Department of Computer Science and Engineering (CSE)
Islamic University of Technology

Lightweight AES Algorithm for Resource Constraint Devices

Authors

Raiyan Rahman Chowdhury – 134424
and
Marjuk Alvy – 134422

Supervisor

Dr. Abu Raihan Mostofa Kamal
Associate Professor
Department of CSE
Islamic University of Technology

A thesis submitted to the Department of CSE
in partial fulfillment of the requirements for the degree of B.Sc.
Engineering in CSE
Academic Year: 2016-17
November – 2017

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Raiyan Rahman Chowdhury and Marjuk Alvy under the supervision of Dr. Abu Raihan Mostofa Kamal, Associate Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Raiyan Rahman Chowdhury
Student ID - 134424

Marjuk Alvy
Student ID - 134422

Supervisor:

Dr. Abu Raihan Mostofa Kamal
Associate Professor
Department of CSE
Islamic University of Technology

Acknowledgement

We would like to express our grateful appreciation for Dr. Abu Raihan Mostofa Kamal, Associate Professor, Department of Computer Science & Engineering, IUT for being our advisor and mentor. His motivation, suggestions and insights for this thesis have been invaluable. Without his support and proper guidance this research would not have been possible. His valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

We are also grateful to Mr. Tariq Tonmoy, M.Sc student & Lecturer, Department of Computer Science & Engineering, Islamic University of Technology for his valuable suggestion.

Abstract

Our world is slowly approaching brink of mankind's next technological revolution since the conception of Internet of Things (IoT) emerges. The realization of IoT requires an enormous amount of sensor nodes which have limited battery power, memory, computational latency and communication bandwidth to acquire inputs from the connected objects. In the current developments of the resource constraint environments, the trend is shifted towards lightweight cryptographic algorithm. Many lightweight cryptographic algorithms have been developed and also existing algorithms are modified in terms of resource constraint environment. In this paper we analyzed a very popular block cipher AES-128 and tried to make it lightweight regarding energy consumption. We also studied some other lightweight block ciphers which are optimized in terms of energy, power and memory.

Contents

Introduction:

Overview:

A dramatical change and development has been occurred since the buzzword "Internet" has been introduced by Sir Tim Berners-Lee on march 12,1989. Ever since then, the Internet has been integrated into ours day to day lives as an essential need at an incredible pace. The geographical barriers between nations has destroyed and the concept of "globalization" has emerged. More importantly, it became the basis of mankind's technological advancement.
Internet of Things(IoT) is the next revolution of Internet which brings such an impact on our everyday lives. IoT is the extension of the Internet to connect just about everything on the planet. This includes real and physical objects ranging from household accessories to health monitoring sensors to industrial controls. As such, these "things" those are connected to the Internet will be able to take actions or make decisions based on the information they obtained from the Internet with or without human intervention. In addition, they also update the Internet with real time information with the help of various sensors. This allows objects connected to the IoT to communicate and work with each other seamlessly.
IoT works with resource constraint components. Those components have limited amount of battery power, memory. Also those component communicate through the Internet which is not secured. As mentioned previously, IoT relies heavily on the sensor nodes on the connected objects to obtain real time information. In certain applications, confidentiality of such information might be extremely important. Therefore, cryptography comes into play.

Due to the resource constrained nature of the sensors, the security requirement has led to various attempts to optimize established cryptographic ciphers for small scale applications. Marginal amount of security is needed for lightweight device like sensors. The most notifying aspect of lightweight device is it's computational speed, battery power.

In this paper, we provide a hardware implementation of Standalone AES Encryption in CC2420(TinyOS 2.1.2 and telosb). We analyze the energy consumption result by sending temperature value which is sensed by telosb from one mote to other with and without encryption.

Problem Statement:

Internet of Things(IoT) devices are lightweight and resource constraint. Those components have limited amount of battery power, memory. Security issue comes when IoT device communicate with each other. Dierent cryptographic algorithm are used to secure the data transmission. But due to resource constraint issue, all standard encryption decryption algorithms can't be used. Many lightweight cryptographic algorithms have already proposed which provides good result regarding energy consumption. We are proposing a light version of standard block cipher AES-128 for IoT devices.

Motivation:

If we classify cryptographic algorithms, we will find to major classifications.
-Symmetric Crypto System
-Asymmetric Crypto System

A secret key algorithm (sometimes called a symmetric algorithm) is a cryptographic algorithm that uses the same key to encrypt and decrypt data. The keys represent a shared secret between two or more parties that can be used to maintain a private information link. The main drawback of Symmetric Crypto System is that both parties have access to the secret key. Symmetric-key encryption can use either stream ciphers or block ciphers. Some important block ciphers are AES, Blowfish DES (Internal Mechanics, Triple DES) etc. Widely used stream ciphers are RC4, Block ciphers in stream mode etc.
Public key cryptography, or asymmetric cryptography, is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, which is when the public key is used to verify that a holder of the paired private key sent the message, and encryption, whereby only the holder of the paired private key can decrypt the message encrypted with the public key. Some well regarded Asymmetric Crypto Systems are Difie{Hellman key exchange protocol, DSS (Digital Signature Standard), Elliptic Curve Crypto System, RSA encryption algorithm etc.

Some of the major motivation and scope are enlisted below:
- Resource constraint components have limited amount of battery power, memory.
-Optimization of AES-128 as lightweight block cipher.
-Secure data transmission between lightweight devices.

Research challenges:

We are working with AES-128 algorithm which is used in resource constraint devices. Though it is energy efficient in comparison to other algorithms, yet we need to face lots of challenges which modifying the algorithm. The core challenges are enlisted below:

- Huge computational power due to number of rounds according to key size
- Energy consumption issue
- S-box and Mix-column operation

Thesis Outline:

In Chapter 1 we have discussed our study in a precise and concise manner. Chapter 2 deals with the necessary literature review for our study and their development so far. In Chapter 3 we have described the original Methodology of AES. In Chapter 4 we have discussed about the original S-Box generation method and in following chapter 5 we have stated our proposed methodology of generating S-Box. In chapter 6 we have discussed about our analyzed result with the final segment of this study which contains all the references and credits used.

Literature review:

Regarding Resource constraint environment, many renowned research works have been done in recent years. Many lightweight cryptographic algorithms have been proposed targeting low area and low power. Some standard block ciphers are also modi_ed for resource constraint environment. Despite a large number of previous works targeting area and power, only limited were devoted to the optimization of the energy parameter. Energy and power are correlated parameters. Power is the amount of energy consumed per unit time. The time integral of power is measured as energy consumption.

1. Energy Efficiency of Lightweight Block Cipher
2. Energy Consumption of Lightweight Block ciphers in FPGA
3. A PUF-Based Paradigm for IoT Security
4. IoT security

1. Energy Efficiency of Lightweight Block Cipher:

Focusing on block ciphers in particular, it is important to notice that AES still remains the preferred choice for providing security also in constrained devices, even if some lightweight algorithms are now standardized. Several implementations of AES and its basic transformations (such as S-boxes) targeting low area and low power were proposed in the past, for example, the implementation of Feldhofer et al.[6] and the one of Moradi et al.[7]. The first design is based on a 8-bit data path, and occupies approximately 3400 Gate Equivalents (GE). The second design features a mixed data path and requires approximately 2400 GE. Both researches are hardware based. Hocquet et al.[8] showed that by exploiting technological advances and algorithmic optimization the AES core, energy can be consumed as little as 740 pJ per encryption. This implementation is also hardware based.

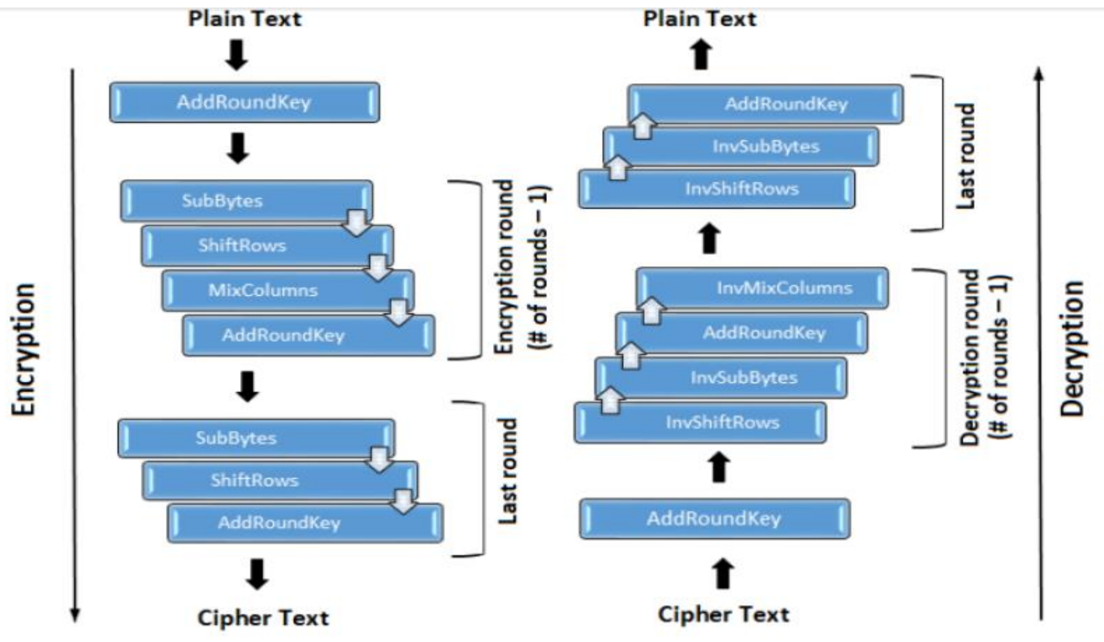
The most significant previous works on this area are the one of Kerckhof et al.[9] and the one of Batina et al.[10]. The first work, addresses the problem of efficiency for lightweight designs. The authors present a comprehensive study comparing a number of algorithms using different metrics such as area, throughput, power, and energy, and applying state of the art techniques for reducing power consumption such as voltage scaling. However, the evaluation reported in the paper is at very high level and concentrates only on a specific implementation, without considering the effects on energy consumption of different design choices, such as size of the data path, amount of serialization, or effects of architectural optimization applied at each stage of the algorithm. The second work explores area, power, and energy consumption of several recently-developed lightweight block ciphers and compares it with the AES algorithm, considering also possible optimization for the non linear transformation. However, no possible optimization was considered for other transformations, and effects of other design choices, such as serialization were not considered in the work.

Bogdanov et. al.[11] have investigated how the variation in (a) architecture of SBox and MixCloumn, (b) frequency of the clock cycle and (c) unrolling the design can affect the energy consumption. They have analyzed the energy consumption figures of several lightweight ciphers(AES-128, Present, Piccolo, TWINE etc.) with different degrees of unrolling. They have proved that the total energy consumed in a circuit during an encryption operation has roughly a quadratic relation with he degree of unrolling. They have also analyzed that the substitution layer consumes the most part of the energy in the round based design and the 2-round unrolled designs respectively. Again, They have found that in the 2-round unrolled design, the second round functions consume more energy than the first due to switching activity.

2. Energy Consumption of Lightweight Block ciphers in FPGA:

Banik et. al.[12] have tackled the problem of energy efficiency of lightweight block ciphers implemented in re-configurable devices(Field Programmable Gate Array) for the _rst time and explored the effects that round unrolling might have on energy consumption. Their result have shown that Present is the most energy efficient algorithm and 2-round unrolling design is the best one. In this paper, authors have compared energy consumption of 7 different lightweight block ciphers which are implemented in re- configurable devices(FPGA).
Wong et. al.[13] have implemented a compact PRESENT cipher on a FPGA platform and proposed a design uses a 8-bit data path by implementing and factorizing a Boolean S-Box through Karnaugh mapping instead of traditional look-up-table which gives the smallest FPGA implementation of the PRESENT cipher to date, requiring only 62 slices. Though the result of their proposal did not produce the highest throughput but the main competing factor falls on the area requirement of the designs and they achieved the best results, requiring only 62 slices. though this paper is not related to energy consumption on RCE but implementation may help us to develop a better design in terms of energy consumption.

Block diagram of AES:



Rijndael AES algorithm:

The Advanced Encryption Standard(AES) is a symmetric key block published by the National Institute of Standards and Technology(NIST) in December 2001. For selecting AES, three areas were defined by NIST such as security, cost, implementation and Rijndael was judged the best at meeting the combination of these criteria.
AES is a non-Feistel cipher that encrypts and decrypts a data block of 128-bits. It uses 10,12,14 rounds. The are three different key lengths. The encryption/decryption is consists of 10 rounds of processing for a 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.
AES uses several rounds in which each round is made of several stages. Data block is transformed from one stage to another. At the beginning and end of the cipher, AES uses the term data block. Before and after each stage, the data block is referred to as a state. Each round, except the last, uses four transformations that are invertible. The last round has only three transformations. Figure 3 shows the AES cipher structure. Stages of each round are:

Substitute Bytes:

The first transformation, SubBytes, is used at the encryption site. It's a non-linear byte substitution that operates independently on each byte of the state using a substitution table(S-Box). All the 16 bytes of the state are substituted by the corresponding values found from lookup table. In decryption, InvSubBytes, is used. States bytes are substituted from InvSubBytes table. Figure 4 shows the SubByte operation.

Shift Rows:

In the encryption, the state bytes are shifted left in each row. It is called ShiftRows operation. The number of the shifts depends on the row number (0,1,2 or 3) of the state matrix. the row 0 bytes are not shifted and row 1, 2, 3 are shifted 1, 2, 3 bytes left accordingly. Figure 5 shows the ShiftRows operation.

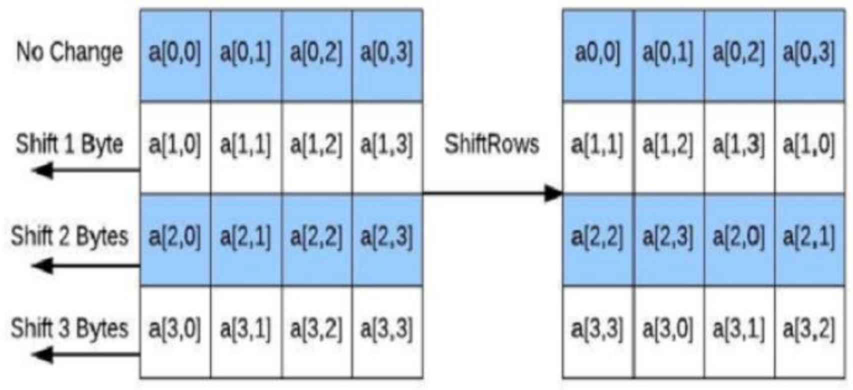

No Change
Shift 1 Byte
Shift 2 Bytes
Shift 3 Bytes
a[0,0] a[0,1] a[0,2] a[0,3]
a[1,0] a[1,1] a[1,2] a[1,3]
a[2,0] a[2,1] a[2,2] a[2,3]
a[3,0] a[3,1] a[3,2] a[3,3]
ShiftRows
a0,0] a[0,1] a[0,2] a[0,3]
a[1,1] a[1,2] a[1,3] a[1,0]
a[2,2] a[2,3] a[2,0] a[2,1]
a[3,3] a[3,0] a[3,1] a[3,2]

Figure 5: Shifting Rows

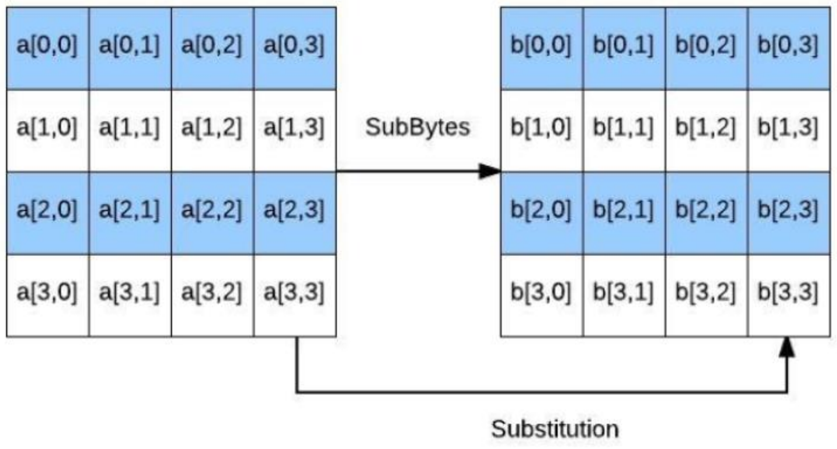


Figure 4: Byte Substitution

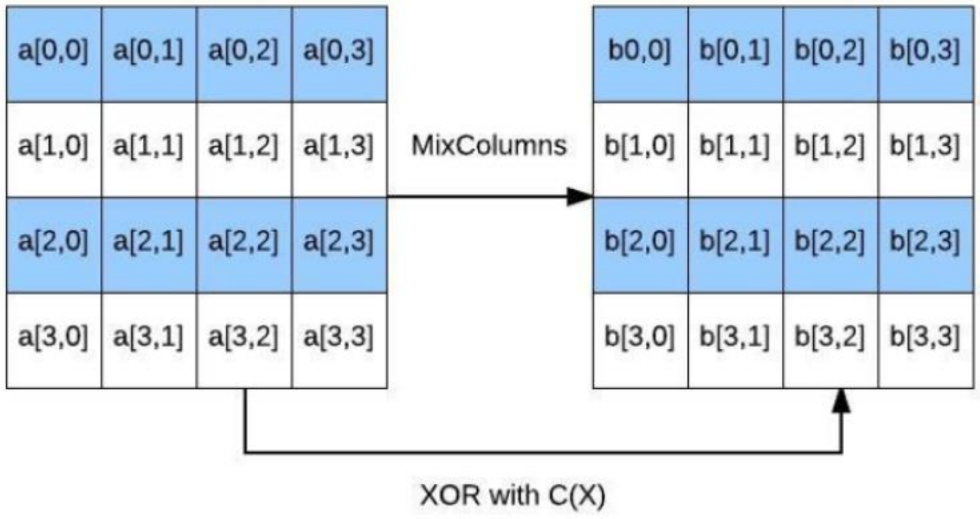


Figure 6: Mixing Columns

<u>Mix columns:</u>

The MixColumn transformation operates at the column level. It transforms each column of the state to a new column. The transformation is a actually the matrix multiplication of a state column by a constant square matrix. The Galois Field is used in this operation. The bytes are treated as polynomials rather than numbers. Figure 6 shows the MixColumn operation.

Add round key:

AddRoundKey proceeds one column at a time. It is similar to MixColumns in this respect. AddRoundKey adds a round key word with each column matrix. The operation in AddRoundKey is matrix addition. Figure 7 shows the AddRoundKey operation. In encryption, SubBytes, ShiftRows, MixColumn and AddRoundKey are performed in all rounds except the last round. MixColumn transformation operation is not performed in the last round of encryption. The decryption process is essentially the same structure as the encryption, following the nine rounds of Inverse ShiftRows, Inverse SubBytes, Inverse AddRoundKey and Inverse MixColumns transformation. In the final round, the Inverse MixColumns is no longer performed.

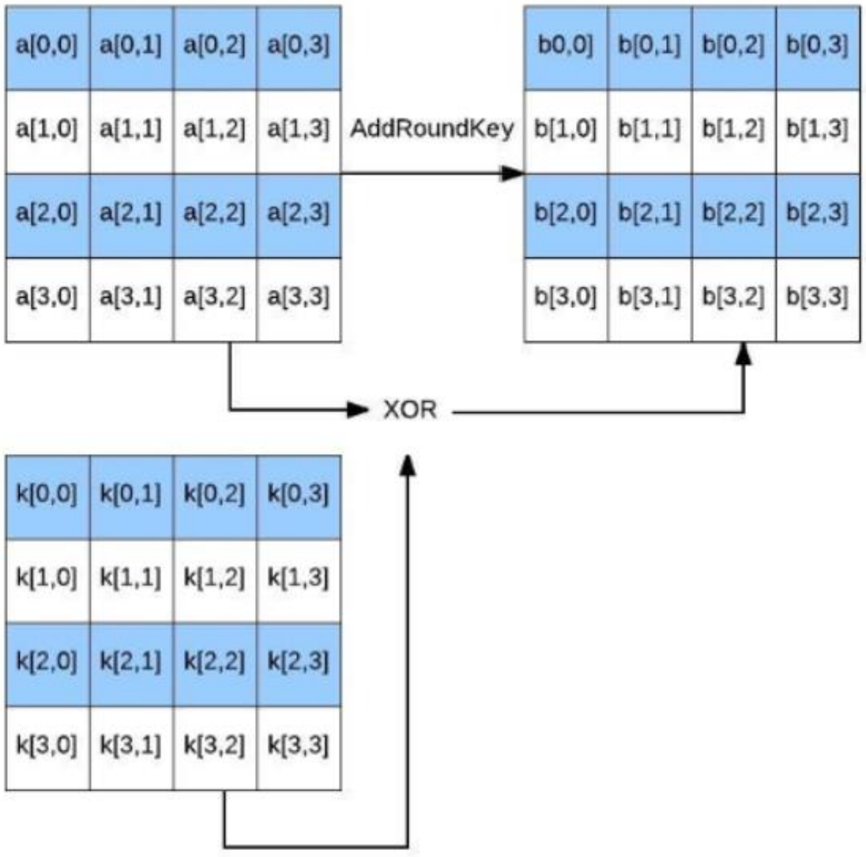

a[0,0] a[0,1] a[0,2] a[0,3]
a[1,0] a[1,1] a[1,2] a[1,3]
a[2,0] a[2,1] a[2,2] a[2,3]
a[3,0] a[3,1] a[3,2] a[3,3]
AddRoundKey
b0,0] b[0,1] b[0,2] b[0,3]
b[1,0] b[1,1] b[1,2] b[1,3]
b[2,0] b[2,1] b[2,2] b[2,3]
b[3,0] b[3,1] b[3,2] b[3,3]
XOR
k[0,0] k[0,1] k[0,2] k[0,3]
k[1,0] k[1,1] k[1,2] k[1,3]
k[2,0] k[2,1] k[2,2] k[2,3]
k[3,0] k[3,1] k[3,2] k[3,3]

Figure 7: Adding Round Key

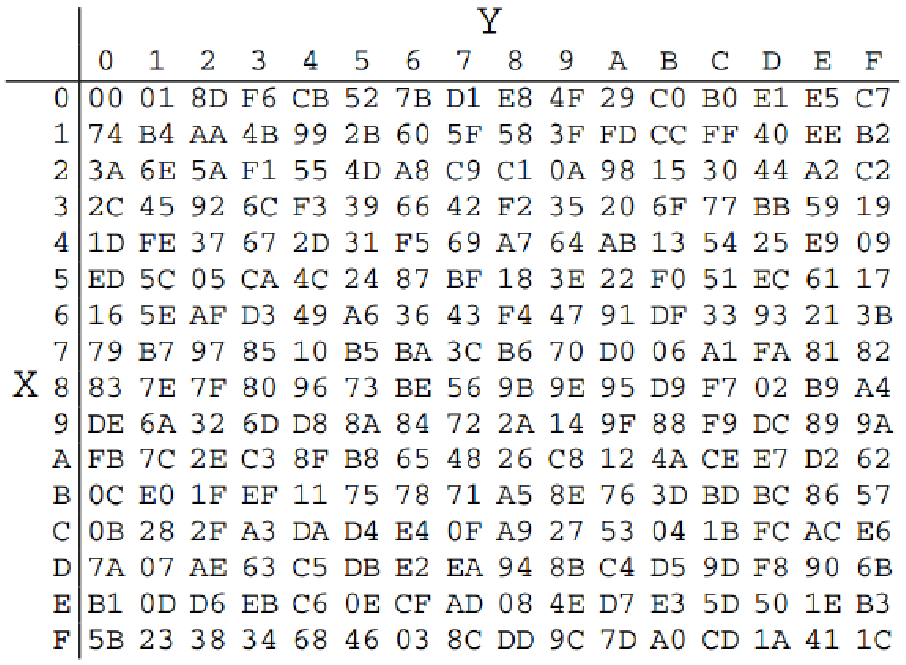
Rijndael S-Box generation method:

In the original AES, S-box is generated in 2 phases.
- Multiplicative inverse phase
-Affine transformation phase

Multiplicative inverse phase: In multiplicative inverse phase, the input byte is in-versed by substituting value from Multiplicative inverse table. The multiplicative inverse table is given in Figure 8.

<u>Affine transformation</u>: Affine transformation consists of two operation. First, 8x8 square matrix's multiplication and secondly, 8x1 constant column matrix addition. In original AES, 0x63 is chosen as the constant column value.

The original AES S-box generation procedure is given step by step below:
- $x^8 + x^4 + x^3 + x + 1$ is used as irreducible polynomial
- b[0]. . . b[7] is the multiplicative inverse of input
- Affine transformation



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

Figure 8: Multiplicative inverse table

-The 8x8 matrix is formed using a formula which is given below:

$$b_i = b_i \ XOR \ b_{(i+4)\ mod\ 8} XOR \ b_{(i+5)\ mod\ 8} XOR \ b_{(i+6)\ mod\ 8} XOR \ b_{(i+7)\ mod\ 8} \ XOR \ C_i$$
$$b_i = i^{th} \ bit \ of \ input \ byte$$
$$C_i = i^{th} \ bit \ of \ a \ specially \ designated \ byte \ which \ is \ 0x63$$

-Finally we get the output byte d[0]. . . d[7]

This figure shows the overall procedure from top to bottom:

Input byte

Inverse

Byte to 8x1 matrix

$$\begin{bmatrix} c[0] \\ c[1] \\ c[2] \\ c[3] \\ c[4] \\ c[5] \\ c[6] \\ c[7] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} b[0] \\ b[1] \\ b[2] \\ b[3] \\ b[4] \\ b[5] \\ b[6] \\ b[7] \end{bmatrix}$$

$$\begin{bmatrix} d[0] \\ d[1] \\ d[2] \\ d[3] \\ d[4] \\ d[5] \\ d[6] \\ d[7] \end{bmatrix} = \begin{bmatrix} c[0] \\ c[1] \\ c[2] \\ c[3] \\ c[4] \\ c[5] \\ c[6] \\ c[7] \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \rightarrow (C)$$

8x1 matrix to byte

Output byte/Substitute byte

Figure : Original S-Box generation process

Proposed Method:

Implementing a modified version of AES-128 so that is consumes less energy and is suitable for resource constrained devices.

Background Study:

As cryptography term comes new to us, we have gone through some basic study related to cryptography.
_ Finite field or Galois Field
_ Encryption and Decryption process of Advanced Encryption Standard (AES)
_ Encryption and Decryption process of Data Encryption Standard (DES)
_ Asymmetric cryptographic algorithm RSA
_ Diffie Hellman key exchange
_ Elliptic Curve Cryptography (ECC)

Hardware Implementation of Standalone AES Encryption in CC2420 (TinyOS 2.1.2 and telosb):

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters.
5.2.2 Telosb
Telosb is a low Power Wireless Sensor Module.
_ TelosB is a mote from Memsic (old Crossbow) technology.
_ It is composed of the MSP430 (the MSP430F1611) microcontroller.
_ It has a radio chip CC2420.
_ The microcontroller of this mote operates at 4.15 MHz.
_ It has a 10 kBytes internal RAM and 48 KBytes of programmable memory.

Applications:

_ Platform for research
_ Wireless sensor network experimentation

Parameters for selecting a suitable encryption algorithm:

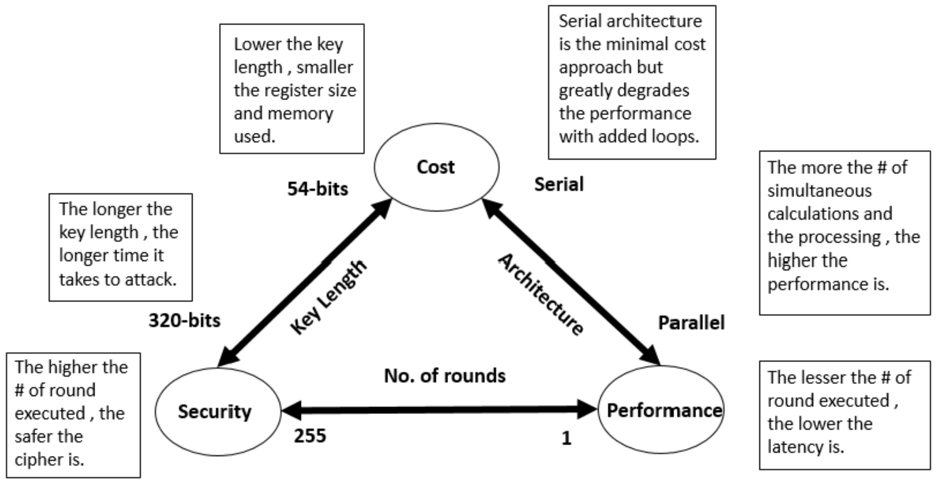

Lower the key length , smaller the register size and memory used.
Serial architecture is the minimal cost approach but greatly degrades the performance with added loops.
Cost
54-bits
Serial
The more the # of simultaneous calculations and the processing , the higher the performance is.
The longer the key length , the longer time it takes to attack.
Key Length
Architecture
320-bits
Parallel
The higher the # of round executed , the safer the cipher is.
No. of rounds
Security
Performance
255
1
The lesser the # of round executed , the lower the latency is.

Figure 10: 3 Parameters for selecting efficient cryptographic algorithms

Choosing appropriate Crypto System for any environment is very important. Proper selection of cryptographic algorithm can help with respect to security issue as well as energy and power. Actually three important parameters help the cryptographer for selecting appropriate algorithm for particular environment. Those parameters are
- Cost
- Security
- Performance

Implementation & Analysis:

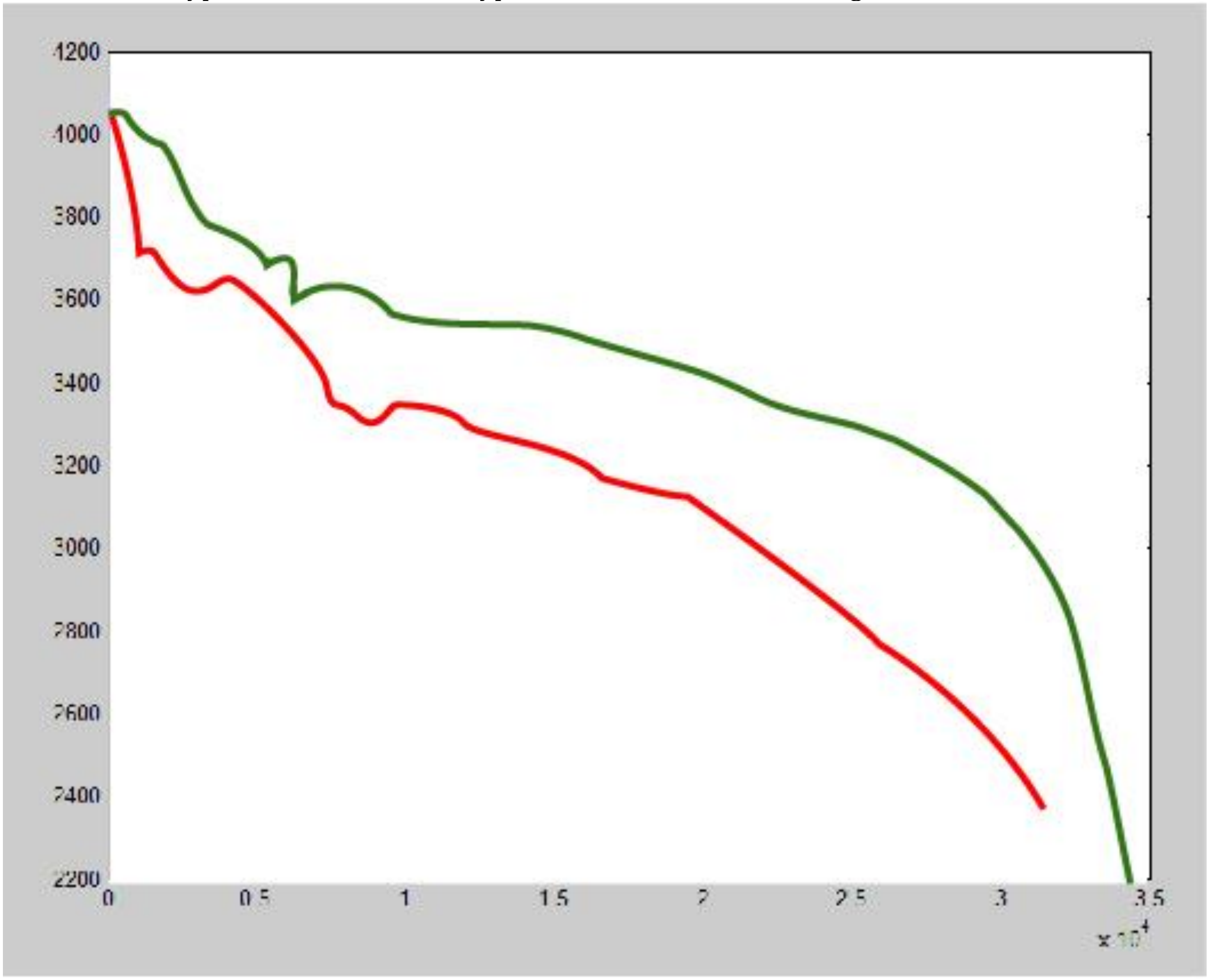
Strength of our Proposal:

- We implemented our modified AES algorithm in C language
- It can encrypt the plain text into the cipher text using a cipher key and decrypt the cipher text into the actual plain text using the same cipher key.
- We analyzed Shift Rows and Mix Column operation which are 2 of the most energy consuming function of AES and tried to reduce the computational power as much as possible.

Implementation:

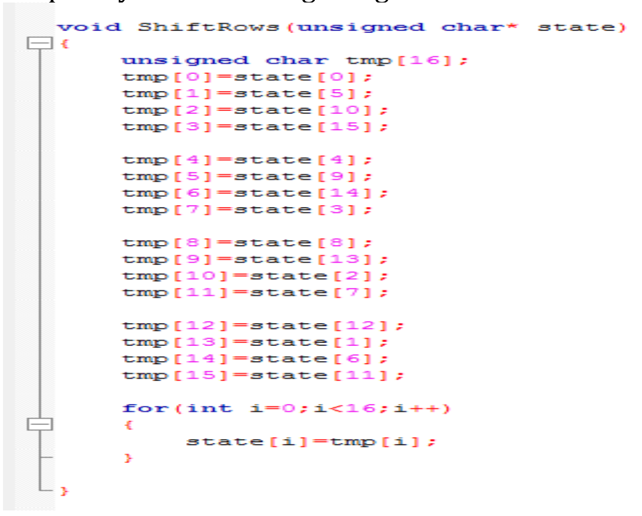
As we are proposing a lightweight version of AES for RCE, first we analyze the actual mechanism of AES. For that purpose we have done the following tasks.
-Understanding the overall mechanism first we have implemented AES algorithm in C language.
- Implementing AES-128 Encryption in CC2420 radio chip of telosb.
- Sensing temperature from environment and sent the sensed data from one mote to

other with encryption and without encryption and measured the voltage deviation.



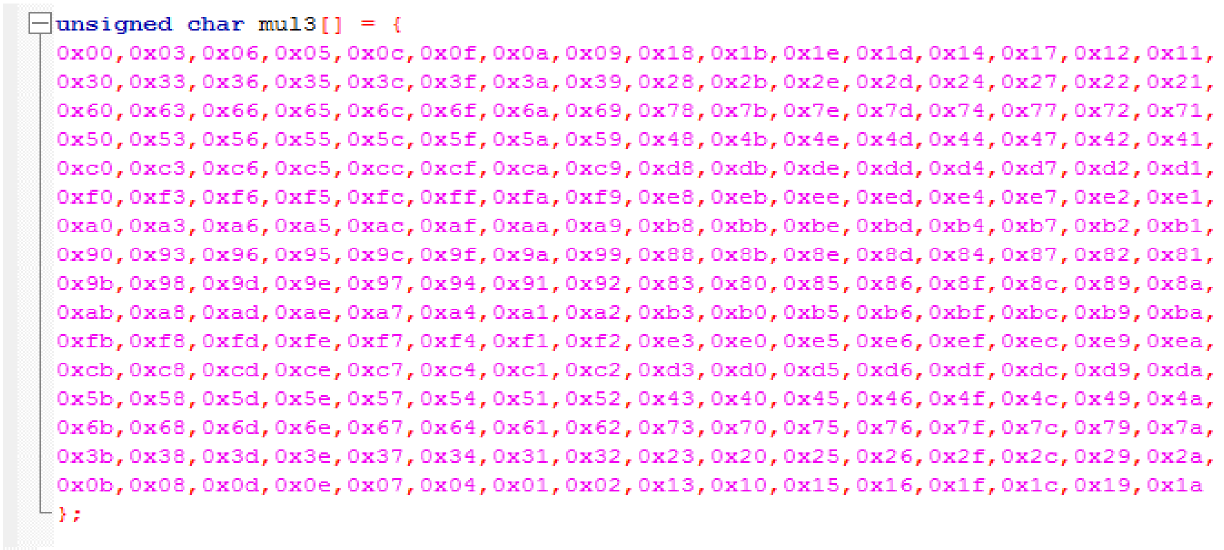
Green Line is data without encryption and the Red line is data with General AES encryption.

-We modified the general AES by hardcoding in shiftrows in the operation of shifting rows to a temporary variable and getting them back instead of a value generating loop.

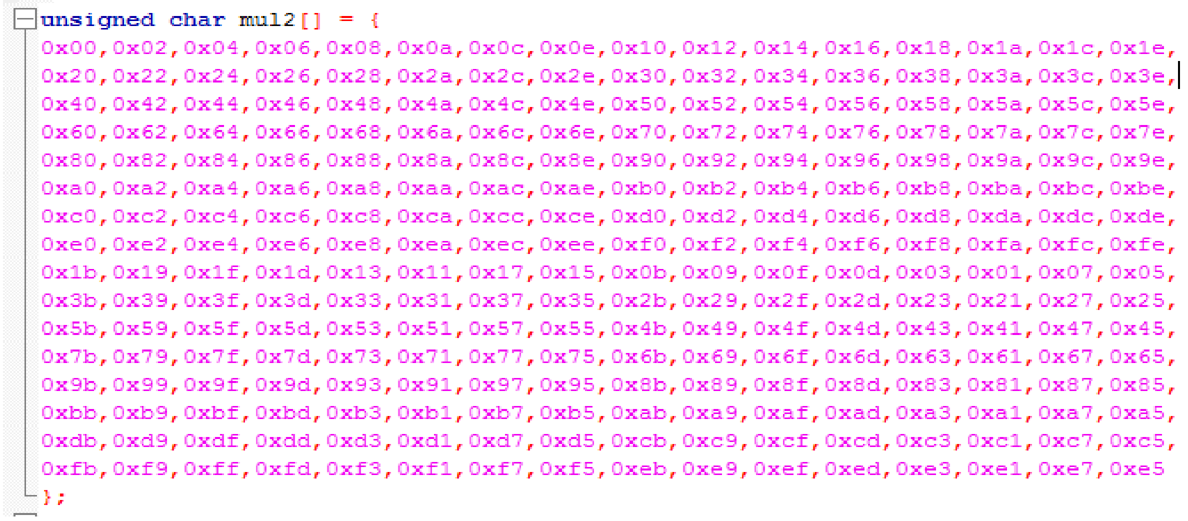


```
void ShiftRows(unsigned char* state)
{
        unsigned char tmp[16];
        tmp[0]=state[0];
        tmp[1]=state[5];
        tmp[2]=state[10];
        tmp[3]=state[15];

        tmp[4]=state[4];
        tmp[5]=state[9];
        tmp[6]=state[14];
        tmp[7]=state[3];

        tmp[8]=state[8];
        tmp[9]=state[13];
        tmp[10]=state[2];
        tmp[11]=state[7];

        tmp[12]=state[12];
        tmp[13]=state[1];
        tmp[14]=state[6];
        tmp[15]=state[11];

        for(int i=0;i<16;i++)
        {
                state[i]=tmp[i];
        }
}
```

- As the computational latency is at most in the mix column function we use 2 extra look up tables to store the multiplication result of state array vs c[x].

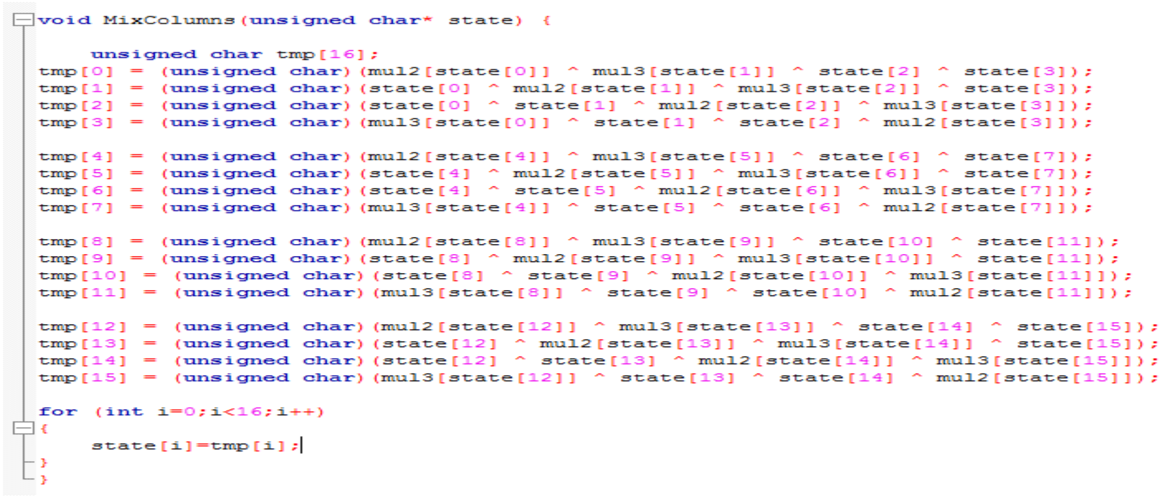


```
unsigned char mul3[] = {
0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
};
```



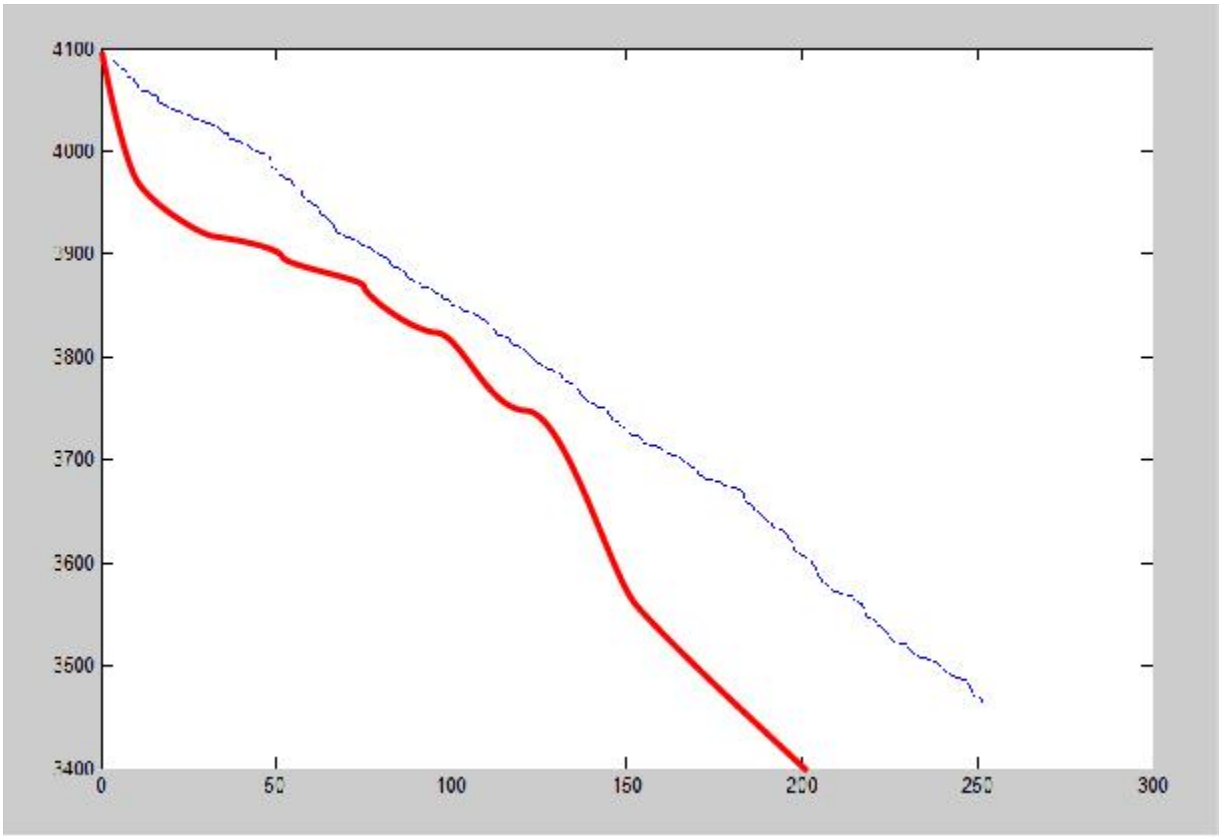
```
unsigned char mul2[] = {
0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
};
```

-This reduces computation drastically as we do not need to generate the values.



```
void MixColumns(unsigned char* state) {

    unsigned char tmp[16];
tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

for (int i=0;i<16;i++)
{
    state[i]=tmp[i];
}
}
```

- Sensing temperature from environment and sent the sensed data from one mote to other with encryption with General AES and with our modified AES and measured the voltage deviation.



Conclusion:

In this paper, we proposed a modified version of AES which is suitable for IoT devices as it consumes less energy than the original one. We analyze that while transmitting data(temperature) from one sensor mote to another without encrypting, consumes less energy than encrypted temperature data through plotting a graph. We also generate a graph which shows us that our proposed AES consumes less energy than the original AES which makes our proposal a lightweight version of AES and it can be used in Resource Constraint Environment.

Future approach:

-Elliptic curve cryptography and hash keys.

References:

[1] J. Daemen and V. Rijmen, \The design of rijndael: Aes-the advanced encryption standard," 2013.
[2] www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf.
[3] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, \Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint," pp. 390{407,2012}
[4] L. Batina, A. Das, B. Ege, E. B. Kavun, N. Mentens, C. Paar, I. Verbauwhede, and T. Yalcin, \Dietary recommendations for lightweight block ciphers: power, energy and area analysis of recently developed architectures," in Radio Frequency Identication, pp. 103{ 112, Springer, 2013.
[5] S. Banik, A. Bogdanov, and F. Regazzoni, \Exploring energy efficiency of lightweight block ciphers," in proceedings of SAC, 2015
[6] S. Banik, A. Bogdanov, and F. Regazzoni, \Exploring the energy consumption of lightweight blockciphers in fpga," in 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1{6, IEEE, 2015.
[7] Tarek Idriss, Haytham Idriss, Magdy Bayoumi \" A PUF-Based Paradigm for IoT Security" IEEE Conference 2016