International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN: 2278-800X, www.ijerd.com
Volume 7, Issue 5 (June 2013), PP. 40-49

AES-128 Performance in Tinyos with CBC Algorithm (WSN)

Ankit Srivastava, Dr. N. Revathi Venkataraman

Department of Computer Science and Engineering, SRM University, Chennai.
Assistant Professor (S.G). Department of Computer Science and Engineering, SRM University, Chennai.

Abstract:- The wireless sensor network (WSN) is a combination of sensing, computation, and communication into a single tiny device known as Sensor nodes. Sensor data is shared between these sensor nodes and used as input to a distributed estimation system. The wireless network environment that consists of the many devices, called the sensor node, which have miniature computing device, small memory space and very limited battery power. Therefore, Symmetric key encryption algorithm with low-Energy consumption is required to the applicable sensor networks. In this paper, we proposed the solution of reliable sensor networks to analyze the communication efficiency through measuring performance of AES-128 CBC algorithm which is selected by default in sensor networks by plaintext size and cost of operation per hop according to the network scale**.** In this work, both encryption and decryption will be carried out with the key length of 128 bits, that is, both AES encrypter and the AES decrypter were integrated. Hence the input block and secret key will be provided for encryption and the cipher block and same secret key will be provided to the decryption to get the proper block as output.

Keywords:- Wireless sensor network(WSN), AES(Advance Encryption Standards),Motes.

I. INTRODUCTION

In today's world most of the communication is done using electronic media. Data Security plays a vital role in such communication. Hence, there is a need to protect data from malicious attacks. Cryptography is the science of secret codes, enabling the confidentiality of communication through an insecure channel. It protects against unauthorized parties by preventing unauthorized alteration of use. Generally speaking, it uses a cryptographic system to transform a plaintext into a cipher text, using most of the time a key.

Advanced Encryption Standard (AES), also known as Rijndael, is an encryption standard used for securing information. AES was published by NIST (National Institute of Standards and Technology). AES is a Chain Block Cipher algorithm that has been analysed extensively and is now used widely. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. The block cipher Rijnddael was designed by Dr. Joan Daemen and Dr. Vincent Rijmen and the name of the algorithm is a combination of the names of its two creators. Rijndael is very secure and has no known weakness. Rijndael is conventional (symmetric key) system and is relatively simple cipher in many respects. It takes an input block of a certain size, usually 128, and produces a corresponding output block of the same size. The transformation requires a second input, which is the secret key. It is important to know that the secret key.

In this work, both encryption and decryption will be carried out with the key length of 128 bits on TinyOS platform, that is, both AES encrypter and the AES decrypter were integrated. Hence the input block and secret key will be provided for encryption and the cipher block and same secret key will be provided to the decryption to get the proper block as output.

The Advanced Encryption Standard (AES) may configured to use different key-lengths, the standard defines 3 lengths and the resulting algorithms are named AES-128, AES-192 and AES-256 respectively to indicate the length in bits of the key. Each additional bit in the key effectively doubles the strength of the algorithm, when defined as the time necessary for an attacker to stage a brute force attack, i.e. an exhaustive search of all possible key combinations in order to find the right one. The key length of 128 bits is used in process of encryption. The AES algorithm is a block cipher that uses the same binary key both to encrypt and decrypt data blocks is called a symmetric key cipher. A commonly accepted definition of a good symmetric key algorithm, such as the AES, is that there exists no attack better than key exhaustion to read an encrypted message.

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. AES algorithm is a symmetric block cipher that can encrypt (encipher) and

decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher-text; decrypting the cipher-text converts the data back into its original form, called plaintext, as illustrated in Figure1.

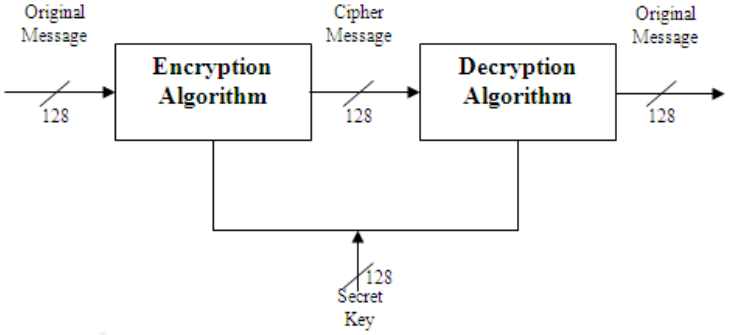

Figure 1 Overall Representations of Encryption and Decryption

II. RELATED WORK

Security aspects in wireless sensor networks are getting more and more attention in the recent years. Wireless sensor networks are being considered for critical applications where the Security issues are a priority. The limited capacities of wireless sensor nodes and the complex algorithms of the security protocols make the subject challenging. In order to provide security in wireless sensor networks, communications should be encrypted and authenticated. The main issue is how to set up secret keys between nodes to be used for the cryptographic operations which are known as the key agreement.

Unfortunately, security is in general considered to be expensive. Its cost is even more noticeable in WSNs due to the limited resources of the sensor nodes. Thus, in order to provide a sufficient level of security while properly utilizing the available resources, it is important to have a good understanding of both the cost and the features of the security algorithms used. For example, if a sensor device does not have enough available memory to run a specific security protocol, it might be better to use an alternative algorithm which requires less memory but might be more secure. This work's contributions are threefold.

First, we have analysed AES, RC5, Skipjack, and XXTEA in terms of their energy consumption and memory requirements for MicaZ and TelosB motes. We have studied how the different algorithm parameters (e.g. key size) influence the energy consumption. Further, based on our study of AES, we have evaluated the tradeoffs between using hardware and software implemented security algorithms. The second contribution is the analysis of a number of block cipher modes of operations and how suitable they are for use in WSN applications. Although several papers, such as [1, 2], have included operation modes in their experiments, few of them have focused on the modes themselves.

Chang et al. [3, 4] use a setup similar to ours to measure the energy consumption introduced by hash functions and symmetric-key algorithms on Mica2 motes with a CC1000 radio and Ember sensors with an EM2420 radio. They use a Pico Scope 3206 oscilloscope to sample the voltage drop across two registers and, based on the measured data, speculate about the energy consumption. They study a different subset of symmetric-key algorithms. In addition, since they run their experiments on Mica2 motes, the results they provide cannot be used as a reference for MicaZ or TelosB motes.

Law et al. [1] analyse the influence of block ciphers on WSNs and compare the MCU cycles and memory used by encryption algorithms. They have studied multiple algorithms (Skipjack, RC5, RC6, MISTY1, Rijndael, Twofish, KASUMI, and Camellia).

They have also measured the memory usage of the standard modes of operation (CBC, CFB, OFB, and CTR) but have not evaluated their energy consumption. In addition, they do not specify the platform they use in their experiments. Compared to their work, we provide a more detailed evaluation of both security algorithms and modes of operation.

As we mentioned earlier, none of the previous work has studied the impact of the different algorithm parameters (e.g. key size) on the energy consumption of AES and RC5. In addition, although hardware implemented AES-128 is expected to be extremely secure, its energy consumption has not been measured. Further, no previous work has evaluated the energy and memory requirements of MAC algorithms.

2.1 EXISTING SYSTEM

Existing approaches difficulty in the Delay for encryption, and memory size. The algorithm like RC5, RC6 and DES takes more time for Encryption and also consumes more Energy. If users send the information from source to destination for RC5, RC6 it has some disadvantages as latency, throughput, packet delivery ratio. For DES there is also some disadvantages like this is chiefly due to the 56-bit key size being too small, in January, 1999 a group of computer experts collaborated to publicly break a DES key in 22 hours and 15 minutes. The key length should thus be chosen after deciding for how long security is required, and what the

cost must be to brute force a secret key. In some military circumstances a few hours or days security is sufficient after that the war or the mission is completed and the information uninteresting and without value. In other cases a lifetime may not be long enough.

III. PROPOSED TECHNIQUES

AES is a symmetric encryption algorithm processing data in block of 128 bits. A bit can take the values zero and one, in effect a binary digit with two possible values as opposed to decimal digits, which can take one of 10 values. Under the influence of a key, a 128-bit block is encrypted by transforming it in a unique way into a new block of the same size.

The AES is an iterated block cipher with a fixed block size of 128 and a variable key length. The different transformations operate on the intermediate results, called state. The state is a rectangular array of bytes and since the block size is 128 bits, which is 16 bytes, the rectangular array is of dimensions 4x4. The basic unit for processing in the AES algorithm is a byte, a sequence of eight bits treated as a single entity. The input, output and Cipher Key bit sequences which are processed as arrays of bytes that are formed by dividing these sequences into groups of eight contiguous bits to form arrays of bytes.

In the Rijndael version with variable block size, the row size is fixed to four and the number of columns varies. The number of columns is the block size divided by 32 and denoted Nb. The cipher key is similarly pictured as a rectangular array with four rows. The number of columns of the cipher key, denoted Nk, is equal to the key length divided by 32. AES uses a variable number of rounds, which are fixed: A key of size 128 has 10 rounds, also the basic inputs to the system and the outputs from the system were clearly represented. As per the standard, 10 rounds for 128 bits key length were carried out in which the last round will be performed separately. For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations:

- Byte substitution using a substitution table (S-box)
- Shifting rows of the State array by different offsets
- Mixing the data within each column of the State array
- Adding a Round Key to the State

Above mentioned functions were carried out for every individual round and in the last round the third function, that is, Mixing the data within each column of the State array will not be performed. Hence the last round is carried out separately. Based on the key provided, the new set of keys will be generated in the Key Expansion block and is given to the each round as input.

IV. SOLUTION ARCHITECTURE

4.1 AES CIPHER

AES algorithm can be defined by the following flow chart. Figure 2, Data flow diagram of AES Encrypter At the start of the Cipher, 128 bits input is copied in the form of State array. This is converted into cipher text by the repeated application of round functions. Value of round function is dependent on cipher key length and is equal to Nr-1. The round function is composed of four transformations of substitution, shifting, mixing the column and adding of round key. The encryption process is initialized by adding the first Round Key, followed by applying the round function Nr-1 times. Last round is different from others which perform substitution, shifting and adding the round key operations while mix column operation is not performed in last round. In following section all transformations are defined. Round Key Addition: Exclusive OR (XOR) operation between the state array and the Round Key. SubBytes Transformation: SubBytes is a 16 Byte input/output nonlinear transformation that uses 1 Byte Substitution table (S-Boxes). ShiftRows Transformation: In this transformation, second, third, and fourth rows are shifted one byte, two bytes, and three bytes to the left, respectively. MixColumns Transformation: It operates on the State column-by-column, treating each column as a four-term polynomial , The columns are considered as polynomials over GF(28) and multiplied modulo x4 + 1 with a fixed polynomial a(x).

4.2 AES ENCRYPTION FLOW CHART

In the flow chart (Figure 2) of AES encryption algorithm round counter for 128 bit is 10. Then all other operation like key addition, S-Table Substitution, Encode Row Shift, Encode Mix Column are performed.

- To implement the AES encryption algorithm, we proceed exactly the same way as for the key expansion, that is, we first implement the basic helper functions and then move up to the main loop. The functions take as parameter a *state*, which is, as already explained, a rectangular 4x4 array of bytes.
- The shiftRows function iterates over all the rows and then call shiftRow with the correct offset. shiftRow does nothing but to shift a 4-byte array by the given offset.
- This is the part that involves the roundKey was generated during each iteration. Here simply XOR each byte of the key to the respective byte of the state.

- The MixColumns implementation was carried out by first one would generate a column and then call mixColumn, which would then apply the matrix multiplication.
- As you can see in the theory, one AES round is the one which has to apply all four operations on the state consecutively. All we have to do is take the state, the ExpandedKey and the number of rounds as parameters and then call the operations one after the other.
- Finally, all we have to do is put it all together. Our parameters are the input plaintext, the key of size keySize and the output. First, we calculate the number of rounds based on they keySize and then the expandedKeySize based on the number of rounds. Then we have to map the 16 byte input plaintext in the correct order to the 4x4 byte state (as explained above), expand the key using our key schedule, encrypt the state using our main AES body and finally un-map the state again in the correct order in order to get the 16 byte output cipher text.

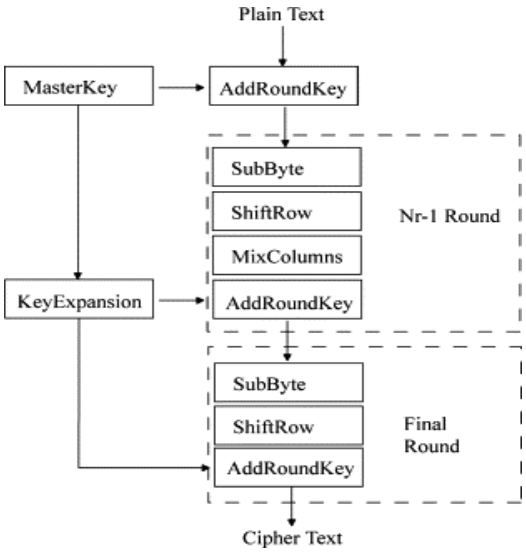

Figure 2 Data flow diagram of AES Encrypter.

V. MODULE DESCRIPTION

AES is based on a secret key with 128, 192 or 256 bits. But if the key is easy to guess it doesn't matter if AES is secure, so it is as critically vital to use good and strong keys as it is to apply AES properly. Creating good and strong keys is a surprisingly difficult problem and requires careful design when done with a computer. The challenge is that computers are notoriously deterministic, but what is required of a good and strong key is the opposite – unpredictability and randomness. Keys derived into a fixed length suitable for the encryption algorithm from passwords or pass phrases typed by a human will seldom correspond to 128 bits much less 256. To even approach 128--bit equivalence in a pass phrase, at least 10 typical passwords of the kind frequently used in day-to-day work are needed.

Weak keys can be somewhat strengthened by special techniques by adding computationally intensive steps which increase the amount of computation necessary to break it. The risks of incorrect usage, implementation and weak keys are in no way unique for AES; these are shared by all encryption algorithms. Provided that the implementation is correct, the security provided reduces to a relatively simple question about how many bits the chosen key, password or pass phrase really corresponds to.

Unfortunately this estimate is somewhat difficult to calculate, when the key is not generated by a true random generator.

The procedures through which the AES algorithm will be processed are as follows:

- Encryption
- Key expansion
- decryption

5.1 ENCRYPTION

At the start of the Encryption or Cipher, the input data and the input key were copied to the State array using the conventions. Initially the XOR operation should be performed between each byte of the input data and the input key and the output will be given as the input of the Round-1. After an initial Round Key addition, the State array is transformed by implementing a round function 10 times, with the final round differing slightly from the first Nr–1 rounds. The final State is then copied to the output.

The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine.
The individual transformations that carried out are listed below.

- Sub Bytes
- Shift Rows
- Mix Columns.
- Add Round Key.

5.2 KEY EXPANSION

Prior to encryption or decryption the key must be expanded. The expanded key is used in the **Add Round** Key function defined above. Each time the Add Round Key function is called a different part of the expanded key is XORed against the state. In order for this to work the Expanded Key must be large enough so that it can provide key material for every time the Add Round Key function is executed. The Add Round Key function gets called for each round as well as one extra time at beginning of the algorithm.

The AES algorithm takes the Cipher Key, ***K***, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of ***Nb*** (***Nr*** + 1) words: the algorithm requires an initial set of ***Nb*** words, and each of the ***Nr*** rounds requires ***Nb*** words of key data. The resulting key schedule consists of a linear array of 4-byte words. Since the key size is much smaller than the size of the sub keys, the key is actually "stretched out" to provide enough key space for the algorithm. Hence an 128 bit key is expanded to an 176 byte key.

There is a relation between the cipher key size, the number of rounds and the ExpandedKey size. For an 128-bit key, there is one initial AddRoundKey operation plus there are 10 rounds and each round needs a new 16 byte key, therefore we require 10+1 RoundKey of 16 byte, which equals 176 byte. An iteration of the above steps is called a round. The amount of rounds of the key expansion algorithm depends on the key size.

Key Size (bytes)	Block Size (bytes)	Expansion Algorithm Rounds	Expanded Bytes / Round	Rounds Key Copy	Rounds Key Expansion	Expanded Key (bytes)
16	16	44	4	4	40	176

Table 1 Key Expansion

The first bytes of the expanded key are always equal to the key. If the key is 16 bytes long the first 16 bytes of the expanded key will be the same as the original key. If the key size is 32 bytes then the first 32 bytes of the expanded key will be the same as the original key. Each round adds 4 bytes to the Expanded Key. With the exception of the first rounds each round also takes the previous rounds 4 bytes as input operates and returns 4 bytes.

5.3 DECRYPTION

The cipher text of 128 bits and the same key of 128 bits will be given as the input to the decryption block. The encrypted data will be decrypted and the original plain message will be achieved as the output of the decryption block. The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher were listed as follows.

- InvShiftRows-Rows
- InvSub-Bytes
- InvMix-Columns
- Add Round-Key

Here also 10 rounds will be carried out and the only difference in the decryption block with respect to the algorithm flow is that the result of the Key Expansion of each round will also be given to the Mix-Columns operation after which the Add Round Key transformation should be carried out. **InvMix-Columns (state XOR Round Key) = InvMix-Columns (state) XOR InvMix-Columns (Round Key)** the above equation represents the basic difference in the process of the AES Encryption and Decryption algorithm.

VI. OPERATIONAL ENVIRONMENT

This project can run in TinyOS environment with Telosb motes. Telosb motes are very tiny device with very limited memory size. One of the most important advantage of using telosb motes is, its low power consumption.

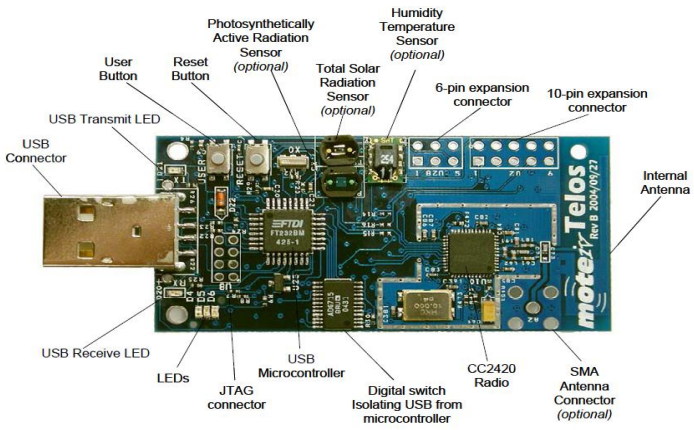

Figure 3 Front side of telosb mote showing its components

6.1 HARDWARE REQUIREMENTS

- Interoperability with other IEEE 802.15.4 devices.
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash).
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller.
- Integrated onboard antenna with 50m range indoors / 125m range outdoors.
- Integrated Humidity, Temperature, and Light sensors.
- Ultra low current consumption.
- Hardware link-layer encryption and authentication.
- Programming and data collection via USB.

6.2 SOFTWARE REQUIREMENTS

- TinyOS
- TinyOS Simulator (TOSSIM)
- Windows XP
- Windows 7
- VM Ware player

VII. IMPLEMENTATION OF CTP(COLLECTION TREE PROTOCOL)

Data collection has been an important area of concern in a wireless sensor network. For data collection in a sensor network environment collection tree protocol was designed. Collection tree protocol is address free. Collection tree protocol works with the assumption that the data link layer provides efficient local broadcast address, synchronous acknowledgements for unicast packets, protocol dispatch field and a single hop source and destination field. Moreover an estimate of link quality of some number of nearby neighbouring nodes is also assumed.

A number of sources can send data to a single sink using this routing protocol. Once data has been collected at the sink, it can be stored, Recorded and will provide the way for future analysis. The data collected can also be immediately processed to react to certain events based on application requirements. Since CTP considers the link quality of the nearby neighbouring nodes, the protocol also works with the assumption about the expected number of transmissions a node will take to send a uni-cast packet to the sink.

CTP uses a set of beacon messages for construction of the tree topology and data messages to report to sink. In particular, application-level modules can call a generic collection service which is in turn implemented through CTP. The standard implementation of CTP consists of three main logical software components, the Routing Engine (RE), the Forwarding Engine (FE), and the Link Estimator (LE).

7.1 ROUTING ENGINE

The Routing Engine, an instance of which runs on each node, takes care of sending and receiving beacons as well as creating and updating the routing table. This table holds a list of neighbours from which the node can select its parent in the routing tree.

The table is filled using the information extracted from the beacons. Along with the identifier of the neighbouring nodes, the routing table holds further information, like a metric indicating the “quality” of a node as a potential parent. In the case of CTP, this metric is the ETX (Expected Transmissions) which is communicated by a node to its neighbours through beacons exchange. A node having an ETX equal to n is

expected to deliver a data packet to the sink with a total of n transmissions. The ETX of a node is defined as the "ETX of its parent plus the ETX of its link to its parent" .

More precisely, a node first computes, for each of its neighbours, the link quality of the current node-neighbour link. This metric, which is referred to as the 1-hop ETX, or ETX1hop, is computed by the LE. For each of its neighbours the node then sums up the 1-hop ETX with the ETX the corresponding neighbours had declared in their routing beacons. The result of this sum is the metric which has been called the multi-hop ETX, or ETXmhop. Since the ETXmhop of a neighbour quantifies the expected number of transmissions required to deliver a packet to a sink using that neighbour as a relay, the node clearly selects the neighbour corresponding to the lowest ETXmhop as its parent. The value of this ETXmhop is then included by the node in its own beacons so as to enable lower level nodes to compute their own ETXmhop. Clearly, the ETXmhop of a sink node is always 0.

7.2 FORWARDING ENGINE

The Forwarding Engine takes care of forwarding data packets which may either come from the application layer of the same node or from neighbouring nodes. The forwarding
Engine is also responsible of detecting and repairing routing loops as well as suppressing duplicate packets.

7.3 LINK SIMULATOR

The Link Estimator takes care of determining the inbound and outbound quality of 1-hop communication link. As mentioned before, reference to the metric that expresses the Quality of such links as the 1-hop ETX has been drawn. The LE computes the 1-hop ETX by collecting statistics over the number of beacons received and the number of successfully transmitted data packets. From these statistics, the LE computes the inbound metric as the expected number of transmission attempts required by the neighbour to successfully deliver a beacon.

VIII. TOPOLOGY

Network topology is the arrangement of the various elements (links, nodes etc.) of a computer essentially, it is the topological structure of a network, and may be depicted physically or logically. Physical topology refers to the placement of the network's various components, including device location and cable installation. The topology used for this experiment is tree topology. Design a topology is necessary for sending and receiving packet efficiently. Tree Topology integrates the characteristics of

- Star and Bus Topology.

IX. PERFORMANCE METRICS

To evaluate a particular protocol, a set of performance metrics are needed. For analyze the performance of Advance encryption standard algorithm the following performance metrics has been taken.

9.1 PACKET DELIVERY RATIO

Packet delivery ratio is the ratio of the total number of packets delivered to the total number of packet sent by the sender. It's calculated under different data rate and radio frequency power. Packet delivery ratio also takes into account the number of data values the network can send to the sink. If the packet delivery ratio equals to 1 then it can be said that all the packets has been delivered successfully by the network. In the worst case, none of the collected data values reaches the sink. This may happen if the sink is disconnected from the network and causes the packet delivery ratio to be zero. Packet delivery ratio being a very important performance measurement metrics, it has been calculated under varying channel frequency and variable radio frequency power as well Packet delivery ratio = Packet received / Packet received.

9.2 THROUGHPUT

Throughput or network throughput is the average rate of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. If a packet arrives at the sink node, then the counter increments by 1 and thus the value is printed in the screen along with the node id. The counter value is depicted in hexadecimal format. With each node id one counter value is associated. . In sensor network throughput can be measured by number of packets successfully received by the sink node in a particular time limit.

9.3 ENERGY CONSUMPTION

Energy consumption can be defined as total energy consumed by each sensor node. Energy can be calculated by the formula written below,

Energy = [voltage (v)*current (I)*packet size] / Source data rate.
Generally voltage = 3 volt. Current = 1 Ampere.
The unit of energy consumption is Joule.

9.4 MEMORY CONSUMPTION

Computational overhead is generally consider any combination of excess or indirect computation time, memory, or other resources that are required to be utilized or expended to enable a particular goal. Computational overhead in TinyOS can be calculated as total number of RAM and ROM used in bytes.

X. PERFORMANCE ANALYSIS

10.1 PACKET DELIVERY RATIO

To analyse the ability of AES to send packets from source to sink either directly or through forwarding node, ten nodes has been taken with one node acting as a sink node and others are source nodes. Different data rare has been taken for this analysis. Figure 4 shows the .Comparison of packet delivery ratio between AES, Blowfish and RC5. Analysis result shows that packet delivery ratio decreases when data rates increase. In this figure when data rate is 0.5 packet/second packet delivery ratio is 98 percent for Blowfish and 97 percent for RC5 and 93-94 percent for AES. But when data rate increases to 10 packets/second Packet delivery ratio decreased to 88 percent for Blowfish and 87 percent for RC5 and 84-85 percent for AES

Figure 4 Comparison of packet delivery ratio with source data rate.

10.2 ENERGY CONSUMPTION

Figure 5 shows the energy with various radio frequency power. analysis result shows that when the radio frequency power increases then the energy power also increases, so that the radio frequency power is directly propositional to the energy consumption. the value of energy is in joules and radio power frequency is in dbm

Figure 5 Comparison of Energy with Radio frequency power

10.3 THROUGHPUT

To know the ability of packet received at sink node in a particular time period for AES, Blowfish and RC5, ten motes have been taken. Packets send by the sender node with various source data rate for ten seconds. And packets received by the sink node have been shown in Figure 6. And packets received by the sink node in a particular time are known as throughput. Analysis result shows that throughput is almost same for Blowfish and RC5. Throughput at sink node has been calculated by deducting the number of lost packets in the networks from the number of sent packets by the sender node in a particular time period. Throughput actually measures the efficiency of an algorithm. Various source data rate has been taken, to know the scalability of the algorithms. For example, In AES when data rate is 10 packets / second then in ten seconds ten packets are sent by the sender node, but unfortunately 11 packets have lost in middle of the network, thus eighty nine packets has been received by the sink node. So, throughput at 10 packets / second data rate for ten seconds is eighty nine.

Figure 6 Comparison of throughput with source data rate

10.4 MEMORY CONSUMPTION

Memory consumption is a very important performance metric, to know the efficiency of a particular algorithm. Wireless sensor network is resource constraints (limited memory size, limited battery life), so those security algorithms which have less overhead, will be more suitable for wireless sensor networks. Memory consumption can be defined as total number of memory used in bytes for RAM and ROM. If RAM and ROM value is more, then overhead is also more. Figure 7 shows the RAM and ROM used in bytes for AES, Blowfish and RC5. Analysis result shows that RAM used in Blowfish and RC5 is almost equal, but ROM used in AES, Blowfish is less than RC5. Experiment result shows that ROM used in AES is varied from 25000-30000 bytes, and in RC5 ROM memory is varied from 30000-35000 bytes. ROM used in Blowfish is less, because after encryption blowfish takes less number of bytes in memory and as AES, Blowfish complexity is less than RC5, RAM used in bytes for AES, Blowfish is less than RC5. In idle condition RC5 takes 28000-30000 bytes memory. Thus memory consumption is less in AES.

Figure 7 Comparison of memory used with source data rate

THE OVERALL ANALYSIS RESULT HAS BEEN SHOWN IN TABULATED FORMAT IN TABLE 2. FROM THIS TABLE

Performance Metrics	Security Schemes		
	AES	Blowfish	RC5
Average Packet Delivery Ratio	0.87	0.88	0.87
Average Throughput	87%	88%	87%
Average Energy Consumed in joules	.067	-	-
Memory Consumed in bytes (ROM)	27600	28000	33000
Memory Consumed in bytes (RAM)	3500	3700	3850

Table 2 Overall analysis result of AES, Blowfish and RC5

XI. CONCLUSION

Firstly, understanding the concept of cryptology and flow of AES algorithm is done. Successful implementation of AES algorithm, make to know one of the encryption and decryption standard available in market and it helps to explore the path to implement such an algorithm using TinyOS. Mainly, the concept of instantiation and arrays plays a major part in implementation.

This is a 128-bit Key dependent algorithm which has control over the 128-bit input data or plaintext. The original message is taken to 10 round operations which produces the cipher text. This resultant encrypted data is fed as the input to the decryption and 10 rounds operations were carried out and hence the same plain text is achieved.

Given the same input key and data (plaintext or cipher text) any implementation that produces the same output (cipher text or plaintext) as the algorithm specified in this standard is an acceptable implementation of the AES.

REFERRENCE

[1]. csrc.nist.gov/groups/ST/toolkit/documents/kms/key-wrap.pdf, AES key wrap specification 16 November 2001

[2]. Yee Wei Law, Jeroen Doumen, Pieter Hartel, Survey and benchmark of block ciphers for wireless sensor networks, ACM Transactions on Sensor Networks (2006) 65–93.

[3]. Devesh Jinwala, Dhiren Patel, Kankar Dasgupta, Optimizing the block cipher and modes of operations overhead at the link layer security framework in the wireless sensor networks, in: ICISS 2008, pp. 258–272.

[4]. Chih-Chun Chang, Sead Muftic, David J. Nagel, Measurement of energy costs of security in wireless sensor nodes, in: ICCCN 2007, pp. 95–102.

[5]. Chih-Chun Chang, David J. Nagel, Sead Muftic, Balancing security and energy consumption in wireless sensor networks, Mobile Ad-Hoc and Sensor Networks (2007) 469–480.]

[6]. Jongdeog Lee a, Krasimira Kapitanova b, Sang H. Son b a Department of Electronics Engineering and Information Science, Korea Military Academy, Seoul, Republic of Korea Department of Computer Science, University of Virginia, Charlottesville, United States

[7]. Daemen, J. & Rijmen, V., 2002 – 'The Design of Rijndael – AES the Advanced Encryption Standard', Springer.

[8]. Bernstein, D.J., 1998 - 'Cache-timing attacks on AES' [online]. Available from:

[9]. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf [Accessed 1st May 2008].

[10]. Gaurav Jolly, Mustafa C. Kusçu, Pallavi Kokate, Mohamed Younis, A low-energy key management protocol for wireless sensor networks, in: ISCC 2003, pp. 335–340.

[11]. David W. Carman, Peter S. Kruus, Brian J. Matt, Constraints and approaches for distributed sensor network security, NAI Labs, The Security Research Division, 2000, no 00-010.