ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Master

Experimental performance evaluation of IoT security solution

Evaluación de prestaciones de soluciones de seguridad en Internet de las Cosas en un entorno experimental

AUTOR: Jose Manuel Martínez Caro

DIRECTOR: María Dolores Cano Baños

June 2016

Siempre parece imposible hasta que se hace

Nelson Mandela

Me gustaría agradecer el trabajo encomiable llevado a cabo por María Dolores Cano Baños para la consecución de este trabajo, el cual, supone el objetivo final de los estudios de Master.

Para mí, la familia siempre ha sido y será un pilar fundamental en mi vida. Me gustaría reconocer la labor de Isabel Caro y Encarnación García, las cuales, me han modelado como persona tal y como un alfarero da forma al barro para crear tan singulares piezas.

No se puede pasar tampoco a aquellas personas que te arropan al pasar por el camino y que pretenden acompañarte hasta el final. Por ello, agradecer a Isabel Martínez su apoyo y su colaboración para la realización de este proyecto.

Por suerte, la universidad me ha permitido conocer a grandes amigos, los cuales echaré en falta cuando ya no los tenga tan cerca día tras día. Gracias de forma especial a Rubén por ayudarme cuando tenía algún problema.

Resumen

Internet de las Cosas (*Internet of Things*, IoT) plantea dudas importantes e introduce nuevos retos en la privacidad de las personas y en la seguridad de los sistemas y de los procesos. Algunas aplicaciones de IoT están estrechamente vinculadas a infraestructuras sensibles y servicios estratégicos tales como la distribución de agua y electricidad. Otras aplicaciones manejan información sensible acerca de las personas, tales como su ubicación y sus desplazamientos, o sus preferencias de salud o de compra. La confianza y la aceptación de los servicios y sistemas creados sobre la base de IoT dependerán de la protección que ofrezca a la intimidad de las personas y a los niveles de seguridad que garantice a los sistemas y procesos. Del mismo modo, IoT permitirá a cualquier objeto convertirse en un participante activo: estos objetos serán capaces de reconocer los acontecimientos y los cambios en su entorno y detectar y reaccionar de forma autónoma, sin intervención humana. La introducción de objetos en los procesos de control hace que la seguridad en IoT sea difícil de abordar. De hecho, IoT es un sistema complejo en el que las personas interactúan con el ecosistema tecnológico basado en objetos inteligentes a través de procesos complejos. Las interacciones de estos cuatro componentes de IoT: personas, objetos inteligentes, ecosistema tecnológico y procesos resaltan una dimensión sistémica y cognitiva en la seguridad de IoT. La interacción de las personas con el ecosistema tecnológico requiere la protección de su privacidad. Del mismo modo, su interacción con los procesos de control requiere garantizar su seguridad ya que los procesos deben garantizar su fiabilidad y dirigirse hacia los objetivos para los que fueron diseñados. El objetivo de este Trabajo Fin de Master es el diseño, la implementación y el uso de un testbed para pruebas experimentales en el que medir las prestaciones de soluciones de seguridad para Internet de las Cosas.

Palabras clave: Internet de las cosas, seguridad, AES, tinyOS, nesC, redes de sensores inalámbricas, prestaciones y testbed.

Abstract

Internet of Things (IoT) proposes important questions, introduces new challenges about people privacy and security in processes and systems. Some IoT applications are closely linked to critical structures and strategic services as or electricity water distribution. Other applications handle critical information about people, such as localization or movements, health or shop preferences. The confidence and acceptance of services and systems built over IoT will depend of the people's intimacy and the security levels. IoT will let to any object become an active: these objects will recognize events and changes in the environment, detecting and responding autonomously, without human intervention. The object introduction in the control processes makes the IoT security difficult to approach. IoT is a complex system where individuals interact with technologic ecosystem based in intelligent objects through complex processes. There are four components that interact: people, intelligent objects, technologic ecosystem and processes that emphasize a systematic and cognitive dimension in IoT security. People interaction with technologic ecosystem requires privacy protection. In addition, the interaction with control processes requires guarantee the security because processes must ensure reliability and be focus to the goals that were designed. The main objective of this Final Master Thesis is the design, implementation and the use of testbed to experimental test and measure performance solutions about security for Internet of Things.

Keywords: Internet of Things, security, AES, nesC, Wireless Sensor Netwoks, performance and tesbed.

List of Contents

Chapter 1 Introduction and objectives **1**
- **1.1. Introduction** **1**
- **1.2. Objectives** **4**
- **1.3. Staff involved** **4**
- **1.4. Memory structure** **4**

Chapter 2 : Wireless Sensor Networks **5**
- **2.1. TinyOS** **5**
  - 2.1.1. Introduction 5
  - 2.1.2. nesC 5
  - 2.1.3. TinyOS tutorial. 7
- **2.2. 802.15.4** **8**
  - 2.2.1. Introduction 8
  - 2.2.2. Network layer 8
  - 2.2.3. Data link layer (DLL) 9
  - 2.2.4. Physical layer 12
  - 2.2.5. Transmission and reception channels 12
  - 2.2.6. Information packet structure 13
  - 2.2.7. Modulation 13
  - 2.2.8. Sensitivity and range 14
  - 2.2.9. Interferences 14
  - 2.2.10. Average power consumption 14
- **2.3. Devices** **15**
  - 2.3.1. Introduction 15
  - 2.3.2. MICAz 15
  - 2.3.3. TelosB 16
  - 2.3.4. Imote2 17
  - 2.3.5. EPIC [15] 18
- **2.4. AES [16]** **19**
  - 2.4.1. Introduction 19
  - 2.4.2. Detailed Structure 21
  - 2.4.3. AES Transformation Functions 22
    - 2.4.3.1. Substitute Bytes Transformation 23
    - 2.4.3.2. ShiftRows Transformation 23
    - 2.4.3.3. MixColumns Transformation 24
    - 2.4.3.4. AddroundKey Transformation 25
  - 2.4.4. AES Key Expansion 26
  - 2.4.5. AES Modes 27
    - 2.4.5.1. Introduction 27
    - 2.4.5.2. Electronic Code Book (ECB) 27
    - 2.4.5.3. Cipher Feedback Mode 28
    - 2.4.5.4. Output Feedback Mode 29
- **2.5. Avrora** **30**
  - 2.5.1. Avrora Utilities 31

Chapter 3 : Implementation **33**
- **3.1. Introduction** **33**

3.2. Installing TinyOS **33**
3.3. Basic Programs **34**
3.3.1. Blink 34
3.3.2. BlinkToRadio 36
3.3.3. Sense 39
3.3.4. Oscilloscope 40
3.3.5. BaseStation 44
3.3.6. TestPrintf 49
3.4. Developed Applications **50**
3.4.1. Introduction 50
3.4.2. BlinkSemaforo 50
3.4.3. BlinkToRadioInv 51
3.4.4. SenseRadio 54
3.4.5. Common Code [Encrypt and Decrypt] 57
3.4.6. EncryptRadioECB 62
3.4.7. DecryptRadioECB 64
3.4.8. EncryptRadioCFB 68
3.4.9. DecryptRadioCFB 73
3.4.10. EncryptRadioOFB 77
3.4.11. DecryptRadioOFB 81
Chapter 4 :Results **87**
4.1. Blink **87**
4.2. BlinkSemaforo **89**
4.3. BlinkToRadioInv **92**
4.4. SenseRadio **96**
4.5. AES ECB **98**
4.6. AES CFB **103**
4.7. AES OFB **108**
Conclusion **113**
References **116**
Annexed I: Artículo AJICT **117**
Annexed II: OscilloscopeC.nc Code **123**
Annexed III: BaseStationP.nc Code **125**
Annexed IV: EncryptRadioECBC.nc Code **130**
Annexed V: DecryptRadioECBC.nc Code **136**
Annexed VI: EncryptRadioCFBC.nc Code **143**
Annexed VII: DecryptRadioCFBC.nc Code **150**
Annexed VIII: EncryptRadioOFBC.nc Code **157**
Annexed IX: DecryptRadioOFBC.nc Code **164**

List of Figures

Figure 1: Wireless Sensor/Actuator Network 1

Figure 2: nesC program application example with different software components 7

Figure 3: Network topology (Star and Peer-to-Peer) 9

Figure 4: Layer distribution on 802.15.4 9

Figure 5: MAC frame format 10

Figure 6: Super-frame structure 11

Figure 7: Slotted CSMA/CA 11

Figure 8: Un-slotted CSMA/CA 11

Figure 9: Channel structure on IEEE 802.15.14 13

Figure 10: PHY Packet 13

Figure 11: Average power over sleep/active duty cycle 15

Figure 12: MICAz device 16

Figure 13: MICAz architecture 16

Figure 14: TelosB device 17

Figure 15: TelosB architecture 17

Figure 16: Imote2 device 18

Figure 17: Imote2 architecture 18

Figure 18: EPIC mote device 19

Figure 19: AES Encryption Process 20

Figure 20: AES Data Structures 20

Figure 21: AES Encryption and Decryption 21

Figure 22: AES Encryption Round 21

Figure 23: AES Byte-Level Operation 23

Figure 24: AES Row and Column Operations 24

Figure 25: AES MixColumns Matrix 24

Figure 26: MixColumns Operations 24

Figure 27: MixColumns Operations II 25

Figure 28: AES MixColumns Inverse Matrix 25
Figure 29: AES MixColumns Operation Demonstrations 25
Figure 30: Inputs for Single AES Round 26
Figure 31: Electronic Codebook (ECB) Mode 28
Figure 32: s-bit Cipher Feedback (CFB) Mode 29
Figure 33: Output Feedback (OFB) Mode 30
Figure 34: BlinkAppC.nc file 35
Figure 35: BlinkC.nc file 36
Figure 36: Blink Makefile 36
Figure 37:BlinkToRadio.h file 36
Figure 38: BlinkToRadioAppC.nc file 37
Figure 39: BlinkToRadioC.nc file 38
Figure 40: BlinkToRadio Makefile 39
Figure 41: SenseAppC.nc file 39
Figure 42: SenseC.nc file 40
Figure 43: Sense Makefile 40
Figure 44: oscilloscope.py file 41
Figure 45: Oscilloscope.h file 41
Figure 46: OscilloscopeAppC.nc file 42
Figure 47: OscilloscopeC.nc file I 42
Figure 48: OscilloscopeC.nc file II 43
Figure 49: OscilloscopeC.nc file III 44
Figure 50: Oscilloscope Makefile 44
Figure 51: Java Oscilloscope application 44
Figure 52: BaseStationC.nc file 45
Figure 53: BaseStationP.nc file I 46
Figure 54: BaseStationP.nc file II 46
Figure 55:BaseStationP.nc file III: uartSendTask() 47
Figure 56: BaseStationP.nc file IV 47

Figure 57:BaseStationP.nc file V 48
Figure 58: BaseStationP file VI 48
Figure 59: BaseStation Makefile 48
Figure 60: TestPrintfAppC.nc file 49
Figure 61: TestPrintfC.nc file 49
Figure 62: Command to print program replies 50
Figure 63: TestPrintf Makefile 50
Figure 64: BlinkSemaforoAppC.nc file 50
Figure 65: BlinkSemaforoC.nc file 51
Figure 66: BlinkSemaforo Makefile 51
Figure 67: BlinkToRadioInv.h file 52
Figure 68: BlinkToRadioInvAppC.nc file 52
Figure 69: BlinkToRadioInvC.nc file 54
Figure 70: BlinkToRadioInv Makefile 54
Figure 71: SenseRadio.h file 54
Figure 72: SenseRadioAppC.nc file 55
Figure 73: SenseRadioC.nc file 56
Figure 74: Command to print program replies 57
Figure 75: SenseRadio Makefile 57
Figure 76: Common methods defined 58
Figure 77: Key expansion process 59
Figure 78: Initial XOR operation 60
Figure 79: ten-iteration loop (Substitute bytes, ShiftRows, MixColumns and AddRoundKey) . 61
Figure 80: Final iteration operation 62
Figure 81: EncryptRadioECB.h file 62
Figure 82: EncryptRadioECBAppC.nc file 63
Figure 83: Variables defined [DecryptRadioECB] 63
Figure 84: Timer is over. Time to send the message [EncryptRadioECB] 64
Figure 85: Message is received via Radio interface [EncryptRadioECB] 64

Figure 86: EncryptRadioECB Makefile 64
Figure 87: DecryptRadioECB.h 64
Figure 88: DecryptRadioECBAppC.nc 65
Figure 89: Variables declarations [DecryptRadioECB] 66
Figure 90: Receiving process message [DecryptRadioECB] 67
Figure 91: DecryptRadioECB Makefile 68
Figure 92: EncryptRadioCFB.h file 68
Figure 93: EncryptRadioCFBAppC.nc file 69
Figure 94: Variables declared [EncryptRadioCFB] 69
Figure 95: Timer is over, time to send one message [EncryptRadioCFB] 71
Figure 96: Specific characteristics of application [EncryptRadioCFB] 72
Figure 97: Mote receive a message [EncryptRadioCFB] 72
Figure 98:EncryptRadioCFB Makefile 72
Figure 99: DecryptRadioCFB.h file 73
Figure 100: DecryptRadioCFBAppC.nc file 73
Figure 101: Variables declared for DecryptRadioCFB 74
Figure 102: Decrypt proccess when a mote receive a message via Radio[DecryptRadioCFB].. 75
Figure 103: Especific characteristics of application [DecryptRadioCFB] 76
Figure 104: DecryptRadioCFB Makefile 76
Figure 105: EncryptRadioOFB.h file 77
Figure 106: EncryptRadioOFBAppC.nc file 78
Figure 107: Variables declared [EncryptRadioOFB] 78
Figure 108: Time is over. Time to send a message [EncryptRadioOFB] 79
Figure 109: Specific characteristics of application [EncryptRadioOFB] 80
Figure 110: Specific task when a mote receive a message[EncryptRadioOFB] 81
Figure 111: EncryptRadioOFB Makefile 81
Figure 112: DecryptRadioOFB.h file 81
Figure 113: DecrtptRadioOFBAppC.nc file 82
Figure 114: Variables declare in DecryptRadioOFB 82

Figure 115: Specific task when a mote receives a message [DecryptRadioOFB] 84

Figure 116: Specific characteristics of application [DecryptRadioOFB]................................... 85

Figure 117: DecryptRadioOFB Makefile.. 85

Figure 118: Blink compilation reply 87

Figure 119: MICAz Leds state [Blink] 88

Figure 120: Total accumulated energy consumption [Blink]... 88

Figure 121: Total energy consumption J [Blink] 89

Figure 122: Total energy consumption W [Blink] 89

Figure 123: BlinkSemaforo compilation reply... 90

Figure 124: MICAz Leds state [BlinkSemaforo] 90

Figure 125:Total accumulated energy consumption (J) [BlinkSemaforo] 91

Figure 126: Total energy consumption J [BlinkSemaforo].. 91

Figure 127: Total energy consumption W [BlinkSemaforo].. 91

Figure 128: BlinkToRadio.h file 92

Figure 129:BlinkToRadioInv compilation reply... 92

Figure 130: 802.15.4 frame [BlinkToRadioInv] 93

Figure 131: MICAz Leds state [BlinkToRadioInv] 94

Figure 132: Total accumulated energy consumption (J) [BlinkToRadioInv] 95

Figure 133: Total energy consumption J [BlinkToRadioInv] 95

Figure 134: Total energy consumption W [BlinkToRadioInv] 95

Figure 135: SenseRadio compilation reply 96

Figure 136: 802.15.4 frame [SenseRadio] 96

Figure 137: Total accumulated energy consumption (J) [SenseRadio]...................................... 98

Figure 138: Total energy consumption J [SenseRadio] 98

Figure 139: Total energy consumption W [SenseRadio] 98

Figure 140: AES online calculator [ECB].. 99

Figure 141: EncryptRadioECB compilation reply 99

Figure 142: DecryptRadioECB compilation reply.. 100

Figure 143: 802.15.4 cipher frame [ECB].. 100

Figure 144: ECB Results 101
Figure 145: Total accumulated Decrypt energy consumption (J) [EncryptRadioECB]............ 102
Figure 146: Total accumulated Encrypt energy consumption (J) [DecryptRadioECB]............ 102
Figure 147: Total accumulated energy consumption (J) [ECB]... 102
Figure 148: Total energy consumption J [ECB] 103
Figure 149: Total energy consumption W [ECB] 103
Figure 150: AES online calculator [CFB].. 104
Figure 151: EncryptRadioCFB compilation reply 104
Figure 152: DecryptRadioCFB compilation reply 105
Figure 153: CFB Results 105
Figure 154: Total accumulated Decrypt energy consumption (J) [DecryptRadioCFB]............ 106
Figure 155: Total accumulated Encrypt energy consumption (J) [EncryptRadioCFB]............ 106
Figure 156: Total accumulated energy consumption (J) [CFB]... 107
Figure 157: Total energy consumption J [CFB].. 107
Figure 158: Total energy consumption W [CFB] 107
Figure 159: AES online calculator [OFB]... 108
Figure 160: EncryptRadioOFB compilation reply 109
Figure 161: DecryptRadioOFB compilation reply.. 109
Figure 162: OFB Results... 110
Figure 163: Total accumulated Decrypt energy consumption (J) [DecryptRadioOFB] 111
Figure 164: Total accumulated Decrypt energy consumption (J) [EncryptRadioOFS]............ 111
Figure 165: Total accumulated energy consumption (J) [OFB]... 111
Figure 166: Total energy consumption J [OFB] 112
Figure 167: Total energy consumption W [OFB] 112

List of Tables

Table 1: IEEE 802.15.4 properties ... 8

Table 2: Modulation parameters ... 14

Table 3: Platforms Comparative. ... 15

Table 4: AES Parameters ... 21

Table 5: AES S-box ... 23

Table 6: AES Inverse S-box ... 23

Table 7: RC[j] Table ... 26

Table 8: AES Calculated Round ... 27

Table 9: AES Comparative Modes ... 27

Table 10: ECB Operations ... 27

Table 11: CFB Operations ... 29

Table 12: OFB Operations ... 29

Table 13: Blink Energy Consumption ... 87

Table 14: BlinkSemaforo Energy Consumption ... 90

Table 15: BlinkToRadioInv Energy Consumption ... 94

Table 16: SenseRadio Energy Consumption ... 97

Table 17: ECB mode Energy Consumption ... 101

Table 18: CFB mode Energy Consumption ... 106

Table 19: OFB mode Energy Consumption ... 110

Table 20: Simulation measures ... 113

Chapter 1 Introduction and objectives.

1.1. Introduction

Internet of Things (IoT) is a new concept where everything, from home appliances to wearables or vehicles will be connected to Internet. It is expected that the new digital ecosystem created by IoT will be the biggest on Information and Communication Technologies (ICT). Several examples of IoT: refrigerator, which knows the remaining food and purchase at the supermarket depending the user plans and stock; wearables, that can be connected to Internet to register all captured data from the user habits; improvements on vehicles driving systems; and home automation, to be more intelligent and energy efficient.

In this content, security arises as one of the main challenges to solve [1]. One logic method to analyze security on IoT is look at existing solutions established to Wireless Sensor Networks (WSN). WSN is designed to create a network with several motes, where having a continuous communication is possible to exchange messages with obtained information from the sensors, take decisions and/or perform simple operations/actions based on the information obtained (Figure 1). These motes can be commercially purchased from several trades. There are several problems and threats on WSN as Denial-of-Service (DoS) [2], Attacks on Information in transit, Sybil Attack, etc. In spite of the fact that several of these problems have been solved, important topics still remain open such as source authentication, cryptography keys establishment, etc. [1]

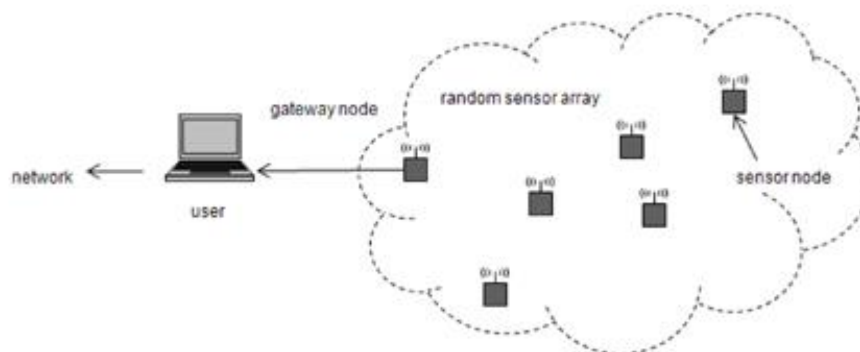

Figure 1: Wireless Sensor/Actuator Network

Since they were designed for low traffic monitor and control applications, it is not necessary for them to support the high data throughput requirements that data networks like Wi-Fi require. Typical WSN over-the-air data rates range from 20kbps to 1Mbps. Consequently, they can operate with much lower power consumption, which in turn allows the motes to be battery powered [3]. Many motes are available today with sub-1µA sleep current. As sleep mode current is important, low power in active mode and processing speed are as also important. Motes must have the ability to wake up quickly, have the processing power to rapidly execute the intended task, which includes processing through the communications protocol, and return to sleep mode in as little time as possible, minimizing the time spent in active mode.

WSNs are typically self-organizing and self-healing. Self-organizing networks allow joining automatically a new mote to the network without the need for manual intervention. Self-

healing networks allow motes to reconfigure their link associations and find alternative pathways around failed or powered-down motes. How these capabilities are implemented is specific to the network management protocol and the network topology, and ultimately will determine the network's flexibility, scalability, cost and performance [3].

The proliferation of 'smart' energy management applications and the abundance of inexpensive, standards-based wireless Microcontroller Units (MCUs) called also motes, are stimulating the growth of WSN across diverse markets, including home and building automation, telemedicine, or lighting, among others. WSN market, will up to $1.8B by 2019 [3]. WSANs provide a simple, economic approach for the deployment of distributed monitor and control devices, avoiding the expensive retrofit necessary in wired systems. A WSN is a collection of small randomly dispersed devices that provide three essential abilities:

- Monitor physical and environmental conditions, often in real time, such as temperature, pressure, light and humidity.
- Operate devices such as switches, motors or actuators that control those conditions.
- Provide efficient, reliable communications via a wireless network. The implementation of this last capability is the most unique to WSNs.

Another important part in this project is TinyOS. TinyOS is an open-source Operative System (OS). It is not an OS for general purpose, it is designed for WSNs. IT is an OS with a component-based architecture and is supported by different platforms as Linux, Windows (using Cygwin), and Macintosh. The programs on this OS need to have the following characteristics [4]:

- Small physical size and low power consumption because are battery powered.
- Concurrency-intensive operation.
- Limited physical parallelism and controller hierarchy.
- Diversity in design and usage.
- Robust operation.

To operate with motes is necessary to build applications. This is possible thanks to the Network Embedded System C (nesC), which is a C extension specially developed to WSN and it works on TinyOS. The main features are [5]:

- Separation of construction and composition.
- Specification of component behavior in terms of set of interfaces.
- Interfaces are bidirectional.
- Components are statically linked to each other via their interfaces.
- nesC is designed under the expectation that code will be generated by whole-program compilers.

IoT is a big concept. One branch of this concept is WSN. All developments can be ready in the computer but is very important to have one or more motes to test the programs. There are several commercial motes on the market. Some of the models are MICAz, TelosB, Imote2, etc. Each one has its own features and in different conditions, different motes can be optimum. So depending on the user's needs one type of mote has to be selected. Their main features can be found on the device's datasheet.

The chip industry is progressively offering more integrated System on Chip (SoC) devices optimized for WSN applications. These typically integrate low power MCUs like the ARM Cortex-M3 with standards based RF communications devices like the IEEE Std 802.15.4 radio. There are some early hints of an emerging trend toward offering devices pre-programmed with ROM-based protocol stacks to further simplify the software development task. Module suppliers take this even further, offering complete wireless modules that include the MCU, radio, protocol stack, and in many cases even the antenna, in a small integrated module that is already tested and certified to Federal Communications Commission (FCC) / European Telecommunications Standards Institute (ETSI) requirements.

In summary, wireless sensor/actuator networks offer a convenient and economical way to deploy smarter controls, but the system developer is faced with a multitude of trade-offs and options. The network's flexibility, performance and robustness to dynamic changes are all shaped by the network architecture and protocol. [3]

The Advanced Encryption Standard (AES) algorithm is a block cipher that uses an encryption key and several rounds of encryption. A block cipher is an encryption algorithm that works on a single block of data at a time. In the case of the AES standard, plain text can be ciphered in blocks of 128, 192, or 256 bits. The term "rounds" refers to the way in which the encryption algorithm mixes the data re-encrypting it ten to fourteen times depending on the length of the key (also 128, 192, or 256 bits). The AES algorithm itself is not a computer program or computer source code. It is a mathematical description of a process of obscuring data. A number of people have created source code implementations of AES encryption, including the original authors [6].

Finally, the project will develop to use all previous concepts. It is divided on phases, where each phase is important. On each phase, we will work on one of the previous concepts (TinyOS, motes, nesC, AES, etc.) to fully understand them and apply them to get the final results. Each concept will be explained with more details in the follows chapters.

1.2. Objectives

The main objective is the experimental performance evaluation of IoT security solutions over different protocol layers. Understand every step on the process is important to know how the technology works. There are important aspects as installation, support, application development, associated tools, standards, performance, etc. Along the project, the following tasks will be addressed (with the corresponding skills acquisition):

1. To familiarize with TinyOS and motes configuration.
2. To build programs and test how the motes work using nesC.
3. To understand the protocol stack defined over the motes.
4. To define and implement security solutions adapted to this technology.

1.3. Staff involved

1. Project director: María Dolores Cano Baños
2. Student: Jose Manuel Martínez Caro

1.4. Memory structure

The project is structured in 4 chapters, where each chapter shows different concepts about WSN, from a brief project introduction until software implementations. Every chapter has a short introduction.

Chapter 1: Introduction and objectives.
Chapter 2: Wireless Sensor Networks
Chapter 3: Implementation
Chapter 4: Results

Chapter 2 : Wireless Sensor Networks

2.1. TinyOS

2.1.1. Introduction

TinyOS is an Operative System (OS) specifically developed for Sensor Networks. The goal is to create the link between hardware capabilities and full systems. This system is available to manage efficiently the limited hardware capabilities and is programmed by using C language nesC. This language will be explained in section 2.1.2.

The OS is developed to be scaled with actual technological trends, to support intensive concurrency operations, and to ensure atomicity (it is possible create statement blocks that can be executed in whole or nothing). It uses a communication paradigm based on Active Message and it does not use dynamic memory [7]. Some of its features are: the memory areas are located in static memory, dynamic memory does not exist, there are not pointers, there is only one stack assigned where one task is executed in a time slot, and memory protection does not exist.

The executed model is based on events, where some aspects are important to consider [7]. First, it supports high concurrency levels in short amount of space. Only one execution context is shared between processing task unrelated (avoids the overhead of context switches). It provides concurrency, due to tasks or long execution processes that execute until be completed on background without interferences with others system events(it can be interrupted by low-level system events). It also provides mechanisms to create mutual exclusion in code sections (atomicity concept).

Regarding the programming model, it is based on components (modules) [7]:

- Each module is developed to operate continuously replying to entry events (alarms, timers, radio, etc.).
- When a new event arrives, brings the required execution context.
- The applications have to declare implicitly when it has finished using the CPU.

The following functions can be executed on TinyOS [7]: Analog-Digital converter, routing, node identification, memory reservation, serial-parallel converter, communication stack, radio, active messages, system logs, system clock, system energy, system reset, random number generator, system leds, data (int to Leds, int to Radio, and vice versa), and error handling (CRC).

2.1.2. nesC

nesC is a programming meta-language based on C, oriented to embedded systems that add network managing. nesC is oriented on components (where the developer creates its own component helped by other components defined previously) and is specially designed to program applications over WSN, particularly over TinyOS. nesC is optimized to WSN memory limitations. Two components can communicate with each other via an interface that will define some methods (commands and events), which must be implemented in each component. Thus, a method could seek the command execution of another component. On the other hand, to send a notification one event will be used [8].

This language also performs optimizations in program compilation, detecting possible data execution that can modify by concurrent method in a same state, on the application execution process. Also, it simplifies the application development, reduces code size, and removes potential error fonts. nesC is a very complete language used by TinyOS, which offer the following aspects [8]:

- The construction and composition are separated. There are two components types in nesC: modules and configurations.
  - The modules define the application code implementing one or more interfaces. These interfaces only can be acceded by the component. One module can provide different interfaces and can implement different functionalities (AMSender module provides Packet interface, AMPacket interface, AMSend interface, etc.). The modules are divided in three parts:
    - Provides: Interfaces that offer one component.
    - Uses: Interfaces that use the component.
    - Implementation: where the different desirable actions that the program has to do are done.
  - The configurations are used to wire the different components, connecting the interfaces that some components provide with interfaces that other components use.
- Interfaces are bidirectional. As it has been explained above, the interfaces are the components access, having commands and events that implement the functions. The interface provider implements the commands, while the user implements events. Also, interfaces can be defined as the "visible" module part.
- Static union of component, via interfaces. This increases the efficiency in execution time, increasing the robust design and allowing a better program analysis.
- nesC has tools that optimize code generation. An example is the execution data detector in compilation time.

In summary, the main aspects that nesC programming model provides and must be understood for understanding TinyOS design are [8]:

- nesC model is composed by interfaces and components.
- One interface can be used or can be provided; the components are modules or configurations.
- One application will be represented as a set of components grouped and related to each other as shown in Figure 2.
    - The interfaces are used to do operations that describe the bidirectional iteration; the interface provider should implement commands, while the interface user should implement events.
    - nesC uses arrows to determine relationships between interfaces. Think of the right arrow (->) as "binds to". The left side of the arrow binds an interface to an implementation on the right side. In other words, the component that uses an interface is on the left, and the component provides the interface is on the right.
    - There exists two component types:
      - Modules: Implement component specification.
      - Configurations: their mission is wire different components depending of the interfaces (commands or events).

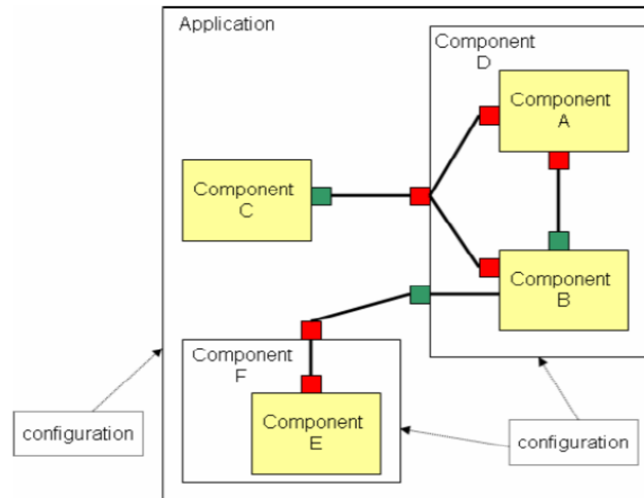

Figure 2: nesC program application example with different software components

2.1.3. TinyOS tutorial.

On TinyOS web site [7] there are wiki links with different tutorials that could be important to understand or improve the TinyOS knowledge. There are different wiki topics at different level as [9]:

- Main TinyOS concepts: components modules, configuration and interfaces. The user will learn how to compile and install a TinyOS program on a mote.
- TinyOS execution model and working with modules. Events, commands and its relation with interfaces in a more depth level, introducing the divided phase operation. It shows what the task is, TinyOS mechanism where the components share the processor in a cooperative mode.
- Main TinyOS communication model. Communication node to node. Different exercises to practice and to know how the network works sending and receiving messages.
- Mote-to-mote and mote-to-PC communication. Communication tools introduction to know how the communications between the different devices and platforms are possible (USB, serial port, etc.).
- Reading data from sensor. It uses different sample examples and show data on LEDs or send the data over radio. It explains how to take data from the sensor in every mote on the network.
- StdControl defines three commands, init(),start(), and stop(). init() is called when a component is first initialized, and start() when it is started, that is, actually executed for the first time. stop() is called when the component is stopped, for example, in order to power off the device that it is controlling. init() can be called multiple times, but will never be called after either start() or stop are called.
- Starting program sequence. It details how start a program on TinyOS and answer the question, where is the main() inside the code on a program written in nesC? Main.StdControl.init() command is the first command executed in TinyOS followed by Main.StdControl.start(). Therefore, TinyOS application must have Main component in its configuration. StdControl is a common interface used to initialize and start TinyOS components.

- Storage. Explain the method used by TinyOS to permanent storage (no volatile). Some application examples are Mount, ConfigStorage, LogRead, LogWrite work.
- Administration or resources arbitration method. Energy management model. How get access to shared resources on TinyOS and how create shared resources.
- Platforms. Know the differences between different motes platforms, including differences between files, directories and definitions to use. It includes an initiation guide to understand better the *make system* or develop a new platform on TinyOS.

2.2. 802.15.4

2.2.1. Introduction

The most important characteristics in this standard are network flexibility, low cost, and low energy consumption. This standard can be used with many applications that require a low rate in data transmission.

The motivation key to use the wireless technology is the reduction installation cost, because changing the cable is never is needed. The wireless networks involve a big information exchange with minimum installation effort. This trend is pushed by the big wireless component integration capability with an economic method and others wireless communication systems success such as mobile communication.

In 2000 two specialist standards groups (ZigBee and 15th group IEEE 802) started to work together to know how to build the new standard for low consumption wireless networks and low cost in industrial and domestic environment. So 802.15.4 was born as a new standard. The main objective was low transition on wireless networks personal area. The main characteristics are shown in Table 1 [10][11]:

Property	Range
Data transmission range	868 MHz: 20 Kbits/s | 915 MHz: 40 Kbits/s | 2.4 GHz: 250 Kbits/s
Scope	20 – 30m (indoor)
Latency	< 15ms
Channels	868/915 MHz: 11 channels 2.4 GHz: 16 channels
Frequency bands	868 MHz | 915 MHz | 2.4 GHz
Addressing	8 bits | 64 bits IEE
Access channel	CSMA-CA y CSMA-CA slotted
Temperature	-40 < ºC < +85

Table 1: IEEE 802.15.4 properties

2.2.2. Network layer

On classic wired networks, network layer is responsible of topology building and network maintenance. It also works for addressing and security. Network Layer in WSN has the same responsibility and has the saving energy challenge. Networks over IEEE 802.15.4 standard need to reduce total costs. IEEE 802.15.4 support multiples topologies as is shown in Figure 3: star topology, peer-to-peer topology, etc. Some peripheries and PC interfaces require low potency connections with star type and others, e.g., using security, need high coverage area and need to implement peer-to-peer network [11].

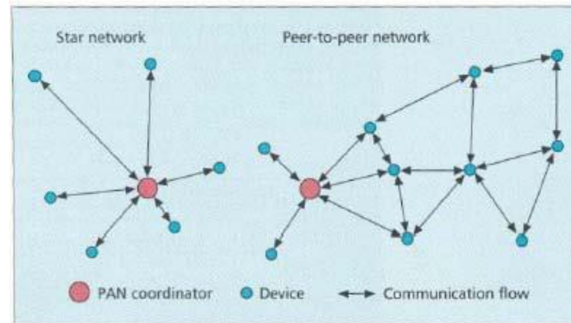

Figure 3: Network topology (Star and Peer-to-Peer)

2.2.3. Data link layer (DLL)

IEEE 802 project separate DLL in two layers. On one hand, Medium Access Control sublayer (MAC) and Logical Link Control (LLC). LLC is common to all 802 standards, while MAC sublayer depends of hardware and varies respect physical layer implementation. Figure 4 shows how the standard IEEE 802.15.4 is based on International Organization for Standardization (ISO), reference model to Open System Interconnection (OSI) [11].

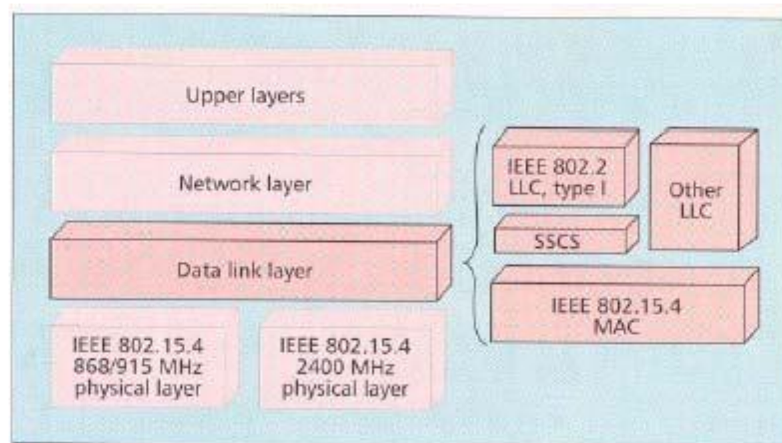

Figure 4: Layer distribution on 802.15.4

MAC IEEE 802.15.4 characteristics are: association and dissociation, recognition frame delivery, channel access mechanisms, frame validation, warranty management time slots and management guidelines. The sublayer MAC provides two services types to higher layers accessed through two Services Access Point (SAP).

MAC service administrator has 26 primitives. Compared with 802.15.1 (Bluetooth) that has 131 primitives and 32 events MAC 802.15.4 is very simple. It make it versatile to designed applications, although is a drawback has an element with minor features as 802.15.1 [11].

MAC frames general format

It is developed to be flexible and be adjusted to the different demanding applications with different networks topologies while still being a simple protocol. MAC frame format is included in Figure 5. MAC frame is denominated MAC Protocol Data Unit (MPDU) and is composed by MAC Header (MHR), MAC Service Data Unit (MSDU) and MAC footer (MFR)[11].

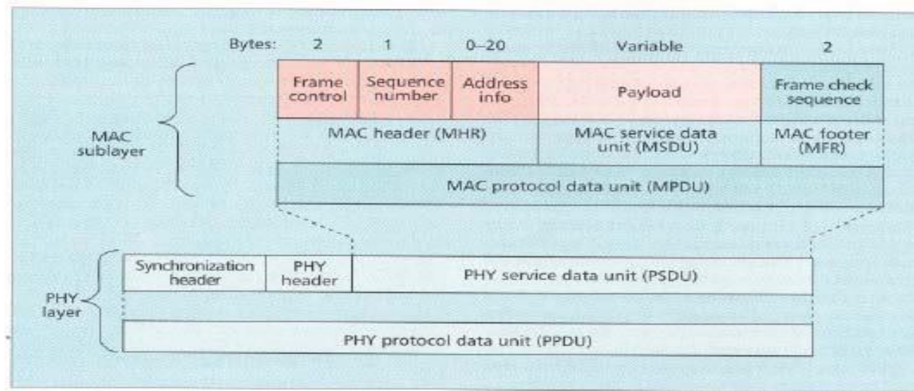

Figure 5: MAC frame format

The first MAC frame header field is control field. It indicates the transmitted MAC frame type, specifies the format, field address and controls Acknowledge (ACK) messages. As summary, control frame indicates how the rest of data frame is and what it contains.

Address info varies between 0-20 bytes. ACK frame has no information about any address. The bacon frame only has the source address information. It increases the protocol efficiency to packet storage.

Payload field is where the data are. It has a variable length, but complete frame cannot exceed 127 bytes. The payload length depends of the frame type. 802.15.4 has four different frame types: *beacon frames, data frames, ACK frames* and *MAC command frames*. Only data and beacon frames have information about higher layers. Others are used to MAC communications peer-to-peer. The transmission is successful when the ACK frame has the same sequence number than the last transmitted frame[11].

Super-frame structure.

Some applications need high dedicated bandwidth to achieve low potency consumption state. In a super-frame, a network coordinator, called Personal Area Network (PAN) coordinator, transmits super-frame beacons in a determinate interval. This interval ranges between 15ms and 245s. The time between each one is divided in 16 slots time, independent to super-frame length. One mote can transmit during a slot time, but has to finish the transmission before next super-frame.

One super-frame consists in a Contention Access Period (CAP) period follow by other Contention Free Period (CFP) and an inactive period where the coordinator could be in an energy saving mode and will not do anything.

One mote can transmit data on a CAP using Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) algorithm, but PAN coordinator can assign intervals or timeslot to only one device that requires a particular fixed bandwidth or low latent transmissions. These time slots assigned are called Guarantee Time Slot (GTS) and together form Contention Free Period (CFP). Its period is variable depending of the rest device demand. When GTS is used, all devices have to complete all contention transmissions before CFP starts. Figure 6 shows the super-frame structure and different slots types [11].

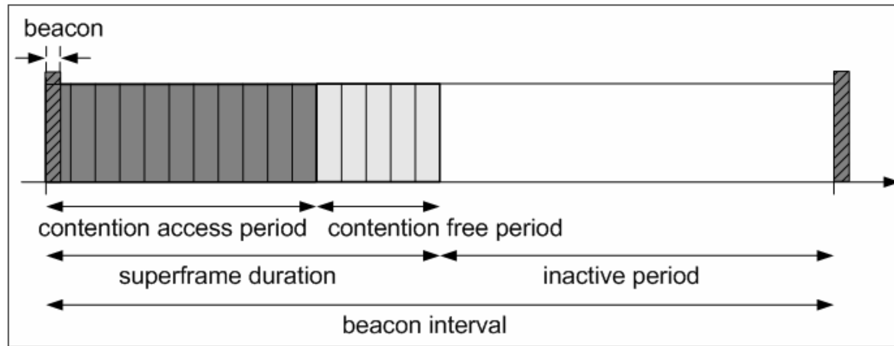

Figure 6: Super-frame structure

Slotted CSMA/CA (Figure 7) is used on beacon-enabled frame, so devices have to wait synchronized frames to transmit data. When a device want to transmit, wait until the next timeslot and ask if any device is transmitting in the same timeslot. If any device is transmitting, the device waits a determinate time or indicates connection failure after some tries. On beacon-enabled network does not use CSMA .

Un-slotted CSMA/CA (Figure 8) is used on non-beacon-enable frame, so devices do not have to wait synchronized frames to transmit data. In this networks type, devices are always active and waiting to receive packets, so there is a high potency requisite [11].

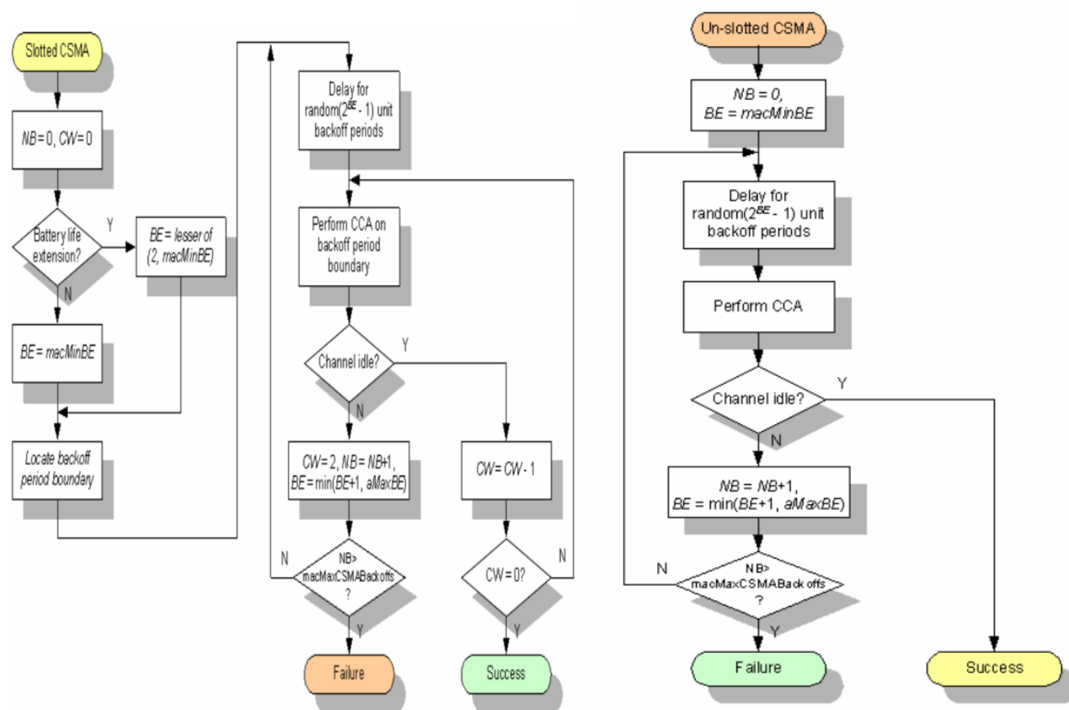

Figure 7: Slotted CSMA/CA *Figure 8: Un-slotted CSMA/CA*

Passive Radio Scanner

It works on devices to be associated to a coordinator. It is simple and consist on analyze every available channels and check if there is any coordinator sending beacons in the channel. From beacon sending, some quality parameters are extracted from a coordinator that offers a higher quality factor and the device is associated to this coordinator [11].

Others MAC characteristics

One important MAC function is confirming successful frames reception from any device. The success receptions and data validations are confirmed by ACK messages. If the receiver cannot receive the information by any reason, the receiver cannot send any acknowledgment. In the frame, the control field indicates if it is necessary any acknowledgment or not. Just after validating the entry frame, the receiver sends the acknowledgement message. The acknowledgement messages are never replied with other acknowledgement message [11].

2.2.4. Physical layer

IEEE 802.15.4 provides two Physical Layer (PHY) options, which combine with MAC to let a large network application range. Both PHYs are based on Direct Sequence Spread Spectrum (DSSS) method. It causes low digital implementation costs on Integrated Circuits (IC) and shares the same basic packet structure low-duty-cycle with low consuming operations.

The main different between both PHY is the frequency band. 2.4 GHz PHY specifies operation on industrial, medical and scientific band available at world level, while 868/915 MHz PHY specifies operations on 868 MHz in Europa and 915 MHz on Industry, Scientific and Medical (ISM) band in EEUU. 2.4 GHz offer more advantages in economics terms, high radio link range, less propagation loss and low manufacturing prices, but this frequency band is highly used because it is a free frequency band.

Another distinction related to PHY characteristics is the transmission range. 2.4 GHz PHY lets a 250 Kbits/s transmission range while 868/915 MHz has 20 Kbits/s and 40 Kbits/s transmission range, respectively. The high transmission range on 2.4 GHz is mainly because of the high modulation order, where every symbol represents multiples bits. The different transmission ranges can be used to get a high application variety. Due low data density, 868/915 MHz PHY can be used to get a high sensitive and have higher coverage areas, so reduce the node numbers to cover a specific area [11].

2.2.5. Transmission and reception channels

IEEE 802.15.4 defines 27 frequency channels between three bands. 868/915 MHz PHY supports only one channel between 868 and 868.6 MHz and 10 channels between 902.0 and 928.0 MHz. It is unlikely that only one network uses 11 channels. Both bands are very close in frequency and the same hardware can be used to reduce manufacturing costs.

2.4 GHz PHY defines 16 channels between 2.4 and 2.4835 GHz with a large gap between channels (5 MHz) to facilitate the filter needs in transmission and reception. The following figure (Figure 9) shows the channel structure and the central frequency calculation of each one [11].

Wireless networks can produce multiples interferences. It is produced because some applications are working at the same frequency band, so, to relocate a specific application on the spectrum will be an important factor to the wireless networks success. The standard was developed to implement a dynamic channel selection through a specific algorithm selection, where the network is the responsible. MAC layer includes search functions that follow step-by-step through an allowed channel list searching the beacon frame. PHY has some low level functions as detection-receiving energy levels, link quality indicator, channel commutation that allow to assign channels and get frequency selection agility. These functions are used by the network to establish the initial channel and change the channel, as response to a high pause [11].

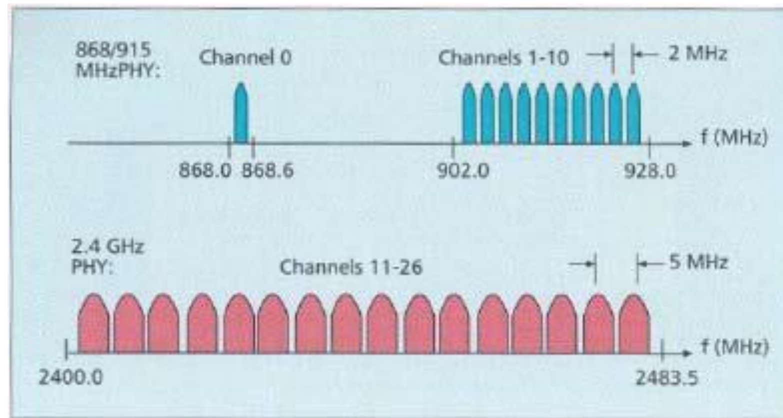

Figure 9: Channel structure on IEEE 802.15.14

2.2.6. Information packet structure

To keep a simple and common interface with MAC, both layers PHY share a simple packet structure (Figure 10). Each packet, or PHY Protocol Data Unit (PPDU), has synchronization header, PHY header to indicate packet length and payload or PHY Sequence Data Unit (PSDU). It does not require PHY channel equalization due to combination of small coverage areas and low transmission range.

On PHY header, 7 bits are used to specify the payload length (bytes). The packet length is between 1 and 127 bytes through MAC layer overhead [11].

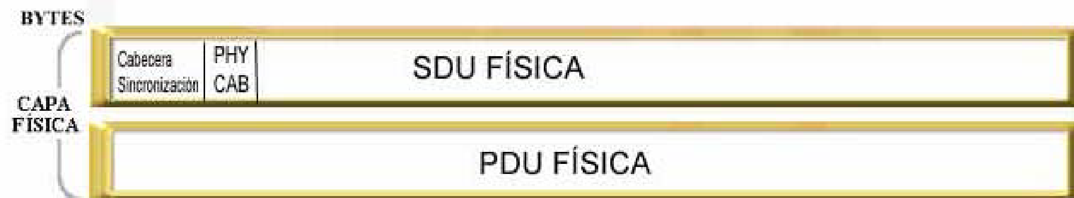

Figure 10: PHY Packet

2.2.7. Modulation

PHY 868/915 MHz uses simple DSSS where each transmitted bit is represented by a chip-15 with maximum sequence length (m sequence). Binaries data are codified to multiply each m sequence by +1 or -1, and the chip sequence is modulated inside the carrier using Binary Phase Shift Keying (BPSK). Before modulation, a differential data codification is used to let a low complexity coherent differential reception.

PHY 2.4 GHz uses a semi-orthogonal modulation technic based on DSSS methods (similar properties). Binaries data are grouped in 4 bytes symbols, and each symbol specifies one of the 16 semi-orthogonal transmission sequences of Pseudo-Noise code (PN). The added sequence to chip is modulated in carrier using Minimum Shift Keying (MSK). The quasi-orthogonal symbols simplifies the implementation to a performance change slightly lower (0.5dB). The modulation parameters to both PHY are in Table 2 [11]:

PHY (MHz)	Band	Data parameters			Risk parameters	
		Bit speed (Kbits/s)	Symbol speed (Kbaud)	Modulation	Chip speed (Mchips/s)	Modulation
868/915 (MHz)	**868.0-868.6 (MHz)**	20	20	BPSK	0.3	BPSK
	902.0-928.0 MHz	40	40	BPSK	0.6	BPSK
2.4 (GHz)	**2.4-2.4835 (GHz)**	250	62.5	16-semiorthogonal	2.0	O-QPSK

Table 2: Modulation parameters

In efficiency terms, orthogonal signaling improves 2 dB more than differential BPSK. In reception sensitivity terms, 868/915 MHz has 6-8 dB advantage because has transmission speed lowers [11].

2.2.8. Sensitivity and range

The current IEEE 802.15.4 sensitivity specifications specify -85 dBm to 2.4 GHz PHY and -92 dBm to 868/915 MHz PHY. These values include enough tolerance range required because manufacturing imperfections in the same way that let implement low cost applications.

Naturally, the wanted range will depend of sensitivity receptor and transmission power. Each device has to be able to transmit at least 1mW, but depending on the application requirements, the transmission power could be higher or lower. The normal devices cover a range between 20-30 meters, with good sensibility and a transmission power moderated increase. A star topology network can provide a total coverage in a house. In high latency applications, mesh topology offers an alternative solution with homemade coverage because each device only needs enough energy to communicate with the closest neighbor[11].

2.2.9. Interferences

The device that works in 2.4 GHz band can be interfered by other services that work on this band (e.g. Wi-Fi). This situation is ok in applications that use IEEE 802.15.4 which need low Quality of Service (QoS), does not need asynchrony communication, and is expected to make several attempts to complete the packet transmission. By contrast, a primary IEEE 802.15.4 application request is long battery duration. This objective is achieved by low power transmission and low service cycles. The IEEE 802.15.4 devices are sleeping 99.9% of the time and use low energy transmission on spread spectrum, so it should works on 2.4 GHz band velocity [11].

2.2.10. Average power consumption

The sensor’s total average power consumption, thus ultimately its battery life, will be determined by the contribution from both of its power specs and by the active/sleep duty cycle, as it is shown in Figure 11.[3]

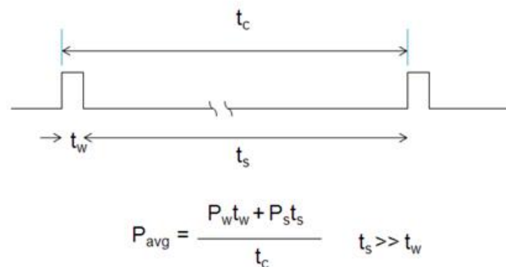

Figure 11: Average power over sleep/active duty cycle

2.3. Devices

2.3.1. Introduction

The processing and communicating units included into the motes define a series of platforms; and TinyOS supports different platforms. Table 3 shows the main features of the most common platforms.

Platform		**Imote2**	**MICAz**	**TelosB**	**EPIC**
Manufacturer		**Crossbow**	**Crossbow**	**Crossbow**	**Arch rock**
Processor		Intel PXA271 XScale	Atmel ATMega 128L	TI MSP430F1611	TI MSP430F1611
	Clock	13 to 416 MHz	8 MHz	8 MHz	8 MHz
	Bits	32	8	16	16
	FLASH	512 KB	128 KB	48 KB	48 KB
	RAM	256 KB	4 KB	10 KB	10 KB
	Max. Consumption	44 mA	19 mA	1.8 mA	1.8 mA
Radio unit		Chipcon CC2420	Chipcon CC2420	Chipcon CC2420	Chipcon CC2420
	Standard	CSMA/CA	CSMA/CA	CSMA/CA	CSMA/CA
	Band/Data rate	2.4 GHz /250 Kbps	2.4 GHz /250 Kbps	2.4 GHz /250 Kbps	2.4 GHz /250 Kbps
	Max. Consumption	23 mA	23 mA	23 mA	23 mA
External memory		32 MB	512 KB	1024 KB	16 MB
Onboard Sensors		-	-	Light (TSR, PAR), temperature and humidity.	Light, temperature and humidity.

Table 3: Platforms Comparative.

2.3.2. MICAz

MICAz (Figure 12) is a 2.4 GHz mote module used for enabling low-power, wireless sensor networks. The product features are included in Table 3. The device is compatible with Crossbow's sensor boards, gateways and MoteWorks™ software enabling the development of custom sensor application, where MoteWorks™ is a wireless sensor network platform for reliable, ad-hoc mesh networking.

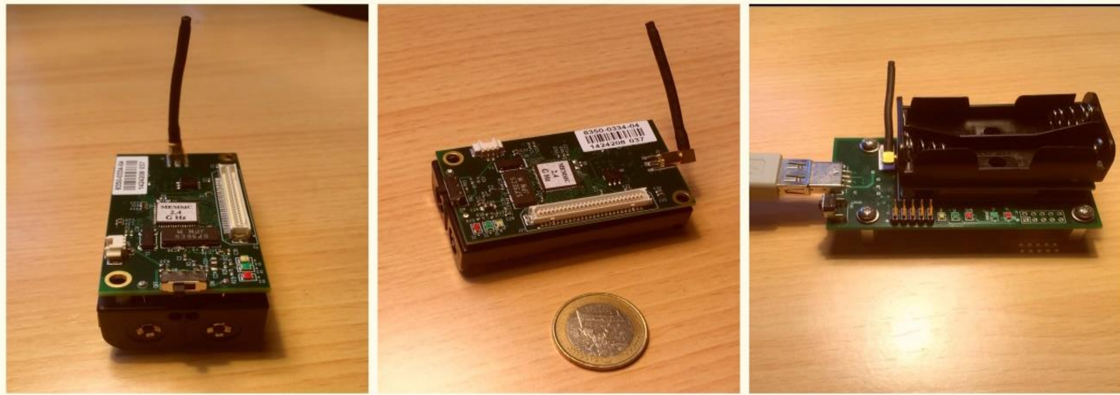

Figure 12: MICAz device

The device is specifically optimized for low power, battery-operated networks. The processor is MPR2400 and is based on Atmel ATmega128L and it is a low-power microcontroller which runs the program from its internal flash memory. A single processor board (MPR2400) can be configured to run the sensor application and the network/radio communications stack simultaneously. The 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces (Figure 13). These interfaces make it easy to connect to a wide variety of external peripherals. MICAz and TelosB are the most used devices.

MICAz uses the ISM band (2.4 – 2.4835 GHz) with 250 Kbps data rate. It has been developed with 4 Kbytes RAM, 4 Kbytes EEPROM, 512 Kbytes Serial Flash, 128 Kbytes Program Flash Memory and 12 bits to Analog-Digital Converter (ADC).

Crossbow offers a variety of sensor and data acquisition boards for the MICAz Mote. All of these boards connect to the MICAz via the standard 51-pin expansion connector. The device uses two AA batteries and can be powered by USB port. The supply voltage needed is 3V [12].

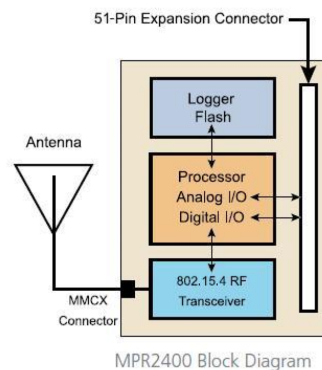

Figure 13: MICAz architecture

2.3.3. TelosB

The platforms TelosB and Tmote Sky are equivalent to MICAz. The hardware was manufactured by Tmote Sky and the platform specified to work in TinyOS is TelosB. TelosB (Figure 14) is developed by Crossbow and uses TinyOS without using an upper-layer application. The user can configure the devices and the networks parameters depending on the final application from the beginning. TelosB has high quality sensors that are made with energy efficient features and is one of the most used devices together with MICAz.

TelosB is a mote, so it has a small size, but includes an onboard integrated antenna and 16-pin I/O connector. There are two different versions of TelosB: TPR2400 and TPR2420. TPR2400 and TPR2420 are very close but the difference is TPR2400 includes an optional sensor suit (light, temperature, and humidity sensor). Both devices are used for lab studies.

About radio communication, TelosB uses the ISM band (2.4 – 2.4835 GHz) with 250 Kbps data rate. It has been developed with 10 Kbytes RAM, 16 Kbytes EEPROM, 1024 Kbytes Serial Flash, 48 Kbytes Program Flash Memory and 12 bits to ADC and Digital-Analog Converter (DAC) showed in Figure 15. Also, TelosB has an external oscillator (32 KHz) and is powered by two AA batteries or USB port. The supply voltage needed is 3V [13].

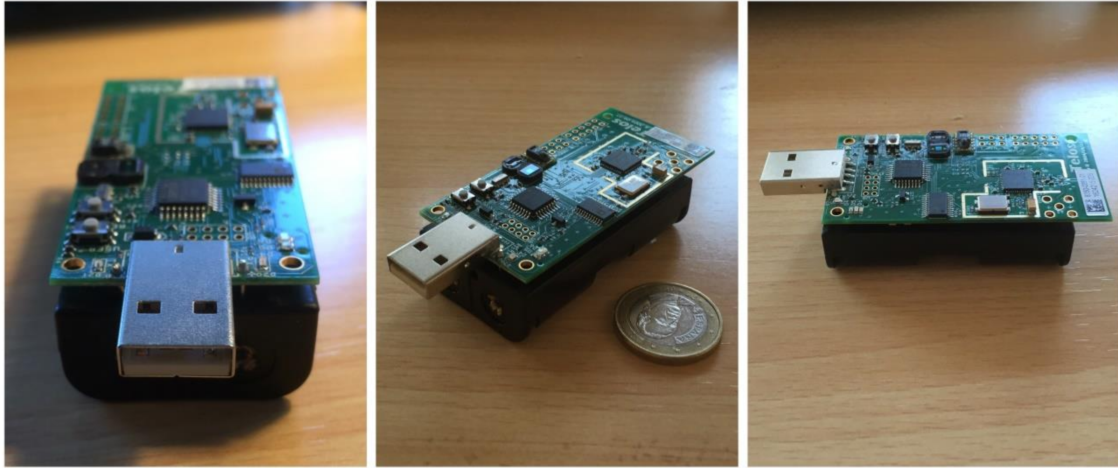

Figure 14: TelosB device

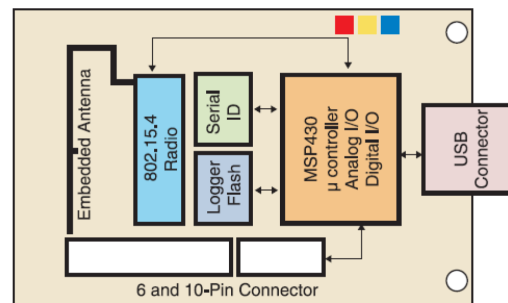

Figure 15: TelosB architecture

2.3.4. Imote2

As other devices, the main technical characteristics of Imote2 are shown in Table 3. Imote2 has not the same popularity as MICAz or TelosB, so is less used.

Imote2 (Figure 16) is developed by Crossbow and is an advanced sensor network node platform designed for demanding wireless sensor network applications requiring high CPU/DSP and wireless link performance and reliability. The platform is built around Intel's XScale® processor, PXA271. It integrates an 802.15.4 radio (TI CC2420) with an on-board antenna, 32MB Flash, 32MB SDRAM , XScale CPU core, XScale DSP, I/O module, RTC and power management (Figure 17). It exposes a "basic sensor board" interface, consisting of two connectors on one side of the board, and an "advanced sensor board" interface, consisting of two high-density connectors on the other side of the board. The Imote2 is a modular stackable

platform and can be stacked with sensor boards to customize the system to a specific application, along with a "battery board" to supply power to the system [14].

Figure 16: Imote2 device

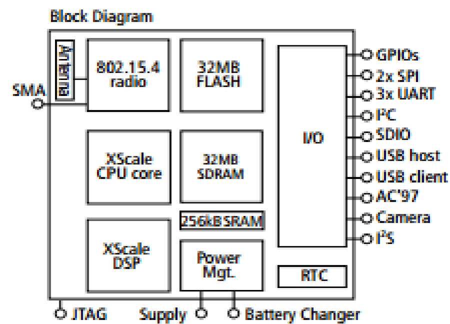

Figure 17: Imote2 architecture

2.3.5. EPIC [15]

Epic is a new open mote platform for application-driven design. Sensornet platforms, like most embedded systems, are tightly coupled to their applications. This coupling can make it difficult for general-purpose platforms to address application-specific needs, forcing platform designers to repeatedly re-implement functionality. Inspired by the hierarchical nature of software and integrated circuit design, EPIC proposes Sensornet platforms, composed hierarchically from a family of modular components. This approach makes platform development accessible to a much wider community; developers do not need to be analog, sensor, or radio frequency experts, and can instead reuse components that encapsulate the needed functionality. The EPIC design is open source, and all design files are available.

A key goal of the EPIC project is to develop a hardware architecture for Sensornet modules that specifically supports prototyping, measurement, and reuse. Prototyping enables design concepts to be explored, parameterized, and manipulated before they are finalized. EPIC (Figure 18) facilitates prototyping through componentized hardware with flexible interconnections between both the components themselves and third-party hardware. Once prototypes are constructed, their properties are measured under a range of input parameters to decide among alternate design choices. Although some properties can be measured directly in software, others like power draw require explicit platform support, which EPIC provides. Efficient reuse reduces the time and cost of going from prototype to production, but the requirements for reuse are at odds with the goals of prototyping and measurement: a final design rarely needs the scaffolding required during design and evaluation. Epic addresses this tension by partitioning hardware into specialized, minimalistic, and reusable core components with wide, connector-free interfaces and no decoding logic. In contrast, prototyping and measurement is supported through generic components that encapsulate core components and export flexible interconnections. Such partitioning of functionality, and adherence to a few simple design rules, can enable faster prototyping without sacrificing measurement-based evaluation or reuse, ultimately allowing systems to be deployed quicker and less expensively [15].

Figure 18: EPIC mote device

2.4. AES [16]

2.4.1. Introduction

The National Institute of Standards and Technology (NIST) published the Advanced Encryption Standard (AES) in 2001. AES is a symmetric block cipher that replaces DES as the approved standard for a wide range of applications. Compared to public-key ciphers such us RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms. In AES, all operations are performed on 8-bit bytes. In particular the arithmetic operations of additions, multiplications and division are performed over the finite field GF(2^8). Virtually all encryption algorithms, both conventional and public-key, involve arithmetic operations on integers.

The cipher takes a plaintext block size of 128 bits or 16 bytes (other allowed length for the plaintext are 192 and 256 bits). Depending of the key length, the algorithm is referred to AES-128, AES-192 or AES-256, where different numbers of iterations is necessary to each case (Figure 19). The input to the encryption and decryption algorithms is a single 128-bit block. This block is depicted as 4x4 square matrix of bytes. This block is copied into State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.

The key is depicted as a square matrix of bytes (in case it has a length of 128 bits – 16 bytes). This key is then expanded into an array of key schedule words. Figure 20 shows the expansion for the 128-bit key. Each word is four bytes and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. For example, the first four bytes of a 128-bits plaintext input to the encryption cipher occupy the first column of the in matrix; the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which from a word, occupy the first column of the w matrix.

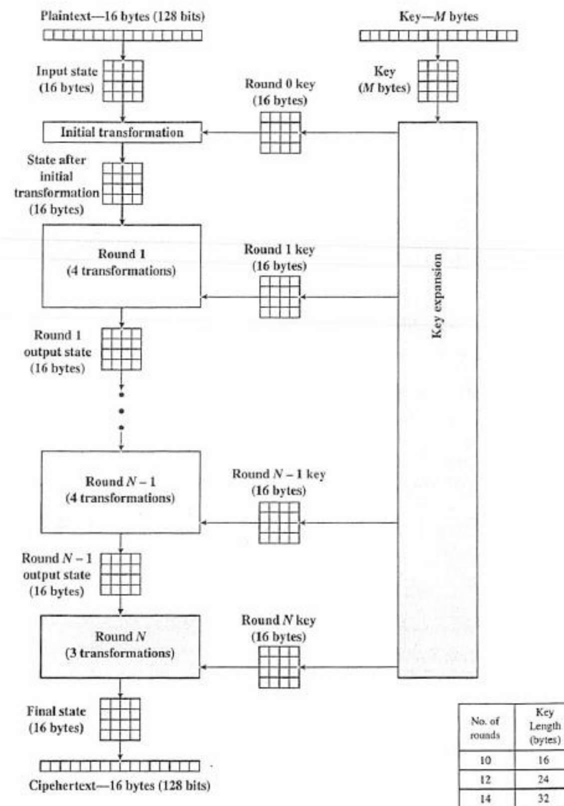

Figure 19: AES Encryption Process

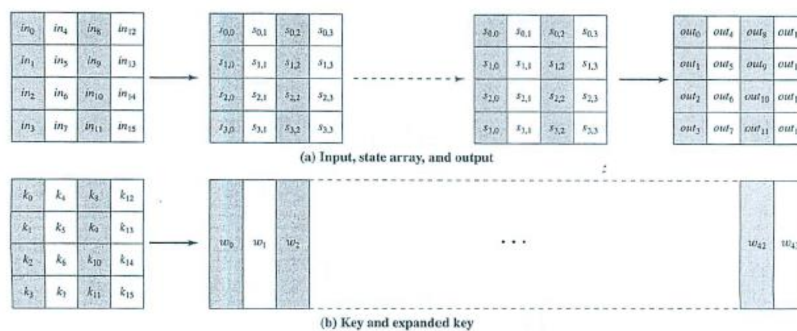

Figure 20: AES Data Structures

The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key 12 rounds for 24-bytes key and 14 rounds for 32-bytes key (Table 4). The first N–1 rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations and there is an initial single transformation (AddRoundKey) before the first round which can be considered Round 0. Each transformation takes one or more 4x4 matrices as input and produces a 4x4 matrix as output. Figure 20 shows that the output of each round is a 4x4 matrix, with the output of the final round being the ciphertext. In addition, the key expansion function generates N+1 round keys, each of which is a distinct 4x4 matrix. Each round key serves as one of the inputs to the AddRoundKey transformation in each round.

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Table 4: AES Parameters

2.4.2. Detailed Structure

Figure 21 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function. Encryption proceeding can be showed. Several comments about the overall AES structure are:

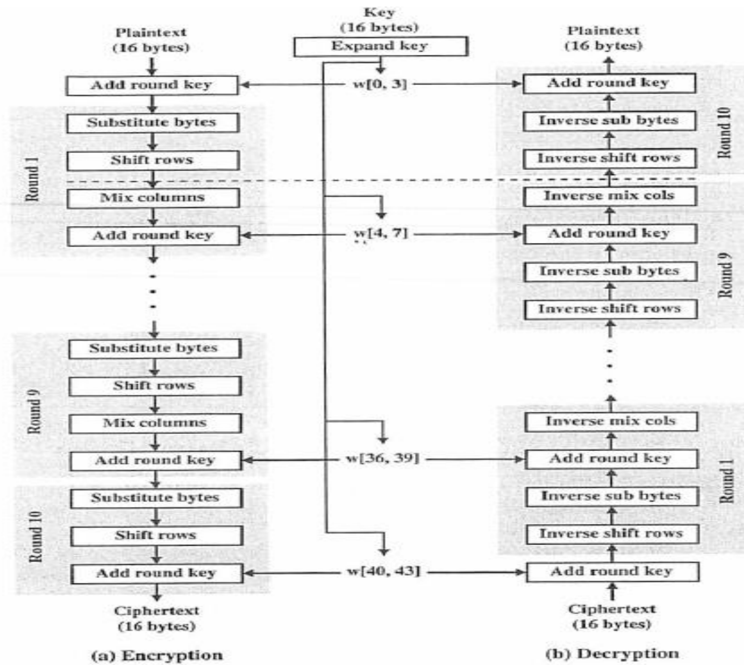

Figure 21: AES Encryption and Decryption

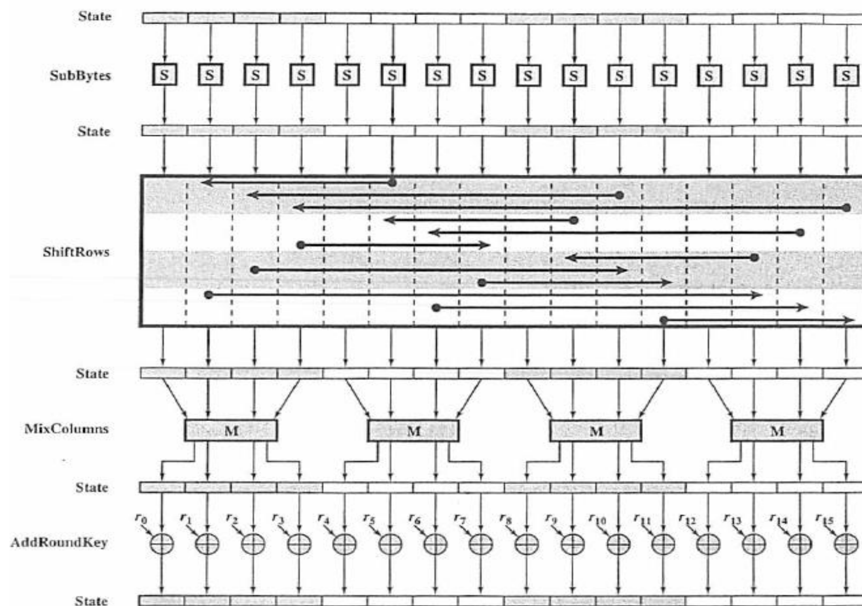

Figure 22: AES Encryption Round

- One noteworthy feature of this structure is that it is not a Feistel structure. Recall that, in the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.
- The key that is provided as input is expanded into an array of forty-four 32-bits words, w[i]. Four distinct words (128 bits) serve as a round key for each round; these are indicated in Figure 21.
- Four different stages are used, one of permutation and three of substitution:
    - *Substitute bytes*: Uses an S-box to perform a byte-by-byte substitution of the block
    - *ShiftRows*: A simple permutation.
    - *MixColumns*: A substitution that make use of arithmetic over GF(2^8).
    - *AddroundKey*: A simple bitwise XOR of the current block with a portion of the expanded key.
- The structure is quite simple. For both, encryption and decryption, the cipher begins with an AddRoundKey stage followed by nine rounds that include all four stages, followed by a tenth round of only three stages. Figure 22 depicts the structure of a full encryption round.
- Only the AddRoundKey stage makes use of the input key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
- The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the blocks (the other three stages), and followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
- Each stage is easily reversible. For the Substitute Byte, ShiftRows and MixColums stages, an inverse function is used in the decryption algorithm. For the AddroundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that A XOR B XOR B = A.
- As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.
- Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 21 lays out encryption and decryption going in opposite vertical directions. At each horizontal point, State is the same for both encryption and decryption.
- The final round of both encryption and decryption consist of only three stages. Again, this is consequence of the particular structure of AES and is required to make the cipher reversible.

2.4.3. AES Transformation Functions

It is time to discussion for each of the four transformations used in AES. For each stage, forward (encryption), inverse (decryption) and rationale for the stage are described.

2.4.3.1. Substitute Bytes Transformation

The forward substitute byte transformation, called SubBytes is a simple table (Figure 23). AES defines a 16x16 matrix of byte values, called an S-box (Table 5) that contains a permutation of all possible 256 8-bits values. Each individual byte of State is mapped into a new byte in the following way. The leftmost 4-bits of the byte are used as a row value and the rightmost 4-bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into value {2A}

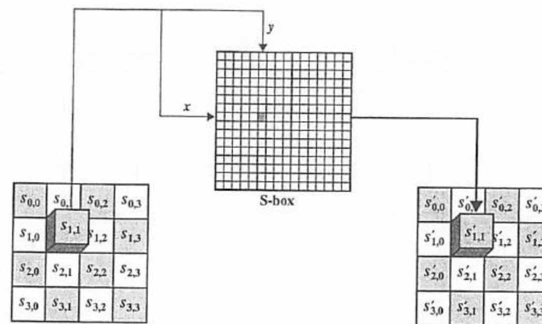

Figure 23: AES Byte-Level Operation

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 5: AES S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Table 6: AES Inverse S-box

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The inverse substitute byte transformation is called InvSubBytes and makes use of the inverse S-box (Table 6). The input {2A} produces the output {95} and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation equation followed by taking the multiplicative inverse in GF(2^8).

2.4.3.2. ShiftRows Transformation

The forward shift row transformation, called ShiftRows is depicted in Figure 24a. The first row of State is not altered. For second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed.

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right for the second row, and so on.

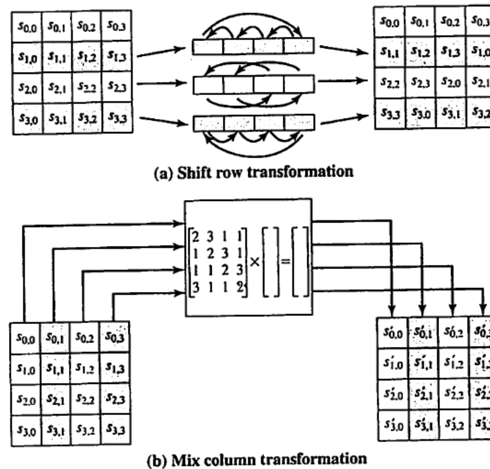

Figure 24: AES Row and Column Operations

2.4.3.3. MixColumns Transformation

The forward mix column transformation, called MixColumns (Figure 24b)operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on State.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Figure 25: AES MixColumns Matrix

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2^8). M matrix is multiplied with resulting matrix of the previous process to obtain the MixColumn result (Figure 25). The MixColumns transformation on a single column of state is represented in Figure 26.

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j} \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

Figure 26: MixColumns Operations

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	7.	9F
94	E4	3A	42
ED	A5	A6	BC

To verify the MixColumns transformation on the first column the necessary operations are (Figure 27):

$$\begin{aligned}
(\{02\}\cdot\{87\}) \oplus (\{03\}\cdot\{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\}\\
\{87\} \oplus (\{02\}\cdot\{6E\}) \oplus (\{03\}\cdot\{46\}) \oplus \{A6\} &= \{37\}\\
\{87\} \oplus \{6E\} \oplus (\{02\}\cdot\{46\}) \oplus (\{03\}\cdot\{A6\}) &= \{94\}\\
(\{03\}\cdot\{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\}\cdot\{A6\}) &= \{ED\}
\end{aligned}$$

Figure 27: MixColumns Operations II

The inverse MixColumn transformation, called InvMixColumns is defined by the following matrix multiplication. It use an inverse M matrix (Figure 28).

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Figure 28: AES MixColumns Inverse Matrix

MixColumns matrix and InvMixColumns are inverse and is demonstrate by the follow operations (Figure 29).

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

Figure 29: AES MixColumns Operation Demonstrations

In some cipher modes as CFB or OFB only encryption section is used. AES can be used to construct a message authentication code and for this, only encryption is used.

2.4.3.4. AddroundKey Transformation

In the forward, add round key transformation, called AddRoundKey the 128-bits of State are bitwise XORed with 128-bits of the round key. As shown in Figure 23b, the operation is viewed as a column wise operation between the 4 bytes of a State column and one word of the round key; it can also viewed as a byte-level operation.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

⊕

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	7B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

The first matrix is State and the second is the round key. The third matrix is the result of XOR operation. The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse. Figure 30 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

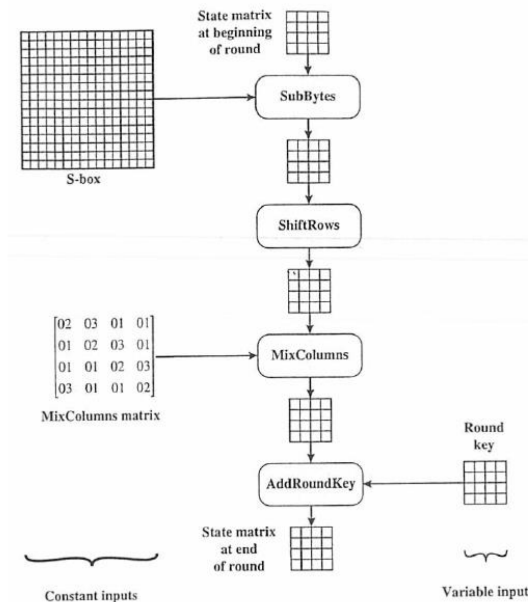

Figure 30: Inputs for Single AES Round

2.4.4. AES Key Expansion

The AES key expansion algorithm takes as input a four-word (16-bytes) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word w[i] depends on the immediately preceding word, w[i-1], and the word four positions back, w[i-4]. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used.

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round a is defined as Rcon[j] = (RC[j],0,0,0) with RC[1]=1, RC[j]=2*RC[j-1] and with multiplication defined over the field GF(2^8).The values of RC[j] in hexadecimal are shown in Table 7:

J	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RC[j]	01	02	04	08	10	20	40	80	1B	36

Table 7: RC[j] Table

Suppose that the round key for round 8 is: *EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F*. Then the first 4-bytes (first column) of the round key for round 9 are calculated by follows (Table 8):

i (decimal)	Temp	After RotWord	After SubWord	Rcon(9)	After XOR with Rcon	w[i-4]	w[i] = temp XORw[i-4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Table 8: AES Calculated Round

2.4.5. AES Modes

2.4.5.1. Introduction

A block cipher takes a fixed-length block of text of length b bits and a key as input and produces a b-bit block of ciphertext. If the amount of plaintext to be encrypted is greater than b bits, then the block cipher can still be used by breaking the plaintext up into b-bit blocks. When multiple blocks of plaintext are encrypted using the same key, a number of security issues arise. There are different modes available to cipher text. Depending of the programmer necessities one of the following modes described in Table 9 can be used.

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of plaintext bits is encoded independently using the same key	Secure transmission of single values (e.g. an encryption key)
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plain text to produce next unit of ciphertext.	General-purpose stream-oriented transmission. Authentication
Output Feedback (OFB)	Similar to CFB except the input to the encryption algorithm is the preceding encryption output and full blocks are used.	Stream-oriented transmission over noisy channel (e.g. satellite communications).

Table 9: AES Comparative Modes

2.4.5.2. Electronic Code Book (ECB)

Electronic Code Book (ECB) is the simplest mode where the plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 34). The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

For a message longer than b bits, the procedure is simply to break the message into b-bits blocks, padding the las block if necessary. Decryption is performed one block at a time, always using the same key. In Figure 31 the plaintext consist of a sequence of b-bits blocks, P_1, P_2, …, P_N; the corresponding sequence of ciphertext blocks is C_1, C_2, …, C_N. We can define ECB mode as follows.

ECB	$C_j = E(K,P_j)$ **j= 1, …, N**	$P_j = D(K,C_j)$ **j = 1, …, N**

Table 10: ECB Operations

The ECB method is ideal for a short amount of data, such as an encryption key. For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always start out with certain predefined fields, the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with. If the message has repetitive elements with a period of repetition a multiple of b bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

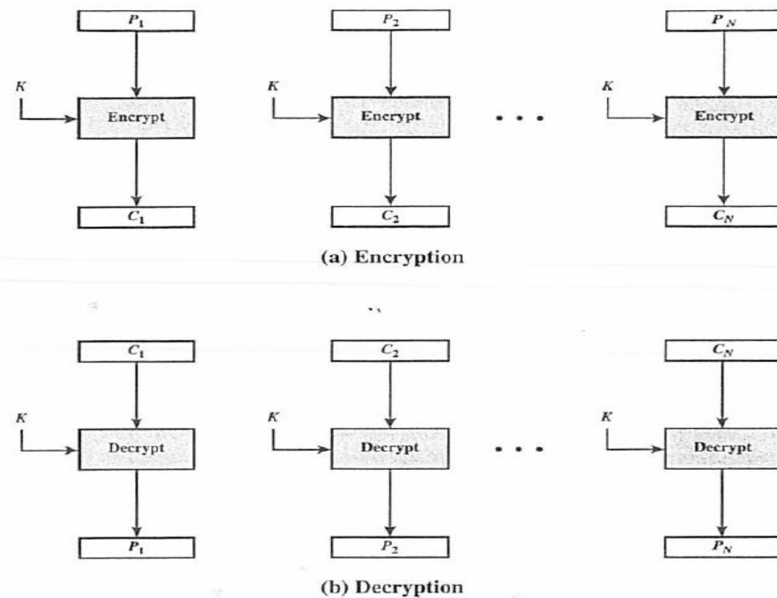

Figure 31: Electronic Codebook (ECB) Mode

2.4.5.3. Cipher Feedback Mode

For AES, DES or any block cipher, encryption is performed on block of b bits. In the case of AES b = 128 bits. Cipher Feedback (CFB) mode can operate in real time. If a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher. One desirable property of a stream cipher is that the ciphertext be of the same length as the plain text. Thus, 8-bit characters are being transmitted, each character should be encrypted to produce a ciphertext output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

Figure 32 depicts the CFB scheme. It is assumed that the unit of transmission is s bits ($s \ll b$) where usually s = 8 bits. In this case, rather than blocks of b bits, the plain text is divided into segments of s bits. The input to the encryption function is a b-bit shift register that is initially set to a specific value, Initialization Vector (IV). The leftmost s bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C_1, which is then transmitted. The contents of the shift register are shifted left by s bits and C_1 is placed in the rightmost s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same schema is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the encryption function that is used, not the decryption function. $MSB_s(X)$ being defined as the most significant s bits of X.

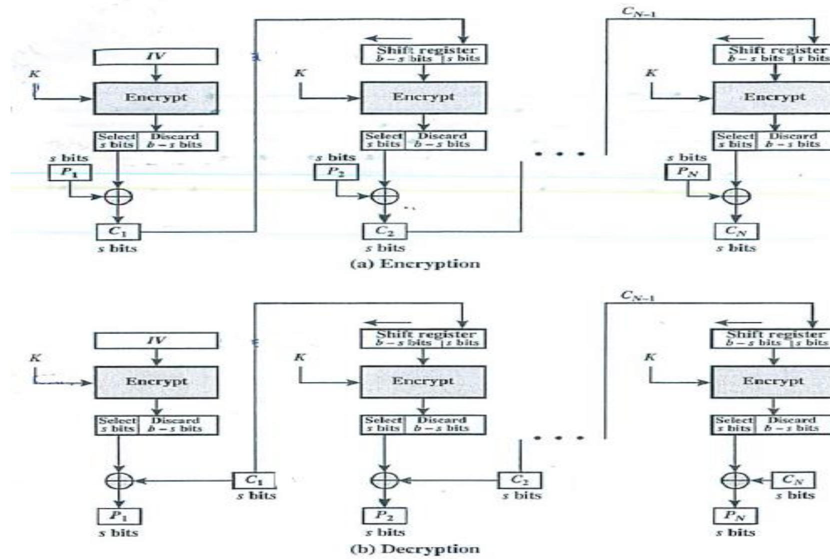

Figure 32: s-bit Cipher Feedback (CFB) Mode

	Encryption		Decryption	
CFB	$I_1 = IV$		I1 = IV	
	$I_j = LSB_{b-s}(I_{j-1}) \parallel C_{j-1}$	j = 2, …, N	Ij = LSBb-s(Ij-1) || Cj-1	j = 2, …, N
	$O_j = E(K,I_j)$	j = 1, …, N	$O_j = E(K,I_j)$	j = 1, …, N
	$C_j = P_j$ XOR $MSB_s(O_j)$	j = 1, …, N	$P_j = C_j$ XOR $MSB_s(O_j)$	j = 1, …, N

Table 11: CFB Operations

In CFB, the stream of bits that is XORed with the plaintext also depends on the plaintext. The input block to each forward cipher function depends on the result of the previous forward cipher function; therefore, multiple forward cipher operations cannot be performed in parallel. In CBF decryption, the required forward cipher operations can be performed in parallel if the input blocks are first constructed (in series) from the IV and ciphertext (Table 11).

2.4.5.4. Output Feedback Mode

The Output Feedback (OFB) mode is similar in structure to that of CFB. For OFB, the output of the encryption function is fed back to become the input for encrypting the next block of plaintext (Figure 33). Other difference is that OFB mode operates on full blocks of plaintext and ciphertext.

	Encryption		Decryption	
OFB	I_1 = Nonce		I1 = IV	
	$I_j = O_{j-1}$	j = 2, …, N	Ij = LSBb-s(Ij-1) || Cj-1	j = 2, …, N
	$O_j = E(K,I_j)$	j = 1, …, N	$O_j = E(K,I_j)$	j = 1, …, N
	$C_j = P_j$ XOR O_j	j = 1, …, N	$P_j = C_j$ XOR $MSB_s(O_j)$	j = 1, …, N
	$C_N^* = P_N^*$ XOR $MSB_u(O_N)$		$P_N^* = C_N^*$ XOR $MSB_u(O_N)$	

Table 12: OFB Operations

The size of a block is b. Of the last block of plaintext contains u bits, with u < b, the most significant u bits of the last output block ON are used for the XOR operation; the remaining b-u bits of the last output block are discarded. OFB mode requires an initialization

vector. In the case of OFB IV must be a nonce; that is, the IV must be unique to each execution of the encryption operation. The reason for this is that the sequence of encryption output blocks, O_i, depends only on the key and the IV and does not depend on the plaintext. Therefore for a given key and IV, the stream of output bits used to XOR with the stream of plaintext bits is fixed (Table 12).

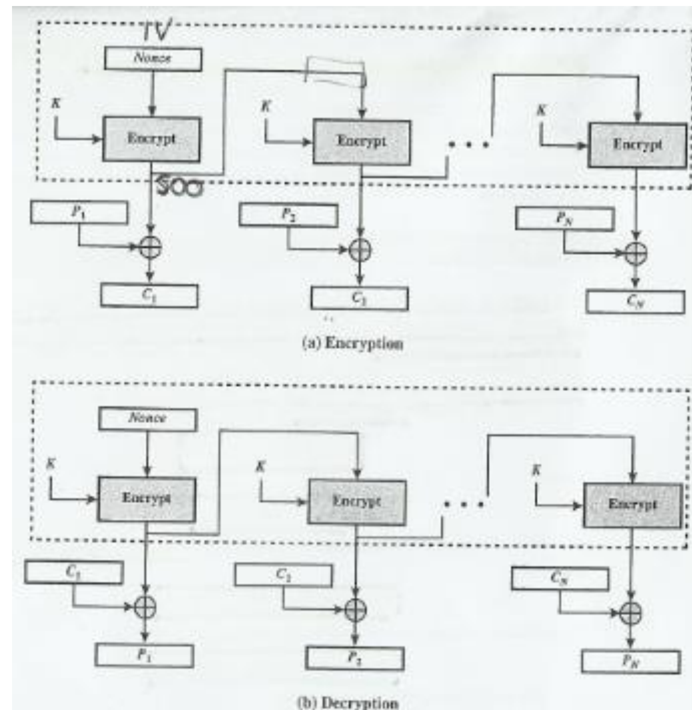

Figure 33: Output Feedback (OFB) Mode

One advantage of OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs, in C_1, only the recovered value of P_1 is affected; subsequent plaintext units are not corrupted. The disadvantage of OFB is that is more vulnerable to a message stream modification attack than other modes.

OFB has the structure of a typical stream cipher, because the cipher generates a stream of bits as a function of an initial value and a key, and that stream of bits is XORed with the plaintext bits. The generated stream that is XORed with the plaintext is itself independent of the plaintext.

2.5. Avrora

Avrora [17], a research project of the UCLA Compilers Group, is a set of simulation and analysis tools for programs written for the AVR microcontroller produced by Atmel and the Mica2 sensor nodes. Avrora contains a flexible framework for simulating and analyzing assembly programs, providing a clean Java API and infrastructure for experimentation, profiling, and analysis.

Simulation is an important step in the development cycle of embedded systems, allowing more detailed inspection of the dynamic execution of microcontroller programs and diagnosis of software problems before the software is deployed onto the target hardware. Avrora is a clean and open implementation motivated by this need.

Avrora also provides a framework for program analysis, allowing static checking of embedded software and an infrastructure for future program analysis research. Avrora is flexible, providing a Java API for developing analyses and removes the need to build a large support structure to investigate program analysis.

2.5.1. Avrora Utilities

- The provided simulator can test your programs before they are deployed onto the hardware device with cycle accurate execution times.
- The monitoring infrastructure allows users to add online monitoring of program behavior for better program understanding and optimization opportunities.
- The profiling utilities allow users to study their program's behavior in simulation.
- The instrumentation capabilities allow for detailed observation of program behavior without disturbing the simulation, and without modifying the simulator source code.
- The GDB debugger hooks allow source-level debugging and integrated development and testing.
- The control flow graph tool can create a graphical representation of your program's instructions that is useful for understanding how it is structured and what the compiler does with your code.
- The energy analysis tool can analyze energy consumption and help to determine the battery life of your device.
- The stack checker tool can be used to bound the maximum stack size used by your program [17].

Chapter 3 : Implementation

3.1. Introduction

The main goal is to define an application, implement it and after that, collect data and process it to show the results using different formats. It is important how the user interact with motes, knowing and taking care about the mote performance. In this chapter, any detail about mote implementation is defined.

To this project, applications have been developed using TinyOS 2.1.2. From the beginning, MICAz motes have been used to create different scenarios with different settings. Later, TelosB motes have been also tested. TelosB devices have the same functionality as MICAz with basic already-developed programs. TelosB has more features than MICAz and TelosB does not need a programming board (MIB520) as MICAz does, because it has a USB interface incorporated and can be directly connected to a computer. Nevertheless, TelosB devices were available for testing at the very end of this final degree project, therefore they have been tested but performance metrics have not been taken.

To start to work with TinyOS, nesC, and several mote boards (MICAz in most of cases) can be difficult if the user has not any previous contact with this platform. It is important to have access to good documentation of TinyOS, nesC, and mote boards datasheets. Also, it is important a comprehensive reading to understand all the concepts about the platform and motes.

All programs contain a Makefile that has a special format. This file with the make utility will help to build and manage the projects. When the make command is run, the program will look for the Makefile in the directory and will execute it. In the process, the compiler takes the source files, outputs object files, and linker takes the object files and creates an executable.

3.2. Installing TinyOS

Once the documentation has been read, it is time to practice all the concepts learned. One source is the official TinyOS web page [7]. This web page shows different options to install TinyOS on the computer. Some options are: Official Supported Method and Others User Contributed Methods. Official Supported Methods is recommended because researchers validate these versions and are more expanded. The user can find different installation methods for different OS. The most used OS is Linux, but users also use MacOS X or Windows.

Linux has two options to install TinyOS. The first one is an automatic installation for Debian based systems where the user has to follow a guide by Eric Decket [18]. This user guide is very complete and specifies every step that the user has to follow. It is worthy to emphasize the need to amend some parameters in the configuration files. The other option is to employ a manual installation using RPM packages. This option is also available on MacOS X and Cygwin (Windows) and it is necessary to follow five basic steps (although Cygwin platform could need some extra step). The steps are:

- Install *Java JDK*.
- Install native compilers.
- Install nesC compiler.
- Install TinyOS source tree

- Install Graphviz visualization tool

Between step one and step two, it is necessary an additional step for the Windows platform. This step is to install Cygwin that is a large collection of GNU and Open Source tools, which provides functionality similar to a Linux distribution on Windows. Cygwin is also a DLL (cygwin1.dll) that provides substantial POSIX API functionality [19].

Other option is using TinyOS in a Virtual Machine (VM). It could be the easiest option in platforms different to Linux (Windows, MacOS X, etc.), but can be applied also in Linux. One advantage is that the virtual machine can be moved and copied unlike a direct install. This method also allows make current state snapshots to return to other previous checkpoint in case of having some incidence. To use this option three steps have to be followed:

- Install software to run a VM (VirtualBox, VMWare, etc.).
- Install an OS that supports automatic TinyOS installation. (Ubuntu, Debian, etc.).
- Install TinyOS.

The last officially supported method is to install TinyOS from source. Basically it consist of installing TinyOS and nesC using a source repository. The most common repository is GitHub [20] and with the clone tool is possible to download from source and install the program using the command contained in README file.

After explaining the Officially Supported Method, there are several User Contributed Methods. There are two VM that can be imported and used to start to work from an initial point after installing all the requirements to work with TinyOS and motes. One option is Ubuntu 9.10 + TinyOS 2.x VirtualBox Image and the user only needs VirtualBox installed, UbunTOS.ovf (.ovf is the extension used by VirtualBox when Virtual Machine is exported) downloaded, and the file UbunTOS.ovf imported into VirtualBox. Once the VM is imported it is ready to be used. Other option is XubunTOS. In this case, the installation method is the same. The file has another extension (XubunTOS.ova) and VirtualBox also works with .ovf extension, so, the install method is the same.

There are others installation methods on MacOS and MacOS X but are not recommended because there are too many steps. One-step can be wrong and all the installation process could be wrong. Some methods use MacPorts, AVR toolchain, Gentoo, etc.

The selected installation method used in this project is VM importation on any program (e.g. VirtualBox, VMWare, etc.) and to work on the VM. The advantage of this method is that if one error happens, the VM could be reinstalled and started again with the process. On the other hand, the dedicated resources used on a standard Linux installation are higher than the resources used by using VM.

3.3. Basic Programs

3.3.1. Blink

This is a counter using mote Leds (1Hz) and is the TinyOS "Hello World", so, the easiest program in TinyOS. Blink application is composed of two components: a module file called BlinkC.nc and a configuration file called BlinkAppC.nc.

BlinkC.nc provides the implementation of the Blink application. BlinkAppC.nc is used to wire the BlinkC.nc interfaces to other components interfaces required. Once the application has been compiled, an executable file is generated and inserted in the mote. Blink applications has Makefile included in the folder and allows selecting the desired platform and different options on application's top-level configuration.

```
//BlinkAppC.nc
  configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;

  BlinkC -> MainC.Boot;
  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```

Figure 34: BlinkAppC.nc file

BlinkAppC.nc (Figure 34) is the configuration file where several components are declared to be used in the application. The components have interfaces that are wired with application interfaces. This file is declared in Makefile in the variable COMPONENT.

```
//BlinkC.nc
#include "Timer.h"
module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }
  event void Timer0.fired()
  {
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
  }
```

```
  event void Timer1.fired()
  {
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
  }
}
```

Figure 35: BlinkC.nc file

At the top of the Figure 35, interfaces are declared in the module section. The application uses three Timers (Timer0, Timer1 and Timer2), where each one has its own Timer period preloaded. Leds and Boot interface are also declared. In the implementation section, events and variables used on the program are declared. *event Boot.booted()* is the "main" method, being the first method executed when the application starts. It initializes the three Timer periods (250ms, 500ms and 1000ms). Once a Timer is fired, it executes the appropriate method running its corresponding defined code. When any Timer is fired, LedX toggle the status on to off or off to on.

```
//Makefile
COMPONENT=BlinkAppC
TINYOS_ROOT_DIR?=../../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 36: Blink Makefile

In addition, the folder includes Makefile (Figure 36) where the programmer indicates to the compiler where the component is, the root TinyOS directory, and one include sentence.

3.3.2. BlinkToRadio

BlinkToRadio is a very simple application, also included in TinyOS applications. It is a modification of Blink, which adds the feature of sending the counter data to other mote that is listening on the Radio interface. This application is composed by three files.

```
// $Id: BlinkToRadio.h,v 1.4 2006-12-12 18:22:52 vlahan Exp $
#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H
enum {
  AM_BLINKTORADIO = 6,
  TIMER_PERIOD_MILLI = 250
};
typedef nx_struct BlinkToRadioMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} BlinkToRadioMsg;
#endif
```

Figure 37:BlinkToRadio.h file

AM_BLINKTORADIO and TIMER_PERIOD_MILLI are declared in BlinkToRadio.h (Figure 37). The first one, defines the used Radio channel as number six and second one, which

every time the Timer is over is reinitialized with 250ms. In addition, there is a struct type called BlinkToRadioMsg that has two unsigned 16-bits integer variables defined.

```
//BlinkToRadioAppC.nc
#include <Timer.h>
#include "BlinkToRadio.h"
configuration BlinkToRadioAppC {
}
implementation {
  components MainC;
  components LedsC;
  components BlinkToRadioC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_BLINKTORADIO);
  components new AMReceiverC(AM_BLINKTORADIO);
  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 38: BlinkToRadioAppC.nc file

In Figure 38, there are two files included at the top of the code to take the functionality and variable declarations of these files. After that, several components are declared, and they will give to the program the functionality that the program needs. Finally, the application interfaces are wired with the components interfaces to provide the necessary functionality.

```
// $Id: BlinkToRadioC.nc,v 1.6 2010-06-29 22:07:40 scipio Exp $
#include <Timer.h>
#include "BlinkToRadio.h"
module BlinkToRadioC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
} implementation {
  uint16_t counter;
  message_t pkt;
  bool busy = FALSE;
  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
```

```
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }
  event void Boot.booted() {
    call AMControl.start();
  }
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }
  event void AMControl.stopDone(error_t err) {
  }
  event void Timer0.fired() {
    counter++;
    if (!busy) {
      BlinkToRadioMsg* btrpkt =
        (BlinkToRadioMsg*)(call Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
      if (btrpkt == NULL) {
        return;
      }
      btrpkt->nodeid = TOS_NODE_ID;
      btrpkt->counter = counter;
      if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
        busy = TRUE;
      }
    }
  }
  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(BlinkToRadioMsg)) {
      BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
      setLeds(btrpkt->counter);
    } return msg;
  }
```

Figure 39: BlinkToRadioC.nc file

As BlinkToRadioAppC.nc, there are two includes declared in Figure 39. Several interfaces are declared in the module section. When the module section is finished, the implementation section begins, where different variables, methods, and events are defined.

BlinkToRadio is a simple application that increments a counter, displays the counter variable using Leds showing the three leftmost bits, and sends a message via Radio interface with the counter value when a Timer fires. At least two motes are needed in this example.

```
//Makefile
COMPONENT=BlinkToRadioAppC
TINYOS_ROOT_DIR?=../../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 40: BlinkToRadio Makefile

In addition, the folder includes the Makefile (Figure 40) where the programmer indicates to the compiler where the component is, the root TinyOS directory, and one include sentence.

3.3.3. Sense

Sense is another basic program to know how the motes works with a Sensorboard inserted that can measure humidity, temperature, etc. This program is a simple sensing demo application. It periodically samples the default sensor on the Sensorboard and displays on the Leds the lowest three bits of the reading.

```
//SenseAppC.nc
configuration SenseAppC
{
}
implementation {

  components SenseC, MainC, LedsC, new TimerMilliC(), new DemoSensorC() as Sensor;

  SenseC.Boot -> MainC;
  SenseC.Leds -> LedsC;
  SenseC.Timer -> TimerMilliC;
  SenseC.Read -> Sensor;
}
```

Figure 41: SenseAppC.nc file

Figure 41 is the configuration file, where five components are declared. In addition, the application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
//SenseC.nc
#include "Timer.h"
module SenseC
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli>;
    interface Read<uint16_t>;
  }
```

```
}
implementation
{
  #define SAMPLING_FREQUENCY 100

  event void Boot.booted() {
    call Timer.startPeriodic(SAMPLING_FREQUENCY);
  }
  event void Timer.fired()
  {
    call Read.read();
  }
  event void Read.readDone(error_t result, uint16_t data)
  {
    if (result == SUCCESS){
      if (data & 0x0004)
        call Leds.led2On();
      else
        call Leds.led2Off();
      if (data & 0x0002)
        call Leds.led1On();
      else
        call Leds.led1Off();
      if (data & 0x0001)
        call Leds.led0On();
      else
        call Leds.led0Off();
    }
  }
```

Figure 42: SenseC.nc file

Figure 42 defines four application interfaces in the module file. In the implementation section, a Timer period is defined every 100ms. The timer is initialized the first time in *event void Boot.booted()*. Once the timer is over, the application reads data from SensorBoard and shows on Leds the three rightmost bits of the acquired data.

```
//Makefile
COMPONENT=SenseAppC
TINYOS_ROOT_DIR?=../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 43: Sense Makefile

In addition, the folder includes a Makefile (Figure 43), where the programmer indicates to the compiler where the component is, the root TinyOS directory, and one include sentence.

3.3.4. Oscilloscope

Oscilloscope is a simple data-collection demo. It periodically samples the default sensor and broadcasts a message over the radio every 10 readings. These readings can be received by a BaseStation mote and displayed by the Java "Oscilloscope" application found in the java subdirectory. The sampling rate starts at 4Hz, but can be changed from the Java application. Oscilloscope can be compiled with a sensor board's default sensor using the following

command: *SENSORBOARD=<sensorboard name> make <mote>*. The sensor can be changed editing OscilloscopeAppC.nc file.

The files that make Oscilloscope to properly operate are oscilloscope.py, Oscilloscope.h, OscilloscopeAppC.nc and OscilloscopeC.nc.

```
#!/usr/bin/env python
import sys
from tinyos import tos
AM_OSCILLOSCOPE = 0x93
class OscilloscopeMsg(tos.Packet):
    def __init__(self, packet = None):
        tos.Packet.__init__(self,
                            [('version',  'int', 2),
                             ('interval', 'int', 2),
                             ('id',       'int', 2),
                             ('count',    'int', 2),
                             ('readings', 'blob', None)],
                            packet)
if '-h' in sys.argv:
    print "Usage:", sys.argv[0], "serial@/dev/ttyUSB0:57600"
    sys.exit()
am = tos.AM()
while True:
    p = am.read()
    if p and p.type == AM_OSCILLOSCOPE:
        msg = OscilloscopeMsg(p.data)
        print msg.id, msg.count, [i<<8 | j for (i,j) in zip(msg.readings[::2],
msg.readings[1::2])]
        #print msg
```

Figure 44: oscilloscope.py file

Figure 44 is a file programmed in Phyton. This script will pretty print the values received by a mote running BaseStation applications. The code defines different values and one class. It uses the print command to show information.

```
#ifndef OSCILLOSCOPE_H
#define OSCILLOSCOPE_H
enum {
  NREADINGS = 10,
  DEFAULT_INTERVAL = 256,
  AM_OSCILLOSCOPE = 0x93
};
typedef nx_struct oscilloscope {
  nx_uint16_t version;
  nx_uint16_t interval;
  nx_uint16_t id;
  nx_uint16_t count;
  nx_uint16_t readings[NREADINGS];
} oscilloscope_t;
#endif
```

Figure 45: Oscilloscope.h file

Figure 45 is a header file, which is included in all the files, where the necessary application values are declared as number of readings, default interval to send data, and Active Message application identifier. In addition, oscilloscope_t struct is declared and it contains unsigned integer values of 16 bits. The oscilloscope_t struct is encapsulated in the payload message to be sent to other mote via Radio interface, normally the BaseStation program.

```
//OscilloscopeAppC.nc
configuration OscilloscopeAppC { }
implementation
{
  components OscilloscopeC, MainC, ActiveMessageC, LedsC,
    new TimerMilliC(), new DemoSensorC() as Sensor,
    new AMSenderC(AM_OSCILLOSCOPE), new AMReceiverC(AM_OSCILLOSCOPE);

  OscilloscopeC.Boot -> MainC;
  OscilloscopeC.RadioControl -> ActiveMessageC;
  OscilloscopeC.AMSend -> AMSenderC;
  OscilloscopeC.Receive -> AMReceiverC;
  OscilloscopeC.Timer -> TimerMilliC;
  OscilloscopeC.Read -> Sensor;
  OscilloscopeC.Leds -> LedsC;

}
```

Figure 46: OscilloscopeAppC.nc file

Figure 46 is the configuration file, where all the components are declared and where application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
  event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    if (call RadioControl.start() != SUCCESS)
      report_problem();
  }
  void startTimer() {
    call Timer.startPeriodic(local.interval);
    reading = 0;
  }
  event void RadioControl.startDone(error_t error) {
    startTimer();
  }
  event void RadioControl.stopDone(error_t error) {
  }
```

Figure 47: OscilloscopeC.nc file I

Oscilloscope is a complex application included by default in TinyOS. OscilloscopeC.nc is the module file, where two includes are defined and several application interfaces declared. In the implementation section, variables, methods, and events are declared (Figure 47). The application starts executing *event void Boot.booted()*. The timer is started when *event void RadioControl.startDone(error_t error)* is executed, because in this event, *startTimer()* method is called.

```
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    oscilloscope_t *omsg = payload;
    report_received();
    if (omsg->version > local.version)
      {
```

```
        local.version = omsg->version;
        local.interval = omsg->interval;
        startTimer();
      }
    if (omsg->count > local.count)
      {
        local.count = omsg->count;
        suppressCountChange = TRUE;
      }
    return msg;
  }
  event void Timer.fired() {
    if (reading == NREADINGS)
      {
        if (!sendBusy && sizeof local <= call AMSend.maxPayloadLength())
          {
            memcpy(call AMSend.getPayload(&sendBuf, sizeof(local)), &local, sizeof local);
            if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf, sizeof local) == SUCCESS)
              sendBusy = TRUE;
          }
        if (!sendBusy)
          report_problem();
        reading = 0;
        if (!suppressCountChange)
          local.count++;
        suppressCountChange = FALSE;
      }
    if (call Read.read() != SUCCESS)
      report_problem();
  }
```

Figure 48: OscilloscopeC.nc file II

event message_t Receive.receive(message_t* msg, void* payload, uint8_t len)* is executed when a mote receives a message from other mote in the network and is included in Figure 48. On this message, the source includes program version, wanted time interval to sample, and counter to synchronize the application. Then the motes read data from the sensor and introduce the reading into the message payload. The mote will send the message to other mote via Radio interface once the Timer is over.

```
  event void AMSend.sendDone(message_t* msg, error_t error) {
    if (error == SUCCESS)
      report_sent();
    else
      report_problem();
    sendBusy = FALSE;
  }
  event void Read.readDone(error_t result, uint16_t data) {
    if (result != SUCCESS)
      {
        data = 0xffff;
        report_problem();
```

```
        }
      if (reading < NREADINGS)
        local.readings[reading++] = data;
    }
}
```

Figure 49: OscilloscopeC.nc file III

When data has been sent the *event void AMSend.sendDone (message_t* msg, error_t error)* releases the radio channel. Finally, *event void Read.readDone(error_t result, uint16_t data)* will fill the buffer with a new reading from sensor (Figure 49).

```
COMPONENT=OscilloscopeAppC
TINYOS_ROOT_DIR?=../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 50: Oscilloscope Makefile

In Makefile (Figure 50), the programmer indicates to the compiler where the component is, the root TinyOS directory, and one include sentence.

On Oscilloscope, in the java subfolder, there is one Java application (Figure 51) to show the acquired data from the sensor on screen. This application runs in a PC. Data sent by Oscilloscope is received by Basestation Radio interface. Basestation forwards data through the serial port and shows it on screen. The application is contained on */apps/Oscilloscope/java/* and the command to start the program is *./run*

The application provides a user graphic interface that allows monitoring data sensor as temperature, pressure, humidity, etc. In addition, it is possible to modify graphic ranges.

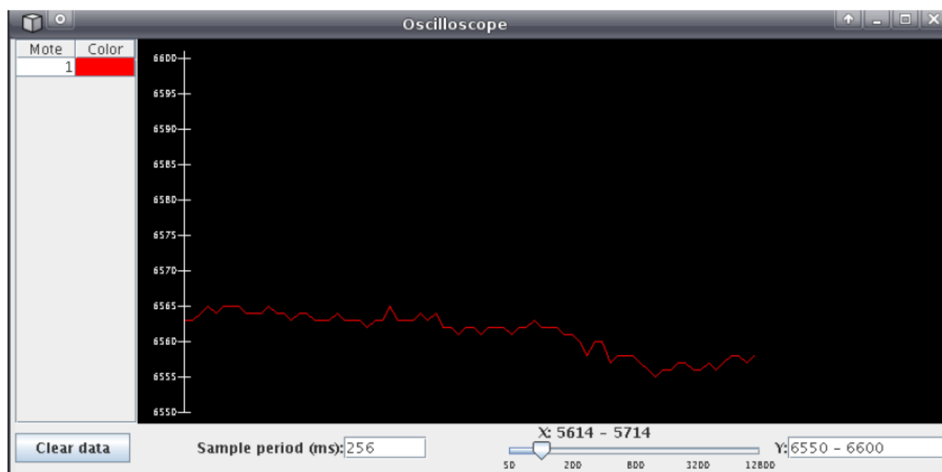

Figure 51: Java Oscilloscope application

3.3.5. BaseStation

BaseStation will be the responsible program of data reception from other mote within the network and subsequent delivery via serial port to the PC. There are transmission buffers where the motes keep the messages avoiding lost packets. Transmission buffers via UART or Radio keeps message_t type messages and default storage size is twelve messages by each buffer. Sent messages in the network have a message struct that contains data to be sent as payload. Message via Radio also includes header and footer of message. To send messages TinyOS follows Active Message model.

```
//BaseStationC.nc
configuration BaseStationC {
}
implementation {
  components MainC, BaseStationP, LedsC;
  components ActiveMessageC as Radio, SerialActiveMessageC as Serial;

  MainC.Boot <- BaseStationP;
  BaseStationP.RadioControl -> Radio;
  BaseStationP.SerialControl -> Serial;

  BaseStationP.UartSend -> Serial;
  BaseStationP.UartReceive -> Serial.Receive;
  BaseStationP.UartPacket -> Serial;
  BaseStationP.UartAMPacket -> Serial;

  BaseStationP.RadioSend -> Radio;
  BaseStationP.RadioReceive -> Radio.Receive;
  BaseStationP.RadioSnoop -> Radio.Snoop;
  BaseStationP.RadioPacket -> Radio;
  BaseStationP.RadioAMPacket -> Radio;

  BaseStationP.Leds -> LedsC;
}
```

Figure 52: BaseStationC.nc file

The configuration is defined in BaseStationC.nc (Figure 52). There are five components defined. The application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
//BaseStationP.nc
#include "AM.h"
#include "Serial.h"
module BaseStationP @safe() {
  uses {
    interface Boot;
    interface SplitControl as SerialControl;
    interface SplitControl as RadioControl;
    interface AMSend as UartSend[am_id_t id];
    interface Receive as UartReceive[am_id_t id];
    interface Packet as UartPacket;
    interface AMPacket as UartAMPacket;
    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Receive as RadioSnoop[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;
    interface Leds;
  }
}
  enum {
    UART_QUEUE_LEN = 12,
```

```
    RADIO_QUEUE_LEN = 12,
  };
  message_t  uartQueueBufs[UART_QUEUE_LEN];
  message_t  * ONE_NOK uartQueue[UART_QUEUE_LEN];


...
```

Figure 53: BaseStationP.nc file I

Figure 53 is the top file of BaseStationP.nc where variables are defined and initialized. In addition, several functions are declared to create subroutines with the objective to create a code structure orderly. These variables and functions will be used to give some functionality to the code.

```
  event void Boot.booted() {
    uint8_t i;
    for (i = 0; i < UART_QUEUE_LEN; i++)
      uartQueue[i] = &uartQueueBufs[i];
    uartIn = uartOut = 0;
    uartBusy = FALSE;
    uartFull = TRUE;
    for (i = 0; i < RADIO_QUEUE_LEN; i++)
      radioQueue[i] = &radioQueueBufs[i];
    radioIn = radioOut = 0;
    radioBusy = FALSE;
    radioFull = TRUE;
    if (call RadioControl.start() == EALREADY)
      radioFull = FALSE;
    if (call SerialControl.start() == EALREADY)
      uartFull = FALSE;
  }
```

Figure 54: BaseStationP.nc file II

The handler has to be initialized and also initialize the components to be managed (*event void Boot.booted()*) (Figure 54). Likewise, it should be marked that have been started properly. Buffers storage for sending messages will be started and variables will be reset.

When any reception event happens *message_t* ONE receive(message_t* ONE msg, void* payload, uint8_t len),* messages via Radio from other mote in the network are received. After that, the data will be kept in a buffer to be transmitted from the mote serial port (UART) to the PC.

```
task void uartSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr, src;
    message_t* msg;
    am_group_t grp;
    atomic
      if (uartIn == uartOut && !uartFull)
        {
          uartBusy = FALSE;
          return;
```

```
      }
    msg = uartQueue[uartOut];
    tmpLen = len = call RadioPacket.payloadLength(msg);
    id = call RadioAMPacket.type(msg);
    addr = call RadioAMPacket.destination(msg);
    src = call RadioAMPacket.source(msg);
    grp = call RadioAMPacket.group(msg);
    call UartPacket.clear(msg);
    call UartAMPacket.setSource(msg, src);
    call UartAMPacket.setGroup(msg, grp);
    if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
      call Leds.led1Toggle();
    else
      {
        failBlink();
        post uartSendTask();
      }
  }
```

Figure 55:BaseStationP.nc file III: uartSendTask()

task void uartSendTask() is contained in Figure 55, which sends data to PC from mote serial port

```
  event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
        if (msg == uartQueue[uartOut])
          {
            if (++uartOut >= UART_QUEUE_LEN)
              uartOut = 0;
            if (uartFull)
              uartFull = FALSE;
          }
    post uartSendTask();
  }
```

Figure 56: BaseStationP.nc file IV

.

event void UartSend.sendDone[am_id_t id](message_t msg, error_t error)* when one mote sends a message via Radio interface (Figure 56), Basestation takes the data and sends the message via serial port, releasing the send buffer.

```
  event message_t *UartReceive.receive[am_id_t id](message_t *msg, void *payload, uint8_t len)
  {
    message_t *ret = msg;
    bool reflectToken = FALSE;
    atomic
      if (!radioFull)
        {
```

```
          reflectToken = TRUE;
          ret = radioQueue[radioIn];
          radioQueue[radioIn] = msg;
          if (++radioIn >= RADIO_QUEUE_LEN)
            radioIn = 0;
          if (radioIn == radioOut)
            radioFull = TRUE;
          if (!radioBusy)
            {
              post radioSendTask();
              radioBusy = TRUE;
            }
        }
      else
        dropBlink();
    if (reflectToken) {
    }
    return ret;
  }
```

Figure 57:BaseStationP.nc file V

*event message_t *UartReceive.receive[am_id_-t id] (message_t *msg, void *payload, uint8_t len)* gets received data from the PC by serial port (UART) and the message will be kept in send buffer to be sent by Radio interface to other mote in the network (Figure 57).

```
event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
        if (msg == radioQueue[radioOut])
          {
            if (++radioOut >= RADIO_QUEUE_LEN)
              radioOut = 0;
            if (radioFull)
              radioFull = FALSE;
          }
    post radioSendTask();
  }
}
```

Figure 58: BaseStationP file VI

event void RadioSend.sendDonde[am_id_t id](message_t msg, error_t error)* is called when sent process is correct using Radio interface to other mote in the network, and send buffer will be refreshed by Radio interface and will be released if its buffer is full (Figure 58).

```
COMPONENT=BaseStationC
CFLAGS += -DCC2420_NO_ACKNOWLEDGEMENTS
CFLAGS += -DCC2420_NO_ADDRESS_RECOGNITION
CFLAGS += -DTASKLET_IS_TASK
TINYOS_ROOT_DIR?=../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 59: BaseStation Makefile

In Makefile (Figure 59), the compiler needs to know where is defined the component. Makefile add three CFLAGS where we should indicate to the compiler: no ACK, no address recognition and programmer use task to build the program. Finally, the compiler knows which is the TinyOS root directory using relative URL and one include is added at the botton of the file.

3.3.6. TestPrintf

This is a very simple program to understand how motes work with the printf module. The application is composed by three files: TestPrintfAppC.nc, TestPrintfC.nc and Makefile.

```
//TestPrintfAppC.nc
#define NEW_PRINTF_SEMANTICS
#include "printf.h"
configuration TestPrintfAppC{
}
implementation {
  components MainC, PrintfC, TestPrintfC,SerialStartC;
  components new TimerMilliC();
  TestPrintfC.Boot -> MainC;
  TestPrintfC.Timer -> TimerMilliC;
}
```

Figure 60: TestPrintfAppC.nc file

Figure 60 corresponds to the configuration file where print.h file is included to use the events, methods, and variables defined in the file. There are five components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
//TestPrintfC.nc
#include "printf.h"
module TestPrintfC @safe() {
  uses {
    interface Boot;
    interface Timer<TMilli>;
  }
}
implementation {
  uint8_t dummyVar1 = 123;
  uint16_t dummyVar2 = 12345;
  uint32_t dummyVar3 = 1234567890;
  event void Boot.booted() {
        call Timer.startPeriodic(1000);
  }
  event void Timer.fired() {
        printf("Hi I am writing to you from my TinyOS application!!\n");
        printf("Here is a uint8: %u\n", dummyVar1);
        printf("Here is a uint16: %u\n", dummyVar2);
        printf("Here is a uint32: %lu\n", dummyVar3);
        printfflush();
  }
}
```

Figure 61: TestPrintfC.nc file

Figure 61 is TestPrintfC.nc, where printf.h file has been included at the top of the file. After that, two application interfaces are declared that interact with components interfaces. In the implementation section three variables are declared with different lengths (8, 16 and 32 bits). Once the application is compiled and inserted into a mote, it initializes the Timer with 1000ms as Timer period, and each second several messages are printed on the screen. To see the printed message is possible introducing the following sentence in a new terminal (Figure 62):

```
java net.tinyos.tools.PrintfClient –comm serial@/dev/ttyUSB01:micaz
```

Figure 62: Command to print program replies

```
//Makefile
COMPONENT=TestPrintfAppC
CFLAGS += -I$(TINYOS_OS_DIR)/lib/printf
TINYOS_ROOT_DIR?=../../..
include $(TINYOS_ROOT_DIR)/Makefile.include
```

Figure 63: TestPrintf Makefile

Figure 63 is the default Makefile that is common to all Makefiles. COMPONENT indicates that the configuration file is TestPrintfAppC. One CFLAGS that indicates in which directory is the printf.h file. Another CFLAGS indicates where the TinyOS root directory is using relative URL. In addition, two more CFLAGS can be added to the Makefile program where the printf sentence is used. These CFLAGS are: CFLAGS +=-DNEW_PRINTF_SEMANTICS, where the compiler recognizes the printf sentences, and CFLAGS += -DPRINTF_BUFFER_SIZE=800, which says to the compiler that the printf buffer size is 800 bits.

3.4. Developed Applications

3.4.1. Introduction

The objective is to collect in this section all the developed applications by the author of this project. Different examples have been done to understand how TinyOS and nesC work. Finally, the encryption method implemented is AES and three different modes have been also developed: ECB, CFB and OFB. Every AES mode has two versions: encrypt and decrypt. The applications are ready to work over TinyOS and have been successfully tested over MICAz and TelosB devices.

3.4.2. BlinkSemaforo

BlinkSemaforo is a very simple program originally based on Blink. This new version has a new function. The original Blink program is a counter, which uses Leds to display three rightmost bits of the variable. BlinkSemaforo always has Led1 on. Led0 and Led2 are blinking. When Led0 is on, Led2 is off and vice versa. This application is composed by three files: BlinkSemaforoAppC.nc, BlinkSemaforoC.nc and Makefile.

```
// $Id: BlinkSemaforoAppC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

configuration BlinkSemaforoAppC
{
}
implementation
{
  components MainC, BlinkSemaforoC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer2;

  BlinkSemaforoC -> MainC.Boot;
  BlinkSemaforoC.Timer0 -> Timer0;
  BlinkSemaforoC.Timer2 -> Timer2;
  BlinkSemaforoC.Leds -> LedsC;
}
```

Figure 64: BlinkSemaforoAppC.nc file

Figure 64 is the configuration file where the components are declared. The application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
// $Id: BlinkSemaforoC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include "Timer.h"

module BlinkSemaforoC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 500 );
    call Leds.led1On();
    call Leds.led0Off();
    call Leds.led2On();
  }

  event void Timer0.fired()
  {
        call Leds.led0Toggle();
        call Leds.led2Toggle();
  }
}
```

Figure 65: BlinkSemaforoC.nc file

BlinkSemaforoC.nc (Figure 65) has the same program structure than Blink. Led0, Led1 and Led2 have an initial state. Only one Timer is necessary because when it is over, it toggles Led0 and Led2 state. Timer period is 500ms. The program continues until the user stops the mote.

```
COMPONENT=BlinkSemaforoAppC
include $(MAKERULES)
```

Figure 66: BlinkSemaforo Makefile

Makefile (Figure 66) only shows which the component file is and one sentence where Makerules is included to be used by the compiler.

3.4.3. BlinkToRadioInv

This is the second program that was developed to understand nesC code and TinyOS platform as a variation of BlinkToRadio. BlinkToRadioInv implements an inverse counter and this inverse counter is sent by Radio interface to synchronize all motes that run BlinkToRadioInv program. The files that define BlinkToRadioInv application are BlinkToRadioInv.h, BlinkToRadioInvAppC.nc, BlinkToRadioInvC.nc, and Makefile.

```
// $Id: BlinkToRadioInv.h,v 1.4 2006-12-12 18:22:52 vlahan Exp $

#ifndef BLINKTORADIOINV_H
#define BLINKTORADIOINV_H

enum {
  AM_BLINKTORADIOINV = 6,
  TIMER_PERIOD_MILLI = 250,
```

```
  CONSTANT_GUYID = 0x3333,
  START_COUNT_INV = 0xFFFFFFFF
};

typedef nx_struct BlinkToRadioInvMsg {
  nx_uint16_t nodeid;
  nx_uint16_t guyid;
  nx_uint32_t inv_counter;
} BlinkToRadioInvMsg;
#endif
```

Figure 67: BlinkToRadioInv.h file

BlinkRadioInv.h (Figure 67) is a file included in BlinkToRadioInvAppC.nc and BlinkToRadioC.nc to use the variables and the struct type defined on this file. There are four variables declared and one struct type defined called BlinkToRadioInvMsg. On BlinkToRadioInvMsg there are three declared variables inside. This struct type contains information about which mote sends the message, a constant value, and the inverse counter by Radio interfaces, and will be received by a receiver mote. The inverse counter register is initialized with 0xffffffff value.

```
// $Id: BlinkToRadioInvAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "BlinkToRadioInv.h"

configuration BlinkToRadioInvAppC {
}
implementation {
  components MainC;
  components LedsC;
  components BlinkToRadioInvC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_BLINKTORADIOINV);
  components new AMReceiverC(AM_BLINKTORADIOINV);

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 68: BlinkToRadioInvAppC.nc file

BlinkToRadioInvAppC.nc (Figure 68) includes BlinkToRadioInv.h and Timer.h file. Seven components have been defined. The application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
// $Id: BlinkToRadioInvC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "BlinkToRadioInv.h"

module BlinkToRadioInvC {
  uses interface Boot;
```

```
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

  uint16_t counter;
  uint32_t inv_counter = START_COUNT_INV;
  message_t pkt;
  bool busy = FALSE;

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  event void Boot.booted() {
    call AMControl.start();
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
    if (!busy) {
      BlinkToRadioInvMsg* btrpkt =
        (BlinkToRadioInvMsg*)(call Packet.getPayload(&pkt, sizeof(BlinkToRadioInvMsg)));
      if (btrpkt == NULL) {
        return;
      }
      btrpkt->nodeid = TOS_NODE_ID;
      btrpkt->guyid = CONSTANT_GUYID;
      btrpkt->inv_counter = inv_counter;
      setLeds(inv_counter);
      if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(BlinkToRadioInvMsg)) == SUCCESS) {
        busy = TRUE;
      }
    }
    counter++;
    if (inv_counter != 0)
        {
                inv_counter--;
    } else {
                inv_counter =  START_COUNT_INV;
    }
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
```

```
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(BlinkToRadioInvMsg)) {
      BlinkToRadioInvMsg* btrpkt = (BlinkToRadioInvMsg*)payload;
      setLeds(btrpkt->counter);
    }
    return msg;
  }
}
```

Figure 69: BlinkToRadioInvC.nc file

At the top of Figure 69, there are two includes and eight interfaces declared. In the implementation section variables, methods, events, and tasks are declared. Every time Timer expires, one integer is subtracted to the variable inv_counter and this data with nodeid and guyid is sent by Radio interface. If other mote has BlinkToRadioInv application, then it receives the messages from the first mote via Radio interface. The mote with BlinkToRadioInv application listen the sent messages via Radio, and displays the three rightmost bits of the inverse counter on Leds.

```
COMPONENT=BlinkToRadioInvAppC
include $(MAKERULES)
```

Figure 70: BlinkToRadioInv Makefile

Makefile (Figure 70) only show which is the component file and one sentence where Makerules is included to be used by the compiler.

3.4.4. SenseRadio

SenseRadio is a very simple application created with the objective of learning how to send messages via Radio from an application without this utility. SenseRadio application is developed based on Sense. Sense application has three files and SenseRadio has four files because SenseRadio add extra variables and one struct type to send information using the Radio interface.

```
#ifndef SENSETORADIO_H
#define SENSETORADIO_H

enum {
  AM_SENSETORADIO = 6,
  TIMER_PERIOD_MILLI = 200
};

typedef nx_struct SenseMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
  nx_uint16_t sensedata;
} SenseMsg;

#endif
```

Figure 71: SenseRadio.h file

SenseRadio.h (Figure 71) is a file where different variables are declared. This file is included on SenseRadioAppC.nc and SenseRadioC.nc. In these lines, Timer period, Radio channel number and SenseMsg struct are declared. As the previous applications in data struct, there are three internal variables. When a message is sent, SenseMsg is included in the message payload. When the receiver mote receives the message, it can obtain the desired struct variable.

```
/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */
#include "SenseRadio.h"

configuration SenseRadioAppC
{
}
implementation {

  components SenseRadioC as App;
  components MainC;
  components LedsC;
  components new TimerMilliC();
  components new DemoSensorC() as Sensor;
  components ActiveMessageC;
  components new AMSenderC(AM_SENSETORADIO);
  components new AMReceiverC(AM_SENSETORADIO);


  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer -> TimerMilliC;
  App.Read -> Sensor;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 72: SenseRadioAppC.nc file

SenseRadioAppC.nc (Figure 72) is the configuration file where eight components are declared. This is the configuration file, where application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include "Timer.h"
#include "SenseRadio.h"

module SenseC
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli>;
    interface Read<uint16_t>;
    interface Packet;
    interface AMPacket;
    interface SplitControl as AMControl;
    interface AMSend;
    interface Receive;
  }
}
implementation
{
  uint16_t counter;
  message_t pkt;
  bool busy = FALSE;


  event void Boot.booted() {
    call AMControl.start();
  }

  event void AMControl.startDone(error_t err){
        if (err == SUCCESS) {
```

```
            call Timer.startPeriodic(TIMER_PERIOD_MILLI);
        } else {
                call AMControl.start();
        }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer.fired()
  {
    call Read.read();
  }

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  event void Read.readDone(error_t result, uint16_t data)
  {
        counter++;
        if (!busy)
        {
                SenseMsg* btrpkt = (SenseMsg*)(call Packet.getPayload(&pkt, sizeof(SenseMsg)));
                if (btrpkt == NULL) {
                        return;
                }
                btrpkt->nodeid = TOS_NODE_ID;
                btrpkt->counter = counter;
                btrpkt->sensedata = data;
                setLeds(btrpkt->sensedata);
                if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(SenseMsg)) == SUCCESS)
                {
                busy = TRUE;
                }
        }
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
        if (&pkt == msg) {
                busy = FALSE;
        }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(SenseMsg)) {
      SenseMsg* btrpkt = (SenseMsg*)payload;
    }
    return msg;
  }
}
```

Figure 73: SenseRadioC.nc file

Figure 73 is a module file. At the top of the code, there are two includes and nine interface declarations. After that, there is an implementation section where variables, methods, events, and taks are defined. Once Timer is expired, the mote takes data from Sensorboard and data is set in a variable into SenseRadioMsg struct called sensedata. After that, SenseRadioMsg is set in the payload message ready to be sent to other mote via Radio interface. A second mote

can be reading Radio (Figure 74) interface using the Basestation application and read the entire frame.

```
java net.tinyos.tools.Listen –comm serial@/dev/ttyUSB1:micaz
```

Figure 74: Command to print program replies

```
COMPONENT=BlinkToRadioInvAppC
include $(MAKERULES)
```

Figure 75: SenseRadio Makefile

Figure 75 only shows which is the component file and one include where Makerules are incorporated to be used by the compiler in Makefile.

3.4.5. Common Code [Encrypt and Decrypt]

```
	void initVar()
	{
		rounds_col = sizeof(basedata[0]);
		rounds = sizeof(basedata)/rounds_col;
		i = 0;
		j = 0;
		k = 0;
		it2 = 0;
		it3 = 0;
		it4 = 0;
		it5 = 0;
		it6 = 0;
	}

	void selectColRow(uint8_t data)
	{
		fh = data & fh_ex;	//AND 4bits mayores
		lh = data & lh_ex;	//AND 4bits menores
		fh = fh >> 4;		//desplazamiento 4 bits derecha
	}

	void	rotacionMatrix(uint8_t	matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
	{ //SALIDA result2
	   do
	   {
		if (i == 0)
		{
			j = 0;
			do {
			  result2[i][j]=matrix[i][j];
			  j++;
			} while (j<sizeof(basedata[0]));
			j = 0;
		} else{
		//datos en forma de matriz
		do
		{
		   it2 = j - i;// round_col - round;
		   if (it2 > 0) //perfecto
		   {
			result2[i][it2] = matrix[i][j];
		   } else if (it2 == 0) {
			result2[i][0] = matrix[i][j];
		   } else if (it2 < 0){
			it3 = rounds_col + it2;
			result2[i][it3] = matrix[i][j];
		   }
		   j++;
		} while (j < rounds_col);
		j = 0;
	      }
	      i++;
	   } while (i < rounds);
	}
```

```
        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }

   // ONLY DECRYPT APPLICATIONS

void printMatrix(uint8_t matrix[sizeof(basedata)/sizeof(basedata[0])] [sizeof(basedata[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%#010x , ", matrix[i][j]);
                                j++;
                        } while(j<rounds_col);
                        printf("\n");
                        j = 0;
                        i++;
                } while (i<rounds);
                printfflush();
        }

        void printVector(uint8_t matrix[sizeof(W)])
        {
                initVar();
                for (j; j < sizeof(W); j++)
                {
                        printf("%#010x , ", matrix[j]);
                }
                printf("\n");
                printfflush();
        }

        void printKey(uint8_t matrix[sizeof(key)/sizeof(key[0])][sizeof(key[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%d , ", matrix[i][j]);
                                j++;
                        } while(j<sizeof(key[0]));
                        printf("\n");
                        j = 0;
                        i++;
                } while (i<sizeof(key)/sizeof(key[0]));
                printfflush();
        }
```

Figure 76: Common methods defined

To facilitate the implementation and to make a clear code it is important to use methods (Figure 76). In this case, there are five methods declared and each one has its purpose. The first method is *initVar()* where the main variables used in the code are initialized to zero. The next one is *selectColRow(data)*, where from an 8-bits data, it separates the 4-bits leftmost and 4-bits rightmost. The leftmost 4-bits of the byte are used to select the row value and the rightmost 4-bits are used to select the column value. The method *rotacionMatrix(matrix[][])* is used to rotate the matrix depending on the row number. The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift

is performed. For the fourth row, a 3- byte circular left shift is performed. Based on input data, *sustituciónMatrix(data)* uses *selectColRow(data)* to take the 4-bits leftmost and 4-bits rightmost. After that, once row and column values have been selected, the next step is to replace data in S-box.

Three methods are exclusive in Decrypt applications. The main objective of these examples is to print the matrix [4][4], print a vector [sizeof(data)], and print the extended key [4][44];

```
event void Boot.booted() {
  //INICIALIZACIÓN DE VARIABLES
      initVar();

      //GENERACIÓN DE SUBMATRICECS
      for (i;i<sizeof(key[0]);i++)
      {
              if (i < sizeof(key_original[0]))
              {
                      k = 0;
                      do
                      {
                              key[k][i] = key_original[k][i];
                              k++;
                      } while (k < sizeof(key_original[0]));
              } else {
                      k = 0;
                      do
                      {
                              W_t [k] = key[k][i-1];
                              k++;
                      } while(k < sizeof(W_t));

                      k = 0;
                      if (i%sizeof(key_original[0]) == 0)
                      {
                              it3 = W_t[0];
                              for (k; k < sizeof(W_t); k++)   // FOR ROTACION
                              {
                                      W_tmp[k] = W_t[k+1];
                              }
                              W_tmp[sizeof(W_t)-1] = it3;

                              k = 0;
                              for (k; k < sizeof(W_t); k++)     // FOR SUSTITUCIÓN
                              {
                                      selectColRow(W_tmp[k]);
                                      W_tmp[k] = table [fh][lh];
                              }

                              k = 0;
                              for (k; k < sizeof(W_t); k++)   // FOR XOR con R
                              {
                                  W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                              }
                      }

                      k = 0;
                      for (k; k < sizeof(W_t); k++)           // FOR XOR con R
                      {
                              W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
                      }

                      k = 0;
                      for (k; k < sizeof(W_t); k++)   // FOR ASIGNACIÓN EN KEY
                      {
                              key [k][i] = W_tmp[k];
                      }
              }
      }
```

Figure 77: Key expansion process

Once the code is running, the shared key is expanded into an array of key schedule words (Figure 77). Each word is four bytes and the total key schedule is 44 words for the 128-bit key. Once the key is created is kept in a variable to be used when the implementation requires it.

```
//INITIAL XOR OPERATION
Col_key = 0;
initVar();
do
{
        do
        {
                result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                j++;
        } while (j < rounds_col);
        j = 0; i++; col_key++;
} while(i < rounds);
```

Figure 78: Initial XOR operation

The first operation is just after expanding the shared key. The objective of this operation is to peform a XOR between plaintext and the expanded key (Figure 78). To do that, it is necessary to define a double loop.

```
//ITERACIONES n = 10
b0 = 0;
for(b0;b0 < n;b0++)
{
        //PROCESO DE SUBSTITUCION       //SALE CON result1
        initVar();
        sustitucionMatrix(result_f);

        //PROCESO DE ROTACIÓN          //SALE CON result2
        initVar();
        rotacionMatrix(result1);

        //PROCESO MIXCOLUMNS    //SALE CON result1
        initVar();
        if ( b0 < n-1)
        {
          for (i;i<rounds;i++)
          {
            for (j;j<rounds;j++)
            {
                for (k;k<rounds;k++)
                {
                        selectColRow(M[i][k]);
                        it3 = L_table[fh][lh];
                        selectColRow(result2[k][j]);
                        it5 = L_table[fh][lh];
                        if(fh == 0x00 && lh == 0x00)
                        {
                                detect = TRUE;
                        }
                        it6 = it5 + it3;

                        if(it6-it5 != it3)
                        {
                        it6++;
                        }

                        selectColRow(it6);
                        if (detect == TRUE)
                        {
                                it4 ^= 0;
                                detect = FALSE;
                        } else {
                                it4 ^= E_table[fh][lh];
                        }
                }
                result1[i][j] = it4;
                k = 0; it3 = 0; it4 = 0;
```

```
            }
            j = 0;
         }
        //COPIA MATRIX
        initVar();
        for(i;i < rounds;i++)
        {
           for(j;j < rounds_col;j++)
           {
               result2[i][j] = result1[i][j];
           }
           j = 0;
        }
   }
```

Figure 79: ten-iteration loop (Substitute bytes, ShiftRows, MixColumns and AddRoundKey)

The previous code (Figure 79) is used in all AES-related programs. Because data and key sizes are 128 bits, the needed AES rounds are ten, so a loop is defined to repeat the process ten times. The first step within a round is the substitution process where 8-bit data in a array field is substituted using the selectColRow(data) method and taking the data from S-box table. This process takes a new data, and the new data is kept in a result1 [][] variable. Next step is the rotation process, where data is rotated x positions depending on the row number. For example, the first row is not rotated so, it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Finally, MixColumns is the most complex process. It is executed one time less than the others functions because this process is not executed in the last round. The process starts initializing three for loops where three matrix are used M (or M_inv in DecryptRadioECB), L_table, and E_table. It is necessary to set selectColRow(data) to replace M[4][4] and the result2[4][4] matrix data in the L_table. After replacing the data, fh and lh variables indicate the row and the column number to replace the data using the L_table. It is important to check these two values because if both values are 0x00 the detect variable is set as TRUE. Next, the replaced values are added. Next point is very important, since there is a verification. The addition done in the previous step (it6 = it3 + it5 as shown in Figure 79) can lead to an overflow. Therefore, a verification is needed. The proposed method to check if there has been an overflow is very simple, and basically consist of making the inverse operation, the subtraction. If it6-it5 is different to it3, the result it6 is an overflow and the solution is to add one to it6 because this is the difference between the subtraction and the addition. It is a very simple and very useful solution. it6 data is again substituted using the selectColRow(data) method and taking data from the E_table in this case. In the previous verification, the detect variable has been set as TRUE, it4 is XORed with 0x00 keeping the result in it4 again. After that, the detect variable is set to FALSE, but if the detect variable was FALSE, then it4 is XORed with it6 data substituted in the E_table keeping the result in the it4 variable. When this first loop is over, data is kept in result1[i][j] and the variables are reinitialized. When the second loop is over, the j variable is reinitialized. When the third and last loop is over, result1[][] is copied to result2[][] to start a new iteration. This process is very complete and one of the most difficult to implement. The verifications were developed checking all the problems of the code and the solution were decided looking and comparing the results.

```
    //XOR CIFRADO FINAL
    initVar();
   do
   {
     do
     {
         result_f[j][i] = result2[j][i] ^ key[j][col_key];
         j++;
     } while (j < rounds_col);
```

```
    j = 0; col_key++; i++;
  } while(i < rounds);
```

Figure 80: Final iteration operation

As the first operation was an XOR with plaintext and key, the last operation is also a XOR (Figure 80) where the data is result2[4][4] and the expanded key[][]. This is the last operation in the encryption process, and is used in ECB, CFB, and OFB modes.

3.4.6. EncryptRadioECB

The objective of this Master Thesis was to develop different security solutions over WSN. The EncrypRadioECB application has been developed based on AES encryption using the ECB mode. In our case, AES has a fixed block size of 128 bits and 128 bits of shared key. In this application, data and shared key are imported using a matrix format. All the processes explained on the AES section are developed on the code and will be explained below step-by-step. This program takes data and the shared key in matrix format making some operations with them to obtain the ciphertext. This data is sent via Radio interface on the payload field every time the Timer expires. To work with the ECB mode is necessary to use DecryptRadioECB. DecryptRadioECB takes data from Radio interface, and decrypts the ciphertext using the extended key with the inverse process used in EncryptRadioECB. EncryptRadioECB is developed using four files: EncryptRadioECB.h, EncryptRadioECBAppC.nc, EncryptRadioECBC.nc and Makefile.

```
// $Id: EncryptRadioECB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef ENCRYPTRADIOECB_H
#define ENCRYPTRADIOECB_H

enum {
  AM_ENCRYPTRADIO = 6,
  TIMER_PERIOD_MILLI = 1000
};

typedef nx_struct EncryptRadioECBMsg {
  nx_uint8_t data[16];
} EncryptRadioECBMsg;

#endif
```

Figure 81: EncryptRadioECB.h file

The file where global variables are defined is EncryptRadioECB.h (Figure 81). There are two significant variables, AM_ENCRYPTRADIO, where the Radio channel number six is specified and TIMER_PERIOD_MILLI, that sets the Timer period to 1000ms. It also declares one struct type called EncryptRadioECBMsg where the internal variable is an array [16] called data. This array will be sent via Radio on the Payload field to DecryptRadioECB. This file is imported on EncryptRadioECBAppC.nc and EncryptRadioECBC.nc and let them use the variables defined.

```
// $Id: EncryptRadioECBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioECB.h"

configuration EncryptRadioECBAppC {
}
implementation {
```

```
  components MainC;
  components LedsC;
  components EncryptRadioECBC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 82: EncryptRadioECBAppC.nc file

Figure 82 contains the configuration file. The file EncryptRadioECB.h is included at the top of the file as Timer.h. There are seven components declared where application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
        uint8_t    basedata    [4][4]    =    {{0x32,0x88,0x31,0xE0},    {0x43,0x5A,0x31,0x37},
{0xF6,0x30,0x98,0x07}, {0xA8,0x8D,0xA2,0x34}};
        uint8_t   key_original   [4][4]   =   {{0xAF,0xAE,0xAD,0xAC},   {0xAB,0xAA,0xA9,0xA8},
{0xA7,0xA6,0xA5,0xA4}, {0xA3,0xA2,0xA1,0xA0}};
        uint8_t     M     [4][4]     =     {{0x02,0x03,0x01,0x01},     {0x01,0x02,0x03,0x01},
{0x01,0x01,0x02,0x03}, {0x03,0x01,0x01,0x02}};
```

Figure 83: Variables defined [DecryptRadioECB]

The previous code is not the full file but only three matrix declarations. The code will be divided in several parts and each part will be explained specifically. The variables (Figure 83) and methods (Figure 76) declaration will be set at the implementation section. There are many variables declared and initialized but the main variables are basedata[4][4] and key_original[4][4], where basedata[4][4] is the plaintext and key_original[4][4] is the shared key used to encrypt the plaintext. In both cases, the variables are expressed as square matrixes, with four columns and four rows.

The full code is included in Annexed III. There are several methods implemented whose main goal is to provide a clear code and to simplify its understanding. These methods are declared in *3.4.5. Common Code*.

The main method is Boot.booted() in TinyOS (Figure 77). In this method, the expanded key is developed to generate the key with 44 columns needed by AES 128 bits encryption.

```
 event void Timer0.fired() {
   if (!busy) {
      EncryptRadioECBMsg* btrpkt =
        (EncryptRadioECBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioECBMsg)));
      if (btrpkt == NULL) {
        return;
      }
      initVar();
          for (i;i < sizeof(basedata); i++)
          {
                btrpkt->data[i] = result_f[j][k];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
          }
```

```
      if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(EncryptRadioECBMsg)) == SUCCESS) {
        busy = TRUE;
      }
    }
  }
```

Figure 84: Timer is over. Time to send the message [EncryptRadioECB]

When the Timer is over (Figure 84), the ciphertext is calculated. The calculated ciphertext is expressed as a square matrix[4][4], but it is necessary to insert the bidirectional array into an unidirectional array[16] to be sent by Radio interface. To add the data to the struct type defined in EncryptRadioECB.h (integer array[16]) a loop is required so that every data in the bi-directional array will be set in the unidirectional array orderly. Then, it is necessary to use the same method at DecryptRadioECB to set the received unidirectional array into a bi-directional array.

```
event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioECBMsg)) {
      EncryptRadioECBMsg* btrpkt = (EncryptRadioECBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Figure 85: Message is received via Radio interface [EncryptRadioECB]

Once the data has been sent, the mote also listens to the Radio interface looking for any message. In this case, if the mote listens a message, it takes the payload and uses the setLeds(0x04) method, where Led2 will be on. This process is defined to check if other mote has sent the message (Figure 85).

```
COMPONENT=EncryptRadioECBAppC
include $(MAKERULES)
```

Figure 86: EncryptRadioECB Makefile

The Makefile (Figure 86) only shows which the component file is and one sentence where Makerules is included to be used by the compiler.

3.4.7. DecryptRadioECB

In this program, the mote has to be connected to the mib520 programming board and complement to EncryptRadioECB. DecryptRadioECB obtains the plaintext from the ciphertext and the shared key using the inverse process. DecryptRadioECB is composed by four files: DecryptRadioECB.h, DecryptRadioECBAppC.nc, DecryptRadioECBC.nc and Makefile.

```
// $Id: DecryptRadioECB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef DECRYPTRADIOECB_H
#define DECRYPTRADIOECB_H
enum {
  AM_ENCRYPTRADIO = 6,
};

typedef nx_struct DecryptRadioECBMsg {
  nx_uint8_t data[16];
} DecryptRadioECBMsg;
#endif
```

Figure 87: DecryptRadioECB.h

The file where global variables are defined is DecryptRadioECB.h (Figure 87). AM_ENCRYPTRADIO is the only variable defined. This variable sets the used Radio channel to transmit and receive data as number six. It also declares one struct type called DecryptRadioECBMsg where data which is sent is in the data[16] variable. This array will be used to keep the information received from EncryptRadioECB via Radio contained on the Payload field. This file is imported on DecryptRadioECBAppC.nc and DecryptRadioECBC.nc and lets them use the variables defined.

```
// $Id: DecryptRadioECBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "DecryptRadioECB.h"
#include "printf.h"

configuration DecryptRadioECBAppC {
}
implementation {
  components MainC;
  components LedsC;
  components DecryptRadioECBC as App;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);
  components PrintfC;
  components SerialStartC;

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 88: DecryptRadioECBAppC.nc

Figure 88 contains the configuration file. DecryptRadioECB.h is included at the top of the file as Timer.h. There are eight components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
// $Id: DecryptRadioECBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
        uint8_t basedata [4][4];
        uint8_t M_inv [4][4]= {{0x0E, 0x0B, 0x0D, 0x09}, {0x09, 0x0E, 0x0B, 0x0D}, {0x0D, 0x09, 
0x0E, 0x0B}, {0x0B, 0x0D, 0x09, 0x0E}};
        uint8_t  key_original  [4][4]=  {{0xAF,  0xAE,  0xAD,  0xAC},  {0xAB,  0xAA,  0xA9,  0xA8}, 
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];

        uint8_t table_inv [16][16] = {{0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 
0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB}, {0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 
0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB}, {0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 
0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E}, {0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 
0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25}, {0x72, 0xF8, 0xF6, 0x64, 0x86, 
0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92}, {0x6C, 0x70, 0x48, 0x50, 
0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84}, {0x90, 0xD8, 0xAB, 
0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06}, {0xD0, 0x2C, 
0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B}, {0x3A, 
0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73}, 
{0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 
```

```
0x6E}, {0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18,
0xBE, 0x1B}, {0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78,
0xCD, 0x5A, 0xF4}, {0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59,
0x27, 0x80, 0xEC, 0x5F}, {0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A,
0x9F, 0x93, 0xC9, 0x9C, 0xEF}, {0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB,
0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61}, {0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1,
0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D}};
```

Figure 89: Variables declarations [DecryptRadioECB]

The necessary variables to DecryptRadioECB are M_inv and the shared key as key_original. The variable basedata is declared to keep the received data via Radio and the variable result_f is where the plaintext will be kept once has been decrypted. A difference regarding other applications is that table_inv is declared as the inverse S-box and is used in the MixColumns process (Figure 76 and Figure 89).

After all variable and method declarations, *event void Boot.booted()* is declared (Figure 77) and it has the same function as "main" has in other programming languages. In this part, the extended key is generated and is kept in a key variable, ready to be used when necessary.

```
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(DecryptRadioECBMsg)) {
      DecryptRadioECBMsg* btrpkt = (DecryptRadioECBMsg*)payload;
        setLeds(0x04);
        col_key = 40;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
                basedata[j][k] = btrpkt->data[i];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
        }
        //IMPRIMIR RESULTADO FINAL DESCIFRADO
        printf("Print Received Data: \n");
        printMatrix(basedata);

        //XOR CIFRADO INICIAL
        initVar();
        do
        {
                do
                {
                        result2[j][i] = basedata[j][i] ^ key[j][col_key];
                        j++;
                } while (j < rounds_col);
                j = 0; i++; col_key++;
        } while(i < rounds);

        //ITERACIONES n = 10
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
                //PROCESO DE SUBSTITUCION  //SALE CON result1
                initVar();
                sustitucionMatrix(result2);

                //PROCESO DE ROTACIÓN A DERECHAS  //SALE CON result2
                initVar();
                rotacionMatrix(result1);

                col_key = col_key - 8;

                //XOR CIFRADO FINAL
                initVar();
                do
                {
```

```
                        do
                        {
                                result1[j][i] = result2[j][i] ^ key[j][col_key];
                                j++;
                        } while (j < rounds_col);
                        j = 0; col_key++; i++;
                } while(i < rounds);

                //COPIA MATRIX
                initVar();
                for(i;i < rounds;i++)
                {
                        for(j;j < rounds_col;j++)
                        {
                                result2[i][j] = result1[i][j];
                        }
                        j = 0;
                }

                //PROCESO MIXCOLUMS  //SALE CON result1
                initVar();
                if ( b0 < n-1)
                {
                    for (i;i<rounds;i++)
                    {
                        for (j;j<rounds;j++)
                        {
                            for (k;k<rounds;k++)
                            {
                                selectColRow(M_inv[i][k]);
                                it3 = L_table[fh][lh];
                                selectColRow(result1[k][j]);
                                it5 = L_table[fh][lh];
                                if(fh == 0x00 && lh == 0x00)
                                {
                                        detect = TRUE;
                                }

                                it6 = it5 + it3;
                                if(it6-it5 != it3)
                                {
                                    it6++;
                                }

                                selectColRow(it6);
                                if (detect == TRUE)
                                {
                                    it4 ^= 0;
                                    detect = FALSE;
                                } else {
                                    it4 ^= E_table[fh][lh];
                                }
                             }
                             result2[i][j] = it4;
                             k = 0; it3 = 0; it4 = 0;
                        }
                        j = 0;
                      }
                }
            }
            //IMPRIMIR RESULTADO FINAL DESCIFRADO
            printf("Print Decrypted Result: \n");
            printMatrix(result2);
    }
    return msg;
  }
}
```

Figure 90: Receiving process message [DecryptRadioECB]

The program continues receiving data every time the Timer expires via Radio interface. Once the data has been received as a unidirectional array[16], data is ordered in a matrix[4][4] called basedata. From this point, data is processes to obtain plaintext using also the extended

key and the M_inv variable. In this step, the received ciphertext is printed to check if the ciphertext is correct comparing with the output of the online AES calculator. After that, the first operation is XOR with the received data via Radio interface in basedata[4][4] matrix and the expanded key, taking columns from leftmost to rightmost. The application continues with ten iterations where several operations are carried out. The first one is a substitution process, where data is separated in 4-bit leftmost and 4-bits rightmost and is substituted as indicated by the S-box. After that, there is a rotation process where the first row is not rotated, so it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Finally, col_key is refreshed reinitializing the value. The next process is an XOR operation between the result of rotacionMatrix(data[4][4]) and the expanded key, taking columns from leftmost ot rightmost. The intermediate variable (result1[4][4]) is copied in other intermediated variable (result2[4][4]). The MixColumn operation is executed always but in the last iteration. This process is the same as in EncryptRadioECB with the difference that in this case the M_inv matrix is used instead of employing the M matrix . When the whole the process is finished, the printMatrix method is used to print the obtained plaintext and to check that the obtained plaintext is the same as the plaintext originally used at EncryptRadioECB (Figure 90).

```
COMPONENT=DecryptRadioECBAppC
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS
CFLAGS += -DPRINTF_BUFFER_SIZE=800
include $(MAKERULES)
```

Figure 91: DecryptRadioECB Makefile

There are some differences compared to other Makefile files (Figure 91). There are two basic lines where the programmer defines which the component is and an include Makerules to add different rules to the compiler. In addition, there are three CFLAGS, which indicate to the compiler where the printf.h file is, a sentence to recognize the printf sentences, and the printf buffer size is initialized to 800.

3.4.8. EncryptRadioCFB

The CFB mode assumes that the transmission unit is 8 bits, so AES blocks of 128bits are divided into segments of 8 bits. It is necessary to set an IV (Initalization Vector). The leftmost 8 bits of the output of the encryption function are XORed with the first byte of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted via Radio interface. Then, the IV is shifted left 8 bits throwing leftmost 8 bits. The rightmost 8 bits are free and C_1 is placed there, thus refreshing the IV at each iteration. This process continues until all plaintext units (bytes) have been encrypted. There are four files required to build the EncryptRadioCFB program: EncryptRadioCFB.h, EncryptRadioCFBAppC.nc, EncryptRadioCFBC.nc, and Makefile.

```
// $Id: EncryptRadioCFB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef ENCRYPTRADIOCFB_H
#define ENCRYPTRADIOCFB_H

enum {
  AM_ENCRYPTRADIO = 6,
  TIMER_PERIOD_MILLI = 1000
};
typedef nx_struct EncryptRadioCFBMsg {
  nx_uint8_t data;
} EncryptRadioCFBMsg;
#endif
```

Figure 92: EncryptRadioCFB.h file

The file where the global variables are defined is EncryptRadioCFB.h (Figure 92). There are two variables, AM_ENCRYPTRADIO where the Radio channel number in use is set to six to send and receive messages, and TIMER_PERIOD_MILLI, where the Timer period is set to1000ms (1s). It also declares one struct type called EncryptRadioCFBMsg where inside, the variable data, 8 bits, is defined. The variable data will be sent via Radio on the Payload field to DecryptRadioCFB. This file is imported on EncryptRadioCFBAppC.nc and EncryptRadioCFBC.nc and lets them use the defined variables.

```
// $Id: EncryptRadioCFBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioCFB.h"
#include "printf.h"

configuration EncryptRadioCFBAppC {
}
implementation {
  components MainC;
  components LedsC;
  components EncryptRadioCFBC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 93: EncryptRadioCFBAppC.nc file

Figure 93 contains the configuration file. EncryptRadioCFB.h is included at the top of the file as Timer.h. There are seven components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
        uint8_t                    basedata                    [4][4]                    =
{{0x11,0x22,0x33,0x44},{0x55,0x66,0x77,0x88},{0x99,0xAA,0xBB,0xCC},{0xDD,0xEE,0xFF,0x00}};
        uint8_t                  text                  [sizeof(basedata)]                =
{0x32,0x88,0x31,0xE0,0x43,0x5A,0x31,0x37,0xF6,0x30,0x98,0x07,0xA8,0x8D,0xA2,0x34};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01,
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
```

Figure 94: Variables declared [EncryptRadioCFB]

There are many variables declared (Figure 76 and Figura 94) but the most important variables in the code are: basedata [4][4] will be the IV; text[16] is a unidirectional array data to be ciphered; M[4][4] will be the matrix to make the MixColumn process; and key_original[4][4] (shared key) will be the seed to generate the expanded key. In addition, there are multiples variables and methods declared to make more dynamic the code and can be used in any part of the code.

The *event void Boot.booted()* is also defined in the code (Figure 77). When this event is run, the extended key is generated. This happens at the beginning of the application, keeping data in an array with the corresponding sizes.

```
event void Timer0.fired() {
   if (!busy) {
        EncryptRadioCFBMsg* btrpkt =
        (EncryptRadioCFBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioCFBMsg)));
        if (btrpkt == NULL) {
           return;
        }

        //XOR CIFRADO INICIAL
        initVar();
        col_key = 0;

        do
        {
               do
               {
                   result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                   j++;
               } while (j < rounds_col);
               j = 0; i++; col_key++;
        } while(i < rounds);


        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
           //PROCESO DE SUSTITUCION  //SALE CON result1
           initVar();
           sustitucionMatrix(result_f);

           //PROCESO DE ROTACIÓN     //SALE CON result2
           initVar();
           rotacionMatrix(result1);

           //PROCESO MIXCOLUMS     //SALE CON result1
           initVar();
           if ( b0 < n-1)
           {
               for (i;i<rounds;i++)
               {
                   for (j;j<rounds;j++)
                   {
                        for (k;k<rounds;k++)
                        {
                           selectColRow(M[i][k]);
                           it3 = L_table[fh][lh];

                           selectColRow(result2[k][j]);
                           it5 = L_table[fh][lh];
                           if(fh == 0x00 && lh == 0x00)
                           {
                                   detect = TRUE;
                           }

                           it6 = it5 + it3;
                           if(it6-it5 != it3)
                           {
                                   it6++;
                           }

                           selectColRow(it6);
                           if (detect == TRUE)
                           {
                               it4 ^= 0;
                               detect = FALSE;
                           } else {
                               it4 ^= E_table[fh][lh];
```

```
                        }
                    }
                    result1[i][j] = it4;
                    k = 0; it3 = 0; it4 = 0;
                }
                j = 0;
            }

            //COPIA MATRIX
            initVar();
            for(i;i < rounds;i++)
            {
                for(j;j < rounds_col;j++)
                {
                        result2[i][j] = result1[i][j];
                }
                j = 0;
            }
        }

        //XOR CIFRADO FINAL
        initVar();
        do
        {
            do
            {
                result_f[j][i] = result2[j][i] ^ key[j][col_key];
                j++;
            } while (j < rounds_col);
            j = 0; col_key++; i++;
        } while(i < rounds);
    }
```

Figure 95: Timer is over, time to send one message [EncryptRadioCFB]

From this point, data is processed to obtain cipherdata using the expanded key, the IV, and the M matrix with text array, which contains the plaindata. The first step is common to all ciphering processes, there is a initial XOR cipher where basedata[4][4], which includes the IV, is XORed with the expanded key. Again, it is necessary to define a loop with ten iterations as defined in the AES algorithm to obtain the cipherdata. The first method within the round is the substitution process, where data is separated in 4-bit leftmost and 4-bits rightmost and is substituted in the S-box. Then, it continues with the rotation process, where the first row is not rotated, so it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Finally, the MixColumns method starts, which is the most complex process. It is executed one time less than the other operations because this process is not executed in the last round. The MixColumn result is copied in another variable to start a new iteration and when the loop is finished, the final operation is an XOR between the MixColumn result and the expanded key (Figure 95).

```
initVar();
for (i;i < sizeof(basedata); i++)
{
    v_fila[i] = basedata[j][k];
    j++;

    if (j>3)
    {
         j = 0; k++;
    }
 }
 it3 = result_f[0][0] ^ text[d3];
 d3++;

 i = 0;
 for (i;i < sizeof(basedata)-1; i++)
 {
```

```
            v_fila[i] = v_fila[i+1];
         }

         v_fila[sizeof(basedata)-1] = it3;

         initVar();
         for (i;i < sizeof(basedata); i++)
         {
            basedata[j][k] = v_fila[i];
            j++;

            if (j>3)
            {
                 j = 0; k++;
            }
         }

         btrpkt->data = it3;

     if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(EncryptRadioCFBMsg)) == SUCCESS)
         {
            busy = TRUE;
         }
      }
   }
```

Figure 96: Specific characteristics of application [EncryptRadioCFB]

Now, it is time to define the specific characteristic of this application (Figure 96). The first operation is to set the basedata[4][4] in a array[16] using one loop. The program continues XORing result_f[0][0] with the plaindata and the result is the cipherdata. Then, the vector is reordenated where 8 leftmost bits are discarded and the remaining data is moved one-byte left position in the array. Following the previous steps, the 8 bits right most vector position is empty so the cipherdata is stored in this position. Again, data is ordered as matrixes and variables are resfreshed to be used in the next round to cipher the next plaindata when the Timer expires. The data is sent via Radio adding the cipherdata to the data integer variable in EncryptRadioCFBMsg struct with the sentence btrpkt->data = it3.

```
  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioCFBMsg)) {
      EncryptRadioCFBMsg* btrpkt = (EncryptRadioCFBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Figure 97: Mote receive a message [EncryptRadioCFB]

EncryptRadioCFB has not any specific task when it listens any other mote sending messages via Radio. The application detects the message, takes the message payload and uses the setLeds(0x04) method (Figure 97), where Led2 will be on and the user can check that a message is being sent by another mote.

```
COMPONENT=EncryptRadioCFBAppC
include $(MAKERULES)
```

Figure 98:EncryptRadioCFB Makefile

The Makefile (Figure 98) only shows which the component file is and one sentence where Makerules is included to be used by the compiler.

3.4.9. DecryptRadioCFB

As it was explained in EncryptRadioCFB, the CFB mode assumes that the reception unit is 8 bits, so, the block of 128bits is divided into segments of 8 bits. It is necessary to set the IV, which is common to both EncryptRadioCFB and DecrypRadioCFB. The program receives 8 bits of cipherdata and starts with the protocol to decrypt it. To do so, this program uses the ECB mode to calculate the ciphertext taking the IV and the expanded key. In addition, only the result[0][0] is taken and is XORed with the cipherdata to obtain the plaindata. The received cipherdata is, at the end of the process, inserted at the end of the IV taking off the leftmost byte value. There are four files needed to build the DecryptRadioCFB program: DecryptRadioCFB.h, DecryptRadioCFBAppC.nc, DecryptRadioCFBC.nc,s and Makefile.

```
// $Id: DecryptRadioCFB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef DECRYPTRADIOCFB_H
#define DECRYPTRADIOCFB_H

enum {
  AM_ENCRYPTRADIO = 6,
};

typedef nx_struct DecryptRadioCFBMsg {
  nx_uint8_t data;
} DecryptRadioCFBMsg;

#endif
```

Figure 99: DecryptRadioCFB.h file

AM_ENCRYPTRADIO value is defined in DecryptRadioCFBC.nc. This variable sets the used Radio channel to transmit and receive the information as number six. It also declares one struct type called DecryptRadioCFBMsg where the variable is an 8-bit integer called data. This file is imported on DecryptRadioCFBAppC.nc and DecryptRadioCFBC.nc. It is important to import it because it lets the files to use the defined variables (Figure 99).

```
// $Id: DecryptRadioCFBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $
/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */
#include <Timer.h>
#include "DecryptRadioCFB.h"
#include "printf.h"

configuration DecryptRadioCFBAppC {
}
implementation {
  components MainC;
  components LedsC;
  components DecryptRadioCFBC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);
  components PrintfC;
  components SerialStartC;

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 100: DecryptRadioCFBAppC.nc file

Figure 100 contains is the configuration file. DecryptRadioECB.h is included at the top of the file as Timer.h. There are nine components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
	uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99, 0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
	uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01, 0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
	uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8}, {0xA7, 0xA6, 0xA5, 0xA4}, {0xA3 ,0xA2, 0xA1, 0xA0}};
```

Figure 101: Variables declared for DecryptRadioCFB

There are many variables declared (Figure 76 and Figure 101) but the most important variables in the code are: basedata [4][4] will be the IV; M[4][4] will be the matrix to make the MixColumn process; and key_original[4][4] (shared key) will be the seed to generate the expanded key. In addition, there are multiples variables and methods declared to make the code more dynamic and can be used in any part of the code.

The *event void Boot.booted()* (Figure 77) is also defined in the code. When this event is run, the extended key is generated. This happens at the beginning of the application, keeping data in an array with the required sizes.

```
  event void Timer0.fired() {
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(DecryptRadioCFBMsg)) {
      DecryptRadioCFBMsg* btrpkt = (DecryptRadioCFBMsg*)payload;
      setLeds(0x04);
      d3 = btrpkt->data;
      printf("Dato recibido sin descifrar: %#010x\n",d3);
      printfflush();

        //XOR CIFRADO INICIAL
        initVar();
        col_key = 0;

        do
        {
            do
            {
                result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                j++;
            } while (j < rounds_col);
            j = 0; i++; col_key++;
        } while(i < rounds);

        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
            //PROCESO DE SUSTITUCION    //SALE CON result1
            initVar();
            sustitucionMatrix(result_f);

            //PROCESO DE ROTACIÓN       //SALE CON result2
            initVar();
            rotacionMatrix(result1);

            //PROCESO MIXCOLUMS         //SALE CON result1
            initVar();
            if ( b0 < n-1)
            {
                for (i;i<rounds;i++)
                {
                    for (j;j<rounds;j++)
```

```
                {
                    for (k;k<rounds;k++)
                    {
                         selectColRow(M[i][k]);
                         it3 = L_table[fh][lh];

                         selectColRow(result2[k][j]);
                         it5 = L_table[fh][lh];
                         if(fh == 0x00 && lh == 0x00)
                         {
                            detect = TRUE;
                         }

                        it6 = it5 + it3;

                        if(it6-it5 != it3)
                        {
                            it6++;
                        }

                        selectColRow(it6);
                        if (detect == TRUE)
                        {
                            it4 ^= 0;
                            detect = FALSE;
                        } else {
                            it4 ^= E_table[fh][lh];
                        }
                    }
                    result1[i][j] = it4;
                    k = 0; it3 = 0; it4 = 0;
                }
                j = 0;
            }

            //COPIA MATRIX
            initVar();
            for(i;i < rounds;i++)
            {
                 for(j;j < rounds_col;j++)
                 {
                    result2[i][j] = result1[i][j];
                 }
                 j = 0;
            }
       }

       //XOR CIFRADO FINAL
       initVar();
       do
       {
            do
            {
                    result_f[j][i] = result2[j][i] ^ key[j][col_key];
                    j++;
            } while (j < rounds_col);
            j = 0; col_key++; i++;
       } while(i < rounds);
   }
```

Figure 102: Decrypt proccess when a mote receive a message via Radio[DecryptRadioCFB]

Different events are declared but at this point they are not relevant. The program continues receiving data from the sent messages via Radio. In this case, the received data is an 8-bit integer type; it is the cipherdata and is kept in the d3 variable. EncryptRadioCFB and DecryptRadioCFB share two variables: the IV, defined as basedata[4][4], and the shared key, defined as key_original[4][4]. From key_original, the extended key is generated. DecryptRadioCFB starts as EncryptRadioECB, with an XOR between basedata[i][j] and key[i][j]. After that, a loop starts with ten iterations where several operations are carried out. The first one is the substitution process where data is separated in 4-bit leftmost and 4-bits

rightmost and then, they are substituted as indicated in the S-box. After that, the rotation operation is performed, where the first row is not rotated so, it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Third process is MixColumns that is the most complex process. In spite of being a decrypt application, the M[4][4] variable is used in this code because DecryptRadioCFB uses an EncryptRadioECB part. MixColumn process is executed one time less than other processes because it is not executed in the last round. The MixColumn result is copied into aother variable to start a new iteration and when the loop is finished, the final operation is an XOR between the MixColumn result and the expanded key (Figure 102).

```
        initVar();
        for (i;i < sizeof(basedata); i++)
        {
                v_fila[i] = basedata[j][k];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
        }

        it3 = result_f[0][0] ^ d3;
        printf("Dato descifrado: %#010x\n",it3);
        printfflush();

        i = 0;
        for (i;i < sizeof(basedata)-1; i++)
        {
                v_fila[i] = v_fila[i+1];
        }
        v_fila[sizeof(basedata)-1] = d3;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
                basedata[j][k] = v_fila[i];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
        }
    }
    return msg;
  }
}
```

Figure 103: Especific characteristics of application [DecryptRadioCFB]

Now, it is time to define the specific characteristic of this application (Figure 103). The first operation is to set the basedata[4][4] in a array[16] using one loop. The program continues XORing result_f[0][0] with the received cipherdata and the result is the plaindata. Then, the vector is reordered where the 8 bits leftmost are discarded and the remaining bits are shifted one-byte left position in the array. Following the previous steps, the 8 bits right most vector position is empty so the received cipherdata, d3, is included in this position. Again, data is ordered as matrix and variables are resfreshed to be used in the next round to decrypt the next cipherdata when data is received.

```
COMPONENT=DecryptRadioCFBAppC
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS
CFLAGS += -DPRINTF_BUFFER_SIZE=800
include $(MAKERULES)
```

Figure 104: DecryptRadioCFB Makefile

There are some differences compared to other Makefile files. There are two basic lines where the programmer defines which the component is and an include Makerules to add different rules to the compiler. In addition, there are three CFLAGS, which indicate to the compiler where the printf.h file is, a sentence to recognize the printf sentences, and the printf buffer size is initialized to 800 (Figure 104).

3.4.10. EncryptRadioOFB

The OFB mode assumes that the transmission unit is 8 bits, so AES blocks of 128bits are divided into segments of 8 bits. It is necessary to set an IV. The leftmost 8 bits of the output of the encryption block are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted via Radio interface. The IV is shifted left 8 bits discarding those leftmost 8 bits. The rightmost 8 bits are free and C_1 is placed there refreshing the IV at each iteration IV. This process continues until all plaintext units (bytes) have been encrypted. There are four files necessaries to build the EncryptRadioOFB program: EncryptRadioOFB.h, EncryptRadioOFBAppC.nc, EncryptRadioOFBC.nc, and Makefile.

```
// $Id: EncryptRadioOFB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef ENCRYPTRADIOOFB_H
#define ENCRYPTRADIOOFB_H

enum {
  AM_ENCRYPTRADIO = 6,
  TIMER_PERIOD_MILLI = 5000
};

typedef nx_struct EncryptRadioOFBMsg {
  nx_uint8_t data;
} EncryptRadioOFBMsg;

#endif
```

Figure 105: EncryptRadioOFB.h file

The file where the global variables are defined is EncryptRadioOFB.h (Figure 105). There are two variables called AM_ENCRYPTRADIO, where the Radio channel to use is set to number six, and TIMER_PERIOD_MILLI, where theTimer period is set to 1000ms (1s). It also declares one struct type called EncryptRadioOFBMsg ,where the variable data is defined. The variable data will be sent via Radio on the Payload field to DecryptRadioOFB. This file is imported on EncryptRadioOFBAppC.nc and EncryptRadioOFBC.nc and lets them use the variables defined.

```
// $Id: EncryptRadioOFBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioOFB.h"
#include "printf.h"

configuration EncryptRadioOFBAppC {
}
implementation {
  components MainC;
  components LedsC;
  components EncryptRadioOFBC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);
```

```
  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 106: EncryptRadioOFBAppC.nc file

Figure 106 contains the configuration file. EncryptRadioOFB.h is included at the top of the file as Timer.h. There are seven components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
        uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99, 0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
        uint8_t text [sizeof(basedata)] = {0x32, 0x88, 0x31, 0xE0, 0x43, 0x5A, 0x31, 0x37, 0xF6, 0x30, 0x98, 0x07, 0xA8, 0x8D, 0xA2, 0x34};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01, 0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8}, {0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
```

Figure 107: Variables declared [EncryptRadioOFB]

There are many variables declared (Figure 76 and Figure 107) but the most important variables in the code are: basedata [4][4] will be the IV; text[16] a unidirectional array data to be ciphered; M[4][4] will be the matrix to make the MixColumn process; and key_original[4][4] (shared key) will be the seed to generate the expanded key. In addition, there are multiples variables and methods declared to make more dynamic the code and can be used in any part of the code.

The event void Boot.booted() (Figure 77) is also defined in the code. When this event is run, the extended key is generated. This happens at the beginning of the application, keeping data in an array with the required sizes.

```
  event void Timer0.fired() {
    if (!busy) {
      EncryptRadioOFBMsg* btrpkt =
        (EncryptRadioOFBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioOFBMsg)));
      if (btrpkt == NULL) {
        return;
      }

        //XOR CIFRADO INICIAL
        initVar();
        col_key = 0;
        do
        {
           do
           {
                result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                j++;
           } while (j < rounds_col);
           j = 0; i++; col_key++;
        } while(i < rounds);


        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
           //PROCESO DE SUSTITUCION     //SALE CON result1
           initVar();
```

```
            sustitucionMatrix(result_f);

            //PROCESO DE ROTACIÓN    //SALE CON result2
            initVar();
            rotacionMatrix(result1);

            //PROCESO MIXCOLUMS          //SALE CON result1
            initVar();
            if ( b0 < n-1)
            {
                 for (i;i<rounds;i++)
                 {
                     for (j;j<rounds;j++)
                     {
                         for (k;k<rounds;k++)
                         {
                             selectColRow(M[i][k]);
                             it3 = L_table[fh][lh];

                             selectColRow(result2[k][j]);
                             it5 = L_table[fh][lh];
                             if(fh == 0x00 && lh == 0x00)
                             {
                                 detect = TRUE;
                             }

                             it6 = it5 + it3;
                             if(it6-it5 != it3)
                             {
                                 it6++;
                             }

                             selectColRow(it6);
                             if (detect == TRUE)
                             {
                                 it4 ^= 0;
                                 detect = FALSE;
                             } else {
                                 it4 ^= E_table[fh][lh];
                             }
                         }
                         result1[i][j] = it4;
                         k = 0; it3 = 0; it4 = 0;
                     }
                     j = 0;
                 }

                 //COPIA MATRIX
                 initVar();
                 for(i;i < rounds;i++)
                 {
                     for(j;j < rounds_col;j++)
                     {
                         result2[i][j] = result1[i][j];
                     }
                     j = 0;
                 }
              }

              //XOR CIFRADO FINAL
              initVar();
              do
              {
                 do
                 {
                       result_f[j][i] = result2[j][i] ^ key[j][col_key];
                       j++;
                 } while (j < rounds_col);
                 j = 0; col_key++; i++;
              } while(i < rounds);
         }
```

Figure 108: Time is over. Time to send a message [EncryptRadioOFB]

From this point, data is processed to obtain the cipherdata using the expanded key, the IV, and the M matrix with the text array, which contains the plaindata. The first step is common to all ciphering processes, there is an initial XOR cipher where basedata[4][4], which represents the IV, is XORed with the expanded key. It is necessary to define a loop with ten iterations because is part of the process to obtain the cipherdata. The first method within the first round is the substitution process, where data is separated in 4-bit leftmost and 4-bits rightmost and is substituted as indicated in the S-box. Then, it continues with a rotation process, where the first row is not rotated so, it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Finally, the MixColumns method starts, which is the most complex process. It is executed one time less than other processes because this process is not executed in the last round. The MixColumn result is copied into another variable to start a new iteration and when the loop is finished, the final operation is an XOR between the MixColumn result and the expanded key (Figure 108).

```
        initVar();
        printf("Imprimir matriz tras cifrado\n");
        printMatrix(result_f);

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
              v_fila[i] = basedata[j][k];
              j++;

              if (j>3)
              {
                   j = 0; k++;
              }
        }

        it3 = result_f[0][0] ^ text[d3];
        d3++;

        i = 0;
        for (i;i < sizeof(basedata)-1; i++)
        {
             v_fila[i] = v_fila[i+1];
        }
        v_fila[sizeof(basedata)-1] = result_f[0][0];
        btrpkt->data = it3;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
             basedata[j][k] = v_fila[i];
             j++;

             if (j>3)
             {
                  j = 0; k++;
             }
        }

        if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(EncryptRadioOFBMsg)) == SUCCESS) {
        busy = TRUE;
        }
    }
 }
```

Figure 109: Specific characteristics of application [EncryptRadioOFB]

Now, it is time to define the specific characteristic of this application (Figure 109). The first operation is to set the basedata[4][4] in a array[16] using a loop. The program continues XORing result_f[0][0] with plaindata and the result is the cipherdata. This cipherdata is kept in

the it3 variable. Then, the vector is reordenated where eight leftmost bits are discarded and the remaining data is moved one (byte) left position in the vector. Following the previous steps, the last vector position is empty so the result_f[0][0] is included in this position. Finally, data is reordered as a matrix (basedata[4][4]) and variables are resfreshed to be used in the next round when the mote will receive a new message via the Radio interface.

```
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioOFBMsg)) {
      EncryptRadioOFBMsg* btrpkt = (EncryptRadioOFBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Figure 110: Specific task when a mote receive a message[EncryptRadioOFB]

EncryptRadioCFB has not any specific task when it listens some other mote sending messages via Radio (Figure 110). The application detects the message, takes the message payload, and uses the setLeds(0x04) method, where Led2 will be on and the user can check that a message is being sent by another mote.

```
COMPONENT=EncryptRadioOFBAppC
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS
CFLAGS += -DPRINTF_BUFFER_SIZE=800
include $(MAKERULES)
```

Figure 111: EncryptRadioOFB Makefile

The Makefile (Figure 111) only shows which the component file is and one sentence where Makerules is included to be used by the compiler.

3.4.11. DecryptRadioOFB

The reception unit is 8 bits, so the blocks of 128 bits are divided into segments of 8 bits. It is necessary to set an IV, which is common to both EncryptRadioOFB and DecrypRadioOFB. The program receives cipherdata (8 bits) and starts to decrypt it. To do so, this program uses the OCB mode to calculate the cipherdata using the IV and the shared key. In addition, only the result[0][0] from the encrypt block is taken and is XORed with the cipherdata to obtain plaindata. The result[0][0] from encrypt block is, at the end of the process, inserted at the end of the IV taking off the leftmost value. There are four files necessaries to build the DecryptRadioOFB program: DecryptRadioOFB.h, DecryptRadioOFBAppC.nc, DecryptRadioOFBC.nc, and Makefile.

```
// $Id: DecryptRadioOFB.h,v 1.4 2016-06-04 22:15:00 vlahan Exp $

#ifndef DECRYPTRADIOOFB_H
#define DECRYPTRADIOOFB_H

enum {
  AM_ENCRYPTRADIO = 6,
};

typedef nx_struct DecryptRadioOFBMsg {
  nx_uint8_t data;
} DecryptRadioOFBMsg;

#endif
```

Figure 112: DecryptRadioOFB.h file

The file where global variables are defined is DecryptRadioOFB.h (Figure 112). There is one variable, AM_ENCRYPTRADIO. This variable sets the used Radio channel to transmit

and receive the information as number six. It also declares one struct type called DecryptRadioOFBMsg where the variable is an 8-bit integer called data. This file is imported on DecryptRadioOFBAppC.nc and DecryptRadioOFBC.nc and lets them use the defined variables.

```
// $Id: DecryptRadioOFBAppC.nc,v 1.5 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "DecryptRadioOFB.h"
#include "printf.h"

configuration DecryptRadioOFBAppC {
}
implementation {
  components MainC;
  components LedsC;
  components DecryptRadioOFBC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_ENCRYPTRADIO);
  components new AMReceiverC(AM_ENCRYPTRADIO);
  components PrintfC;
  components SerialStartC;

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

Figure 113: DecrtptRadioOFBAppC.nc file

Figure 113.nc contains the configuration file. DecryptRadioOFB.h is included at the top of the file as Timer.h. There are nine components declared. Application and component interfaces are wired to define how the application works and how to use different methods defined by TinyOS.

```
        uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99, 
0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01, 
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8}, 
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t v_fila [sizeof(basedata)];
```

Figure 114: Variables declare in DecryptRadioOFB

There are many variables declared (Figure 76 and Figure 114) but the most important variables in the code are: basedata [4][4] will be the IV; M[4][4] will be the matrix to make the MixColumn process; and key_original[4][4] (shared key) will be the seed to generate the expanded key. In addition, there are multiples variables and methods declared to make more dynamic the code and can be used in any part of the code.

The *event void Boot.booted()* (Figure 77) is also defined in the code. When this even is run, the extended key is generated. This happens at the beginning of the application, keeping data in an array with the required sizes.

```
event void Timer0.fired() {
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
  if (len == sizeof(DecryptRadioOFBMsg)) {
    DecryptRadioOFBMsg* btrpkt = (DecryptRadioOFBMsg*)payload;
    setLeds(0x04);
    d3 = btrpkt->data;
    printf("Dato recibido sin descifrar: %#010x\n",d3);
    printfflush();

    //XOR CIFRADO INICIAL
    initVar();
    col_key = 0;

      do
      {
          do
          {
               result_f[j][i] = basedata[j][i] ^ key[j][col_key];
               j++;
          } while (j < rounds_col);
          j = 0; i++; col_key++;
      } while(i < rounds);

      //ITERACIONES n = 10 [0-9]
      b0 = 0;
      for(b0;b0 < n;b0++)
      {
          //PROCESO DE SUSTITUCION    //SALE CON result1
          initVar();
          sustitucionMatrix(result_f);

          //PROCESO DE ROTACIÓN       //SALE CON result2
          initVar();
          rotacionMatrix(result1);

          //PROCESO MIXCOLUMS         //SALE CON result1
          initVar();
          if ( b0 < n-1)
          {
              for (i;i<rounds;i++)
              {
                  for (j;j<rounds;j++)
                  {
                      for (k;k<rounds;k++)
                      {
                          selectColRow(M[i][k]);
                          it3 = L_table[fh][lh];

                          selectColRow(result2[k][j]);
                          it5 = L_table[fh][lh];
                          if(fh == 0x00 && lh == 0x00)
                          {
                              detect = TRUE;
                          }

                          it6 = it5 + it3;
                          if(it6-it5 != it3)
                          {
                              it6++;
                          }

                          selectColRow(it6);
                          if (detect == TRUE)
                          {
                              it4 ^= 0;
                              detect = FALSE;
                          } else {
                              it4 ^= E_table[fh][lh];
                          }
                      }
                      result1[i][j] = it4;
                      k = 0; it3 = 0; it4 = 0;
```

```
                    }
                    j = 0;
                }
                //COPIA MATRIX
                initVar();
                for(i;i < rounds;i++)
                {
                    for(j;j < rounds_col;j++)
                    {
                        result2[i][j] = result1[i][j];
                    }
                    j = 0;
                }
            }

            //XOR CIFRADO FINAL
            initVar();
            do
            {
                do
                {
                      result_f[j][i] = result2[j][i] ^ key[j][col_key];
                      j++;
                } while (j < rounds_col);
                j = 0; col_key++; i++;
            } while(i < rounds);
        }
```

Figure 115: Specific task when a mote receives a message [DecryptRadioOFB]

The program continues receiving messages via Radio interface. The received data is an 8-bit integer type and is kept in the d3 variable. EncryptRadioOFB and DecryptRadioOFB share two variables: the IV defined as basedata[4][4] and the shared key defined as key_original[4][4]. From key_original, the extended key called key was generated previously. DecryptRadioOFB starts with basedata[i][j] and key[i][j] making the initial XOR cipher. After that, a loop starts with ten iterations where several procedures are carried out. The first one is the substitution process, where data is separated in 4-bit leftmost and 4-bits rightmost and is substituted as indicated in S-box. After that, the rotation process occurs, where the first row is not rotated so, it keeps the original state. For the second row, 1-byte circular left shift is performed. For the third row, 2-byte circular left shift is performed. For the fourth row, a 3- byte circular left shift is performed. Third process is the MixColumns that is the most complex process. In spite of being a decryption application, the M[4][4] variable is used in this code because DecryptRadioOFB share some code with EncryptRadioOFB. MixColumns is executed one time less than other processes because this process is not executed in the last round. The MixColumn result is copied into another variable to start a new iteration and when the loop is finished, the final operation is an XOR between the MixColumn result and the expanded key (Figure 115).

```
initVar();
        for (i;i < sizeof(basedata); i++)
        {
             v_fila[i] = basedata[j][k];
             j++;
             if (j>3)
             {
                 j = 0; k++;
             }
        }

        it3 = result_f[0][0] ^ d3;
        printf("Dato descifrado: %#010x\n",it3);
        printfflush();

        i = 0;
        for (i;i < sizeof(basedata)-1; i++)
```

```
        {
              v_fila[i] = v_fila[i+1];
        }
        v_fila[sizeof(basedata)-1] = result_f[0][0];

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
            basedata[j][k] = v_fila[i];
            j++;
            if (j>3)
            {
                j = 0; k++;
            }
        }
    }
    return msg;
  }
}
```

Figure 116: Specific characteristics of application [DecryptRadioOFB]

It is time to define the specific characteristics of this application (Figure 116). Once the cipherdata is received, the program prints the cipherdata on the screen. The first operation is to set the basedata[4][4] in a array[16] using a loop. The program continues XORing result_f[0][0] with cipherdata and the result is the plaindata. When plaindata is decrypted it is printed on the screen. Then, the vector is reordered, where the 8 leftmost bits are discarded and the remaining data are shifted one (byte) left position in the vector. Following the previous steps, the last vector position is empty so the result obtained of Encrypt process (result_f[0][0]) is set in this position. Again, data is ordered as a matrix, basedata[4][4], and variables are resfresehd to be used in the next round to decrypt the next cipherdata when it is received.

```
COMPONENT=DecryptRadioOFBAppC
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS
CFLAGS += -DPRINTF_BUFFER_SIZE=800
include $(MAKERULES)
```

Figure 117: DecryptRadioOFB Makefile

There are some differences compared to other Makefile files (Figure 117). There are two basic lines where the programmer defines which the component is and an include Makerules to add different rules to the compiler. In addition, there are three CFLAGS, which indicate to the compiler where printf.h file is, a sentence to recognize the printf sentences, and the printf buffer size is initialized to 800.

Chapter 4 :Results

In this chapter, the objective is to show and compare the obtained results so we get a better knowledge of the characteristics of the program running on the motes. In addition, is important to perform measurements about energy consumption for different simulation times and take the compilation and execution time. There are ten different programs that led to the reader know how the motes works under different circumstances. The applications that have been tested are Blink, BlinkSemaforo, BlinkToRadioInv, SenseRadio, ECB, CFB and OFB. All programs but Blink have been developed in this final degree project.

4.1. Blink

This is the only one application contained in TinyOS. Blink is the simplest code, where a counter is increasing every time the Timer is over. Leds will represent the three rightmost bits of the counter.

```
    compiling BlinkAppC to a micaz binary
ncc -o build/micaz/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -ta
rget=micaz -fnesc-cfile=build/micaz/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x2
2 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkAppC\" -DIDENT_USE
RNAME=\"user\" -DIDENT_HOSTNAME=\"XubunTOS\" -DIDENT_USERHASH=0x184e7486L -DIDENT_
TIMESTAMP=0x5756fb4cL -DIDENT_UIDHASH=0x119ff571L -fnesc-dump=wiring -fnesc-dump='
interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components)' -fnes
c-dumpfile=build/micaz/wiring-check.xml BlinkAppC.nc -lm
    compiled BlinkAppC to build/micaz/main.exe
            2044 bytes in ROM
              51 bytes in RAM
```

Figure 118: Blink compilation reply

Once the code has been compiled, the compiler replies with the free necessary resources to execute the application in a mote (Figure 118). To compile the program it is necessary to execute the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. The necessary resources to run the application into a mote are 2044 bytes in ROM and 51 bytes in RAM

Time (s)	CPU Energy Consumption (J)	Yellow Led Energy Consumption (J)	Green Led Energy Consumption (J)	Red Led Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0000	0,0000	**0,0227**	**0,0227**	**0,0227**
2	0,0250	0,0000	0,0028	0,0028	**0,0306**	**0,0079**	**0,0153**
3	0,0254	0,0062	0,0062	0,0062	**0,0440**	**0,0133**	**0,0147**
4	0,0258	0,0064	0,0096	0,0096	**0,0514**	**0,0074**	**0,0128**
5	0,0262	0,0129	0,0129	0,0129	**0,0649**	**0,0135**	**0,0130**
6	0,0266	0,0131	0,0161	0,0161	**0,0719**	**0,0070**	**0,0120**
7	0,0270	0,0193	0,0193	0,0193	**0,0850**	**0,0131**	**0,0121**
8	0,0274	0,0199	0,0226	0,0226	**0,0923**	**0,0074**	**0,0115**
9	0,0278	0,0258	0,0258	0,0258	**0,1051**	**0,0128**	**0,0117**
10	0,0282	0,0266	0,0290	0,0290	**0,1128**	**0,0077**	**0,0113**

Table 13: Blink Energy Consumption

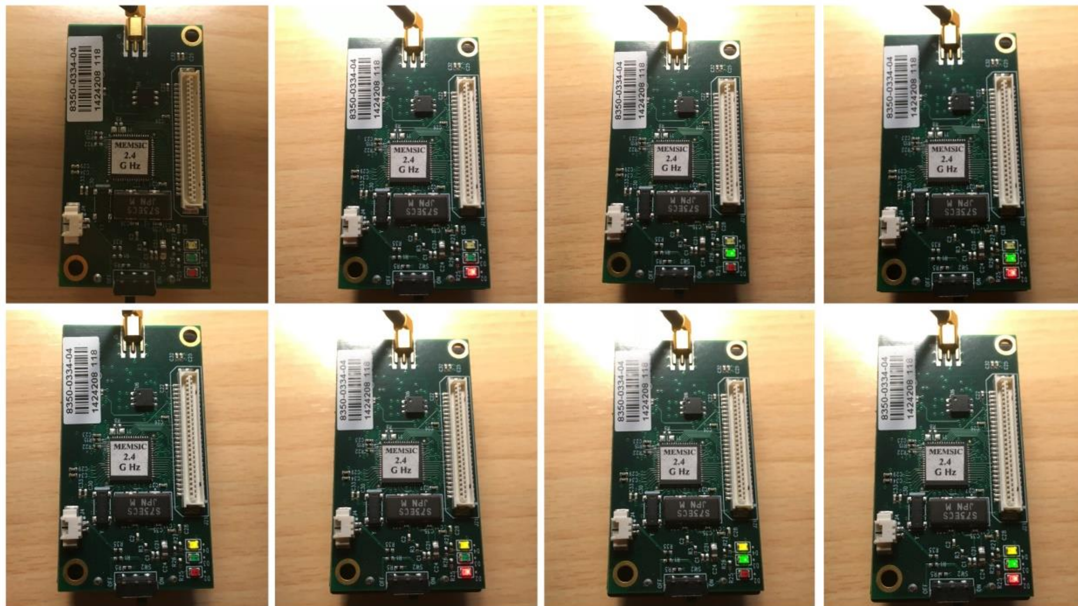

Figure 119: MICAz Leds state [Blink]

Counter value start in 0x0000 and it is increasing every time the Timer is over. The mote only has three Leds, so, it can represent the three rightmost bits of the counter as is represented in Figure 119.

There are different formats to represente data. The first one is to use a table as the previous one (Table 13). Another method is to represent the obtained values in a graph, which can be easier to understand and interpretate. The program has been run for 10 seconds. Along the time, total accumulated energy consumption measured in Joules is increasing.. On this case, Avrora detect energy consumtion in Leds (yellow, green and red) and CPU, which controls the application and order commands to different components to make necessary operations. Having the follow mathematical relationship it is possible to obtain the energy consumtion in Watts:

$$1W = 1^{J}/_{s}$$

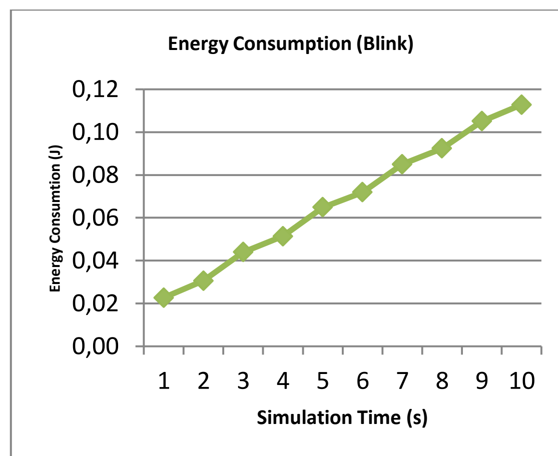

Figure 120: Total accumulated energy consumption [Blink]

Figure 120, shows the total accumulated energy consumption measured in Joules, as time goes on, the total accumulated energy consumption increases. The program has been run for 10 seconds. The result is a positive linear tendency of the energy consumption along the time.

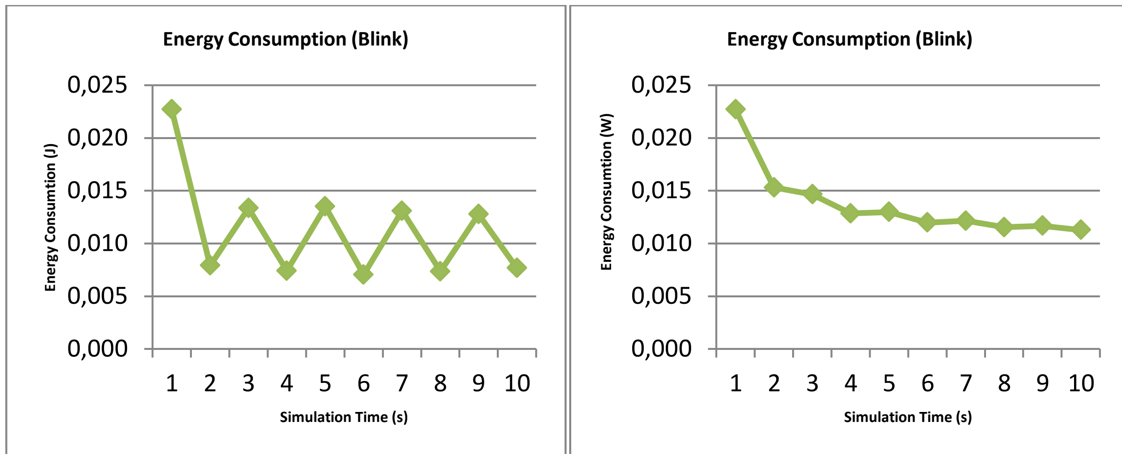

Figure 121: Total energy consumption J [Blink] *Figure 122: Total energy consumption W [Blink]*

It is very important know the total energy consumption where acts CPU and Leds (yellow, green and red). Figure 121 shows the total energy consumption expressed in Joules. In the first second, the mote has the higher consumption because has to initialize all the process. Once it has been done, the application is the simplest one, increasing the counter and displaying the 3-bits leftmost in Leds. After the first second, the consumption drops until approximately 0.01 J making a “saw-tooth”. Figure 122 shows the total energy consumption expressed in Watts. As Figure 121, the first second is when the mote has the highest consumption but the consumption is decreasing with the time following a negative exponential tendency.

Given that motes have limited resources, it is also interesting to know how long it takes to compile the code and to include the binary file in the mote. Blink application spends 0.064s to compile the application and 0.008s to insert application into the mote.

4.2. BlinkSemaforo

BlinkSemaforo is a very simple program to understand how to program in nesC. The source code is Blink and it has been modified in this final master project to keep Led1 always in the on mode. Led0 and Led2 will blink every time the Timer is over, having always different states (Led0 on | Led2 off, and vice versa).

This application is very simple, as Blink application. Once the code has been compiled, the compiler replies with the free necessary resources to execute the application in a mote (Figure 123). To compile the program is necessary to execute the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process starts. The necessary resources to run the application into a mote are 2212 bytes of ROM and 43 bytes of RAM.

```
    compiling BlinkSemaforoAppC to a micaz binary
ncc -o build/micaz/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -ta
rget=micaz -fnesc-cfile=build/micaz/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x2
2 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkSemaforoAp\" -DIDE
NT_USERNAME=\"user\" -DIDENT_HOSTNAME=\"XubunTOS\" -DIDENT_USERHASH=0x184e7486L -D
IDENT_TIMESTAMP=0x5756f4f5L -DIDENT_UIDHASH=0xf7d2772cL -fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components)'
 -fnesc-dumpfile=build/micaz/wiring-check.xml BlinkSemaforoAppC.nc -lm
    compiled BlinkSemaforoAppC to build/micaz/main.exe
            2212 bytes in ROM
              43 bytes in RAM
```

Figure 123: BlinkSemaforo compilation reply

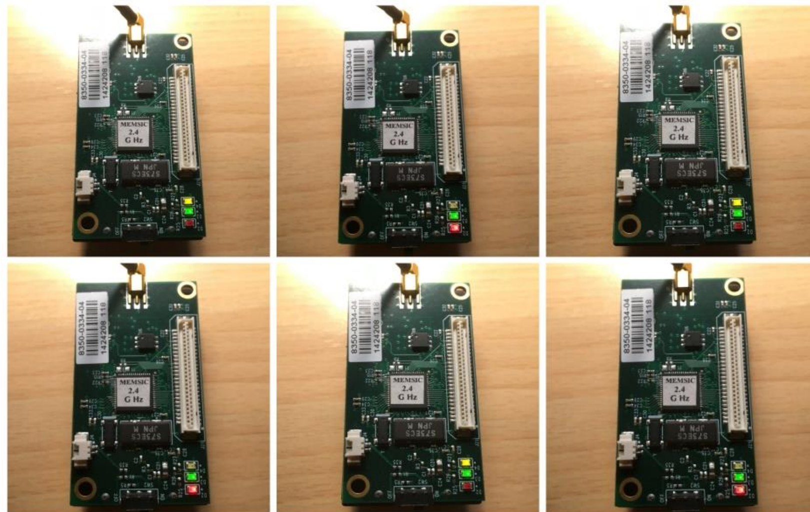

Figure 124: MICAz Leds state [BlinkSemaforo]

The previous picture (Figure 124) shows how Leds works. Led1 or green Led is always on. When Timer is over, Led1 is on all the time. Led0 and Led2 are toggling every time the Timer is over having Led0 and Led2 different states.

Time (s)	CPU Energy Consumption (J)	Yellow Led Energy Consumption (J)	Green Led Energy Consumption (J)	Red Led Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0000	0,0000	**0,0227**	**0,0227**	**0,0227**
2	0,0250	0,0028	0,0060	0,0000	**0,0338**	**0,0111**	**0,0169**
3	0,0254	0,0062	0,0126	0,0032	**0,0474**	**0,0136**	**0,0158**
4	0,0258	0,0096	0,0192	0,0064	**0,0610**	**0,0136**	**0,0153**
5	0,0262	0,0129	0,0258	0,0097	**0,0746**	**0,0136**	**0,0149**
6	0,0265	0,0161	0,0324	0,0131	**0,0882**	**0,0136**	**0,0147**
7	0,0269	0,0193	0,0390	0,0165	**0,1018**	**0,0136**	**0,0145**
8	0,0273	0,0226	0,0456	0,0199	**0,1153**	**0,0136**	**0,0144**
9	0,0277	0,0258	0,0522	0,0232	**0,1289**	**0,0136**	**0,0143**
10	0,0281	0,0290	0,0588	0,0266	**0,1425**	**0,0136**	**0,0143**

Table 14: BlinkSemaforo Energy Consumption

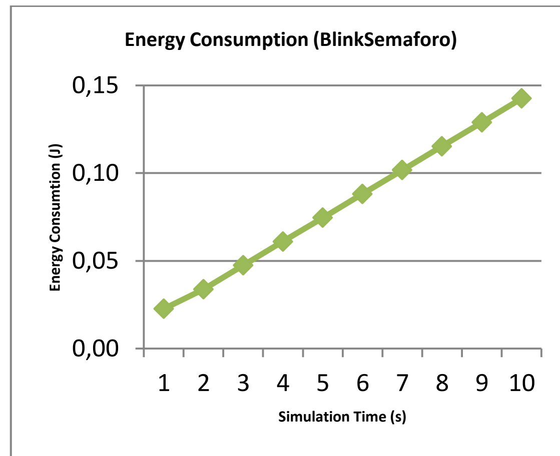

Figure 125:Total accumulated energy consumption (J) [BlinkSemaforo]

Table 14 and Figure 125 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 10 seconds. In this case, Avrora detect energy consumtion in Leds (yellow, green and red); and CPU, which controls the applincation and order commands to different components to make necessary operations. As time goes on, the total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

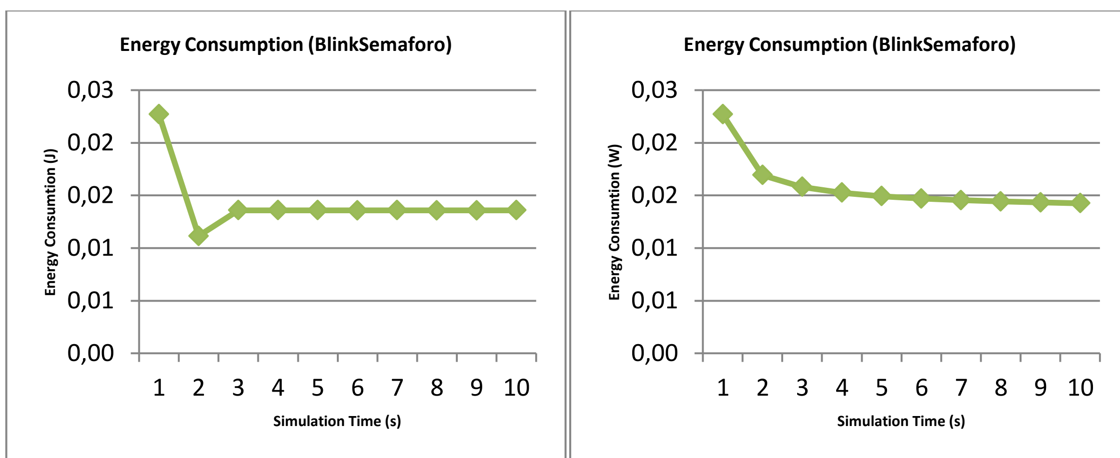

Figure 126: Total energy consumption J [BlinkSemaforo]

Figure 127: Total energy consumption W [BlinkSemaforo]

It is very important know the total energy consumption where acts CPU and Leds (yellow, green and red). Figure 126 shows the total energy consumption expressed in Joules. In the first second, the mote has the higher consumption because has to initialize all the process. After the first second, the consumption drops until the minimum value in second two when all process are initialized to 0.11 J and then, the total energy consumption is kept in approximately 0.0145 J. Figure 127 shows the total energy consumption expressed in Watts. As Figure 126, the first second is when the mote has the highest consumption because has to initialize all process but then, the consumption is decreasing with the time following a negative exponential tendency.

Regarding compilation and inclusion of the binary files in the mote, the BlinkSemaforo application spends 0.048s compiling the application and 0.012s inserting the application into the mote.

4.3. BlinkToRadioInv

The objective of this application is to modify the original BlinkToRadio application and create an inverse counter. The initial value is initialized as 0xFFFFFFFF in a 32-bits variable and each time the Timer is over, it subtracts one integer and refreshes the variable. Led representation could be similar to Blink but with an inverse count. In spite of creating a new inverse counter, it is important to check how variables are defined to work in the application. Inside the BlinkToRadioInvMsg, there are three 16-bits unsigned integer variables and one 32-bit unsigned integer variable. 16-bits variables are: *nodeid*, which identifies the node that sends the message; *counter*, that is the original counter; and *guyid*, that identifies the person who sends the data. In this case, this *guyid* is a constant defined previously in the enum section. The 32-bits variable is initialized with the highest possible value and then, decreases every time the Timer is over (every 250ms) (Figure 128).

```
// $Id: BlinkToRadioInv.h,v 1.4 2006-12-12 18:22:52 vlahan Exp $

#ifndef BLINKTORADIOINV_H
#define BLINKTORADIOINV_H

enum {
  AM_BLINKTORADIOINV = 6,
  TIMER_PERIOD_MILLI = 250,
  CONSTANT_GUYID = 0x3333,
  START_COUNT_INV = 0xFFFFFFFF
};

typedef nx_struct BlinkToRadioInvMsg {
  nx_uint16_t nodeid;
  nx_uint16_t guyid;
  nx_uint16_t counter;
  nx_uint32_t inv_counter;
} BlinkToRadioInvMsg;
#endif
```

Figure 128: BlinkToRadio.h file

BlinkToRadioInv is an application, which uses the Radio interface to communicate with other motes close to it. Radio interface is used to send a message with the info detailed in BlinkToRadioInvMsg. One of these variables is *inv_counter*, which includes the inverse counter variable and is sent to the remote mote with the goal of seting Leds as a function of this value.

```
    compiling BlinkToRadioInvAppC to a micaz binary
ncc -o build/micaz/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -ta
rget=micaz -fnesc-cfile=build/micaz/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x2
2 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkToRadioInv\" -DIDE
NT_USERNAME=\"user\" -DIDENT_HOSTNAME=\"XubunTOS\" -DIDENT_USERHASH=0x184e7486L -D
IDENT_TIMESTAMP=0x5756f494L -DIDENT_UIDHASH=0x8ec8ea65L -fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components)'
 -fnesc-dumpfile=build/micaz/wiring-check.xml BlinkToRadioInvAppC.nc -lm
/home/user/Downloads/tinyos/tos/chips/cc2420/lpl/DummyLplC.nc:39:2: warning: #warn
ing "*** LOW POWER COMMUNICATIONS DISABLED ***"
    compiled BlinkToRadioInvAppC to build/micaz/main.exe
           12766 bytes in ROM
             325 bytes in RAM
```

Figure 129:BlinkToRadioInv compilation reply

Once the code has been compiled, the compiler shows in the screen the resources neeeded to execute this program into a mote. To compile the program is necessary to execute the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. In this case, it is a simple application in TinyOS and no many resources are needed to run this program into a mote, exactly 12766 bytes of ROM and 325 bytes of RAM (Figure 129).

```
serial@/dev/ttyUSB1:57600: resynchronising
00 FF FF 00 02 0A 22 06 00 02 00 00 33 33 FF FF FF FF
00 FF FF 00 02 0A 22 06 00 02 00 01 33 33 FF FF FF FE
00 FF FF 00 02 0A 22 06 00 02 00 02 33 33 FF FF FF FD
00 FF FF 00 02 0A 22 06 00 02 00 03 33 33 FF FF FF FC
00 FF FF 00 02 0A 22 06 00 02 00 04 33 33 FF FF FF FB
00 FF FF 00 02 0A 22 06 00 02 00 05 33 33 FF FF FF FA
00 FF FF 00 02 0A 22 06 00 02 00 06 33 33 FF FF FF F9
00 FF FF 00 02 0A 22 06 00 02 00 07 33 33 FF FF FF F8
00 FF FF 00 02 0A 22 06 00 02 00 08 33 33 FF FF FF F7
00 FF FF 00 02 0A 22 06 00 02 00 09 33 33 FF FF FF F6
00 FF FF 00 02 0A 22 06 00 02 00 0A 33 33 FF FF FF F5
00 FF FF 00 02 0A 22 06 00 02 00 0B 33 33 FF FF FF F4
00 FF FF 00 02 0A 22 06 00 02 00 0C 33 33 FF FF FF F3
00 FF FF 00 02 0A 22 06 00 02 00 0D 33 33 FF FF FF F2
00 FF FF 00 02 0A 22 06 00 02 00 0E 33 33 FF FF FF F1
00 FF FF 00 02 0A 22 06 00 02 00 0F 33 33 FF FF FF F0
00 FF FF 00 02 0A 22 06 00 02 00 10 33 33 FF FF FF EF
00 FF FF 00 02 0A 22 06 00 02 00 11 33 33 FF FF FF EE
00 FF FF 00 02 0A 22 06 00 02 00 12 33 33 FF FF FF ED
00 FF FF 00 02 0A 22 06 00 02 00 13 33 33 FF FF FF EC
00 FF FF 00 02 0A 22 06 00 02 00 14 33 33 FF FF FF EB
00 FF FF 00 02 0A 22 06 00 02 00 15 33 33 FF FF FF EA
00 FF FF 00 02 0A 22 06 00 02 00 16 33 33 FF FF FF E9
00 FF FF 00 02 0A 22 06 00 02 00 17 33 33 FF FF FF E8
00 FF FF 00 02 0A 22 06 00 02 00 18 33 33 FF FF FF E7
00 FF FF 00 02 0A 22 06 00 02 00 19 33 33 FF FF FF E6
00 FF FF 00 02 0A 22 06 00 02 00 1A 33 33 FF FF FF E5
00 FF FF 00 02 0A 22 06 00 02 00 1B 33 33 FF FF FF E4
00 FF FF 00 02 0A 22 06 00 02 00 1C 33 33 FF FF FF E3
00 FF FF 00 02 0A 22 06 00 02 00 1D 33 33 FF FF FF E2
00 FF FF 00 02 0A 22 06 00 02 00 1E 33 33 FF FF FF E1
00 FF FF 00 02 0A 22 06 00 02 00 1F 33 33 FF FF FF E0
00 FF FF 00 02 0A 22 06 00 02 00 20 33 33 FF FF FF DF
00 FF FF 00 02 0A 22 06 00 02 00 21 33 33 FF FF FF DE
00 FF FF 00 02 0A 22 06 00 02 00 22 33 33 FF FF FF DD
```

Figure 130: 802.15.4 frame [BlinkToRadioInv]

With *java net.tinyos.tools.Listen –comm serial@/dev/ttyUSB1:micaz* command, sent messages from a mote via Radio interface can be read using the Basestation application in a mote connected to the programming board mib520 (Figure 130). Different fields are used in the frame.

- First 0x00 byte data is the preamble data (1 byte).
- 0xFFFF data is the destination address (2 bytes).
- 0x0002 data is the link source address (2 bytes).
- 0x0A data is message length (1 byte).
- 0x22 data is group ID (1 byte).
- 0x06 data is Active Message handler type (1 byte).
- From 0x00 until last byte, the payload message:
  - 0x0002 data is the source mote ID.
  - 0x00XX data is the original counter which start in 0x0000 and the last data is 0x0022.
  - 0x3333 data is guyid value defined as a constant.
  - 0xFFFFFFXX is the new inverse counter, which start in 0XFFFFFFFF and finish as 0XFFFFFFDD.

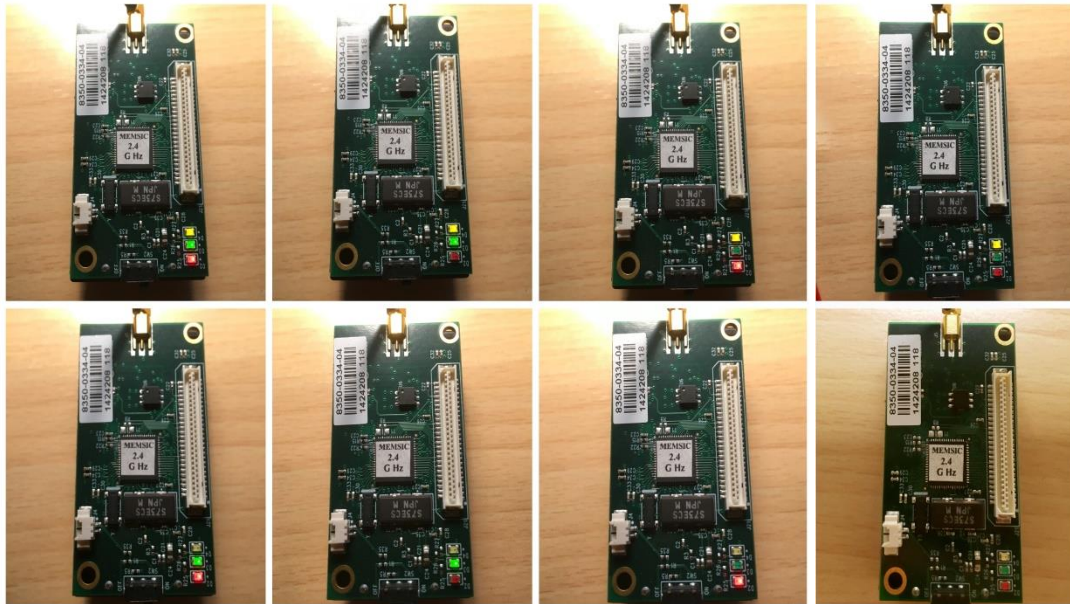

Figure 131: MICAz Leds state [BlinkToRadioInv]

In BlinkToRadioInv application, the mote has an inverse counter, which decreases the internal variable and only shows the three rightmost bits. The internal variable is initialized with 0xFFFFFFFF value and this value is decreased one unit when Timer is over. At the previous picture (Figure 131), when the program starts, Leds are initialized with '111' and can take other values as '110','101', etc. Because the mote only shows the three rightmost bits, when the Led show '000' value the next value is again '111' having a cyclical process.

Time (s)	CPU Energy Consumption (J)	Yellow Led Energy Consumption (J)	Green Led Energy Consumption (J)	Red Led Energy Consumption (J)	Radio Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0000	0,0000	0,0021	**0,0248**	**0,0227**	**0,0248**
2	0,0339	0,0044	0,0016	0,0016	0,0514	**0,0929**	**0,0079**	**0,0465**
3	0,0440	0,0048	0,0048	0,0048	0,1078	**0,1663**	**0,0133**	**0,0554**
4	0,0541	0,0112	0,0081	0,0086	0,1642	**0,2460**	**0,0074**	**0,0615**
5	0,0642	0,0113	0,0113	0,0113	0,2205	**0,3187**	**0,0135**	**0,0637**
6	0,0743	0,0177	0,0147	0,0147	0,2770	**0,3984**	**0,0070**	**0,0664**
7	0,0844	0,0181	0,0181	0,0181	0,3333	**0,4719**	**0,0131**	**0,0674**
8	0,0945	0,0242	0,0215	0,0215	0,3897	**0,5512**	**0,0074**	**0,0664**
9	0,1045	0,0248	0,0248	0,0248	0,4460	**0,6250**	**0,0128**	**0,0694**
10	0,1146	0,0306	0,0282	0,0282	0,5024	**0,7041**	**0,0077**	**0,0704**

Table 15: BlinkToRadioInv Energy Consumption

On this case, Avrora detects energy consumtion in Leds (yellow, green and red) that represent the three rightmost bits of the counter; CPU that controls the applincation and order commands to different components to make necessary operations; and Radio interface that sends messages to other motes or receive messages from other motes. The total accumulated energy consumtion is caculated adding the consumption of each active part in the proccess detailed previously.

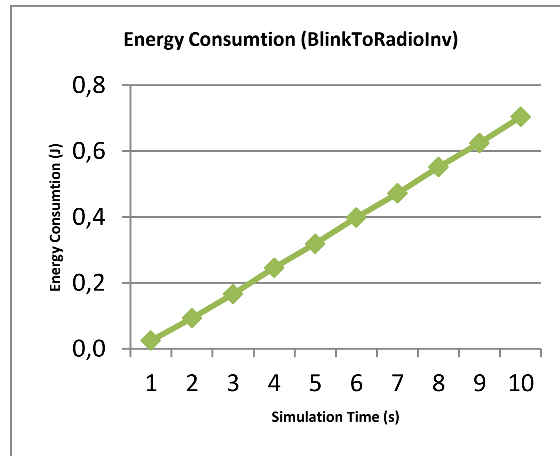

Figure 132: Total accumulated energy consumption (J) [BlinkToRadioInv]

The Figure 132 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 10 seconds. As time goes on, the total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

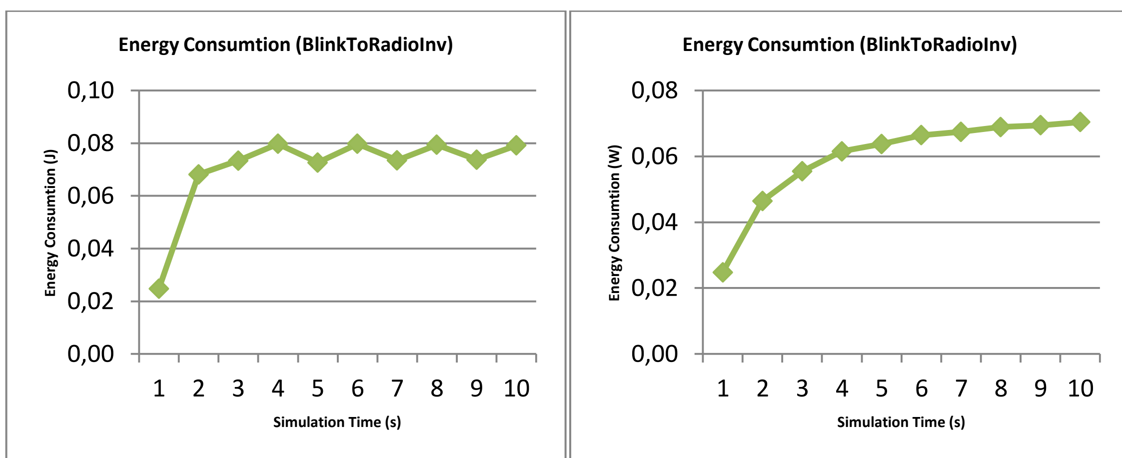

Figure 133: Total energy consumption J [BlinkToRadioInv]

Figure 134: Total energy consumption W [BlinkToRadioInv]

It is very important know the total energy consumption where acts CPU, Leds (yellow, green and red) and Radio interface. Figure 133 shows the total energy consumption expressed in Joules. Contrary to the previous application, the first second is when the mote has the lowest energy consumption. From that moment, mote actives Radio interface to send data to other motes and the total energy consumption is increased. From third second, the total energy consumption is stablish in 0.075 J making “saw-tooth” Figure 134 shows the total energy consumption expressed in Watts. As Figure 133, the first second is when the mote has the lowest consumption and since second two, the total energy consumption is increasing because Radio interface is on to send and receive messages. Both Figures follow a positive logarithm tendency.

Regarding compilation and inclusion of the binary files in the mote, BlinkToRadioInv spends 0.096s to compile the application and 0.044s to insert application into the mote.

4.4. SenseRadio

One simple application is Sense, which takes data, forms Sensorboard, and displays that data using Leds in the mote. The goal now is to know how to implement the Radio interface using an application, which by default, has not implemented the necessary components and interfaces to make it possible. The application is developed to work, at least, with two motes. The first mote has the SenseRadio application, where the mote takes data from the SensorBoard and sends data via Radio interface. A mote set in the programming board (mib520) can read these sent messages and with the Basestation application shows the messages in the PC screen using the Listen java command.

```
    compiling SenseRadioAppC to a micaz binary
ncc -o build/micaz/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -ta
rget=micaz -fnesc-cfile=build/micaz/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x2
2 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"SenseRadioAppC\" -DIDEN
T_USERNAME=\"user\" -DIDENT_HOSTNAME=\"XubunTOS\" -DIDENT_USERHASH=0x184e7486L -DI
DENT_TIMESTAMP=0x5756f155L -DIDENT_UIDHASH=0xbaa76ae3L -fnesc-dump=wiring -fnesc-d
ump='interfaces(!abstract())' -fnesc-dump='referenced(interfacedefs, components)'
-fnesc-dumpfile=build/micaz/wiring-check.xml SenseRadioAppC.nc -lm
In component `SenseRadioAppC':
SenseRadioC.nc:9: warning: expected component `SenseRadioC', but got component 'Se
nseC'
/home/user/Downloads/tinyos/tos/chips/cc2420/lpl/DummyLplC.nc:39:2: warning: #warn
ing "*** LOW POWER COMMUNICATIONS DISABLED ***"
SenseRadioC.nc: In function 'SenseRadioC__Receive__receive':
SenseRadioC.nc:93: warning: unused variable 'btrpkt'
    compiled SenseRadioAppC to build/micaz/main.exe
           13428 bytes in ROM
             333 bytes in RAM
```

Figure 135: SenseRadio compilation reply

Once the code has been compiled, the compiler shows in the screen the necessary resources to execute this program into a mote. To compile the program is necessary to execute the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. On this case, is a more complex application than Sense but even so, the needed resources to run the application into a mote are 13428 bytes in ROM and 333 bytes in RAM (Figure 135).

```
serial@/dev/ttyUSB1:57600: resynchronising
00 FF FF 00 02 06 22 06 00 02 00 01 01 C4
00 FF FF 00 02 06 22 06 00 02 00 02 01 C4
00 FF FF 00 02 06 22 06 00 02 00 03 01 C4
00 FF FF 00 02 06 22 06 00 02 00 04 01 C4
00 FF FF 00 02 06 22 06 00 02 00 05 01 C4
00 FF FF 00 02 06 22 06 00 02 00 06 01 C4
00 FF FF 00 02 06 22 06 00 02 00 07 01 C4
00 FF FF 00 02 06 22 06 00 02 00 08 01 C4
00 FF FF 00 02 06 22 06 00 02 00 09 01 C4
00 FF FF 00 02 06 22 06 00 02 00 0A 01 C5
00 FF FF 00 02 06 22 06 00 02 00 0B 01 C5
00 FF FF 00 02 06 22 06 00 02 00 0C 01 C5
00 FF FF 00 02 06 22 06 00 02 00 0D 01 C5
00 FF FF 00 02 06 22 06 00 02 00 0E 01 C5
00 FF FF 00 02 06 22 06 00 02 00 0F 01 C5
00 FF FF 00 02 06 22 06 00 02 00 10 01 C5
```

Figure 136: 802.15.4 frame [SenseRadio]

The application uses Radio interface. With *java net.tinyos.tools.Listen –comm serial@/dev/ttyUSB1:micaz* command, sent messages via Radio interface can be read (Figure 136). Different fields are used in the frame.

- First 0x00 byte data is the preamble data (1 byte).
- 0xFFFF data is destination address (2 bytes).
- 0x0002 data is link source address (2 bytes).
- 0x06 data is message length (1 byte).
- 0x22 data is group ID (1 byte).
- 0x06 data is Active Message handler type (1 byte).
- From 0x00 until last byte, the payload message:
  - 0x0002 data is the source mote ID.
  - 0x00XX data is the original counter which start in 0x0000 and the last data is 0x0012.
  - 0x01C4 and 0x01C5 are data that have been taken using SensorBoard.

Time (s)	CPU Energy Consumtion (J)	Yellow Led Energy Consumtion (J)	Green Led Energy Consumtion (J)	Red Led Energy Consumtion (J)	Radio Energy Consumtion (J)	SensorBoard Energy Consumtion (J)	Total Energy Consumtion (J)	Total Energy Consumtion (J)	Energy (W)
1	0,0227	0,0000	0,0000	0,0000	0,0000	0,0021	**0,0248**	**0,0248**	**0,0248**
2	0,0339	0,0047	0,0047	0,0047	0,0514	0,0042	**0,1037**	**0,0789**	**0,0518**
3	0,0440	0,0113	0,0113	0,0113	0,1078	0,0063	**0,1920**	**0,0883**	**0,0640**
4	0,0541	0,0179	0,0179	0,0179	0,1641	0,0084	**0,2804**	**0,0884**	**0,0701**
5	0,0642	0,0245	0,0245	0,0245	0,2205	0,0105	**0,3688**	**0,0884**	**0,0738**
6	0,0743	0,0311	0,0311	0,0311	0,2769	0,0126	**0,4572**	**0,0884**	**0,0762**
7	0,0844	0,0377	0,0377	0,0377	0,3333	0,0147	**0,5455**	**0,0883**	**0,0779**
8	0,0946	0,0443	0,0443	0,0443	0,3897	0,0168	**0,6340**	**0,0885**	**0,0792**
9	0,1047	0,0509	0,0509	0,0509	0,4460	0,0189	**0,7223**	**0,0884**	**0,0803**
10	0,1146	0,0575	0,0575	0,0575	0,5024	0,0210	**0,8106**	**0,0883**	**0,0811**

Table 16: SenseRadio Energy Consumption

On this case, Avrora detects energy consumtion in Leds (yellow, green and red) that represent the three rightmost bits of the counter; CPU that controls the applincation and order commands to different components to make necessary operations; Radio interface that sends messages to other motes or receive messages from other motes; And Sensorboard that take data from enviroment. The total accumulated energy consumtion is caculated adding the consumption of each active part in the proccess detailed previously.

The Figure 137 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 10 seconds. As time goes on, the total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

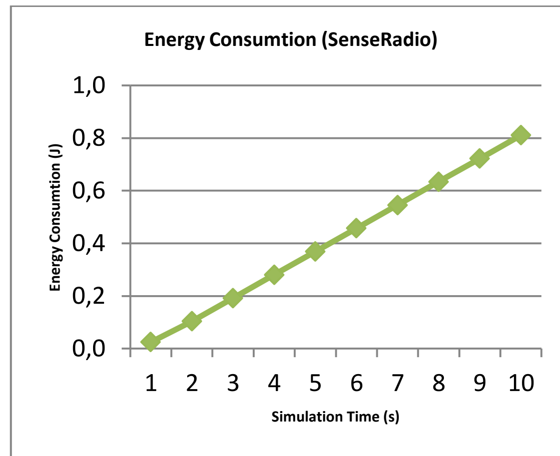

Figure 137: Total accumulated energy consumption (J) [SenseRadio]

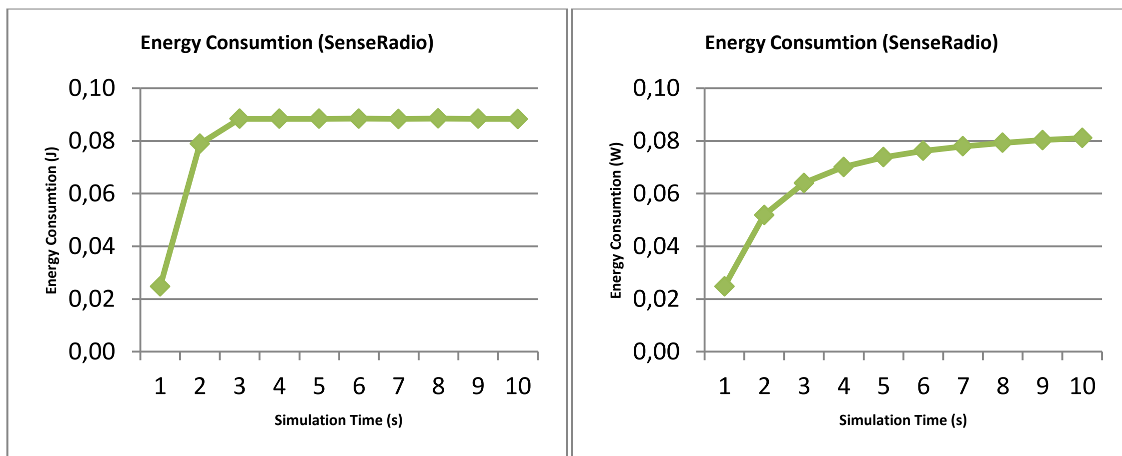

Figure 138: Total energy consumption J [SenseRadio] *Figure 139: Total energy consumption W [SenseRadio]*

It is very important know the total energy consumption where acts CPU, Leds (yellow, green and red), Radio interface and Sensorboard. Figure 138 shows the total energy consumption expressed in Joules. As BlinkToRadioInv application, the first second is when the mote has the lowest energy consumption. From that moment, mote actives Radio interface to send data to other motes and the total energy consumption is increased. From third second, the total energy consumption is stablish in 0.088 J. Figure 139 shows the total energy consumption expressed in Watts. As Figure 138, the first second is when the mote has the lowest consumption and since second two, the total energy consumption is increasing because Radio interface is on to send and receive messages. Both Figures follow a positive logarithm tendency.

Motes have limited resources and the objective is to know how long is necessary to compile the code and include the binary file in the mote. In SenseRadio application spends 0.104s to compile the application and 0.032s to insert application into the mote.

4.5. AES ECB

ECB mode (Figure 31) is the simplest programmed method using AES encryption to cipher data and to send these data via Radio interface to other mote. To do that, it is important to have at least two motes, where each one has its own program. The first mote has EncryptRadioECB application, which ciphers a plaintext and the obtain ciphertext is sent via

Radio interface. The second mote runs DecryptRadioECB, which receives the ciphertext and decrypts it to obtain the plaintext.

For simplicity, the application has a constant plaintext and a constant shared key. If the application is run more than once, the result will be the same to all iterations because the plaintext and the shared key are not refreshed.

Input type: Text

Input text: (hex) 3243F6A8885A308D313198A2E0370734

Plaintext ◉ Hex   Autodetect: ON | OFF

Function: AES

Mode: ECB (electronic codebook)

Key: (hex) AFABA7A3AEAAA6A2ADA9A5A1ACA8A4A0

Plaintext ◉ Hex

> Encrypt! > Decrypt!

Encrypted text:

00000000 | 21 0e 4c 33 89 df 35 01 dd cd e9 0a 83 27 af 92 | ! . L 3 . ß 5 . Ý Í é . . ' ¯ .

[Download as a binary file] [?]   Inactive

Figure 140: AES online calculator [ECB]

To check that the application is developed correctly, we have made use of an online AES calculator tool (Figure 140) [21]. It lets to encrypt data choosing different options. Input type is, by default, set as Text. Insert plaintext in Input text box, selecting hexadecimal data. AES calculator allows the use of different functions as AES, DES, etc. The chosen mode is ECB and it is necessary to choose a shared key selecting hexadecimal data. In addition, encryption calculator lets encrypt and decrypt once the user has added the different inputs.

```
    compiled EncryptRadioECBAppC to build/micaz/main.exe
           15054 bytes in ROM
            1444 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 141: EncryptRadioECB compilation reply

EncryptRadioECB is a very important part in the ECB application and if this program does not send any message via Radio interface, the application does not work. Plaintext and shared key are necessary resources to get the ciphertext. To compile the program is necessary to execute the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. The application has different loops defined in the code, which add extra consumption and extra necessary resources to the application. In addition, loops lets develop reiterate tasks saving programming time and program code. When the compilation process finishes, the needed resources to run the program in the mote are shown . In this case, we need 15054 bytes of ROM and 1444 bytes of RAM (Figure 141).

```
    compiled DecryptRadioECBAppC to build/micaz/main.exe
           21216 bytes in ROM
            2807 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 142: DecryptRadioECB compilation reply

DecryptRadioECB is the other half application, which is also very important because if the application is not listening continuously Radio interface, data would be sent but none will take and process the received data. The goal is to received data (ciphertext) and using the shared key to get the plaintext. To do that, there are different loops that will let to develop reiterate tasks, saving programming time and program code but, the application uses extra consumption and extra necessary resources. To start with the compilation process it is necessary to introduce the following command *make micaz install,1 mib520,/dev/ttyUSB0.* After this command, when the compiled process finishes, it shows the necessaries resources in the mote to execute the application: 21216 bytes in ROM and 2807 bytes in RAM (Figure 142).

In this case, DecryptRadioECB uses more resources than EncryptRadioECB because the process to DecryptRadioECB is a little bit different to EncryptRadioECB. This method adds different verifications that need more memory and spend more time to execute the tasks.

```
^Cuser@XubunTOS:~/Downloads/tinyos/apps/DecryptRadioECB$ java net.tinyos.tools.L
ten -comm serial@/dev/ttyUSB1:micaz
serial@/dev/ttyUSB1:57600: resynchronising
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
00 FF FF 00 01 10 22 06 21 0E 4C 33 89 DF 35 01 DD CD E9 0A 83 27 AF 92
```

Figure 143: 802.15.4 cipher frame [ECB]

The message is sent from EncryptRadioECB application via Radio interface to DecryptRadioECB application. If other mote connected to PC is listening Radio interface (Figure 143), it can read the previous message structure using *java net.tinyos.tools.Listen –comm serial@/dev/ttyUSB1:micaz* command. The frame structure is:

- First 0x00 byte data is the preamble data (1 byte).
- 0xFFFF data is destination address (2 bytes).
- 0x0001 data is link source address (2 bytes).
- 0x10 data is message length (1 byte).
- 0x22 data is group ID (1 byte).
- 0x06 data is Active Message handler type (1 byte).
- From 0x21 until 0x92, is the payload field that contains the ciphertext.

If the payload field is compared with the online AES calculator using ECB mode, we can see that both have obtained the same result.

```
user@XubunTOS:~$ java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSB1:micaz
Thread[Thread-1,5,main]serial@/dev/ttyUSB1:57600: resynchronising
Print Received Result:
0x00000021 , 0x00000089 , 0x000000dd , 0x00000083 ,
0x0000000e , 0x000000df , 0x000000cd , 0x00000027 ,
0x0000004c , 0x00000035 , 0x000000e9 , 0x000000af ,
0x00000033 , 0x00000001 , 0x0000000a , 0x00000092 ,
Print Decrypted Result:
0x00000032 , 0x00000088 , 0x00000031 , 0x000000e0 ,
0x00000043 , 0x0000005a , 0x00000031 , 0x00000037 ,
0x000000f6 , 0x00000030 , 0x00000098 , 0x00000007 ,
0x000000a8 , 0x0000008d , 0x000000a2 , 0x00000034 ,
```

Figure 144: ECB Results

Once the message has been ciphered, EncryptRadioECB sends the ciphertext and it is received by DecryptRadioECB via Radio interface. Once the mote receives the frame, it is time to decrypt the ciphertext and obtain the plaintext using the shared key. DecryptRadioECB prints the ciphertext when it receives the message via Radio interface. When the received message is printed (ciphered), it starts to decrypt it. Once the ciphertext has been decrypted, DecryptRadioECB prints the obtained plaintext. Therefore, as it is shown in Figure 144, DecryptRadioECB prints the received ciphertext and the decrypted plaintext.

Time (s)	Decrypt CPU Energy Consumption (J)	Decrypt Radio Energy Consumption (J)	Encrypt CPU Energy Consumption (J)	Encrypt Radio Energy Consumption (J)	Total Decrypt Energy Consumption (J)	Total Encrypt Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0227	0,0000	**0,0227**	**0,0227**	**0,0454**	**0,0454**	**0,0454**
2	0,0339	0,0511	0,0342	0,0499	**0,0850**	**0,0841**	**0,1691**	**0,1237**	**0,0846**
3	0,0439	0,1075	0,0442	0,1063	**0,1515**	**0,1505**	**0,3020**	**0,1329**	**0,1007**
4	0,0540	0,1639	0,0543	0,1627	**0,2179**	**0,2170**	**0,4349**	**0,1329**	**0,1087**
5	0,0640	0,2203	0,0643	0,2191	**0,2843**	**0,2834**	**0,5678**	**0,1329**	**0,1136**
6	0,0741	0,2767	0,0744	0,2755	**0,3508**	**0,3499**	**0,7006**	**0,1329**	**0,1168**
7	0,0841	0,3331	0,0844	0,3319	**0,4172**	**0,4163**	**0,8335**	**0,1329**	**0,1191**
8	0,0941	0,3895	0,0945	0,3882	**0,4836**	**0,4827**	**0,9664**	**0,1329**	**0,1208**
9	0,1042	0,4459	0,1045	0,4446	**0,5501**	**0,5492**	**1,0993**	**0,1329**	**0,1221**
10	0,1142	0,5023	0,1150	0,5009	**0,6165**	**0,6159**	**1,2324**	**0,1332**	**0,1232**
11	0,1242	0,5587	0,1246	0,5574	**0,6829**	**0,6821**	**1,3650**	**0,1326**	**0,1241**
12	0,1343	0,6151	0,1347	0,6138	**0,7494**	**0,7485**	**1,4979**	**0,1329**	**0,1248**
13	0,1443	0,6715	0,1447	0,6702	**0,8158**	**0,8150**	**1,6308**	**0,1329**	**0,1254**
14	0,1543	0,7279	0,1548	0,7266	**0,8822**	**0,8814**	**1,7636**	**0,1329**	**0,1260**
15	0,1644	0,7843	0,1648	0,7830	**0,9487**	**0,9478**	**1,8965**	**0,1329**	**0,1264**
16	0,1744	0,8407	0,1749	0,8394	**1,0151**	**1,0143**	**2,0294**	**0,1329**	**0,1268**

Table 17: ECB mode Energy Consumption

Table 17 collect data from ECB simulation, which is composed by two differents application. This two applications are complementary. It is possible obtains total accumulated energy consumption to each application (EncryptRadioECB and DecryptRadioECB). In the applications, CPU and Radio energy consumption are measured in Joules. The total accumulated energy consumption is calculated adding the simulation results for each application. Also is posible obtain total energy consumption measured in Joules and Watts using a mathematical relationship.

To simulate ECB, CFB and OFB modes are necessary 16 seconds because the application works with 128-bits and AES and data unit is 8-bits. Timer is set with 1000ms. Planing the algorithm, 128/8 = 16 iterations where each iteration will be calculated when the Timer is over.

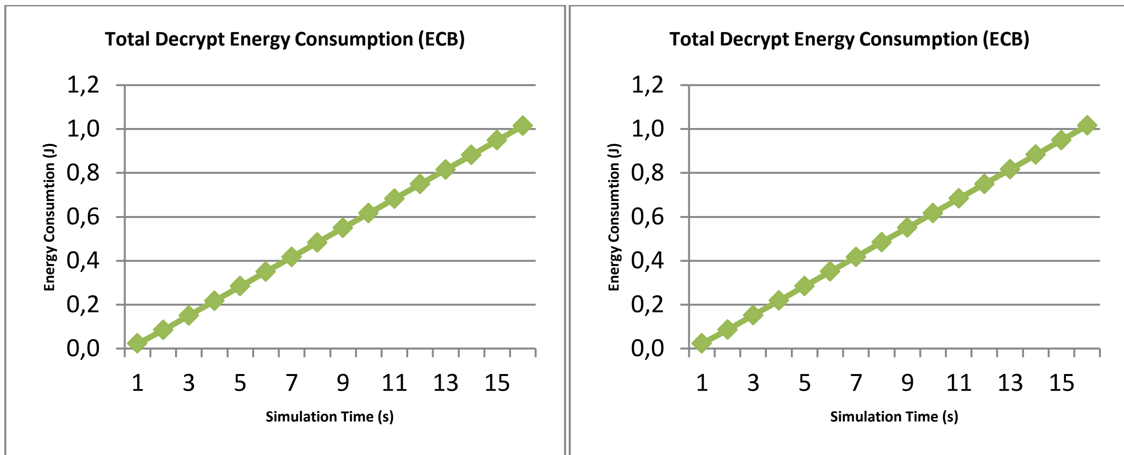

Figure 145: Total accumulated Decrypt energy consumption (J) [EncryptRadioECB]

Figure 146: Total accumulated Encrypt energy consumption (J) [DecryptRadioECB]

Figure 145 and Figure 146 are the total accumulated Encrypt energy consumption and the total accumulated Decrypt energy consumption for the ECB mode respectively. Total encrypt and total decrypt energy consumption have very similar results. Both graphs have a positive linear tendency along the time.

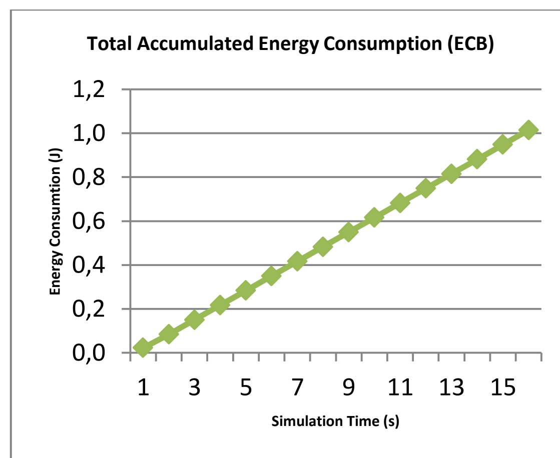

Figure 147: Total accumulated energy consumption (J) [ECB]

The Figure 147 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 16 seconds. As time goes on, the total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

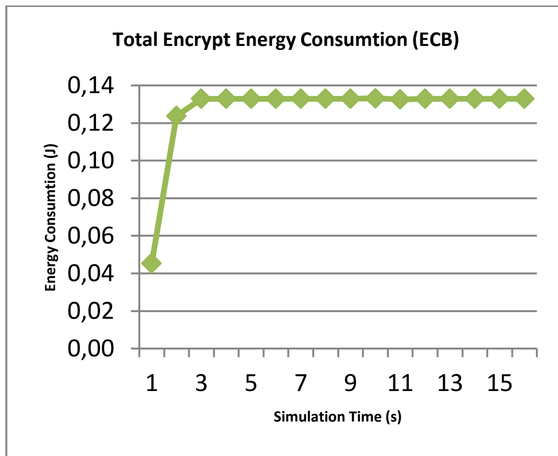

Figure 148: Total energy consumption J [ECB]

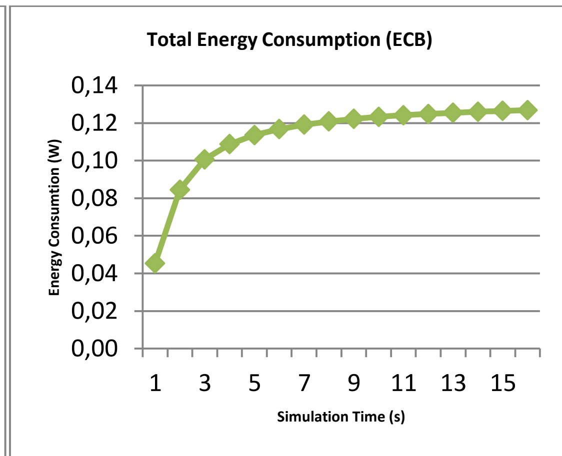

Figure 149: Total energy consumption W [ECB]

It is very important know the total energy consumption where acts CPU, and Radio interface. Figure 148 shows the total energy consumption expressed in Joules. As previous applications, the first second is when the mote has the lowest energy consumption. From that moment, mote actives Radio interface to send (in EncryptRadioECB) or receive (in EncryptRadioECB) data to/from other motes and the total energy consumption is increased. From third second, the total energy consumption is stablished in 0.133 J, significantly higher value than previous applications. It demonstrates that cipher applications are more complex to the motes and they need more resources. Figure 149 shows the total energy consumption expressed in Watts. As Figure 148, the first second is when the mote has the lowest consumption and since second two, the total energy consumption is increasing because Radio interface is on to send and receive messages. The total energy consumption trends to 0.13W. Both Figures follow a positive logarithm tendency.

Motes have limited resources and the objective is to know how long is necessary to compile the code and include the binary file in the mote. In ECB mode, there are two applications. First, EncryptRadioECB spends 0.124s to compile the application and 0.024s to insert application into the mote. The second one, DecryptRadioECB spends 0.136s to compile the application and 0.048s to insert application into the mote once the application is compiled.

4.6. AES CFB

The CFB mode (Figure 32) is a more complex method than ECB mode, but very similar to the OFB mode. At the same time, it is based on the ECB code to cipher and when the encryption process finishes, it sends cipherdata via Radio interface to other mote. CFB is composed by two applications: EncryptRadioCFB and DecryptRadioCFB, where each one has to be compiled and executed in a different mote to make possible all the process.

On one hand, EncryptRadioCFB calculates 8-bits cipherdata with IV and a shared key every time that the Timer is over (Timer period defined to 1000ms). The application refreshes the IV each iteration and the cipherdata is sent via Radio interface. On the other hand, DecryptRadioCFB is listening Radio interface and when the application receives 8-bit cipherdata it starts to decrypt it using the IV and the shared key, refreshing the IV each iteration as the operation of this mode indicates. Either application, when data is encrypted or when data is decrypted, refreshes the IV by removing the 8 leftmost bits, moving all 1-byte values to the

left in the array position and adding the calculated cipherdata (EncryptRadioCFB) or received data via Radio interface (DecryptRadioCFB) to the 8-bits rightmost.

Input type: Text
Input text: (hex) 328831E0435A3137F6309807A88DA234
Plaintext Hex Autodetect: ON | OFF
Function: AES
Mode: CFB (cipher feedback)
Key: (hex) AFABA7A3AEAAA6A2ADA9A5A1ACA8A4A0
Plaintext Hex
Init. vector: 11 55 99 DD 22 66 AA EE 33 77 BB FF 44 88 CC 00
> Encrypt! > Decrypt!

Initialization vector:
115599dd2266aaee3377bbff4488cc00 (256 bits)
Encrypted text:
00000000 43 4e bf 0b 7b 7e 93 f5 15 4b cc 1d 09 70 5c be | C N ¿ . { ~ . õ . K Ì . . p \ ¾
[Download as a binary file] [?] Inactive

Figure 150: AES online calculator [CFB]

We will use again the online AES calculator (Figure 150) to check the proper operation of the application. In this case, the chosen mode is CFB and it is necessary to set a shared key selecting hexadecimal data. Due the CFB mode implementation, IV will be set in Initialized Vector textbox. The used algorithm will process the data as was defined before. In addition, encryption calculator lets encrypt and decrypt once the user has added the different inputs.

```
    compiled EncryptRadioCFBAppC to build/micaz/main.exe
           21632 bytes in ROM
            2630 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 151: EncryptRadioCFB compilation reply

EncryptRadioCFB is a very important part in CFB application and if this program does not send any message via Radio interface, the application does not work. To compile the program is necessary to introduce the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. The application has different loops defined in the code, which add extra consumption and extra necessary resources to the application. In addition, loops lets develop reiterate tasks saving programming time and program code. The necessary resources to run the application into a mote are: 21632 bytes in ROM and 2630 bytes in RAM (Figure 151).

```
    compiled DecryptRadioCFBAppC to build/micaz/main.exe
           21234 bytes in ROM
            2596 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 152: DecryptRadioCFB compilation reply

DecryptRadioCFB is the other half application, which also is very important because if the application is not listening continuously Radio interface, data would be sent but none will take and process the received data. To start with compilation process, is necessary to introduce the following command: *make micaz install,1 mib520,/dev/ttyUSB0*. Defined in the code, there are different loops that will let to develop reiterate tasks, saving programming time but useing extra consumption and extra necessary resources. The necessary resources to run the application into a mote are: 21234 bytes of ROM and 2596 bytes of RAM (Figure 152).

In this case, EncryptRadioCFB uses more resources than DecryptRadioCFB and that is because EncryptRadioCFB runs parallel processes. Timer is always counting down and, when Timer is over, a new data is encrypted and is sent via Radio. At the same time, EncryptRadioCFB application is always reading Radio interface but the only task is to turn on Led2 if any message is detected. On other hand, DecryptRadioCFB is always reading Radio interface waiting any message, but it only works when a message has been received.

```
user@XubunTOS:~$ java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSB1:micaz
Thread[Thread-1,5,main]serial@/dev/ttyUSB1:57600: resynchronising
Cipherdata received: 0x00000043 | DecryptData received: 0x00000032
Cipherdata received: 0x0000004e | DecryptData received: 0x00000088
Cipherdata received: 0x000000bf | DecryptData received: 0x00000031
Cipherdata received: 0x0000000b | DecryptData received: 0x000000e0
Cipherdata received: 0x0000007b | DecryptData received: 0x00000043
Cipherdata received: 0x0000007e | DecryptData received: 0x0000005a
Cipherdata received: 0x00000093 | DecryptData received: 0x00000031
Cipherdata received: 0x000000f5 | DecryptData received: 0x00000037
Cipherdata received: 0x00000015 | DecryptData received: 0x000000f6
Cipherdata received: 0x0000004b | DecryptData received: 0x00000030
Cipherdata received: 0x000000cc | DecryptData received: 0x00000098
Cipherdata received: 0x0000001d | DecryptData received: 0x00000007
Cipherdata received: 0x00000009 | DecryptData received: 0x000000a8
Cipherdata received: 0x00000070 | DecryptData received: 0x0000008d
Cipherdata received: 0x0000005c | DecryptData received: 0x000000a2
Cipherdata received: 0x000000be | DecryptData received: 0x00000034
```

Figure 153: CFB Results

Once data has been ciphered and sent via Radio interface, it is received by DecryptRadioCFB and it initializes the decrypt process. DecryptRadioCFB prints cipherdata when it receives the message via Radio interface and just after that, it starts the decryption process to obtain the plain data using IV and the shared key. Once the cipherdata has been decrypted, DecryptRadioCFB prints decrypted plaintext obtained with the application. Therefore, as can be observed in Figure 153, DecryptRadioCFB prints the received ciphertext and the decrypted plaintext.

Time (s)	Decrypt CPU Energy Consumption (J)	Decrypt Radio Energy Consumption (J)	Encrypt CPU Energy Consumption (J)	Encrypt Radio Energy Consumption (J)	Total Decrypt Energy Consumption (J)	Total Encrypt Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0227	0,0000	**0,0227**	**0,0227**	**0,0454**	**0,0454**	**0,0454**
2	0,0339	0,0511	0,0339	0,0512	**0,0850**	**0,0851**	**0,1701**	**0,1247**	**0,0851**
3	0,0439	0,1075	0,0442	0,1076	**0,1515**	**0,1518**	**0,3033**	**0,1332**	**0,1011**
4	0,0540	0,1639	0,0546	0,1640	**0,2179**	**0,2186**	**0,4365**	**0,1332**	**0,1091**
5	0,0640	0,2203	0,0650	0,2204	**0,2843**	**0,2854**	**0,5697**	**0,1332**	**0,1139**
6	0,0740	0,2767	0,0756	0,2768	**0,3508**	**0,3524**	**0,7032**	**0,1335**	**0,1172**
7	0,0841	0,3331	0,0860	0,3332	**0,4172**	**0,4191**	**0,8363**	**0,1332**	**0,1195**
8	0,0941	0,3895	0,0963	0,3896	**0,4836**	**0,4859**	**0,9695**	**0,1332**	**0,1212**
9	0,1042	0,4459	0,1066	0,4460	**0,5501**	**0,5526**	**1,1027**	**0,1332**	**0,1225**
10	0,1142	0,5023	0,1170	0,5024	**0,6165**	**0,6194**	**1,2359**	**0,1332**	**0,1236**
11	0,1242	0,5587	0,1273	0,5588	**0,6830**	**0,6861**	**1,3691**	**0,1332**	**0,1245**
12	0,1343	0,6151	0,1377	0,6152	**0,7494**	**0,7529**	**1,5022**	**0,1332**	**0,1252**
13	0,1443	0,6715	0,1480	0,6716	**0,8158**	**0,8196**	**1,6354**	**0,1332**	**0,1258**
14	0,1543	0,7279	0,1584	0,7280	**0,8823**	**0,8863**	**1,7686**	**0,1332**	**0,1263**
15	0,1644	0,7843	0,1687	0,7843	**0,9487**	**0,9531**	**1,9018**	**0,1332**	**0,1268**
16	0,1744	0,8407	0,1791	0,8407	**1,0151**	**1,0198**	**2,0350**	**0,1332**	**0,1272**

Table 18: CFB mode Energy Consumption

Table 18 collect data from CFB simulation, which is composed by two differents application. This two applications are complementary. It is possible obtains total accumulated energy consumption to each application (EncryptRadioCFB and DecryptRadioCFC). In the applications, CPU and Radio energy consumption are measured in Joules. The total accumulated energy consumption is calculated adding the simulation results for each application. Also is posible obtain total energy consumption measured in Joules and Watts using a mathematical relationship.

To simulate ECB, CFB and OFB modes are necessary 16 seconds. The application works with 128-bits and AES and data unit is 8-bits. Timer is set with 1000ms. Planing the algorithm, 128/8 = 16 iterations where each iteration is calculated when the Timer is over.

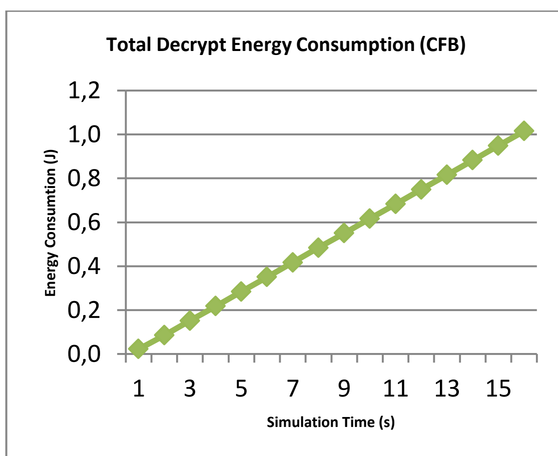

Figure 154: Total accumulated Decrypt energy consumption (J) [DecryptRadioCFB]

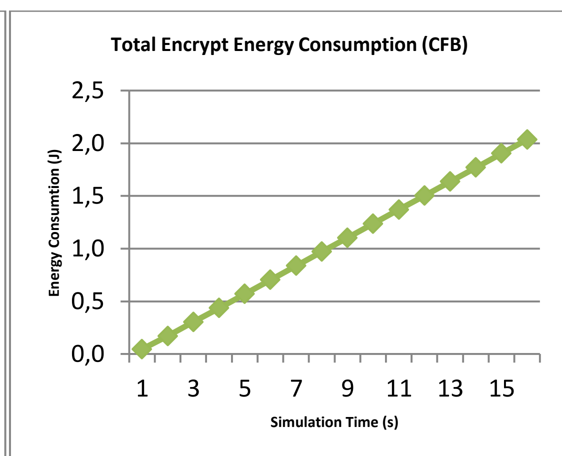

Figure 155: Total accumulated Encrypt energy consumption (J) [EncryptRadioCFB]

Figure 154 and Figure 155 are the total accumulated Encrypt energy consumption and the total accumulated Decrypt energy consumption for the CFB mode respectively. Total encrypt and total decrypt energy consumption have very similar results. Both graphs have a positive linear tendency along the time.

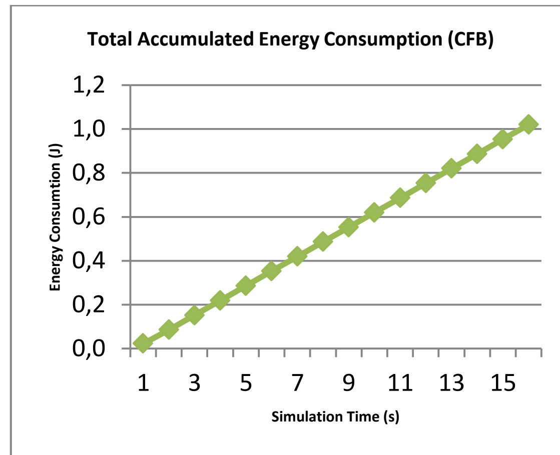

Figure 156: Total accumulated energy consumption (J) [CFB]

The Figure 156 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 16 seconds. As time goes on, total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

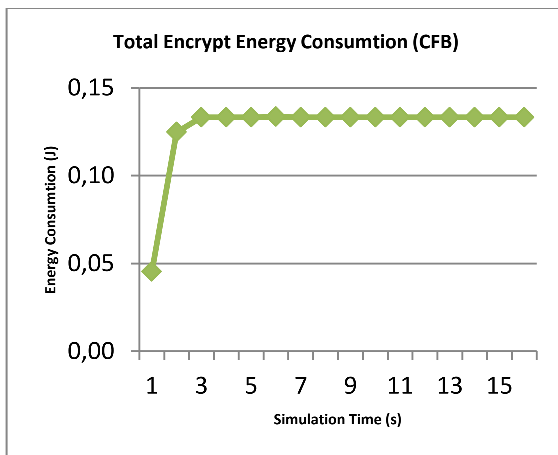

Figure 157: Total energy consumption J [CFB]

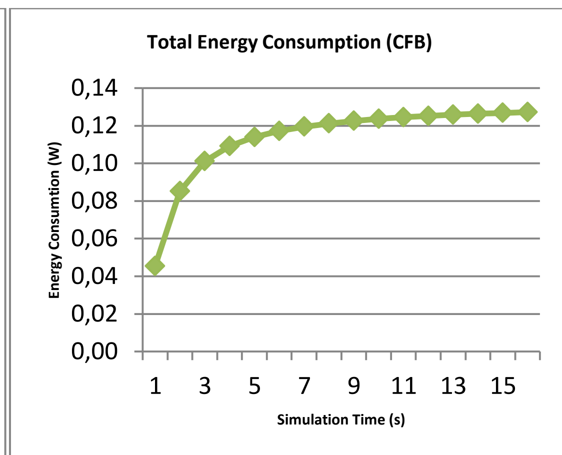

Figure 158: Total energy consumption W [CFB]

It is very important know the total energy consumption where acts CPU, and Radio interface. Figure 157 shows the total energy consumption expressed in Joules. As previous applications, the first second is when the mote has the lowest energy consumption. From that moment, mote actives Radio interface to send (in EncryptRadioCFB) or receive (in EncryptRadioCFB) data to/from other motes and the total energy consumption is increased. From third second, the total energy consumption is stablished in 0.134 J, significantly higher value than previous applications. It demonstrates that cipher applications are more complex to the motes and they need more resources. Figure 158 shows the total energy consumption expressed in Watts. As Figure 157, the first second is when the mote has the lowest

consumption and since second two, the total energy consumption is increasing because Radio interface is on to send and receive messages. The total energy consumption trends to 0.13W. Both Figures follow a positive logarithm tendency.

Motes have limited resources and the objective is to know how long is necessary to compile the code and include the binary file in the mote. In CFB mode, there are two applications. First, EncryptRadioCFB spends 0.14s to compile the application and 0.044s to insert application into the mote. The second one, DecryptRadioCFB spends 0.132s to compile the application and 0.048s to insert application into the mote once the application is compiled.

4.7. AES OFB

The OFB mode (Figure 33) is very similar to the CFB mode, where the only difference is the process to refresh the IV. OFB is based on the ECB code to encrypt and decrypt data. OFB mode is composed by two applications: EncryptRadioOFB and DecryptRadioOFB, where each one has to be compiled and executed in a different mote to make possible all the process.

On one hand we have EncryptRadioOFB, where every time that the Timer is over (Timer period defined to 1000ms) the application calculates the 8-bit cipherdata with the IV and the shared key (refreshing IV at each iteration) and this is sent via Radio interface. On the other hand, DecryptRadioOFB is listening Radio interface and when the application receives 8-bit cipherdata it starts to decrypt it using IV and shared key, refreshing also IV at each iteration. At either application, when data is being encrypted or decrypted, the IV is refreshed by removing the 8-bits leftmost, moving all data 1-byte value to the left in the array position and adding the calculated cipherdata (result_f[0][0]) obtained from encryption or decryption (EncryptRadioOFB or DecryptRadioOFB) at the 8-bits rightmost.

Input type: Text

Input text: (hex) 328831E0435A3137F6309807A88DA234

Plaintext Hex Autodetect: ON | OFF

Function: AES

Mode: OFB (output feedback, in 8bit)

Key: (hex) AFABA7A3AEAAA6A2ADA9A5A1ACA8A4A0

Plaintext Hex

Init. vector: 11 55 99 dd 22 66 aa ee 33 77 bb ff 44 88 cc 00

> Encrypt! > Decrypt!

Initialization vector:

115599dd2266aaee3377bbff4488cc00 (256 bits)

Encrypted text:

00000000 | 43 f0 68 f0 f1 5f 30 b8 69 e3 cd 77 68 ad 34 8d | C ð h ð ñ _ 0 , i ã í w h . 4

[Download as a binary file] [?] Inactive

Figure 159: AES online calculator [OFB]

To check that the application is developed correctly, we have made use of an online AES calculator tool (Figure 159) [21]

```
    compiled EncryptRadioOFBAppC to build/micaz/main.exe
           21596 bytes in ROM
            2612 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 160: EncryptRadioOFB compilation reply

EncryptRadioOFB is a very important part in OFB application and if this program does not send any message via Radio interface, the application does not work. To compile the program is necessary to introduce the next command: *make micaz install,1 mib520,/dev/ttyUSB0* and the compilation process will start. The application has different loops defined in the code, which add extra consumption and extra necessary resources to the application. In addition, loops lets develop reiterate tasks saving programming time and program code. The necessary resources to run the application into a mote are: 21596 bytes in ROM and 2612 bytes in RAM (Figure 160).

```
    compiled DecryptRadioOFBAppC to build/micaz/main.exe
           21238 bytes in ROM
            2596 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Figure 161: DecryptRadioOFB compilation reply

DecryptRadioOFB is the other half application, which also is very important because if the application is not listening continuously Radio interface, data would be sent but none will take and process the received data. To start with compilation process, is necessary to introduce the following command: *make micaz install,1 mib520,/dev/ttyUSB0*. Defined in the code, there are different loops that will let to develop reiterate tasks, saving programming time but useing extra consumption and extra necessary resources. The necessary resources to run the application into a mote are: 21238 bytes of ROM and 2596 bytes of RAM (Figure 161).

In this case, EncryptRadioOFB uses more resources than DecryptRadioOFB and that is because EncryptRadioOFB runs parallel processes. Timer is always counting down and, when Timer is over, a new data is encrypted and is sent via Radio. At the same time, EncryptRadioOFB application is always reading Radio interface but the only task is to turn on Led2 if any message is detected. On other hand, DecryptRadioOFB is always reading Radio interface waiting any message, but it only works when a message has been received.

```
 user@XubunTOS:~$ java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSB1:micaz
Thread[Thread-1,5,main]serial@/dev/ttyUSB1:57600: resynchronising
CipherData received: 0x00000043 | DecryptData received: 0x00000032
CipherData received: 0x000000f0 | DecryptData received: 0x00000088
CipherData received: 0x00000068 | DecryptData received: 0x00000031
CipherData received: 0x000000f0 | DecryptData received: 0x000000e0
CipherData received: 0x000000f1 | DecryptData received: 0x00000043
CipherData received: 0x0000005f | DecryptData received: 0x0000005a
CipherData received: 0x00000030 | DecryptData received: 0x00000031
CipherData received: 0x000000b8 | DecryptData received: 0x00000037
CipherData received: 0x00000069 | DecryptData received: 0x000000f6
CipherData received: 0x000000e3 | DecryptData received: 0x00000030
CipherData received: 0x000000cd | DecryptData received: 0x00000098
CipherData received: 0x00000077 | DecryptData received: 0x00000007
CipherData received: 0x00000068 | DecryptData received: 0x000000a8
CipherData received: 0x000000ad | DecryptData received: 0x0000008d
CipherData received: 0x00000034 | DecryptData received: 0x000000a2
CipherData received: 0x0000008d | DecryptData received: 0x00000034
```

Figure 162: OFB Results

Once data has been ciphered and sent via Radio interface, is received by DecryptRadioOFB and initializes the decrypt process. DecryptRadioOFB prints cipherdata when it receives the message via Radio interface and just after that, it starts decryption process to obtain plaindata using IV and the shared key. Once the cipherdata has been decrypted, DecryptRadioOFB prints decrypted plaindata obtained with the application. Therefore, as can be observed in Figure 162, DecryptRadioOFB prints the received ciphertext and the decrypted plaintext.

Time (s)	Decrypt CPU Energy Consumption (J)	Decrypt Radio Energy Consumption (J)	Encrypt CPU Energy Consumption (J)	Encrypt Radio Energy Consumption (J)	Total Decrypt Energy Consumption (J)	Total Encrypt Energy Consumption (J)	Total Energy Consumption (J)	Total Energy Consumption (J)	Energy (W)
1	0,0227	0,0000	0,0227	0,0000	**0,0227**	**0,0227**	**0,0454**	**0,0454**	**0,0454**
2	0,0339	0,0511	0,0339	0,0512	**0,0850**	**0,0851**	**0,1701**	**0,1247**	**0,0851**
3	0,0439	0,1075	0,0442	0,1076	**0,1515**	**0,1518**	**0,3033**	**0,1332**	**0,1011**
4	0,0540	0,1639	0,0546	0,1640	**0,2179**	**0,2186**	**0,4365**	**0,1332**	**0,1091**
5	0,0640	0,2203	0,0650	0,2204	**0,2843**	**0,2854**	**0,5697**	**0,1332**	**0,1139**
6	0,0740	0,2767	0,0756	0,2768	**0,3508**	**0,3524**	**0,7032**	**0,1335**	**0,1172**
7	0,0841	0,3331	0,0860	0,3332	**0,4172**	**0,4191**	**0,8363**	**0,1332**	**0,1195**
8	0,0941	0,3895	0,0963	0,3896	**0,4836**	**0,4859**	**0,9695**	**0,1332**	**0,1212**
9	0,1042	0,4459	0,1066	0,4460	**0,5501**	**0,5526**	**1,1027**	**0,1332**	**0,1225**
10	0,1142	0,5023	0,1170	0,5024	**0,6165**	**0,6194**	**1,2359**	**0,1332**	**0,1236**
11	0,1242	0,5587	0,1273	0,5588	**0,6830**	**0,6861**	**1,3691**	**0,1332**	**0,1245**
12	0,1343	0,6151	0,1377	0,6152	**0,7494**	**0,7529**	**1,5022**	**0,1332**	**0,1252**
13	0,1443	0,6715	0,1480	0,6716	**0,8158**	**0,8196**	**1,6354**	**0,1332**	**0,1258**
14	0,1543	0,7279	0,1584	0,7280	**0,8823**	**0,8863**	**1,7686**	**0,1332**	**0,1263**
15	0,1644	0,7843	0,1687	0,7843	**0,9487**	**0,9531**	**1,9018**	**0,1332**	**0,1268**
16	0,1744	0,8407	0,1791	0,8407	**1,0151**	**1,0198**	**2,0350**	**0,1332**	**0,1272**

Table 19: OFB mode Energy Consumption

Table 19 collect data from OFB simulation, which is composed by two differents application. This two applications are complementary. It is possible obtains total accumulated energy consumption to each application (EncryptRadioOFB and DecryptRadioOFC). In the applications, CPU and Radio energy consumption are measured in Joules. The total accumulated energy consumption is calculated adding the simulation results for each application. Also is posible obtain total energy consumption measured in Joules and Watts using a mathematical relationship.

To simulate ECB, CFB and OFB modes are necessary 16 seconds because the application works with 128-bits and AES and data unit is 8-bits. Timer is set with 1000ms. Planing the algorithm, 128/8 = 16 iterations where each iteration will be calculated when the Timer is over.

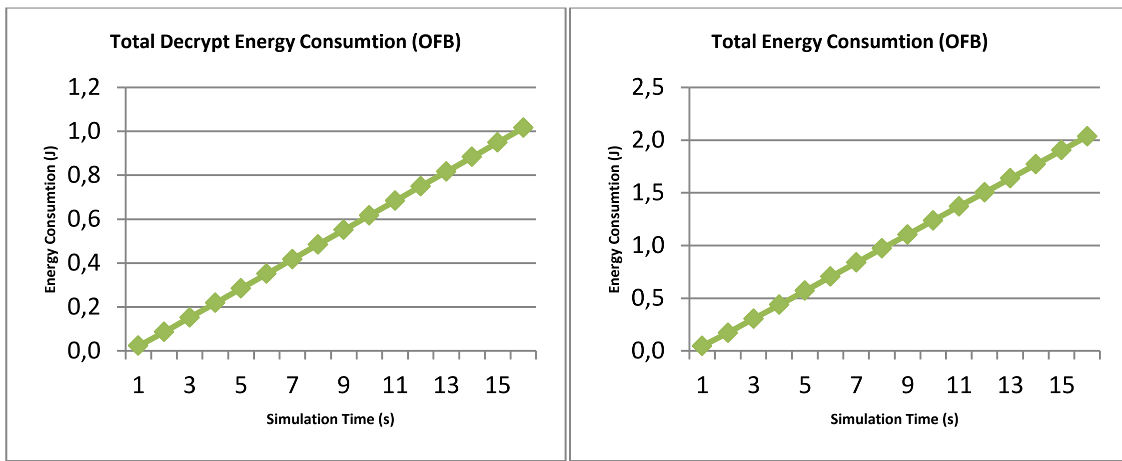

Figure 163: Total accumulated Decrypt energy consumption (J) [DecryptRadioOFB]

Figure 164: Total accumulated Decrypt energy consumption (J) [EncryptRadioOFS]

Figure 163 and Figure 164 are the total accumulated Encrypt energy consumption and the total accumulated Decrypt energy consumption for the OFB mode respectively. Total encrypt and total decrypt energy consumption have very similar results. Both graphs have a positive linear tendency along the time.

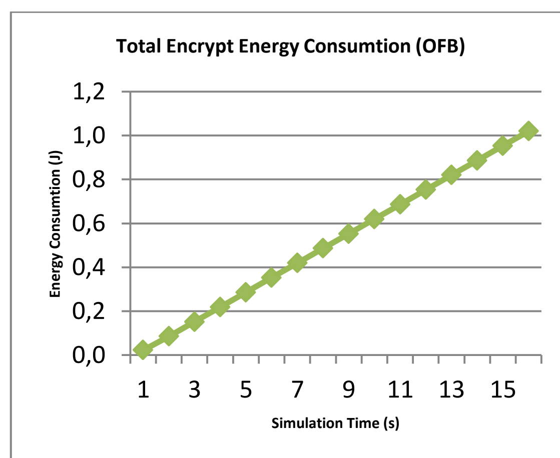

Figure 165: Total accumulated energy consumption (J) [OFB]

The Figure 165 shows the total accumulated energy consumption measured in Joules along the time. The program has been run for 16 seconds. As time goes on, the total accumulated energy consumption increases. The result is a positive linear tendency of the energy consumption along the time.

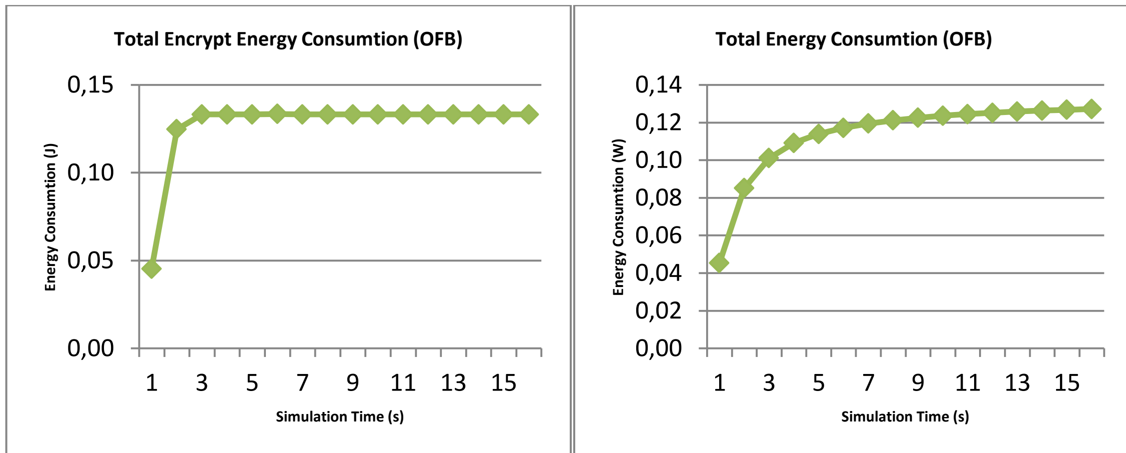

Figure 166: Total energy consumption J [OFB] *Figure 167: Total energy consumption W [OFB]*

It is very important know the total energy consumption where acts CPU, and Radio interface. Figure 166 shows the total energy consumption expressed in Joules. As previous applications, the first second is when the mote has the lowest energy consumption. From that moment, mote actives Radio interface to send (in EncryptRadioOFB) or receive (in EncryptRadioOFB) data to/from other motes and the total energy consumption is increased. From third second, the total energy consumption is stablished in 0.134 J, significantly higher value than previous applications. It demonstrates that cipher applications are more complex to the motes and they need more resources. Figure 167 shows the total energy consumption expressed in Watts As Figure 166, the first second is when the mote has the lowest consumption and since second two, the total energy consumption is increasing because Radio interface is on to send and receive messages. The total energy consumption W trends to 0.13W. Both Figures follow a positive logarithm tendency.

Motes have limited resources and the objective is to know how long is necessary to compile the code and include the binary file in the mote. In OFB mode, there are two applications. First, EncryptRadioOFB spends 0.12s to compile the application and 0.036s to insert application into the mote. The second one, DecryptRadioOFB spends 0.104s to compile the application and 0.048s to insert application into the mote once the application is compiled.

Conclusion

WSN and even more so, IoT, are not single technologies but instead represent complex systems using various technologies from physical communication layers to application programmes and are used in many application areas and different environments. This diversity has resulted in a complex standardization environment. As discussed in this White Paper there is already a large set of existing applications, challenges and ongoing standardization activities for WSNs. This can create opportunities for industry, research organizations and standardization bodies due to the unique characteristics of WSNs. This makes them attractive in current and future infrastructure applications.

It is important have a good understanding of the applications, the standardization environment, and the specific needs of WSN. As the number of nodes in large-scale WSN increase, the density of the network is also increased and the possibility of link failure becomes more frequent. Is recommended that further research should consider other network performance criteria such as the quality of service (QoS) issues for the real-time applications and node mobility in some special environments.

The fact, that the monitored data of the sensors, is first converted into digital signals, and then transmitted, benefits the fact, that a special WSN can contain a variety of different sensors in one network. Every node can also have multiple different sensors implemented on it. Of course, this is interesting for a huge amount of applications, e.g. weather surveillance or disease prevention systems. The number of resources necessaries to execute a program are also important because these data tell basically if the mote can run the program or not. Also the compilation and integration time can be very useful to know how the complex is a program. The resources necessaries in a mote, and times as compilation and compilation are collected in Table 20.

	Compilation Time (s)	Integration Time (s)	ROM (bytes)	RAM (bytes)
Blink	0,064	0,008	2044	51
BlinkSemaforo	0,048	0,012	2212	43
BlinkToRadioInv	0,096	0,044	12766	325
SenseRadio	0,104	0,032	13428	333
EncryptRadioECB	0,124	0,024	15054	1444
DecryptRadioECB	0,136	0,048	21216	2807
EncryptRadioCFB	0,14	0,044	21632	2630
DecryptRadioCFB	0,132	0,048	21234	2596
EncryptRadioOFB	0,12	0,036	21596	2612
DecryptRadioOFB	0,104	0,048	21238	2596

Table 20: Simulation measures

There is not any common platform where find relevant data about WSN but specifically about TinyOS, which is considered as a drawback . With the absence of a fixed infrastructure, wireless sensor nodes are forced to manage the small amounts of battery provided power, they have, carefully. This limits their computational power and memory size, and prevents them

from using full bandwidth due to higher energy costs. Working only on battery power, also means, that after a certain life span, a sensor node will die, because the battery is empty. Among other things, this fact leads to serious security that have to be kept in sight. Table 21 collects the energy consumption in AES applications as ECB, CFB and OFB.

Time (s)	ECB				CFB				OFB			
	EncryptRadioECB		DecryptRadioECB		EncryptRadioCFB		DecryptRadioCFB		EncryptRadioOFB		DecryptRadioOFB	
	Total Encrypt Energy Consumption (J)	Total Encrypt Energy Consumption (W)	Total Decrypt Energy Consumption (J)	Total Decrypt Energy Consumption (W)	Total Encrypt Energy Consumption (J)	Total Encrypt Energy Consumption (W)	Total Decrypt Energy Consumption (J)	Total Decrypt Energy Consumption (W)	Total Encrypt Energy Consumption (J)	Total Encrypt Energy Consumption (W)	Total Decrypt Energy Consumption (J)	Total Decrypt Energy Consumption (W)
1	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227	0,0227
2	0,0841	0,0421	0,0850	0,0425	0,0851	0,0426	0,0850	0,0425	0,0851	0,0425	0,0850	0,0426
3	0,1505	0,0502	0,1515	0,0505	0,1518	0,0506	0,1515	0,0505	0,1518	0,0505	0,1515	0,0506
4	0,2170	0,0543	0,2179	0,0545	0,2186	0,0547	0,2179	0,0545	0,2186	0,0545	0,2179	0,0547
5	0,2834	0,0567	0,2843	0,0569	0,2854	0,0571	0,2843	0,0569	0,2854	0,0569	0,2843	0,0571
6	0,3499	0,0583	0,3508	0,0585	0,3524	0,0587	0,3508	0,0585	0,3524	0,0585	0,3508	0,0587
7	0,4163	0,0595	0,4172	0,0596	0,4191	0,0599	0,4172	0,0596	0,4191	0,0596	0,4172	0,0599
8	0,4827	0,0603	0,4836	0,0605	0,4859	0,0607	0,4836	0,0605	0,4859	0,0605	0,4836	0,0607
9	0,5492	0,0610	0,5501	0,0611	0,5526	0,0614	0,5501	0,0611	0,5526	0,0611	0,5501	0,0614
10	0,6159	0,0616	0,6165	0,0617	0,6194	0,0619	0,6165	0,0617	0,6194	0,0617	0,6165	0,0619
11	0,6821	0,0620	0,6829	0,0621	0,6861	0,0624	0,6830	0,0621	0,6861	0,0621	0,6830	0,0624
12	0,7485	0,0624	0,7494	0,0625	0,7529	0,0627	0,7494	0,0625	0,7529	0,0625	0,7494	0,0627
13	0,8150	0,0627	0,8158	0,0628	0,8196	0,0630	0,8158	0,0628	0,8196	0,0628	0,8158	0,0630
14	0,8814	0,0630	0,8822	0,0630	0,8863	0,0633	0,8823	0,0630	0,8863	0,0630	0,8823	0,0633
15	0,9478	0,0632	0,9487	0,0632	0,9531	0,0635	0,9487	0,0632	0,9531	0,0632	0,9487	0,0635
16	1,0143	0,0634	1,0151	0,0634	1,0198	0,0637	1,0151	0,0634	1,0198	0,0634	1,0151	0,0637

References

[1] H. Sye Loong Keoh; Kumar, S.S; Tschofening, “Securing the Internet of Things: A Standardization Perspective,” *IEEE*, vol. Vol. 1, no. no. 3, p. pp.265,275.

[2] M. Dener, “Security analysis in wireless sensor networks,” *Int. J. Distrib. Sens. Networks*, vol. 2014, 2014.

[3] Joe Tillison, “An introduction to wireless sensor network concepts: http://www.eetimes.com/document.asp?doc_id=1278992.”

[4] W. Xu, “Introduction to TinyOS Programming TinyOS Network Communication TinyOS Programming UC Berkeley Family of Motes MTS300CA Sensor Board Programming Board (MIB510) Hardware Setup Overview Lecture Overview Hardware Primer Introduction to TinyOS Programming T.”

[5] T. Resources, “Getting started with nesC: http://www.advanticsys.com/wiki/index.php?title=Getting_started_with_nesC.”

[6] A. Ferah, “White Paper AES Encryption AES Encryption and Related Concepts,” pp. 0–4, 2013.

[7] TinyOS, “TinyOS Official Web Page: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki.”

[8] D. Gay, P. Levis, D. Culler, and E. Brewer, “nesC 1.1 Language Reference Manual,” *Changes*, no. May, pp. 1–28, 2003.

[9] P. Levis and D. Gay, “TinyOS Programming,” *Cambridge Univ. Press*, vol. 1st. Ed., pp. 21–36, 2009.

[10] J. T. Adams, “An introduction to IEEE STD 802.15.4,” *2006 IEEE Aerosp. Conf.*, pp. 1–8, 2006.

[11] IEEE 802.15, “El estandar IEEE 802.15.4,” pp. 1–18, 2007.

[12] MEMSIC Inc., “MICAz datasheet: 6020-0065-05 rev,” *San Jose, CA, Calif.*, pp. 1–2, 2003.

[13] Crossbow Technology Inc., “TelosB Mote Platform,” pp. 1–2, 2004.

[14] Crossbow Technology, “iMote 2 datasheet,” pp. 2–4, 2012.

[15] “EPIC Datasheet: http://people.eecs.berkeley.edu/~prabal/projects/epic/.”

[16] W. Stallings, *Cryptography and Network Security:Principles and Practice*, Prentice H. 2013.

[17] B. L. Titzer, J. Palsberg, D. K. Lee, and O. Landsiedel, “Official Avrora web page: http://compilers.cs.ucla.edu/avrora/.”

[18] E. Decket, “User guide (Debian based systems): http://tinyprod.net/repos/debian/.”

[19] C. Solutions, “Cygwin official web page: https://www.cygwin.com/.”

[20] “GitHub official web page: https://github.com/.”

[21] “AES online calculator tools: http://aes.online-domain-tools.com/.”

Annexed I: Artículo AJICT

Un caso práctico de aporte de seguridad en IoT

Martinez-Caro J.-M.; Cano M.-D.
Departamento de Tecnologías de la Información y las Comunicaciones
Universidad Politécnica de Cartagena
Teléfono: 968325953
Email: martinezcaro92@gmail.com

Resumen. *Se espera que el nuevo ecosistema digital creado a partir de Internet de las Cosas sea el mayor en el ámbito de las Tecnologías de la Información y las Comunicaciones. En este contexto, la seguridad se presenta como uno de los mayores retos a abordar desde distintas vertientes: provisión de privacidad, relaciones de confianza, diseño nativo de aplicaciones seguras, etc. En este trabajo, presentamos un caso práctico de implementación software del algoritmo de cifrado AES en dispositivos inalámbricos Micaz usados en redes inalámbricas de sensores basadas en el estándar IEEE 802.15.4.*
Palabras clave. *IoT, Seguridad, IEEE 802.15.4, AES y TinyOS.*

Abstract. *It is expected that the new digital ecosystem created from the Internet of Things will be the greatest in the field of Information and Communication Technology. In this context, security is presented as one of the main challenges to be addressed from different areas: provision of privacy, trust relationships, native design of secure applications, etc. In this paper, we present an experimental example of software implementation of the AES encryption algorithm for Micaz motes used in IEEE 802.15.4-based wireless sensor networks.*
Keywords. *IoT, Security, IEEE 802.15.4, AES, and TinyOS.*

1. Introducción

La Internet de las Cosas (*Internet of Things*, IoT) es un nuevo paradigma en el que todas las "cosas", desde electrodomésticos hasta *wearables* pasando por vehículos o sensores estarán conectados a Internet. Se espera que el nuevo ecosistema digital creado a partir de IoT sea el mayor en el ámbito de las Tecnologías de la Información y las Comunicaciones (Aguzzi, 2015). Ejemplos clásicos de uso de IoT son: frigoríficos capaces de conocer los alimentos restantes en su interior y llevar a cabo la compra en función de los planes del usuario y el stock, los *weareables* que se conectan a Internet para registrar todos los datos que captan sobre su usuario, mejoras en el sistema de conducción de vehículos, la automatización del hogar o la automatización de las ciudades, haciéndolas más inteligentes (*SmartGrids*) y eficaces.

En este mundo IoT híper-conectado, la seguridad surge como uno de los principales retos a resolver (Keoh, 2014). Una forma lógica de analizar la seguridad en IoT es revisar las soluciones existentes para las ya establecidas redes de sensores inalámbricas (*Wireless Sensor Networks*, WSN). Las WSN están pensadas para que los diferentes nodos, denominados *motes*, se comuniquen entre ellos intercambiando información obtenida de sensores, tomen decisiones y/o lleven a cabo operaciones simples en base a la información obtenida, entre otras. Son varios los problemas y amenazas posibles en las WSN, por ejemplo (Dener, 2014): *Denial-of Service (DoS)*, *Attacks on Information in transit*, *Sybil Attack*, etc. Aunque algunos de estos problemas han sido solucionados, todavía quedan importantes temas a resolver como por ejemplo (Granjal, 2015) la autenticación de origen, el establecimiento de claves criptográficas, etc. En este trabajo, presentamos un estudio inicial de las prestaciones del

algoritmo criptográfico AES (*Advanced Encryption Standard*) (FIPS, 2001), en términos de consumo energético, que pueda servir de ejemplo práctico de cómo introducir un alto nivel de confidencialidad y autenticación en WSN como base para un estudio posterior ampliado con IoT. Las pruebas se han llevado a cabo en un banco de pruebas experimental montado y configurado a tal efecto.

El resto del artículo se organiza de la siguiente forma. La sección 2 describe los elementos HW y SW del banco de pruebas e introduce el algoritmo criptográfico AES. La sección 3 incluye las pruebas realizadas y los resultados. El documento finaliza con las conclusiones en la sección 4.

2. Descripción del banco de pruebas

En esta sección se describen los componentes hardware (Micaz) y software (TinyOS y nesC) del banco de pruebas que se ha configurado para llevar a cabo el estudio. Asimismo, se resumen las características más importantes del algoritmo AES.

2.1 Hardware y software

TinyOS es un sistema operativo *(Operating System*, OS*)* diseñado para sistemas embebidos inalámbricos de baja potencia (TinyOS, 2016). Básicamente, se trata de un planificador de trabajo y una colección de *drivers* para microcontroladores y otros circuitos integrados comúnmente usados en plataformas embebidas inalámbricas. Algunos de los aspectos importantes para el diseño de TinyOS son los siguientes (Levis, 2005):

1. Recursos limitados. Los *motes* tienen recursos físicos limitados debido a que están planteados para localizarse en lugares estratégicos, minimizando el coste y donde las fuentes de energía son limitadas.
2. Concurrencia reactiva. Un *mote* es el responsable de muestrear datos mediante el uso de sensores, procesar dichos datos y transmitirlos; también pueden participar en tareas de procesamiento distribuido.
3. Flexibilidad. Esta característica se atribuye a las diferentes aplicaciones y hardware que se pueden encontrar, donde el objetivo fundamental es reducir el espacio y el consumo de los *motes*.
4. Baja consumo. Para ahorrar en el consumo de energía sólo se activan las partes a utilizar del circuito que serán definidas en el código.

Para instalar TinyOS se puede optar por diferentes opciones, entre ellas:

1. Descarga de máquina virtual donde quedará todo listo para ser utilizado. Esta máquina virtual se importará en *VMWare* o *VirtualBox* para su utilización. Entre ellas, *XubuntOS* y *UbuntOS*.
2. Utilizando la herramienta clone de *Github* y definiendo ciertas variables dentro del fichero *~bashrc*.
3. Importando librerías y usando el *comando apt-get install*. Posteriormente se deberán definir ciertas variables dentro del fichero *~bashrc*.
4. Utilización de paquetes *RPM* para las versiones de *Unix* y *Windows*.

Por su parte, nesC es una extensión de *C* especialmente diseñada para correr sobre TinyOS y ser utilizada en WSN. Los conceptos básicos de este lenguaje de programación son (TinyOS, 2016; Levis, 2005; Gay, 2003):

1. Separación de la construcción y la composición: los programas son construidos a partir de componentes y son ensamblados para formar programas completos.
2. Las interfaces son bidireccionales y proporcionadas o usadas por componentes.
3. Los componentes se unen estáticamente entre sí a través de sus interfaces.
4. El modelo de concurrencia de nesC está basado en tareas de ejecución. Los controladores de interrupciones pueden interrumpir las tareas.

Finalmente, el nodo inalámbrico (*mote*) escogido para el banco de pruebas es el *Micaz* (Micaz, 2016) (véase Fig. 1). Este dispositivo utiliza la banda de radio 2.4GHz y el estándar IEEE 802.15.4 (IEEE802.15.4, 2006). Es capaz de medir parámetros como temperatura, presión, aceleración o sonido, entre otros, mediante el uso de la placa de sensores mts300.

Además, utiliza la programadora mib520 (USB) o mib510 (puerto serie) (Micaz, 2016).

2.2. El algoritmo AES

Cuando se emplea el término seguridad se hace referencia a la consecución de tres objetivos: autenticación, integridad y confidencialidad. Mediante la autenticación se garantiza que un usuario/máquina es quien dice ser. Habrá integridad si durante el intercambio de información, ésta no ha sido modificada o eliminada por parte de usuarios malintencionados. Por último, la confidencialidad se produce cuando sólo los componentes autorizados del sistema pueden acceder al elemento protegido, por ejemplo cuando sólo transmisor y receptor pueden ver el contenido de un mensaje. El algoritmo AES es el estándar actual para el cifrado de comunicaciones (FIPS, 2001). Se trata de un algoritmo de cifrado en bloque simétrico que acepta bloques de entrada de 128, 192 o 256 bits y tamaños de clave de 128, 192 o 256 bits. Realiza todas las operaciones a nivel de byte. Es un algoritmo iterativo en el que el número de rondas depende del tamaño del bloque de texto plano y del tamaño de la clave. Hoy en día no se conoce ningún ataque por criptoanálisis a AES, siendo el ataque por fuerza fruta inviable.

El estándar 802.15.4 (IEEE802.15.4, 2006) define 8 *suits* de seguridad en la capa de control de acceso al medio que se pueden agrupar en: sólo autenticación

Fig. 1. *Mote* Micaz empleado en el estudio.

(AES-CBC-MAC), sólo cifrado (AES-CTR), cifrado y autenticación (AES-CCM), y no seguridad. Cada una de estas *suits* ofrece variantes en función del tamaño del código de autenticación del mensaje (*Message Authentication Code*, MAC) empleado para la autenticación pero en todas ellas se hace uso de AES para proporcionar confidencialidad (si procede). No todas las *suits* son obligatorias a la hora de crear implementaciones de este estándar. Es importante señalar que se han detectado fuertes vulnerabilidades a las *suits* propuestas, por ejemplo (Cao, 2016).

3. Caso práctico

En primer lugar, se trata de conocer el entorno en el que el investigador se va a manejar. Uso de una máquina que utiliza un sistema *Linux (XubuntOS)* sobre una máquina virtual *(VirtualBox)*. Una vez sea importado todo, se conocerán las herramientas o programas disponibles en abierto para su funcionamiento dependiendo de las necesidades del usuario. No sólo están disponibles herramientas o programas sino también código *nesC* y aplicaciones predefinidas, que ayudarán al usuario a comprender el funcionamiento de esta plataforma.

Se pretende desarrollar una solución de seguridad para la transferencia de datos por el medio. En este caso, se trata de la implementación software de un cifrado y descifrado AES para WSN implementado y testado en los dispositivos Micaz.

3.1. Primeros pasos

Una vez se conoce el entorno en el que el usuario va a trabajar, llega el momento donde hay que empezar a modificar y crear

nuevo código para comprender y asimilar el lenguaje *nesC*. Con este fin, se han creado diferentes aplicaciones para modificar y/o ampliar las funcionalidades a los programas predefinidos (este trabajo no se ha incluido aquí dada la limitación de espacio).

Una vez que los cambios han sido definidos (o el nuevo código creado), es necesario compilar el código e introducirlo en los *motes* para comprobar que el código creado tiene la funcionalidad deseada. Es posible que sean necesarios varios *motes* con programas diferentes para testear los cambios realizados.

3.2. Desarrollo

Se trata de definir la estructura básica de las aplicaciones definidas en *TinyOS*. Consta de diferentes secciones como: definición de *#includes*, *module*{}, e *implementation*{}.

En primer lugar, se especifican una serie de *#includes* donde se importan las declaraciones definidas en otros ficheros, incluyendo los *#includes* anidados. Continúa con una sección *module*{} donde principalmente se declaran las interfaces que serán utilizadas en esta aplicación. Por último, se define la sección *implementation*{} donde las variables son definidas, inicializadas y se desarrollan operaciones con las mismas. En esta sección se define un *event void Boot.booted()*{} que tiene la misma funcionalidad que el clásico método *main()* definido en múltiples aplicaciones.

En el caso práctico que se desea abordar, son varios los tipos de datos a utilizar, desde datos tipo enteros hasta *arrays* unidimensionales y bidimensionales de datos enteros (de tipo *signed* y *unsigned*). Para optimizar el código se pretende llevar a cabo funciones o métodos de las operaciones más comunes como imprimir matriz o inicializar variables. Las etapas a desarrollar en nuestro caso para implementar AES son: *(1)SubBytes*, *(2)ShiftRows*, *(3)MixColums* y *(4)AddRoundKey*. De forma previa a cada etapa, se lleva a cabo una inicialización de variables, o *reset*, para evitar así que estas tomen valores erróneos de usos anteriores. Otro objetivo es el de minimizar el número de variables en la aplicación para que el código sea lo más eficiente posible, usando así una menor cantidad de memoria, debido a las limitadas capacidades disponibles en los dispositivos.

La etapa *(1)SubBytes* sustituye el dato de 8 bits introducido en función de la *S-box*. Los cuatro bits más significativos indican la fila y los cuatro bits menos significativos indican la columna del dato en la *S-Box*. Por ejemplo, véase la Fig.2, si el dato introducido es 0x31 buscamos en la *S-Box* fila 3, columna 1 y el nuevo dato será 0xC7.

	0	1	2
0	63	7C	77
1	CA	82	C9
2	B7	FD	93
3	04	C7	23

Fig.2. Sustitución de valores del proceso *SubBytes*.

En *(2)ShiftRows* se modificará la matriz rotando las columnas hacia la izquierda de la matriz a partir de la segunda fila (la primera fila se mantiene). El número de columnas rotadas se incrementará una unidad por cada fila (véase ejemplo en Fig. 3).

En el proceso *(3)MixColums*, se procede a multiplicar la matriz resultante con una matriz predefinida [4x4]. Como las multiplicaciones son computacionalmente costosas, se lleva a cabo un proceso de sustitución, similar al anterior, usando la *tabla L* y la *tabla E* de AES. Las operaciones llevadas a cabo se muestran en la Fig.4. Por último, en el proceso *(4)AddRoundKey* cada elemento de la matriz [i, j] es combinado con un elemento

de la matriz de subclave [i, j] usando la operación XOR como indica la Fig. 5.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & A & B \\ C & D & E & F \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 4 \\ A & B & 8 & 9 \\ F & C & D & E \end{bmatrix}$$

Fig. 3. Rotación de matrices del proceso *ShiftRows*.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} D4 \\ BF \\ 5D \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ E5 \end{bmatrix}$$

$$R_{0,0} = (D4 * 02) \oplus (BF * 03) \oplus (5D * 01) \oplus (30 * 01)$$
$$R_{0,0} = E(L(D4)+L(02)) \oplus E(L(BF)+E(03)) \oplus E(L(5D)+L(01)) \oplus E(L(30)+L(01))$$
$$R_{0,0} = E(19+41) \oplus E(01+9D) \oplus (0+88) \oplus (0+65)$$
$$R_{0,0} = E(5A) \oplus E(9E) \oplus E(88) \oplus E(65) = B3 \oplus DA \oplus 5D \oplus 30$$
$$R_{0,0} = 04$$

Fig. 4. Operaciones del proceso *MixColums*.

$$b_{2,2} = a_{2,2} \oplus k_{2,2}$$

Fig. 5. Operación *XOR* del proceso *AddRoundKey*.

Para una descripción completa del algoritmo por favor acudir a (FIPS, 2001).

3.3. Resultados

Los resultados se han obtenido una vez creado y compilado el código. El objetivo es conocer el tiempo necesario para llevar a cabo la operación, analizar el correcto funcionamiento del proceso de cifrado/descifrado y el consumo. Es importante señalar que para este estudio inicial sólo se ha tenido en cuenta una iteración de AES, por lo que los resultados serán del orden de diez veces menores que el final.

El tiempo de compilación e instalación de la aplicación en el sensor es de 9,6 s. La cantidad de memoria *ROM* necesaria para este caso es de 10950 bytes y 2292 bytes de memoria *RAM*. Los resultados obtenidos tras el cifrado se muestran en la Fig. 6.

Mediante la herramienta *Avrora* (Avrora, 2016) se pretende conocer el consumo energético necesario por el *mote* para llevar a cabo las operaciones programadas. Debido a que la ejecución tarda cinco segundos, la potencia eléctrica consumida por el dispositivo es 12,9 mW, tal y como se indica en (1) y en la Fig. 7.

```
Print Value to Cipher:
0x00000011 , 0x00000012 , 0x00000013 , 0x00000014
0x00000021 , 0x00000022 , 0x00000023 , 0x00000024
0x00000031 , 0x00000032 , 0x00000033 , 0x00000034
0x00000041 , 0x00000042 , 0x00000043 , 0x00000044
Print Key:
0x00000001 , 0x00000002 , 0x00000003 , 0x00000004
0x00000005 , 0x00000006 , 0x00000007 , 0x00000008
0x00000009 , 0x0000000a , 0x0000000b , 0x0000000c
0x0000000d , 0x0000000e , 0x0000000f , 0x00000010
Print RESULT CIFRADO:
0x0000001a , 0x000000a1 , 0x000000e0 , 0x0000004d
0x00000013 , 0x00000097 , 0x0000008b , 0x000000d1
0x00000054 , 0x00000075 , 0x00000068 , 0x0000002b
0x000000cb , 0x000000e1 , 0x000000e2 , 0x00000084
```

Fig. 6. Resultados de cifrado

```
CPU: 0.06448136112255859 Joule
   Active: 0.02567715432023112 Joule, 8339722 cycles
   Idle: 0.03880420680232747 Joule, 28524278 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Extended Standby: 0.0 Joule, 0 cycles
```

Fig. 7. Resultados de consumo energético.

$$W = \frac{J}{s} = \frac{0.06448136}{5s} = 0.012896\frac{J}{s} = 0.012896W \ (1)$$

4. Conclusión

En este trabajo, hemos mostrado de forma sucinta cómo implementar el algoritmo AES tal y como aparece en el estándar para el envío de datos cifrados en una WSN. Los nodos inalámbricos de las WSN son limitados en recursos, pero aun así, podrían ser suficientes para poder desarrollar medidas de seguridad. Como próximo trabajo, se finalizará la implementación completa de AES y se evaluarán soluciones de seguridad en la capa de control de acceso al medio de 802.15.4 como uno de los componentes de la futura IoT.

Referencias

[1] Aguzzi S. *et al.* (2015). *Definition of a Research and Innovation Policy Leveraging Cloud Computing and IoT*

Combination. Available online: <https://goo.gl/TuLlpj>.

[2] Avrora. The AVR simulation and analysis framework, UCLA. Available online: <http://compilers.cs.ucla.edu/avrora/profiling.html>

[3] Cao X. *et al.* (2016), “Ghost-in-ZigBee: Energy Depletion Attack on ZigBee based Wireless Networks”, IEEE Internet of Things Journal, DOI 10.1109/JIOT.2016.2516102.

[4] Dener M. (2014). “Security Analysis in Wireless Sensor Networks”, Intl Journal of Distributed Sensor Networks, pp. 1-9.

[5] FIPS Pub 197 (2001). *Advanced Encryption Standard*, National Institute of Standards and Technology.

[6] Gay D. *et al.* (2003). “nesC 1.1 Language Reference Manual”, manual.

[7] Granjal J. *et al.* (2015). “Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues”, IEEE Communication Surveys & Tutorials, 17 (3).

[8] IEEE Standard 802.15.4 (2006). “Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)”.

[9] Keoh S. L. *et al.* (2014). “Securing the Internet of Things: A Standardization Perspective”, IEEE Internet of Things Journal, 1 (3), 2014.

[10] Levis P. *et al.* (2005). “TinyOS: An Operating System for Sensor Networks”, Ambient Intelligence, pp 115-148.

[11] Micaz datasheet, Memsic Inc (2016). Available online: <http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf>

[12] TinyOS Official Website (2016). Available online: <http://tinyos.stanford.edu/tinyos-wiki/index.php/>

Annexed II: OscilloscopeC.nc Code

```
//OsciloscopeC.nc
#include "Timer.h"
#include "Oscilloscope.h"
module OscilloscopeC @safe()
{
  uses {
    interface Boot;
    interface SplitControl as RadioControl;
    interface AMSend;
    interface Receive;
    interface Timer<TMilli>;
    interface Read<uint16_t>;
    interface Leds;
  }
}
implementation
{
  message_t sendBuf;
  bool sendBusy;
  oscilloscope_t local;
  uint8_t reading;
  bool suppressCountChange;
  void report_problem() { call Leds.led0Toggle(); }
  void report_sent() { call Leds.led1Toggle(); }
  void report_received() { call Leds.led2Toggle(); }
  event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    if (call RadioControl.start() != SUCCESS)
      report_problem();
  }
  void startTimer() {
    call Timer.startPeriodic(local.interval);
    reading = 0;
  }
  event void RadioControl.startDone(error_t error) {
    startTimer();
  }
  event void RadioControl.stopDone(error_t error) {
  }
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    oscilloscope_t *omsg = payload;
    report_received();
    if (omsg->version > local.version)
      {
        local.version = omsg->version;
        local.interval = omsg->interval;
        startTimer();
```

```
      }
    if (omsg->count > local.count)
      {
        local.count = omsg->count;
        suppressCountChange = TRUE;
      }
    return msg;
  }
  event void Timer.fired() {
    if (reading == NREADINGS)
      {
        if (!sendBusy && sizeof local <= call AMSend.maxPayloadLength())
          {
            memcpy(call AMSend.getPayload(&sendBuf, sizeof(local)), &local, sizeof local);
            if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf, sizeof local) == SUCCESS)
              sendBusy = TRUE;
          }
        if (!sendBusy)
          report_problem();
        reading = 0;
        if (!suppressCountChange)
          local.count++;
        suppressCountChange = FALSE;
      }
    if (call Read.read() != SUCCESS)
      report_problem();
  }
  event void AMSend.sendDone(message_t* msg, error_t error) {
    if (error == SUCCESS)
      report_sent();
    else
      report_problem();
    sendBusy = FALSE;
  }
  event void Read.readDone(error_t result, uint16_t data) {
    if (result != SUCCESS)
      {
        data = 0xffff;
        report_problem();
      }
    if (reading < NREADINGS)
      local.readings[reading++] = data;
  }
}
```

Annexed III: BaseStationP.nc Code

```
//BaseStationP.nc
#include "AM.h"
#include "Serial.h"
module BaseStationP @safe() {
  uses {
    interface Boot;
    interface SplitControl as SerialControl;
    interface SplitControl as RadioControl;
    interface AMSend as UartSend[am_id_t id];
    interface Receive as UartReceive[am_id_t id];
    interface Packet as UartPacket;
    interface AMPacket as UartAMPacket;

    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Receive as RadioSnoop[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;
    interface Leds;
  }
}
implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };
  message_t  uartQueueBufs[UART_QUEUE_LEN];
  message_t  * ONE_NOK uartQueue[UART_QUEUE_LEN];
  uint8_t    uartIn, uartOut;
  bool       uartBusy, uartFull;
  message_t  radioQueueBufs[RADIO_QUEUE_LEN];
  message_t  * ONE_NOK radioQueue[RADIO_QUEUE_LEN];
  uint8_t    radioIn, radioOut;
  bool       radioBusy, radioFull;
  task void uartSendTask();
  task void radioSendTask();
  void dropBlink() {
    call Leds.led2Toggle();
  }
  void failBlink() {
    call Leds.led2Toggle();
  }
  event void Boot.booted() {
    uint8_t i;
    for (i = 0; i < UART_QUEUE_LEN; i++)
      uartQueue[i] = &uartQueueBufs[i];
```

```
    uartIn = uartOut = 0;
    uartBusy = FALSE;
    uartFull = TRUE;
    for (i = 0; i < RADIO_QUEUE_LEN; i++)
      radioQueue[i] = &radioQueueBufs[i];
    radioIn = radioOut = 0;
    radioBusy = FALSE;
    radioFull = TRUE;
    if (call RadioControl.start() == EALREADY)
      radioFull = FALSE;
    if (call SerialControl.start() == EALREADY)
      uartFull = FALSE;
  }
  event void RadioControl.startDone(error_t error) {
    if (error == SUCCESS) {
      radioFull = FALSE;
    }
  }
  event void SerialControl.startDone(error_t error) {
    if (error == SUCCESS) {
      uartFull = FALSE;
    }
  }
  event void SerialControl.stopDone(error_t error) {}
  event void RadioControl.stopDone(error_t error) {}
  uint8_t count = 0;
  message_t* ONE receive(message_t* ONE msg, void* payload, uint8_t len);

  event message_t *RadioSnoop.receive[am_id_t id](message_t *msg,
                                                   void *payload,
                                                   uint8_t len) {
    return receive(msg, payload, len);
  }

  event message_t *RadioReceive.receive[am_id_t id](message_t *msg,
                                                     void *payload,
                                                     uint8_t len) {
    return receive(msg, payload, len);
  }
  message_t* receive(message_t *msg, void *payload, uint8_t len) {
    message_t *ret = msg;
    atomic {
      if (!uartFull)
        {
          ret = uartQueue[uartIn];
          uartQueue[uartIn] = msg;
          uartIn = (uartIn + 1) % UART_QUEUE_LEN;

          if (uartIn == uartOut)
            uartFull = TRUE;
          if (!uartBusy)
```

```
          {
            post uartSendTask();
            uartBusy = TRUE;
          }
      }
    else
      dropBlink();
  }

  return ret;
}
uint8_t tmpLen;

task void uartSendTask() {
  uint8_t len;
  am_id_t id;
  am_addr_t addr, src;
  message_t* msg;
  am_group_t grp;
  atomic
    if (uartIn == uartOut && !uartFull)
      {
        uartBusy = FALSE;
        return;
      }
  msg = uartQueue[uartOut];
  tmpLen = len = call RadioPacket.payloadLength(msg);
  id = call RadioAMPacket.type(msg);
  addr = call RadioAMPacket.destination(msg);
  src = call RadioAMPacket.source(msg);
  grp = call RadioAMPacket.group(msg);
  call UartPacket.clear(msg);
  call UartAMPacket.setSource(msg, src);
  call UartAMPacket.setGroup(msg, grp);
  if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
    call Leds.led1Toggle();
  else
    {
      failBlink();
      post uartSendTask();
    }
}
event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
  if (error != SUCCESS)
    failBlink();
  else
    atomic
      if (msg == uartQueue[uartOut])
        {
          if (++uartOut >= UART_QUEUE_LEN)
            uartOut = 0;
```

```
          if (uartFull)
            uartFull = FALSE;
        }
    post uartSendTask();
  }
  event message_t *UartReceive.receive[am_id_t id](message_t *msg,
                                                   void *payload,
                                                   uint8_t len) {
    message_t *ret = msg;
    bool reflectToken = FALSE;
    atomic
      if (!radioFull)
        {
          reflectToken = TRUE;
          ret = radioQueue[radioIn];
          radioQueue[radioIn] = msg;
          if (++radioIn >= RADIO_QUEUE_LEN)
            radioIn = 0;
          if (radioIn == radioOut)
            radioFull = TRUE;
          if (!radioBusy)
            {
              post radioSendTask();
              radioBusy = TRUE;
            }
        }
      else
        dropBlink();
    if (reflectToken) {
      //call UartTokenReceive.ReflectToken(Token);
    }
    return ret;
  }
  task void radioSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr,source;
    message_t* msg;

    atomic
      if (radioIn == radioOut && !radioFull)
        {
          radioBusy = FALSE;
          return;
        }
    msg = radioQueue[radioOut];
    len = call UartPacket.payloadLength(msg);
    addr = call UartAMPacket.destination(msg);
    source = call UartAMPacket.source(msg);
    id = call UartAMPacket.type(msg);
    call RadioPacket.clear(msg);
```

```
    call RadioAMPacket.setSource(msg, source);

    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
      call Leds.led0Toggle();
    else
      {
        failBlink();
        post radioSendTask();
      }
  }
  event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
        if (msg == radioQueue[radioOut])
          {
            if (++radioOut >= RADIO_QUEUE_LEN)
              radioOut = 0;
            if (radioFull)
              radioFull = FALSE;
          }

    post radioSendTask();
  }
}
```

Annexed IV: EncryptRadioECBC.nc Code

```
// $Id: EncryptRadioECBC.nc,v 1.6 2010-06-29 22:07:40 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioECB.h"

module EncryptRadioECBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;
        bool detect = FALSE;

        uint8_t     basedata     [4][4]=     {{0x32,0x88,0x31,0xE0},     {0x43,0x5A,0x31,0x37},
{0xF6,0x30,0x98,0x07}, {0xA8,0x8D,0xA2,0x34}};
        uint8_t M [4][4]= {{0x02,0x03,0x01,0x01}, {0x01,0x02,0x03,0x01}, {0x01,0x01,0x02,0x03},
{0x03,0x01,0x01,0x02}};
        uint8_t    key_original    [4][4]=    {{0xAF,0xAE,0xAD,0xAC},    {0xAB,0xAA,0xA9,0xA8},
{0xA7,0xA6,0xA5,0xA4}, {0xA3,0xA2,0xA1,0xA0}};

       uint8_t n = 10;
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W_t [4];
        uint8_t W_tmp [4];
        uint8_t  R[15][4]=  {{0x01,  0x00,  0x00,  0x00},  {0x02,  0x00,  0x00,  0x00},  {0x04,  0x00,
0x00,  0x00},  {0x08,  0x00,  0x00,  0x00},  {0x10,  0x00,  0x00,  0x00},  {0x20,  0x00,  0x00,  0x00},
{0x40,  0x00,  0x00,  0x00},  {0x80,  0x00,  0x00,  0x00},  {0x1B,  0x00,  0x00,  0x00},  {0x36,  0x00,
0x00,  0x00},  {0x6C,  0x00,  0x00,  0x00},  {0xD8,  0x00,  0x00,  0x00},  {0xAB,  0x00,  0x00,  0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        int8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 0;
        uint8_t b0;
        uint8_t rounds;
        uint8_t rounds_col;


        uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
```

```
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

        uint8_t E_table [16][16] = {{0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4,
0xF7, 0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB,
0x26, 0x6A, 0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C,
0x44, 0xCC, 0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0,
0x3B, 0x4D, 0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0,
0x6B, 0xBD, 0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57,
0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19,
0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB,
0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41},
{0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA,
0x75}, {0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E,
0x89, 0x80}, {0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8,
0x43, 0xC5, 0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99,
0xB0, 0xCB, 0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E,
0x42, 0xC6, 0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A,
0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01}};

        uint8_t L_table [16][16] = {{0x00, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};

        void initVar()
        {
                rounds_col = sizeof(basedata[0]);
                rounds = sizeof(basedata)/rounds_col;
                i = 0;
                j = 0;
                k = 0;
                it2 = 0;
                it3 = 0;
                it4 = 0;
                it5 = 0;
                it6 = 0;
        }

        void selectColRow(uint8_t data)
        {
                fh = data & fh_ex;      //AND 4bits mayores
                lh = data & lh_ex;      //AND 4bits menores
                fh = fh >> 4;           //desplazamiento 4 bits derecha
        }
```

```
        void         rotacionMatrix(uint8_t         matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        { //SALIDA result2
           do
           {
                if (i == 0)
                {
                        j = 0;
                        do {
                          result2[i][j]=matrix[i][j];
                          j++;
                        } while (j<sizeof(basedata[0]));
                        j = 0;
                } else{
                //datos en forma de matriz
                do
                {
                   it2 = j - i;// round_col - round;
                   if (it2 > 0) //perfecto
                   {
                        result2[i][it2] = matrix[i][j];
                   } else if (it2 == 0) {
                        result2[i][0] = matrix[i][j];
                   } else if (it2 < 0){
                        it3 = rounds_col + it2;
                        result2[i][it3] = matrix[i][j];
                   }
                   j++;
                } while (j < rounds_col);
                j = 0;
             }
             i++;
           } while (i < rounds);
        }

        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  event void Boot.booted() {
    //INICIALIZACIÓN DE VARIABLES
        initVar();

        //GENERACIÓN DE SUBMATRICECS
        //for (i;i<sizeof(W);i++)
        for (i;i<sizeof(key[0]);i++)
        {
```

```
		if (i < sizeof(key_original[0]))
		{
			k = 0;
			do
			{
				key[k][i] = key_original[k][i];
				k++;
			} while (k < sizeof(key_original[0]));
			} else {
			k = 0;
			do
			{
				W_t [k] = key[k][i-1];
				k++;
			} while(k < sizeof(W_t));

			k = 0;
			if (i%sizeof(key_original[0]) == 0)
			{
				it3 = W_t[0];
				for (k; k < sizeof(W_t); k++)   // FOR ROTACION
				{
					W_tmp[k] = W_t[k+1];
				}
				W_tmp[sizeof(W_t)-1] = it3;

				k = 0;
				for (k; k < sizeof(W_t); k++)     // FOR SUSTITUCIÓN
				{
					selectColRow(W_tmp[k]);
					W_tmp[k] = table [fh][lh];
				}

				k = 0;
				for (k; k < sizeof(W_t); k++)   // FOR XOR con R
				{
				    W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
				}
			}

			k = 0;
			for (k; k < sizeof(W_t); k++)           // FOR XOR con R
			{
				W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
			}

			k = 0;
			for (k; k < sizeof(W_t); k++)   // FOR ASIGNACIÓN EN KEY
			{
				key [k][i] = W_tmp[k];
			}
		}
	}

	//XOR CIFRADO FINAL
	initVar();
	do
	{
		do
		{
			result_f[j][i] = basedata[j][i] ^ key[j][col_key];
			j++;
		} while (j < rounds_col);
		j = 0; i++; col_key++;
	} while(i < rounds);

	//ITERACIONES n = 10
	b0 = 0;
	for(b0;b0 < n;b0++)
	{
		//PROCESO DE SUBSTITUCION       //SALE CON result1
		initVar();
		sustitucionMatrix(result_f);
```

```
                //PROCESO DE ROTACIÓN          //SALE CON result2
                initVar();
                rotacionMatrix(result1);

                //PROCESO MIXCOLUMNS    //SALE CON result1
                initVar();
                if ( b0 < n-1)
                {
                  for (i;i<rounds;i++)
                  {
                    for (j;j<rounds;j++)
                    {
                        for (k;k<rounds;k++)
                        {
                                selectColRow(M[i][k]);
                                it3 = L_table[fh][lh];
                                selectColRow(result2[k][j]);
                                it5 = L_table[fh][lh];
                                if(fh == 0x00 && lh == 0x00)
                                {
                                        detect = TRUE;
                                }
                                it6 = it5 + it3;

                                if(it6-it5 != it3)
                                {
                                it6++;
                                }

                                selectColRow(it6);
                                if (detect == TRUE)
                                {
                                        it4 ^= 0;
                                        detect = FALSE;
                                } else {
                                        it4 ^= E_table[fh][lh];
                                }
                        }
                        result1[i][j] = it4;
                        k = 0; it3 = 0; it4 = 0;
                    }
                    j = 0;
                 }

               //COPIA MATRIX
               initVar();
               for(i;i < rounds;i++)
               {
                  for(j;j < rounds_col;j++)
                  {
                      result2[i][j] = result1[i][j];
                  }
                  j = 0;
               }
          }

          //XOR CIFRADO FINAL
          initVar();
         do
         {
           do
           {
               result_f[j][i] = result2[j][i] ^ key[j][col_key];
               j++;
           } while (j < rounds_col);
           j = 0; col_key++; i++;
         } while(i < rounds);
     }
    call AMControl.start();
  }

 event void AMControl.startDone(error_t err) {
   if (err == SUCCESS) {
     call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
```

```
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
    if (!busy) {
      EncryptRadioECBMsg* btrpkt =
        (EncryptRadioECBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioECBMsg)));
      if (btrpkt == NULL) {
        return;
      }
      initVar();
          for (i;i < sizeof(basedata); i++)
          {
                btrpkt->data[i] = result_f[j][k];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
          }
      if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(EncryptRadioECBMsg)) == SUCCESS) {
        busy = TRUE;
      }
    }
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioECBMsg)) {
      EncryptRadioECBMsg* btrpkt = (EncryptRadioECBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Annexed V: DecryptRadioECBC.nc Code

```
// $Id: DecryptRadioECBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "DecryptRadioECB.h"
#include "printf.h"

module DecryptRadioECBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;

        uint8_t basedata [4][4];
        uint8_t M_inv [4][4]= {{0x0E, 0x0B, 0x0D, 0x09}, {0x09, 0x0E, 0x0B, 0x0D}, {0x0D, 0x09,
0x0E, 0x0B}, {0x0B, 0x0D, 0x09, 0x0E}};
        uint8_t key_original [4][4]= {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t n = 10;
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W [4];
        uint8_t W_t [4];
        uint8_t W_tmp [4];
        uint8_t R[15][4]= {{0x01, 0x00, 0x00, 0x00}, {0x02, 0x00, 0x00, 0x00}, {0x04, 0x00,
0x00, 0x00}, {0x08, 0x00, 0x00, 0x00}, {0x10, 0x00, 0x00, 0x00}, {0x20, 0x00, 0x00, 0x00},
{0x40, 0x00, 0x00, 0x00}, {0x80, 0x00, 0x00, 0x00}, {0x1B, 0x00, 0x00, 0x00}, {0x36, 0x00,
0x00, 0x00}, {0x6C, 0x00, 0x00, 0x00}, {0xD8, 0x00, 0x00, 0x00}, {0xAB, 0x00, 0x00, 0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        uint8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 40;
        uint8_t b0;
        uint8_t rounds;
        uint8_t rounds_col;
        bool detect = FALSE;

        uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
```

```
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

        uint8_t table_inv [16][16] = {{0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF,
0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB}, {0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87,
0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB}, {0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23,
0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E}, {0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9,
0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25}, {0x72, 0xF8, 0xF6, 0x64, 0x86,
0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92}, {0x6C, 0x70, 0x48, 0x50,
0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84}, {0x90, 0xD8, 0xAB,
0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06}, {0xD0, 0x2C,
0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B}, {0x3A,
0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73},
{0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF,
0x6E}, {0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18,
0xBE, 0x1B}, {0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78,
0xCD, 0x5A, 0xF4}, {0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59,
0x27, 0x80, 0xEC, 0x5F}, {0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A,
0x9F, 0x93, 0xC9, 0x9C, 0xEF}, {0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB,
0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61}, {0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1,
0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D}};

uint8_t E_table [16][16] = {{0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A, 0x2E, 0x72,
0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4, 0xF7, 0x02,
0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB, 0x26, 0x6A,
0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C, 0x44, 0xCC,
0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0, 0x3B, 0x4D,
0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0, 0x6B, 0xBD,
0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57, 0xF9, 0x10,
0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19, 0x2B, 0x7D,
0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB, 0x16, 0x3A,
0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41}, {0xC3, 0x5E,
0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA, 0x75}, {0x9F,
0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E, 0x89, 0x80},
{0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8, 0x43, 0xC5,
0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99, 0xB0, 0xCB,
0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E, 0x42, 0xC6,
0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A, 0x85, 0x94,
0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C, 0x24, 0x6C,
0xB4, 0xC7, 0x52, 0xF6, 0x01}};

        uint8_t L_table [16][16] = {{0x00, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};

        void initVar()
        {
                rounds_col = sizeof(basedata[0]);
```

```
                rounds = sizeof(basedata)/rounds_col;
                i = 0;
                j = 0;
                k = 0;
                it2 = 0;
                it3 = 0;
                it4 = 0;
                it5 = 0;
                it6 = 0;
        }

        void          printMatrix(uint8_t          matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%#010x , ", matrix[i][j]);
                                j++;
                        } while(j<rounds_col);
                        printf("\n");
                        j = 0;
                        i++;
                } while (i<rounds);
                printfflush();
        }

        void printVector(uint8_t matrix[sizeof(W)])
        {
                initVar();
                for (j; j < sizeof(W); j++)
                {
                        printf("%#010x , ", matrix[j]);
                }
                printf("\n");
                printfflush();
        }

        void printKey(uint8_t matrix[sizeof(key)/sizeof(key[0])][sizeof(key[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%d , ", matrix[i][j]);
                                j++;
                        } while(j<sizeof(key[0]));
                        printf("\n");
                        j = 0;
                        i++;
                } while (i<sizeof(key)/sizeof(key[0]));
                printfflush();
        }

        void selectColRow(uint8_t data)
        {
                fh = data & fh_ex;      //AND 4bits mayores
                lh = data & lh_ex;      //AND 4bits menores
                fh = fh >> 4;           //desplazamiento 4 bits derecha
        }

        void         rotacionMatrix(uint8_t        matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        { //SALIDA result2
           do
           {
                if (i == 0)
                {
                        j = 0;
                        do {
                          result2[i][j]=matrix[i][j];
```

```
                          j++;
                        } while (j<sizeof(basedata[0]));
                        j = 0;
                } else{
                //datos en forma de matriz
                do
                {
                   it2 = j - i;// round_col - round;
                   if (it2 > 0) //perfecto
                   {
                        result2[i][it2] = matrix[i][j];
                   } else if (it2 == 0) {
                        result2[i][0] = matrix[i][j];
                   } else if (it2 < 0){
                        it3 = rounds_col + it2;
                        result2[i][it3] = matrix[i][j];
                   }
                   j++;
                } while (j < rounds_col);
                j = 0;
             }
             i++;
           } while (i < rounds);
        }

        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }

 void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  event void Boot.booted() {
      //INICIALIZACIÓN DE VARIABLES
        initVar();

        //GENERACIÓN DE SUBMATRICECS
        //for (i;i<sizeof(W);i++)
        for (i;i<sizeof(key[0]);i++)
        {
                if (i < sizeof(key_original[0]))
                {
                        k = 0;
                        do
                        {
                                key[k][i] = key_original[k][i];
                                k++;
                        } while (k < sizeof(key_original[0]));
                        } else {
                        k = 0;
                        do
```

```
                         {
                                 W_t [k] = key[k][i-1];
                                 k++;
                         } while(k < sizeof(W_t));

                         k = 0;
                         if (i%sizeof(key_original[0]) == 0)
                         {
                                 it3 = W_t[0];
                                 for (k; k < sizeof(W_t); k++)   // FOR ROTACION
                                 {
                                         W_tmp[k] = W_t[k+1];
                                 }
                                 W_tmp[sizeof(W_t)-1] = it3;

                                 k = 0;
                                 for (k; k < sizeof(W_t); k++)     // FOR SUSTITUCIÓN
                                 {
                                         selectColRow(W_tmp[k]);
                                         W_tmp[k] = table [fh][lh];
                                 }

                                 k = 0;
                                 for (k; k < sizeof(W_t); k++)   // FOR XOR con R
                                 {
                                    W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                                 }
                         }

                         k = 0;
                         for (k; k < sizeof(W_t); k++)           // FOR XOR con R
                         {
                                 W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
                         }

                         k = 0;
                         for (k; k < sizeof(W_t); k++)   // FOR ASIGNACIÓN EN KEY
                         {
                                 key [k][i] = W_tmp[k];
                         }
                 }
         }

    call AMControl.start();
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      //call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }


  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(DecryptRadioECBMsg)) {
      DecryptRadioECBMsg* btrpkt = (DecryptRadioECBMsg*)payload;
        setLeds(0x04);
        col_key = 40;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
```

```
		basedata[j][k] = btrpkt->data[i];
		j++;
		if (j>3)
		{
			j = 0; k++;
		}
	}
	//IMPRIMIR RESULTADO FINAL DESCIFRADO
	printf("Print Received Data: \n");
	printMatrix(basedata);

	//XOR CIFRADO INICIAL
	initVar();
	do
	{
		do
		{
			result2[j][i] = basedata[j][i] ^ key[j][col_key];
			j++;
		} while (j < rounds_col);
		j = 0; i++; col_key++;
	} while(i < rounds);

	//ITERACIONES n = 10
	b0 = 0;
	for(b0;b0 < n;b0++)
	{
		//PROCESO DE SUBSTITUCION  //SALE CON result1
		initVar();
		sustitucionMatrix(result2);

		//PROCESO DE ROTACIÓN A DERECHAS  //SALE CON result2
		initVar();
		rotacionMatrix(result1);

		col_key = col_key - 8;

		//XOR CIFRADO FINAL
		initVar();
		do
		{
			do
			{
				result1[j][i] = result2[j][i] ^ key[j][col_key];
				j++;
			} while (j < rounds_col);
			j = 0; col_key++; i++;
		} while(i < rounds);

		//COPIA MATRIX
		initVar();
		for(i;i < rounds;i++)
		{
			for(j;j < rounds_col;j++)
			{
				result2[i][j] = result1[i][j];
			}
			j = 0;
		}

		//PROCESO MIXCOLUMS  //SALE CON result1
		initVar();
		if ( b0 < n-1)
		{
		    for (i;i<rounds;i++)
		    {
			for (j;j<rounds;j++)
			{
			    for (k;k<rounds;k++)
			    {
				selectColRow(M_inv[i][k]);
				it3 = L_table[fh][lh];
				selectColRow(result1[k][j]);
				it5 = L_table[fh][lh];
```

```
                                if(fh == 0x00 && lh == 0x00)
                                {
                                        detect = TRUE;
                                }

                                it6 = it5 + it3;
                                if(it6-it5 != it3)
                                {
                                    it6++;
                                }

                                selectColRow(it6);
                                if (detect == TRUE)
                                {
                                    it4 ^= 0;
                                    detect = FALSE;
                                } else {
                                    it4 ^= E_table[fh][lh];
                                }
                             }
                             result2[i][j] = it4;
                             k = 0; it3 = 0; it4 = 0;
                        }
                        j = 0;
                     }
                }
            }
            //IMPRIMIR RESULTADO FINAL DESCIFRADO
            printf("Print Decrypted Result: \n");
            printMatrix(result2);
    }
    return msg;
  }
}
```

Annexed VI: EncryptRadioCFBC.nc Code

```
// $Id: EncryptRadioCFBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioCFB.h"

module EncryptRadioCFBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;
        bool detect = FALSE;

        uint8_t                                basedata                                [4][4]=
{{0x11,0x22,0x33,0x44},{0x55,0x66,0x77,0x88},{0x99,0xAA,0xBB,0xCC},{0xDD,0xEE,0xFF,0x00}};
        uint8_t                            text                            [sizeof(basedata)]=
{0x32,0x88,0x31,0xE0,0x43,0x5A,0x31,0x37,0xF6,0x30,0x98,0x07,0xA8,0x8D,0xA2,0x34};
        uint8_t M [4][4]= {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01,
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4]= {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t v_fila [sizeof(basedata)];
        uint8_t v_result [sizeof(basedata)];
        uint8_t n = 10;
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W [4];
        uint8_t W_t [4];
        uint8_t W_tmp [4];
        uint8_t R[15][4]= {{0x01, 0x00, 0x00, 0x00}, {0x02, 0x00, 0x00, 0x00}, {0x04, 0x00,
0x00, 0x00}, {0x08, 0x00, 0x00, 0x00}, {0x10, 0x00, 0x00, 0x00}, {0x20, 0x00, 0x00, 0x00},
{0x40, 0x00, 0x00, 0x00}, {0x80, 0x00, 0x00, 0x00}, {0x1B, 0x00, 0x00, 0x00}, {0x36, 0x00,
0x00, 0x00}, {0x6C, 0x00, 0x00, 0x00}, {0xD8, 0x00, 0x00, 0x00}, {0xAB, 0x00, 0x00, 0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        int8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 0;
        uint8_t b0;
        uint8_t d1 = 0;
        uint8_t d2 = 0;
        uint8_t d3 = 0;
```

```
	uint8_t rounds;
	uint8_t rounds_col;

	uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};
	uint8_t E_table [16][16] = {{0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4,
0xF7, 0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB,
0x26, 0x6A, 0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C,
0x44, 0xCC, 0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0,
0x3B, 0x4D, 0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0,
0x6B, 0xBD, 0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57,
0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19,
0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB,
0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41},
{0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA,
0x75}, {0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E,
0x89, 0x80}, {0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8,
0x43, 0xC5, 0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99,
0xB0, 0xCB, 0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E,
0x42, 0xC6, 0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A,
0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01}};

	uint8_t L_table [16][16] = {{0x00, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};

	void initVar()
	{
		rounds_col = sizeof(basedata[0]);
		rounds = sizeof(basedata)/rounds_col;
		i = 0;
		j = 0;
		k = 0;
		fh = 0;
		lh = 0;
		it2 = 0;
		it3 = 0;
		it4 = 0;
		it5 = 0;
		it6 = 0;
	}
```

```
        void selectColRow(uint8_t data)
        {
                fh = data & fh_ex;      //AND 4bits mayores
                lh = data & lh_ex;      //AND 4bits menores
                fh = fh >> 4;           //desplazamiento 4 bits derecha
        }

        void        rotacionMatrix(uint8_t        matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        { //SALIDA result2
           do
           {
                if (i == 0)
                {
                        j = 0;
                        do {
                          result2[i][j]=matrix[i][j];
                          j++;
                        } while (j<sizeof(basedata[0]));
                        j = 0;
                } else{
                //datos en forma de matriz
                do
                {
                   it2 = j - i;// round_col - round;
                   if (it2 > 0) //perfecto
                   {
                        result2[i][it2] = matrix[i][j];
                   } else if (it2 == 0) {
                        result2[i][0] = matrix[i][j];
                   } else if (it2 < 0){
                        it3 = rounds_col + it2;
                        result2[i][it3] = matrix[i][j];
                   }
                   j++;
                } while (j < rounds_col);
                j = 0;
             }
             i++;
           } while (i < rounds);
        }

        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

   event void Boot.booted() {
```

```
      //INICIALIZACIÓN DE VARIABLES
      initVar();

      //GENERACIÓN DE SUBMATRICECS
      for (i;i<sizeof(key[0]);i++)
      {
         if (i < sizeof(key_original[0]))
         {
              k = 0;
              do
              {
                  key[k][i] = key_original[k][i];
                  k++;
              } while (k < sizeof(key_original[0]));

         } else {
              k = 0;
              do
              {
                  W_t [k] = key[k][i-1];
                  k++;
              } while(k < sizeof(W_t));

              k = 0;
              if (i%sizeof(key_original[0]) == 0)
              {
                  it3 = W_t[0];
                  for (k; k < sizeof(W_t); k++)  // FOR ROTACION
                  {
                      W_tmp[k] = W_t[k+1];
                  }
                  W_tmp[sizeof(W_t)-1] = it3;

                  k = 0;
                  for (k; k < sizeof(W_t); k++)   // FOR SUSTITUCIÓN
                  {
                      selectColRow(W_tmp[k]);
                      W_tmp[k] = table [fh][lh];
                  }

                  k = 0;
                  for (k; k < sizeof(W_t); k++) // FOR XOR con R
                  {
                      W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                  }
              }

              k = 0;
              for (k; k < sizeof(W_t); k++)   // FOR XOR con R
              {
                  W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
              }

              k = 0;
              for (k; k < sizeof(W_t); k++)           // FOR ASIGNACIÓN EN KEY
              {
                  key [k][i] = W_tmp[k];
              }

          }
      }
      call AMControl.start();
   }

   event void AMControl.startDone(error_t err) {
      if (err == SUCCESS) {
          call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
      }
      else {
          call AMControl.start();
      }
   }

   event void AMControl.stopDone(error_t err) {
```

```
    }

    event void Timer0.fired() {
       if (!busy) {
            EncryptRadioCFBMsg* btrpkt =
            (EncryptRadioCFBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioCFBMsg)));
            if (btrpkt == NULL) {
               return;
            }

            //XOR CIFRADO INICIAL
            initVar();
            col_key = 0;

            do
            {
                   do
                   {
                       result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                       j++;
                   } while (j < rounds_col);
                   j = 0; i++; col_key++;
            } while(i < rounds);


            //ITERACIONES n = 10 [0-9]
            b0 = 0;
            for(b0;b0 < n;b0++)
            {
               //PROCESO DE SUSTITUCION  //SALE CON result1
               initVar();
               sustitucionMatrix(result_f);

               //PROCESO DE ROTACIÓN     //SALE CON result2
               initVar();
               rotacionMatrix(result1);

               //PROCESO MIXCOLUMS     //SALE CON result1
               initVar();
               if ( b0 < n-1)
               {
                   for (i;i<rounds;i++)
                   {
                       for (j;j<rounds;j++)
                       {
                            for (k;k<rounds;k++)
                            {
                               selectColRow(M[i][k]);
                               it3 = L_table[fh][lh];

                               selectColRow(result2[k][j]);
                               it5 = L_table[fh][lh];
                               if(fh == 0x00 && lh == 0x00)
                               {
                                       detect = TRUE;
                               }

                               it6 = it5 + it3;
                               if(it6-it5 != it3)
                               {
                                       it6++;
                               }

                               selectColRow(it6);
                               if (detect == TRUE)
                               {
                                   it4 ^= 0;
                                   detect = FALSE;
                               } else {
                                   it4 ^= E_table[fh][lh];
                               }
                            }
                            result1[i][j] = it4;
                            k = 0; it3 = 0; it4 = 0;
```

```
                        }
                      j = 0;
                   }

                   //COPIA MATRIX
                   initVar();
                   for(i;i < rounds;i++)
                   {
                      for(j;j < rounds_col;j++)
                      {
                              result2[i][j] = result1[i][j];
                      }
                      j = 0;
                   }
              }

              //XOR CIFRADO FINAL
              initVar();
              do
              {
                  do
                  {
                      result_f[j][i] = result2[j][i] ^ key[j][col_key];
                      j++;
                  } while (j < rounds_col);
                  j = 0; col_key++; i++;
              } while(i < rounds);
          }

          initVar();
          for (i;i < sizeof(basedata); i++)
          {
              v_fila[i] = basedata[j][k];
              j++;

              if (j>3)
              {
                   j = 0; k++;
              }
           }
           it3 = result_f[0][0] ^ text[d3];
           d3++;

           i = 0;
           for (i;i < sizeof(basedata)-1; i++)
           {
              v_fila[i] = v_fila[i+1];
           }

           v_fila[sizeof(basedata)-1] = it3;
           v_result[d1] = it3;
           d1++;

           initVar();
           for (i;i < sizeof(basedata); i++)
           {
              basedata[j][k] = v_fila[i];
              j++;

              if (j>3)
              {
                   j = 0; k++;
              }
           }

           btrpkt->data = v_result[d2];
           d2++;

           if  (call  AMSend.send(AM_BROADCAST_ADDR,  &pkt,  sizeof(EncryptRadioCFBMsg))  ==
SUCCESS) {
              busy = TRUE;
           }
      }
  }
```

```
  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioCFBMsg)) {
      EncryptRadioCFBMsg* btrpkt = (EncryptRadioCFBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Annexed VII: DecryptRadioCFBC.nc Code

```
// $Id: EncryptRadioCFBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */
#include <Timer.h>
#include "DecryptRadioCFB.h"
#include "printf.h"

module DecryptRadioCFBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;
        bool detect = FALSE;

        uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99,
0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01,
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3 ,0xA2, 0xA1, 0xA0}};
        uint8_t v_fila [sizeof(basedata)];
        uint8_t v_result [sizeof(basedata)];
        uint8_t n = 10;
        uint8_t n2 = sizeof(basedata);
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W [4];
        uint8_t W_t [4];
        uint8_t R[15][4]= {{0x01, 0x00, 0x00, 0x00}, {0x02, 0x00, 0x00, 0x00}, {0x04, 0x00,
0x00, 0x00}, {0x08, 0x00, 0x00, 0x00}, {0x10, 0x00, 0x00, 0x00}, {0x20, 0x00, 0x00, 0x00},
{0x40, 0x00, 0x00, 0x00}, {0x80, 0x00, 0x00, 0x00}, {0x1B, 0x00, 0x00, 0x00}, {0x36, 0x00,
0x00, 0x00}, {0x6C, 0x00, 0x00, 0x00}, {0xD8, 0x00, 0x00, 0x00}, {0xAB, 0x00, 0x00, 0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        int8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 0;
        uint8_t b0;
        uint8_t b1;
        uint8_t d1 = 0;
        uint8_t d2 = 0;
        uint8_t d3 = 0;
        uint8_t rounds;
```

```
        uint8_t rounds_col;


        uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

        uint8_t E_table [16][16] = {{0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4,
0xF7, 0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB,
0x26, 0x6A, 0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C,
0x44, 0xCC, 0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0,
0x3B, 0x4D, 0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0,
0x6B, 0xBD, 0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57,
0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19,
0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB,
0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41},
{0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA,
0x75}, {0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E,
0x89, 0x80}, {0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8,
0x43, 0xC5, 0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99,
0xB0, 0xCB, 0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E,
0x42, 0xC6, 0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A,
0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01}};

        uint8_t L_table [16][16] = {{0x00, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};

        void initVar()
        {
                rounds_col = sizeof(basedata[0]);
                rounds = sizeof(basedata)/rounds_col;
                i = 0;
                j = 0;
                k = 0;
                fh = 0;
                lh = 0;
                it2 = 0;
                it3 = 0;
                it4 = 0;
                it5 = 0;
                it6 = 0;
```

```
        }

        void selectColRow(uint8_t data)
        {
                fh = data & fh_ex;      //AND 4bits mayores
                lh = data & lh_ex;      //AND 4bits menores
                fh = fh >> 4;           //desplazamiento 4 bits derecha
        }

        void        rotacionMatrix(uint8_t        matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        { //SALIDA result2
           do
           {
                if (i == 0)
                {
                        j = 0;
                        do {
                          result2[i][j]=matrix[i][j];
                          j++;
                        } while (j<sizeof(basedata[0]));
                        j = 0;
                } else{
                //datos en forma de matriz
                do
                {
                   it2 = j - i;// round_col - round;
                   if (it2 > 0) //perfecto
                   {
                        result2[i][it2] = matrix[i][j];
                   } else if (it2 == 0) {
                        result2[i][0] = matrix[i][j];
                   } else if (it2 < 0){
                        it3 = rounds_col + it2;
                        result2[i][it3] = matrix[i][j];
                   }
                   j++;
                } while (j < rounds_col);
                j = 0;
             }
             i++;
           } while (i < rounds);
        }

        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }


        void          printMatrix(uint8_t          matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%#010x , ", matrix[i][j]);
                                j++;
                        } while(j<rounds_col);
                        printf("\n");
                        j = 0;
                        i++;
```

```
                } while (i<rounds);
                printfflush();
        }

        void printKey(uint8_t matrix[sizeof(key)/sizeof(key[0])][sizeof(key[0])])
        {
                initVar();
                do
                {
                        do
                        {
                                printf("%d , ", matrix[i][j]);
                                j++;
                        } while(j<sizeof(key[0]));
                        printf("\n");
                        j = 0;
                        i++;
                } while (i<sizeof(key)/sizeof(key[0]));
                printfflush();
        }

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }
  event void Boot.booted() {
        //INICIALIZACIÓN DE VARIABLES
        initVar();

        //GENERACIÓN DE SUBMATRICECS
        for (i;i<sizeof(key[0]);i++)
        {
           if (i < sizeof(key_original[0]))
           {
                k = 0;
                do
                {
                    key[k][i] = key_original[k][i];
                    k++;
                } while (k < sizeof(key_original[0]));

           } else {
                k = 0;
                do
                {
                    W_t [k] = key[k][i-1];
                    k++;
                } while(k < sizeof(W_t));

                k = 0;
                if (i%sizeof(key_original[0]) == 0)
                {
                    it3 = W_t[0];
                    for (k; k < sizeof(W_t); k++)  // FOR ROTACION
                    {
                        W_tmp[k] = W_t[k+1];
                    }
                    W_tmp[sizeof(W_t)-1] = it3;

                    k = 0;
                    for (k; k < sizeof(W_t); k++)   // FOR SUSTITUCIÓN
                    {
                        selectColRow(W_tmp[k]);
                        W_tmp[k] = table [fh][lh];
```

```
                    }

                    k = 0;
                    for (k; k < sizeof(W_t); k++) // FOR XOR con R
                    {
                        W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                    }
                }

                k = 0;
                for (k; k < sizeof(W_t); k++)   // FOR XOR con R
                {
                    W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
                }

                k = 0;
                for (k; k < sizeof(W_t); k++)           // FOR ASIGNACIÓN EN KEY
                {
                    key [k][i] = W_tmp[k];
                }

            }
        }
        call AMControl.start();
     }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(DecryptRadioCFBMsg)) {
      DecryptRadioCFBMsg* btrpkt = (DecryptRadioCFBMsg*)payload;
      setLeds(0x04);
      d3 = btrpkt->data;
      printf("Cipherdata received: %#010x | ",d3);
      printfflush();

        //XOR CIFRADO INICIAL
        initVar();
        col_key = 0;

        do
        {
            do
            {
                result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                j++;
            } while (j < rounds_col);
            j = 0; i++; col_key++;
        } while(i < rounds);

        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
            //PROCESO DE SUSTITUCION    //SALE CON result1
```

```
            initVar();
            sustitucionMatrix(result_f);

            //PROCESO DE ROTACIÓN        //SALE CON result2
            initVar();
            rotacionMatrix(result1);

            //PROCESO MIXCOLUMS          //SALE CON result1
            initVar();
            if ( b0 < n-1)
            {
                for (i;i<rounds;i++)
                {
                    for (j;j<rounds;j++)
                    {
                        for (k;k<rounds;k++)
                        {
                             selectColRow(M[i][k]);
                             it3 = L_table[fh][lh];

                             selectColRow(result2[k][j]);
                             it5 = L_table[fh][lh];
                             if(fh == 0x00 && lh == 0x00)
                             {
                                detect = TRUE;
                             }

                            it6 = it5 + it3;

                            if(it6-it5 != it3)
                            {
                                it6++;
                            }

                            selectColRow(it6);
                            if (detect == TRUE)
                            {
                                it4 ^= 0;
                                detect = FALSE;
                            } else {
                                it4 ^= E_table[fh][lh];
                            }
                        }
                        result1[i][j] = it4;
                        k = 0; it3 = 0; it4 = 0;
                    }
                    j = 0;
                }

                //COPIA MATRIX
                initVar();
                for(i;i < rounds;i++)
                {
                     for(j;j < rounds_col;j++)
                     {
                        result2[i][j] = result1[i][j];
                     }
                     j = 0;
                }
            }

           //XOR CIFRADO FINAL
           initVar();
           do
           {
                do
                {
                        result_f[j][i] = result2[j][i] ^ key[j][col_key];
                        j++;
                } while (j < rounds_col);
                j = 0; col_key++; i++;
           } while(i < rounds);
        }
```

```
        initVar();
        for (i;i < sizeof(basedata); i++)
        {
                v_fila[i] = basedata[j][k];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
        }

        it3 = result_f[0][0] ^ d3;
        printf("DecryptData received: %#010x\n",it3);
        printfflush();

        i = 0;
        for (i;i < sizeof(basedata)-1; i++)
        {
                v_fila[i] = v_fila[i+1];
        }
        v_fila[sizeof(basedata)-1] = d3;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
                basedata[j][k] = v_fila[i];
                j++;
                if (j>3)
                {
                        j = 0; k++;
                }
        }
    }
    return msg;
  }
}
```

Annexed VIII: EncryptRadioOFBC.nc Code

```
// $Id: EncryptRadioOFBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "EncryptRadioOFB.h"
#include "printf.h"

module EncryptRadioOFBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;
        bool detect = FALSE;

        uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99,
0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
        uint8_t text [sizeof(basedata)] = {0x32, 0x88, 0x31, 0xE0, 0x43, 0x5A, 0x31, 0x37,
0xF6, 0x30, 0x98, 0x07, 0xA8, 0x8D, 0xA2, 0x34};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01,
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t v_fila [sizeof(basedata)];
        uint8_t n = 10;
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W [4];
        uint8_t W_t [4];
        uint8_t W_tmp [4];
        uint8_t R[15][4]= {{0x01, 0x00, 0x00, 0x00}, {0x02, 0x00, 0x00, 0x00}, {0x04, 0x00,
0x00, 0x00}, {0x08, 0x00, 0x00, 0x00}, {0x10, 0x00, 0x00, 0x00}, {0x20, 0x00, 0x00, 0x00},
{0x40, 0x00, 0x00, 0x00}, {0x80, 0x00, 0x00, 0x00}, {0x1B, 0x00, 0x00, 0x00}, {0x36, 0x00,
0x00, 0x00}, {0x6C, 0x00, 0x00, 0x00}, {0xD8, 0x00, 0x00, 0x00}, {0xAB, 0x00, 0x00, 0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        int8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 0;
        uint8_t b0;
        uint8_t d1 = 0;
        uint8_t d2 = 0;
        uint8_t d3 = 0;
```

```
	uint8_t rounds;
	uint8_t rounds_col;

	uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

	uint8_t  E_table  [16][16]  =  {{0x01,  0x03,  0x05,  0x0F,  0x11,  0x33,  0x55,  0xFF,  0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4,
0xF7, 0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB,
0x26, 0x6A, 0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C,
0x44, 0xCC, 0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0,
0x3B, 0x4D, 0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0,
0x6B, 0xBD, 0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57,
0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19,
0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB,
0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41},
{0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA,
0x75}, {0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E,
0x89, 0x80}, {0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8,
0x43, 0xC5, 0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99,
0xB0, 0xCB, 0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E,
0x42, 0xC6, 0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A,
0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01}};

	uint8_t  L_table  [16][16]  =  {{0x00,  0x00,  0x19,  0x01,  0x32,  0x02,  0x1A,  0xC6,  0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};
	void initVar()
	{
		rounds_col = sizeof(basedata[0]);
		rounds = sizeof(basedata)/rounds_col;
		i = 0;
		j = 0;
		k = 0;
		fh = 0;
		lh = 0;
		it2 = 0;
		it3 = 0;
		it4 = 0;
		it5 = 0;
		it6 = 0;
	}
```

```
        void selectColRow(uint8_t data)
        {
                fh = data & fh_ex;      //AND 4bits mayores
                lh = data & lh_ex;      //AND 4bits menores
                fh = fh >> 4;           //desplazamiento 4 bits derecha
        }

        void        rotacionMatrix(uint8_t        matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        { //SALIDA result2
           do
           {
                if (i == 0)
                {
                        j = 0;
                        do {
                          result2[i][j]=matrix[i][j];
                          j++;
                        } while (j<sizeof(basedata[0]));
                        j = 0;
                } else{
                //datos en forma de matriz
                do
                {
                   it2 = j - i;// round_col - round;
                   if (it2 > 0) //perfecto
                   {
                        result2[i][it2] = matrix[i][j];
                   } else if (it2 == 0) {
                        result2[i][0] = matrix[i][j];
                   } else if (it2 < 0){
                        it3 = rounds_col + it2;
                        result2[i][it3] = matrix[i][j];
                   }
                   j++;
                } while (j < rounds_col);
                j = 0;
             }
             i++;
           } while (i < rounds);
        }

        void       sustitucionMatrix(uint8_t       matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
        {
                //PROCESO SUSTITUCIÓN
                do {
                        do {
                        selectColRow(matrix[i][j]);
                        result1[i][j] = table [fh][lh];
                        j++;
                        } while (j<rounds_col);
                j = 0;
                i++;
                } while (i<rounds);
        }

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

        void          printMatrix(uint8_t          matrix[sizeof(basedata)/sizeof(basedata[0])]
```

```
[sizeof(basedata[0])])
        {
            initVar();
            do
            {
                do
                {
                    printf("%#010x , ", matrix[i][j]);
                    j++;
                } while(j<rounds_col);
                printf("\n");
                j = 0;
                i++;
            } while (i<rounds);
            printfflush();
        }

        void printKey(uint8_t matrix[sizeof(key)/sizeof(key[0])][sizeof(key[0])])
        {
             initVar();
             do
             {
                do
                {
                     printf("%d , ", matrix[i][j]);
                     j++;
                } while(j<sizeof(key[0]));
                printf("\n");
                j = 0;
                i++;
             } while (i<sizeof(key)/sizeof(key[0]));
             printfflush();
        }

  event void Boot.booted() {
        //INICIALIZACIÓN DE VARIABLES
        initVar();

        //GENERACIÓN DE SUBMATRICECS
        for (i;i<sizeof(key[0]);i++)
        {
           if (i < sizeof(key_original[0]))
           {
                k = 0;
                do
                {
                    key[k][i] = key_original[k][i];
                    k++;
                } while (k < sizeof(key_original[0]));

           } else {
                k = 0;
                do
                {
                    W_t [k] = key[k][i-1];
                    k++;
                } while(k < sizeof(W_t));

                k = 0;
                if (i%sizeof(key_original[0]) == 0)
                {
                    it3 = W_t[0];
                    for (k; k < sizeof(W_t); k++)  // FOR ROTACION
                    {
                        W_tmp[k] = W_t[k+1];
                    }
                    W_tmp[sizeof(W_t)-1] = it3;

                    k = 0;
                    for (k; k < sizeof(W_t); k++)   // FOR SUSTITUCIÓN
                    {
                        selectColRow(W_tmp[k]);
                        W_tmp[k] = table [fh][lh];
                    }
```

```
                k = 0;
                for (k; k < sizeof(W_t); k++) // FOR XOR con R
                {
                    W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                }
            }

            k = 0;
            for (k; k < sizeof(W_t); k++)   // FOR XOR con R
            {
                W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
            }

            k = 0;
            for (k; k < sizeof(W_t); k++)           // FOR ASIGNACIÓN EN KEY
            {
                key [k][i] = W_tmp[k];
            }

        }
      }
      call AMControl.start();
   }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
    if (!busy) {
      EncryptRadioOFBMsg* btrpkt =
        (EncryptRadioOFBMsg*)(call Packet.getPayload(&pkt, sizeof(EncryptRadioOFBMsg)));
      if (btrpkt == NULL) {
        return;
      }

        //XOR CIFRADO INICIAL
        initVar();
        col_key = 0;
        do
        {
           do
           {
                result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                j++;
           } while (j < rounds_col);
           j = 0; i++; col_key++;
        } while(i < rounds);


        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
           //PROCESO DE SUSTITUCION     //SALE CON result1
           initVar();
           sustitucionMatrix(result_f);

           //PROCESO DE ROTACIÓN    //SALE CON result2
           initVar();
           rotacionMatrix(result1);

           //PROCESO MIXCOLUMS          //SALE CON result1
           initVar();
           if ( b0 < n-1)
```

```
            {
                for (i;i<rounds;i++)
                {
                    for (j;j<rounds;j++)
                    {
                        for (k;k<rounds;k++)
                        {
                            selectColRow(M[i][k]);
                            it3 = L_table[fh][lh];

                            selectColRow(result2[k][j]);
                            it5 = L_table[fh][lh];
                            if(fh == 0x00 && lh == 0x00)
                            {
                                detect = TRUE;
                            }

                            it6 = it5 + it3;
                            if(it6-it5 != it3)
                            {
                                it6++;
                            }

                            selectColRow(it6);
                            if (detect == TRUE)
                            {
                                it4 ^= 0;
                                detect = FALSE;
                            } else {
                                it4 ^= E_table[fh][lh];
                            }
                        }
                        result1[i][j] = it4;
                        k = 0; it3 = 0; it4 = 0;
                    }
                    j = 0;
                }

                //COPIA MATRIX
                initVar();
                for(i;i < rounds;i++)
                {
                    for(j;j < rounds_col;j++)
                    {
                        result2[i][j] = result1[i][j];
                    }
                    j = 0;
                }
            }

            //XOR CIFRADO FINAL
            initVar();
            do
            {
               do
               {
                     result_f[j][i] = result2[j][i] ^ key[j][col_key];
                     j++;
               } while (j < rounds_col);
               j = 0; col_key++; i++;
            } while(i < rounds);
        }

        initVar();
        printf("Imprimir matriz tras cifrado\n");
        printMatrix(result_f);

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
              v_fila[i] = basedata[j][k];
              j++;

              if (j>3)
```

```
              {
                   j = 0; k++;
              }
        }

        it3 = result_f[0][0] ^ text[d3];
        d3++;

        i = 0;
        for (i;i < sizeof(basedata)-1; i++)
        {
             v_fila[i] = v_fila[i+1];
        }
        v_fila[sizeof(basedata)-1] = result_f[0][0];
        btrpkt->data = it3;

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
             basedata[j][k] = v_fila[i];
             j++;

             if (j>3)
             {
                  j = 0; k++;
             }
        }

        if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(EncryptRadioOFBMsg)) == SUCCESS) {
        busy = TRUE;
        }
     }
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(EncryptRadioOFBMsg)) {
      EncryptRadioOFBMsg* btrpkt = (EncryptRadioOFBMsg*)payload;
      setLeds(0x04);
    }
    return msg;
  }
}
```

Annexed IX: DecryptRadioOFBC.nc Code

```
// $Id: DecryptRadioOFBC.nc,v 1.6 2016-06-04 22:15:00 scipio Exp $

/*
 * @authors Martinez-Caro J.-M, Cano M.-D
 * @date   Jun 4, 2016
 */

#include <Timer.h>
#include "DecryptRadioOFB.h"
#include "printf.h"

module DecryptRadioOFBC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
}
implementation {

        message_t pkt;
        bool busy = FALSE;
        bool detect = FALSE;

        uint8_t basedata [4][4] = {{0x11, 0x22, 0x33, 0x44}, {0x55, 0x66, 0x77, 0x88}, {0x99,
0xAA, 0xBB, 0xCC}, {0xDD, 0xEE, 0xFF, 0x00}};
        uint8_t M [4][4] = {{0x02, 0x03, 0x01, 0x01}, {0x01, 0x02, 0x03, 0x01}, {0x01, 0x01,
0x02, 0x03}, {0x03, 0x01, 0x01, 0x02}};
        uint8_t key_original [4][4] = {{0xAF, 0xAE, 0xAD, 0xAC}, {0xAB, 0xAA, 0xA9, 0xA8},
{0xA7, 0xA6, 0xA5, 0xA4}, {0xA3, 0xA2, 0xA1, 0xA0}};
        uint8_t v_fila [sizeof(basedata)];
        uint8_t v_result [sizeof(basedata)];
        uint8_t n = 10;
        uint8_t n2 = sizeof(basedata);
        uint8_t key [sizeof(basedata)/sizeof(basedata[0])][11*sizeof(basedata[0])];
        uint8_t result1 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result2 [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t result_f [sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])];
        uint8_t W [4];
        uint8_t W_t [4];
        uint8_t W_tmp [4];
        uint8_t R[15][4]= {{0x01, 0x00, 0x00, 0x00}, {0x02, 0x00, 0x00, 0x00}, {0x04, 0x00,
0x00, 0x00}, {0x08, 0x00, 0x00, 0x00}, {0x10, 0x00, 0x00, 0x00}, {0x20, 0x00, 0x00, 0x00},
{0x40, 0x00, 0x00, 0x00}, {0x80, 0x00, 0x00, 0x00}, {0x1B, 0x00, 0x00, 0x00}, {0x36, 0x00,
0x00, 0x00}, {0x6C, 0x00, 0x00, 0x00}, {0xD8, 0x00, 0x00, 0x00}, {0xAB, 0x00, 0x00, 0x00},
{0x4D, 0x00, 0x00, 0x00}, {0x9A, 0x00, 0x00, 0x00}};
        uint8_t i;
        uint8_t j;
        uint8_t k;
        uint8_t fh;
        uint8_t lh;
        uint8_t fh_ex = 0xF0;
        uint8_t lh_ex = 0xF;
        int8_t it2;
        uint8_t it3;
        uint8_t it4;
        uint8_t it5;
        uint8_t it6;
        uint8_t col_key = 0;
        uint8_t b0;
        uint8_t b1;
        uint8_t d1 = 0;
        uint8_t d2 = 0;
        uint8_t d3 = 0;
        uint8_t rounds;
        uint8_t rounds_col;
```

```
	uint8_t table [16][16] = {{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76}, {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0}, {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15}, {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05,
0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75}, {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E,
0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84}, {0x53, 0xD1, 0x00, 0xED, 0x20,
0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF}, {0xD0, 0xEF, 0xAA, 0xFB,
0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8}, {0x51, 0xA3, 0x40,
0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2}, {0xCD, 0x0C,
0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73}, {0x60,
0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},
{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4,
0x79}, {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A,
0xAE, 0x08}, {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A}, {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
0x86, 0xC1, 0x1D, 0x9E}, {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF}, {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

	uint8_t E_table [16][16] = {{0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35}, {0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4,
0xF7, 0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA}, {0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB,
0x26, 0x6A, 0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31}, {0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C,
0x44, 0xCC, 0x4F, 0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD}, {0x4C, 0xD4, 0x67, 0xA9, 0xE0,
0x3B, 0x4D, 0xD7, 0x62, 0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88}, {0x83, 0x9E, 0xB9, 0xD0,
0x6B, 0xBD, 0xDC, 0x7F, 0x81, 0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A}, {0xB5, 0xC4, 0x57,
0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B, 0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3}, {0xFE, 0x19,
0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F, 0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0}, {0xFB,
0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D, 0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41},
{0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B, 0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA,
0x75}, {0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82, 0x9D, 0xBC, 0xDF, 0x7A, 0x8E,
0x89, 0x80}, {0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA, 0x25, 0x6F, 0xB1, 0xC8,
0x43, 0xC5, 0x54}, {0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B, 0x2D, 0x77, 0x99,
0xB0, 0xCB, 0x46, 0xCA}, {0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8, 0xE3, 0x3E,
0x42, 0xC6, 0x51, 0xF3, 0x0E}, {0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F, 0x8A,
0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17}, {0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01}};

	uint8_t L_table [16][16] = {{0x00, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03}, {0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF,
0x4C, 0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1}, {0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27,
0x6A, 0x4D, 0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78}, {0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F,
0xE1, 0x24, 0x12, 0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E}, {0x96, 0x8F, 0xDB, 0xBD, 0x36,
0xD0, 0xCE, 0x94, 0x13, 0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38}, {0x66, 0xDD, 0xFD, 0x30,
0xBF, 0x06, 0x8B, 0x62, 0xB3, 0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10}, {0x7E, 0x6E, 0x48,
0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A, 0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA}, {0x2B, 0x79,
0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E, 0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57}, {0xAF,
0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F, 0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8},
{0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59, 0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51,
0xA0}, {0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8, 0x43, 0x1F, 0x2D, 0xA4, 0x76,
0x7B, 0xB7}, {0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B, 0x52, 0xA1, 0x6C, 0xAA,
0x55, 0x29, 0x9D}, {0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC, 0x95, 0xCF, 0xCD,
0x37, 0x3F, 0x5B, 0xD1}, {0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14, 0x2A, 0x9E,
0x5D, 0x56, 0xF2, 0xD3, 0xAB}, {0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4, 0x7C,
0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5}, {0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07}};

	void initVar()
	{
		rounds_col = sizeof(basedata[0]);
		rounds = sizeof(basedata)/rounds_col;
		i = 0;
		j = 0;
		k = 0;
		fh = 0;
		lh = 0;
		it2 = 0;
		it3 = 0;
		it4 = 0;
		it5 = 0;
		it6 = 0;
	}
```

```
	void selectColRow(uint8_t data)
	{
		fh = data & fh_ex;	//AND 4bits mayores
		lh = data & lh_ex;	//AND 4bits menores
		fh = fh >> 4;		//desplazamiento 4 bits derecha
	}

	void	rotacionMatrix(uint8_t	matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
	{ //SALIDA result2
	   do
	   {
		if (i == 0)
		{
			j = 0;
			do {
			  result2[i][j]=matrix[i][j];
			  j++;
			} while (j<sizeof(basedata[0]));
			j = 0;
		} else{
		//datos en forma de matriz
		do
		{
		   it2 = j - i;// round_col - round;
		   if (it2 > 0) //perfecto
		   {
			result2[i][it2] = matrix[i][j];
		   } else if (it2 == 0) {
			result2[i][0] = matrix[i][j];
		   } else if (it2 < 0){
			it3 = rounds_col + it2;
			result2[i][it3] = matrix[i][j];
		   }
		   j++;
		} while (j < rounds_col);
		j = 0;
	     }
	     i++;
	   } while (i < rounds);
	}

	void	sustitucionMatrix(uint8_t	matrix[sizeof(basedata)/sizeof(basedata[0])]
[sizeof(basedata[0])])
	{
		//PROCESO SUSTITUCIÓN
		do {
			do {
			selectColRow(matrix[i][j]);
			result1[i][j] = table [fh][lh];
			j++;
			} while (j<rounds_col);
		j = 0;
		i++;
		} while (i<rounds);
	}

	void								printMatrix(uint8_t
matrix[sizeof(basedata)/sizeof(basedata[0])][sizeof(basedata[0])])
	{
		initVar();
		do
		{
			do
			{
				printf("%#010x , ", matrix[i][j]);
				j++;
			} while(j<rounds_col);
			printf("\n");
			j = 0;
			i++;
		} while (i<rounds);
		printfflush();
```

```
	}

	void printKey(uint8_t matrix[sizeof(key)/sizeof(key[0])][sizeof(key[0])])
	{
		initVar();
		do
		{
			do
			{
				printf("%d , ", matrix[i][j]);
				j++;
			} while(j<sizeof(key[0]));
			printf("\n");
			j = 0;
			i++;
		} while (i<sizeof(key)/sizeof(key[0]));
		printfflush();
	}

  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  event void Boot.booted() {
	//INICIALIZACIÓN DE VARIABLES
	initVar();

	//GENERACIÓN DE SUBMATRICECS
	for (i;i<sizeof(key[0]);i++)
	{
	   if (i < sizeof(key_original[0]))
	   {
		k = 0;
		do
		{
		    key[k][i] = key_original[k][i];
		    k++;
		} while (k < sizeof(key_original[0]));

	   } else {
		k = 0;
		do
		{
		    W_t [k] = key[k][i-1];
		    k++;
		} while(k < sizeof(W_t));

		k = 0;
		if (i%sizeof(key_original[0]) == 0)
		{
		    it3 = W_t[0];
		    for (k; k < sizeof(W_t); k++)  // FOR ROTACION
		    {
			W_tmp[k] = W_t[k+1];
		    }
		    W_tmp[sizeof(W_t)-1] = it3;

		    k = 0;
		    for (k; k < sizeof(W_t); k++)   // FOR SUSTITUCIÓN
		    {
			selectColRow(W_tmp[k]);
			W_tmp[k] = table [fh][lh];
		    }
```

```
                    k = 0;
                    for (k; k < sizeof(W_t); k++) // FOR XOR con R
                    {
                        W_tmp [k] = W_tmp[k] ^ R[(i/sizeof(key_original[0]))-1][k];
                    }
                }

                k = 0;
                for (k; k < sizeof(W_t); k++)   // FOR XOR con R
                {
                    W_tmp[k] = key[k][i-sizeof(key_original[0])] ^ W_tmp[k];
                }

                k = 0;
                for (k; k < sizeof(W_t); k++)           // FOR ASIGNACIÓN EN KEY
                {
                    key [k][i] = W_tmp[k];
                }

            }
        }
        call AMControl.start();
     }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    if (len == sizeof(DecryptRadioOFBMsg)) {
      DecryptRadioOFBMsg* btrpkt = (DecryptRadioOFBMsg*)payload;
      setLeds(0x04);
      d3 = btrpkt->data;
      printf("Cipherdata received: %#010x | ",d3);
      printfflush();

      //XOR CIFRADO INICIAL
      initVar();
      col_key = 0;

        do
        {
            do
            {
                 result_f[j][i] = basedata[j][i] ^ key[j][col_key];
                 j++;
            } while (j < rounds_col);
            j = 0; i++; col_key++;
        } while(i < rounds);

        //ITERACIONES n = 10 [0-9]
        b0 = 0;
        for(b0;b0 < n;b0++)
        {
            //PROCESO DE SUSTITUCION    //SALE CON result1
            initVar();
```

```
            sustitucionMatrix(result_f);

            //PROCESO DE ROTACIÓN       //SALE CON result2
            initVar();
            rotacionMatrix(result1);

            //PROCESO MIXCOLUMS         //SALE CON result1
            initVar();
            if ( b0 < n-1)
            {
                for (i;i<rounds;i++)
                {
                    for (j;j<rounds;j++)
                    {
                        for (k;k<rounds;k++)
                        {
                            selectColRow(M[i][k]);
                            it3 = L_table[fh][lh];

                            selectColRow(result2[k][j]);
                            it5 = L_table[fh][lh];
                            if(fh == 0x00 && lh == 0x00)
                            {
                                detect = TRUE;
                            }

                            it6 = it5 + it3;
                            if(it6-it5 != it3)
                            {
                                it6++;
                            }

                            selectColRow(it6);
                            if (detect == TRUE)
                            {
                                it4 ^= 0;
                                detect = FALSE;
                            } else {
                                it4 ^= E_table[fh][lh];
                            }
                        }
                        result1[i][j] = it4;
                        k = 0; it3 = 0; it4 = 0;
                    }
                    j = 0;
                }
                //COPIA MATRIX
                initVar();
                for(i;i < rounds;i++)
                {
                    for(j;j < rounds_col;j++)
                    {
                        result2[i][j] = result1[i][j];
                    }
                    j = 0;
                }
            }

            //XOR CIFRADO FINAL
            initVar();
            do
            {
                do
                {
                      result_f[j][i] = result2[j][i] ^ key[j][col_key];
                      j++;
                } while (j < rounds_col);
                j = 0; col_key++; i++;
            } while(i < rounds);
        }

        initVar();
        for (i;i < sizeof(basedata); i++)
        {
```

```
              v_fila[i] = basedata[j][k];
              j++;
              if (j>3)
              {
                  j = 0; k++;
              }
         }

         it3 = result_f[0][0] ^ d3;
         printf("DecryptData received: %#010x\n",it3);
         printfflush();

         i = 0;
         for (i;i < sizeof(basedata)-1; i++)
         {
               v_fila[i] = v_fila[i+1];
         }
         v_fila[sizeof(basedata)-1] = result_f[0][0];

         initVar();
         for (i;i < sizeof(basedata); i++)
         {
             basedata[j][k] = v_fila[i];
             j++;
             if (j>3)
             {
                 j = 0; k++;
             }
         }
     }
     return msg;
   }
}
```