Cryptographic Techniques over a Wireless Micro-Controller Connection for Wireless Sensor Networks using nRF24L01+ Antenna

Spanos Georgios
Department of Digital Security
University of Piraeus
Department of Telecommunication
University of Malaga
Malaga, Spain
spaniakos@gmail.com

Abstract—**This paper proposes the usage of a new wireless antenna (nRF24L01+) in sensors along with cryptographic functions. In this research the antenna is used in mirco-controllers in order to simulate a Wireless Sensor Network along with cryptographic algorithms and techniques. The research aims to prove that the usage of this antenna will be reliable and feasible secure in such networks using micro-controllers as a base. There are many different libraries used and extensively enhanced so that the algorithm will ultimately provide a unified framework for micro-controllers to use with this antenna. Furthermore, the libraries are built against the standard micro-controller libraries and are following similar standards in the information technology field, like openssl for cryptography and OSI network layers for the communication. The algorithms are coded in a way to ensure further development as well as support for many different micro-controllers, architectures and specifications. Moreover, this research shows that the nRF24L01+ antenna modules can be used alongside with several micro-controllers efficiently and the cryptographic algorithms are robust and secure.**

Key Words— **Intel Galileo, Arduino, Raspberry pi, nRF24L01+, Micro-controller, Cryptography, Wireless Sensor Networks, WSN, X86, ARM.**

I. Introduction

In the new information era there is an increasing demand for sensors in many fields of commercial and non-commercial use. Moreover, along with this demand the security concerns and requirements from the consumers rise as well. In addition, the sensors are required to be cheap but reliable and efficient. Furthermore, they need to be appealing and enduring in order to be used in commercially. Finally, for the case that the sensor use batteries, it is required that the sensors should have low power consumptions.

Regarding those requirements, the first thought is to use Ethernet for the data communication, which is fast, reliable, secure and well established, but is not wireless, thus inconvenient and requires one cable per sensor which can be expensive in long distances.

Furthermore, in order to solve the disadvantages of the Ethernet solution, there is the standard Wireless solution, Wi-Fi. Wi-Fi has all the Ethernet advantages and in addition it has good encryption techniques which are well documented and secure. It provides good and reliable communication even at long distances using specific antennas. In the other hand, Wi-Fi has high power consumption and can deplete a sensor battery in just a few hours.

Considering the two well established solutions and the sensor requirements, there is one more solution, which should be developed and studied more in order to ensure its security, reliability and functionality. This solution is using nRF24L01+ antennas (1) which may be a good solution for wireless communication on sensors and specifically on Wireless Sensor Networks.

II. Motivation

The idea behind this research was initially originated from the need of low cost but power efficient wireless connection between micro-controllers such as Intel Galileo, Arduino, Raspberry pi and similar boards. Moreover, nowadays more people are concerned for their privacy and security over such networks, as said in section (I), so it`s a requirement for such networks to include security measures in there toolset. The first step towards this goal was to implement a wireless connection between such micro-controller using bleeding edge libraries to analyze their capabilities and note possible limitations, improvements and cryptographic primitives that can be used. Finally it was required to implement such improvements and cryptographic techniques, which should be compared with already existing libraries in similar systems in order to prove there usability.

III. Background

The cryptographic libraries that will be used, existing or not, will follow the openssl standards for the coding part, and will be tested against standard test vectors, such as FIPS from NIST (2). All cryptographic libraries and there origin are explained in the section (X).

On top of the standard cryptographic techniques that will be implemented, the communication will follow the Encrypt-then-MAC (3) implementation, which is proposed as the next standard, in order to avoid the weaknesses of the previous implementation (MAC-Then-Encrypt and Encrypt-and-MAC). Finally, HMAC will be used for every MAC module in order to ensure both the uniqueness of the mac for the similar packets, and the authenticity of the data.

In the area of wireless micro-controller communication there are several wireless modules to choose from. The most known modules are: Wi-Fi Shields, ZigBee, XBee, Nordic nRF24L01+ transceiver and Bluetooth. In this project the Nordic nRF24L01+ transceiver is selected as it is the one with the lowest cost that combines high speed communication, up to 2Mbps, and low power consumption, 1uA at standby mode. Moreover, there are bleeding edge libraries derived from community work that includes the basic nRF24L01+ library, a network layer and a mesh implementation. As noted above, the libraries and their origin are explained in the section (X).

IV. Limitations

The micro controllers have limited memory and processing power, thus the coding requires optimizations in order to work efficient. Moreover, the encryptions have to be fast and efficient, resulting to symmetric cryptographic schemes and leaving public key cryptography as a future work.

The majority of boards have embedded EEPROM, but this rom has limited read and writes. Some boards offer a guarantee for 100.000 erase/write cycle to the EEPROM slot. This cycle may seem large, but in networks that will have rapid developments and power downs, it may lead to node malfunction. Furthermore, The EEPROM data is saved as plaintext, therefore it may lead to node compromise if there is physical access.

The architecture in each micro controller may differ. Some controllers runs with ARM chipset while other, like Intel Galileo, function with X86. This difference broadens, as some micro controllers do not support AVR headers and pgmspace. This raises the complexity, as the program that will be created will have to support the majority of the official micro controllers with concern to the differences of each one.

Finally, the RF24L01+ antenna does not support more than 6 channels or communication, thus connections (1). This results to the need of implementing and more complex network scheme, like true Mesh networks.

V. Cryptographic primitives

The Base of the implementation coding is the cryptography that will be used. The implementation requires that End-to-End encryption should be used in order to ensure the secrecy of the messages. Moreover, the program has to ensure the Integrity of the messages and the Authenticity of the clients. In addition a key function is that there will not be a single message that will be transmitted unencrypted except the Initialization Vector (IV) that is required for the encryption modules.

Server-side, the primary concern is that the server should be available at all times to the clients, and should be able to produce and deliver network keys for every node at the beginning of the communication, specifically during the authentication process, and at required time intervals, thus the cryptographic classes have to be fast but efficient and secure. Furthermore, the program have to support as many cryptographic modules as possible, thus it have to handle all cryptographic modules as a “black box” in order to be easy to change the cryptography fast and without a lot of effort.

As noted before, the libraries are explained in the section (X).

VI. System Design

For the micro-controllers, there are several boards with a wide range of specification. The list includes, but is not limited to the following boards**:**

- More than 130 Arduino like boards
- 5 Raspberry pi boards
- Several boards with similar functionality.

The boards that were selected are Raspberry pi B+ for the server and Intel Galileo Gen 1 as Nodes. The selection was based on available hardware with consideration to compatibility with the rest of the controllers.

The wireless antenna selected is the nRF24L01+ wireless module described in section (III).

For the programming part, OOP (object oriented programming) was used for all classes while C++ and ANSI C were used as the programming languages. In addition each class is separated in at least three files for the code, plus some compile files if necessary. These compile files are mainly for the Raspberry pi. The three separate files include: the configuration file, the header file, and the method coding. The configuration file includes all the library calls and definitions. In addition, the configuration file includes definitions for different architectures and development boards in order for the library to be able to support different architectures and systems. The header file includes the class definition, methods and notes for the documentation, while the method coding includes all class methods and their code. Using this coding method, the program ensures easy debugging and further development. For the documentation of each class, a code parser is used. This parser is reading notes from the header file in order to generate html pages with the notes and explanations for each class.

The program except from the classes uses a handler for some libraries, like the encryption libraries, in order to be able to encrypt, decrypt, hash and verify the messages using different encryptions and hashing algorithms. Those algorithms are defined in the program configuration file and the nodes configuration structure that will be explained later.

There are two different types of message frames that the RF24 libraries are using. The first frame is the header frame and the seconds is the message frame.

The first frame, as shown in Figure VI-1, is 18bytes long. The first 4 bytes are the Node address that the message originates, the following 4 bytes are the address of the node that the message will be delivered. The next 4 bytes represents the sequential ID of the message (mainly used in order to avoid replay attacks). The subsequent 1 byte is the Message type header while the successive 1 byte is a reserved byte for future use. Finally the last 4 bytes are used for the next message id.

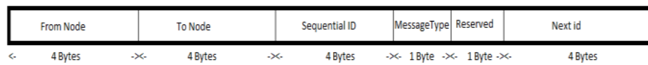

Figure VI-1Header frame

The second frame, as shown in Figure VI-2, has dynamic payload length. This length depends from the payload length plus the hash length and an 8 byte IV. As the nRF24l01+ antenna does not support frames larger than 32bytes in total (4), the library supports message assembly/disassembly for larger frames. That single frame is not going to be explained as it`s plain 32 bytes of data and is depended on the header type. For the frame of Figure VI-2, as the program uses Encrypt-then-MAC (3), the first X bytes are the encrypted payload while the following bytes hold the hash of the encrypted message using HMAC and the last 8 bytes are the IV in plaintext. The hash size is 16 or 20 or 32 bytes long, depending on the hashing algorithm used. The IV is in plaintext as all the encryption methods are used in CBC mode and the CBC mode requires the random IV to be known from both sides.

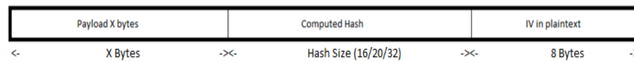

Figure VI-2Message frame

All these information about the encryption mechanism, hashing procedure and cryptographic keys are known to both server and client nodes because each node uses a Structure named NodeStuct that holds all these information. This NodeStuct differs from the client and the sever nodes.

The Client node Structure holds the information about the cryptographic algorithms, the hash functions, the encryption keys and the default encryption keys. For time being, the default encryption keys are hard-coded inside the nodes code. In addition, the NodeId and the re-key time interval is also hard coded.

The server node Structure is an array of NodeStuct that includes information about each node. The index for this array is the NodeID. This Array of structures contains information about the cryptographic algorithms of each node, the hash functions, the encryptions keys and the time interval for the re-key requests. All this information is requested from a local database during the authentication process. The data inside the database can be inserted or manipulated from the administrator. The default key is replaced from the random generated key during the re-key function but only inside the NodeStruct and not inside the database. The Re-key process is made using Encrypted messages with key the default node key that is stored in the database. In the case of continues non-authentication, the client node is requested to de-authenticate and wait for re-authentication in 30 minutes. In the case of continues message transmission with false password or with bad hash generally, the server cuts the node from the network requesting re-authentication. The above description of the authentication process can be seen visually in Figure VI-3.

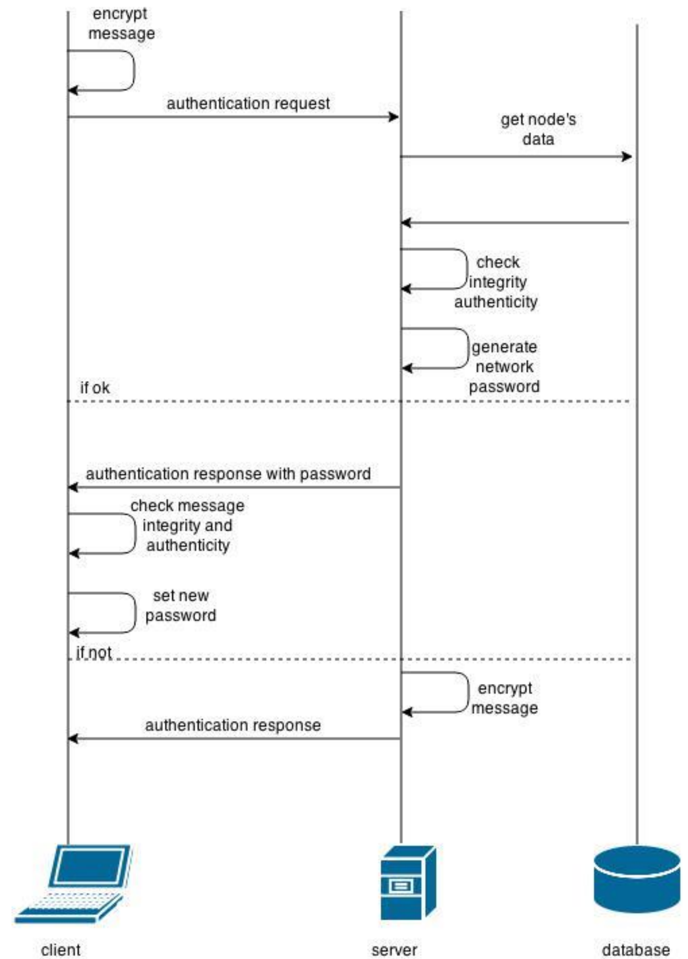

Figure VI-3Authentication process

The Figure IV-4 shows the information exchange process using direct transmission of the message from client to server or server to client. Initially the Node that sends the message encrypts and hashed the payload using Encrypt-then-MAC and HMAC functions. Following, it transmits the message using the wireless channel. The message as previously described has two frames, the header and the message frame. Both frames are transmitted. When both the header and the message frame are received, the receiving node acknowledges the sending node that the message has been received correctly. Consecutive, the receiving node takes appropriate actions depending from the

header frame. The first actions are always to check the integrity and authenticity of the messages. Depending on the outcome of the authenticity and the integrity check the receiving node continues with the appropriate actions.

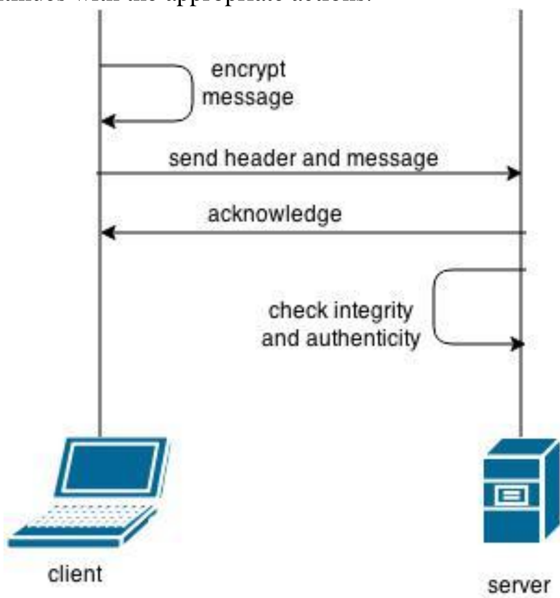

Figure VI-4 Information exchange process using direct message transmission

The Figure VI-5 shows the same process as above, but because the network can be a Mesh or a tree network, the information exchange process may include proxy clients from the sending to the receiving node. In this case the proxy client just proxies the messages without taking any action. It can be considered as a router that forwards the messages that are not for it.

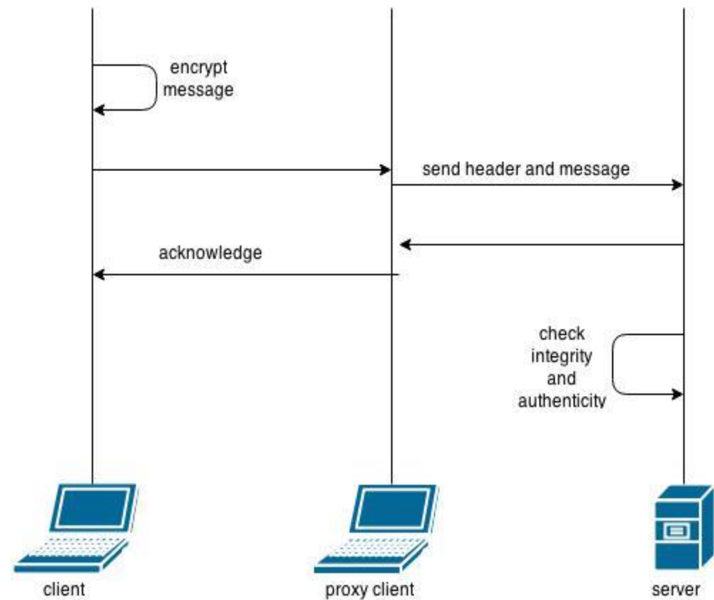

Figure VI-5Information exchange process using a proxy client

As previously described, the implementation can support various different types of network schemes. The basic network, using one node to each channel out of six that the antenna supports, is considered to be a star network. Derived from that and using the RF24Netowkr library, the network is expanded to a tree like implementation with sudo-mesh capabilities. The network is a tree network scheme but can support reconnection to the network using a different path if the parent node cannot be found.

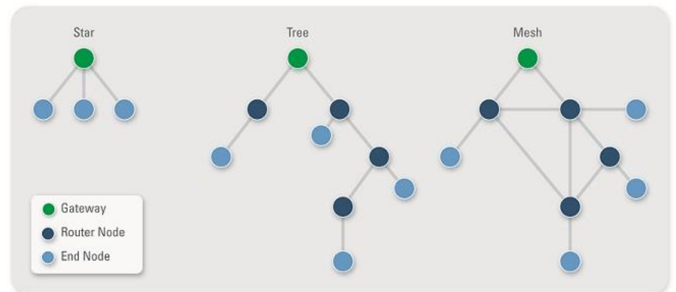

Figure VI-6Network explanation

As the three main communication libraries are mostly community driven, the primary focus of this research in the system design was to program the encryption libraries alongside with the network libraries in order to have an efficient system that will use the network libraries in an effective way and enhances the libraries in a way that the encryption methods will provide a unified framework for the network libraries to use.

For the coding of the cryptographic libraries, the openssl code was taken as a base, and the function were adapted in

order to be usable in micro-controllers. Furthermore, the classes were enchanted in a way that they can easily be used from the encryption handler.

After the coding of the libraries and the encryption handler layer, it was required to have some comparisons with this libraries using Raspberry pi and Intel Galileo with openssl and the custom framework, wherever was possible.

VII. COMPARISONS

There are three different comparison arrays for the cryptographic class. The first array makes a comparison between Raspberry pi and Intel Galileo encryption and decryption times using the same classes. The second compares the times in Raspberry pi between custom classes and the openssl framework; the openssl framework is measured using ANSI C functions. The third array compares the openssl framework in both Raspberry pi and Intel Galileo. In the Intel Galileo a simple linux OS was used, which is downloaded from Intel website; the openssl framework is measured using openssl speed function for both devices.

There are more than 10.000 measurements per module per table in order to have a good specimen to compare.

In the first table (VII-1) the top values are for Raspberry pi while the bottom values are for Intel Galileo. The values are measured in milliseconds.

Raspberry Pi **Intel Galileo**	**MIN**	**MAX**	**AVERAGE**	**MEDIAN**
3DES-CBC ENC	0,2930 1,3390	1,1510 1,8950	0,3084 1,3564	0,2970 1,3460
3DES-CBC DEC	0,5640 26,6900	1,1590 32,2000	0,5889 26,9378	0,5680 26,7300
AES-CBC ENC	0,0300 0,0410	0,4420 1,8930	0,0344 0,0501	0,0330 0,0440
AES-CBC DEC	0,0340 0,0520	0,1740 1,9080	0,0386 0,0613	0,0370 0,0550
MD5	0,0320 0,0180	0,6120 1,8220	0,0394 0,0278	0,0380 0,0200
SHA1	0,0260 0,0560	0,3040 1,6610	0,0385 0,0650	0,0420 0,0620
SHA256	0,0380 0,0670	0,2480 1,6940	0,0560 0,0782	0,0630 0,0690

Table VII-1Raspberry pi vs Intel Galileo using the framework

In the second table (VII-2) the top values are for the custom classes while the bottom values are for the openssl framework using ANSI C. All the measurements are taken using Raspberry pi. The values are measured in milliseconds.

Custom **Openssl**	**MIN**	**MAX**	**AVERAGE**	**MEDIAN**
3DES-CBC ENC	0,2930 0,0410	1,1510 0,3240	0,3084 0,0466	0,2970 0,0450
3DES-CBC DEC	0,5640 0,0410	1,1590 0,1800	0,5889 0,0461	0,5680 0,0450
AES-CBC ENC	0,0300 0,0180	0,4420 0,2690	0,0344 0,0206	0,0330 0,0200
AES-CBC DEC	0,0340 0,0250	0,1740 0,1870	0,0386 0,0286	0,0370 0,0270
MD5	0,0320 0,0290	0,6120 0,1570	0,0394 0,0323	0,0380 0,0310
SHA1	0,0260 0,0250	0,3040 0,3440	0,0385 0,0280	0,0420 0,0270
SHA256	0,0380 0,0390	0,2480 0,1760	0,0560 0,0429	0,0630 0,0420

Table VII-2Custom framework vs openssl in Raspberry pi

In the third table (VII-3) the top values are for Raspberry pi while the bottom values are for the Intel Galileo. All the measurements are taken using openssl speed function and the values are operations using x bit inputs where x is the header.

R. pi **Intel**	**16**	**64**	**256**	**1024**	**8192**
DES-CBC	1500320 313545	397488 81146	100734 20327	25264 5099	3150 632
AES-CBC	3138198 213357	898188 55665	233251 14064	58891 6007	7388 745
MD5	364478 81067	352552 72950	270153 56262	13980 1 30084	25185 5417
SHA 1	517674 73910	426695 59687	253254 36856	96750 15107	14247 2223
SHA 256	863934 104447	510689 57222	223078 24484	68733 7407	9219 985

Table VII-3Rasberry pi vs Intel Galileo using openssl

VIII. VALIDATION

The implementation and code have been extensively tested using various re-key intervals, various information retrieving intervals, with one client node and three client nodes. Moreover, there are several tests made using a mixture of the above settings.

The results were satisfying, as the majority of the messages were transmitted successfully inside the mesh network. The re-key function was working without any error except in the cases of re-key intervals setting lower than 5 seconds for each node. Therefore, the implementation has good functionality for time being.

The results for the cryptographic library and framework are also satisfying, and the majority of the cryptographic algorithms have similar execution time compared to the standards, except the DES and 3DES section, as shown in the previous section.

In general the encryption mechanism for the micro-controllers using nRFL01+ antennas is robust and considered secure, depending on the setting for the encryption algorithm

and the re-key intervals. Though, the communication is not DDOS resistant as the antennas cannot handle several messages and their buffers can be overflown. Finally as the antennas are working using 2.4GHz radio signals, the communication can be jammed using market jammers, but this flaws are known in the field of wireless communication (5).

The implementation is considered secure, but as stated in the following section (IX) there are a lot more to be implemented.

IX. FUTURE WORK

Considering that this system is not in a mature state but close to the initial state of development there are several aspects of functionality and usability to be added. This section is separated in two different sub-sections. General future work and specific future work in cryptography.

In the first sub-section it is recommended an implementation of a Command Line Interface (CLI) to be added. Moreover, a Dynamic NodeId system that won't extensively use the EEPROM should be developed in order to avoid static NodeId declaration inside the code of each respective node. Following, a configuration profile template should be created in order to be fairly easy to support more microcontrollers in the near future. It is also recommended the addition of sensor functionality. Finally there is a need to enhance the RF24Mesh library in order to function like a true mesh network instead of a tree and sudo-mesh network that is currently implemented. On top of that, it is mandatory to add sleep functionality to all non-AVR compatible devices like Intel Galileo, in order to optimize power consumption.

In the second sub-section, Cryptography, it is recommended to implement public key cryptography function using Elliptic curve cryptography (ECC) for 8-bit micro-controllers. Moreover the key-handing function should be enhanced in order to use Diffie-hellman key exchange; provided that, will eliminate the need to have hard coded encryption keys in the sensor code. Finally all the encryption and hash classes should be encapsulated into a single framework, like openssl, in order to have a unified library for the encryption in the micro-controllers. Furthermore, this framework should be optimized using inline assembly wherever possible in order to achieve lower encryption-decryption times.

X. LIBRARIES

All the libraries used except the main program, are listed in this section. The classes are of three types:

a) Forked and extended to use in Intel Galileo

b) Forked and extensively enchanted in order to help usability and efficiency.

c) Created from scratch.

All the libraries are Arduino, Intel Galileo and Raspberry pi compatible unless noted differently.

In the first section (a), the Libraries are:

RF24 (6)	This is the underlying network driver.
RF24Network (4)	This Library resembles the OSI network layer
RF24Mesh (7)	This library is used to enhance the Network layer in order to create a dynamic mesh layer.

Table X-1Forked and extended

In the second section (b) the libraries are:

Cryptosuite (8)	Implements Sha, Sha25, HMAC-Sha, HMAC-sha256.
ArduinoDES (9)	Implements DES, 3DES in CBC mode.
ArduinoMD5 (10)	Implements MD5, HMAC-MD5.
AES (11)	AES in CBC mode, 128, 192, 256 bits

Table X-2 Forked and extensively enchanted

In the Third section (c) the libraries are:

TempSensor (12)	This library uses the Temperature sensor from the seed kit or any TTC03 Thermistor and creates a usability layer. The library is Arduino and Intel Galileo compatible.
LightSensor (13)	This library uses the Light sensor from the seed kit or any GL5528 LDR sensor and creates a usability layer. The library is Arduino and Intel Galileo compatible.
LCD (14)	This library inherits the LCD screen class of the Arduino seed kit and creates a layer with usability functions and ease of usage for the LCD screen The library is Arduino and Intel Galileo compatible.
KEYGEN (15)	This library is a key generation algorithm with ease of use in generation of passwords and cryptographic IV.

Table X-3Created from scratch

XI. ACKNOWLEDGMENT

I would like to acknowledge the University of Malaga for hosting me as an Erasmus student. Also my professor in University of Piraeus, mr. K. Lamprinoudakis for recommended me for this master thesis. My professor in Spain Mr. Javier Lopez for making me a part of his research team and his assistant professor Isaac Agudo Ruiz for giving me all the materials needed and guided me though the whole thesis. Moreover I would like to acknowledge the Erasmus+ program for funding me as a master Erasmus student and made this option available to me. In addition I would like to acknowledge the rest of the research group that helped me with my assignment and made me feel a part of the team. I would like to acknowledge the contributors for designing and updating the RF24 (6) driver, the RF24Network layer library (4) and the RF24Mesh library (7) and all other libraries used in this Project. Finally I would like to acknowledge the Intel Corporation for Granting the University of Malaga the hardware needed in order to complete this thesis, such as Intel Galileo boards and Arduino Seed kits.

XII. WORKS CITED

1. **semiconductor, Nordic.** nRF24L01 - 2.4Ghz RF Products - Nordic Semiconductor. *Ultra Low Powered Wireless solutions from NORDIC SEMICODUCTOR.* [Online] july 2007. http://www.nordicsemi.com/eng/content/download/2730/34105/file/nRF24L01_Product_Specification_v2_0.pdf.

2. **NIST.** NIST Computer Security Publications - FIPS (Federal Information Processing Standards). *National Institute of Standards and Technology.* [Online] http://csrc.nist.gov/publications/PubsFIPS.html.

3. **Gutmann, P.** Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). *information on RFC 7366.* [Online] September 2014. http://www.rfc-editor.org/info/rfc7366.

4. **TMRh20.** TMRh20/RF24Netowork at Development. *Github.* [Online] https://github.com/TMRh20/RF24Network/tree/Development.

5. **Coleman D., Diener N.** Protecting WiFi Networks from Layer 1 Security Threats. *cisco.* [Online] 2007. http://www.cisco.com/c/dam/en/us/products/collateral/wireless/spectrum-expert-wi-fi/prod_white_paper0900aecd807395b9.pdf.

6. **TMRh20.** TmRh20/RF24. *Github.* [Online] https://github.com/TMRh20/RF24.

7. —. TMRh20/RF24Mesh. *Github.* [Online] https://github.com/TMRh20/RF24Mesh.

8. **Spanos, Georgios.** spaniakos/Cryptosuite. *Github.* [Online] https://github.com/spaniakos/Cryptosuite.

9. —. spaniakos/ArduinoDES. *Github.* [Online] https://github.com/spaniakos/ArduinoDES.

10. —. spaniakos/ArduinoMD5. *Github.* [Online] https://github.com/spaniakos/ArduinoMD5.

11. —. spaniakos/AES. *Github.* [Online] https://github.com/spaniakos/AES.

12. —. spaniakos/Tempsensor. *Github.* [Online] https://github.com/spaniakos/TempSensor.

13. —. spaniakos/Lightsensor. *Github.* [Online] https://github.com/spaniakos/LightSensor.

14. —. spanaikos/LCD. *Github.* [Online] https://github.com/spaniakos/LCD.

15. —. spaniakos/KEYGEN. *Github.* [Online] https://github.com/spaniakos/KEYGEN/.