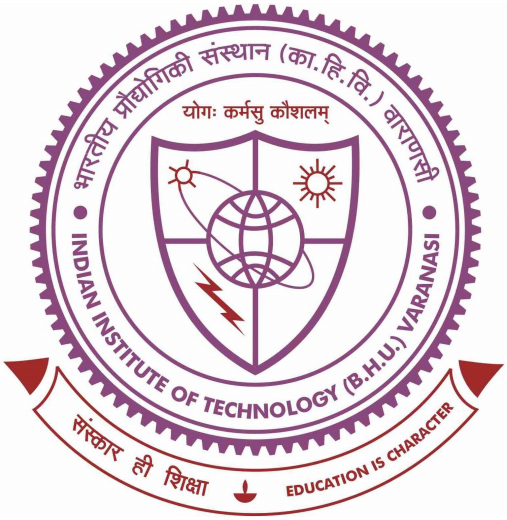
Enhancing Security in Smart Metering Architecture



Thesis submitted in partial fulfilment

for the Award of

BACHELOR IN TECHNOLOGY (B.TECH.)

in

ELECTRICAL ENGINEERING

by

SHASHANK PATHAK, SHREYASH KR. SINGH & HARSH SHUKLA

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY

(BANARAS HINDU UNIVERSITY)

VARANASI – 221 005

Shashank Pathak	Shreyash Kr. Singh	Harsh Shukla
19085107	19085109	19084026

CERTIFICATE

It is certified that the work contained in the thesis titled **Enhancing Security in Smart Metering Architecture** by **Shashank Pathak, Shreyash Kr. Singh & Harsh Shukla** has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

It is further certified that the student has fulfilled all the requirements of Comprehensive Examination, Candidacy and SOTA for the award of **Bachelor in Technology (B.Tech.)**.

Supervisor
Dr. Santosh Kr. Singh
Department of Electrical Engineering
Indian Institute Of Technology
(Banaras Hindu University)
Varanasi – 221 005

DECLARATION BY THE CANDIDATES

We, **Shashank Pathak, Shreyash Kr. Singh & Harsh Shukla**, certify that the work embodied in this thesis is our own bona fide work and carried out by us under the supervision of **Dr. Santosh Kr. Singh** from **January 2022** to **November 2022**, at the **Department of Electrical Engineering**, Indian Institute of Technology (Banaras Hindu University), Varanasi. The matter embodied in this thesis has not been submitted for the award of any other degree/diploma. We declare that we have faithfully acknowledged and given credits to the research workers wherever their works have been cited in my work in this thesis. We further declare that we have not willfully copied any other's work, paragraphs, text, data, results, *etc.*, reported in journals, books, magazines, reports dissertations, theses, *etc.*, or available at websites and have not included them in this thesis and have not cited as my own work.

Date: December 4, 2022
Place: Varanasi, India

Signature of the Students

CERTIFICATE BY THE SUPERVISOR

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Supervisor
Dr. Santosh Kr. Singh
Department of Electrical Engineering
Indian Institute Of Technology
(Banaras Hindu University)
Varanasi – 221 005

Signature of Head of Department

COPYRIGHT TRANSFER CERTIFICATE

Title of the Thesis: Enhancing Security in Smart Metering Architecture

Name of the Students: Shashank Pathak, Shreyash Kr. Singh & Harsh Shukla

COPYRIGHT TRANSFER

The undersigned hereby assigns to the Indian Institute of Technology (Banaras Hindu University), Varanasi all rights under copyright that may exist in and for the above thesis submitted for the award of the Bachelor in Technology (B.Tech.) in Electrical Engineering.

Signature of the Students

Date: December 4, 2022
Place: Varanasi, India

Note: However, the author may reproduce or authorize others to reproduce material extracted verbatim from the thesis or derivative of the thesis for author's personal use provided that the source and the Institute's copyright notice are indicated.

Contents

1. Introduction 1
2. Smart Meters 1
3. Long Range (LoRa) 2
    1. Introduction
    2. LoRa End Node
    3. LoRa Gateway
    4. LoRaWAN Network
    5. How LoRaWan works
    6. Operating frequency for LoRaWAN
4. Chirp Spread Spectrum 6
    1. Modulation
    2. Correlation
    3. Demodulation
5. Advanced Encryption Standard (AES) 9
    1. Introduction
    2. The AES Cipher
    3. The inner workings of a round
    4. Substitute bytes
    5. Shift row transformation
    6. Mix column transformation
    7. AES key expansion

6. A shortcoming: Bit-Flipping/Man-in-the-Middle Attacks 17
7. Galois Counter Mode (GCM) 18
    1. Introduction
    2. Definitions, Abbreviations, and Symbols
    3. Elements of GCM
        1. Block Cipher
        2. Two GCM Functions
            1. Authenticated Encryption Function
            2. Authenticated Decryption Function
        3. Primitives for Confidentiality and Authentication
    4. Mathematical Components of GCM
        1. Multiplication Operation on Blocks
        2. GHASH Function
        3. GCTR Function
    5. GCM Specification
        1. Algorithm for the Authenticated Encryption Function
        2. Algorithm for the Authenticated Decryption Function
    6. Uniqueness requirement on IVs
    7. Authentication Assurance
8. Simulations and Results 28
9. Conclusion 31
10. References 32

List of Figures

1. *Fig 3.1* - A LoRa End Node
2. *Fig 3.2* - A LoRa gateway
3. *Fig 3.3* - LoRaWAN Architecture
4. *Fig 3.4* - LoRa Stack: Application Layer, MAC Layer and Physical Layer
5. *Fig 4.1* - An up-chirp
6. *Fig 4.2* - Types of chirps
7. *Fig 4.3* - A CSS Modulated LoRa signal
8. *Fig 4.4* - Modulation Flow
9. *Fig 4.5* - Demodulation Flow
10. *Fig 5.1* - Flowchart of AES Encryption/Decryption
11. *Fig 5.2* - Substitute Bytes stage of the AES algorithm
12. *Fig 5.3* - Substitution and Inverse substitution boxes
13. *Fig 5.4* - Shift Rows operation
14. *Fig 5.5* - AES Encryption round
15. *Fig 7.1* - GHASH Function
16. *Fig 7.2* - GCTR Function
17. *Fig 7.3* - Authenticated-Encryption Function
18. *Fig 7.4* - Authenticated-Decryption Function
19. *Fig 8.1* - Plaintext message creation using temperature and SOC information received from model sensors
20. *Fig 8.2* - Encrypted and Modulated LoRa Signal
21. *Fig 8.3* - Encryption of message using AES-128

22. *Fig 8.4* - Demodulated Signal Received. LoRa works even under heavy noise
23. *Fig 8.5* - Demodulated payload matched the ciphertext
24. *Fig 8.6* - Decrypted message does not match the initial plaintext message due to bit-flipping attack
25. *Fig 8.7* - Absence of adversary: no bit-flipping attack
26. *Fig 8.8* - Adversary performs a bit-flipping attack; It is detected by comparing auth tag generated by GCM

1 Introduction

With the advent of the new internet age, internet connectivity has become an integral part of everyone's life. Whether banking, shopping, or other day-to-day activities, they are all now run on the internet. Because of easily available internet facilities in many countries, small, low-powered devices connected to the internet have become quite common. The interconnection of such devices is called the Internet of Things (IoT), and these devices are then called IoT devices. The advantage of these devices is the remote access they provide to the user. For example, it can be quite uncomfortable for someone to come back to a cold house from work. An IoT-based thermostat can solve this problem, as this would allow the user to heat their house remotely and reach their cozy house. The government of various countries is also taking note of such advantages and are implementing IoT-based devices in agriculture, power systems, banking, etc. One particularly important use case is the Smart Metering infrastructure.

2 Smart Meters

Smart Meters (SMs) are devices that build upon that normal traditional meters that measure, for example, power flow and add communication capabilities. Generally, Smart Meters have two-way communication capabilities, i.e., they are transceivers. They measure and collect data and send that data at regular intervals to a central hub. They are specially designed to be low-powered, low-bandwidth, and low-data rate, along with weatherproofing and robustness, so they require low maintenance and operation cost.

Smart Meters also bring benefits for suppliers as it allows remote control of supply, thus reducing the cost of manual labor, and also pricing models based on time of use can be implemented. For example, an increased rate could be charged per unit during peak times. This would also decrease the load on the grid during peak times.

Apart from suppliers, consumers also reap benefits from SM installation. First, consumers would not need to submit meter readings now as they would be automatically sent to the power generation company. Secondly, this would allow accurate usage tracking and bill generation. From the usage tracking, users can infer the time durations where they consume the most power, and some smart meters with additional capabilities could track distortion in power factors and faulty appliances.

Due to these benefits, the Government of India plans to replace 25 crore conventional meters with SMs under Smart Meter National Program (SMNP). Under this program, 52,03,425 SMs have been installed (as of 04-12-2022), and in New Delhi Municipal Council, SMs have enabled a billing efficiency of 99% and an increase in revenue of Rs.500/month/meter. Implementation of prepaid models using SMs has increased ROI by 0.5%.

To attain these low-power capabilities, SM uses some common communication protocols like ZigBee, Z-Wave, Bluetooth Low Energy (BLE) or LoRaWAN

3 Long Range (LoRa)

3.1 Introduction

LoRaWAN (Long Range Wide Area Network) is a specification for a telecommunication network suitable for long-distance communication with little power. LoRa is the technology, and LoRaWAN is the network working based on this technology. The architecture consists of gateways, network servers, and application servers. There are RF chips from Semtech used to transmit a spread spectrum. The range for LoRa is:

- 2-5 km in urban areas
- 5-15 km in rural areas and countryside
- More than 15 km in direct line of sight

3.2 LoRa End Node

A LoRa end node consists of 2 parts

- A radio module with an antenna
- A microprocessor to process the data

End nodes are generally battery-powered. The end node uses a RF module to send and receive data from the gateway in the form of radio signals

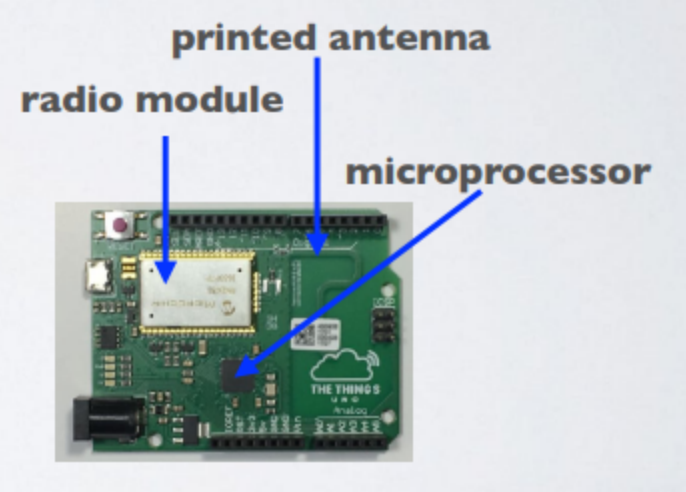


Fig 3.1: A LoRa end node

3.3 LoRa Gateway

Similar to end nodes, the LoRa gateway also consists of 2 parts

- A radio module with an antenna
- A microprocessor to process the data

Gateways are mains powered and connected to the internet. Multiple gateways can receive data from the same end node and vice versa

Fig 3.2: A LoRa gateway

3.4 LoRaWAN Network

LoRaWAN network architecture is deployed in a star topology. Communication between the end node and gateway is bidirectional, which means that data can be sent and received by both - the end node and the gateway

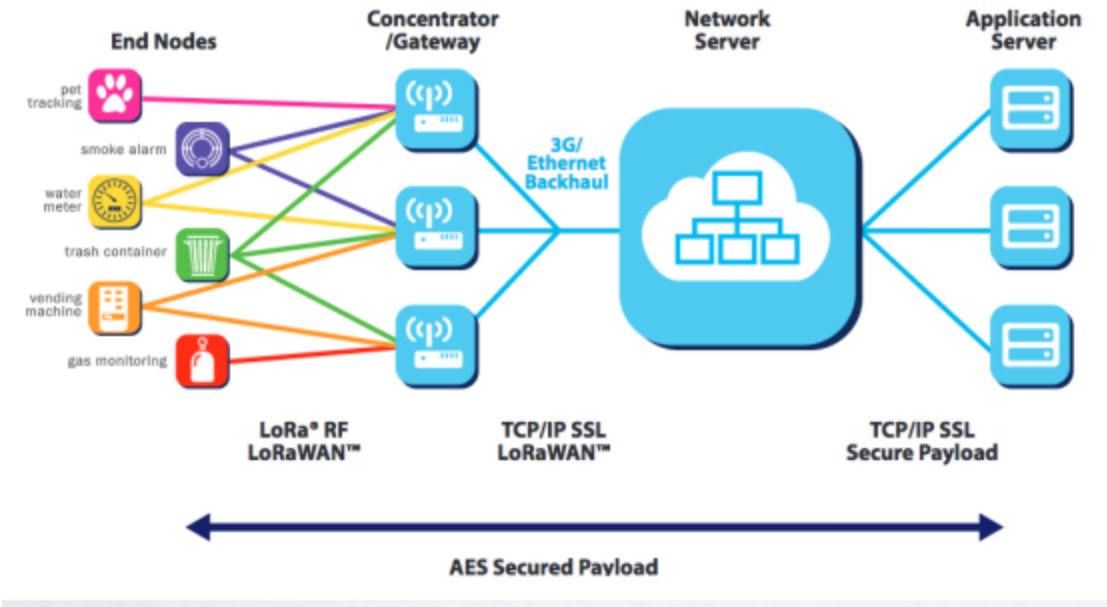


Fig 3.3: LoRaWAN Architecture

3.5 How LoRaWAN works

An end node broadcast its data to every gateway in its vicinity. The gateways forward this packet to the network server. The network server collects the messages from all gateways and filters out the duplicate data, and determines the gateway that has the best reception. The network server forwards the packet to the correct application server, where the end user can process the sensor data

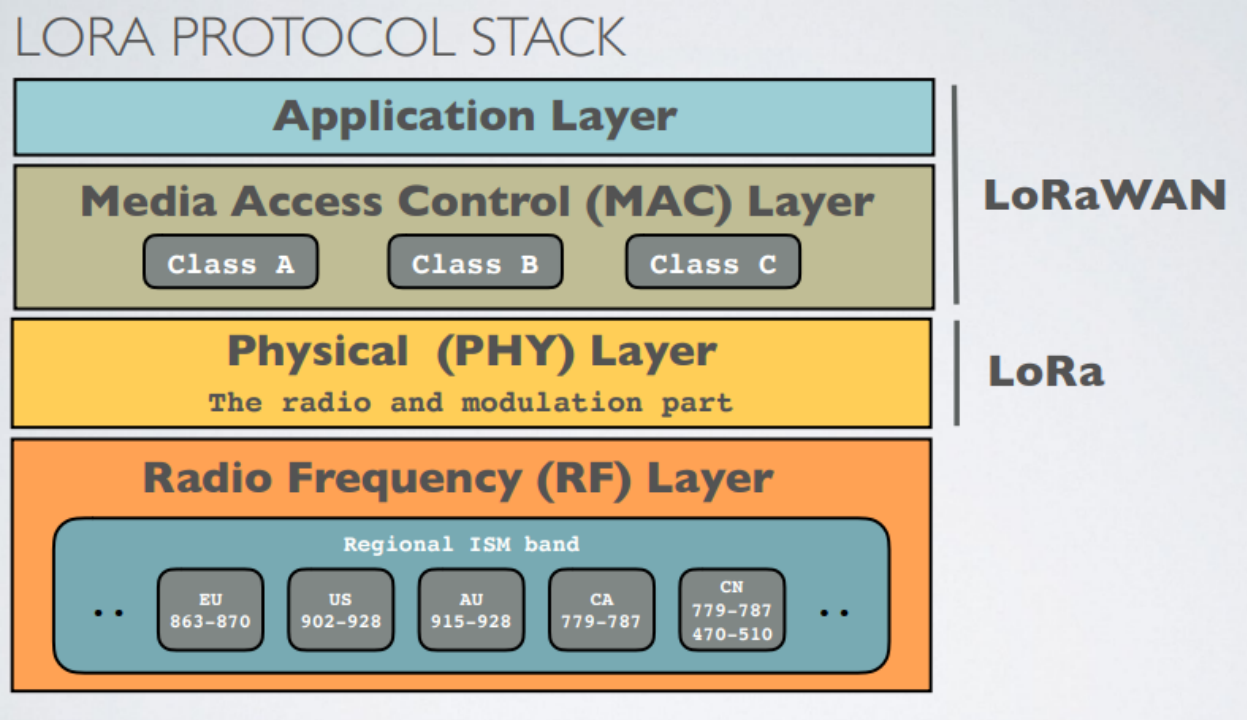


Fig 3.4: LoRa Stack: Application Layer, MAC Layer & Physical Layer

3.7 Operating Frequency for LoRaWAN

LoRa operates in the unlicensed ISM (Industrial, Scientific, and Medical) radio bands that is available worldwide. In India, this frequency lies in the range of 863-870 MHz. The advantage of ISM is that anyone is allowed to use these frequencies, and no license fee is required for the same. The disadvantage of ISM is that it has a low data rate, and lots of interference

4 Chirp Spread Spectrum

Chirp Spread Spectrum (CSS for short) is the modulation technique employed by LoRa for modulating and demodulating signals at end nodes and the gateway. Unlike other methods of modulation/demodulation like AM (Amplitude Modulation) or FM (Frequency Modulation), this method of modulation uses Frequency Shift Chirp Modulation. This means that each symbol that needs to be sent as a packet corresponds to a different phase of the same base signal, called chirps. These chirps can be of two types: (a) Up-Chirp is a sinusoidal signal whose frequency increases linearly with time. (b) Down-Chirp is a sinusoidal wave whose frequency decreases linearly with time.

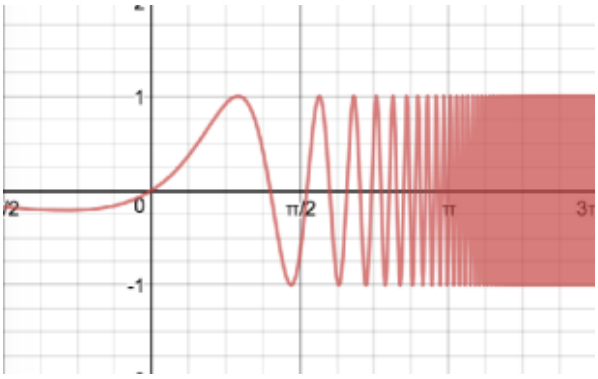


Fig 4.1: An up-chirp

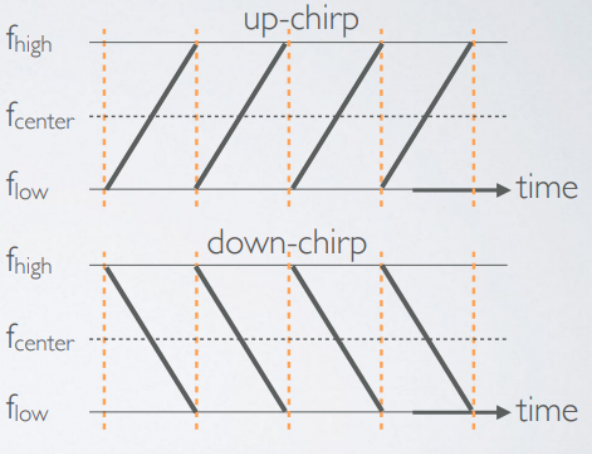


Fig 4.2: Types of chirps

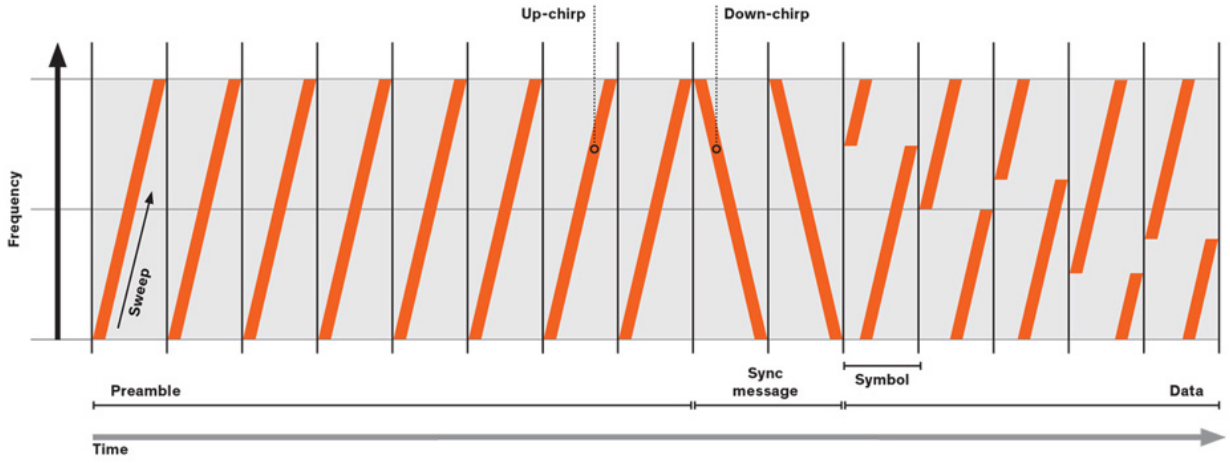


Fig 4.3: A CSS Modulated LoRa Signal

4.1 Modulation

The modulation step is tasked with converting a message into LoRa-compatible signals. We define the number of signals as 2^{SF}, where SF stands for spreading factor, and it is directly related to the sweep of any symbol; the smaller the SF, the steeper the sweep will be, and vice versa. Let's say we need to send a symbol s. Thus the Semtech LoRa standard specifies the following equation to attain a corresponding modulated signal for it:

$$c(nT_s + kT) = \frac{1}{\sqrt{2^{SF}}} e^{j2\pi((s(nT_s)+k) \bmod 2^{SF})\frac{k}{2^{SF}}}$$

$$\text{where, } T_s = \text{Switching Period}$$

$$SF = \text{Spreading Factor}$$

$$s = \text{Symbol}$$

$$T = \text{Sampling Time}$$

This expression provides us with an up-chirp signal that starts at a frequency s and rises in frequency until it hits the bandwidth upper limit and then rolls back to the lower limit of the bandwidth. Essentially it gives us a discontinuous up-chirp or an up-chirp that has been shifted by s in the frequency-time domain.

Fig 4.4: Modulation flow

4.2 Correlation

Now that we have created the symbol corresponding to the message, we also need to demodulate it or match it with the same symbol at the receiver end to obtain a 1-1 mapping between the sender and receiver. This is can be traditionally done in digital systems using sample-by-sample multiplication of the message and reference symbol. The reference that gives the highest multiplication product with the message is chosen as the transmitted symbol. This an intuitive and simple but very inefficient technique, as it would need to multiply each sample in the message with the corresponding sample in reference symbol (i.e., 2^{SF} multiplications) for each reference symbol (2^{SF} in number). Thus we would need $(2^{SF})^2$ multiplications to demodulate a single symbol.

4.3 Demodulation

As we saw that a simple correlation would not be fit for our task, hence scientists at Semtech came up with clever mathematical manipulation of the complex conjugate of the up-chirp symbols to obtain a very efficient modulation.

$$
\begin{aligned}
c^*(nT_s + kT) &= \frac{1}{\sqrt{2^{SF}}} e^{-j2\pi((s(nT_s)+k) \bmod 2^{SF})\frac{k}{2^{SF}}} \\
&= \frac{1}{\sqrt{2^{SF}}} e^{-j2\pi((s(nT_s)+k) \bmod 2^{SF}+k-k)\frac{k}{2^{SF}}} \\
&= \frac{1}{\sqrt{2^{SF}}} \underbrace{e^{-j2\pi\frac{k^2}{2^{SF}}}}_{\text{Base Down Chirp}} \underbrace{e^{-j2\pi(s(nT_s) \bmod 2^{SF})\frac{k}{2^{SF}}}}_{\text{Sine Wave at frequency s}}
\end{aligned}
$$

This shows that the complex conjugate of symbols can be represented by the product of the base down chirp and a pure sine wave at frequency s. That means if we multiply our message symbol with the base down chirp, we can then analyze the frequency of the result to obtain our symbol. This is mathematically equivalent to multiplying our message symbol by a base down chirp (called dechirping) and then taking the Discrete Fourier Transform (we use DFT as the signal is not analogous, but digital in sample time), more precisely, Fast Fourier Transform (FFT) to obtain the frequency s.

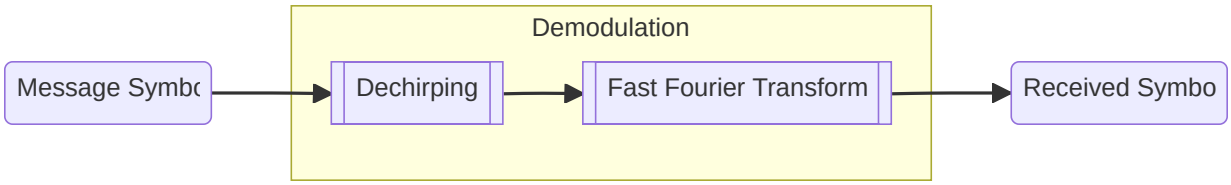


Fig 4.5: Demodulation flow

5 Advanced Encryption Standard (AES)

5.1 Introduction

Numerous outdated ciphers have serious security weaknesses. On contemporary computer platforms, the older ciphers are easily cracked. In 1998, a machine that cost roughly $250,000 was used to crack the DES algorithm. As it was created for hardware in the middle of the 1970s and did not produce effective software code, it was also far too slow in terms of software. The Triple DES, which is an upgrade, is slower and has three times as many rounds as DES. Aside from that, it is significantly slower, and its security is dubious in light of current hardware capabilities.

A completely new encryption algorithm was needed. One that would be impervious to all recognized assaults. A new standard needed to be created, and the National Institute of Standards

and Technology (NIST) wanted to assist. They held a contest in which anyone in the world could take part. Once the candidate algorithms had been submitted, several years of scrutinization in the form of cryptographic conferences took place. The algorithms were tested for efficiency and security both by some of the world's best publicly renowned cryptographers and by NIST itself.

After all this investigation, NIST finally chose an algorithm known as **Rijndael**. Rijndael was named after the two Belgian cryptographers who developed and submitted it. Thus Rijndael became to be known as AES.

5.2 The AES Cipher

AES is a symmetric block cipher. This means that it uses the same key for both encryption and decryption. The block and key can be chosen independently from 128, 160, 192, 224, *and* 256 bits and need not be the same. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, and 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192, or AES- 256, respectively. AES differs from DES in that it is not a Feistel structure. Recall that in a Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. In this case, the entire data block is processed in parallel during each round using substitutions and permutations.

Several AES parameters depend on the key length. For example, if the key size used is 128, then the number of rounds is 10, whereas it is 12 and 14 for 192 and 256 bits, respectively. At present, the most common key size likely to be used is the 128-bit key. This description of the AES algorithm, therefore, describes this particular implementation.

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of hardware

The input is a single 128-bit block for decryption and encryption and is known as the in matrix. This block is copied into a $state$ array which is modified at each stage of the algorithm and then copied to an output matrix. The plaintext and key are depicted as a 128-bit square matrix. This key is then expanded into an array of key schedule words (the w matrix). It must be noted that the ordering of bytes within in matrix column-wise. The same applies to the w matrix.

5.3 Inner Workings of a Round

The algorithm begins with an Add Round Key stage followed by 9 rounds of four stages and the tenth round of three stages. This applies to both encryption and decryption with the exception that in each stage of a round the decryption algorithm is the inverse of its counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows
3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the **Mix Columns** stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key
4. Inverse Mix Columns

Again, the tenth round simply leaves out the **Inverse Mix Columns** stage. Each of these stages will now be considered in more detail.

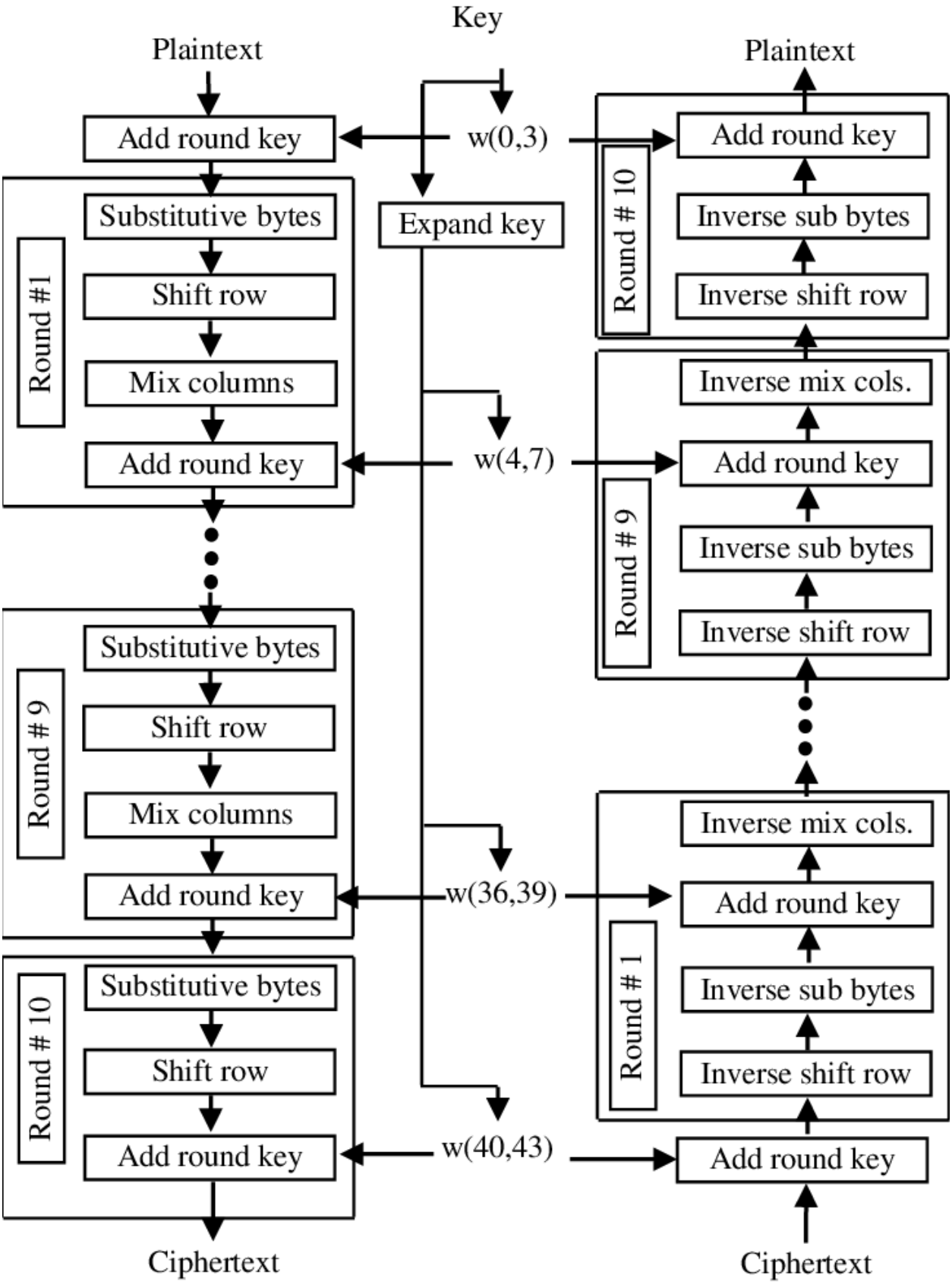

Key
Plaintext
Add round key
w(0,3)
Expand key
Round #1
Substitutive bytes
Shift row
Mix columns
Add round key
w(4,7)
Round # 9
Substitutive bytes
Shift row
Mix columns
Add round key
w(36,39)
Round # 10
Substitutive bytes
Shift row
Add round key
w(40,43)
Ciphertext
Plaintext
Round # 10
Add round key
Inverse sub bytes
Inverse shift row
Round # 9
Inverse mix cols.
Add round key
Inverse sub bytes
Inverse shift row
Round # 1
Inverse mix cols.
Add round key
Inverse sub bytes
Inverse shift row
Add round key
Ciphertext

Fig 5.1: Flowchart of AES Encryption/Decryption

5.4 Substitute Bytes

This stage is a table lookup using a 16x16 matrix of byte values called an $s-box$. This matrix consists of all the possible combinations of an 8-bit sequence ($2^8 = 16 * 16 = 256$). However, the $s-box$ is not just a random permutation of these values, and there is a well-defined method for creating the $s-box$ tables. We will not be too concerned here about how the s-boxes are made up and can simply take them as table lookups.

Again the matrix that gets operated upon throughout the encryption is known as the $state$. We will be concerned with how this matrix is affected in each round.

For this particular round, each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column. For example, byte 95 selects row 9, column 5, which turns out to contain the value 2*A*. This is then used to update the $state$ matrix.

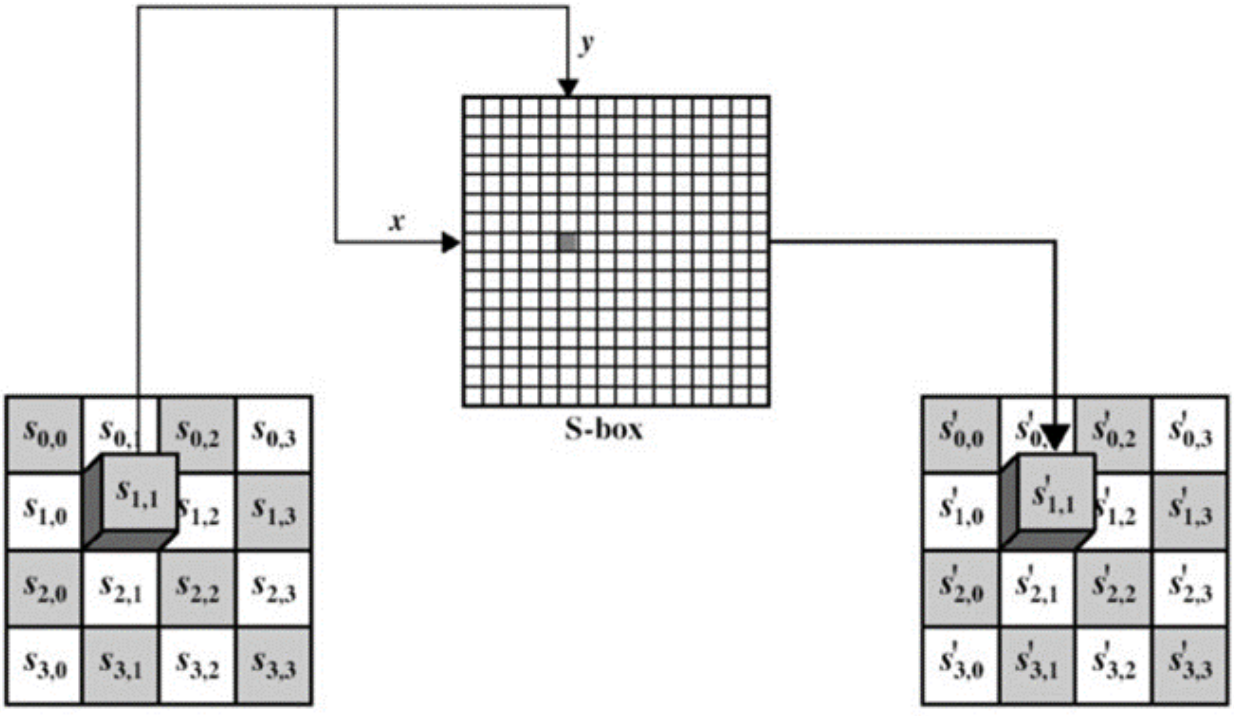


Fig 5.2: Substitute Bytes Stage of the AES algorithm.

The Inverse substitute byte transformation makes use of an inverse s-box called $Is-box$. In this case, what is desired is to select the value 2A and get the value 95.

The $s-box$ is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design with a low correlation between input bits and output bits and the property that the output cannot be described as a simple mathematical function of the input. In addition, the s-box has no fixed points $(s-box(a)=a)$ and no opposite fixed points $(s-box(a)=\bar{a})$ where $\bar{a}$ is the bitwise complement of a. The $s-box$ must be invertible if decryption is to be possible $(Is-box(s-box(a))=a)$. However, it should not be its self inverse i.e. $s-box(a)\neq Is-box(a)$.

(a) S-box

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

(b) Inverse S-box

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

Fig 5.3: Substitution and Inverse substitution boxes

5.5 Shift Row Transformation

This stage is shown in the figure. This is a simple permutation and nothing more. It works as follows:

- The first row of the $state$ is left as it is
- The second row is shifted 1 byte to the left in a circular
- The third row is shifted 2 bytes to the left in a circular
- The fourth row is shifted 3 bytes to the left in a circular

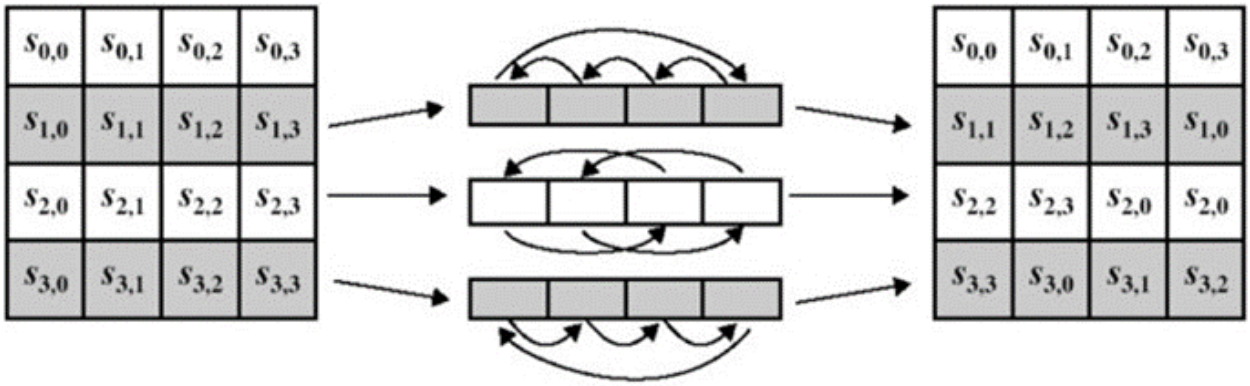


Fig 5.4: Shift Rows operation

The transformation also ensures that the four bytes of one column are spread out to four different columns. The Inverse Shift Rows transformation performs these circular shifts in the opposite direction for each of the last three rows (the first row was unaltered, to begin with).

5.6 Mix Column Transformation

This stage is a substitution but uses the arithmetic of $GF(2^8)$. Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be determined by the following matrix multiplication on the $state$:

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

Each element of the product matrix is the sum of the products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2^8).

5.7 AES Key Expansion

The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words. Each word contains 32 bytes, which means each subkey is 128 bits long.

```
KeyExpansion(byte key[16], word w[44]):
  word temp
  for i in range(4):
    w[i]=(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3])
  end
  for i in range(4, 44):
    temp=w[i-1]
    if (i mod 4 == 0):
      temp=SubWord(RotWord(temp)) XOR Rcon[i/4]
    end
    w[i]=w[i-4] XOR temp
  end
```

The figure gives a summary of each of the rounds. The ShiftRows column is depicted here as a linear shift which gives a better idea of how this section helps in the encryption.

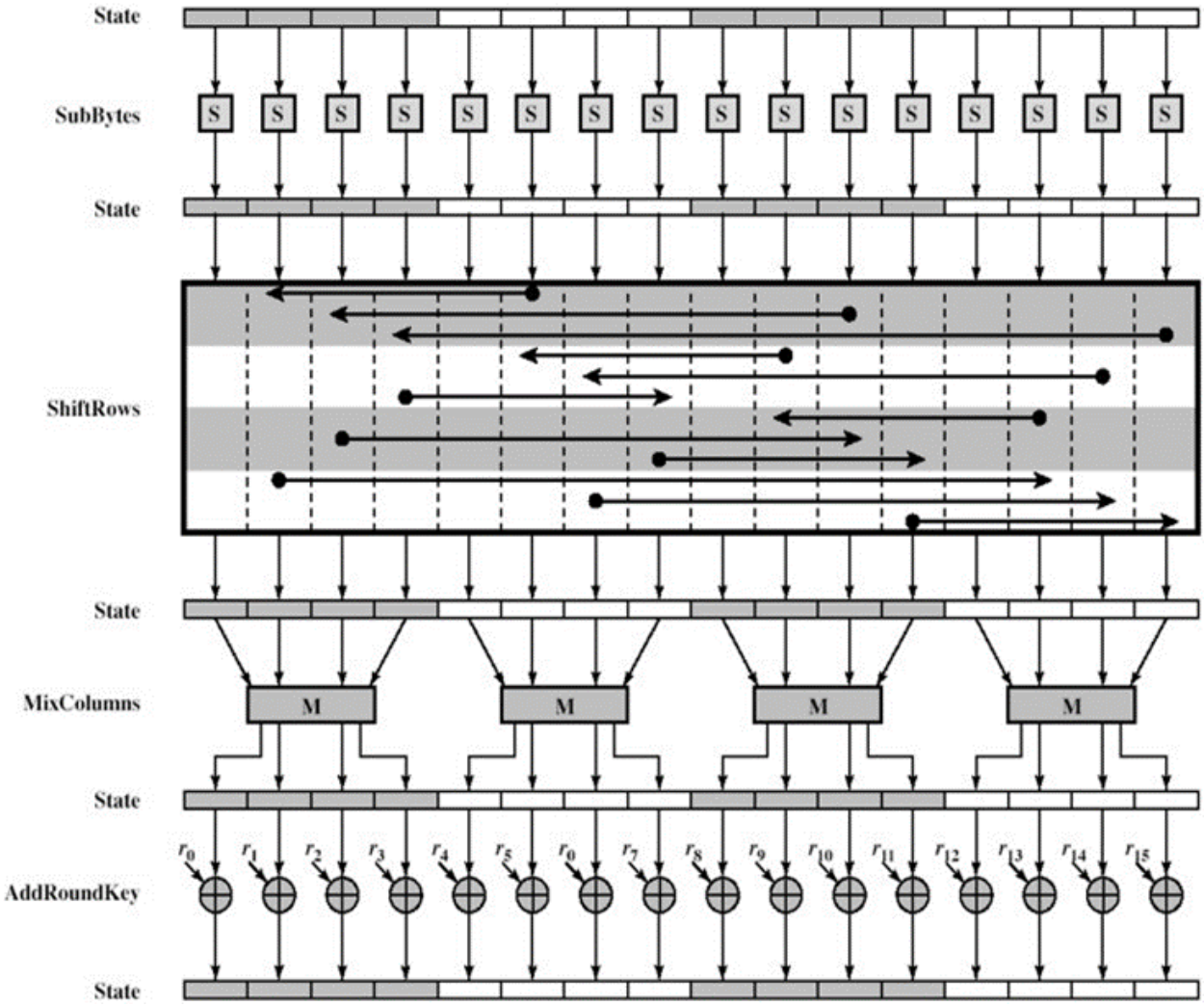


Fig 5.5: AES encryption round

6 A shortcoming: Bit-Flipping/Man-in-the-Middle Attacks

The Advanced Encryption Standard (AES) used in LoRaWAN 1.1 (current implementation) does provide almost impermeable security for the message it carries as the ciphertext. However, it provides next to none protection against a type of attack called a Man-in-the-Middle Attack (MIMA). In this type of attack, the message packets that are being transmitted from one device to another are intercepted by an adversary using loop-holes in communication protocol or an affected device. This type of attack is generally achieved by spoofing one of the devices in the network. Once the other devices are fooled into trusting our device, we can start reading and

intercepting packets. If unencrypted data is being transmitted, the attacker can easily read them, modify them and transmit them further in the network to obtain *any* desired activity.

As this is a major security issue, encryption is provided so that in case the network is exposed to malicious entities, the message that is being sent and/or received cannot be read. But this does not stop the attacker from modifying the packets that are being communicated. An adversary can, for example, flip any random bit in the message, which would cause a garbled message to be propagated further into the network. As this is unexpected behavior, this garbled message would lead to loss of data or, worse cases, even cause malfunctioning of devices that are connected to the network.

The attack that is mentioned above is called a Bit-Flipping attack, and the current implementation of AES (AES-128, AES-192, AES-256) in Counter (CTR) Mode does not provide any protection against this. To be able to discern that the message that is being received has been tampered with, we need to add some sort of authentication code. The comparison of this authentication code with the received message should signal the tampering and alert the network to not act on this data. The method that we propose is the use of Galois Counter Mode (GCM) and Galois Message Authentication Code (GMAC) along with AES-128.

7 Galois Counter Mode (GCM)

7.1 Introduction

This Recommendation specifies Galois/Counter Mode (GCM) algorithm for authenticated encryption with associated data. GCM is constructed from an approved symmetric key block cipher with a block size of 128 bits. Thus, GCM is a mode of operation of the AES algorithm.

GCM provides assurance of the confidentiality of data using a variation of the Counter mode of operation for encryption. GCM assures the authenticity of the confidential data (up to about 64 gigabytes per invocation) using a universal hash function that is defined over a binary Galois (i.e., finite) field. GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted.

If the GCM input is restricted to data that is not to be encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data. In the rest of this document, statements about GCM also apply to GMAC.

GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error-detecting code; in particular, GCM can detect both accidental modifications of the data and

intentional, unauthorized modifications.

The two functions of GCM are called authenticated encryption and authenticated decryption. Each of these functions is relatively efficient and parallelizable; consequently, high-throughput implementations are possible in both hardware and software.

7.2 Definitions, Abbreviations, and Symbols

AAD	Additional Authenticated Data
Additional Authenticated Data	The input data to the authenticated encryption function is authenticated but not encrypted.
AES	Advanced Encryption Standard.
Authenticated Decryption	The function of GCM in which the ciphertext is decrypted into the plaintext and the authenticity of the ciphertext and the AAD is verified.
Authenticated Encryption	The function of GCM is that the plaintext is encrypted into the ciphertext, and an authentication tag is generated on the AAD and the ciphertext.
Authentication Tag (Tag)	A cryptographic checksum on data is designed to reveal both accidental errors and the intentional modification of the data.
Authenticity	The property that data originated from its purported source.
Bit	A binary digit: 0 or 1.
Bit String	A finite, ordered sequence of bits.
Block	For a given block cipher, a bit string whose length is the block size of the block cipher.
Block Cipher	A parameterized family of permutations on bit strings of a fixed length; the parameter that determines the permutation is a bit string called the key.
Block Size	For a given block cipher and key, the fixed length of the input (or output) bit strings.
Byte	A sequence of 8 bits.
Byte String	A finite, ordered sequence of bytes

Ciphertext	The encrypted form of the plaintext.
GCM	Galois/Counter Mode
Initialization Vector	A nonce that is associated with an invocation of authenticated encryption on a particular plaintext and AAD.
Inverse Cipher Function	The inverse of the forward cipher function for a given key.
IV	Initialization Vector
Key	The parameter of the block cipher determines the selection of the forward cipher function from the family of permutations.
Least Significant Bit(s)	The right-most bit(s) of a bit string
Most Significant Bit(s)	The left-most bit(s) of a bit string.

Symbols

A	The additional authenticated data
C	The ciphertext.
H	The hash subkey
ICB	The initial counter block
IV	The initialization vector.
K	The block cipher key.
P	The plaintext.
R	The constant within the algorithm for the block multiplication operation.
T	The authentication tag.
t	The bit length of the authentication tag.
0*s*	The bit string consists of *s* '0' bits.
CIPH*K* (*X*)	The output of the forward cipher function of the block cipher under the key K *is* applied to block *X*.
GCTR*K* (*ICB*, *X*)	The output of the GCTR function for a given block cipher with key *K* applied to the bit string *X* with an initial counter block *ICB*.

GHASH$_H$ (X)	The output of the GHASH function under the hash subkey H *is* applied to the bit string X.
incs(X)	The output of incrementing the right-most s bits of the bit string X, regarded as the binary representation of an integer, by 1 modulo 2^s.
int(X)	The integer for which the bit string X is a binary representation.
len(X)	The bit length of the bit string X.
LSB$_s$ (X)	The bit string consists of the s right-most bits of the bit string X.
MSB$_s$ (X)	The bit string consists of the s left-most bits of the bit string X.
$X >> 1$	The bit string that results from discarding the rightmost bit of the bit string X and prepending a '0' bit on the left.
$X \| \| Y$	The concatenation of two bit strings X and Y.
$X \bullet Y$	The product of two blocks, X and Y, regarded as elements of a certain binary Galois field.
$x \cdot y$	The product of two integers, x and y.

7.3 Elements of GCM

The elements of GCM and the associated notation and requirements are introduced in the three sections below.

7.3.1 Block Cipher

The operations of GCM depend on the choice of an underlying symmetric key block cipher and thus can be considered a mode of operation (mode, for short) of the block cipher. The GCM key is the block cipher key (the key, for short).

For any given key, the underlying block cipher of the mode consists of two functions that are inverses of each other. The choice of the block cipher includes the designation of one of the two functions of the block cipher as the forward cipher function, as in the specification of the AES algorithm. GCM does not employ the inverse cipher function.

The forward cipher function is a permutation on bit strings of a fixed length; the strings are called blocks. The length of a block is called the block size. The key is denoted K, and the

resulting forward cipher function of the block cipher is denoted $CIPH_K$.

The underlying block cipher shall be approved, the block size shall be 128 bits, and the key size shall be at least 128 bits. The key shall be generated uniformly at random or close to uniformly at random, i.e. so that each possible key is (nearly) equally likely to be generated. Consequently, the key will be fresh, i.e., unequal to any previous key, with high probability. The key shall be secret and shall be used exclusively for GCM with the chosen block cipher.

7.3.2 Two GCM Functions

The two functions that comprise GCM are called authenticated encryption and authenticated decryption. The authenticated encryption function encrypts the confidential data and computes an authentication tag on both the confidential data and any additional, non-confidential data. The authenticated decryption function decrypts the confidential data, contingent on the verification of the tag.

An implementation may restrict the input to non-confidential data, i.e., without any confidential data. The resulting variant of GCM is called GMAC. For GMAC, the authenticated encryption, and decryption functions become the functions for generating and verifying an authentication tag on the non-confidential data.

7.3.2.1 Authenticated Encryption Function

7.3.2.1.1 Input Data

Given the selection of an approved block cipher and key, there are three input strings to the authenticated encryption function:

- A plaintext denoted P;
- Additional authenticated data (AAD), denoted A; and
- An initialization vector (IV), denoted IV.

The plaintext and the AAD are the two categories of data that GCM protects. GCM protects the authenticity of the plaintext and the AAD; GCM also protects the confidentiality of the plaintext, while the AAD is left in the clear. For example, within a network protocol, the AAD might include addresses, ports, sequence numbers, protocol version numbers, and other fields that indicate how the plaintext should be treated.

The IV is essentially a nonce, i.e., a unique value within the specified context, which determines an invocation of the authenticated encryption function on the input data to be protected.

Although GCM is defined on bit strings, the bit lengths of the plaintext, the AAD, and the IV shall all be multiples of 8, so these values are byte strings.

An implementation may further restrict the bit lengths of these inputs, consistent with the above requirements; for example, an implementation may establish smaller maximum values. The bit lengths that an implementation allows are called the supported bit lengths. A single set of supported bit lengths for each of the three inputs should be established for the entire implementation, independent of the key.

For IVs, it is recommended that implementations restrict support to the length of 96 bits to promote interoperability, efficiency, and simplicity of design.

7.3.2.1.2 Output Data

The following two-bit strings comprise the output data of the authenticated encryption function:

- A ciphertext denoted C, whose bit length is the same as that of the plaintext.
- An authentication tag, or tag, for short, is denoted T.

The bit length of the tag, denoted t, is a security parameter. In general, t may be any one of the following five values: 128, 120, 112, 104, or 96.

7.3.2.2 Authenticated Decryption Function

Given the selection of an approved block cipher, key, and an associated tag length, the inputs to the authenticated decryption function are values for *IV*, *A*, *C*, and *T*. The output is one of the following:

- The plaintext P that corresponds to the ciphertext C, or
- A special error code denoted *FAIL*.

The output P indicates that T is the correct authentication tag for IV, A, and C; otherwise, the output is *FAIL*.

The values for len(*C*), len (*A*), and len(*IV*) that an implementation supports for the authenticated decryption function shall be the same as the values for len(*P*), len (*A*), and len(*IV*) that the implementation supports for the authenticated encryption function.

7.3.3 Primitives for Confidentiality and Authentication

The mechanism for the confidentiality of the plaintext within GCM is a variation of the Counter mode, with a particular incrementing function, denoted $inc32$, for generating the necessary

sequence of counter blocks. The first counter block for the plaintext encryption is generated by incrementing a block that is generated from the IV.

The authentication mechanism within GCM is based on a hash function, called GHASH, which features multiplication by a fixed parameter, called the hash subkey, within a binary Galois field. The hash subkey, denoted H, is generated by applying the block cipher to the “zero” block. The resulting instance of this hash function, denoted GHASH*H*, is used to compress an encoding of the AAD and the ciphertext into a single block, which is then encrypted to produce the authentication tag.

7.4 Mathematical Components of GCM

This section presents the mathematical components that appear in the specifications of the authenticated encryption and authenticated decryption functions

7.4.1 Multiplication Operation on Blocks

Let R be the bit string 11100001 || 0^{120}. Given two blocks, X and Y, Algorithm 1 below computes a product block:

Input: Blocks *X*, *Y*.

Output: Block *X* •*Y*.

7.4.2 GHASH Function

Algorithm 2 below specifies the GHASH function:

Prerequisites: Block H, the hash subkey.

Input: Bit string X such that $len(X) = 128m$ for some positive integer m.

Output: Block GHASH*H* (*X*).

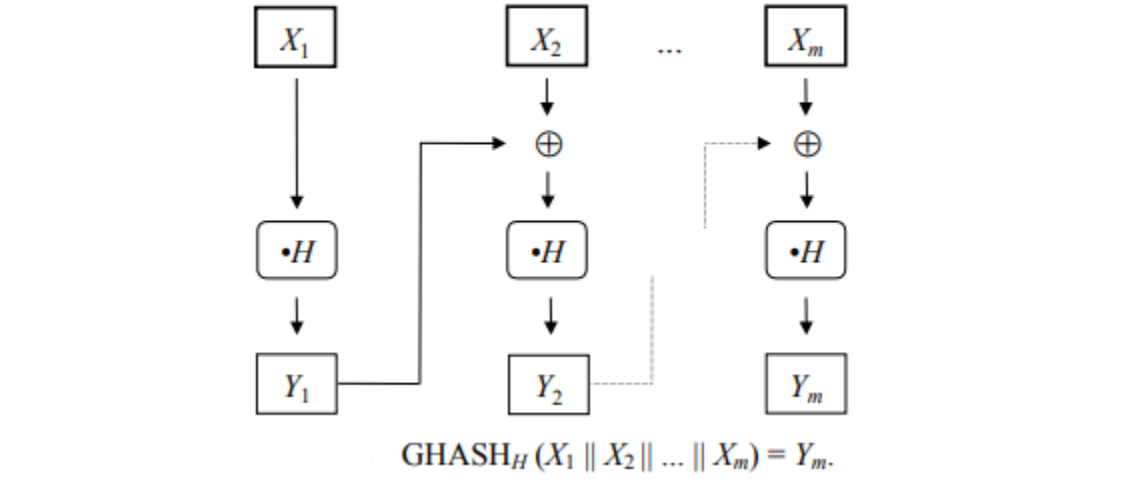


Fig 7.1: GHASH Function

7.4.3 GCTR Function

Algorithm 3 below specifies the GCTR function. The suggested notation does not indicate the choice of the underlying block cipher.

Prerequisites: Approved block cipher CIPH with a 128-bit block size; key *K*.

Input: Initial counter block *ICB*; Bit string *X*, of arbitrary length.

Output: Bit string *Y* of bit length len(*X*).

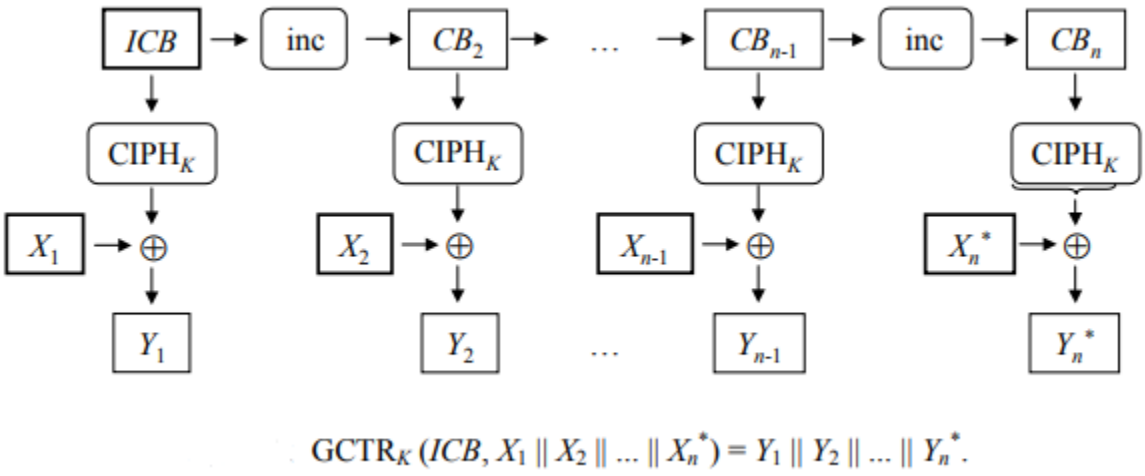


Fig 7.2: GCTR Function

7.5 GCM Specification

The specifications include the inputs, the outputs, the steps of the algorithm, diagrams, and summaries. The suggested notation does not indicate the choice of the underlying block cipher. The inputs that are typically fixed across many invocations of the function are called the prerequisites; however, some of the prerequisites may also be regarded as (varying) input. For both algorithms, equivalent sets of steps that produce the correct output are permitted.

7.5.1 Algorithm for the Authenticated Encryption Function

Algorithm 4 below specifies the authenticated encryption function:

Prerequisites: Approved block cipher CIPH with a 128-bit block size; key K; Definitions of supported input-output lengths; supported tag length t associated with the key.

Input: Initialization vector IV (whose length is supported); plaintext P (whose length is supported); Additional authenticated data *A* (whose length is supported).

Output: Ciphertext *C*; authentication tag *T*.

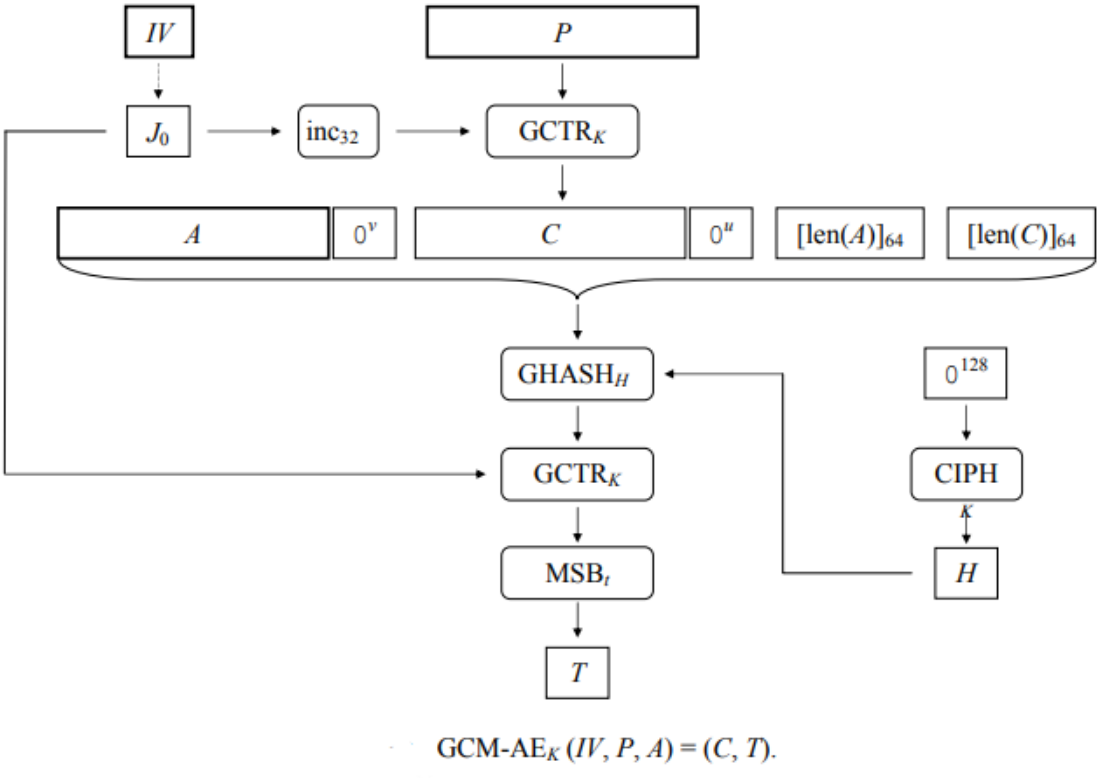


Fig 7.3: Authenticated-Encryption Function

7.5.2 Algorithm for the Authenticated Decryption Function

Algorithm 5 below specifies the authenticated decryption function:

Prerequisites: Approved block cipher CIPH with a 128-bit block size; key K; Definitions of supported input-output lengths; supported tag length *t* associated with the key.

Input: Initialization vector IV; ciphertext C; Additional authenticated data A; authentication tag T.

Output: Plaintext P or indication of inauthenticity *FAIL*.

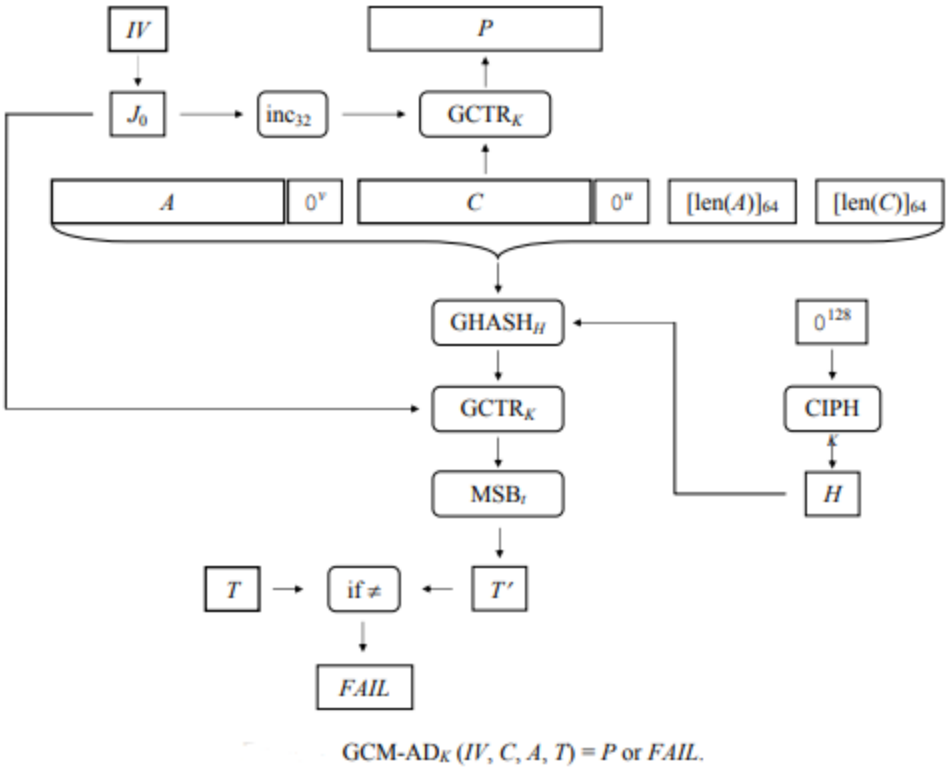


Fig 7.4: Authenticated-Decryption Function

7.6 Uniqueness requirement on IVs

The IVs in GCM must fulfill the following “uniqueness” requirement:

The probability that the authenticated encryption function will ever be invoked with the same IV and the same key on two (or more) distinct sets of input data shall be no greater than 2^{-32}.

Compliance with this requirement is crucial to the security of GCM. Across all instances of the authenticated encryption function with a given key, if even one IV is ever repeated, then the

implementation may be vulnerable to forgery attacks. In practice, this requirement is almost as important as the secrecy of the key.

7.7 Authentication Assurance

The creation of an authentication tag by the authenticated encryption function provides the mechanism whereby assurance of the authenticity of the plaintext and AAD (and IV) can be obtained upon the execution of the authenticated decryption function. The nature of this assurance depends on the output of the authenticated decryption function:

- If the output is plaintext, then the design provides strong, but not absolute, assurance that the purported source of the data created the tag, i.e., that the plaintext and the AAD (and the IV and T) are authentic. Consequently, the mode also provides strong assurance that this information was not subsequently altered, either intentionally or unintentionally.
- If the output fails, certainly, at least one of the given inputs (i.e., C, A, IV, or T) is not authentic.

In the first case, the assurance is not absolute because forgeries are possible, in principle. In other words, an adversary, i.e., a party without access to the key or to the authenticated encryption function, may be able to produce the correct tag for some triple of C, A, and IV. As with any tag-based authentication mechanism, if the adversary chooses a t-bit tag at random, it is expected to be correct for given data with probability $\frac{1}{2^t}$.

Independent of this attack, an adversary may attempt to systematically guess many different tags for a given input to authenticated decryption and thereby increase the probability that one (or more) of them, eventually, will be accepted as valid. For this reason, the system or protocol that implements GCM should monitor and, if necessary, limit the number of unsuccessful verification attempts for each key.

Moreover, as with most block cipher modes of operation, the security assurance of GCM degrades as more data is processed with a single key. Therefore, the total number of blocks of plaintext and AAD that are protected by invocations of the authenticated encryption function during the key's lifetime should be limited.

8 Simulations and Results

To carry out a sanity check of our proposed security enhancements, we first simulated a Bit-Flipping attack in MATLAB to gauge the effectiveness of this attack. We create a mockup of sensors measuring temperature and state-of-charge in a battery charging system. The data from these sensors were then created into a payload message of 16 bytes.

```
Plaintext Message Creation

5   plaintext_msg = "{T:" + temperature + "C, SOC:" + soc + "%}"

    plaintext_msg = "{T:24C, SOC:65%}"

6   plaintext = convertStringsToChars(plaintext_msg);
```

Fig 8.1: Plaintext message creation using temperature and SOC information received from model sensors

This payload was then encrypted using AES-128 (AES with a 128-bit key). This encrypted message was modulated using Chirp Spread Spectrum, and to test out the robustness of this modulation technique, Gaussian White Noise was added.

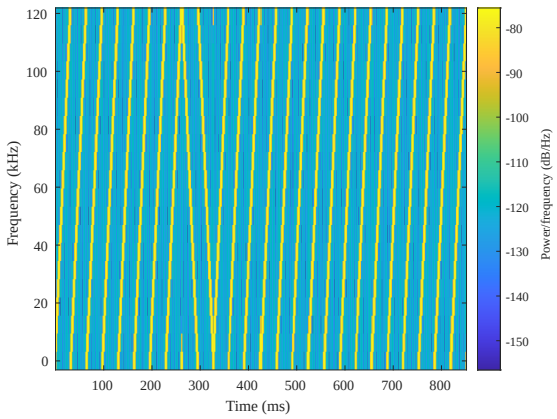


Fig 8.2: Encrypted and Modulated LoRa Signal

```
8   ciphertext = cipher(ascii_plaintext, w, s_box, poly_mat, 1)

    ********************************************
    *                                          *
    *              C I P H E R                 *
    *                                          *
    ********************************************

    Initial state :                 7b 34 53 36
                                    54 43 4f 35
                                    3a 2c 43 25
                                    32 20 3a 7d

    Initial round key :             00 04 08 0c
                                    01 05 09 0d
                                    02 06 0a 0e
                                    03 07 0b 0f

    State at start of round 1 :     7b 30 5b 3a

     ciphertext = 1×16
          37   254   254     2    17   155    51     2    13   137    51   121   159   167   152    43

9   char(ciphertext)

    ans =
        '%þþ3
         3y§+'
```

Fig 8.3: Encryption of message using AES-128

At the receiver's end, this message was demodulated successfully, showing that LoRa can work in severe noise conditions. A bit in this demodulated message was then flipped to emulated Bit-Flipping Attack, considering that the network has been compromised. This tampered message was then decrypted to obtain a garbled message, which shows the effect of the Bit-Flipping Attack.

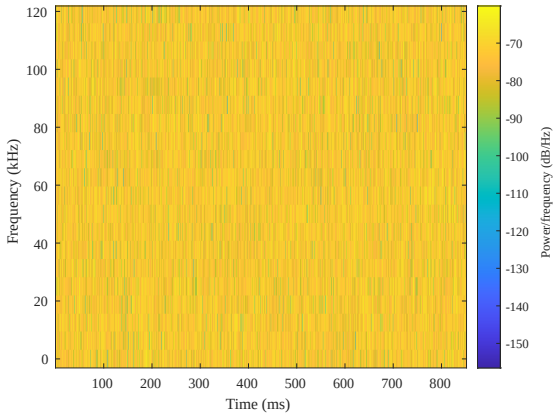


Fig 8.4: Demodulated Signal Received. LoRa works even under heavy noise

```
81  demodulated_payload = payload(11:end)

    demodulated_payload = 1×16
        37   254   254     2    17   155    51     2    13   137    51   121   159   167   152    43

82  ciphertext

    ciphertext = 1×16
        37   254   254     2    17   155    51     2    13   137    51   121   159   167   152    43
```

Fig 8.5: Demodulated payload matched the ciphertext

```
87  char(decrypted_plaintext)

    ans =
        ')p^;$zA;£

88  plaintext_msg

    plaintext_msg = "{T:24C, SOC:65%}"
```

Fig 8.6: Decrypted message does not match the initial plaintext message due to bit-flipping attack.

Our proposed encryption method is AES in GCM. To check out its effectiveness, we built a prototype in Python3. In GCM mode, an authentication code called GMAC is appended to the ciphertext. GMAC allows ciphertext authentication to check if the message has been tampered with. If the message has been tampered with, decryption does not take place, and we are alerted; on the other hand, untampered messages are decrypted without any obstruction.

```
319
320  # Encryption
321  C, Tag = Encryption(P, IV, K, A)
322
323  # Bit-Flipping Attack
324  # C = BitFlippingAttack(C)
325
326  # Decryption
327  decrypted_plaintext = Decryption(C, Tag, K, A, IV)
328
329  print("Original plaintext: ", P)
330  print("Decrypted plaintext: ", decrypted_plaintext)
331
```

```
BitFlippingAttack()
Run:  GCM ×
C:\Python39\python.exe "E:\Class\EE491 {BTP}\Galois Counter Mode\GCM.py"
Original plaintext:  {T:32C, SOC:91%}
Decrypted plaintext:  {T:32C, SOC:91%}

Process finished with exit code 0
```

Fig 8.7: Absence of adversary: no bit-flipping attack

Fig 8.8: Adversary performs a bit-flipping attack; It is detected by comparing auth tag generated by GCM

9 Conclusion

Thus, with this, we can conclude that the current implementation of AES in LoRaWAN 1.1 is insufficient to provide security against bit-flipping attacks or man-in-the-middle attacks. We propose the implementation of AES in Galois Counter Mode so that we can avoid the effect of such adversaries. However, only implementing AES with GCM is not enough as it still results in the loss of packets under a bit-flipping attack, which can greatly slow down the transfer rate as well as reduce the robustness of the network. We also call out the need for better protection standards to eliminate the possibility of Man-in-the-middle attacks.

References

[1] *State Wise Map*. (n.d.). State Wise Map. https://www.nsgm.gov.in/en/state-wise-map

[2] Henke, C. (2022, July 28). *IoT Security: Risks, Examples, and Solutions*. EMnify Blog. https://www.emnify.com/blog/iot-security

[3] Electric, V. (2021, July 24). *How LoRa Modulation really works - long range communication using chirps* [Video]. YouTube. https://youtu.be/jHWepP1ZWTk

[4] Mohamed, Ali et al. "Enhancing Cyber Security of LoRaWAN Gateways under Adversarial Attacks." *Sensors (Basel, Switzerland)* vol. 22,9 3498. 4 May. 2022, doi:10.3390/s22093498

[5] X. Yang, E. Karampatzakis, C. Doerr and F. Kuipers, "Security Vulnerabilities in LoRaWAN," 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI), 2018, pp. 129-140, doi: 10.1109/IoTDI.2018.00022.

[6] JungWoon Lee, DongYeop Hwang, JiHong Park and Ki-Hyung Kim, "Risk analysis and countermeasure for bit-flipping attack in LoRaWAN," 2017 International Conference on Information Networking (ICOIN), 2017, pp. 549-551, doi: 10.1109/ICOIN.2017.7899554.

[7] Computerphile (2020, May 21). *Modes of Operation.* YouTube. https://youtu.be/Rk0NIQfEXBA

[8] Computerphile (2019, Nov 22). *AES Explained (Advanced Encryption Standard).* YouTube. https://youtu.be/O4xNJsjtN6E