# ELEC5514 Lab Project

| Group 22 | |
|---|---|
| **Member Name** | **SID** |
| Ke Zhang | 450432501 |
| Hao Qin | 440413879 |
| Xiangyu Yan | 450069156 |
| Yahong Liu | 440358628 |

## Content

# 1 Understanding of Wireless sensor network

Nowadays, with the development of Micro-Electro-Mechanism System, On-chip System and low consumption embedded technology, there is a huge improvement in Wireless Sensor Network (WSN). Some researcher said that there will be a tremendous revolution in the area of communication in the next few years.

Wireless sensor network is based on ad hoc network and get some improvement as its special advantages. It can be used in particular environments in which is hard to use cable and power supply or somewhere that people cannot get in. For example, when a chemical explosion happens, it is very dangerous for the fireman to detect the situation closely, so it is significantly to use something to collect the important information and transmit these data back to the fire station. Another example is that the military need to investigate enemy's rear area which is hard to detect by the radar. The wireless sensor network has the ability of self-adaption and do not need to be supported by other network. In the military example just mentioned before, the WSN nodes can be thrown to the rear area from the airplane when these nodes dropping on the ground, they can self-organize a network by themselves, each nodes can both act as coordinator and router. Each nodes using sensors to collect data and transmit these data to the sink node. The sink nodes send data to the internet and people can get these data remotely.

## 1.1 Embedded system

An embedded system is a computer system with real-time computing constrains used to control larger electrical or mechanical systems. For example, like MP3 player, GPS receiver, car brake system and so on are all the applications using embedded system to control. Embedded systems often use microcontrollers which include CPU, memory and peripherals. For all of the embedded systems, the ARM is most widely used 32-bit reduced instruction set architecture (RISC), and most of the communication devices use it. There are many companies have the ARM manufacture licenses like Apple, Atmel, Texas Instrument and so on. Wide range of processors available to meet conflicting goals of performance, cost and size.

Embedded system is used to control or monitor other machines. At first, it is only the mechanism link between the real time and the device. People write programs in it initially, and other devices are controlled by these programs. If we want to change the function of these devices, the only way to change them is to change the chips inside them. The classic aiming of the embedded system is to handle specific problems with embedded computing.

But now, with the development of internet technics, there is a new concept called 'the internet of things' became more and more popular. The internet of things (IoT) is the network linking with embedded objects and internet. Differ from the previous

embedded systems, the objects can collect and exchange data. IoT can achieve the goal of intelligent identify, positioning, tracking, monitoring and managing. There are four layers in concept, sensing layer, network layer and application layer. The sensing layer, as the lowest layer, is used to collect information and transmit data to the sink node by wire or wireless, some novel technology like multisensory technology, RFID technology 2-D barcode technology are used in this layer. The network layer which include communication network and internet is used to manage the transferred data and with security method. The highest layer is the application layer, this layer is used to provide the human-computer interface designing. For example, one can use personal device to control the electrical equipment in his house,

## 1.2 Performance metrics

### 1.2.1 Robustness

As WSN must face to various working environment, the strain capacity is needed indeed. In some tough conditions, the nodes may invalid or power off. The most remarkable difference between classic network and WSN is that the latter is highly limited. Because of the data transfer mechanism of WSN, every node can collect and transfer data. If information from remote distance transferred to the sink node, it may travel though many nodes. Once some nodes worked out, the data need to find another route to the sink nodes automatically. During the changing period, how to make sure that every information is complete is very important. By robustness protocol, the WSN system can have a better adaptability and hence have the survival ability in tough environment which is very important for the whole system.

### 1.2.2Scalability

For common WSN system, the amount of the nodes is usually very huge. It is needed for the protocol that has the ability of scalability to make the network bigger and bigger. [1] In some area, we need to increase the nodes density when the ambient environment changes. A good WSN system should have a good routing protocol to balance the flexibility and extendibility at the same time.

### 1.2.3Energy efficiency

WSN is made up of a lot of low-cost and low-power wireless sensor nodes which are deployed in a particular area. Some of these area is difficult for human to reach. That is too say it is hard to change the battery for these nodes. So how to improve the energy efficiency is very important. Nowadays, there are three methods to solve this problem. The first one is the optimize the data communication protocol through the network, as the WSN nodes need to form a private network automatically, each node must flood information to its neighbor to find the partner joined into a network. This step may waste a lot of energy for the detection. In addition, the nodes must transmit and collect data faithfully. So decreasing the energy cost during these process is very important. The second method is try to use a more efficiency batteries, but now it seems that the development of battery technology meets a bottleneck constraint. If the

limitation of the battery makes a big breakthrough, the use of WSN can will be broader. The third method is trying to gain energy from the surrounding environment, like solar energy, wind power and microwave power which could extend the nodes lifetime. But using this method may probably sacrifice the size of each node. For the above-mentioned three ways to increase the energy efficiency, optimize the transfer protocol is more feasible.

### 1.2.4 Security

Recently the security problem become more and more important. As the WSN have a high scalability which means that it could add or drop nodes easily. If there is a malicious attack try to join the network, the system will be very dangerous. The attacker can disguise as a node who wants to join the network. If the system can nor figure out, the fake node can steal the information or even it may destroy the whole system. So it is necessary to establish a security management protocol for the WSN. But unfortunately, establish a security method means that this may cost more energy during verification and identification. It is necessary to design a WSN security mechanism in an energy-save way.

### 1.2.5 Self-organization

The WSN should have the ability of the self-organization. It is important for WSN to self-organize whenever the situation changes. [2] When many nodes try to establish a network together, they should act as coordinator and sink automatically without any other hardware support.

### 1.2.6 Throughput

Throughput is a very important performance metric of WSN. It reflects the working efficiency directly in the transmission process. The higher the throughput, the bigger data rate it is. Due to the WSN operation principle, there is a huge confliction between energy limitation and data rate. In the IEEE 802.15.6 protocol stacks, the data link layer ensures the reliability for point to point and multi-point connection. In data link layer, the MAC sub layer is used to create network infrastructure and share communication resources. Throughput is highly depending on the MAC and connectivity. If they get failures, there will be a huge problem. Data collected by the sensor node transfer to the sink nodes to establish a connection between detection area and the outside internet communication. Because the sensor nodes and sink nodes are randomly formed, the signal strength through the different path are quite different. To reach the highest throughput target, we need to find out the best transmission route.

### 1.2.7 Average delay

Information transmit between different nodes. When the information reaches the sink node, it could be transmitted to the outside world. Every node obtains the information from its neighbor and send the information to its neighbor. It could spend a little time during the period for each node. If the number of nodes is significantly huge, that will cause a big delay. For a common WSN network, if we increase the WSN range, we

will decrease the nodes density and then reduce the time delay. However, reference [3] shows that the average delay is relate to the throughput. The bigger transmission radius means that the lower throughput. So how to balance these two factor is very important.

## 1.3 ZigBee Protocol

The familiarity of the ZigBee protocol and wireless sensor network protocol Wireless Sensor Network (WSN) system consist of many static nodes and dynamic nodes. WSN has self-organizing and cooperative capability. The sensor nodes mainly use broadcast communication paradigm and do not have global identification. ZigBee is one of the standard that have a significant potential in recent years which is based on the IEEE 802.15.4 Standard and created by the ZigBee Alliance.

The characteristics of ZigBee include:

### 1.3.1 Low date rate

The bandwidth is from 20 kb/s to 250 kb/s, that means it can be only used in low rate application.

### 1.3.2 Ultra Low power

As the data sheet said, in low power standby mode, ZigBee can maintain working 1 year by using two AA batteries. This is one of the biggest advantages of ZigBee.

### 1.3.3 Low cost

Because of the low transmission rate, the protocol of ZigBee is very simple, so that it doesn't need to spend a lot of money during the productive process.

### 1.3.4 High network capacity

Every ZigBee network can afford 255 devices, that is to say each ZigBee can communicate with 254 devices which may increase the scalability.

### 1.3.5 Flexible frequency range

There are three band for ZigBee utilization: 2.4GHz, 868MHz and 915MHz. Both of them are unlicensed.

### 1.3.6 Extreme low duty-cycle

Taking into account the applications of low communication frequency, ZigBee protocol implements the strategy of choosing sleeping or working period. This kind of strategy has provided a good characteristic to achieve the goal of low duty cycle.

## 2 Energy Efficiency of Wireless Sensor Network

In recent years, wireless sensor network has been greatly developed, but power consumption is still an important factor restricting the development of wireless sensor networks, wireless sensor nodes. Since the small size of sensor nodes, portable battery power is limited, the sensor node is discarded and failure due to the limitation of energy, therefore the energy supply constraints are obstacles sensor network applications. [4] In addition, the number of multi-sensor nodes, low cost requirements, wide distribution area, and regional environmental complex deployments, some regions even personnel are unavailable, the sensor node by way of replacing the battery to supplement energy is unrealistic. The above two points makes the application of sensor networks has been greatly restricted. If the primary design goals of traditional wireless network are to provide high quality of service and efficient bandwidth utilization, followed before considering energy saving, then, the primary design goals sensor network is efficient use of energy, and how to maximize the efficient use of energy is a sensor network life cycle the primary challenge facing the network, which is one of the most important differences between sensor networks and traditional networks.

The energy consumption of different parts of sensor nodes is somewhat different.

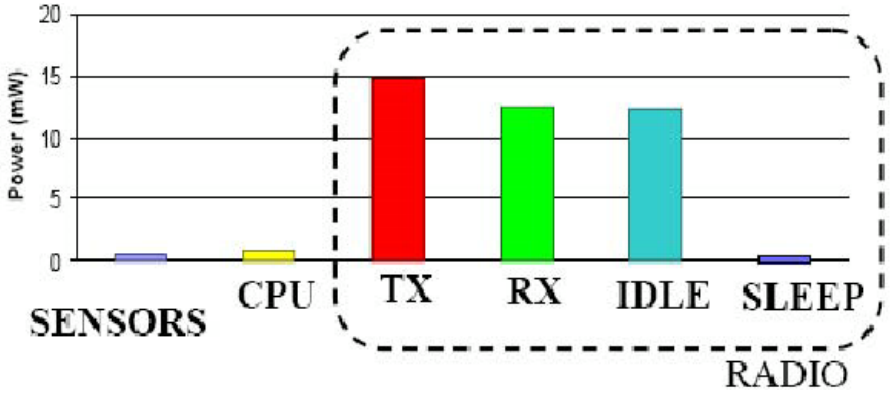


*Fig.1 The energy consumption of different parts of WSN*

The figure above shows the different energy consumption of different parts of WSN. Although different nodes on the respective power values are vary, but for most sensor nodes, power distribution is consistent with the general characteristics: firstly, the power consumption of sensor unit and the processor unit is far less than the energy consumption of the communication unit; secondly communication unit transmits data, the most of energy consumes when data is received and the unit is sleep, and the energy consumption is much smaller when the communication unit power consumption in sleep mode.

For the characteristics above, the methods of energy-saving for wireless sensor networks can be divided into two parts: hardware and software. The hardware

schemes are mainly including the efficient use of batteries, reduce energy consumption when the sensor is working, energy harvesting technology. For the main methods of software, there are protocol stack, cross-layer optimization, topology control algorithms and so on.

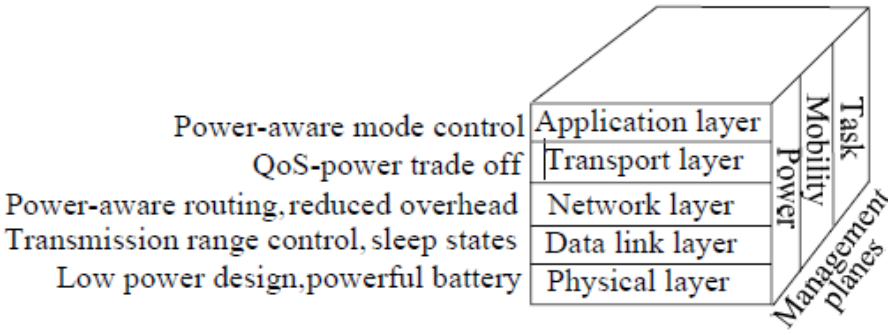


*Fig.2 Wireless sensor network protocol stack*

Currently, there are a number of ways from each layer of the sensor node protocol stack.

At the physical layer, the use of low-power devices and turn off unnecessary transaction performance can save a lot of energy. The energy consumption of the radio transceiver can be reduced by reducing the transmitting time. According to the instantaneous load flow, dynamic use of different modulation schemes can save energy. Determining the energy consumed on each node, identify the factors affecting the energy consumption of different components. Factors include the sensing element signal samples consumed, power, signal conditioning physical signal conversion, analog-to-digital converter (ADC).

Foe the data link layer, which is includes MAC and error control protocol. Since transport consumes most energy in the sensor nodes, the protocol of MAC should reduce the minimum energy efficiency of access to save the energy.

At the network layer, according to the versatility of the different nodes of the network may take directional, multi-hop, hop single cluster, or multi-hop clustering transmission scheme.

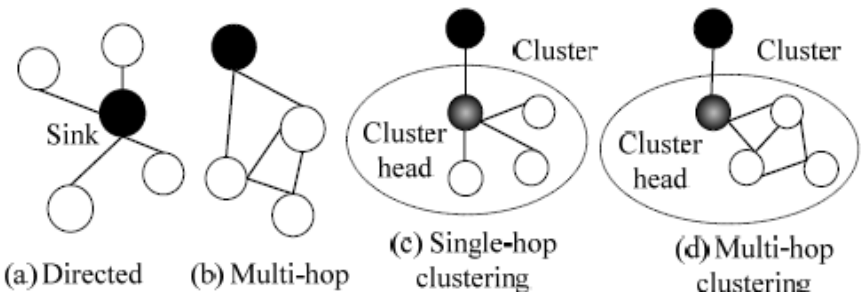


*Fig.3 Different transmission schemes in WSN*

And routing protocol is the protocol to find the most energy-efficient route without increasing latency.

## 2.1 Classification of Wireless Sensor Network routing protocol

WSNs routing protocol is responsible for the reliable transmission of data between the sink nodes and the remaining nodes. Since WSNs highly correlated with the application, a single routing protocol cannot meet the needs of a variety of applications, which have been studied numerous routing protocols. In order to reveal the characteristics of the agreement, a variety of classification methods can be used to classify them, such as communication mode, routing protocol, the route structure, establishment of timing routes, state maintenance, node identification strategies and delivery methods, using a variety of classification methods classify them. [5][6] Because the researchers combine multiple strategies to achieve routing mechanism, it belongs to the same routing protocols can be divided into different categories.

(1) According to how many paths are used during transmission path can be divided into single-path routing protocol and multi-path routing protocol. For single-path routing, which can save storage space and the amount of communication data are less. For multipath routing, which is fault-tolerant, robust good, and an optimal route can be selected from the numerous routes.

(2) According to whether there is a hierarchy node in the routing process, it can be divided into flat and hierarchical routing protocols. Plane routing is simple, robust good, but the cost of establishing and maintaining is high, and the number of hops of data transmission is more. So it is more suitable for small-scale network. Hierarchical routing scalability for large size of the network, but the cost of cluster maintenance is much, and the cluster head node is the key route, whose failure would cause routing fail.

(3) According to the relationship between the timing of the data transmission route established and data transmission, routing protocols can be divided into active, on-demand routing protocol and hybrid routing protocol. The cost of active routing is overhead and resource requirements are high; the delay of demand routing is large.

(4) Depending on whether the geographical position is used to identify the destination and the route calculation, it can be divided into location-based routing protocol and non-location-based routing protocols. There are a lot of WSNs applications need to know the location of the incident, but it requires a GPS positioning system, or other targeting methods to assist node computes position information.

(5) According to consider whether there is QoS routing constraints, it can be divided into guaranteed QoS routing protocol and does not guarantee QoS routing protocol.

QoS guaranteed routing protocol refers to consider QoS parameters, such as delay, packet loss rate when a routing is establishing.

(6) According to whether the data aggregation process during transmission, it can be divided into data aggregation routing protocol and non-data aggregation routing protocol. Aggregation routing protocol can reduce the traffic, but it takes time synchronization technology support, and increase the transmission delay.

## 2.2 Analysis of Routing Protocols in WSN

### 2.2.1 Flooding Protocol and Gossiping Protocol

These two protocols are the most classical and traditional protocols in WSN. In flooding protocol, nodes broadcasts to all neighbor nodes when they are transmitting or receiving data, and the broadcast stopping when the data arriving the destination. Flooding is very simple to implement, because it does not maintain any routing table and does not require discovering any routes. But this technique has several disadvantages, such as it is responsible for large bandwidth consumption and it wastes valuable energy. This is not an energy aware protocol. Gossiping protocol is the improved flooding protocol, it avoiding implosion but increase the delay.

### 2.2.2 SPIN Protocol

This is the first protocol based on data routing. When the node generates or receives data, in order to avoid the spread of the blind, it will transmit a notice to neighbors with ADV message, and the node that need the data will send REQ message as request, the data is sent to the requesting node as DATA message. The advantage of this protocol is: small ADV message alleviate the implosion problem; by naming data to solve the overlap problem; avoiding the use of resources blindly. Comparing with Flooding and Gossiping protocol, this protocol effectively saving energy, but the drawback is: when all neighbors do not need this data, the data will not be forwarded, which will cause distant nodes can not get the data, when most of the network node is a potential sink points, the problem is not serious, but if there are less sink point, it is a very serious problem; and when a sink point request every data, its surrounding nodes will run out of energy easily.

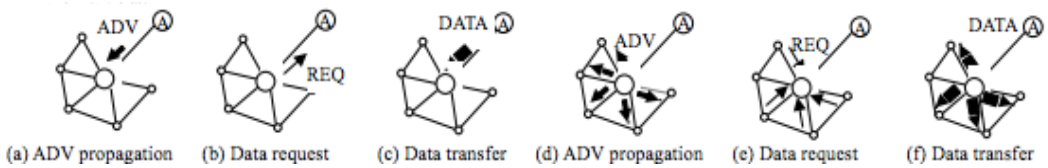


*Fig.4 Routing setup and data transmission in SPIN protocol*

### 2.2.3 LEACH and DEEC protocol

LEACH protocol is the first data fusion hierarchical routing protocol. To balance the energy consumption of each node, the heading node is selected randomly by round.

Nodes creates a random number between 0 and 1, and if the number less than T(n), the node will be the heading cluster.

$$T(n) = \begin{cases} \dfrac{p}{1 - p \times \left(r \bmod \dfrac{1}{p}\right)}, n \in G \\ 0, \quad else \end{cases}$$

Where p is the percentage of the number of cluster and the number of nodes, r is the number of rounds and G is the set of nodes that are not heading of cluster in recent 1/p round.

The heading cluster will broadcast this message via the radio channel, and the remaining nodes choose to join the cluster that has the strongest signal cluster header. Node will delivery the data to cluster head by a single hop, and cluster head also send the aggregated data to sink point by one hop. This protocol uses random cluster head selection to avoid excessive consumption of energy cluster heads, and increases network lifetime; data aggregation can effectively reduce the level of traffic. In this protocol, transmission delay is small, but it requires a higher level of power. Even in small-scale networks, nodes farther away from the sink point will use high-power communications, and it will lead to a shorter life time.

DEEC protocol is an improved LEACH protocol, the only different of these two protocols is, DEEC will chose the node with higher rest energy as the cluster head. [7]

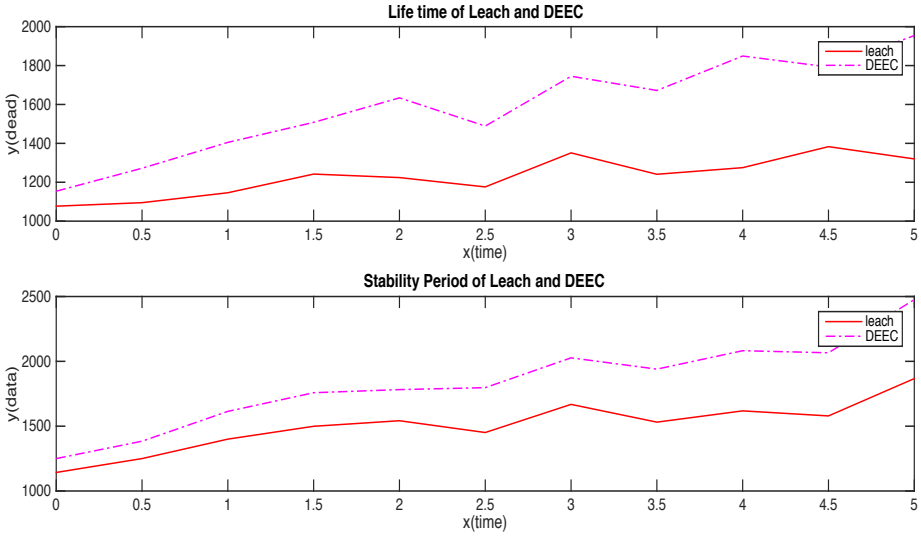


*Fig.5 Performance of LEACH and DEEC*

The figure above shows the performance of two different protocols in WSN. The life time of DEEC is longer than LEACH. And the longer life time means less energy consumption.

## 2.3 MIMO technique in WSN

Multiple-Input and Multiple-Output (MIMO), is a kind of smart antenna technology, which uses multiple antennas at both transmitter and receiver to optimize the performance of communication systems. In this case, the data throughput and link range increase significantly compared with original technology. Furthermore, MIMO do not need to raise the power of transmitter or channel bandwidth, it divided integral transmit power into multiple antennas to achieve array gain and diversity gain that improves the spectral and the link reliability respectively.
This energy efficiency investigation of MIMO scheme is broadened to individual single antenna nodes that collaborate to form multiple antenna transmitters or receivers. [8]

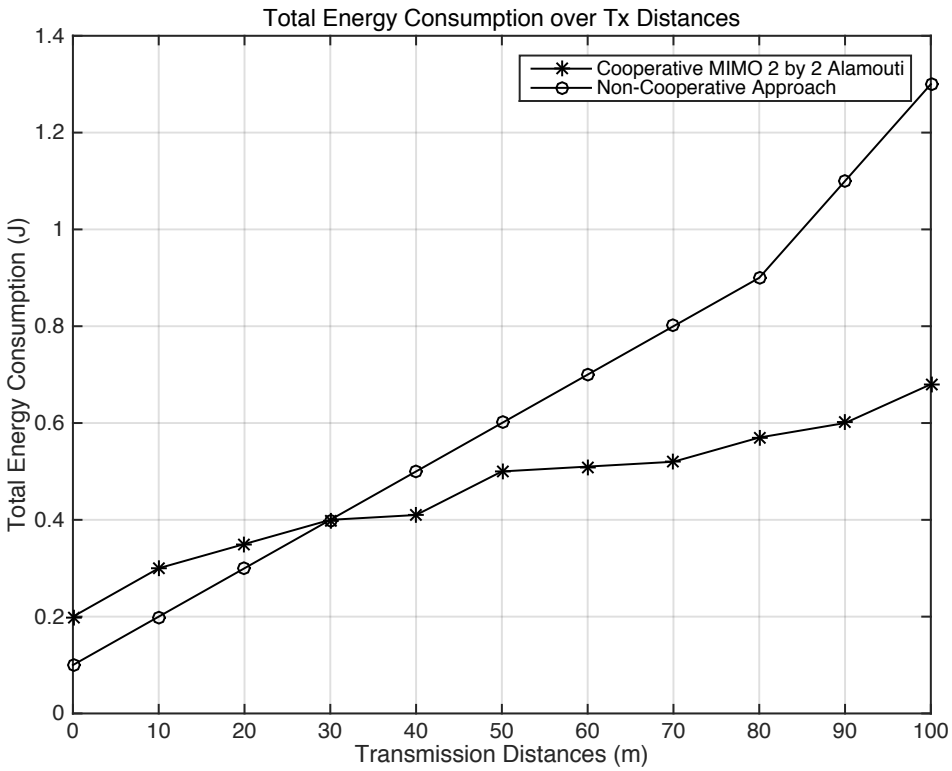


*Fig.6 Total Energy Consumption Comparison*

The figure above shows the total energy consumption of WSN with and without use MIMO. MIMO can reduce the total energy consumption when the distance is long.

# 3 Security of Wireless Sensor Network

In this section, we tried to analyze the security in ZigBee specification released by ZigBee Alliance, including whole security architecture, security mechanisms of each layer and the cryptography in current ZigBee specification.

At last, we provide a method of improving the encryption method in ZigBee specification. As a comparison, we formulate the current cryptography and the providing one.

## 3.1 Security Mechanism in ZigBee Specification

We learned about the security mechanisms elaborated in the specification released by ZigBee Alliance.

### 3.1.1 Security Service Provider

As Specified in ZigBee Specification [9], security service provider includes methods for key establishment, key transport, frame protection and device management. As Fig.7 shows, ZigBee standard construct mechanisms for the network layer and application support sub-layer. Application support sub-layer provides security management and mapping services. ZDO is responsible for device management, including security policies management and security configuration management. Application layer provides the application services to ZDO and ZigBee.

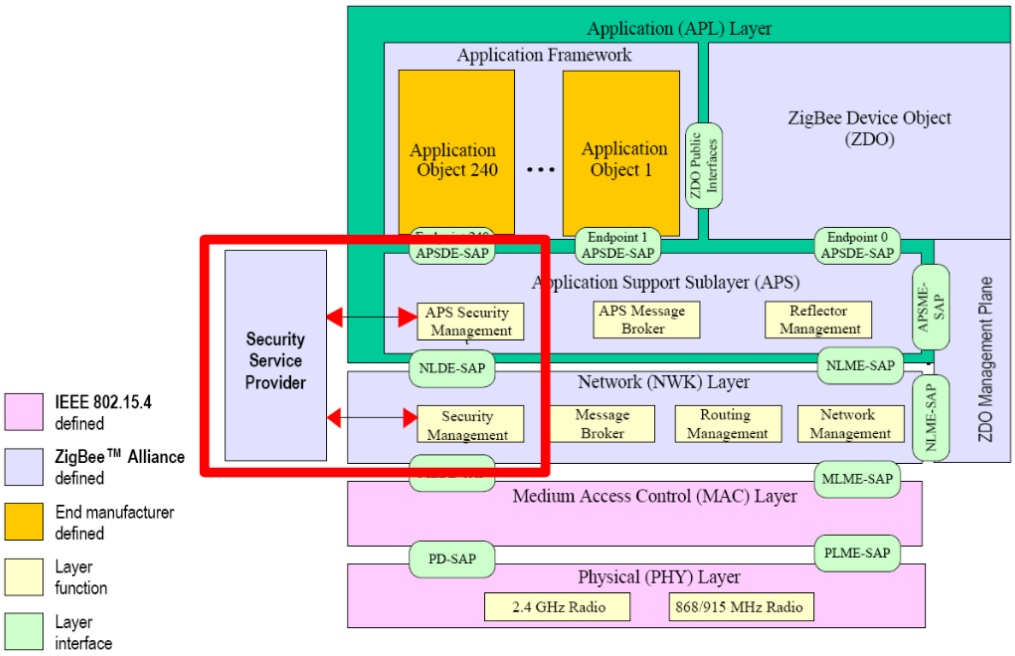


*Fig.7 Security Service Provider in ZigBee stack [10]*

### 3.1.2 Security Design Choices

Based on the assumption of ZigBee security implementations, the ZigBee specification used the notion of Open Trust Model.

It allows re-use of the same keying material among different layers on the same device and it allows end-to-end security to be realized on a device-to-device basis rather than between pairs of particular layers (or even pairs of applications) on two communicating devices. [1]

### 3.1.3 Security Keys

ZigBee devices use 128-bit symmetric encryption keys to provide security amongst a network. It is based on a link key and a network key.

Unicast communication between APL peer entities is secured by means of a 128-bit link key shared by two devices, while broadcast communications are secured by means of a 128-bit network key shared amongst all devices in the network. [9]

If using the security mode, the ZigBee stack will offer three types of keys used to secure communications:

(1) Master key:
used to derive the other key assisting symmetric key process. The network must first have the trust center to generate the master key to derive network key and link key to other devices.

(2) Network Key:
used to protect the confidentiality and integrity of broadcast and data. It also provide protection for network authentication. The network key is shared by multiple devices in the network, only in the broadcast message.

(3) Link key:
used to protect the confidentiality and integrity of the unicast data between two devices. The link key is only owned by two devices in communication, so that a single device requires multiple keys to protect each end to end session.

In ZigBee Pro, the administrator uses a symmetric key establishment method to derive network key and link key, but this requires the master key generated by the trust center.

### 3.1.4 Trust Centre

As we mentioned above, Trust Center is a secure device for distributing keys and allowing devices to join the network. Trust Center has two modes, namely residential mode and business mode (known as the standard mode and high-safety mode in ZigBee Pro).

Residential mode consumes fewer resources, but does not create keys or expand with the growing of network size. Commercial mode establishes and maintains keys, therefore has good scalability, but it requires more memory. In order to reduce storage requirements, the commercial mode can also share security key for high security applications.

As known to all, there are three device types in a ZigBee network: coordinator, router, and end device. Trust center is an application among the network. In a wireless network, the coordinator usually configures the security level. In most applications, the coordinator is configured as trust center.

## 3.2 Security Risks

### 3.2.1 Physical Attack

The most common way of attacking is physical attack. Most ZigBee devices record their keys in the hardware memory, attackers could get these keys by interfacing the devices. As known to all, the ZigBee devices use AES algorithm as the cryptography method. So that the key will be the same among the whole wireless sensor network. [11]

Obviously, it is a big risk of security.

### 3.2.2 Eavesdropping Attack

When the ZigBee is working in non-secure mode, the transmission data will not be encrypted and therefore data may be eavesdropping from external.

As we did in the previous lab, Fig.8 shows the packet of communication among the wireless sensor network. With the Wireshark tool, we could get the data easily. As we see in this picture, the ‘AF Data’ in red square frame shows the ASCII of the data, it indicates the data is ‘2 blank 3’, ‘5’, ‘4 blank 5’ and ‘9’ respectively.

This undoubtedly left a security risk of wireless sensor network.

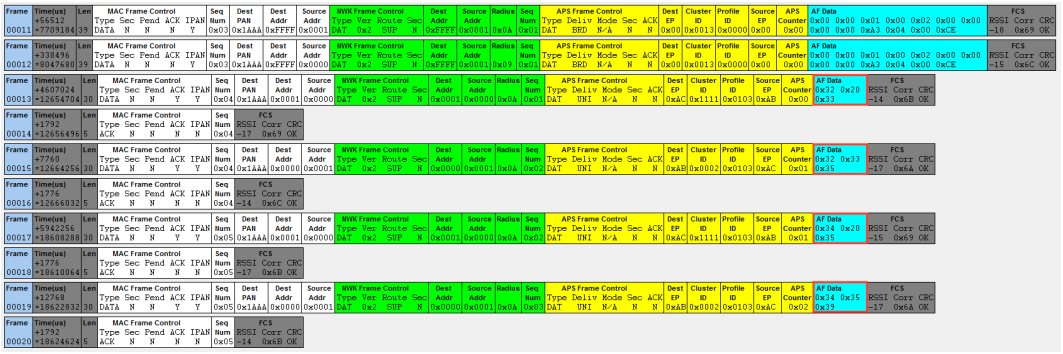

*Fig.8 Packets of communication among WSN*

### 3.2.3 Security Key Attack

As the key may be transmitted in plain text during transmission process, therefore key is likely to be stolen and thus for decrypting the communication data.
In addition, attackers may use reverse engineering to analysis firmware of some of the smart devices, derive key to decrypt communications command, and then forged the trusted command to attack.

Researchers at Black Hat and Def Con warned about security flaws in Internet of Things devices. Li Jun and Yang Qing of Qihoo360's Unicorn team, presented their idea at Dec Con 23. [12] In their report, they focused on showing how to get the key by accessing to the devices and sniffering the network packets.

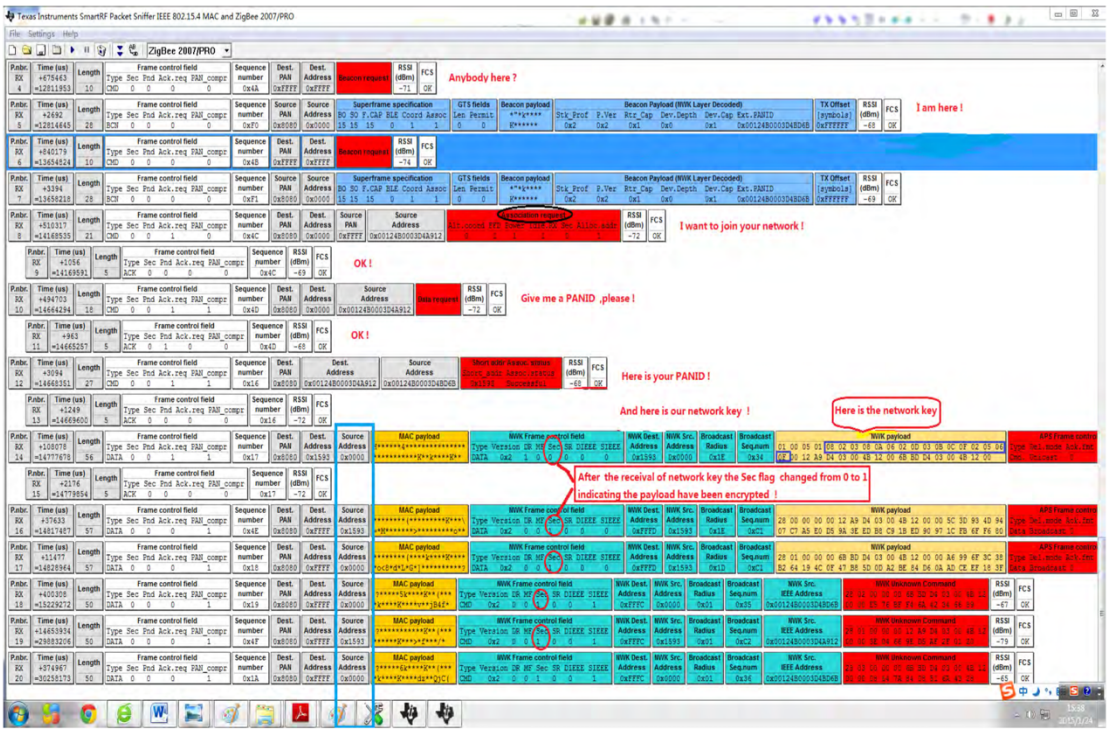


*Fig.9 Find the encryption key by sniffing* [12]

## 3.3 A Method of Improving ZigBee Security Performance

Our application is based on Microchip ZigBee Stack 2006, Microchip PIC18 MCU and MRF24J40 transceivers. As the security function in Microchip ZigBee Stack is only supplied in the Pro Stack version and is not free to use, there is a security risk for the ZigBee network based on Microchip ZigBee Stack 2006. Attackers could easily get the data by sniffering the network.

So that we implement the method of cryptography in order to improve the security of currently used Microchip hardware.

### 3.3.1 Symmetric Key Cryptography

As we reviewed before, the ZigBee specification is using AES-128 method for cryptography. AES-128 is known as the symmetric key cryptography, which means both sides share the same private key for encrypting and decrypting.

The main reason for using symmetric key cryptography in embedded systems is that they are simple and fast. It has a very big security issue that both sides need to know the key in some way. In ZigBee networks, this private key is usually installed during the device production period.

AES is based on the Rijndael cipher developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST during the AES selection process. [13]

We will elaborate our algorithm later in part 3.3.3.

### 3.3.2 Cryptography based on Microchip ZigBee Stack

As we stated in before, the ZigBee specification defined to use AES algorithm as the encryption method. And as the Microchip Stack 2006 normal version does not implement the security function. So we choose to implement an AES-128 cryptography in the application layer.

We tested the method with a coordinator and router. When the coordinator sends a string, it will encrypt the string before sending. The router gets the encrypted string and decrypts it into the real message, then it will send it back with plain text to coordinator for verifying. Figures below shows the Serial command and the packets sniffered among transmission. The data in transmission is surrounded by red squares.

We could see that it perfectly implements the cryptography on the device application layer. With the packets sniffered by ZENA, we could see that the attackers could not get the information simply from the packets. Without the key, no one could crack the cipher text. In addition, we won't transmit the key among devices to prevent interception. For the future work, it is better to implement a key management strategy to maintain the symmetric key.

(1) Case 1: Input Plain text is "helloworldabcdef".

```
Enter a menu choice:
     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice: 0
Please input string: helloworldabcdef
Input string:helloworldabcdef

string to hex (Input array): helloworldabcdef


CBC encrypt:
In function input data: hdnokriudmkioiki


In function encrypted data: ▒*'▒t▒▒▒▒▒xZh▒


returned encrypted data: ▒*'▒t▒▒▒▒▒xZh▒


Sending bufferEncrypt to router...

     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice:  Message sent successfully.

Received From Router: helloworldabcdef
```

*Fig.10 Serial command of Case 1*

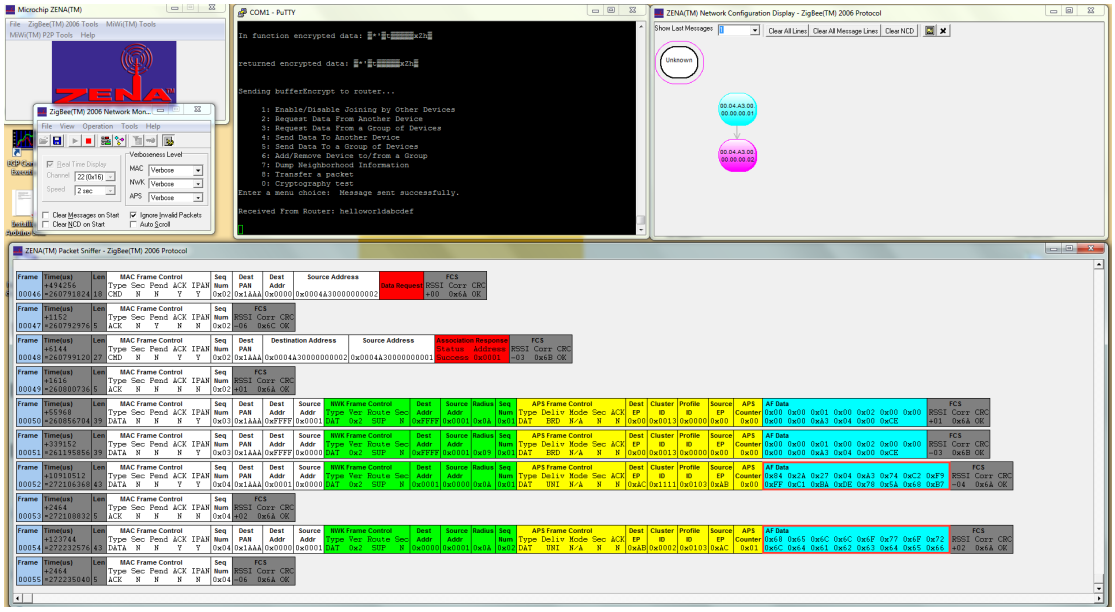

*Fig.11 Transmission packets of Case 1*

(2) Case 2: Input Plain text is “abcdefghijklmnop”.

```
Enter a menu choice:
     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice: 0
Please input string: abcdefghijklmnop
Input string:abcdefghijklmnop

string to hex (Input array): abcdefghijklmnop


CBC encrypt:
In function input data: acagacaoacagac


In function encrypted data: ▒   2N▒▒L|▒)▒


returned encrypted data: ▒      2N▒▒L|▒)▒


Sending bufferEncrypt to router...

     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice:  Message sent successfully.

Received From Router: abcdefghijklmnop
```

*Fig.12 Serial command of Case 2*

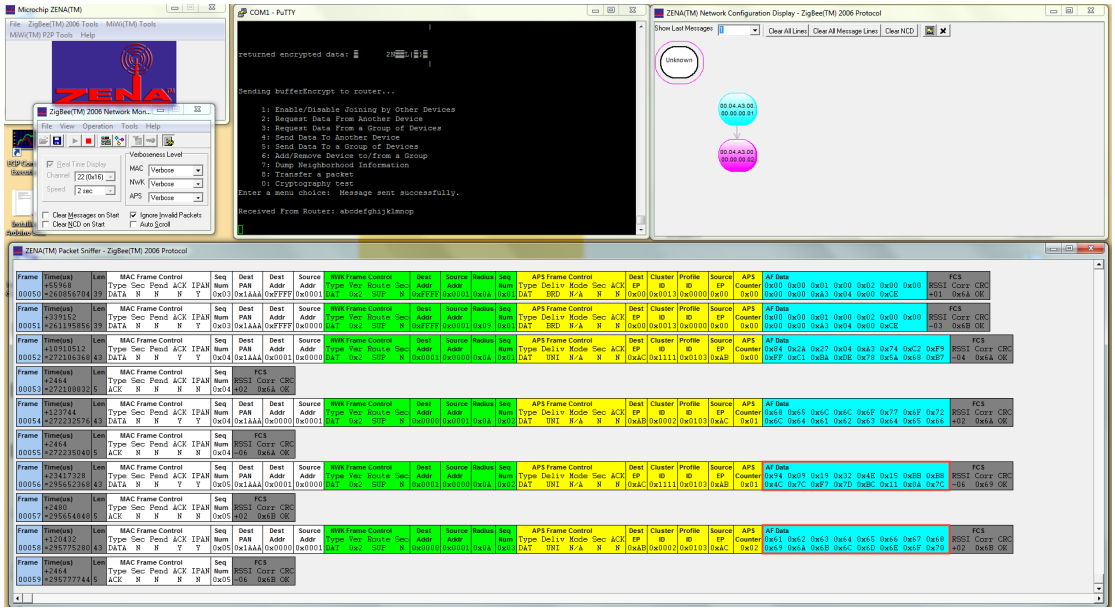

*Fig.13 Transmission packets of Case 2*

(3) Case 3: Input Plain text is “1234567890123456”.

```
Enter a menu choice:
     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice: 0
Please input string: 1234567890123456
Input string:1234567890123456

string to hex (Input array): 1234567890123456


CBC encrypt:
In function input data: 1317131?19;9?9;9


In function encrypted data: :▒*w▒)"▒▒D\▒▒


returned encrypted data: :▒*w▒)"▒▒D\▒▒


Sending bufferEncrypt to router...

     1: Enable/Disable Joining by Other Devices
     2: Request Data From Another Device
     3: Request Data From a Group of Devices
     4: Send Data To Another Device
     5: Send Data To a Group of Devices
     6: Add/Remove Device to/from a Group
     7: Dump Neighborhood Information
     8: Transfer a packet
     0: Cryptography test
Enter a menu choice:  Message sent successfully.

Received From Router: 1234567890123456
```

*Fig.14 Serial command of Case 3*

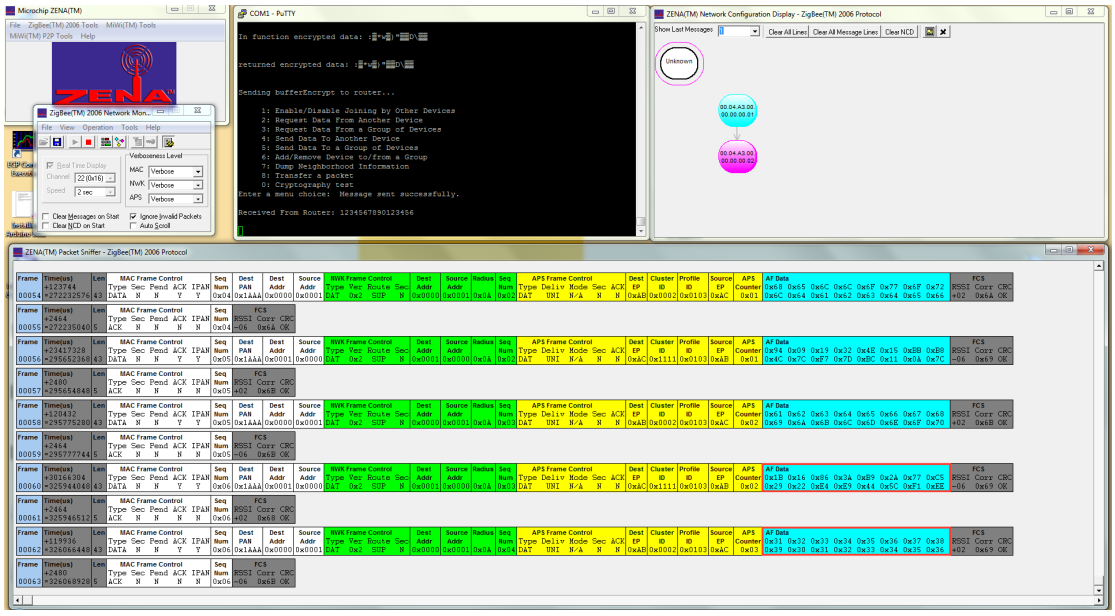

*Fig.15 Transmission packets of Case 3*

### 3.3.3 Elaboration of Cryptography Algorithm

We choose AES-CBC (Cipher Block Chaining) mode as our algorithm, which is also part of the implementation in the ZigBee specification. Referring to the documents released by National Institute of Standards and Technology (NIST), we used the recommended vectors and keys for CBC mode. [6]

#### 3.3.3.1 AES Algorithm

We used AES-128 which means the length of key is 128 bits. All the encryption is operating in a 4 × 4 byte matrix called "cipher". The initial value is the first block of plain text. The encryption actually consists of many rounds of operation. Each round of operation contains four functions: AddRoundKey, SubBytes, ShiftRows and MixColumns. These functions are the main operation of AES algorithm and as elaborated below.

(1) AddRoundKey: Each byte will XOR (exclusive or) with the round key (produced by the function KeyExpansion).
(2) SubBytes: Then each byte will be replaced according to a nonlinear method.
(3) ShiftRows: After that, each row of the matrix will be cyclic shifted to left.
(4) MixColumns: At the end, mix each column according to linear conversion.

Obviously, the decryption is the inverse operation of encryption.

#### 3.3.3.2 CBC mode

The CBC (Cipher Block Chaining) mode is a confidentiality mode whose encryption process features the combining (Chaining) of the plaintext blocks with the previous cipher text blocks. [14]

Similar to other modes in AES, it also requires the initialization vector (IV) to do the operations. The IV vector could be public but it has to be unpredictable. The CBC mode and the process is shown as follows:

CBC Encryption:

$$C_1 = CIPH_K(P_1 \oplus IV);$$
$$C_j = CIPH_K(P_j \oplus C_{j-1}) \quad \text{for } j = 2 \ldots n.$$

CBC Decryption:

$$P_1 = CIPH^{-1}_K(C_1) \oplus IV;$$
$$P_j = CIPH^{-1}_K(C_j) \oplus C_{j-1} \quad \text{for } j = 2 \ldots n.$$

*Fig.16 AES CBC mode definition [14]*

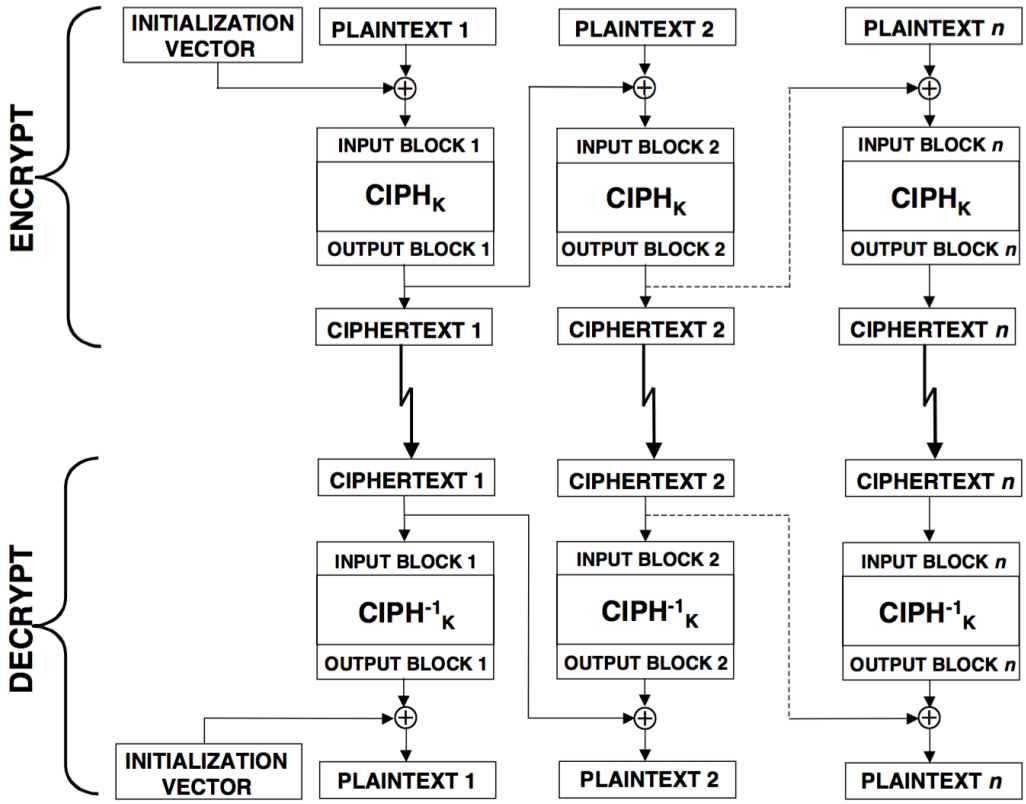


*Fig.17 AES CBC mode process* [14]

CBC mode is actually a cyclic pattern. In the very beginning, the first block plain text do the XOR operation with the initialization vector (IV) to get the first block of cipher text. Then this first block of cipher text will XOR with the second block of plain text to get second cipher text. Each block of plain text will XOR with the previous output block of cipher text to get the output cipher text of this round. The purpose of using CBC mode is to enhance the difficulty of cracking.

The complete codes for cryptography implementation in Coordinator and Router are attached in appendix.

## References


[1] Naumov, V. & Gross, T. 2005, "Scalability of routing methods in ad hoc networks", *Performance Evaluation,* vol. 62, no. 1, pp. 193-209.

[2] Dressler, F. 2007, *Self-organization in sensor and actor networks,* John Wiley & Sons, Hoboken, NJ;Chichester, England;.

[3] Dai, H. 2009, "Throughput and delay in wireless sensor networks using directional antennas", , pp. 421.

[4] Limin, Sun, Li jianzhong, Chen Yu, Zhu Hongsong. "wireless Sensor Networks [M]". Peking: Tsinghua University Press, 2005.

[5] Haas, Z., Halpern, J. & Li, L. 2006, "Gossip-based ad hoc routing", *IEEE/ACM Transactions on Networking (TON),* vol. 14, no. 3, pp. 479-491.

[6] Kulik, J., Heinzelman, W. & Balakrishnan, H. 2002, "Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks", *Wireless Networks,* vol. 8, no. 2, pp. 169-185.

[7] S. Tripti, B. Kumar & G.S.Tomar, “Performance Comparision of LEACH, SEP and DEEC Protocol in Wireless Sensor Network”, *Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering,* pp. 10-17.

[8] Sachan, V.K. & Imam, S.A. 2012, "Performance Analysis of Cooperative MIMO Techniques in Wireless Sensor Networks", *IUP Journal of Telecommunications,* vol. 4, no. 2, pp. 32.

[9] ZigBee Specification - Document 053474r17. Retrieved from: http://www.zigbee.org/

[10] Robert Cragie. "ZigBee Security." 2009. ZigBee Alliance. Retrieved from: https://docs.zigbee.org/

[11] Brad Bowers. "ZigBee Wireless Security: A New Age Penetration Tester's Toolkit." 2012. CISCO Press.

[12] LI Jun, YANG Qing. "I’M A NEWBIE YET I CAN HACK ZIGBEE: Take Unauthorized Control Over ZigBee Devices." 2015. DEF CON 23 Hacking Conference.

[13] Daemen, Joan, and Vincent Rijmen. "AES proposal: Rijndael." 1999.

[14] Dworkin, Morris. "Recommendation for block cipher modes of operation. methods and techniques." No. NIST-SP-800-38A. NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2001.

# Appendix I Variables and functions definitions in both Coordinator and Router

```
/
    ********************************************************************************
    ***/
/* Defines:
    */
/
    ********************************************************************************
    ***/
// The number of columns comprising a state in AES. This is a constant in AES.
    Value=4
#define Nb 4
// The number of 32 bit words in a key.
#define Nk 4
// Key length in bytes [128 bit]
#define KEYLEN 16
// The number of rounds in AES Cipher.
#define Nr 10

#define Multiply(x, y)                          \
(  ((y & 1) * x) ^                              \
((y>>1 & 1) * xtime(x)) ^                       \
((y>>2 & 1) * xtime(xtime(x))) ^                \
((y>>3 & 1) * xtime(xtime(xtime(x)))) ^         \
((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))))   \


/
    ********************************************************************************
    ***/
/* Private variables:
    */
/
    ********************************************************************************
    ***/
```

```
// state - array holding the intermediate results during decryption.
typedef unsigned char state_t[4][4];
static state_t* state;

// The array that stores the round keys.
static unsigned char RoundKey[176];

// The Key input to the AES Program
static const unsigned char* Key;

// Initial Vector used only for CBC mode
static unsigned char* Iv;


// The lookup-tables are marked const so they can be placed in read-only
    storage instead of RAM
// The numbers below can be computed dynamically trading ROM for RAM -
// This can be useful in (embedded) bootloader applications, where ROM is
    often limited.
static const unsigned char sbox[256] =   {
    //0     1    2      3     4    5     6     7      8    9     A      B    C
        D     E     F
    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 };

static const unsigned char rsbox[256] =
{ 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 };


// The round constant word array, Rcon[i], contains the values given by
// x to th e power (i-1) being powers of x (x is denoted as {02}) in the field
    GF(2^8)
// Note that i starts at 1, not 0).
static const unsigned char Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
        0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
        0xef, 0xc5, 0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
        0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
        0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,
        0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,
        0x94, 0x33, 0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,
        0x20, 0x40, 0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
        0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2,
        0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
        0x04, 0x08, 0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc,
        0x63, 0xc6, 0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,
        0xbd, 0x61, 0xc2, 0x9f,
```

```
        0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb,
             0x8d, 0x01, 0x02, 0x04,
        0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
             0x2f, 0x5e, 0xbc, 0x63,
        0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
             0x72, 0xe4, 0xd3, 0xbd,
        0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
             0x74, 0xe8, 0xcb  };

/
    ********************************************************************************
    ***/
/* Private functions:
    */
/
    ********************************************************************************
    ***/

void *memsettest(void *src, int c, unsigned int count)
{
    //assert(src!=NULL);
    char *tmpsrc=(char*)src;
    while(count--)
        *tmpsrc++ =(char)c;
    return src;
}


static unsigned char getSBoxValue(unsigned char num)
{
    return sbox[num];
}

static unsigned char getSBoxInvert(unsigned char num)
{
    return rsbox[num];
}

// This function produces Nb(Nr+1) round keys. The round keys are used in each
    round to decrypt the states.
static void KeyExpansion(void)
{
    unsigned int i, j, k;
    unsigned char tempa[4]; // Used for the column/row operations

    // The first round key is the key itself.
    for(i = 0; i < Nk; ++i)
    {
        RoundKey[(i * 4) + 0] = Key[(i * 4) + 0];
        RoundKey[(i * 4) + 1] = Key[(i * 4) + 1];
        RoundKey[(i * 4) + 2] = Key[(i * 4) + 2];
        RoundKey[(i * 4) + 3] = Key[(i * 4) + 3];
    }

    // All other round keys are found from the previous round keys.
    for(; (i < (Nb * (Nr + 1))); ++i)
    {
        for(j = 0; j < 4; ++j)
        {
            tempa[j]=RoundKey[(i-1) * 4 + j];
```

```
            }
            if (i % Nk == 0)
            {
                // This function rotates the 4 bytes in a word to the left once.
                // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

                // Function RotWord()
                {
                    k = tempa[0];
                    tempa[0] = tempa[1];
                    tempa[1] = tempa[2];
                    tempa[2] = tempa[3];
                    tempa[3] = k;
                }

                // SubWord() is a function that takes a four-byte input word and
                // applies the S-box to each of the four bytes to produce an
                     output word.

                // Function Subword()
                {
                    tempa[0] = getSBoxValue(tempa[0]);
                    tempa[1] = getSBoxValue(tempa[1]);
                    tempa[2] = getSBoxValue(tempa[2]);
                    tempa[3] = getSBoxValue(tempa[3]);
                }

                tempa[0] =  tempa[0] ^ Rcon[i/Nk];
            }
            else if (Nk > 6 && i % Nk == 4)
            {
                // Function Subword()
                {
                    tempa[0] = getSBoxValue(tempa[0]);
                    tempa[1] = getSBoxValue(tempa[1]);
                    tempa[2] = getSBoxValue(tempa[2]);
                    tempa[3] = getSBoxValue(tempa[3]);
                }
            }
            RoundKey[i * 4 + 0] = RoundKey[(i - Nk) * 4 + 0] ^ tempa[0];
            RoundKey[i * 4 + 1] = RoundKey[(i - Nk) * 4 + 1] ^ tempa[1];
            RoundKey[i * 4 + 2] = RoundKey[(i - Nk) * 4 + 2] ^ tempa[2];
            RoundKey[i * 4 + 3] = RoundKey[(i - Nk) * 4 + 3] ^ tempa[3];
        }
    }

    // This function adds the round key to state.
    // The round key is added to the state by an XOR function.
    static void AddRoundKey(unsigned char round)
    {
        unsigned char i,j;
        for(i=0;i<4;++i)
        {
            for(j = 0; j < 4; ++j)
            {
                (*state)[i][j] ^= RoundKey[round * Nb * 4 + i * Nb + j];
            }
        }
    }
```

```
// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
static void SubBytes(void)
{
    unsigned char i, j;
    for(i = 0; i < 4; ++i)
    {
        for(j = 0; j < 4; ++j)
        {
            (*state)[j][i] = getSBoxValue((*state)[j][i]);
        }
    }
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
static void ShiftRows(void)
{
    unsigned char temp;

    // Rotate first row 1 columns to left
    temp           = (*state)[0][1];
    (*state)[0][1] = (*state)[1][1];
    (*state)[1][1] = (*state)[2][1];
    (*state)[2][1] = (*state)[3][1];
    (*state)[3][1] = temp;

    // Rotate second row 2 columns to left
    temp           = (*state)[0][2];
    (*state)[0][2] = (*state)[2][2];
    (*state)[2][2] = temp;

    temp       = (*state)[1][2];
    (*state)[1][2] = (*state)[3][2];
    (*state)[3][2] = temp;

    // Rotate third row 3 columns to left
    temp       = (*state)[0][3];
    (*state)[0][3] = (*state)[3][3];
    (*state)[3][3] = (*state)[2][3];
    (*state)[2][3] = (*state)[1][3];
    (*state)[1][3] = temp;
}

static unsigned char xtime(unsigned char x)
{
    return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
}

// MixColumns function mixes the columns of the state matrix
static void MixColumns(void)
{
    unsigned char i;
    unsigned char Tmp,Tm,t;
    for(i = 0; i < 4; ++i)
    {
        t   = (*state)[i][0];
        Tmp = (*state)[i][0] ^ (*state)[i][1] ^ (*state)[i][2] ^ (*state)[i][3
             ] ;
```

```
            Tm  = (*state)[i][0] ^ (*state)[i][1] ; Tm = xtime(Tm);  (*state)[i][0
                ] ^= Tm ^ Tmp ;
            Tm  = (*state)[i][1] ^ (*state)[i][2] ; Tm = xtime(Tm);  (*state)[i][1
                ] ^= Tm ^ Tmp ;
            Tm  = (*state)[i][2] ^ (*state)[i][3] ; Tm = xtime(Tm);  (*state)[i][2
                ] ^= Tm ^ Tmp ;
            Tm  = (*state)[i][3] ^ t ;        Tm = xtime(Tm);  (*state)[i][3] ^=
                Tm ^ Tmp ;
        }
    }



    // MixColumns function mixes the columns of the state matrix.
    // The method used to multiply may be difficult to understand for the
        inexperienced.
    // Please use the references to gain more information.
    static void InvMixColumns(void)
    {
        int i;
        unsigned char a,b,c,d;
        for(i=0;i<4;++i)
        {
            a = (*state)[i][0];
            b = (*state)[i][1];
            c = (*state)[i][2];
            d = (*state)[i][3];

            (*state)[i][0] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c,
                0x0d) ^ Multiply(d, 0x09);
            (*state)[i][1] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c,
                0x0b) ^ Multiply(d, 0x0d);
            (*state)[i][2] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c,
                0x0e) ^ Multiply(d, 0x0b);
            (*state)[i][3] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c,
                0x09) ^ Multiply(d, 0x0e);
        }
    }


    // The SubBytes Function Substitutes the values in the
    // state matrix with values in an S-box.
    static void InvSubBytes(void)
    {
        unsigned char i,j;
        for(i=0;i<4;++i)
        {
            for(j=0;j<4;++j)
            {
                (*state)[j][i] = getSBoxInvert((*state)[j][i]);
            }
        }
    }

    static void InvShiftRows(void)
    {
        unsigned char temp;

        // Rotate first row 1 columns to right
        temp=(*state)[3][1];
```

```
        (*state)[3][1]=(*state)[2][1];
        (*state)[2][1]=(*state)[1][1];
        (*state)[1][1]=(*state)[0][1];
        (*state)[0][1]=temp;

        // Rotate second row 2 columns to right
        temp=(*state)[0][2];
        (*state)[0][2]=(*state)[2][2];
        (*state)[2][2]=temp;

        temp=(*state)[1][2];
        (*state)[1][2]=(*state)[3][2];
        (*state)[3][2]=temp;

        // Rotate third row 3 columns to right
        temp=(*state)[0][3];
        (*state)[0][3]=(*state)[1][3];
        (*state)[1][3]=(*state)[2][3];
        (*state)[2][3]=(*state)[3][3];
        (*state)[3][3]=temp;
    }


    // Cipher is the main function that encrypts the PlainText.
    static void Cipher(void)
    {
        unsigned char round = 0;

        // Add the First round key to the state before starting the rounds.
        AddRoundKey(0);

        // There will be Nr rounds.
        // The first Nr-1 rounds are identical.
        // These Nr-1 rounds are executed in the loop below.
        for(round = 1; round < Nr; ++round)
        {
            SubBytes();
            ShiftRows();
            MixColumns();
            AddRoundKey(round);
        }

        // The last round is given below.
        // The MixColumns function is not here in the last round.
        SubBytes();
        ShiftRows();
        AddRoundKey(Nr);
    }

    static void InvCipher(void)
    {
        unsigned char round=0;

        // Add the First round key to the state before starting the rounds.
        AddRoundKey(Nr);

        // There will be Nr rounds.
        // The first Nr-1 rounds are identical.
        // These Nr-1 rounds are executed in the loop below.
        for(round=Nr-1;round>0;round--)
```

```
        {
            InvShiftRows();
            InvSubBytes();
            AddRoundKey(round);
            InvMixColumns();
        }

        // The last round is given below.
        // The MixColumns function is not here in the last round.
        InvShiftRows();
        InvSubBytes();
        AddRoundKey(0);
    }

    static void BlockCopy(unsigned char* output, unsigned char* input)
    {
        unsigned char i;
        for (i=0;i<KEYLEN;++i)
        {
            output[i] = input[i];
        }
    }


    static void XorWithIv(unsigned char* buf)
    {
        unsigned char i;
        for(i = 0; i < KEYLEN; ++i)
        {
            buf[i] ^= Iv[i];
        }
    }


    /
        ****************************************************************************
        ***/
    /* Cryptography:
        */
    /
        ****************************************************************************
        ***/
    void AES128_CBC_encrypt_buffer(unsigned char* output, unsigned char* input,
        unsigned int length, const unsigned char* key, const unsigned char* iv)
    {
        unsigned int i;
        unsigned char remainders = length % KEYLEN; /* Remaining bytes in the last
             non-full block */

        BlockCopy(output, input);
        state = (state_t*)output;

        // Skip the key expansion if key is passed as 0
        if(0 != key)
        {
            Key = key;
            KeyExpansion();
        }

        if(iv != 0)
```

```
        {
            Iv = (unsigned char*)iv;
        }

        for(i = 0; i < length; i += KEYLEN)
        {
            XorWithIv(input);
            BlockCopy(output, input);
            state = (state_t*)output;
            Cipher();
            Iv = output;
            input += KEYLEN;
            output += KEYLEN;
        }

        if(remainders)
        {
            BlockCopy(output, input);
            memsettest(output + remainders, 0, KEYLEN - remainders); /* add 0-
                 padding */
            state = (state_t*)output;
            Cipher();
        }
    }

    void AES128_CBC_decrypt_buffer(unsigned char* output, unsigned char* input,
        unsigned int length, const unsigned char* key, const unsigned char* iv)
    {
        unsigned int i;
        unsigned char remainders = length % KEYLEN; /* Remaining bytes in the last
             non-full block */

        BlockCopy(output, input);
        state = (state_t*)output;

        // Skip the key expansion if key is passed as 0
        if(0 != key)
        {
            Key = key;
            KeyExpansion();
        }

        // If iv is passed as 0, we continue to encrypt without re-setting the Iv
        if(iv != 0)
        {
            Iv = (unsigned char*)iv;
        }

        for(i = 0; i < length; i += KEYLEN)
        {
            BlockCopy(output, input);
            state = (state_t*)output;
            InvCipher();
            XorWithIv(output);
            Iv = input;
            input += KEYLEN;
            output += KEYLEN;
        }

        if(remainders)
```

```
        {
            BlockCopy(output, input);
            memsettest(output+remainders, 0, KEYLEN - remainders); /* add 0-
                padding */
            state = (state_t*)output;
            InvCipher();
        }
    }


    /
        ***************************************************************************
        ***/
    /* Testing:
        */
    /
        ***************************************************************************
        ***/
    // print string to hex
    static void printStringToHex(unsigned char* str)
    {
        unsigned char i;
        for(i = 0; i < 16; i++){
            //printf("%.2x", str[i]);
             ConsolePut(str[i]);
        }
        printf("\n");
    }

    unsigned char bufferEncrypt[16];
    unsigned char encryptArrayInput[16] = {};

    static void test_encrypt_cbc(unsigned char* in)
    {
        unsigned char key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
             0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
        unsigned char iv[]  = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
             0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
        //unsigned char inside[]  = {0x68,0x65,0x6c,0x6c,0x6f,0x77,0x6f,0x72,0x6c,
             0x64,0x61,0x62,0x63,0x64,0x65,0x66};
        unsigned char out[] = { 0x84,0x2a,0x27,0x04,0xa3,0x74,0xc2,0xf9,0xff,0xc1,
             0xba,0xde,0x78,0x5a,0x68,0xb7 };

        //unsigned char bufferEncrypt[16];

        AES128_CBC_encrypt_buffer(bufferEncrypt, encryptArrayInput, 16, key, iv);


        printf("\r\nCBC encrypt: ");
        /*
        if(0 == strncmp((char*) out, (char*) bufferEncrypt, 16))
        {
            printf("SUCCESS!\r\n");
        }
        else
        {
            printf("FAILURE!\r\n");
        }
        */
```

```
        unsigned char i;
        printf("\r\nIn function input data: ");
        for(i = 0; i < 1; ++i)
        {
            printStringToHex(in + (i*16));
        }
        printf("\r\n");

        printf("\r\nIn function encrypted data: ");
        for(i = 0; i < 1; ++i)
        {
            printStringToHex(bufferEncrypt + (i*16));
        }
        printf("\r\n");


    //    return bufferEncrypt;

    }

    unsigned char bufferDecrypt[16];

    static void test_decrypt_cbc(unsigned char* in)
    {
        unsigned char key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
             0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
        unsigned char iv[]  = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
             0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
        //unsigned char inside[]  = { 0x84,0x2a,
             0x27,0x04,0xa3,0x74,0xc2,0xf9,0xff,0xc1,0xba,0xde,0x78,0x5a,
             0x68,0xb7 };
        unsigned char out[] = { 0x68,0x65,0x6c,0x6c,0x6f,0x77,0x6f,0x72,0x6c,0x64,
             0x61,0x62,0x63,0x64,0x65,0x66 };

    //    unsigned char bufferDecrypt[16];

        AES128_CBC_decrypt_buffer(bufferDecrypt, bufferEncrypt,  16, key, iv);

        printf("\r\nCBC decrypt: ");
        /*
        if(0 == strncmp((char*) out, (char*) bufferDecrypt, 16))
        {
            printf("SUCCESS!\r\n");
        }
        else
        {
            printf("FAILURE!\r\n");
        }
        */

        unsigned char i;
        printf("\r\nIn function input data: ");
        for(i = 0; i < 1; ++i)
        {
            printStringToHex(in + (i*16));
        }
        printf("\r\n");

        printf("\r\nIn function decrypted data: ");
        for(i = 0; i < 1; ++i)
```

```
        {
            printStringToHex(bufferDecrypt + (i*16));
        }
        printf("\r\n");


    //    return bufferDecrypt;

    }
```

# Appendix II Process menu definition in Coordinator (Encryption)

```
            case '0':
                        printf("\r\nPlease input string: ");
                        /* Load the payload with the data to send */
                        unsigned char encryptStringInput[16];

                        for(i = 0; i < 16; i++)
                        {
                            encryptStringInput[i] = ConsoleGet();
                            ConsolePut(encryptStringInput[i]);
                        }

                        /* String?Hex */
                        //    unsigned char encryptStringInput[] =
                            "helloworldabc";
                    printf("\r\nInput string:");
                    printf(encryptStringInput);
                    printf("\r\n");


                    //unsigned char encryptArrayInput[] = {};
                    //unsigned long len = strlen(encryptStringInput);
```

```
                    int i;
                    for(i = 0; i < 16; i++)
                    {
                        //if(i<len){
                        encryptArrayInput[i] = encryptStringInput[i];
                        //}else{
                                  //encryptArrayInput[i] = 0;
                        //}
                         }
                         printf("\r\nstring to hex (Input array): ");
                         printStringToHex(encryptArrayInput);
                     printf("\r\n");


                    /* AES CBC?? */
                        //??????
                        //    unsigned char encryptOutputArray[16];
                        //????
                        test_encrypt_cbc(encryptArrayInput);
                        //    memcpy(encryptOutputArray,
                            test_encrypt_cbc(encryptArrayInput), 16);
                        //???????
                        printf("\r\nreturned encrypted data: ");
                        printStringToHex(bufferEncrypt);
                        printf("\r\n");



                        /* AES CBC?? */
    /*                  //??????
                        //    unsigned char decryptOutputArray[16];
                        //????
                    test_decrypt_cbc(bufferEncrypt);
                        //    memcpy(decryptOutputArray,
                            test_decrypt_cbc(bufferEncrypt), 16);
                        //???????
                        printf("\r\nreturned decrypted data: ");
                        //printf("%s",bufferDecrypt);
                        for(i = 0; i < 16; i++)
                        {

                            ConsolePut(bufferDecrypt[i]);
                        }
                        printf("\r\n");
    */


                        printf("\r\nSending bufferEncrypt to router...\r\n");
                        for(i = 0; i < 16; i++)
                        {

                            TxBuffer[TxData++] = bufferEncrypt[i];
                        }


                        params.APSDE_DATA_request.DstAddrMode =
                            APS_ADDRESS_16_BIT; // use network address
                        params.APSDE_DATA_request.DstAddress.ShortAddr.v[1] =
                            0x00; //MSB of Dest address
```

```
params.APSDE_DATA_request.DstAddress.ShortAddr.v[0] =
    0x01; //LSB of Dest address
params.APSDE_DATA_request.SrcEndpoint = 171; // set the
    endpoints
params.APSDE_DATA_request.DstEndpoint = 172;
params.APSDE_DATA_request.ProfileId.Val =
    MY_PROFILE_ID; // ignore this
//params.APSDE_DATA_request.asduLength; TxData
params.APSDE_DATA_request.RadiusCounter =
    DEFAULT_RADIUS; // ignore this
params.APSDE_DATA_request.TxOptions.bits.acknowledged = 1
    ; // usd ACK or not
params.APSDE_DATA_request.DiscoverRoute =
    ROUTE_DISCOVERY_SUPPRESS; // select therouting mode

/* Test Secure Send */
params.APSDE_DATA_request.TxOptions.bits.securityEnabled
    = 1;

// ignore the security
#ifdef I_SUPPORT_SECURITY
params.APSDE_DATA_request.TxOptions.Val = 1;
#else
params.APSDE_DATA_request.TxOptions.Val = 0;
#endif
params.APSDE_DATA_request.ClusterId.Val= 0x1111; //set
    the cluster ID
ZigBeeBlockTx(); // block other transmission until finish
currentPrimitive = APSDE_DATA_request; // give the
    command of transmission
break;
```

# Appendix III Message receiving and handling in Router (Decryption)

```
                    case 172:
                        {/*firstly you may want to read the payload
                            first*/









                    // then you can do some processing or other work
                        WORD_VAL clusterID = params.
                            APSDE_DATA_indication.ClusterId;
                       switch(clusterID.Val)
                       {

                            case 0x1111:


                                for(i = 0; i < 16; i++)
        {

                        bufferEncrypt[i] = APLGet();
        }
        printf("\r\nReceived encrypted data: ");
       printStringToHex(bufferEncrypt);
        printf("\r\n");



        /* AES CBC?? */
        //??????
        //    unsigned char decryptOutputArray[16];
        //????
    test_decrypt_cbc(bufferEncrypt);
```

```
//    memcpy(decryptOutputArray,
    test_decrypt_cbc(bufferEncrypt), 16);
//???????
printf("\r\nreturned decrypted data: ");
//printf("%s",bufferDecrypt);
for(i = 0; i < 16; i++)
{

    ConsolePut(bufferDecrypt[i]);
}
printf("\r\n");




                break;
            }
            for(i = 0; i < 16; i++)
            {
                TxBuffer[TxData++] = bufferDecrypt[i];
            }
            /* you may also want to send some data back*/

            params.APSDE_DATA_request.DstAddrMode =
                APS_ADDRESS_16_BIT;  //use network address
            params.APSDE_DATA_request.DstAddress.ShortAddr=
                params.APSDE_DATA_indication.SrcAddress.
                ShortAddr;  // use the source address of
                the received packet as Dest address

            params.APSDE_DATA_request.SrcEndpoint    =
                172;   // set the endpoints
            params.APSDE_DATA_request.DstEndpoint    = 171;
            params.APSDE_DATA_request.ProfileId.Val  =
                MY_PROFILE_ID; // ignore this one

            //params.APSDE_DATA_request.asduLength; TxData
            params.APSDE_DATA_request.RadiusCounter =
                DEFAULT_RADIUS;  // ignore it
            params.APSDE_DATA_request.TxOptions.bits.
                acknowledged = 1;    // use ACK or not
            params.APSDE_DATA_request.DiscoverRoute =
                ROUTE_DISCOVERY_SUPPRESS; //set the routing
                mode

          /* Test Secure Send */
          params.APSDE_DATA_request.TxOptions.bits.
                securityEnabled = 1;

            // ignore the security
            #ifdef I_SUPPORT_SECURITY
            params.APSDE_DATA_request.TxOptions.Val = 1;
        #else
        params.APSDE_DATA_request.TxOptions.Val = 0;
            #endif

        params.APSDE_DATA_request.ClusterId.Val= 0x0002;  //
                set the cluster ID
```

```
                                ZigBeeBlockTx();                                //
                                    block other transmission until finish
                                currentPrimitive = APSDE_DATA_request;          //
                                    give the command of transmission
                                break;
                                }

                            default:
                                break;
                        }
                        APLDiscardRx();
                    }
                    break;
```

# Appendix IV Displaying of plain text in Coordinator

```
        //
              ************************************************
              ************************
            // Place a case for each user defined endpoint.
            //
              ************************************************
              ************************

              case 171:
                       {
                            unsigned char receivedFromRouter[16];
                            WORD_VAL clusterID = params.
                                APSDE_DATA_indication.ClusterId;
                            switch(clusterID.Val)
                            {


                                 case 0x0002:




                                                  for(i = 0; i < 16; i++)
{

    receivedFromRouter[i] = APLGet();
}
printf("\r\nReceived From Router: ");
printStringToHex(receivedFromRouter);
printf("\r\n");



                                                  break;
                            }
                            break;
                       }
```