256
32.394853	36.779752	0.880780	1GB	ofb	256
2.750986	0.985772	2.790692	10kB	ctr	256
2.809482	1.2313574	2.281615	100kB	ctr	256
2.962209	1.145039	2.586994	1MB	ctr	256
6.443095	1.803577	3.572398	10MB	ctr	256
5.294896	3.681520	1.438236	100MB	ctr	256
32.403780	37.154787	0.872129	1GB	ctr	256
3.473117	1.174671	2.956672	10kB	ecb	256
3.235725	1.064082	3.040861	100kB	ecb	256
2.895189	1.338829	2.162479	1MB	ecb	256
3.053618	1.376410	2.218538	10MB	ecb	256
4.605876	3.974040	1.158991	100MB	ecb	256
28.840649	85.850834	0.335939	1GB	ecb	256
2.664263	1.260979	2.112853	10kB	cbc	192
3.487846	1.220820	2.856970	100kB	cbc	192
2.674239	1.496751	1.786696	1MB	cbc	192
2.721208	1.233728	2.205679	10MB	cbc	192
4.297932	4.041473	1.063457	100MB	cbc	192
34.207591	41.702918	0.820269	1GB	cbc	192
4.386269	2.248492	1.950760	10kB	ofb	192
3.764350	0.993167	3.790249	100kB	ofb	192
2.542937	1.173368	2.167212	1MB	ofb	192
2.816929	1.340247	2.101798	10MB	ofb	192
4.324710	2.417315	1.789055	100MB	ofb	192
30.155195	71.393832	0.422378	1GB	ofb	192
2.263101	1.423235	1.590111	10kB	ctr	192
3.976254	1.078373	3.687271	100kB	ctr	192
3.101877	1.254282	2.473030	1MB	ctr	192
3.008374	1.293588	2.325604	10MB	ctr	192
2.299011	1.037749	2.215383	100MB	ctr	192
27.897818	30.173756	0.924572	1GB	ctr	192

Always with AES algorithm, continuing the same table of the previous page:

Speed Encryption	Speed Decryption	Ratio Speed	File-size	block-cipher-modes	key-size
2.642226	1.184654	2.230378	10kB	ecb	192
2.918350	1.306981	2.232894	100kB	ecb	192
5.047215	1.228987	4.106809	1MB	ecb	192
3.866786	2.035690	1.899496	10MB	ecb	192
4.730603	3.893356	1.215045	100MB	ecb	192
29.964810	81.293902	0.368598	1GB	ecb	192
2.545410	1.114235	2.284446	10kB	cbc	128
3.797344	1.130814	3.358062	100kB	cbc	128
3.490262	1.733501	2.013418	1MB	cbc	128
2.550369	3.829071	0.666054	10MB	cbc	128
5.259847	4.407423	1.193406	100MB	cbc	128
31.896840	35.456416	0.899607	1GB	cbc	128
6.991025	1.030888	6.781556	10kB	ofb	128
7.179308	1.661498	4.320985	100kB	ofb	128
2.396045	1.059333	2.261843	1MB	ofb	128
2.389374	1.302880	1.833917	10MB	ofb	128
3.980628	2.263071	1.758950	100MB	ofb	128
32.980584	37.530310	0.878772	1GB	ofb	128
2.531761	1.212310	2.088378	10kB	ctr	128
2.496855	1.620326	1.540958	100kB	ctr	128
2.587879	1.200809	2.155113	1MB	ctr	128
4.141702	1.295550	3.196868	10MB	ctr	128
4.651084	2.392923	1.943683	100MB	ctr	128
30.861134	48.041700	0.642382	1GB	ctr	128
2.499045	1.341384	1.863035	10kB	ecb	128
3.201166	1.301443	2.459705	100kB	ecb	128
3.801868	1.158342	3.282164	1MB	ecb	128
3.022085	1.244395	2.428558	10MB	ecb	128
5.032240	3.670736	1.370908	100MB	ecb	128
34.269277	34.326665	0.998328	1GB	ecb	128

4.2 DES

In DES I decide to suppress CTR cipher block in contrast to AES algorithm, and in this approach I don't need to specify the key-size because it's 64 by default.

Speed Encryption	Speed Decryption	Ratio Speed	File-size	block-cipher-modes
2.954931	1.077989	2.741151	10kB	cbc
2.639069	1.338793	1.971230	100kB	cbc
3.170957	1.494152	2.122245	1MB	cbc
2.561888	1.409673	1.817363	10MB	cbc
11.463205	4.768831	2.403777	100MB	cbc
46.305483	76.893940	0.602199	1GB	cbc
4.026351	1.121402	3.590462	10kB	ofb
2.235728	1.384352	1.615000	100kB	ofb
3.110097	1.217782	2.553903	1MB	ofb
3.582814	2.698605	1.327654	10MB	ofb
9.607530	7.949134	1.208626	100MB	ofb
54.048726	91.494808	0.590730	1GB	ofb
3.911184	1.013949	3.857377	10kB	ecb
3.049417	1.124306	2.712266	100kB	ecb
3.085795	1.449298	2.129165	1MB	ecb
5.790610	2.028355	2.854831	10MB	ecb
8.416116	4.756507	1.769390	100MB	ecb
52.840533	90.616351	0.583124	1GB	ecb

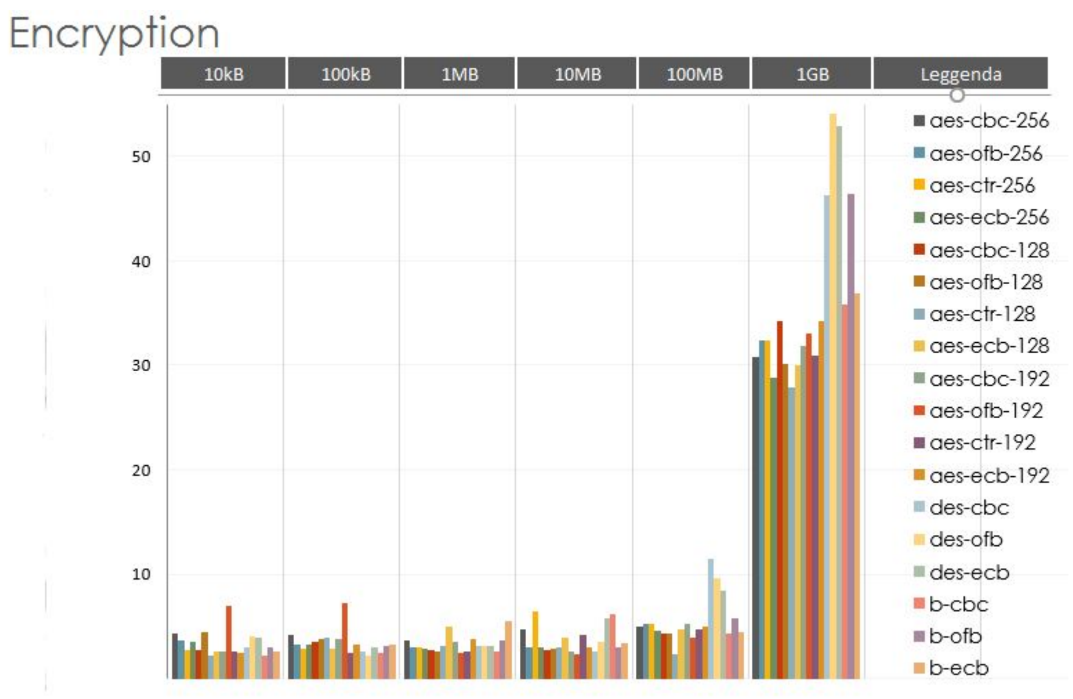
4.3 Blowfish

In Blowfish I decide to suppress CTR cipher block in contrast to AES algorithm, and in this approach I don't need to specify the key-size because it's 64 by default.

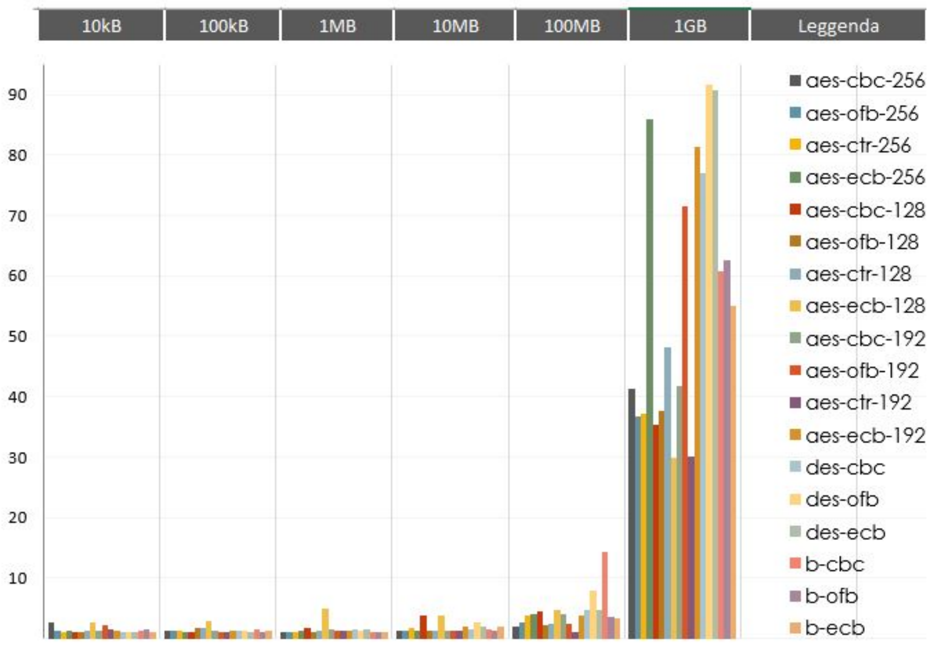
Speed Encryption	Speed Decryption	Ratio Speed	File-size	block-cipher-modes
2.209927	1.253396	1.763151	10kB	cbc
2.488629	1.496010	1.663511	100kB	cbc
2.583968	1.092927	2.364264	1MB	cbc
6.202695	1.586198	3.910417	10MB	cbc
4.291672	14.418871	0.297643	100MB	cbc
35.994797	60.678888	0.593201	1GB	cbc
2.943310	1.561219	1.885264	10kB	ofb
3.143248	1.154229	2.723245	100kB	ofb
3.710955	1.129895	3.284336	1MB	ofb
3.003093	1.362103	2.204747	10MB	ofb
5.833491	3.637203	1.603840	100MB	ofb
46.418584	62.481357	0.742919	1GB	ofb
2.540340	1.164139	2.182162	10kB	ecb
3.240409	1.170075	2.769403	100kB	ecb
5.579889	1.053722	5.295409	1MB	ecb
3.414078	1.984801	1.720111	10MB	ecb
4.443070	3.405212	1.304785	100MB	ecb
36.944939	55.069305	0.670881	1GB	ecb

4.4 Comparison between configuration

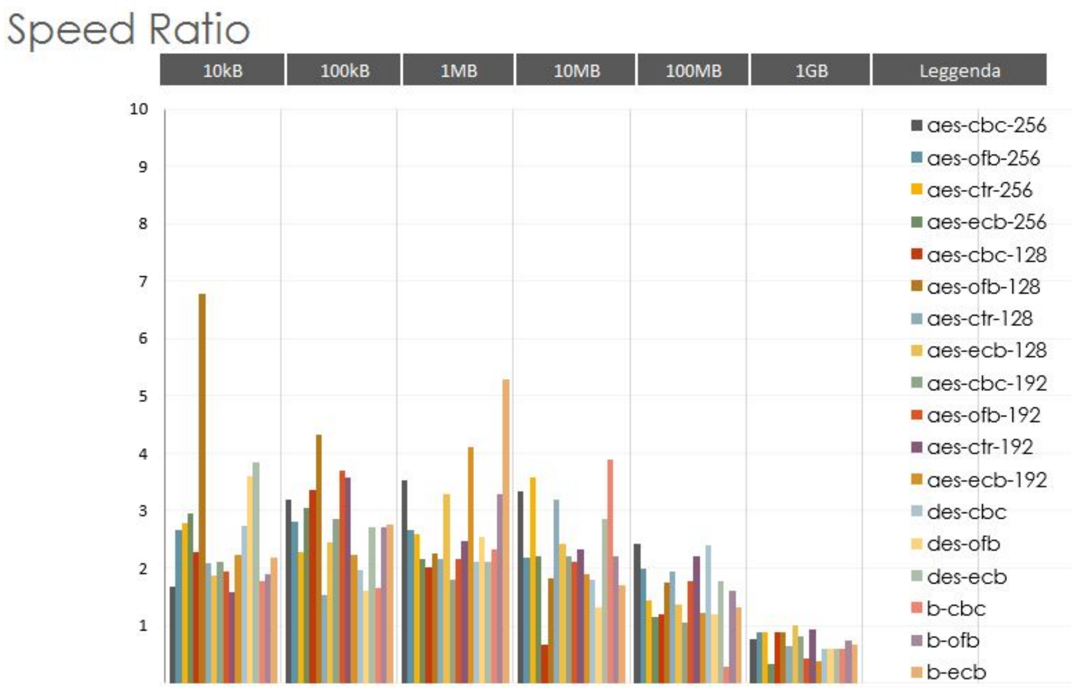
We can see in the picture behind the trend of each configuration analysed and the comparison between them. We should take notices about the range on the legend at left side of the picture. From this picture where we show Encryption speed with a range from 1 to 50.



We can see in the picture behind the trend of each configuration analysed and the comparison between them. We should take notices about the range on the legend at left side of the picture. From this picture where we show Decryption speed with a range from 1 to 90.



We can see in the picture behind the trend of each configuration analysed and the comparison between them. We should take notices about the range on the legend at left side of the picture. From this picture where we show Ratio speed with a range from 1 to 10.



5 Conclusion

In conclusion how we can see we have proof that the system works faster if the $< File - size >$ is small, also in comparison between the different configurations of (aes,blowfish,des) we prefer using aes not only for security reason but also for a matter of time spent. Observing the different operating modes used we can notice: in the tests performed with the ECB, CTR mode, then these were fastest than others. Another important issue is that I want to test the performance also from 10kB also if it's not required and I notice that performances are similar to 100kB. However we don't forget that the range from one speed test to the next one change in fact in general the average Decryption is higher than average Encryption but both continue increasing with the grown of the size file.

References

[1] https://pubs.opengroup.org/onlinepubs/7908799/xsh/gettimeofday.html

[2] https://pubs.opengroup.org/onlinepubs/7908799/xsh/systime.h.html

[3] https://codescracker.com/c/program/c-program-encrypt-file.htm

[4] https://www.techiedelight.com/des-implementation-c/

[5] https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_chapter/libc_32.html#SEC662

[6] http://www.hoozi.com/posts/advanced-encryption-standard-aes-implementation-in-cc-

[7] https://www.example-code.com/C/crypt2_aes.asp

[8] https://www.codingalpha.com/file-handling-code-to-encrypt-and-decrypt-c-program-t

[9] http://www.trytoprogram.com/c-examples/c-program-to-encrypt-and-decrypt-string/

[10] https://tls.mbed.org/des-source-code

[11] http://cprogrampracticals.blogspot.com/2017/02/key-generation-in-simplified-des.

[12] https://cs.slu.edu/ d̃ferry/courses/csci3500/labs/crypt_example.c

[13] http://www.regatta.cs.msu.su/doc/usr/share/man/info/ru_RU/a_doc_lib/libs/commtrfi

[14] https://wiki.openssl.org/index.php/Command_Line_Utilities#Commands

[15] https://gnu.huihoo.org/glibc-2.2.3/html_chapter/libc_32.html

[16] https://it.wikipedia.org/wiki/Modalit%C3%A0_di_funzionamento_dei_cifrari_a_blocch

[17] https://stackoverflow.com/questions/3382147/how-to-encrypt-a-text-file-using-c

[18] http://www.lepillole.it/fondamenti/slides/InputOutput.pdf

[19] https://www.dreamincode.net/forums/topic/290062-encrypting-and-decrypting-text-

[20] https://linux.die.net/man/3/ecb_crypt

[21] https://wiki.openssl.org/index.php/Command_Line_Utilities#Commands

[22] http://www.di-srv.unisa.it/professori/paodar/EC2018/slides/EC-DES-AES.pdf

[23] http://wwwcdf.pd.infn.it/AppuntiLinux/introduzione_a_openssl.htm

[24] https://it.wikipedia.org/wiki/Data_Encryption_Standard

[25] https://blog.1password.com/guess-why-were-moving-to-256-bit-aes-keys/

[26] https://crypto.stackexchange.com/questions/50442/is-des-faster-than-aes-on-softw

[27] https://www.solarwindsmsp.com/blog/aes-256-encryption-algorithm