D7.1

Report on AES implementation with speed and side channel immunity improvements

Project title:	iDev40: Integrated Development 4.0
Project start date:	1st of May 2018
Duration:	36 months

Deliverable type:	Report
Activity and Work package contributing to the deliverable:	Work Package 7 / Objective 7.1 & Objective 1.1 / Task 7.1.1
Due date:	30.09.2020
Actual submission date:	30.09.2020

Responsible lead organisation:	UPB
Lead Author:	Sorin Chițu
Dissemination level:	Public
Reviewer:	Germar Schneider (IFD)
Revision Date:	27.09.2020
Document Reference	D7.1/D76

This project has received funding from the ECSEL Joint Undertaking under grant agreement No 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme. It is co-funded by the consortium members, grants from Austria, Germany, Belgium, Italy, Spain and Romania.

Authors (all contributors to the Deliverable)

Organization	Name	Chapters
UPB	Sorin Chițu	All
UPB	Ciprian Daniel Vasile	Chapters 3, 4
UPB	Ionuț Daniel Trămândan,	Chapter 5
UPB	Tudor Ioan Honceriu	Chapter 5

All figures are from the authors, if not stated otherwise.

Publishable Executive Summary

Trust is one of the most desired goals in information security being involved in any of information security attributes like confidentiality, integrity and availability of information. But trust must be maintained by permanently improving data security measures in accordance with threats evolution.

Regarding encryption algorithms, trust relates to powerful forms of encryption, but also to secure implementation and self-protected cryptographic hardware modules against any side channel attacks. The paper considers the choosing of optimal security solutions according to functional necessities of the beneficiary IoT network and security risk environment.

On the other hand, side channel attacks refers to any kind of information which can be obtained by measuring indirect effects of the encryption process, like current consumption, execution time or electromagnetic waves, which can be used by an attacker.

This report presents the results of the analysis of AES implementation on different hardware platforms, considering the functional performance, associated key management and mitigation on side channel attack, highlighting an innovative tamperproof solution. Also, it provides useful solutions to developers to implement efficient cryptographic modules, hardware or software, and to design high performance tamperproof envelopes. Based on the mentioned tamperproof system, consisting of a conductive mesh and some dedicated electronic circuits, it was developed a True Random Number Generator which was used for cryptographic secret key expansion, thus increasing the overall security. The main advantage of this type of True Random Number Generator resides in not being influential through external factors without detection, due to its common usage in tamperproof systems. Not in the last, the presented principle could be the basis of a full automatized design and production line for cryptographic modules, each of them with a random spectral power distribution, which represent an insurmountable barrier against an attacker.

Key Words:

Encryption, Key Management, Secure communication, Side channel attack, Tamperproof

Contents

Publishable Executive Summary II

1 Introduction 1

1.1 Overview 1

1.2 Involved Project Partner 2

1.3 Motivation 3

2 Keys Generation and Management for usage in AES 4

2.1 Pseudo and True Random Number Generators 4

2.1.1 Motivation 4

2.1.2 Pseudo Random Number Generators 4

2.1.3 True Random Number Generators 5

2.2 Cryptographic Keys Management 6

2.2.1 Motivation 6

2.2.2 Keys Management Architecture 6

2.3 Expanding the Secret Keys Space by Usage of Conductive Mesh Networks 7

2.3.1 Motivation 7

2.3.2 Results 8

3 Mitigation of Sight Channel Attacks 9

3.1 General Presentation of Side Channel Attacks Types 9

3.2 Tamper proof Solutions 9

3.3 Design of Conductive Mesh Networks Based on the Output of a True Random Number Generator 10

3.3.1 Motivation 10

3.3.2 Results 10

4 Hardware modules 12

4.1 Implementation of cryptographic algorithms 12

4.2 Processors cryptographic implementations: hardware versus software 13

4.3 Software implementations tests in microcontrollers 14

5 Software modules 16

5.1 Overview 16
5.2 The security of the processor 17
5.3 Operating system security 19
5.4 Security breaches 22
5.5 Performance testing for different implementations 23
5.5.1 Motivation 23
5.5.2 Test Method 23
5.5.3 Test environment 24
5.5.4 Observation and discussion 26
5.5.5 Test results 33
5.6 AES implementation 33
5.7 Key Security Concepts 38
5.8 Guidelines for a secured code 38
5.9 Vulnerabilities and countermeasures 40
6 Conclusion 42
7 Appendix 43
8 List of Abbreviations 50
9 References 51

List of Figures

Figure 1: Different approaches of security 1
Figure 2: General architecture of iDev40 IT&C network 2
Figure 3: TRNG based on a CMN and LFSR 5
Figure 4: An efficient implementation of a challenge response mutual authentication protocol 8
Figure 5: Exemplification of TRNG output 10
Figure 6: The route design corresponding to TRNG output 11
Figure 7: Encryption and decryption tests for STM32F769NI (200MHz) microcontroller using the AES256-CBC algorithm 14
Figure 8: Encryption and decryption tests for STM32H743ZI (400MHz) microcontroller using the AES256-CBC algorithm 15
Figure 9: Positive security rings 17
Figure 10: Positive and negative security rings 18
Figure 11: OS modes 22
Figure 12: Hardware comparison 25
Figure 13: Windows 10 distributions 26
Figure 14: Ubuntu distributions 26
Figure 15: Encryption of 400MB file 27
Figure 16: Decryption of 400MB file 27
Figure 17: Encryption of files from 1MB to 500MB 28
Figure 18: Decryption of files from 1MB to 500MB 28
Figure 19: Encrypt/Decrypt using NET CORE 28
Figure 20: Encrypt/Decrypt using Python 29
Figure 21: Encryption of a 400MB file on i5 9600k CPU 29
Figure 22: Decryption of a 400MB file on i5 9600k CPU 30
Figure 23: Encryption C# vs Python on i5 9600k CPU 30
Figure 24: Decryption C# vs Python on i5 9600k CPU 30
Figure 25: Encrypt/Decrypt C# on i5 9600k CPU 31
Figure 26: Encrypt/Decrypt Python on i5 9600k CPU 31

Figure 27: Encrypt/Decrypt files from 1MB to 500MB on i5 9600k CPU using Python on Ubuntu 32
Figure 28: Encrypt Ubuntu vs. Windows on i5 9600k CPU 32
Figure 29: Decrypt Ubuntu vs. Windows on i5 9600k CPU 32
Figure 30: Encrypt Ubuntu vs Windows, 1MB to 500MB file sizes, on i5 9600k CPU 33
Figure 31: Decrypt Ubuntu vs Windows, 1MB to 500MB file sizes, on i5 9600k CPU 33
Figure 32: Symmetric Algorithms hierarchy 35
Figure 33: Cryptographic flow 37

List of Tables

Table 1: The dispersion of amplitude values measured at the output of conductive mesh 8
Table 2: Features of implementations for cryptographic primitives 14
Table 3: AES Instruction Set 34

1 Introduction

1.1 Overview

Information protection involves multiple approaches belonging to different fields, not all of them being technical. In order to defend a value, measures must be taken in the areas of physical security, information assurance, administrative security, communication security, electromagnetic radiation security, cryptographic security, IT security and personnel security. All these domains could be considered as a wall that surrounds the defended value (Figure 1), the general protection level being determined by the least tall segment of the wall.

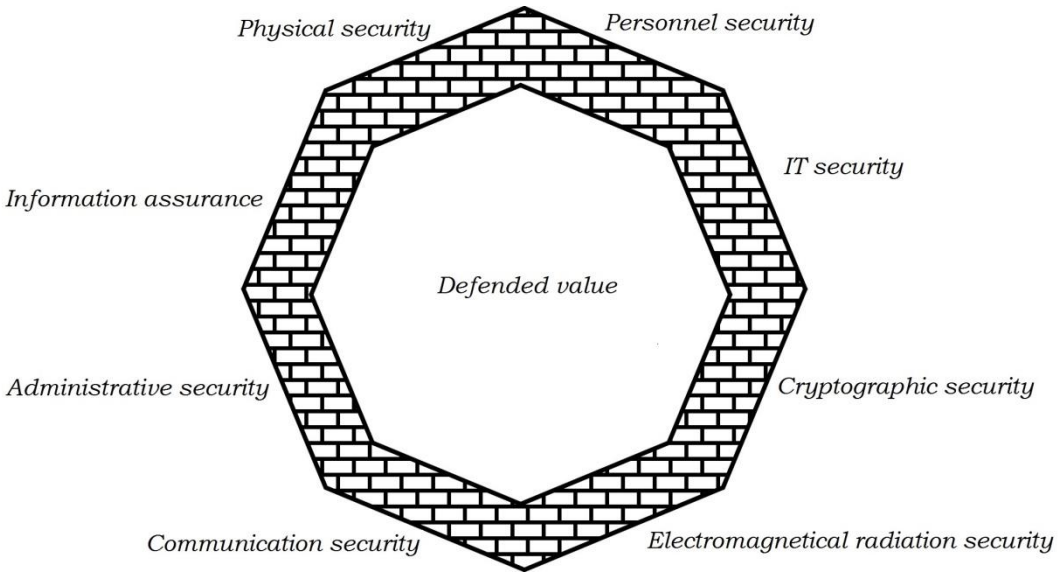

Figure 1: Different approaches of security

In cryptographic segment are included all algorithms, protocols and associated implementations which are dedicated to protect information by cryptographic means. The presented work is focused both on hardware and software implementations of security functions on different platforms.

Due to functional necessities of iDev40 project, the IT&C network terminals are spread in different locations (Figure 2), most of them being placed in production area, but also in partner's sites. This is the reason why there are necessary self-protection measures to be implemented in all the terminals in order to preserve the confidentiality of secret data.

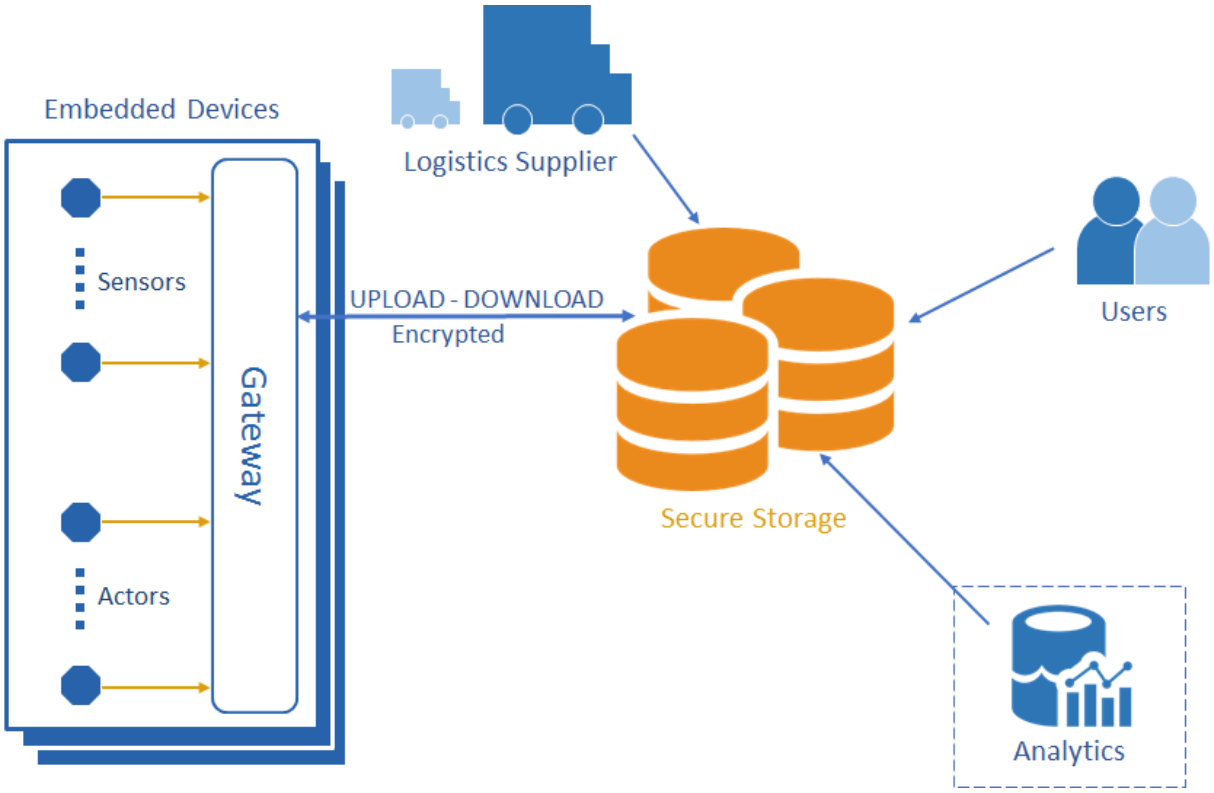

Figure 2: General architecture of iDev40 IT&C network

1.2 Involved Project Partner

This chapter presents background information of UPB and how it is involved in the IT&C security domain.

The University POLITEHNICA of Bucharest (UPB – www.upb.ro) is the oldest and most prestigious engineering school in Romania, with a tradition of 200 years made possible by the efforts of some of the greatest Romanian professors. Its specificity relies in creating knowledge through research and technological innovation, as well as through its implementation by means of education and professional training at a European level.

In iDev40, UPB is participating through "Center for Technological Electronics and Interconnection Techniques" (CETTI, www.cetti.ro) under the umbrella of which are gathered specialists with experience in the areas of interest for this project. Regarding to iDev40 objectives, the main activities of this group, reflected in the research results or projects that has been implemented, are related to cryptology as it follows:

- Research on Pseudo and True Random Number Generators
- Design of authentication protocols
- Design of hardware and software cryptographic modules
- Design of Cryptographic Key Management Systems

- Implementations of Public Key Infrastructure Systems
- Implementations of Digital Signature
- Design of tamperproof mechanisms

1.3 Motivation

According to iDev40 Objective 1.1 – *Data Management and storage*, UPB is involved in development of the IT security concepts. The results of UPB activities in this domain have been materialized up to now in:

- the contribution to D1.2 – *Report on security concepts*,
- two papers that have been presented at the IEEE 26th International Symposium for Design and Technology in electronic Packaging (SIITME) 2020 (*"Key Expansion in Cryptographic Systems"* and *"Algorithm to Design Conductive Mesh for Tamperproof Envelope")* and
- the present report.

In this deliverable report, UPB presents the research results and some practical implementations of an Advanced Encryption Standard, an innovative tamper proof mechanism which include a conductive mesh network, but also a microcontroller signal analysis implementation and an innovative secret key expansion solution based on previous mentioned tamper proof solution.

In the present report, the research was focused on core implementation of cryptographic algorithms on different hardware or software platforms, and on associated dedicated tamper proof solutions in network terminals. In the next deliverable of this task, *D7.2 - Report on security improvements in hardware and software implementation in a production area*, the research will be focused on the cryptographic security on the system layer, covering all technical measures that have to be taken in order to protect data in an IoT network, as it is generally presented in Figure 2.

2 Keys Generation and Management for usage in AES

2.1 Pseudo and True Random Number Generators

2.1.1 Motivation

According to Kerckhoffs's principle, one of the most important attribute of a cryptographic system is the confidentiality of the secret keys. This is the reason why their generation has to be done by using trusted devices, based on physical non-deterministic principles, and not Pseudorandom Random Number Generators, which have a predictable behaviour even if their outputs have very good statistical distributions.

Random numbers are used not only as secret keys (in symmetrical cryptography), but also in in authentication (e.g. challenge-response protocol) or in any other situations where unpredictable numbers are necessary.

The objective of this part of work is to provide a technical implementable solution to generate random numbers in all cryptographic devices which are part of iDev40 IT&C network in order to perform strong cryptographic protocols. Also, these generators are proposed to be involved in secret keys exchange procedure, having as effect the extension period of lifetime of the keys and, implicit, reducing security associated risks.

2.1.2 Pseudo Random Number Generators

One of the most used types of PRNGs is the Linear – Feedback Shift Register. Due to its simplicity of implementation and high throughput (one bit at each clock cycle) is very common in cryptographic applications. It provides at its output a bit stream with very good guaranteed statistical properties, but deterministic.

In principle, it is based on a shift register and some XOR gates which provide input from a linear function of bits which represent the previous state of the register. Choosing of feedback is very important in order to obtain a maximum period of evolution of the states of the register. So, by choosing a prime polynomial to implement register's feedback it will be obtained the maximum period of evolution, otherwise, depend on register's initialisation, the register's evolution states will belongs to a loop, bigger or smaller, associated to the primitive polynomials which divide the original polynomial. The maximum LFSR's period that can be obtained in these conditions cover all 2^n - 1 possible states, where n is the number of bits of the register. All bits "0" represent a blocking state, so measures have to be taken to avoid such an initialisation.

Even if LFSR's output has very good statistical characteristics, risks could appear in condition of inadequate usage. So, if a LFSR is initialised every time with the same value (seed), its evolution will be identical, and an attacker will have an easier job. Due to this reason, the seed is better to be generated randomly, in order to cover uniform the 2^n - 1 space states.

A bit stream sequence could be characterised by its linear complexity, meaning the LFSR's feedback minimal polynomial that will generate the same sequence. In this approach, a new

class of PRNGs could be developed, Gollmann cascade, with better linear complexity values. This principle is based on a sum of LFSRs, with different polynomial feedbacks, each clock register signal being conditioned by the state of previous register. The introduced nonlinearity increases the complexity of generated bit streams, the work of an attacker being significantly harder.

2.1.3 True Random Number Generators

Depending on security necessities and estimated threats, but also on the adequate hardware resources, TRNGs can be implemented based on different physical phenomena, like: thermal noise, ring oscillators jitter, radioactive decay etc. Each of them has particularities which make them proper or not to be used in different cryptographic systems or in different environments. An essential characteristic of a TRNG, beside its statistic output distribution, it's the immunity on external environmental factors, which otherwise can be exploited by an attacker in order to influence one or more of its parameters. For example, in some cases, an unbalanced distribution of "1" and "0" can considerable ease the work of a cryptanalyst, reducing the necessary time to success an attack.

According to general objectives of iDev40, continuing the work whose results were already presented in UPB contribution to D1.2 – "Report on Security Concepts", an innovative TRNG was developed and tested. So, based on tamperproof mechanism for securing electronic modules, presented in Chapter 2.2 of D1.2, the noise present on the output of the CMN, unwanted according to that objective, were now used to generate random numbers.

For this purpose, in experimental research work, a 200kB bit stream was formed by capturing and concatenation last significant bits of every sample. A set of statistical tests were used according to SP 800-22 (NIST, n.d.) and the results, presented in Appendix 1, were far to be proper in order to be used as it is, as random numbers, in cryptographic applications.

But, even in these conditions, the range of its output qualified the bit stream to be used as seed for a LFSR, so the ensemble of those two modules provides a very good solution for generation of unpredictable random numbers. As a supplementary measure, according to presented aspects in Chapter 2.1.2, a testing module will be inserted between TRNG based on CMN and LFSR, in the scope of avoiding initialisation with "0"s in case of hardware damaging of TRNG. The module block scheme is presented in Figure 3.

Figure 3: TRNG based on a CMN and LFSR

2.2 Cryptographic Keys Management

2.2.1 Motivation

According to the proposed IT&C architecture of iDev40 network, different types of equipment and devices are involved. Due to the advantages of asymmetric cryptography, a combined key management system is proper to be used. Therefore, a Diffie – Hellman based key exchange method will be used in order to securely transport a secret key which previously has been generated by a unique TRNG based device.

Therefore, the secret keys are generated periodically, or whenever necessary, by a unique TRNG device being under the control of crypto administrator. All these keys are validated by passing through various dedicated statistical tests, which guarantee that they are suitable for cryptographic use. Also, due to sensitivity of key generation and management system, before delivery to each terminal, each key transmission is individually prepared such that only the legitimate beneficiary by an authentication process could receive and use the secret key.

2.2.2 Keys Management Architecture

According to iDev40 objective to secure the template IT&C infrastructure presented in Figure 1.2, a complex keys management system has to be implemented, taking into account the risks due to several factors, including:

- Interconnection of many LANs over insecure communications channels (Internet);
- Interconnection with LANs belongs to other organisations, like providers or customers
- Need to know principle
- Need to share principle
- Outside or Inside attacks

In these conditions, there has to be implemented measures and countermeasures in order to manage these risks or to mitigate the effects of such attacks. One of the necessary measures is to separate each user, through cryptographic tunnels, so, as soon as a vulnerability or an attack will be detected, it can be isolated starting with blocking authentication and, obviously, access to system's resources. A Public Key Infrastructure will offer adequate resources in order to manage these security necessities.

Depending on the data sensitivity and on the amount of data that has to be sent through the network, a proper solution will consist in communication ciphering with symmetrical algorithms and key exchange through asymmetric cryptography or only ciphering by asymmetric cryptography. Due to PKI resources, immediate measures could be taken in order to manage user rights or keys exchange.

2.3 Expanding the Secret Keys Space by Usage of Conductive Mesh Networks

2.3.1 Motivation

The manner in which a secret key is used in symmetric cryptographic systems is a permanent challenge for cryptographers due to its importance in overall security required for information protection. Also, it has to be taken into account that changing the keys too often or too rarely could be exploited by an attacker in different ways. In order to mitigate the success rate of an attacker, a key expansion stage is included in cryptographic algorithms. This work presents an innovative solution which expands the key space, without reveal any information to an attacker who might monitor the communication channel. The prerequisites of this approach are the secret keys, a pseudorandom numbers generator and a random number generator based on unpredictable results of some measurements, all of them being aggregate by a challenge response mutual authentication protocol and a permutation function.

Applicable in the symmetrical cryptography domain, the proposed solution to extend the lifetime of the secret keys without increasing the risk in security is to identically permute the keys in each corresponding terminal according to the result of a challenge response protocol. So, after each logical (re)synchronisation of cryptographic terminals, different keys will be used in ciphering algorithm. During this protocol, random numbers are generated inside of each device. For an efficient implementation, a conductive mesh and the associated electronic circuits (tamper proof solution) (Vasile and Svasta2019), dedicated to protect the cryptographic module against physical intrusion attempts of an attacker, is used to generate random numbers. Due to this keys space extension process the confusion and diffusion properties belonging to cipher algorithm are improved.

To implement the mentioned tamperproof solution, 32 different frequency signals are injected into the conductive network and the output signals are measured. The resulted values characterize the conductive mesh, being dependent on design, and detect any tamper intrusion by observing any changes. Performing repetitive measurements on each frequency it was observed a dispersion of amplitude values, caused by environmental noise, with an average of 4 quanta of a 12 bit ADC. Coding this dispersion on 2 bits and considering the mentioned 32 frequencies result an unpredictable 64 bits stream that can be used as a random number. It is important to notice that an attacker couldn’t successfully influence the electromagnetically environment or external temperature, in order to control the randomness, due to implicitly affecting of tamperproof sensor.

In continuation, the obtained random 64 bits are used in a challenge response protocol in order to establish, in negotiation with a corresponding terminal, a common number that will be used, in the next stage, by the permutation function (Knuth 1997). The permutation function generates a new key by rearranging the original bits of the initial key according to a dedicated rule being in correspondence with each random number. The usage of this permutation function provides uniform cover of keys space determined by the all bits combination.

2.3.2 Results

Targeting the key expansion, by proposed method not only the overall security of a cryptographic terminal was achieved, but also confusion and diffusion of a secure cipher were improved (Shannon 1945). This method could be applied to any cipher type belonging do symmetrical cryptography, either block (like AES) or stream ciphers. The experimental results, exemplified in Table 1, confirm that a conductive mesh could be used in order to generate random numbers.

Frequency [MHz]	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190	195	200	205
max	3067	2871	3028	3187	3171	3104	3057	3126	3183	3205	3175	3095	3021	2997	3032	3058	3093	3117	3111	3041	2857	2637	2640	2674	2535	2369	2604	2638	2611	2507	2340	2220
min	3062	2867	3025	3183	3167	3100	3053	3121	3179	3203	3171	3091	3017	2994	3029	3054	3088	3114	3106	3037	2852	2632	2635	2669	2531	2363	2601	2633	2608	2503	2337	2216
average	3065	2868	3026	3186	3170	3102	3055	3124	3180	3204	3172	3093	3019	2995	3030	3057	3090	3116	3109	3038	2854	2634	2638	2672	2533	2366	2602	2635	2610	2505	2338	2218
difference	5	4	3	4	4	4	4	5	4	2	4	4	4	3	3	4	5	3	5	4	5	5	5	5	4	6	3	5	3	4	3	4

Table 1: The dispersion of amplitude values measured at the output of conductive mesh

Further, random numbers, RNDa and RNDb, that were been generated by each communication terminal, are used in a challenge response mutual authentication protocol, improved with time stamp labels, against time replay attacks which are targeted to discover authentication key, Ka, supposing the hash function, h, is known, according Kerckhoffs's principle.

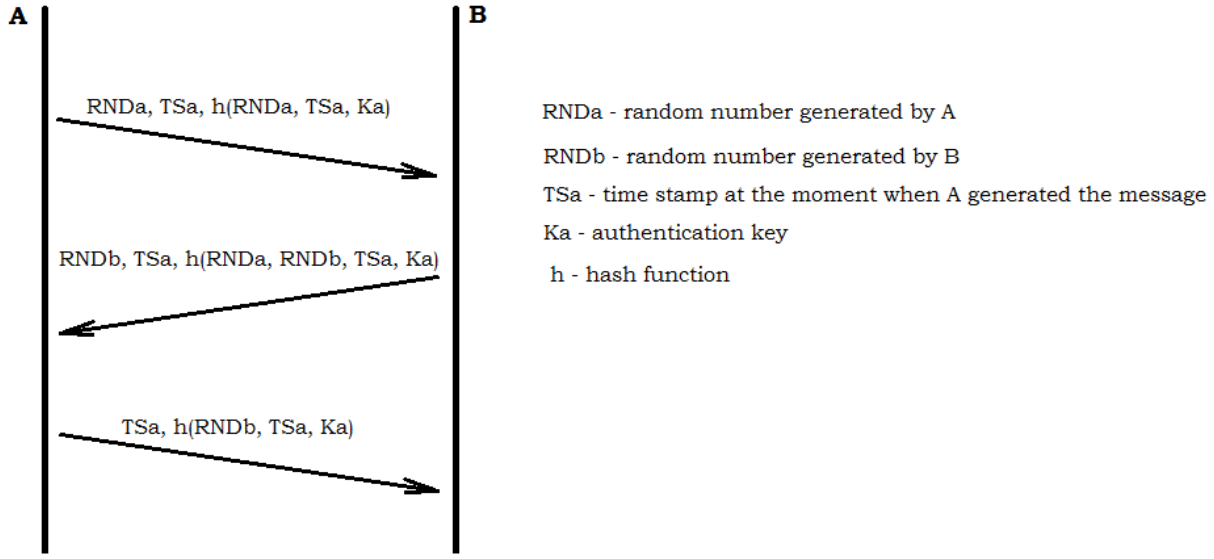

Figure 4: An efficient implementation of a challenge response mutual authentication protocol

The presented solution, besides its contribution to security improvement of the secret keys usage in a cipher algorithm, is costless due to already existing conductive mesh, which is integrated in the tamperproof module.

3 Mitigation of Sight Channel Attacks

3.1 General Presentation of Side Channel Attacks Types

Side channel attacks consist in any type of actions that exploits unwanted signals transmission (through any environment) or behaviour of a cryptographic module which can cause the loss of any of information security attributes. So, it must be assumed that an attacker can monitor any communication channel, radio signals which are generated by any device and can produce any environment conditions that could affect the proper functioning of a cryptographic device in order to gain access to sensitive data.

Among the most well-known factors that could be analysed in this scope there are:

- Noise signals superimposed over normal communication signals, which could contain useful information for an attacker;
- Unwanted electromagnetic waves generated by electronic components of a cryptographic module;
- Processing time of some cryptographic electronic circuits depending on keys structure;
- Unwanted or unpredictable behaviour of a cryptographic module when it is forced in abnormal conditions (physical or logical).

3.2 Tamper proof Solutions

One of the mandatory solutions to secure cryptographic modules against physical intrusions is the tamper proof category. By implementing a tamper proof solution, a large number of possible attacks against a cryptographic module are annihilated.

Depending on targeted security level, it can be implemented solutions starting from a simple switch, which detect the opening of the case, to complex circuits, which can monitor the whole volume of the cryptographic module. Obviously, the more complex electronic circuits will consume more electrical energy, affecting the autonomy of the cryptographic module in power off mode and, implicit, its capacity of self-protection.

The previous work in this domain, whose results were presented in UPB contribution to *D1.2 – Security Concepts*, constituted a starting point for actual research. Even if the electronic characteristics of a Conductive Mesh Network doesn't reveal any possibilities for an attacker to penetrate into a cryptographic module without detection and, implicit, destruction of all sensitive data, the actual challenge was to develop a solution which can be designed without involvement of human operators so, the CMN's electrical characteristics could not be known by any person.

3.3 Design of Conductive Mesh Networks Based on the Output of a True Random Number Generator

3.3.1 Motivation

In order to improve the security provided by this principle, based on the flexibility of common technologies that can be used to produce PCBs, an algorithm to produce particular designs of conductive mesh on PCBs starting from random bit strings is present in this work. Random design of conductive mesh is useful in order to increase the unpredictability of its electrical characteristics so, in addition to the sensitivity of this conductive mesh which will detect and react even to any attempt of measuring it by probes, an attacker will not have any information which can be exploited. The proposed innovative algorithm provide filling of the full area of envelope which cover the cryptographic module, even if its perimeter is irregular, according to necessary dimensions and profile, keeping traces on a dense grid, without any uncovered areas. The distance between the grid dots shall be close enough to avoid any intrusion attempt by any means, even cut or bypass of any part of the route. Also, a true random number generator was used, as an input to algorithm, in order to obtain various route designs, which will have unpredictable electrical characteristics. The main effort in algorithm design was focused on identifying all criteria that has to be taken into consideration in route design in order to obtain the desired objective.

The main advantage of the proposed solution consist of the possibility to implement a fully automated production flux, without human participation and with an increased level of security due to unpredictable electrical characteristic of conductive mesh generated from a true random bit string.

3.3.2 Results

The algorithm developed is proper to design various forms of PCB's routes, starting from predefined perimeter of the tamperproof envelope and true random bit string as it can be shown, as an exemplification, in Figure 5 and Figure 6.

```
01111000 01101100 00110100 01111000 01001100 11110111 10001010 01001010
11101110 11010011 00100100 10101100 10111111 11101101 11011001 11100011
00011110 01000110 10000110 10000011 11101101 00001110 10110110 01100100
10001110 01010101 11111110 10110110 11110100 11101100 00100011 10111010
00100000 00001001 10100011 00111011 00111101 10110010 01011011 10000111
00100110 10100000 00100010 11011110 00100000 10111101 11011111 11111100
10010000 11010101 10101100 00110110 01101100 00000111 11001110 11101010
01001011 11010101 01001010 00110001 00111110 11101101 10011000 01000010
11111011 11010111 10000010 11000100 00101010 11100011 11111111 01110111
10101001 00110111 00101010 11011010 11011101 00010011 10111101 11001000
11011011 00011001 10010110 00010101 11010111 01010110 01100111 10001000
10010000 00110100 11100111 00010111 00011001 01100001 10111100 00111011
11010111 01110000 00000011 11010100 00110000 10111110 10010110 00010111
00100100 11101000 01101111 01101010 11010011 00001000 00011010 00100101
00001111 00010000 11010100 00100010 10100111 11111001 00000101 11000010
01000100 01111000 00100110 01110000 00111001 01110100 11100110 00001100
```

Figure 5: Exemplification of TRNG output

Figure 6: The route design corresponding to TRNG output

Basically, the algorithm use an association between a pair of bits and a movement direction and, most important, its implement many sophisticated criteria in order to cover all the points of envelope grid and to avoid blocking situations like closed loops or local areas without any exit route.

The algorithm is applicable to any envelope shape, according to cryptographic modules dimensions necessities, but also to technological resolution limits of PCBs manufacturer. Trough integration of this algorithm in a production chain, besides the advantage of extended range of routes design, will be possible full automation, without any human intervention, increasing the overall security of cryptographic modules.

4 Hardware modules

4.1 Implementation of cryptographic algorithms

Industrial enterprises use automation systems to control and monitor the production processes, known as Industrial Control Systems (ICS). They include control systems and instrumentation connected to all devices and processing equipment used in production. Many ICS were defined along the actual industrial history like:

- DCS (distributed control systems);
- PLC (programmable logic controllers);
- SCADA (supervisory control and data acquisition);
- PAC (programmable automation controllers);
- IPC (industrial PC).

ICSs started from simple controllers, usually isolated from a central control unit, using one or more control loops to complex controllers characterized by a high number of inputs and outputs (logic or analogic) and communications capabilities. They also can interface with operators at the facility location using monitors and input/output devices. Actual ICSs are provided with a high processing power and high-speed communications interfaces, allowing for implementing control loops mapped for large areas. They are capable of controlling industrial processes based on decisions taken at the central coordination facility, possibly located anywhere on the Earth, or based on their own decision, computed using Artificial Intelligence (AI) algorithms.

Many ICSs are used in infrastructures with special requirements: they must operate safely and protected from attacks on their security resources (firmware, internal data, communication, protocols etc.). These types of infrastructures are known as critical infrastructures and include public safety, electric energy production and distribution, nuclear energy, chemical production, banking etc. Basically, an infrastructure is considered to be critical if its disruption can impact on economy, security or health of a nation.

The security of infrastructures is achieved at two levels: physical and data.

Physical security aims the protection of the resources of a digital equipment (firmware, software, security data) from being directly accessed by observation, measuring, acquisition etc. It is achieved by designing a special enclosure and sensors that detects tampering attempts and takes the appropriate actions so as not to reveal the security resources.

Data security means protecting stored data and communications of equipment that are connected in networks. Cryptography is the science that provides the methods for protecting the information in electronic form. It is focused on two main directions: cryptographic primitives and cryptographic protocols.

4.2 Processors cryptographic implementations: hardware versus software

Cryptographic algorithms can be implemented in a large range of logic integrated circuits (processors, graphic processors, FPGAs, ASICs etc.) depending on application requirements like: physical size, computational power, power consumption, security evaluation criteria etc. For distributed industrial control there is necessary for relatively small devices that are characterized by small dimensions, low power consumption, software updates (including security updates), sensors interfaces and communications ports. These devices usually do not process high data volumes, so this is the reason why they are mostly implemented in processors and microcontrollers. The microcontroller has the advantage that they have a compact structure which enclose FLASH memory, RAM memory and many ports for communications and interconnections to sensors.

The actual advances in technology endowed the processors (and microcontrollers as well) with cryptographic engines, which are logic blocks that perform cryptographic primitives as: one-way hash functions, symmetric key cryptography, public-key cryptography, digital signatures, random number generators etc. Processors producers ensure that these primitives are safe and reliable but they are not certified by cryptographic evaluation laboratories. In many fields of activity, the cryptographic certification of equipment is required. In addition to this, the cryptographic equipment has a controlled regime for the entire operational life.

In addition to the cryptographic primitives implemented by the producer in the processor's chip, the developers can implement these primitives in software/firmware. In this way the code can be fully evaluated by the cryptographic evaluation laboratories based on the provided documentation. As expected, the processing speed is slower than in the case of using the cryptographic coprocessor. The software/firmware can run at different levels when using an operating system: application or kernel driver. The less interfere with the operating system the more rapid is the implementation. The fastest implementation is when the cryptographic primitives are implemented as bare-metal routines, as used in microcontrollers. If the application is simple enough, there is no need for an operating system or a real time operating system (RTOS).

The following table lists the most important features of the two types of implementation for cryptographic primitives:

Feature	**Cryptographic coprocessor**	**Software-based cryptography**
Speed	Faster execution, use dedicated hardware	Slower execution, runs firmware on a processor
Power consumption	Low	Medium to high
Continuity in execution	Does not depend on the operating system	Depends on the operating system and its security features
Firmware upgrade	Cannot be upgraded	Can be upgraded when new security features appear (secure download)

Cryptographic and security evaluation	Partial	Fully evaluable
Application security protocols	Implemented in software	Implemented in software
Security memory access	Dedicated isolated memory	Shared memory with access based on security criteria
Booting	No need to boot	Secure boot
Reverse engineering	No	Firmware read protection

Table 2: Features of implementations for cryptographic primitives

4.3 Software implementations tests in microcontrollers

In order to test the capabilities of microcontrollers, as regards the software implementation of cryptographic primitives, two types of 32 bits microcontrollers were used: STM32F769NI (200MHz) and STM32H743ZI (400MHz). Both microcontrollers are made by STMicroelectronics. The test conditions are as follows:

- algorithm: AES256-CBC;
- encryption/decryption of successive 1kB buffers;
- data volumes used for encryption/decryption: 1MB, 10MB, 100MB, 500MB;
- implementation: **mbedTLS** library (written in C);
- L1 I/D cache: 16kB;
- implementation doesn't use an operating system.

The results of the encryption and decryption tests for the STM32F769NI and STM32H743ZI microcontrollers are presented in Figure 7, respectively in Figure 8.

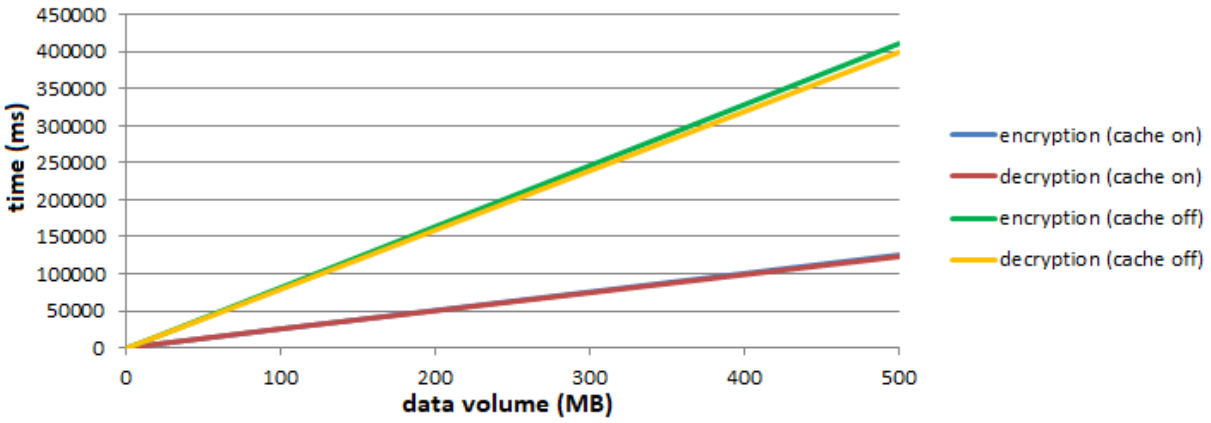

Figure 7: Encryption and decryption tests for STM32F769NI (200MHz) microcontroller using the AES256-CBC algorithm

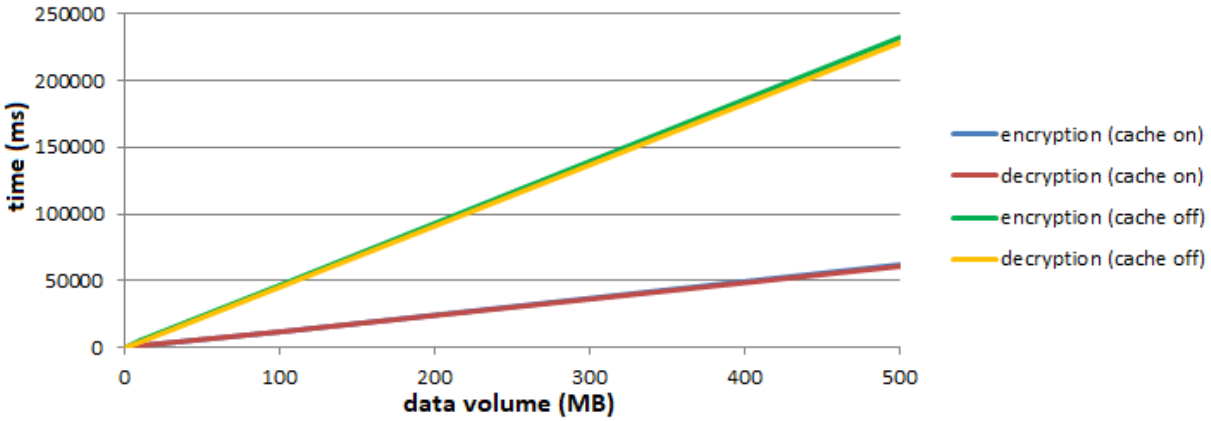

Figure 8: Encryption and decryption tests for STM32H743ZI (400MHz) microcontroller using the AES256-CBC algorithm

The following observations are derived from the analysed data:

- The variation of the encryption / decryption time, depending on the data volume, is linear because there are no other processes that interrupt these functions.
- Enabling internal cache memory (16kB data and 16kB instructions) improves processing speed more than 3 times. Iterative processing of a 1kB buffer was used in the tests.
- Higher optimization is possible if the size of the work buffers is correlated with the maximum size of the cache.
- The higher the speed of the microcontroller the higher the speed of the cryptographic primitive. The relationship is directly proportional.

5 Software modules

5.1 Overview

The security of a computation system consists of both the strength of the algorithm and the protection against cryptographic attacks. As the technologies evolve, the current state of the art of the cryptographic algorithms is practically unbreakable for brute force techniques. Thus the weak link of a system is not the algorithm itself but the entire configuration of the system, for instance: how reliable the hardware is, the operating system used by the processor, the type of software installed on the system, the security policies implemented, how reliable the key management mechanism is, etc.

An important criterion for choosing a secure platform is the price. A tamper/tempest proof hardware system may end up being very expensive. Also the operating system must be certified from the security perspective. One of the goals of a security system designer may be how he chooses a solution which offers an optimum combination between price and specifications.

Due to the existence of a wide range of operating systems and hardware equipment available on the market, attackers have found vulnerabilities and bugs that allowed them to break into the system. The users needed to know how safe a system can be and what they needed to choose in order to guarantee that the applications running on these systems were protected against leakage of information. In order to evaluate the security of the operating systems, security-auditing organizations were born. These third party entities developed standards which allow a system to be checked by using a series of tests. If a system passes a set of tests then it receives a certification together with the test results. Thus the potential user of the system will know how much he can trust the system and which are the vulnerabilities and how he can use the countermeasures in order to obtain a safe system for his applications. For software assurance, there are used the main level two standards: Common Criteria and FIPS(Federal Information Processing Standard) 140-2. For hardware assurance, level FIPS 140-2 is widely used and implemented by all major hardware makers.

Speaking about cryptographic attacks, a significant (and relatively new) class of cryptographic attacks is the side channel attack. This class of attacks doesn't use mathematical ways to break into the system but relies on different techniques such as power consumption analysis, radiation emissions, or elapsed time for running a given data process and so on. The goal of such an attack is to work in a reverse manner to figure out the "keys" for a cryptographic module. This class of attacks should be very carefully taken into consideration because it can exploit a huge amount of vulnerabilities, starting with hardware, continuing with the operating systems and going to high level software applications.

As the software depends on the hardware platform where it will run, in the following chapters we consider that is necessary to present some basic concepts regarding security on both hardware and software layers together with their vulnerabilities and also how we choose a proper implementation for the security algorithm (together with performance tests).

5.2 The security of the processor

The continuous evolution of electronics and the improvement of manufacturing technologies had led to the emergence of more and more powerful processors. In order to achieve an even higher degree of performance, hardware manufacturers analysed software requirements and tried to move as much as possible directly into the chip. Currently the market is dominated by two major general purpose CPU makers, AMD and Intel. The competition between these two giants in the PC market had led to many improvements in performance, efficiency, stability, security and so on. Each new feature added into the CPU generated software architectures changes in order to properly use the new hardware. In the following paragraphs we chose to present the Intel CPU structure from the security perspective, having in mind the goal of the project.

The standard Intel Architecture (IA) provides a logical structure based on four protection rings. Figure 9 illustrates the Standard IA Protection Rings.

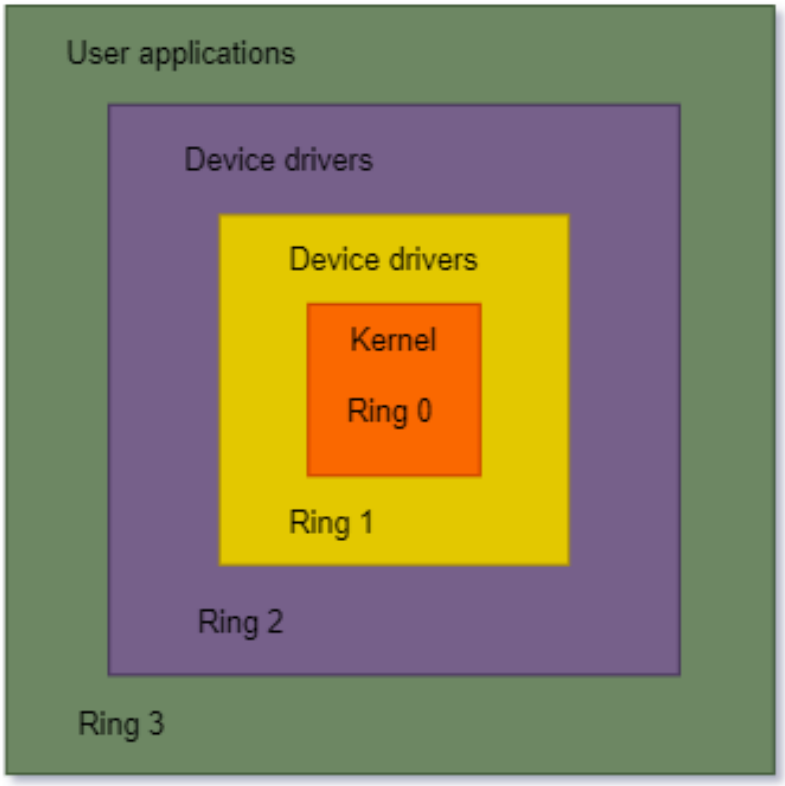

Figure 9: Positive security rings

Each ring is allocated in the following manner:

- Ring 0 represents the highest privilege ring, which is used by the kernel
- Ring 1 and Ring 2 are destined to device drivers calls
- Ring 3 represents the lowest privilege where all user applications run.

Over this base architecture, Intel provides another 3 auxiliary security rings, which are used by the processor's internal mechanisms. The four standard security rings are called "positive rings". The hardware dependent security rings are called "negative rings" and are indexed

from Ring -3 to Ring -1. Each of these negative rings allows operations for specific components. The diagram with all positive and negative rings is shown in the Figure 10.

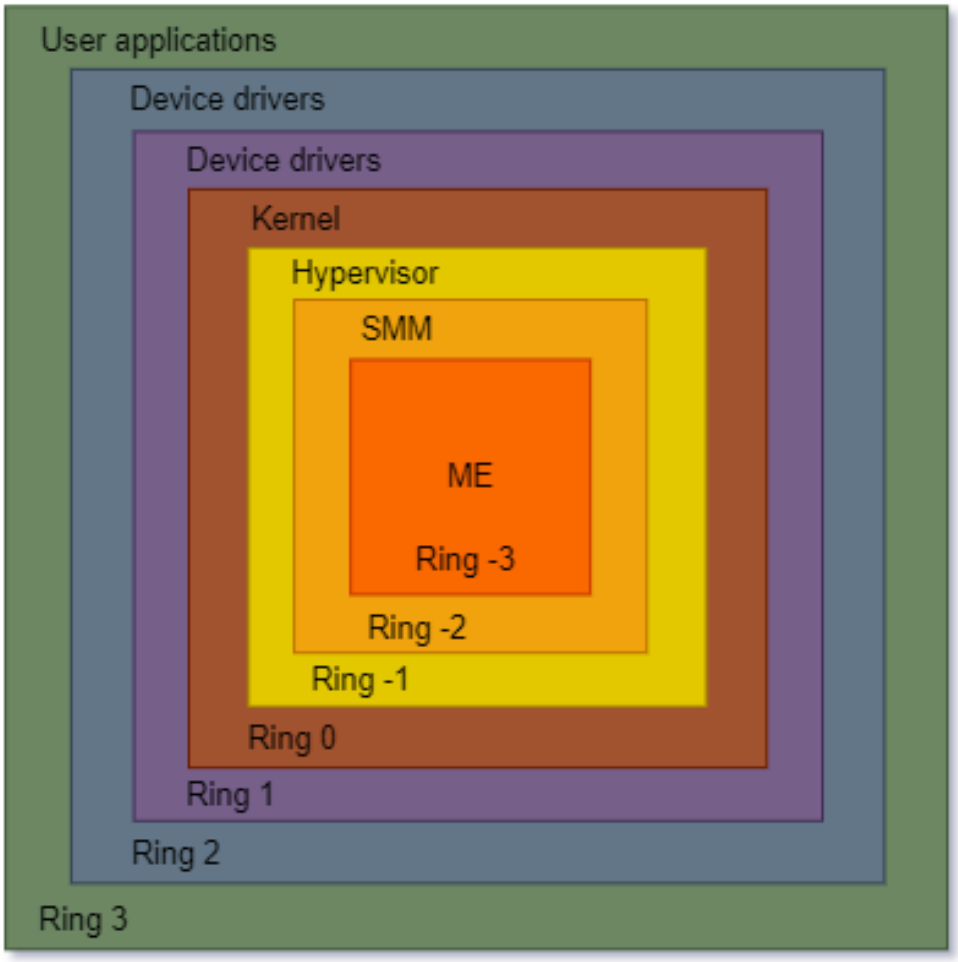

Figure 10: Positive and negative security rings

The description of the negative rings is presented in the following paragraph (RealWorldCyberSecurity, 2020):

- Ring -1 is allocated for the Hypervisor. The hypervisor is the program that boots when the computer starts. Some examples of hypervisors are Microsoft Hyper-V, VMware ESXi or Linux KVM. The hypervisor runs only after the firmware was configured to allow virtualization.
- Ring -2 represent the System Management Mode (SMM). This security ring is a lower ring than the Ring -1. The processor has two modes of operation: real address mode and virtual address mode. When a computer starts or is restarted, the first thing it does is to load the processor's firmware. When running the firmware, the processor is in real mode address, where all peripheral equipment or the memory is addressed directly, without using memory translation address operations. After the code from the firmware is executed and the processor starts loading the operating system, the processor moves from real address mode to virtual address mode and the real addresses will be mapped to virtual addresses in order to create security rules and to

supervise others applications access to memory and shared resources. SMM is an OEM (Original Equipment Manufacturer) computer-specific software which is able to manage power, thermal and some other hardware management. This firmware is executed only when the processor is in SMM and any other instructions are ignored (including interrupts). Execution of the SMM program is completely transparent to the entire rest of the processor. The SMM permits the processor to fully control the memory and peripherals.

- Ring -3 is called the Management Engine or ME. The Management Engine runs in a different chip and the entire structure is called the Platform Controller Hub (PCH). The older chip architectures have used two auxiliary controllers, the Nord Bridge Controller and the South Bridge Controller in order to communicate with different components (North Bridge was used for high speed peripherals and South Bridge was used for the lower speed peripherals). In modern architectures the PCH or the Platform Controller Hub is used in order to include the System Clock (SC), the Flexible Display Interface (FDI) and the Direct Media Interface (DMI). The processor has the features of the North Bridge Controller implemented inside and only communicates with the PCH, obtaining a performance increase. So the ME application is always running, even when the system is powered off (as long as the mainboard has power, either from a power line or from a battery). The ME has full access to the main processor, all memory (including SMM) and all peripherals. The system cannot start without PCH and this chip cannot be disabled. Due to the fact that the PCH is transparent for the CPU, ME software cannot be protected with any antivirus or antimalware application.

The negative security rings are not susceptible to vulnerabilities. The hypervisor can be attacked using a side channel attack, where a guest operating system tries to extract information from another operating system. The SMM is a software module so it is susceptible to BIOS Rootkits or Bootkits. The ME, which is the most sensitive component, can be exploited in order to steal data by using malware. All these vulnerabilities are detailed in speciality literature.

These concepts are presented only to be aware that the hardware chips are susceptible to different attack types and to emphasize the fact that all the security criterias for a secure system must be accomplished in order to have a safe system (security policies, RBAC, etc.).

5.3 Operating system security

An operating system is a software application which makes the link between hardware and high-level user's applications. It manages all the resources and allows applications to have access and to use only the resources that they need. An important feature of modern operating systems is that they allow concurrent process running while having well defined boundaries for each process. Today on the market are many operating system categories like Unix, Linux, Windows, macOS and the list can continue with hundreds of items.

In order to respond to requirements such as having a general purpose system for running different applications, not only security ones, and for allowing an easy to use and easy to administer OS, we are proposing the usage of the Microsoft's Windows platform which is well known by the vast majority of users. In the following lines, having in mind the project's current context, we'll make an analysis of how this platform responds to security requirements and which are the strong and which are the weak points of it.

In the beginning of this chapter we want to emphasize that starting with Windows 10 and Windows Server 2012, Microsoft has obtained EAL4 certifications, according to the Common Criteria Standard (Microsoft 2019a). Going even further, these operating systems have also been evaluated against FIPS 140-2 and have passed the security checks (Microsoft 2019b).

Below there are listed the Windows 10 approved FIPS modules:

- Cryptographic Primitives Library (bcryptprimitives.dll and ncryptsslp.dll)
- Kernel Mode Cryptographic Primitives Library (cng.sys)
- Boot Manager
- BitLocker® Windows OS Loader (winload)
- BitLocker® Windows Resume (winresume)
- BitLocker® Dump Filter (dumpfve.sys)
- Code Integrity (ci.dll)
- Secure Kernel Code Integrity (skci.dll)

We summarized that the processor's built in AES instruction set determined the operating system to implement it. That is why Windows has specific libraries which can be referenced from code when writing an application that requires access to the processor's hardware crypto system.

From the logical structure point of view, the processor's security rings have a two-way correspondence between the Windows operating system working modes. The CPU's four security rings translated into two levels: kernel mode and user mode. An operating system segregates virtual memory into kernel space and user space. This separation provides memory protection and hardware protection against malicious software.

Virtual memory is a technique for memory management. It acts as an abstracting layer which allows the system to see the whole memory as a standalone entity, regardless of the real configuration of the memory in the system. For example, application data that a running process rarely, may be kept on the hard disk but the process itself will not be aware of that, it only sees that its data is into the virtual memory. Address translation hardware is located in the CPU (MMU – Memory Management Unit) and the operating system holds a map table where the relations between real addresses and virtual addresses are stored. In the context of multithreading virtual memory provides a space address that can exceed the real capacity size of the computer's memory. It is well known that a thread is the smallest unit of work that a scheduler can execute. The scheduler is a part of the operating system that plans the execution of available instructions. The operating system logically isolates the thread and thus

a thread is not aware of another thread's existence and, with the help of virtual memory, both threads can use the same memory space. When the scheduler commutes between threads, it saves the old thread context in the virtual memory (which can also be the hard drive). The new thread can receive the same memory space, but the data are not visible between the two threads. Usage of virtual memory mechanisms generates an increased security level and also an increased performance.

Kernel mode refers to a mean of execution in a processor that grants access to all system memory and all CPU instructions. The reason for giving a higher privilege level to the operating system is that the processor already provides the necessary base so the operating system designers are able to create internal mechanisms to ensure that a misbehaving application can't disrupt the stability of the system as a whole. Related to the virtual memory space address, it must be said that although each Windows process has its own private memory space, the kernel-mode operating system and device driver code share a single virtual address space. Each page from the virtual memory is tagged to indicate the access mode of the processor in order to read and/or write the page. The pages from the system space can be accessed only from kernel mode, whereas all the pages in the user address space are accessible from user mode. Read-only pages (such as those that contain static data) are not writable from any mode. Additionally, on processors that support no-execute memory protection, Windows marks pages containing data as non-executable, thus preventing inadvertent or malicious code execution in data areas. The 32-bit Windows doesn't provide any protection to private read/write system memory being used by components running in kernel mode. In other words, once in kernel mode, operating system and device driver code has complete access to system space memory and can bypass Windows security to access objects. Knowing that the bulk of the Windows operating system code runs in kernel mode, it is vital that the components that run in kernel mode should be carefully designed and tested to ensure that they don't violate system security or cause system instability. This lack of protection also emphasizes the need to take care when loading a third-party device driver, because once in kernel mode the software has complete access to all operating system data. This weakness was one of the reasons behind the driver-signing mechanism introduced in Windows, which warns (and, if configured as such, blocks) the user if an attempt is made to add an unsigned Plug and Play driver. On the 64-bit versions of Windows, the Kernel Mode Code Signing (KMCS) policy dictates that any 64-bit device drivers (not just Plug and Play) must be signed with a cryptographic key assigned by one of the major code certification authorities. The user cannot explicitly force the installation of an unsigned driver, even as an administrator, but, as a one-time exception, this restriction can be disabled manually at boot time (Russinovich et al. 2012). How the supervisor can perform operations on both kernel space and user space while user mode is limited to access only the user space, is shown in figure 11.

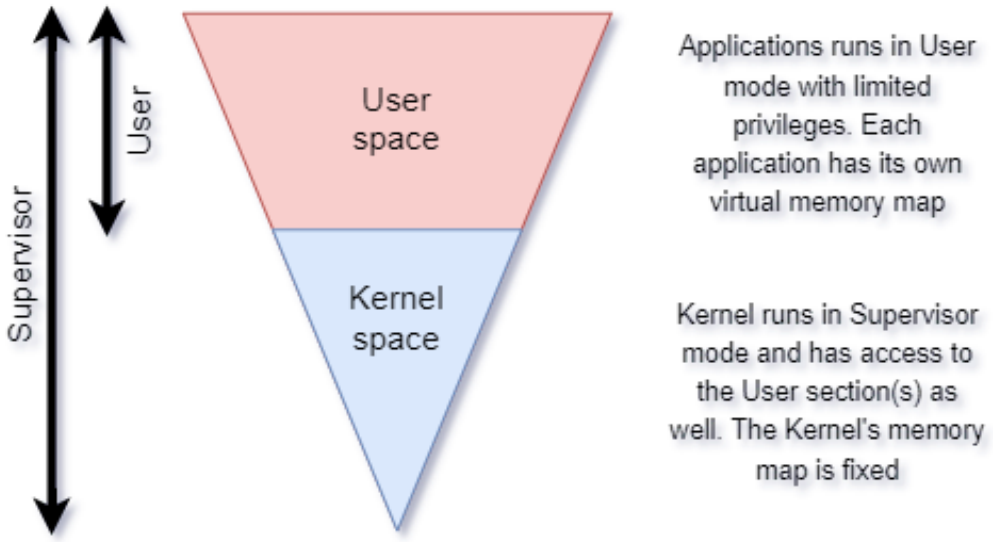

Figure 11: OS modes

5.4 Security breaches

As presented in the previous chapters, running software within different security rings allows a clear separation of concerns and ideally increases the overall system security. Intel's architecture, with not only “positive rings” (from 0 to 3), but also “negative rings” (-3, -2 and -1 rings) created security breaches. Christopher Domas, who works as a researcher for the Battelle Memorial Institute, found a hardware bug and revealed it at the Black Hat conference in Vegas, 2015 (Thomson 2015). This issue appeared for the first time in 1995 with the Pentium Pro processor. Considered to be the lowest security ring to run a code, Ring 0 which is dedicated to kernel mode operations is bypassed by the negative rings which allow running even lower level code. For example, ring -1 which is the hypervisor, has a greater privilege than ring 0. For example, when a computer is started, the operating system is not loaded immediately to RAM, but first the hypervisor checks if there are multiple operating systems on the computer and then asks which operating system to load. This behaviour stops a guest operating system from taking full control of a system. But under the ring -1 is the ring -2, which contains an even lower level for software execution. At this level the Intel System Management Mode (SMM) software runs. This level is completely independent from the upper levels and it has software that allows controlling the power of the motherboard when an operator sends wake up or sleeping commands to the system. Also this built in software emulates a PS/2 keyboard when a standard USB device is connected. The security ring -2 allocates a space in the RAM memory which is blocked by the motherboard’s Memory Controller Hub, so theoretically no one can access this memory space. Nevertheless, hackers managed to obtain access at ring -2 level and to insert rootkit code at this level. This means that the entire security of the system is compromised and the hacker can intercept every executed instruction and the operating system will not be aware of this. In this case no matter

how strong the cryptographic algorithm is, the data can be intercepted before encryption or the secret keys can be extracted.

Creating a processor, which has a hardware structure with built in components that implements an architecture based on multiple security layers is conceptually a good thing. However, this led to an increase of complexity that can later exposes bugs, which are very hard to find, and if such kind of bugs are exploited then it is very hard to find countermeasures for.

The conclusion is that it is recommended to choose a processor, which has been evaluated regarding the cryptographic matter and has a certain level of thrust.

5.5 Performance testing for different implementations

5.5.1 Motivation

The performance test aims to make a comparison between existing encryption methods for the AES algorithm using different existing libraries for different programming languages, all with the aim of analysis as detailed as possible on the methods and speed of encryption obtained.

The software module will perform a relatively large number of processes that will involve encrypting and decrypting data, in order to find the best implementation of the AES algorithms and due to the multitude of existing libraries that aim to perform encryption, the necessity to make a detailed comparison between the existing options is mandatory.

5.5.2 Test Method

In order to perform this test, it was decided to implement the two applications in the most used programming languages: Python and C# .NET Core.

Related to the used source code for python implementation of AES algorithms, it has to be mentioned that UPB has developed the test application by using Python 3.7 and Pycrypto 2.6.1. The purpose of those test suits was to compare the same algorithm implemented in different programming languages. The operating system has an important role because of distinct approaches for the kernel for both Linux and Windows OS. UPB has run the Python script on a couple of Linux distributions without any major improvement in terms of performance. Hence, in this report the focus is on the tests for Linux (2016 distribution, Build August 2020) and Windows 10 because the goal was not to test many version of the same OS but to compare the same program on distinct OSs. Regarding the performance counter, the Python's interpreter gives the major overhead (and implicitly different execution times), compared to .NET's compiler. During the compilation process the compiler optimize the source code for obtaining the .NET IL bytes code, which will be executed by the runtime. The Python's interpreter will execute the code as it is, without any optimization at runtime. Another important aspect to mention is the level of abstractization that each programming language uses. It is well known that state of the art software is designed to be modular,

scalable and flexible. That is why many wrappers are used instead of calling directly base functions provided by the OS. Each abstractization layer or each wrapper built around a set of OS calls generating a small overhead. Indeed this quantity is negligible but when speaking about near-real-time processes it is important to take that into consideration. Concluding, the target was to compare the performance of the AES algorithm when executed with an interpreter and with a compiler.

The test proposes observing the speed of the encryption and decryption process of the AES algorithm performed on different size files whose content was generated randomly. For maximum accuracy, specific timing functions have been exercised to time the encryption and decryption process, these functions returning the time required for processing in microseconds.

Due to the generality of the system and the wide variety of use cases, it will have, a range of generalized sizes was chosen for the files, used in order to observe the time differences for the encryption/decryption processes between different platforms and programming languages. The file sizes to which the test was reported are: 1MB, 5MB, 10MB, 25MB, 50MB, 100MB, 200MB, 300MB, 400MB and 500MB.

The core of each speed test consists of timing an encryption and decryption process on the received data package. For a more rigorous analysis that is not influenced by other processes running in parallel on the processor, 10 such processes were performed for each file, at a time interval of 20 seconds. The obtained times are divided by the number of tests performed so that after each file analysed, we can conclude on an average encryption/decryption period.

5.5.3 Test environment

Nowadays there are many systems widely used to meet the computational needs that have arisen with the development of technologies. There are three main types of computing machines on the market: single-board computer, personal computer, and mainframe computers.

The single-board computer is a computing machine based on a single circuit board, and even if their price is low, also the processing power is very low, and the possibility of expanding with peripherals (like RAM) does not exist.

The mainframe computers, it is a performant system that offers processing power and advanced level of reliability, but at a much higher cost.

The personal computer type represents the most common type of computer, combining the increased performance of hardware components with a price according to the needs of the users.

Due to the reduced processing power encountered at single-board computers and the high price that the mainframe computing machine has, we opted to test the speed of encryption algorithms on the range of personal computers.

To perform this test, two computers were chosen whose processor is produced by Intel. These processors are of a different model:

- Intel® Core™ i7-9750H Processor 12M Cache, up to 4.50 GHz
- Intel® Core™ i5-9600K Processor 9M Cache, up to 4.60 GHz

In the following figure, a comparison between the two systems is presented.

	Intel Core i7-9750H @ 2.60GHz	Intel Core i5-9600K @ 3.70GHz
Socket Type	FCBGA1440	FCLGA1151-2
CPU Class	Laptop	Desktop
Clockspeed	2.6 GHz	3.7 GHz
Turbo Speed	Up to 4.5 GHz	Up to 4.6 GHz
# of Physical Cores	6 (Threads: 12)	6 (Threads: 6)
Max TDP	45W	95W
Yearly Running Cost	$8.21	$17.34
First Seen on Chart	Q2 2019	Q4 2018
# of Samples	4867	2740
Cross-Platform Rating	22451	20702
Single Thread Rating	2506	2786
CPU Mark	11318	10839

PassMark Software © 2008-2021

Figure 12: Hardware comparison

Considering the comparison of the processors according to the obtained speed related to the number of cores, it can be assumed that the performance of the 2nd processor is 20% better than the first one, even if the last processor is an i7 model, compared to the first which is i5 model.

The tests were performed on two different operating systems. The operating systems were selected so that distributions were launched at about the same time. Figure 13 shows the distribution on Window 10 Pro version (Build 19041) and Figure 14 shows the distribution on Ubuntu 16.04 version (Ubuntu 16.04.7 LTS – Xenial Xerus).

Version	Codename	Marketing name	Build	Release date
1507	Threshold 1	N/A	10240	July 29, 2015
1511	Threshold 2	November Update	10586	November 10, 2015
1607	Redstone 1	Anniversary Update	14393	August 2, 2016
1703	Redstone 2	Creators Update	15063	April 5, 2017[d]
1709	Redstone 3	Fall Creators Update	16299[g]	October 17, 2017
1803	Redstone 4	April 2018 Update	17134	April 30, 2018
1809	Redstone 5	October 2018 Update	17763	November 13, 2018[j]
1903	19H1	May 2019 Update	18362	May 21, 2019
1909	19H2	November 2019 Update	18363	November 12, 2019
2004	20H1	May 2020 Update	19041	May 27, 2020
20H2	20H2	October 2020 Update	19042	October 20, 2020
21H1	21H1	TBA	19043	TBA

Figure 13: Windows 10 distributions

Name	Last modified	Size	Description
12.04.5/	2019-03-12 05:16	-	Ubuntu 12.04.5 LTS (Precise Pangolin)
12.04/	2019-03-12 05:16	-	Ubuntu 12.04.5 LTS (Precise Pangolin)
14.04.6/	2020-08-18 08:05	-	Ubuntu 14.04.6 LTS (Trusty Tahr)
14.04/	2020-08-18 08:05	-	Ubuntu 14.04.6 LTS (Trusty Tahr)
16.04.6/	2020-08-18 17:01	-	Ubuntu 16.04.7 LTS (Xenial Xerus)
16.04.7/	2020-08-18 17:01	-	Ubuntu 16.04.7 LTS (Xenial Xerus)
16.04/	2020-08-18 17:01	-	Ubuntu 16.04.7 LTS (Xenial Xerus)
18.04.4/	2020-08-13 15:39	-	Ubuntu 18.04.5 LTS (Bionic Beaver)
18.04.5/	2020-08-13 15:39	-	Ubuntu 18.04.5 LTS (Bionic Beaver)
18.04/	2020-08-13 15:39	-	Ubuntu 18.04.5 LTS (Bionic Beaver)

Figure 14: Ubuntu distributions

5.5.4 Observation and discussion

In this section, we are going to analyse the graphs resulting from running the tests which aims to obtain the encryption and decryption speed of the AES algorithm produced by the two libraries on which we agreed to analyse (System.Net.Cryptography and Crypto).

In order to observe as correctly as possible the data obtained after running the tests, it was decided that these values will be illustrated with the help of graphs, where, on Y-axis, it will be displayed the number of milliseconds needed to perform an encryption or decryption process, and on the X-axis, the number of the sample.

The following figures (figure 15 to 20) represent the data recorded from the tests run on the computer whose processor is i7 9750H. The computer runs on a Windows 10 Pro operating system, and the tests were performed using both computer programs written in C# .NET Core and Python.

The first set of tests (see figures 15 and 16) were performed using both programs, implicitly both libraries, on a file with an exact size of 400MB. Analysing a single file, we can observe the encryption and decryption times of both programs for each iteration of the 10 existing ones, being able to study, possibly, certain anomalies, in case they exist.

Analysing the Figures 15 and 16, we notice that the elapsed times for the encryption process obtained by the library offered by Python (Crypto), take much longer, having an almost double value compared to the library used in the 2nd program (System.Net.Cryptography). Simultaneously, the processing times for the decryption process decrease to 970 milliseconds for the program made in Python, but all the while, they still take longer than the decryption times for the C # .NET Core library.

Encryption:400MB OS:Windows PC: i7 9750H 2.60 GHZ		
Test NO	Elapsed time(C#)	Elapsed time(Python)
1	680	1198
2	671	1245
3	677	1222
4	674	1234
5	670	1220
6	676	1223
7	672	1299
8	668	1276
9	683	1309
10	669	1250

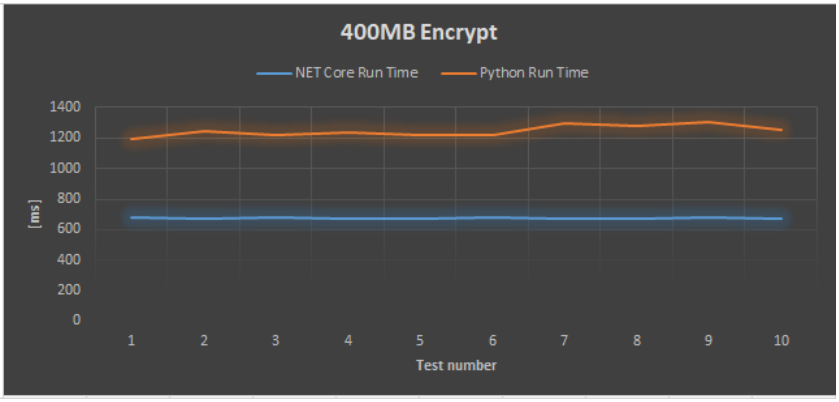

Figure 15: Encryption of 400MB file

Decryption:400MB OS:Windows PC: i7 9750H 2.60 GHZ		
Test NO	Elapsed time(C#)	Elapsed time(Python)
1	736	941
2	777	978
3	753	969
4	736	978
5	775	976
6	755	1017
7	734	1070
8	769	1016
9	771	1007
10	754	990

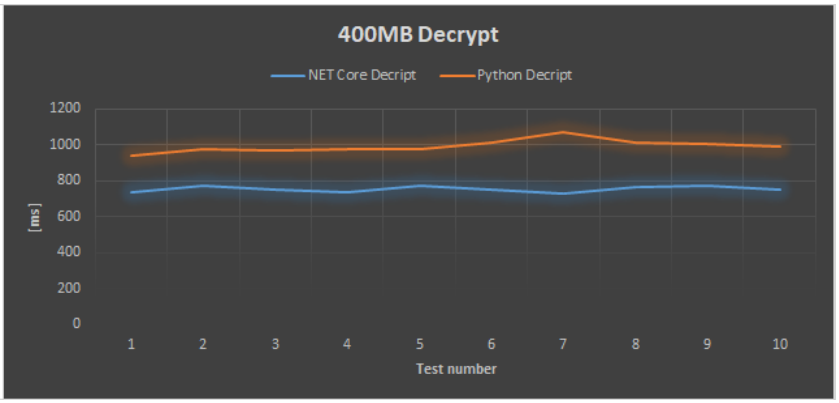

Figure 16: Decryption of 400MB file

The following set of tests were done on a number of 10 files, sizes ranging from 1MB to 500MB. For each file, 10 tests were performed, their arithmetic means being recorded in the table. In these two graphs, we can see that the specific elapsed times of the encryption process but also of the decryption process, of the program written in Python, increase steeper than the one written in .NET Core. This difference becomes obvious starting with 100MB files. The data is presented in Figure 17 and 18.

Encryption:1 - 500MB OS:Windows PC: i7 9750H 2.60 GHZ		
File size	Elapsed time(C#)	Elapsed time(Python)
1MB	1	3
5MB	7	17
10MB	14	30
25MB	42	78
50MB	86	157
100MB	168	305
200MB	336	628
300MB	502	920
400MB	674	1247
500MB	845	1551

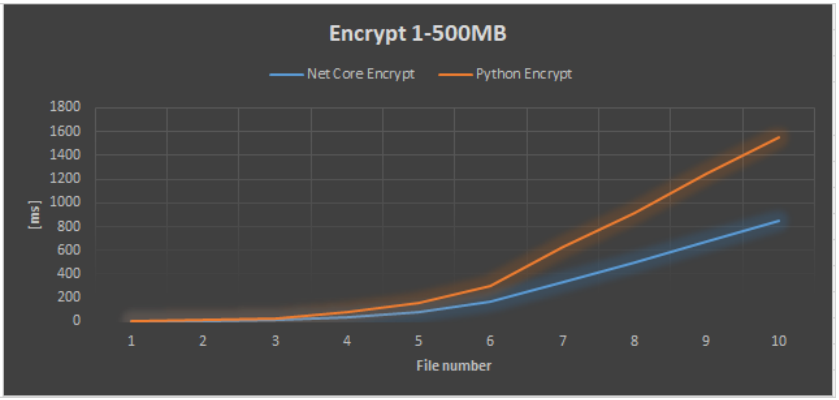

Figure 17: Encryption of files from 1MB to 500MB

Decryption:1 - 500MB OS:Windows PC: i7 9750H 2.60 GHZ		
File size	Elapsed time(C#)	Elapsed time(Python)
1MB	1	2
5MB	10	13
10MB	18	24
25MB	48	62
50MB	92	126
100MB	186	248
200MB	376	495
300MB	544	750
400MB	756	994
500MB	935	1267

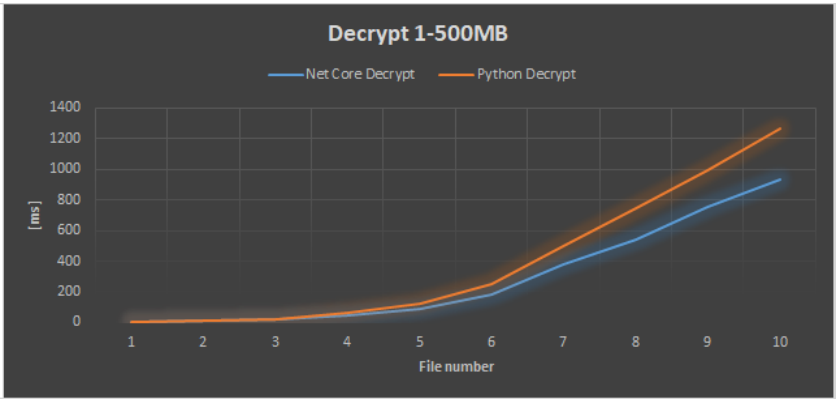

Figure 18: Decryption of files from 1MB to 500MB

The graphs shown in Figure 19 and 20 with data obtained from the computer with i7 9750H processor, illustrate the time differences between the encryption and decryption process specific to each library.

Encryption-Decryption:1 - 500MB OS:Windows PC: i7 9750H 2.60 GHZ		
File size	Enc. elapsed time(C#)	Dec. elapsed time(C#)
1MB	1	1
5MB	7	10
10MB	14	18
25MB	42	48
50MB	86	92
100MB	168	186
200MB	336	376
300MB	502	544
400MB	674	756
500MB	845	935

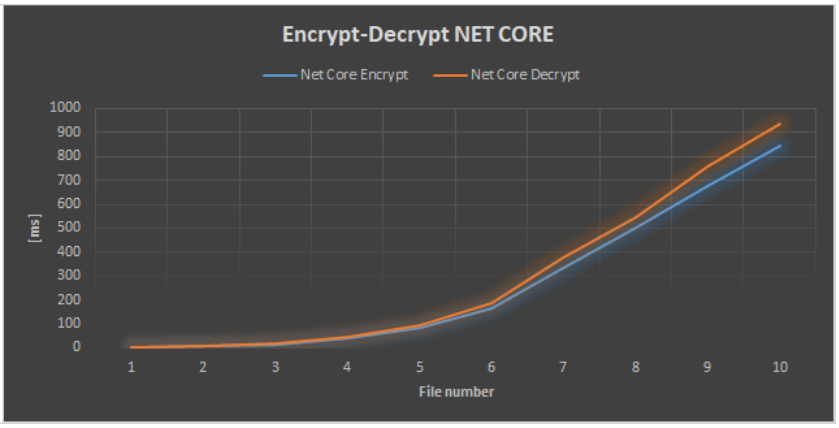

Figure 19: Encrypt/Decrypt using NET CORE

Encryption-Decryption:1 - 500MB OS:Windows PC: i7 9750H 2.60 GHZ		
File size	Enc. elapsed time(Python)	Dec. elapsed time(Python)
1MB	3	2
5MB	17	13
10MB	30	24
25MB	78	62
50MB	157	126
100MB	305	248
200MB	628	495
300MB	920	750
400MB	1247	994
500MB	1551	1267

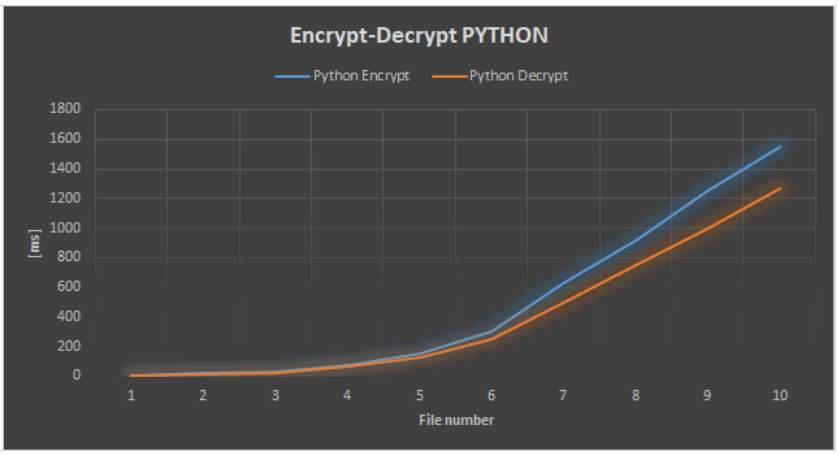

Figure 20: Encrypt/Decrypt using Python

Furthermore, in the Figure 19 it can be observed that the decryption time for the library from .NET Core increases slightly compared to the encryption time, this increase occurs after processing files with a size larger than 100MB. On the other hand, the period required for the encryption process for the Python library increases sharply compared to the decryption time, a rise occurring following the processing of files larger than 100 MB. Weighing the two graphs, we notice that, overall, the encryption and decryption processing times for .NET Core are smaller than those of the Python library.

The Figures from 21 to 26 represent the data recorded from the tests run on the computer whose processor is i7 9600K. The computer also runs on a Windows 10 Pro operating system, and the tests were performed using both programs.

Identical to studying the data specific to the first computer targeted for testing, we start by analysing the graphs regarding the encryption and decryption times for a file of 400MB.

In Figure 21 and 22, it is shown that the time difference between the two languages on the first computing device is maintained on the second computer, even if there is a slight decrease in encryption time for the Python program. In addition, for the decryption process, we see a significant increase of approximately 50 milliseconds for the .NET Core library, and a decrease of approximately 50 milliseconds for the Python library.

Encryption:400MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
Test NO	Elapsed time (C#)	Elapsed time(Python)
1	574	1107
2	601	1103
3	592	1082
4	608	1084
5	590	1129
6	600	1128
7	601	1112
8	602	1116
9	599	1123
10	602	1120

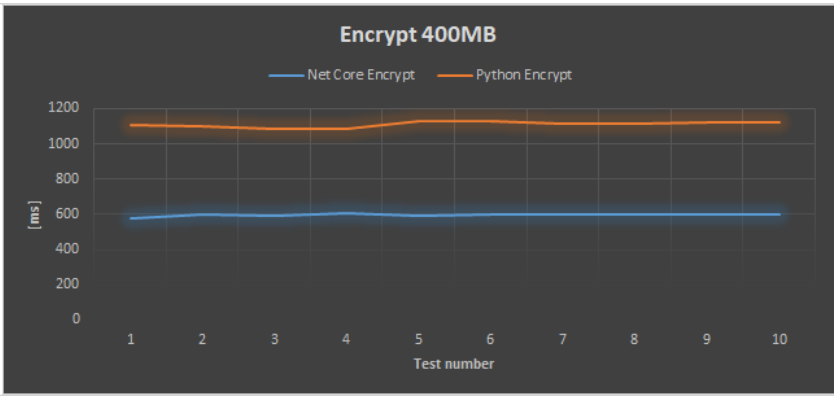

Figure 21: Encryption of a 400MB file on i5 9600k CPU

Decryption:400MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
Test NO	Elapsed time (C#)	Elapsed time(Python)
1	815	850
2	812	856
3	832	858
4	822	848
5	820	915
6	829	884
7	827	885
8	830	917
9	844	855
10	837	883

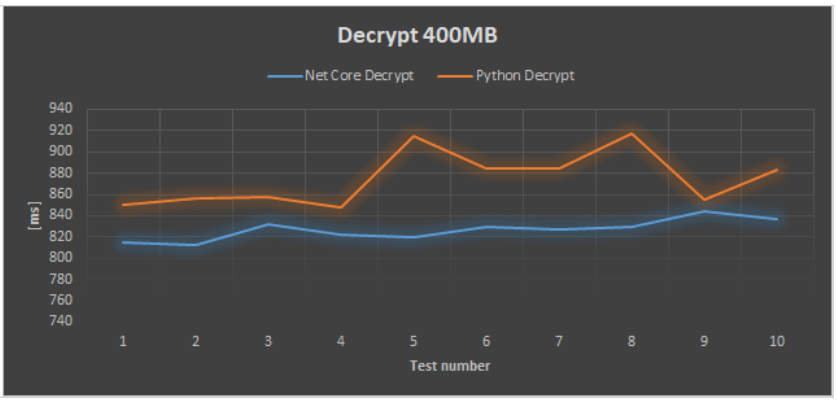

Figure 22: Decryption of a 400MB file on i5 9600k CPU

Figure 23 and 24 demonstrate that, as with the first computer, the encryption interval for the program written in Python increases compared to the encryption interval for the program written in .NET Core, this pattern can be observed starting above 100MB file. The difference from the first computer analysed is given by the time contrast of the decryption processes. Here we can see that the processing times for decryption are almost identical. This is due to the increase of the processing time for the program written in .NET Core.

Encryption: 1 - 500MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
File size	Elapsed time(C#)	Elapsed time(Python)
1MB	1	2
5MB	9	13
10MB	16	28
25MB	36	68
50MB	73	138
100MB	149	266
200MB	299	538
300MB	449	826
400MB	584	1129
500MB	744	1380

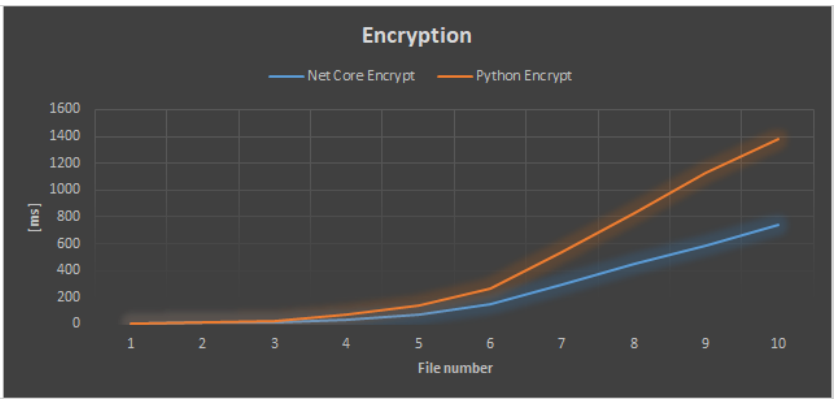

Figure 23*:* Encryption C# vs Python on i5 9600k CPU

Decryption:1 - 500MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
File size	Elapsed time(C#)	Elapsed time(Python)
1MB	1	2
5MB	7	10
10MB	15	22
25MB	36	59
50MB	104	109
100MB	194	217
200MB	412	428
300MB	604	643
400MB	810	859
500MB	1025	1130

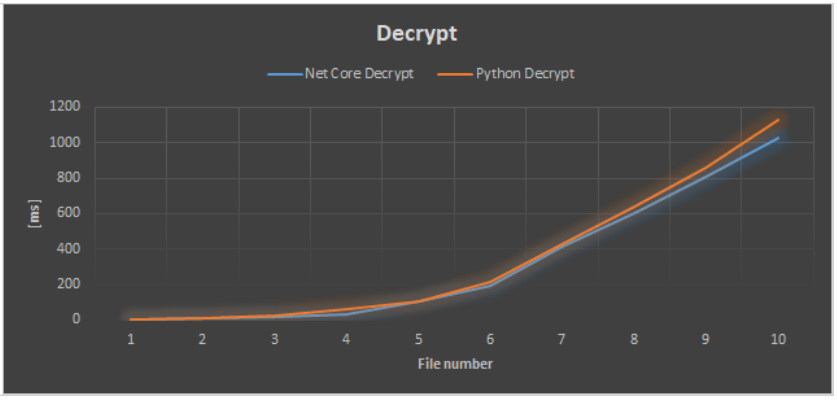

Figure 24*:* Decryption C# vs Python on i5 9600k CPU

Comparing the two diagrams presented in Figure 25 and 26 with the two specific graphs from the first computer, we notice that the processing time evolves identically for each file, but we can observe a small improvement in processing time for the second computer.

Encryption-Decryption:1 - 500MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
File size	Enc. elapsed time(C#)	Dec. elapsed time(C#)
1MB	1	1
5MB	9	7
10MB	16	15
25MB	36	36
50MB	73	104
100MB	149	194
200MB	299	412
300MB	449	604
400MB	584	810
500MB	744	1025

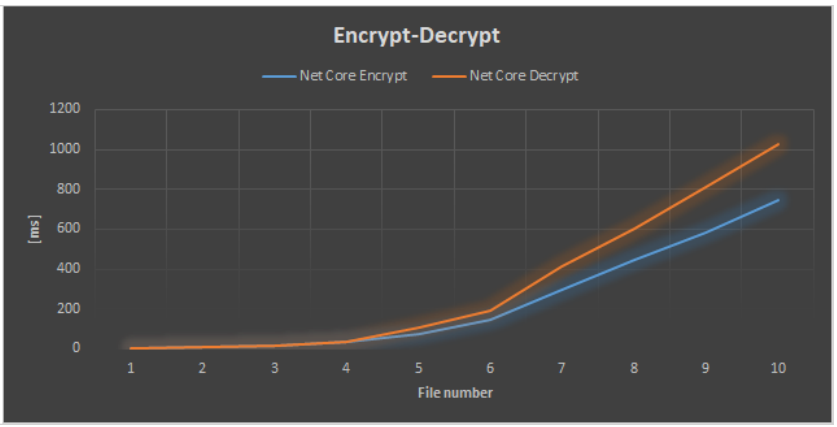

Figure 25: Encrypt/Decrypt C# on i5 9600k CPU

Encryption-Decryption:1 - 500MB OS:Windows 10 PC: i5 9600K 3.7 Ghz		
File size	Enc. elapsed time Python	Dec. elapsed time Python
1MB	2	2
5MB	13	10
10MB	28	22
25MB	68	59
50MB	138	109
100MB	266	217
200MB	538	428
300MB	826	643
400MB	1129	859
500MB	1380	1130

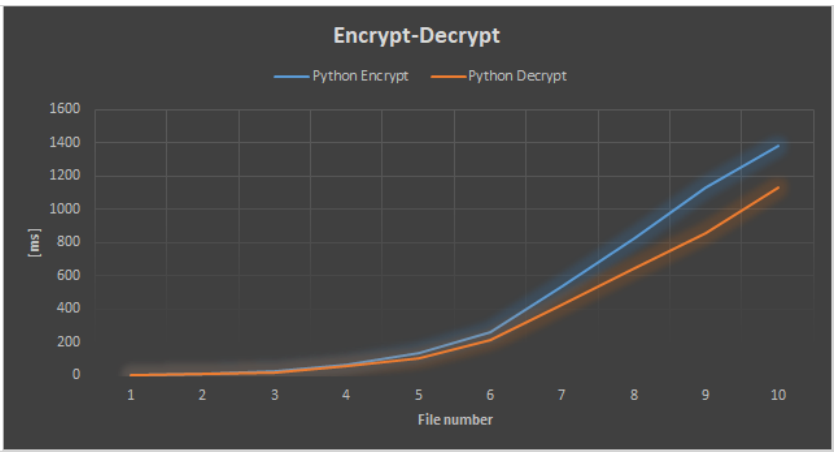

Figure 26: Encrypt/Decrypt Python on i5 9600k CPU

With the help of the graphs shown in Figures 27 to 31, it is possible to observe the differences of the processing times for encryption and decryption between two identical computers, each running on a different operating system (Windows 10 PRO and Ubuntu 16.4), using the same test implemented in Python.

The graph presented in Figure 27 illustrates the time differences between the encryption and decryption process on the Ubuntu operating system. We can see that the time difference decreases compared to the same test run on windows.

Encryption-Decryption:1 - 500MB OS: UBUNTU16.4 PC: i5 9600K 3.7 Ghz		
File size	Enc. elapsed time(Pyt)	Dec. elapsed time(Pyt)
1MB	1	1
5MB	8	8
10MB	17	19
25MB	49	37
50MB	123	114
100MB	250	228
200MB	493	468
300MB	740	682
400MB	998	912
500MB	1230	1147

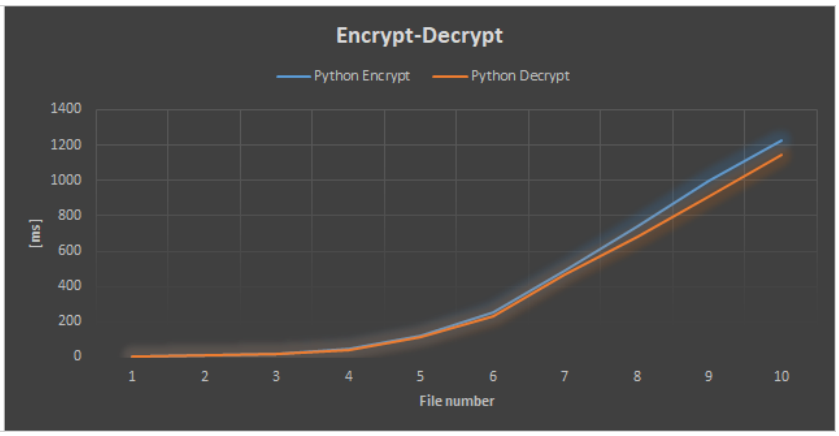

Figure 27: Encrypt/Decrypt files from 1MB to 500MB on i5 9600k CPU using Python on Ubuntu

Performing the analysis of the processing times on a file with a size of 400 MB (Figure 28 and 29), it is discovered that, even if the encryption times are similar between the 2 operating systems, the computer that uses Windows takes a shorter period of time to decrypt the file.

Encryption:400MB S: UBUNTU16.4/WINDOWS - PYTHO PC: i5 9600K 3.7 Ghz		
Test NO	Elapsed time(Ubuntu)	Elapsed time(Win)
1	984	1107
2	990	1103
3	988	1082
4	977	1084
5	994	1129
6	992	1128
7	984	1112
8	998	1116
9	912	1123
10	989	1120

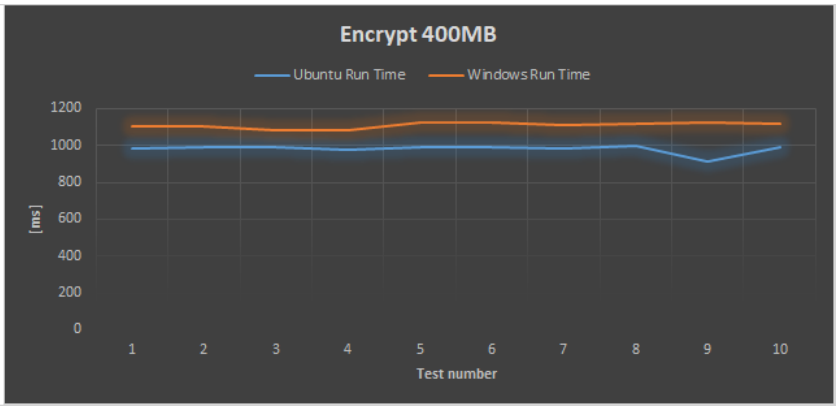

Figure 28: Encrypt Ubuntu vs. Windows on i5 9600k CPU

Decryption:400MB S: UBUNTU16.4/WINDOWS - PYTHO PC: i5 9600K 3.7 Ghz		
Test NO	Elapsed time(Ubuntu)	Elapsed time(Win)
1	914	850
2	915	856
3	924	858
4	910	848
5	910	915
6	912	884
7	909	885
8	915	917
9	912	855
10	911	883

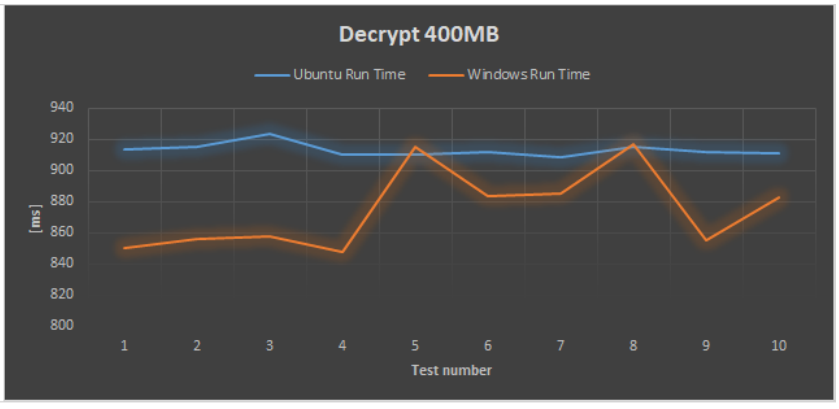

Figure 29: Decrypt Ubuntu vs. Windows on i5 9600k CPU

Analysing the two graphs from Figures 30 and 31 that illustrate the different times of the encryption and decryption process on the 2 operating systems, we notice they are almost identical. The only difference that appears is in the encryption process on larger files, where the Windows operating system obtaining a slightly longer time than Ubuntu.

Encryption:1 - 500MB S: UBUNTU16.4/WINDOWS - PYTHO PC: i5 9600K 3.7 Ghz		
File size	Elapsed time(Ubuntu)	Elapsed time(Win)
1MB	1	2
5MB	8	13
10MB	17	28
25MB	49	68
50MB	123	138
100MB	250	266
200MB	493	538
300MB	740	826
400MB	998	1129
500MB	1230	1380

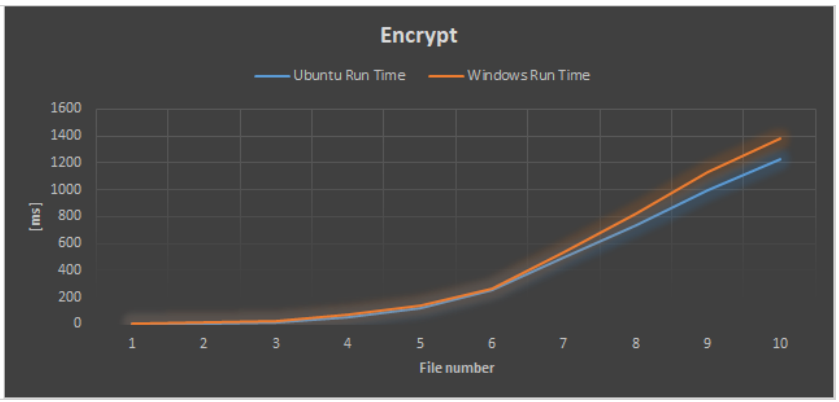

Figure 30: Encrypt Ubuntu vs Windows, 1MB to 500MB file sizes, on i5 9600k CPU

Decryption:1 - 500MB S: UBUNTU16.4/WINDOWS - PYTHO PC: i5 9600K 3.7 Ghz		
File size	Elapsed time(Ubuntu)	Elapsed time(Win)
1MB	1	2
5MB	8	10
10MB	19	22
25MB	37	59
50MB	114	109
100MB	228	217
200MB	468	428
300MB	682	643
400MB	912	859
500MB	1147	1130

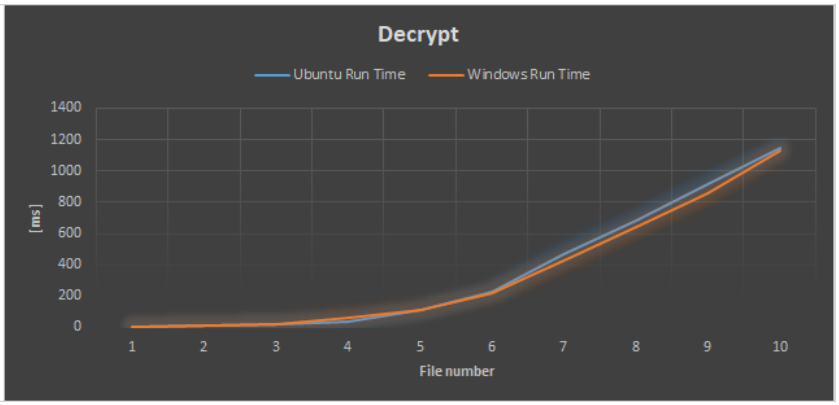

Figure 31: Decrypt Ubuntu vs Windows, 1MB to 500MB file sizes, on i5 9600k CPU

5.5.5 Test results

Considering the comparisons listed above related to time differences in the encryption and decryption processes, we conclude that the most suited operating system on which we will develop the application is Windows. In addition, the programming language that performed best and set the best times is C#, this leads to the choice of the .NET framework as the most suited for the development of the application.

5.6 AES implementation

Choosing a software platform in order to implement the cryptographic module requires analysis and comparison of the state of the art technologies available in the market. Creating strong cryptographic components led to hardware related design. For example, Intel, from 2010 started to produce a new processor (code name Westmere) where it's internal mechanisms allow the usage of a new instruction set - Intel Advanced Encryption Standard New Instructions (AES-NI). These instructions were created for moving intensive calculations required by the AES from software to hardware and thus accelerating the AES algorithms performance. By using the AES-NI instruction set the performance of an implementation of AES can be increased by 3 to 10x, according to Intel measurements. The AES algorithm divides the plain text into 128 bits blocks, so the basic unit size is 128 bits. The keys used for

cryptographic operation can have 128, 192, or 256 bits. Depending on the size of the key which will be used, the AES will perform a number of 10, 12, or 14 rounds. Each round consists of a set of mathematical operations and the result of each round is then passed to the following round as an input. Also, each round is encrypted using a subkey that is generated using a key schedule. The new AES-NI instruction set consists of six new instructions that perform several compute-intensive parts of the AES algorithm. These instructions can execute using significantly fewer clock cycles than a software solution. Four of the new instructions are for accelerating the encryption/decryption of a round and two new instructions are for round key generation. The following is a description of the new instructions (Rott 2012).

Instruction	Comment
AESENC	This instruction performs a single round of encryption. The instruction combines the four steps of the AES algorithm - ShiftRows, SubBytes, MixColumns & AddRoundKey into a single instruction.
AESENCLAST	Instruction for the last round of encryption. Combines the ShiftRows, SubBytes, & AddRoundKey steps into one instruction.
AESDEC	Instruction for a single round of decryption. This combines the four steps of AES - InvShiftRows, InvSubBytes, InvMixColumns, AddRoundKey into a single instruction
AESDECLAST	Performs last round of decryption. It combines InvShiftRows, InvSubBytes, AddRoundKey into one instruction.
AESKEYGENASSIST	Is used for generating the round keys used for encryption.
AESIMC	Is used for converting the encryption round keys to a form usable for decryption using the Equivalent Inverse Cipher.

Table 3: AES Instruction Set

The benefits of using AES-NI are visible when creating applications that use intensively cryptographic functions. Instead of doing a software routine the methods calls are made to the hardware level.

Choosing the right platform for implementing the security module depends on the following specifications:

- transparency
- portability
- performance
- known and documented vulnerabilities

As already described in the previous chapter, we chose to test the performance of the AES algorithm in two different implementations: one in Python and the other in C#. Python is a high-level object oriented programming widely used in the last years. It supports a lot of 3rd party libraries which makes it useful for a lot of applications, including the research areas. Its main disadvantage is that it is an interpreted language, which automatically tends to be slower than a compiled program. A plus for Python is that it has a lot of interpreters for various operating systems, which makes it portable. On the other hand, DOT NET C# is a very powerful programming language developed by Microsoft. It is a compiled language, which gives it better performance than Python. Also it uses the concept of *managed code*. This means that by default in DOT NET there is no need to use the pointers, memory allocation being made by internal mechanisms. From the security point of view this feature is a plus.

Net Core offers 3 sets of cryptographic algorithms:

- Hash algorithms
- Symmetric algorithms
- Asymmetric algorithms

Figure 32 illustrates the class diagram with all dependencies for *symmetric algorithms*. It is very important to emphasize that such kind of architecture are hierarchical architectures because this gives the possibility to easily extend the functionalities of a specific module.

The base class is *SymmetricAlgorithm* which is an abstract class. The *abstract* modifier indicates that the entity which will be implemented has a missing or an incomplete implementation. All members marked as *abstract* must be implemented by the non-abstract classes that derive from the abstract class.

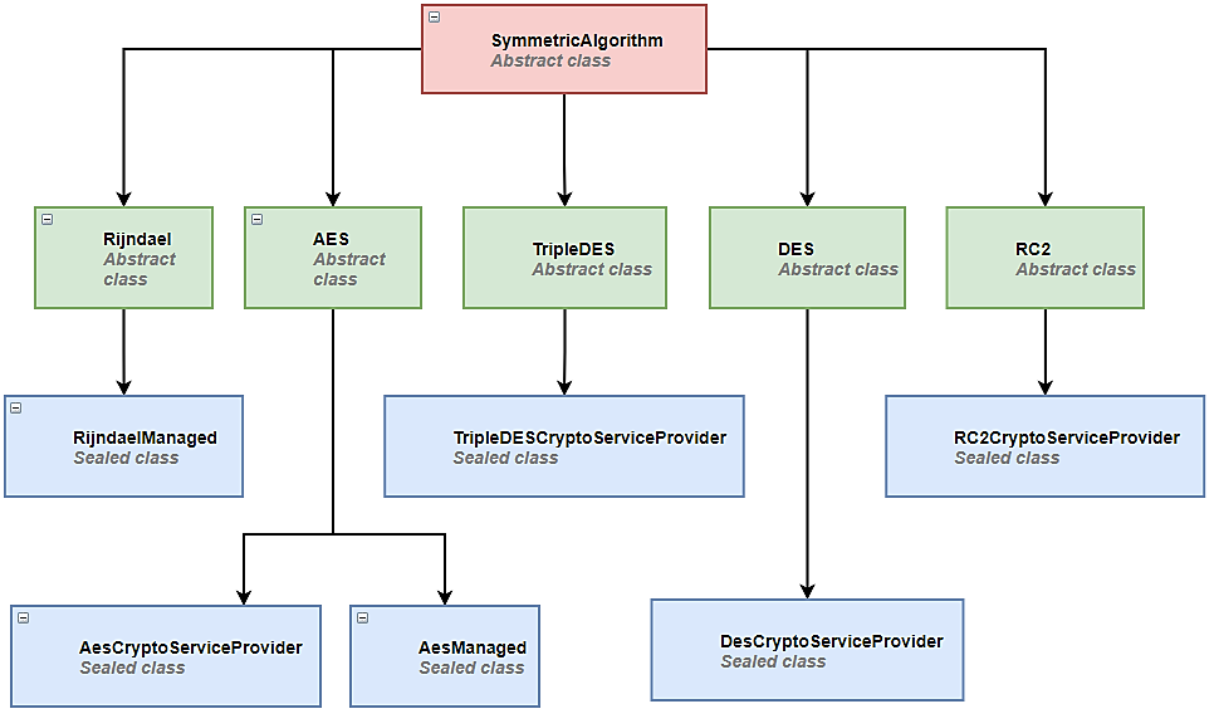

Figure 32: Symmetric Algorithms hierarchy

The *SymmetricAlgorithm* class has a predefined constructor, fields, properties and methods that will have to be implemented by all derived classes. The next level consists of a set of abstract classes which describes different symmetric algorithms.

AesManaged class performs symmetric cryptographic operations but without using the processor's crypto module.

AesCryptoServiceProvider class performs symmetric encryption and decryption using the Cryptographic Application Programming Interfaces (CAPI) implementation of the Advanced Encryption Standard (AES) algorithm.

Starting with Windows NT 4.0, Microsoft has generated a dedicated library for cryptographic functions, called CryptoAPI. Once Windows Vista was introduced, a new platform was developed, all cryptographic functions being placed into a new framework called CNG - Crypto Api Next Generation. The library which encapsulates all this functionality is Bcrypt.dll. Bcrypt.dll is able to run in both user mode and kernel mode.

In the Figure 33 it is shown a cryptographic process which uses *AesCryptoServiceProvider* class.

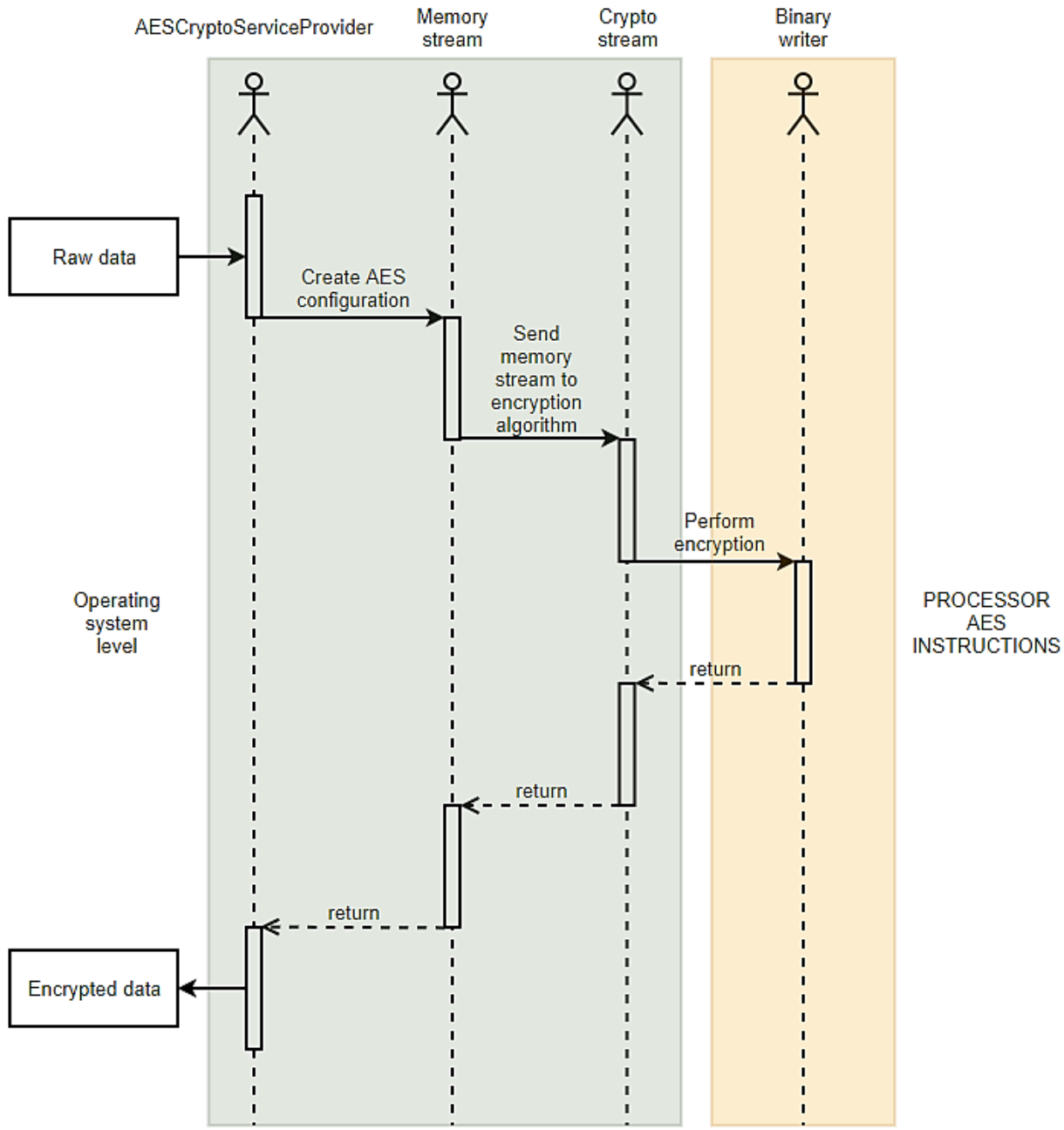

Figure 33: Cryptographic flow

The cryptographic flow has the following path:

- Raw data are loaded and passed to the AESCryptoServiceProvider which initializes the key, the initialization vector and configures the type of the padding, cipher mode or key size.
- Next the data is sent to the memory stream which is the fastest way of reading information.
- After that the date is passed to the crypto stream which receives as a parameter the memory stream.

- The last step is to call the binary writer which takes as argument the crypto stream. In this step the data is passed to the CPU FIPS validated crypto module, all cryptographic operation are performed and the result is written back to the RAM memory.

5.7 Key Security Concepts

Developers of applications that perform processes on important and even secret data must follow some key security concepts to ensure the security of metadata circulated in the system.

The first property to consider refers to the Type-safe code. This principle presupposes the constraint of the application in order to access only the data on which it has access authorization and cannot access private variables that are part of another object.

Verification is performed during the time of JIT compilation, when an optional verification process examines the metadata and Microsoft intermediate language (MSIL) of a method to be JIT-compiled into native machine code to verify that they are type safe.

Benefits of type-safe code:

- assemblies cannot adversely affect each other;
- increases application reliability;
- components are executed safely within a single process, it is not taken into account if it is trusted at different levels;
- Runtime's security enforcement mechanism access only the code on which it has permissions.

Another important principal is related to Principal, this represent the identity and role of a user and perform the actions on his behalf. At the current level there are 3 main principals:

- generic principals;
- Windows principals;
- custom principals.

The main verification steps of these principals have been performed by authentication process, verifying the user's credentials and validating them through an authority. The obtained data can be used directly in the program code.

5.8 Guidelines for a secured code

The core of an application is the source code written by the developers. Simultaneously, it is one of the tenuous links of a program, a weakness that can be exploited by a malicious person. The problem we are referring to is that certain vulnerabilities in the code can be discovered by an attacker and then exploited to obtain information that they should not gain access to. To prevent this, a number of rules have been developed that must be followed in order to remove vulnerabilities that may occur in the operation of the program. These rules apply to data protection, communications security, error handling, logging, and cryptographic practices.

Given that the security of the data used by the module is a key point for the designed model, it was decided to utilize the .NET framework, because it has protection mechanisms implemented directly in the source code, which can be used by extending those specific classes.

Being one of the largest frameworks used today, it receives regular updates that have the role of removing recently discovered vulnerabilities.

In order to implement applications based on the use of the infrastructure implemented by the .Net framework, Microsoft has developed a set of rules to be considered in the application implementation process. Following these rules can improve the level of trust in the application and reduce the possibility of being exploited by an attacker.

In the process of implementing the code, we must take into account a series of measures that we must follow:

1. Secured resource access. One of the most important vulnerabilities that exist, results from non-compliance with the access rules on the data with which the application works. To limit this vulnerability, Microsoft encourages the use of the following techniques (Microsoft 2020):

- Do not use Code Access Security (CAS).
- Do not use partially trusted code.
- Do not use the **AllowPartiallyTrustedCaller** attribute (APTCA).
- Do not use .NET Remoting.
- Do not use Distributed Component Object Model (DCOM).
- Do not use binary formatters.

In addition, to limit the access of unsecured programs to significant data, we can use one of the following alternative measures:

- Virtualization
- AppContainers
- Operating system (OS) users and permissions
- Hyper-V containers

2. Security-neutral code. Another way to reduce vulnerabilities is to use security-neutral code. APIs that are built on security-neutral code do not need permissions at running time, which increases the security of the data used by the program.

3. Not reusable component. If the source code is clustered in a single program and there is no risk of it being used from outside, the degree of safety increases, and the need to make certain improvements on the security side decreases.

4. Native code implementation. The use of managed wrappers is encouraged by Microsoft. They allow manipulating imported codes, performing a strict verification on them. Furthermore, the permissions of the imported code was inherited from the managed wrapper permissions.

5. Protected resources. In case our library offers access to certain resources of interest, checking the permissions of the code that accesses our library is mandatory. If this is not done properly, we may provide access to secure data to applications that do not have the necessary permissions to view that information.

6. Securing State Data. Another vulnerability often encountered in programs refers to the type of data existing in the internal memory of the application. Even if those data can be protected by declaring them private or internal, we must keep in mind a few rules (Microsoft 2017):

- Using reflection mechanisms, highly trusted code that can reference your object can get and set private members.
- Using serialization, highly trusted code can effectively get and set private members if it can access the corresponding data in the serialized form of the object.
- Under debugging, this data can be read.

7. Security and User Input. In the case of applications that allow the user to insert data into the program, they have a high exposure when parsing that data. This vulnerability occurs because the text received from the user is entered as a parameter in the functions of the applications. Being sent directly to functions, that metadata may contain malicious code. In order to remove this vulnerability, we can make some changes to the input:

- Escape the characters (Unicode escapes, Hexadecimal escapes);
- Remove the special characters;
- Constrain specific data entry fields.

8. Security and On-the-Fly Code Generation. Specific to libraries that generate and run code on runtime, there is the possibility to generate a "less-trust" code that will be run as a highly trusted one. This vulnerability can be avoided by strictly controlling the calls and metadata received in the library. The best way to remove this vulnerability is by generating the code with reflection emit.

5.9 Vulnerabilities and countermeasures

One of the biggest problems that occur when using encryption algorithms occurs when an attacker obtains encrypted data that is circulated between users, and he manages to decrypt the data without knowing the initial key that was used in the encryption process.

Microsoft considers that it is no longer safe to decrypt data encrypted with CBC mode of symmetric encryption if verifiable padding has been applied without first checking the integrity of the ciphertext.

One of the most well-known vulnerabilities that occur when using CBC mode for encryption is the so-called "padding oracle attack". This attack allows the attacker to decrypt the intercepted data, without knowing the key with which the encryption of the targeted data was performed in the first instance.

This oracle attack targets 2 important components existing in the encryption process: padding and the existence of a connection between consecutive blocks in the encryption process.

The existence of padding allows the attacker, with the help of a software implementation with oracle padding, to detect the padding used to encrypt the targeted message. This process involves performing consecutive steps that try to detect the correct size of the padding.

In addition, because it is based on the existence of connections between consecutive blocks that appeared after the encryption process, the CBC encryption mode generates a random data vector (IV) that is used in the encryption process of the first data block. Taking advantage of the existence of the data structure in CBC mode, combined with a padding oracle, by performing a large number of attempts, the attacker can decrypt a message byte by byte.

One method of protection against this type of attack is to implement a signature on the data to be sent, and the recipient of the message will first verify the signature using a key known only to the two users. This process is known as HMAC (keyed-hash message authentication code). In order not to produce another vulnerability related to the calculation of this signature, the HMAC calculation process must be performed in a constant time.

6 Conclusion

The results of research activities presented in this deliverable contain useful information to implement cryptographic security measures in the core of each device or terminal that is part of an IoT network. The research was conducted focusing on AES implementations on various platforms, hardware or software, highlighting the aspects which can improve performance or, otherwise, which can create vulnerabilities.

Having in mind all the aspects presented above, the choice of the software platform involved in application's development will be related to:

- Security strength of the programming language
- Execution speed
- Operating system's security and certifications according to security standards (Common Criteria and FIPS 140-2)
- Facilities to use cryptographic hardware (CPU internal components) based on software calls in order to perform encryption/decryption.

Windows' EAL4+ certification provided by Common Criteria analysis together with FIPS 140-2 evaluation for cryptographic libraries that communicate with the CPU allow this operating system to be used for development and running of a cryptographic application.

DOTNET Framework or DOTNET Core, which are wrappers over the operating system's libraries provide proper methods to access specific security functions from the operating system. Also all the programming languages offered by these frameworks (C#, C++/CLI, F#, J#, VB, etc.) are compiled languages so they are much faster than Python for example (which is an interpreted language). The ease-of-use for these tools is another important aspect which is taken into consideration. Moreover DOTNET Core is a cross platform which allows operation on different operating systems.

Also, the work present improvements of a tamper proof solution in the scope of minimizing side channel attack risks.

Not in the last, the same principle implemented in tamper proof solution was used, in an innovative manner, to generate random number streams. Being generated locally, inside of each device or network terminal, the random numbers thus generated are involved in cryptographic protocols or are used to expand the life time of the secret keys.

7 Appendix

Appendix 1 - Tests of Distinguishability from Random 200kB

SP800-22 tests were performed using the implementation presented in "A python implementation of the SP800-22 Rev 1a PRNG test suite" (https://github.com/dj-on-github/sp800_22_tests).

E:\LUCRU\PYTHON\SP800_22\sp800_22_tests-master>python -m sp800_22_tests.py rnd.bin

Tests of Distinguishability from Random

TEST: monobit_test

Ones count = 1061102

Zeroes count = 577298

FAIL

P=0.0

TEST: frequency_within_block_test

n = 1638400

N = 99

M = 16549

FAIL

P=0.0

TEST: runs_test

prop 0.647645263671875

tau 0.0015625

FAIL

P=0.0

TEST: longest_run_ones_in_a_block_test

n = 1638400

K = 6

M = 10000

N = 75

chi_sq = 852.6905259225296

FAIL

P=6.326085848635833e-181

TEST: binary_matrix_rank_test

Number of blocks 1600

Data bits used: 1638400

Data bits discarded: 0

Full Rank Count = 461

Full Rank -1 Count = 932

Remainder Count = 207

Chi-Square = 0.2869478317657266

PASS

P=0.8663433976951915

TEST: dft_test

N0 = 778240.000000

N1 = 792949.000000

FAIL

P=0.0

TEST: non_overlapping_template_matching_test

PASS

P=0.9904867968470971

TEST: overlapping_template_matching_test

B = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

m = 10

M = 1062

N = 968

K = 5

model = [352, 179, 134, 97, 68, 135]

v[j] = [8, 15, 33, 30, 35, 847]

chisq = 12555.624363408284

FAIL

P=0.0

TEST: maurers_universal_test

sum = 1336441.3361360026

fn = 5.741294613024493

FAIL

P=0.0

TEST: linear_complexity_test

M = 512

N = 3200

K = 6

chisq = 6.42070091130002

P = 0.3777477696067265

PASS

P=0.3777477696067265

TEST: serial_test

psi_sq_m = 648230.8836718751

psi_sq_mm1 = 467284.70205078134

psi_sq_mm2 = 290570.0715917968

delta1 = 180946.18162109377

delta2 = 4231.551162109245

P1 = 0.0

P2 = 0.0

FAIL

P=0.0

P=0.0

TEST: approximate_entropy_test

n = 1638400

m = 3

Pattern 1 of 8, count = 68658

Pattern 2 of 8, count = 121313

Pattern 3 of 8, count = 152965

Pattern 4 of 8, count = 234362

Pattern 5 of 8, count = 121313

Pattern 6 of 8, count = 266014

Pattern 7 of 8, count = 234362

Pattern 8 of 8, count = 439413

phi(3) = -4.246841

Pattern 1 of 16, count = 22745

Pattern 2 of 16, count = 45913

Pattern 3 of 16, count = 44125

Pattern 4 of 16, count = 77188

Pattern 5 of 16, count = 44846

Pattern 6 of 16, count = 108119

Pattern 7 of 16, count = 74324

Pattern 8 of 16, count = 160038

Pattern 9 of 16, count = 45913

Pattern 10 of 16, count = 75400

Pattern 11 of 16, count = 108840

Pattern 12 of 16, count = 157174

Pattern 13 of 16, count = 76467

Pattern 14 of 16, count = 157895

Pattern 15 of 16, count = 160038

Pattern 16 of 16, count = 279375

phi(3) = -4.893008

AppEn(3) = 0.646167

ChiSquare = 153943.93212476827

FAIL

P=0.0

TEST: cumulative_sums_test

FAIL: Data not random

FAIL

P=0.0

P=0.0

TEST: random_excursion_test

J=15

x = -4 chisq = 2.144357 p = 0.828831

x = -3 chisq = 3.000720 p = 0.699875

x = -2 chisq = 5.001500 p = 0.415697

x = -1 chisq = 14.998500 p = 0.010369

x = 1 chisq = 4.205338 p = 0.520247

x = 2 chisq = 3.642156 p = 0.601995

x = 3 chisq = 3.566630 p = 0.613330

x = 4 chisq = 4.801013 p = 0.440644

J too small (J < 500) for result to be reliable

PASS

P=0.828831423785823

P=0.6998748216021984

P=0.4156971521741977

P=0.010368749105646946

P=0.5202470850491279

P=0.6019950717407222

P=0.6133304972973969

P=0.4406444169768629

TEST: random_excursion_variant_test

J= 15

x = -9 count=0 p = 0.506555

x = -8 count=0 p = 0.479500

x = -7 count=0 p = 0.447521

x = -6 count=0 p = 0.408961

x = -5 count=0 p = 0.361310

x = -4 count=0 p = 0.300623

x = -3 count=0 p = 0.220671

x = -2 count=0 p = 0.113846

x = -1 count=10 p = 0.361310

x = 1 count=13 p = 0.715001

x = 2 count=9 p = 0.527089

x = 3 count=2 p = 0.288487

x = 4 count=2 p = 0.369673

x = 5 count=1 p = 0.394207

x = 6 count=1 p = 0.440900

x = 7 count=1 p = 0.478376

x = 8 count=1 p = 0.509275

x = 9 count=3 p = 0.595163

J too small (J=15 < 500) for result to be reliable

PASS

P=0.5065551690490404

P=0.4795001221869534

P=0.4475209101304692

P=0.40896134203687456

P=0.36131042852617884

P=0.30062298819690675

P=0.22067136191984682

P=0.11384629800665803

P=0.36131042852617884

P=0.7150006546880892

P=0.5270892568655381

P=0.2884874633234893

P=0.3696734409705522

P=0.3942069504679318

P=0.44089980988590066

P=0.47837563924795834

P=0.5092754371240833

P=0.5951631467095724

SUMMARY

---

monobit_test 0.0 FAIL

frequency_within_block_test 0.0 FAIL

runs_test 0.0 FAIL

longest_run_ones_in_a_block_test 6.326085848635833e-181 FAIL

binary_matrix_rank_test 0.8663433976951915 PASS

dft_test 0.0 FAIL

non_overlapping_template_matching_test 0.9904867968470971 PASS

overlapping_template_matching_test 0.0 FAIL

maurers_universal_test 0.0 FAIL

linear_complexity_test 0.3777477696067265 PASS

serial_test 0.0 FAIL

approximate_entropy_test 0.0 FAIL

cumulative_sums_test 0.0 FAIL

random_excursion_test 0.010368749105646946 PASS

random_excursion_variant_test 0.11384629800665803 PASS

8 List of Abbreviations

Abbreviation	Meaning
AES	Advanced Encryption Standard
AES-NI	Intel Advanced Encryption Standard New Instructions
AI	Artificial Intelligence
CMN	Conductive Mesh Network
IA	Intel Architecture
ICS	Industrial Control Systems
IoT	Internet of Things
IT	Information Technology
IT&C	Information and Communication Technology
LFSR	Linear – Feedback Shift Register
PKI	Public Key Infrastructure
SMM	Intel System Management Mode
PRNG	Pseudo Random Number Generator
TRNG	True Random Number Generator
UPB	University POLITEHNICA of Bucharest

9 References

Knuth, D.E. (1997): The Art of Computer Programming: Seminumerical Algorithms", Vol. 2, 3rd Edition, Pearson Education, Boston.

Microsoft 2017: [online] https://docs.microsoft.com/en-us/dotnet/standard/security/securing-state-data

Microsoft 2019a: [online] https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-platform-common-criteria

Microsoft 2019b: [online] https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation

Microsoft 2020: [online] https://docs.microsoft.com/en-us/dotnet/standard/security/secure-coding-guidelines#securing-resource-access

NIST, n.d.: National Institute of Standards and Technology – Special Publication 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

RealWorldCyberSecurity, 2020: Negative Rings in Intel Architecture: The Security Threats That You've Probably Never Heard Of, [online] https://medium.com/swlh/negative-rings-in-intel-architecture-the-security-threats-youve-probably-never-heard-of-d725a4b6f831

Rott, J. K. (2012): Intel® Advanced Encryption Standard Instructions (AES-NI), [online] https://software.intel.com/content/www/us/en/develop/articles/intel-advanced-encryption-standard-instructions-aes-ni.html

Russinovich, M.E., Solomon, D.A., Ionescu, A. (2012): Windows Internals, Part I, 6th Edition, Microsoft Press.

Shannon, C. (1945): classified report "A Mathematical Theory of Cryptography", [online] https://www.iacr.org/museum/shannon/shannon45.pdf

Thomson, I. 2015: Intel left a fascinating security flaw in its chips for 16 years – here's how to exploit it, in: The Register [online] https://www.theregister.com/2015/08/11/memory_hole_roots_intel_processors/.

Vasile, D.C, Svasta, P. (2019): Protecting the Secrets: Advanced Technique for Active Tamper Detection Systems, in SIITME 2019.