ISSN:0975-9646

Secure Data in Wireless Sensor Network via AES (Advanced Encryption Standard)

P.D. Khambre 1,S.S.Sambhare[2], P.S. Chavan[1]
[1] *BVUCOE,Pune*
[2]*PCCOE,Pune*

Abstract- **One of the main goals of sensor networks is to provide accurate information about a sensing field for an extended period of time. The emergence of sensor networks as one of the dominant technology trends in the coming decades has posed numerous unique challenges to researchers. Because sensor networks may interact with sensitive data and/or operate in hostile unattended environments, it is imperative that these security concerns be addressed from the beginning of the system design. These networks are likely to be composed of hundreds, and potentially thousands of tiny sensor nodes, functioning autonomously, and in many cases. While the set of challenges in sensor networks are diverse, we focus on security of Wireless Sensor Network in this paper. We propose some of the security goal for Wireless Sensor Network. Further, security being vital to the acceptance and use of sensor networks for many applications; we have made in depth threat analysis of Wireless Sensor Network. We also propose some countermeasures against these threats in Wireless Sensor Network. So, in this paper we have implemented Encryption Algorithm like - AES to provide sufficient levels of security for protecting the confidentiality of the data in the WSN network. This paper also analyzes the performance of AES algorithm against Attacks in WSN Network**.

Keywords- **WSN , Sensor node , Gateway , Security , AES.**

1. INTRODUCTION

Wireless sensor networks are quickly gaining popularity due to the fact that they are potentially low cost solutions to a variety of real-world challenges. Their low cost provides a means to deploy large sensor arrays in a variety of conditions capable of performing both military and civilian tasks. But sensor networks also introduce severe resource constraints due to their lack of data storage and power. Both of these represent major obstacles to the implementation of traditional computer security techniques in a wireless sensor network. The unreliable communication channel and unattended operation make the security defenses even harder. Indeed, as pointed out in wireless sensors often have the processing characteristics of machines that are decades old (or longer), and the industrial trend is to reduce the cost of wireless sensors while maintaining similar computing power. With that in mind, many researchers have begun to address the challenges of maximizing the processing capabilities and energy reserves of wireless sensor nodes while also securing them against attackers. All aspects of the wireless sensor network are being examined including secure and efficient routing, data aggregation, group formation, and so on. In addition to those traditional security issues, we observe that many general-purpose sensor network techniques (particularly the early research) assumed that all nodes are cooperative and trustworthy. This is not the case for most, or much of, real-world wireless sensor networking applications, which require a certain amount of trust in the application in order to maintain proper network functionality. Researchers therefore began focusing on building a sensor trust model to solve the problems beyond the capability of cryptographic security. In addition, there are many attacks designed to exploit the unreliable communication channels and unattended operation of wireless sensor networks. Furthermore, due to the inherent unattended feature of wireless sensor networks, we argue that physical attacks to sensors play an important role in the operation of wireless sensor networks. Thus, we include a detailed discussion of the physical attacks and their corresponding defenses, topics typically ignored in most of the current research on sensor security. We classify the main aspects of wireless sensor network security into four major categories: the obstacles to sensor network security, the requirements of a secure wireless sensor network, attacks, and defensive measures. We also give a brief introduction of related security techniques and summarize the obstacles for the sensor network security. The security requirements of a wireless sensor network are listed as below:

1.1. Obstacles of Sensor Security

A wireless sensor network is a special network which has many constraints compared to a traditional computer network. Due to these constraints it is difficult to directly employ the existing security approaches to the area of wireless sensor networks. Therefore, to develop useful security mechanisms while borrowing the ideas from the current security techniques like (AES).

2. WSN ARCHITECTURE

In a typical WSN we see following network components –

[A]. Sensor motes (Field devices) – Field devices are mounted in the process and must be capable of routing packets on behalf of other devices. In most cases they characterize or control the process or process equipment. A router is a special type of field device that does not have process sensor or control equipment and as such does not interface with the process itself.

[B]. Gateway or Access points – A Gateway enables communication between Host application and field devices.

[C].Network manager – A Network Manager is responsible for configuration of the network, scheduling communication between devices (i.e., configuring super

frames), management of the routing tables and monitoring and reporting the health of the network.
[D].Security manager – The Security Manager is responsible for the generation, storage, and management of keys[5,18,19].

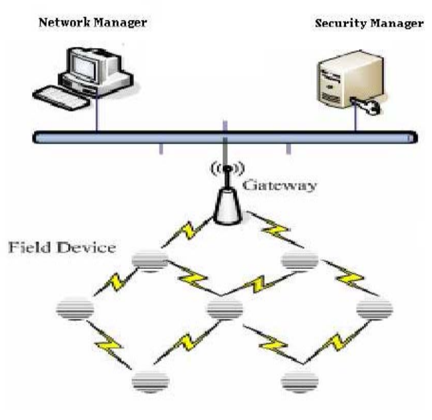


Figure 1 WSN Architecture

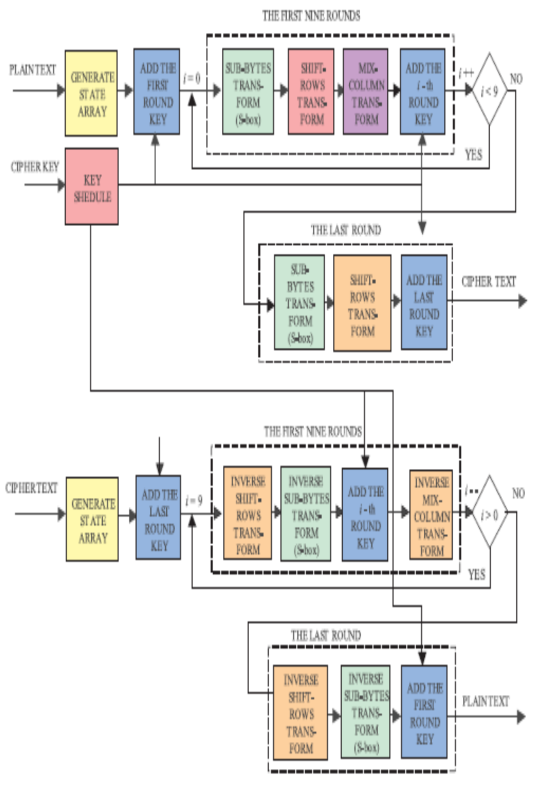
3.WSN SECURITY ANALYSIS

Simplicity in Wireless Sensor Network with resource constrained nodes makes them extremely vulnerable to variety of attacks. Attackers can eavesdrop on our radio transmissions, inject bits in the channel, replay previously heard packets and many more. Securing the Wireless Sensor Network needs to make the network support all security properties: confidentiality, integrity, authenticity and availability. Attackers may deploy a few malicious nodes with similar hardware capabilities as the legitimate nodes that might collude to attack the system cooperatively. The attacker may come upon these malicious nodes by purchasing them separately, or by "turning" a few legitimate nodes by capturing them and physically overwriting their memory. Also, in some cases colluding nodes might have high-quality communications links available for coordinating their attack. Sensor nodes may not be tamper resistant and if an adversary compromises a node, she can extract all key material, data, and code stored on that node. While tamper resistance might be a viable defense for physical node compromise for some networks, we do not see it as a general purpose solution. Extremely effective tamper resistance tends to add significant per-unit cost, and sensor nodes are intended to be very inexpensive.

3.1 AES (Rijndael) Overview

Rijndael (pronounced as in "rain doll" or "rhine dahl") is a block cipher designed by Joan Daemen and Vincent Rijmen, both cryptographers in Belgium. Rijndael can operate over a variable-length block using variable-length keys; the version 2 specification submitted to NIST describes use of a 128-, 192-, or 256-bit key to encrypt data blocks that are 128, 192, or 256 bits long; note that all nine combinations of key length and block length are possible. The algorithm is written in such a way that block length and/or key length can easily be extended in multiples of 32 bits and it is specifically designed for efficient implementation in hardware or software on a range of processors. The design of Rijndael was strongly influenced by the block cipher called square, also designed by Daemen and Rijmen.

Structure Of AES:



Structure of AES algorithm

3.2 In Depth

Rijndael is an iterated block cipher, meaning that the initial input block and cipher key undergoes multiple rounds of transformation before producing the output. Each intermediate cipher result is called a State.
For ease of description, the block and cipher key are often represented as an array of columns where each array has 4 rows and each column represents a single byte (8 bits). The number of columns in an array representing the state or cipher key, then, can be calculated as the block or key length divided by 32 (32 bits = 4 bytes). An array representing a State will have Nb columns, where Nb values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit block, respectively. Similarly, an array representing a Cipher Key will have Nk columns, where Nk values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit key, respectively. An example of a 128-bit State (Nb=4) and 192-bit Cipher Key (Nk=6) is shown below:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

The number of transformation rounds (**Nr**) in Rijndael is a function of the block length and key length, and is given by the table below:

No. of Rounds Nr		Block Size		
		128 bits Nb = 4	192 bits Nb = 6	256 bits Nb = 8
Key Size	128 bits Nk = 4	10	12	14
	192 bits Nk = 6	12	12	14
	256 bits Nk = 8	14	14	14

Now, having said all of this, the AES version of Rijndael does not support all nine combinations of block and key lengths, but only the subset using a 128-bit block size. NIST calls these supported variants AES-128, AES-192, and AES-256 where the number refers to the key size. The **Nb**, **Nk**, and **Nr** values supported in AES are:

	Parameters		
Variant	Nb	Nk	Nr
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

The AES/Rijndael cipher itself has three operational stages:

- AddRound Key transformation
- **Nr**-1 Rounds comprising:
  - SubBytes transformation
  - ShiftRows transformation
  - MixColumns transformation
  - AddRoundKey transformation
- A final Round comprising:
  - SubBytes transformation
  - ShiftRows transformation
  - AddRoundKey transformation

The paragraphs below will describe the operations mentioned above. The nomenclature used below is taken from the AES specification although references to the Rijndael specification are made for completeness. The arrays *s* and *s'* refer to the State before and after a transformation, respectively (**NOTE:** The Rijndael specification uses the array nomenclature a and b to refer to the before and after States, respectively). The subscripts *i* and *j* are used to indicate byte locations within the State (or Cipher Key) array.

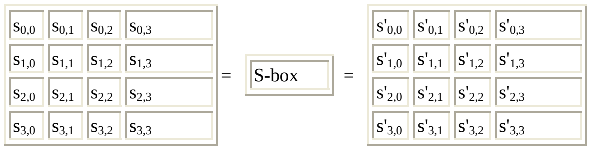
The SubBytes transformation

The substitute bytes (called *ByteSub* in Rijndael) transformation operates on each of the State bytes independently and changes the byte value. An S-box, or *substitution table*, controls the transformation. The characteristics of the S-box transformation as well as a compliant S-box table are provided in the AES specification; as an example, an input State byte value of 107 (0x6b) will be replaced with a 127 (0x7f) in the output State and an input value of 8 (0x08) would be replaced with a 48 (0x30).

One way to think of the SubBytes transformation is that a given byte in State s is given a new value in State s' according to the S-box. The S-box, then, is a function on a byte in State s so that:

$s'_{i,j}$ = S-box ($s_{i,j}$)

The more general depiction of this transformation is shown by:



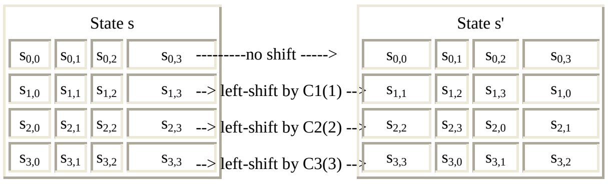
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
=
S-box
=
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

The ShiftRows transformation

The shift rows (called *ShiftRow* in Rijndael) transformation cyclically shifts the bytes in the bottom three rows of the State array. According to the more general Rijndael specification, rows 2, 3, and 4 are cyclically left-shifted by C1, C2, and C3 bytes, respectively, per the table below:

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

The current version of AES, of course, only allows a block size of 128 bits (**Nb** = 4) so that C1=1, C2=2, and C3=3. The diagram below shows the effect of the ShiftRows transformation on State s:



State s
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
---------no shift ----->
--> left-shift by C1(1) -->
--> left-shift by C2(2) -->
--> left-shift by C3(3) -->
State s'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

The MixColumns transformation

The mix columns (called *MixColumn* in Rijndael) transformation uses a mathematical function to transform the values of a given column within a State, acting on the four values at one time as if they represented a four-term polynomial. In essence, if you think of MixColumns as a function, this could be written:

$s'_{i,c}$ = MixColumns ($s_{i,c}$)

for 0<=i<=3 for some column, c. The column position doesn't change, merely the values within the column.

Round Key generation and the AddRoundKey transformation

The AES Cipher Key can be 128, 192, or 256 bits in length. The Cipher Key is used to derive a different key to be applied to the block during each round of the encryption operation. These keys are called the Round Keys and each will be the same length as the block, i.e., **Nb** 32-bit words (words will be denoted W).

The AES specification defines a key schedule by which the original Cipher Key (of length **Nk** 32-bit words) is used to form an *Expanded Key*. The Expanded Key size is equal to the block size times the number of encryption rounds plus 1, which will provide **Nr**+1 different keys. (Note that there are **Nr** encipherment rounds but **Nr**+1 AddRoundKey transformations.)

Expanded Key:	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	W_{15}	...	W_{44}	W_{45}	W_{46}	W_{47}	W_{48}	W_{49}	W_{50}	W_{51}
Round keys:	Round key 0				Round key 1				Round key 2				Round key 3				...	Round key 11				Round key 12			

Consider that AES uses a 128-bit block and either 10, 12, or 14 iterative rounds depending upon key length. With a 128-bit key, for example, we would need 1408 bits of key material (128x11=1408), or an Expanded Key size of 44 32-bit words (44x32=1408). Similarly, a 192-bit key would require 1664 bits of key material (128x13), or 52 32-bit words, while a 256-bit key would require 1920 bits of key material (128x15), or 60 32-bit words. The key expansion mechanism, then, starts with the 128-, 192-, or 256-bit Cipher Key and produces a 1408-, 1664-, or 1920-bit Expanded Key, respectively. The original Cipher Key occupies the first portion of the Expanded Key and is used to produce the remaining new key material.

The result is an Expanded Key that can be thought of and used as 11, 13, or 15 separate keys, each used for one AddRoundKey operation. These, then, are the *Round Keys*. The diagram below shows an example using a 192-bit Cipher Key (**Nk**=6), shown in ***magenta italics***:

The AddRoundKey (called *Round Key addition* in Rijndael) transformation merely applies each Round Key, in turn, to the State by a simple bit-wise exclusive OR operation. Recall that each Round Key is the same length as the block.

SUMMARY

Recall from the beginning of the AES overview that the cipher itself comprises a number of rounds of just a few functions:

- *SubBytes* takes the value of a word within a State and substitutes it with another value by a predefined S-box
- *ShiftRows* circularly shifts each row in the State by some number of predefined bytes
- *MixColumns* takes the value of a 4-word column within the State and changes the four values using a predefined mathematical function
- *AddRoundKey* XORs a key that is the same length as the block, using an Expanded Key derived from the original Cipher Key

As a last and final demonstration of the operation of AES, above Figure is a pseudocode listing for the operation of the AES cipher. In the code:

- *in[]* and *out[]* are 16-byte arrays with the plaintext and cipher text, respectively. (According to the specification, both of these arrays are actually 4***Nb** bytes in length but **Nb**=4 in AES.)
- *state[]* is a 2-dimensional array containing bytes in 4 rows and 4 columns. (According to the specification, this arrays is 4 rows by **Nb** columns.)
- *w[]* is an array containing the key material and is 4*(**Nr**+1) words in length. (Again, according to the specification, the multiplier is actually **Nb**.)
- *AddRoundKey()*, *SubBytes()*, *ShiftRows()*, and *MixColumns()* are functions representing the individual transformations.

```
Cipher (byte in[4*Nb], byte out[4*Nb], word
w[Nb*(Nr+1)])
begin
 byte state[4,Nb]

 state = in

 AddRoundKey(state, w)

 for round = 1 step 1 to Nr-1
  SubBytes(state)
  ShiftRows(state)
  MixColumns(state)
  AddRoundKey(state, w+round*Nb)
 end for

 SubBytes(state)
 ShiftRows(state)
 AddRoundKey(state, w+Nr*Nb)

 out = state
end
```

AES pseudocode.

IMPLEMENTATIONS

- Optimized Software Implementation. The pure software implementation is bounded by the load/store behavior and byte arithmetic of the algorithm. The encryption requires 774 cycles per block on a MIPS32 processor and the decryption requires 837 cycles.
- AES Primitives. This is the simplest form of Vocal's hardware acceleration. The AES Primitives extend the capabilities of the MIPS32 processor by taking advantage of MIPS Technologies CorExtend capability to decrease the number of cycles to 393 cycles to encrypt and 460 cycles to decrypt per block on the MIPS32 processor.
- AES Round Accelerator. The Round Accelerator requires 1024 bytes of local memory, but increases the performance to 117 cycles per block to encrypt and 127 cycles per block to decrypt.
- AES 32-bit Block Accelerator. The Block Accelerator is designed to be a good mid-scale solution. It uses 2048 bytes of local memory. The number of cycles to process a block on a MIPS32 cpu falls to 64 cycles for both encryption and decryption using this implementation.
- AES 32-bit Co-Processor. The Co-Processor implementation uses 2048 bytes of memory to deliver performance of 45 cycles per block on the MIPS32.
- AES 64-bit Co-Processor. The same amount of the memory is required for the 64-bit implementation, but the performance increases to just 25 cycles per block on the MIPS32.

REFERENCES

[1]International journal on "Security issues in wireless sensor networks" Issue 1 volume 2 2008.by Zoran S.Bojkovic,Bojan M.Bakmaz
[2] International journal on "A secure mechanism for data collection in wireless sensor network "by Yusin Xao Issue 5 volume 2 2011
[3] International journal on "Secure data in wireless sensor network via DES" volume 1 issue 2 2011 by Vimal Upadhay,Pintu Kashyap
[4] International journal on "Security threats & issues in wireless sensor networks "Volume 2 issue 1 2012 by D.G.Anand.