PERFORMANCE MEASUREMENT OF VARIOUS SECURITY MECHANISMS AND SIMULATION OF CHACHA ENCRYPTION AND BLAKE2S INTEGRITY ALGORITHM FOR THE LORAWAN NETWORK

A PROJECT REPORT

Submitted by

R ABHINAV	**2019504502**
K UDAY KUMAR REDDY	**2019504542**
AKHILESH CHANDRA	**2019504611**

in partial fulfillment for the award of degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY :: CHENNAI 600 044

MAY 2023

ANNA UNIVERSITY : CHENNAI 600044

BONAFIDE CERTIFICATE

Certified that this project **" PERFORMANCE MEASUREMENT OF VARIOUS SECURITY MECHANISMS AND SIMULATION OF CHACHA ENCRYPTION AND BLAKE2S INTEGRITY ALGORITHM FOR THE LORAWAN NETWORK "** is the bonafide work of

R ABHINAV	**2019504502**
K UDAY KUMAR REDDY	**2019504542**
AKHILESH CHANDRA	**2019504611**

who carried out the project work under my supervision.

SIGNATURE

Dr. P. INDUMATHI
HEAD OF THE DEPARTMENT
Professor,
Department of Electronics Engineering,
Madras Institute of Technology,
Anna University,
Chennai - 600 044

SIGNATURE

Dr. T.SUBASHRI
SUPERVISOR
Assistant Professor,
Department of Electronics Engineering,
Madras Institute of Technology,
Anna University,
Chennai - 600 044

ACKNOWLEDGEMENT

We consider it as our privilege and our primary duty to express our gratitude and respect to all those who guided and inspired us in the successful completion of the project.

We owe solemn gratitude to **Dr. J. PRAKASH** , Dean, Madras Institute of Technology, for having given consent to carry out the project work at MIT Campus, Anna University.

We wish to express our sincere appreciation and gratitude to **Dr. P. INDUMATHI,** Professor and Head of the Department of Electronics Engineering, who has encouraged and motivated us in our endeavours.

We are extremely grateful to our project guide **Dr .T. SUBASHRI**, Assistant Professor, Department of Electronics Engineering, for her timely and thoughtful guidance and encouragement for the completion of the project.

We sincerely thank all our panel members **Ms. G.SUMITHRA** and **Dr . T. SUBASHRI** for their valuable suggestions.

We also thank our project coordinator **Mrs. K. KARTHIKA** for her co-ordination for the project work presentation.

We also thank all the teaching and non-teaching staff members of the Department of Electronics Engineering for their support in all aspects.

Place: **R ABHINAV** **2019504502**

Date: **K UDAY KUMAR REDDY** **2019504542**

AKHILESH CHANDRA **2019504611**

ABSTRACT

Low Power Wide Area Network (LPWAN) protocols have been proposed for supporting resource-constrained Internet of Things (IoT) devices by negligible power consumption. Long Range Wide Area Network (LoRaWAN) is a low power communication protocol based on LoRa which uses Chirp spread spectrum in Physical Layer of LoRaWAN . LoRaWAN supports security algorithms for message encryption and authentication. The challenges in LoRaWAN relate to complexity of AES algorithm used and its security challenges. This motivates to this project work to introduce a new light weight and efficient security mechanism which minimizes complexity and removes security vulnerabilities of LoRaWAN. In this project, simulation of LoRaWAN protocol stack using the Network Simulator (NS-3) is completed and cryptographic algorithms provided by CryptoPP library are tested to obtain the most efficient and secure algorithm which is then implemented into LoRaWAN network.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS,ABBREVIATIONS AND NOMENCLATURE

SYMBOL & ABBREVIATION	EXPANSION
LoRaWAN	Long Range Wide Area Network
CSS	Chirp Spread Spectrum
NS3	Network Simulator 3
MAC	Medium Access Control
CRC	Cyclic Redundancy Check
MIC	Message Integrity Code
AES ECB	Advanced Encryption Standard Electronic Code Book mode
AES CTR	Advanced Encryption Standard Counter mode
AES CMAC	Advanced Encryption Standard Cipher based Message Authentication Code
G	Network Traffic
S	Throughput as a function of G
L	Path Loss of LoRa Channel
n	Path Loss Exponent
L_0	Path Loss of LoRa Channel at reference distance

d	Distance between sender and receiver
T_p	Propagation Delay of LoRa Channel
P_r	Received Power for LoRa Modulated Signal
P_t	Transmitted Power for LoRa Modulated Signal
$n_{preamble}$	Number of preamble bits required to synchronize data
$T_{preamble}$	Preamble duration to reach the receiver
PayloadSymbNb	Number of symbols in payload period
T_{sym}	Symbol or Chirp duration
$T_{payload}$	Payload and header duration to reach the receiver
T_{total}	Total time on air for the packet

CHAPTER 1
INTRODUCTION

Internet of Things (IoT) describes the network of physical objects or things that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. Many communication protocols have been proposed to support different types of IoT networks. Low Power Wide Area Networks (LPWAN) technologies have been proposed to satisfy the IoT constrained devices requirements and are maintained to enable a wide range of communication with low power consumption.

LoRaWAN(Long Range Wide Area Network) is a LPWAN communication protocol used in unlicensed ISM(Industrial, Scientific and Medical) band which uses LoRa modulation in its physical layer. LoRa modulation uses CSS(Chirp Spread Spectrum) where CHIRP(Compressed High Intensity Radio Pulse) pulse having same amplitude is increasing or decreasing in frequency along with time. LoRaWAN is MAC layer protocol that defines and controls how data is communicated in the network where devices use LoRa modulation. LoRaWAN network architecture is of star of stars topology consisting of end devices which send data to gateways through LoRa modulation .Gateways forward data to network server which monitors end devices, gateways and application server and authenticates the message and sends it to application server to decode the message.

LoRaWAN has a set of security features with set of keys used for communication between end device , network server and application server with AES CTR used for encryption and AES CMAC for message authentication . A join procedure is also defined to join end devices with the network server so that it can send data to that network server. But security problems exist leaving it vulnerable to attacks like replay attacks, key compromise attacks etc. AES is computationally heavy which is not desired in low power devices.

Goal of this project is to create an efficient and secure security mechanism for LoRaWAN capable of overcoming the security issues present while maintaining or even improving its performance. For that purpose, lightweight security algorithms are studied and a LoRaWAN Protocol stack is implemented in NS3(Network Simulator 3) to simulate a real life LoRaWAN network .

After Studying various algorithms that are compatible with LoRaWAN for confidentiality and Integrity, Suitable algorithms were chosen to replace AES CTR and AES CMAC algorithms keeping in mind that algorithm occupies less space as well as it is fast enough compared to others hence maintaining better software efficiency as a ratio of speed and space occupation. Security algorithm implemented is more light weight and less vulnerable to attacks when compared to preexisting algorithms that were already present . After implementation of lightweight and secure LoRaWAN network, performance of the implemented network is analyzed.

CHAPTER 2
LITERATURE REVIEW

This chapter presents the literature completed on the works completed on the LoRaWAN-IoT. Following papers are referred for study of LoRa and LoRaWAN. Devlal et al [1] presented the work of comparison of LoRaWAN protocol with other popular communication protocols like WiFI, Zigbee, Nb-IoT etc. In this paper, the description of LoRa network architecture is discussed and also presented features of physical layer and MAC layer are explained clearly. An also presented the security aspects of LoRaWAN and its Applications along with its limitations are given at the end. Augustin et al [2] explains in detail about LoRa Protocol Stack and more detailed explanation and experimental analysis of physical layer and MAC layer parameters are done. This paper provides a more in-depth experimental analysis of physical and MAC layer parameters for better understanding of LoRaWAN. Silva et al [3] compares various modules used for LoRaWAN network simulation in NS3(Network Simulator 3) and describes about experimental results obtained and user experience thus helping in finalizing which module can be used for simulation of LoRaWAN protocol stack in NS3. Following papers are referred for understanding security features and security limitations of LoRaWAN. Aras et al [4] explores security vulnerabilities in LoRaWAN protocol stack and describes about attacks(Key compromise, Replay, Jamming, Wormhole attacks) thus providing a basic insight into security vulnerabilities in LoRaWAN which need to be rectified. X. Yang et al [5] explores various attacks in LoRaWAN like acknowledgement spoofing , bit flipping attack etc and sample experiments are done for each attack with suggestions for stopping those attacks. Following papers are referred for understanding various solutions given to improve security of LoRaWAN . Chen et al [6] proposes a detailed key management scheme for LoRaWAN with simplified rabbit cipher used for session key generation which is compared with key generation scheme used in LoRaWAN (AES ECB) to show

improvement in performance. Hakeem et al [7] proposes a key management algorithm for LoRaWAN implemented in NS3 where hash chains are created based on SHA 256 after initial key exchange for root key update. Salting (XOR ing) of key with random number is done to protect key from key compromise attacks. Following papers are referred for understanding and studying performance of light weight security algorithms. Bharadwaj et al [8] explores various lightweight security algorithms like SIMON ,SPECK,ECC etc and various parameters like energy, gate area, throughput and security attacks are compared and studied .Thakor et al [9] compares various lightweight cryptographic algorithms and compares implementation efficiency in hardware as a ratio of throughput and gate area and in software as a ratio of throughput and space occupied . Security attacks for various algorithms are also studied. Following papers are referred for understanding security vulnerabilities and cryptanalysis done on the light weight security algorithms considered. Jian et al [10] provides complete security analysis of various possible attacks like collision attack, near collision attack, weak preimage/key attack etc on Blake2 algorithm and its reduced rounds. Shotaro et al[11] provides differential cryptanalysis of reduced round ChaCha cipher based on Probabilistic Neural bits. Time complexity ,space complexity and success probability of attack on reduced rounded Chacha cipher is calculated. San et al[12] provides a way to improve preimage attacks on full Tiger Hash function and SHA-256 function by using Meet in the middle method to split function into 2 parts and trying to find exact matches. Following paper is referred for understanding performance of LoRaWAN network in NS3. Magrin et al[13] compares throughput taken as a function of network traffic for various scenarios like variations in spreading factor, duty cycle limitations etc to understand performance of LoRaWAN network in NS3 for smart city scenario.

CHAPTER 3
OVERVIEW OF LORA AND LORAWAN

This chapter presents the overview of the new IoT technology of LoRa and LoRaWAN.

3.1 PHYSICAL LAYER OF LORA TECHNOLOGY

LoRa is a wireless modulation technique derived from Chirp Spread Spectrum (CSS) technology. It encodes information on radio waves using chirp pulses .

CSS is a spread spectrum technique uses wideband linear frequency modulated chirp pulses to encode information. A chirp is a sinusoidal signal whose frequency increases or decreases over time. In the chirp spread spectrum modulation, the wanted data signal is multiplied with the chirp signal . This spreads the bandwidth beyond the bandwidth of the original data signal. At the receiver end the received signal is re-multiplied with the locally generated copy of the chirp signal. This compresses the modulated signal back to the original bandwidth.

Data can be transmitted at a longer range compared to technologies like WiFi, Bluetooth or ZigBee. LoRa is ideal for applications that transmit small chunks of data with low bit rates over a long distance. LoRa can be operated on the license free ISM (Industrial Scientific Medical) bands that are reserved internationally for industrial, scientific, and medical purposes . Example:- 915 MHz, 868 MHz, and 433 MHz.

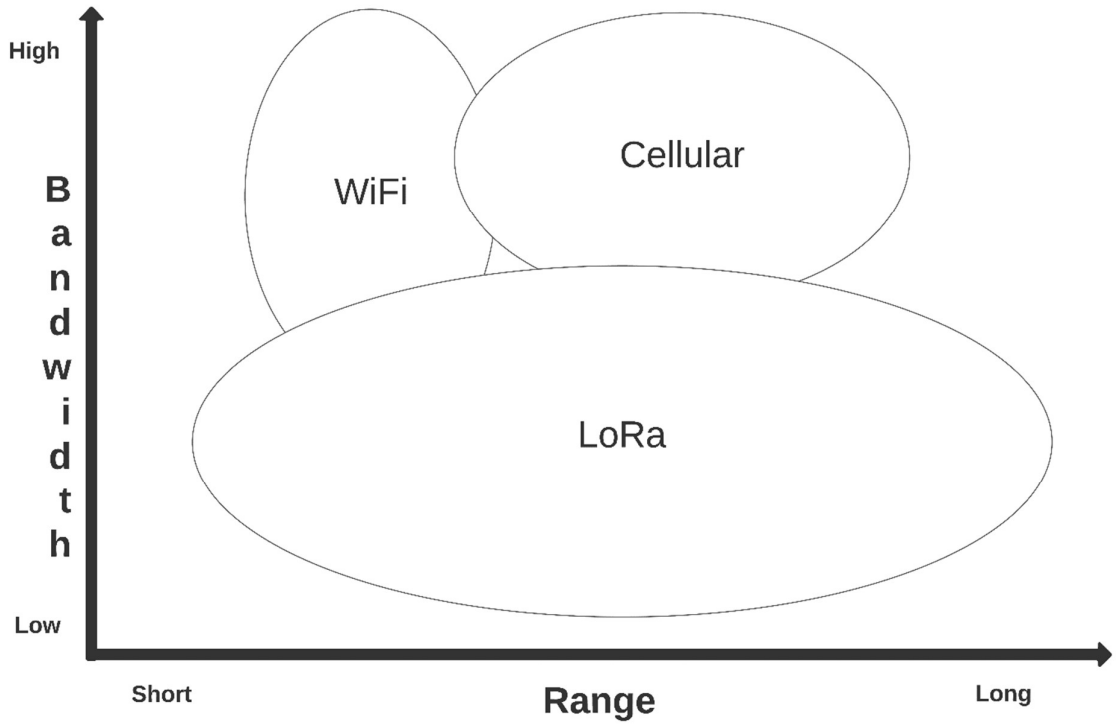

Figure 3.1: Comparison of range and bandwidth of various wireless technologies

3.2 LORAWAN PROTOCOL

LoRaWAN is a networking MAC(Medium Access Control) layer protocol that uses LoRa modulation scheme. It consists of end device, gateways and network server. It follows star of stars topology and connects end device and gateway through LoRa modulation and gateway and network server through IP connection.

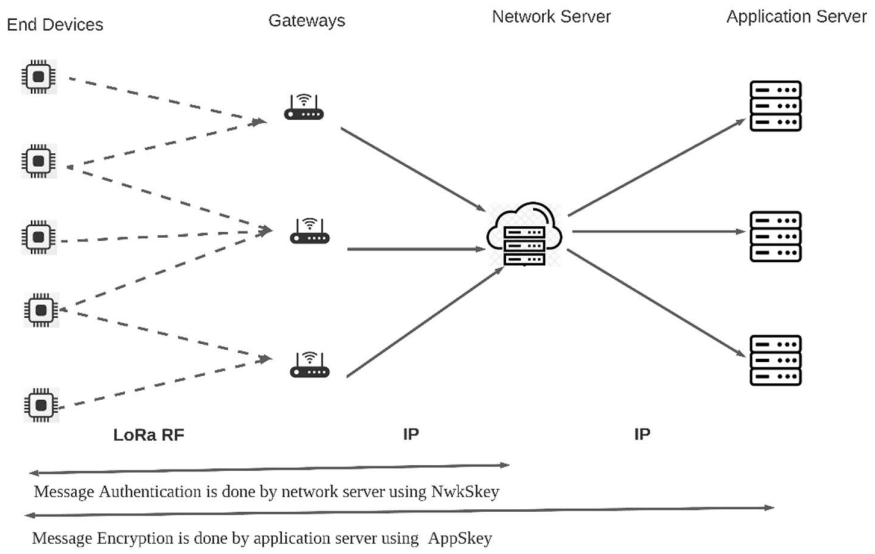

Figure 3.2: LoRaWAN architecture

3.2.1 END DEVICE CLASSES IN LORAWAN

End device in LoRaWAN are of 3 types Class A, Class B and Class C based on their receiving slots present in them.

Class A end devices have 2 receiving windows with fixed time intervals and delays from original transmission window based on operating region to receive downlink frames after uplink transmission. In addition to the 2 receive windows in Class A, Class B provides additional receive windows created periodically a beacon till it is over. Class C end devices have a receiving window which opens immediately after transmission and stays open till next transmission window is opened.

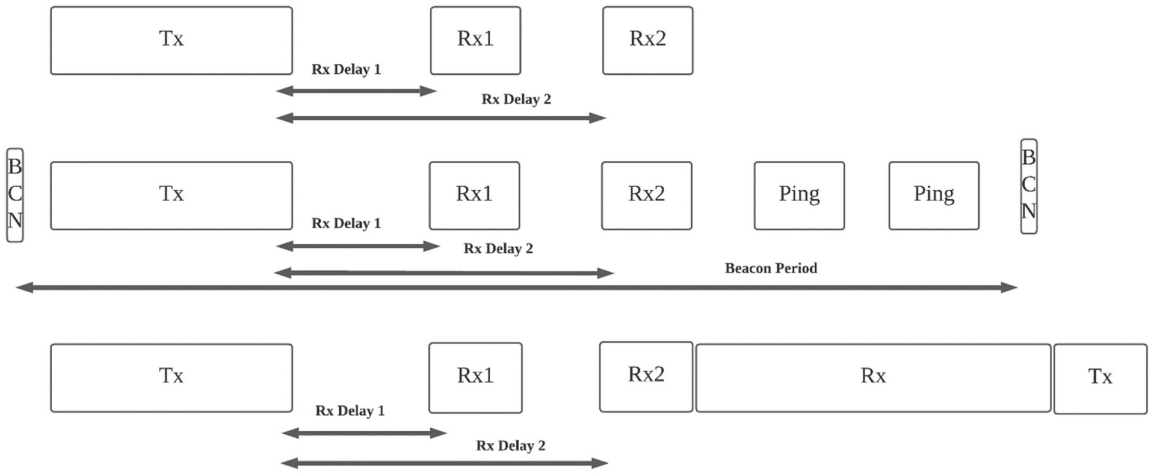

Figure 3.3 : LoRaWAN end device classes

3.2.2 MESSAGE TYPES AND FRAME FORMATS IN LORAWAN

Message formats are used to transport MAC commands and application data.Uplink messages are sent by end devices to the Network Server through one or many gateways. Each downlink message is sent by the Network Server to only one end device by a single gateway. Join procedure in OTAA(Over The Air Activation) for end device activation uses join request and join accept message formats. Join-request message is initiated by an end device and sent to the Network Server to join into the network . Join-accept message is generated by the Network Server to indicate joining of the end device into the network. There are 4 other data message types used in LoRaWAN. These data message types are used to transport both MAC commands and application data which can be combined together in a single message. Data messages can be confirmed or unconfirmed.

Message Type ID	Message Type	Comments/Function
0	Join Request	Uplink message used in Over the Air Activation Procedure by an End Device to request a network Server to join to it
1	Join Accept	Downlink message used in Over the Activation Procedure when Network Server Accept the Join Request of End Device
2	Unconfirmed Data Uplink	Uplink Data Message for which Confirmation is Not required
3	Unconfirmed Data Downlink	Downlink Message for which Confirmation is not required
4	Confirmed Data Uplink	Uplink Message for which Confirmation is required
5	Confirmed Data Downlink	Downlink Message for which Confirmation is required
7	Proprietary	Used to implement non- standard message formats

Table 3.1: Message types in LoRaWAN with purpose

Frame format in physical layer has preamble for 8 symbols to indicate receiver that packet is about to arrive and 4.25 symbols are used for synchronization word (0x34). Physical layer header with coding rate, length of payload along with its CRC(Cyclic Redundancy Check) with a combined length of 8 symbols for header verification. Physical layer payload is present having the necessary information to be carried. CRC for 4 bytes is also added for the overall packet during uplink for packet verification.

Phy Layer (Size in bytes)	Preamble (8 Symbols)	Synchronisation Word (4.25 Symbols)	PHDR (8 Symbols)	PHDR_CRC (8 Symbols)	PHY Payload (L bytes)	CRC [Uplink only] (2 bytes)

Figure 3.4 a : Frame format of a packet in physical layer of LoRaWAN

Physical layer payload consists of MAC header for 1 byte specifying the message type in 3 bits and major version of frame format used in 2 bits It also has either join request message or join accept message or MAC payload of message depending on the requirements.

Phy Payload (Size in bytes)	MHDR (1)	MAC Payload or Join Accept or Join Request (0….N)	MIC (4)

Figure 3.4 b: Frame format for physical layer payload of a packet in LoRaWAN

MHDR (Size in bits)	MType (3)	RFU (3)	Major (2)

Figure 3.4 c: Frame format for MAC header of a packet in LoRaWAN

MAC layer payload consists of frame header of 7 to 22 bytes specifying uplink or downlink frame count in 2 bytes , device address in 4 bytes , control field in 1 byte and optional field to send MAC commands in 15 bytes . Frame payload is also present in the MAC payload and MIC(Message Integrity Code) is calculated with NwkSkey for message authentication by network server or with Appkey in join procedure for device authentication by network server.

MAC Payload (Size in bytes)	FHDR (7..22)	FPort (0..1)	FRMPayload (0…N)

Figure 3.4 d: Frame format for MAC payload of a packet in LoRaWAN

FHDR (Size in bytes)	DevAddr (4)	FCtrl (1)	FCnt (2)	Fopts (0..15)

Figure 3.4 e: Frame format for Frame Header of MAC payload in LoRaWAN

Frame control field has a bit to indicate whether ADR(Adaptive data rate has been enable or not. In case of uplink frames, a bit is used to check whether ADR acknowledgement request is set or not which is set after 64 frames are sent with ADR without receiving an acknowledgement. Acknowledgement bit is present in both uplink and downlink frames. For downlink frames, no of pending downlink frames is indicated by a bit . Length of frame options is indicated by 4 bits for both uplink and downlink.

FCtrl(Uplink) (Size in bits)	ADR (7)	ADRACKReq (6)	ACK (5)	RFU (4)	FOptsLen (3..0)

Figure 3.4 f: Frame format for Frame Control field of an uplink message in LoRaWAN

FCtrl(Downlink) (Size in bits)	ADR (7)	RFU (6)	ACK (5)	Fpending (4)	FoptsLen (3…0)

Figure 3.4 g: Frame format for Frame Control field of a downlink message in LoRaWAN

Join request message sent by end device to network server has 8 bytes of application identifier and 8 bytes of device identifier which are unique address for each end device along with a random number generated called DevNonce for 2 bytes.

Join Request (Size in bytes)	AppEUI (8)	DevEUI (8)	DevNonce (2)

Figure 3.4 h: Frame format for Join request message in LoRaWAN

Join accept message sent by network server to end device after end device verification . It has 3 bytes of random number called AppNonce along with 3 bytes network ID and 4 bytes of device address generated by the network server for that end device in its network . It also has a byte for delay between transmitter and receiver , a byte for down link configuration settings with 4 bits indicating data rate for second receive window and 3 bits indicating change in data rate between uplink and downlink to set it for the first receive window and optional 16 bytes for list of channel frequencies to be used other than frequencies defined based on device location.

Join Accept (Size in bytes)	AppNonce (3)	NetID (3)	DevAddr (4)	DLSettings (1)	RXDelay (1)	CFList [Optional] (16)

Figure 3.4 i : Frame format for Join Accept message in LoRaWAN

DLSettings (Size in bits)	RFU (8)	RX1DROffset (6..4)	RX2 Data Rate (3:0)

Figure 3.4 j : Frame format for Down link Settings in Join Accept message in LoRaWAN

CFList (Size in bytes)	Freq Ch4 (3)	Freq Ch5 (3)	Freq Ch6 (3)	Freq Ch7 (3)	Freq Ch8 (3)	RFU (1)

Figure 3.4 k : Frame format for Channel Frequency list in Join accept message in LoRaWAN

3.2.3 VERFICATION OF END DEVICE THROUGH NETWORK ACTIVATION

LoRaWAN end device needs to be registered with the LoRaWAN network server before communication. Registration or activation into LoRaWAN network can be done in 2 methods namely Activation By Personalization and Over The Air Activation.

In Activation by Personalization, LoRaWAN end device has device address DevAddr , network session key NwkSkey and application session AppSkey and already stored in it. LoRaWAN Network server has DevAddr and NwkSkey of LoRaWAN end device already present in them. Application server has AppSkey of LoRaWAN end device in it. These are used for message authentication in LoRaWAN network server and message decryption in LoRaWAN application server.

In Over The Air Activation, a join request is sent by LoRaWAN end device to LoRaWAN network server with AppEUI, DevNonce and DevEUI fields. Message Integrity Code (MIC) is generated with these fields using AES CMAC algorithm in LoRaWAN end device which is then verified in LoRaWAN Network server for the received Join Request to ensure that authenticated LoRa end devices are connected to it. LoRaWAN network server will send a Join Accept message with the following fields namely AppNonce, DevAddr, RxDelay ,CFlist, DLSettings and NetID which are then used for LoRaWAN session key generation in LoRa end devices and network server.

3.2.4 EXISTING LORAWAN SECURITY SYSTEMS

The security services supported by LoRaWAN network is message authentication for verification of origin of message and encryption for confidentiality. LoRaWAN end device stores NwkSkey and AppSkey in Activation by Personalization procedure which also has a copy in LoRaWAN network server and LoRaWAN application server. It can also be generated through cryptographic parameters exchanged namely AppNonce, DevNonce and NetID during OTAA join procedure execution in LoRaWAN end device and LoRaWAN network server. Network server passes the AppSkey to Application server after key generation . Session Keys are generated through AppKey present in both LoRaWAN end device and LoRaWAN network server which is used as key for encrypting concatenated values of AppNonce, DevNonce, NetID padded to 16 bytes using AES ECB algorithm.

NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad16)

AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad16)

Figure 3.5 a : Algorithm for Session key generation in LoRaWAN end device after Join Procedure using Over The Air Activation

After successful generation of 2 keys NwkSkey and AppSkey , LoRaWAN payload is encrypted with AppSkey in end device which will be decrypted in the application server . In both cases, AES CTR algorithm is used for encryption confidentiality services where Ai is taken as the counter and k is taken as number of 16 byte divisions for the payload.

Ai (Size in bytes)	0x01 (1)	4 x 0x00 (4)	Dir (1)	DevAddr (4)	FCntUp or FCntDown (4)	0x00 (1)	i (1)

Figure 3.5 b : Frame Format for Ai used as counter in AES CTR algorithm for encryption and decryption of message in LoRaWAN

k=ceil(len(payload))/16

Si = aes128_encrypt(K, Ai) for (i = 1,2,..k)

S = (S1 | S2 | .. | Sk)

Cipher text= ((payload | pad16) ⊕ S)

Figure 3.5 c : Algorithm used for encryption of message in LoRaWAN

Message Integrity Code (MIC) using AES CMAC algorithm is generated for each message along with B0 combined before it using NwkSkey in end device which will be verified in the LoRaWAN network server . If MIC matches, it is verified that the received messages are not tampered by any 3rd person and also proves that origin of the message is from authenticated source .

B0 Size (in bytes)	0x49 (1)	4 x 0x00 (4)	Dir (1)	DevAddr (4)	FCntUp or FCntDown (4)	0x00 (1)	len(msg) (1)

Figure 3.5 d : Frame Format for B0 which is combined with the message in AES CMAC algorithm for authentication of message in LoRaWAN

cmac = aes128_cmac(NwkSKey, B0 | msg)

MIC = cmac[0..3]

Where msg=MHDR | FHDR | FPort | FRMPayload

Figure 3.5 e : Algorithm used in Message Integrity Code generation for authentication of message in LoRaWAN

CHAPTER 4

SIMULATION AND TESTING OF LORAWAN NETWORK PERFORMANCE USING LINUX BASED LORA PROTOCOL STACK ON NETWORK SIMULATOR TOOL (NS3)

This chapter explains the simulation of the LoRaWAN protocol stack and and LoRaWAN network deployment on NS3.

4.1 BASIC CLASS FILES FOR LORAWAN NETWORK DEPLOYMENT

LoRaWAN Protocol Stack is simulated on open source software tool NS3. NS3 module for LoRaWAN network comprises of two main classes namely **'LoraPhy class'** which is extended to '**EndDeviceLoraPhy'** and '**GatewayLoraPhy'** classes to model gateway and end device in physical layer and **'LorawanMac'** class is extended to '**EndDeviceLorawanMac'**, '**ClassAEndDevice LorawanMac'** and '**GatewayLorawanMac'** in order to model LoRaWAN gateway and LoRaWAN end device characteristics in MAC layer. **'NetworkServer'** class is created to receive packets from LoRaWAN Gateways and send acknowledgement to LoRaWAN End Device .

4.2 EXECUTION SCENARIO OF LINUX FILES ON NS3 SIMULATION TOOL

Simulation of LoRaWAN Stack is implemented by NS3 using C++ programming language. Two LoRaWAN End Devices, a LoRaWAN Gateway and a Network Server are implemented to complete the LoRaWAN network .Packet transmission from LoRaWAN End Device to Network Server is done and acknowledgement from LoRaWAN Network Server is received by LoRaWAN End Device.

4.2.1 HIERARCHY OF C++ INTERPRETER OF NS3

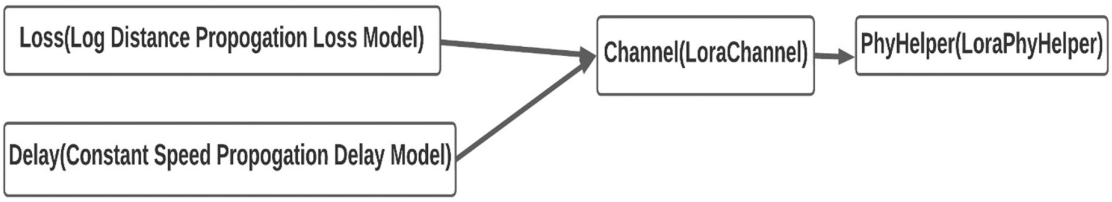

Fig 4.1 (a) : C++ Interpreter with objects used for scripting physical layer on NS-3.

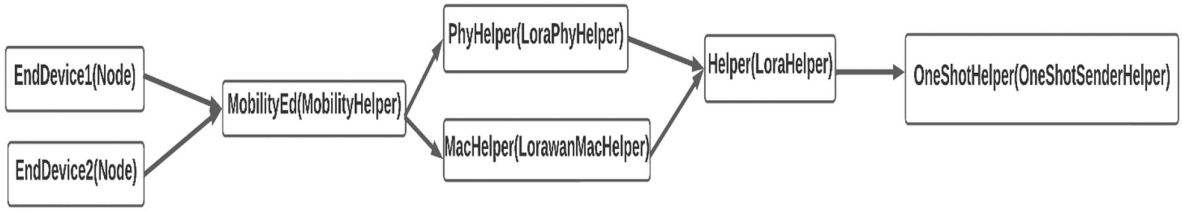

Fig 4.1 (b) : C++ Interpreter with objects used for End Device node deployment on NS-3

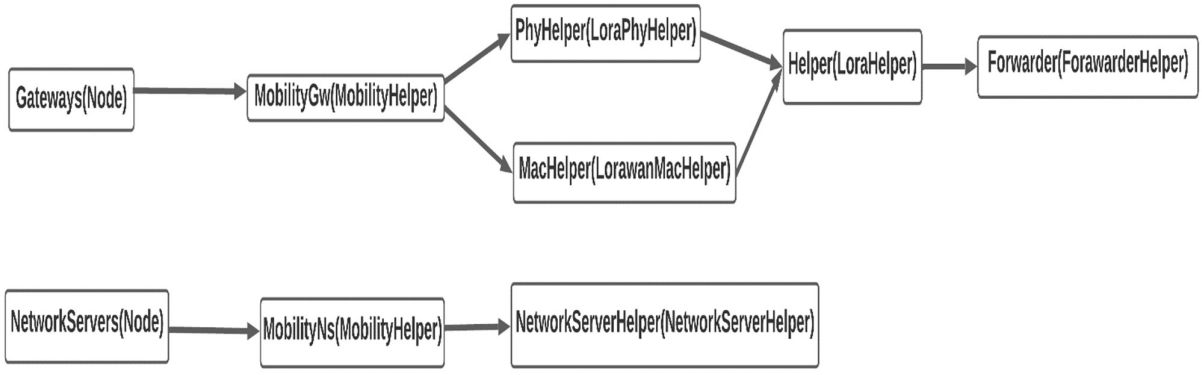

Figure 4.1 (c) : C++Interpreter with objects used for Gateway and Network Server node deployment on NS-3.

Figure 4.1 (a) shows the C++ Interpreter objects used for defining physical layer of LoRaWAN on the NS-3 Simulation Script. Figure 4.1 (b) shows C++ Interpreter objects used for End Device node deployment on NS-3 . For End Device deployment, '**PhyHelper**' and '**MacHelper'** class functions are called for creating physical layer and MAC layer for the End Device in NS-3 programming Scenario. It is then assigned for that node using '**LoraHelper**' class. '**OneShotSenderHelper**' class is used to send a packet from one LoRaWAN End Device when it is called. Figure 4.1 (c) shows C++ Interpreter objects used for Gateway and Network Server node deployment on NS-3. Procedure similar to deployment of End Device is repeated for creation of LoRaWAN Gateway. '**ForwarderHelper**' installs Forwarder in LoRaWAN Gateway which is used to send uplink data to Network Server and downlink data to the End Device. Finally, Network Server is created with '**NetworkServerHelper**' class.

4.2.2 SIMULATION RESULTS FOR LORAWAN NETWORK

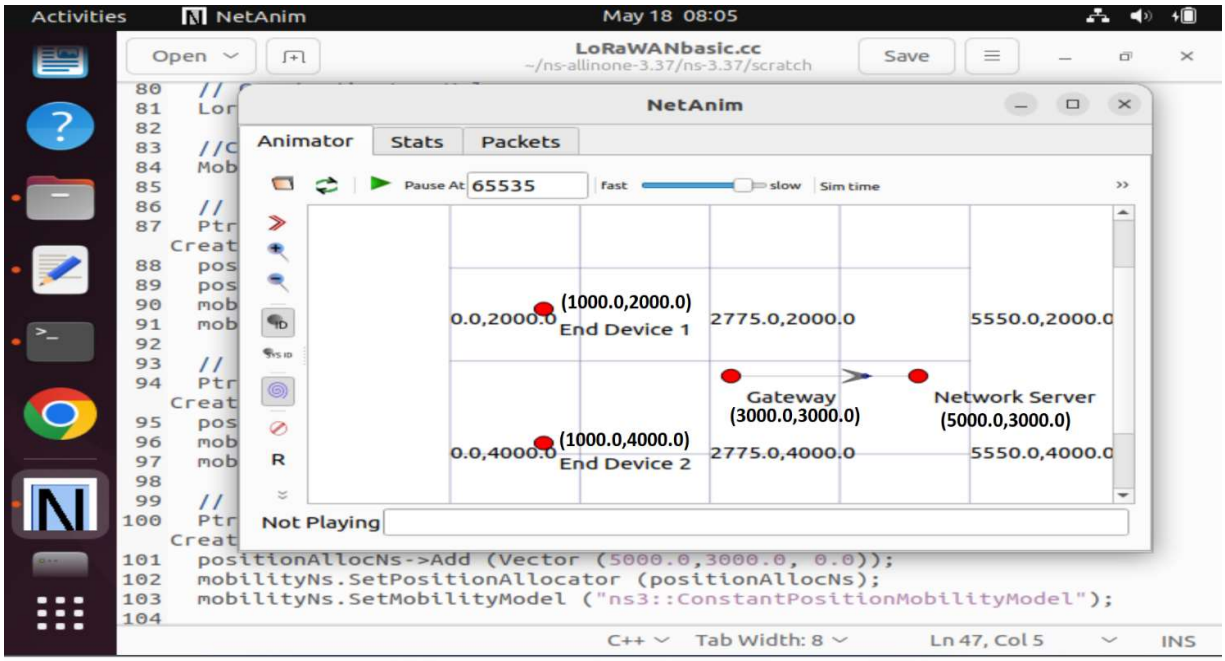

Figure 4.2 :Simulation Result of LoRaWAN network for the data transmission on uplink from End Device to Network Server

Figure 4.2 shows the simulation of LoRaWAN network for the data transmission on uplink from End Device to Network Server. Here, position of end devices are identified as (1000.0,2000.0) and (1000.0,4000.0) respectively in X-Y coordinates. Postion of Gateway and Network Server are identified as (3000.0,3000.0) and (5000.0,3000,0) respectively in X-Y coordinates.

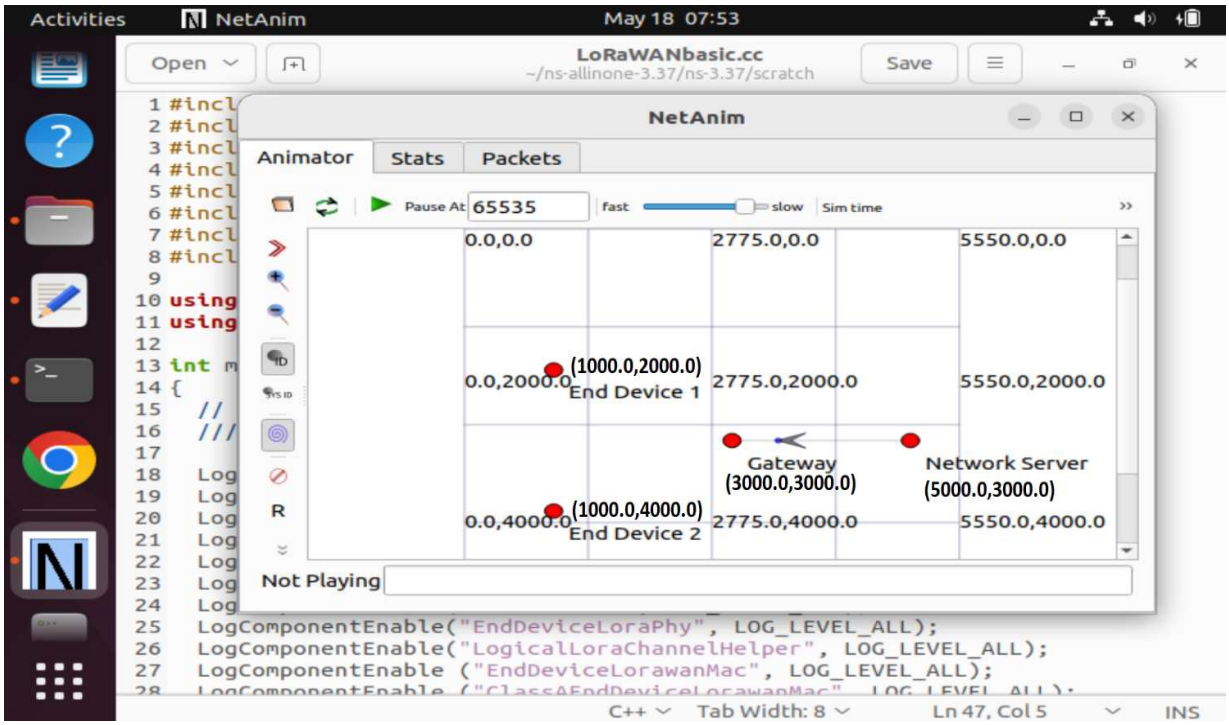

Figure 4.3: Simulation result of LoRaWAN network for the data transmission on downlink from Network Server to End Device

Figure 4.3 shows the simulation of LoRaWAN network for the data transmission on downlink from Network Server to End Device . Here, position of end devices are identified as (1000.0,2000.0) and (1000.0,4000.0) respectively in X-Y coordinates. Postion of Gateway and Network Server are identified as (3000.0,3000.0) and (5000.0,3000,0) respectively in X-Y coordinates.

CHAPTER 5

SELECTION OF EFFICIENT AND SECURITY ALGORITHM FOR LORAWAN BY PERFORMANCE MEASUREMENT

This chapter explains about selection of efficient and secure lightweight algorithms for confidentiality and integrity services in LoRaWAN by performance measurement using CryptoPP library .

5.1 MOTIVATION FOR SELECTION OF LIGHT WEIGHT SECURITY MECHANISM IN LORAWAN

- LoRaWAN is a low power communication scheme used in IoT which needs to be fast and efficient in its low power environment.
- LoRaWAN is used in resource constrained devices operating at low power, so cryptographic algorithms can't be too complex and occupy huge space in the device as it would in turn reduce efficiency of communication.
- As complexity of cryptographic algorithms depend on the number of operations being used, cryptographic techniques using lesser number of operations are preferred.
- Even though AES methods are strong in protecting the message , these algorithms are complex and consume high power which are not ideal for a resource constrained technology like LoRaWAN.
- Hence in this project, some existing lightweight algorithms for message confidentiality and integrity and performance results are compared to find a balance between space and speed using CryptoPP library .

5.2 APPROACH TO MEASURE SPEED OF ALGORITHM

Measurement of speed of various security algorithm for LoRaWAN is performed using CryptoPP library in Linux OS. Machine for testing algorithms has AMD Ryzen 5 5625U processor with a CPU frequency of 2.3 GHz. Simulations are run for 3 second duration.

Metrics that are taken into consideration are Bytes processed, Speed, Latency and Space Occupied.

1. Bytes processed(B)= 2048 X number of buffer blocks **->Eq1**
2. Speed(S)= bytes / (elapsedTimeInSeconds X1024 X1024) **->Eq2**
3. Latency(L) = (elapsedTimeInSeconds X cpuFreq) / bytes processed **-> Eq3**
4. Software Efficiency(SE)= Speed/Space occupied **-> Eq4**

Figure 5.1: Formulas used for calculation of performance metrics of lightweight algorithms

```
Performance Measurement
{
        Include cryptopp library files(cryptlib,secblock,hrtimer,osrng,cipher)
        Set  cpu frequency and runtime limit
        Create function Randomno()
        Create Byte Key[ksize] and IV[ivsize]

        //Intialisation
        Assign Key=randomno(Key,ksize)
        Assign IV=randomno(IV,ivsize)
        Set buffersize = 2048
        Assign Byte buffer[buffersize]= randomno(buffer,buffersize)
        Set elapsedtimeforiteration=0
        Set number of blocks=1

        // Elapsed time calculation for cipher for performance metric measurement
        Create timer() function
        Call start timer() function
        While( elapsedtimeforiteration < runtime limit)
        {
                Set number of blocks= number of blocksX 2
                For( i = 0; ;i< to number of blocks:i++){
                        Call Cipher block encryption()
```

```
            }
            Assign elapsedtimeforiteration= timer.elapsedtime()
        }
}
```

Figure 5.2: Procedure for performance measurement of algorithms using CryptoPP library

The following values namely key, initial vector IV of fixed size are initialised to start counting of logical operations. Now create a buffer with size of 2048 bytes. Next, initialise the number of input block to 1 and every block of input is encrypted and cipher text is obtained. Now measure the elapsed time for execution of one block encryption by one time iterated cipher . If the measured execution time is within the time limit increase the number of input blocks to the algorithm as size of multiples of two. Now, the execution time for continuous two block of inputs are calculated and compared with maximum execution time limit. If suppose the elapsed time spend for running the updated input size is greater than the execution time limit then the algorithm comes out of the loop and stop. The following formulas in Figure 5.1 are used to estimate the speed of algorithm by this approach .

5.3 LIST OF ALGORITHM TESTED WITH EXECUTION TIME

The following algorithms of message confidentiality and integrity are tested in Linux based CryptoPP library. Algorithms considered for message confidentiality and integrity are mentioned below:-

Confidentiality Algorithms	**Integrity Algorithms**
1. AES CTR (Default)	1. AES CMAC (Default)
2. AES ECB (Default)	2. Blake2s

3. Chacha8

4. Salsa8

5. Rabbit

6. Simon

7. Speck

3. SipHash

4. Poly1305

5. SHA-256

6. Tiger

5.3.1 PSEUDOCODE FOR SECURITY ALGORITHMS

5.3.1.1 CONFIDENTIALITY ALGORITHMS

Algorithm 1: ChaCha8 Algorithm

Step 1:- Set plaintext

Set no of plaintext blocks= length(plaintext)/512

Step 2:- Set constant = "expand 32-byte k"

Set 256 bit key and 64 bit IV

Set counter to 0

Step 3:- Set initial state= constant|key|counter|iv of size 512 bits as 16 blocks of size 32 bits

Set working state= initial state

Step 4:- Define quarter round Qround(a,b,c,d) taking 4 blocks a,b,c,d as

1. a += b; d ⊕= a; d <<<= 16;
2. c += d; b ⊕= c; b <<<= 12;
3. a += b; d ⊕= a; d <<<= 8;
4. c += d; b ⊕= c; b <<<= 7;

Step 5 :- Call Inner block (working state) which implements Qround(a,b,c,d) operations for following combinations of blocks 4 times

- Qround(working state, 0, 4, 8,12)
- Qround(working state, 1, 5, 9,13)
- Qround (working state, 2, 6,10,14)
- Qround(working state, 3, 7,11,15)
- Qround(working state, 0, 5,10,15)

- Qround(working state, 1, 6,11,12)
- Qround(working state, 2, 7, 8,13)
- Qround(working state, 3, 4, 9,14)

Step 6 :- Set initial state= initial state + working state which is the keystream

Step 7 :- Steps 1 to 5 are repeated by incrementing the counter for iteration of i=1 to no of plaintext blocks-1 till counter reaches number of plaintext blocks and corresponding key stream values are stored

Step 8 :- Set encrypt= keystream $\oplus$ plaintext for corresponding plaintext blocks

Figure 5.3: Pseudocode for Chacha8 algorithm

As per algorithm 1, the size of input key is 256 bit, the size of counter is 64 bit , the size of initial vector is 64 bit and constant with size 128 bit is considered. The value “expand 32-byte k” is by default assigned to the 128 bit constant . Key Stream is generated with 8 number of rounds in this algorithm and it is xor ed with plain text block of size 512 bit in order to encrypt the plain text. . For creation of key stream , a 512 bit initial state in figure 5.4 is created with constant, key, counter and IV respectively as 16 blocks with each block consisting of 32 bits. State is copied to working state and the working state undergoes 8 combination of Quarter rounds 4 times . Each quarter round represented in figure 5.4 has addition, rotation and xor ing of states. Working state obtained is added with state to create keystream.

Algorithm 2: Salsa8 Algorithm

Step 1:- Set plaintext

Set no of plaintext blocks= length(plaintext)/512

Step 2:- Set constant = “expand 32-byte k”

Set 256 bit key and 64 bit IV

Set counter to 0

Step 3:- Set initial state= constant(32 bit) |key(128 bit) |constant(32 bit)|iv|counter | constant(32 bit)|key(128 bit) |constant(32 bit) of size 512 bits as 16 blocks of size 32 bits

Set working state= initial state

Step 4:- Define quarter round Qround(a,b,c,d) taking 4 blocks a,b,c,d as

1. b ⊕= (a + d)<<< 7),
2. c ⊕= (b + a)<<< 9),
3. d ⊕= (c + b)<<<13),
4. a v= (d + c)<<<18))

Step 5 :- Call Inner block (working state) which implements Qround(a,b,c,d) operations for following combinations of blocks 4 times

- Qround(working state, 0, 4, 8,12)
- Qround(working state, 5, 9, 13,1)
- Qround(working state, 10, 14,2,6)
- Qround(working state, 15, 3,7,11)
- Qround(working state, 0, 1,2,3)
- Qround(working state, 5, 6,7,4)
- Qround(working state, 10, 11, 8,9)
- Qround(working state, 15, 12, 13,14)

Step 6 :- Set initial state= initial state + working state which is the keystream

Step 7 :- Steps 1 to 5 are repeated by incrementing the counter for iteration of i=1 to no of plaintext blocks-1 till counter reaches number of plaintext blocks and corresponding key stream values are stored

Step 8 :- Set encrypt= keystream ⊕ plaintext for corresponding plaintext blocks

Figure 5.4: Pseudocode for Salsa8 algorithm

As per algorithm 2, the size of input key is 256 bit, the size of counter is 64 bit , the size of initial vector is 64 bit and constant with size 128 bit is considered. The value “expand 32-byte k” is by default assigned to the 128 bit constant . Key Stream is generated with 8 number of rounds in this algorithm and it is xor ed with plain text block of size 512 bit in order to encrypt the plain text. For creation of key stream , a 512 bit initial state is created with constant, key, counter and IV respectively as 16 blocks with each block consisting of 32 bits.

State is copied to working state and the working state undergoes 8 combination of Quarter rounds 4 times . Each quarter round has addition, rotation and xor ing of states Working state obtained is added with state to create keystream.

Algorithm 3: Rabbit Algorithm

```
Step1:- Set 128 byte key and plaintext
        Set no of plaintext blocks= length(plaintext)/128
        Create  state X0..7  and counter C0..7
        Set K0 = K[15..10], K1 = K[31..316], ... K7= K[127..112]
        For ( j=0,j< 7,j++)
        {
                If ( j is even){
                        Set Xj = K(j+1 mod 8) || Kj
                        Set Cj = K(j+4 mod 8) || K(j+5 mod 8)
                }
                else{
                        Set Xj = K(j+5 mod 8) || K(j+4 mod 8)
                        Set Cj = Kj || K(j+1 mod 8)
                }
        }
Step 2:-Set 64 bit IV
        Set C0 = C0 ⊕IV[31...30]
         Set C1 = C1 ⊕ (IV[48...63] || IV[16...31])
        Set C2 = C2 ⊕ IV[63..32]
        Set   C3 = C3 ⊕(IV[47..32] || IV[15..0])
        Set   C4 = C4 ⊕IV[31..0]
        Set  C5 = C5 ⊕ (IV[63..48] || IV[31..16])
        Set   C6 = C6 ⊕ IV[63..32]
        Set    C7 = C7 ⊕ (IV[47..32] || IV[15..0])
Step 3:-  Set A0 = 0x4D34D34D,  A1 = 0xD34D34D3,  A2 = 0x34D34D34
```

A3 = 0x4D34D34D , A4 = 0xD34D34D3 , A5 = 0x34D34D34 , A6 = 0x4D34D34D , A7 = 0xD34D34D3

For (j=0 ,j<7,j++){

Set b =0

Set temp = Cj + Aj + b

Set b = temp div 2^{32}

Set Cj = temp mod 2^{32}

}

Step 4:- Set g(u,v) = $(u+v \bmod 2^{32})^2$ ^ $((u+v \bmod 2^{32})^2)>>32$)

For (i=0,i< 7,i++){

Set Gi = g(Xi,Ci)

}

Set X0 = G0 + (G7 <<< 16) + (G6 <<< 16) mod 2^{32}

Set X1 = G1 + (G0 <<< 8) + G7 mod 2^{32}

Set X2 = G2 + (G1 <<< 16) + (G0 <<< 16) mod 2^{32}

Set X3 = G3 + (G2 <<< 8) + G1 mod 232

Set X4 = G4 + (G3 <<< 16) + (G2 <<< 16) mod 2^{32}

Set X5 = G5 + (G4 <<< 8) + G3 mod 232

Set X6 = G6 + (G5 <<< 16) + (G4 <<< 16) mod 2^{32}

Set X7 = G7 + (G6 <<< 8) + G5 mod 2^{32}

Step 5:- Create Keystream S[0..127]

Set S[15..0] = X0[15..0] ⊕ X5[31..16]

Set S[31..16] = X0[31..16] ⊕ X3[15..0]

Set S[47..32] = X2[15..0] ⊕ X7[31..16]

Set S[63..48] = X2[31..16] ⊕ X5[15..0]

Set S[79..64] = X4[15..0] ⊕ X1[31..16]

Set S[95..80] = X4[31..16] ⊕ X7[15..0]

Set S[111..96] = X6[15..0] ⊕ X3[31..16]

Set S[127..112] = X6[31..16] ⊕ X1[15..0]

Step 6 :- For(i=1 ,i < number of plaintext blocks-1,i++){

Set encrypteddata[i]= plaintext[i] ⊕ S

}

Figure 5.5 : Pseudocode for Rabbit algorithm

As per algorithm 3, size of input key is 128 bit and size of counter is 64 bit . 8 state X0..X7 and 8 counters C0..C7 are initialised with key and IV. Counter system is updated with constants A0..A7 and carry bit b .Next state function is called to change states with help of g function . Updated states generate keystream S which is xor ed with plaintext to obtain cipher text.

Algorithm 4 :Simon Algorithm

```
Step 1:- Set plaintext and 64 bit key
        Set n= length(msg)/ 32
        Set k[0]=key[0..15],k[1]=key[16..31],k[2]=[32..47],k[3]=key[48..63]
        Set block pt[0] = left part of plaintext blocks ,pt[1]= right part of plaintext block
Step 2:- Set 64 bit random value as z0
        Define key expansion ke(k)
        For ( i=4 ;i<31;i++){
                Set a= k[i-1]>>3
                Set a=a ⊕ k[i-3]
                Set a=a ⊕ (a>>1)
                Set  round key k[i]= k[i-4] ⊕ a ⊕ z0[(i-4) mod 62] ⊕ 3
        }
Step 3:- Call ke(k)
        For ( i=0 ;i< 31;i++){
                Set x=pt[0]
                Set pt[0]=(((pt[0]<<1&pt[0]<<8)⊕(pt[0]<<2) )⊕ pt[1]) ⊕ k[i]
                Set pt[1]= x
         }
Step 4:- After 32 iterations,
        Set CT[0]= pt[1]
```

```
        Set CT[1]=pt[0]
Step 5:- Set Cipher text block by combing CT[0] and CT[1]
Step 6 :- For ( i=1;i< n;i++){
        Each plaintext block is encrypted to corresponding cipher text block
        }
```

Figure 5.6: Pseudocode for Simon algorithm

As per algorithm 4, size of input key is 64 bit. Input key is split and changed using shift operations and 64 bit random value z0 to generate keys for 32 rounds. Plaintext block is split into 2 parts and is shifted and xor ed with key for 32 rounds to produce updated plaintext blocks and plaintext blocks are swapped to obtain cipher text.

Algorithm 5 : Speck Algorithm

```
Step 1:- Set plaintext and 64 bit key
        Split  Plaintext  into n blocks of size 32
        Set block PT[0] = left part of plaintext block ,PT[1]= right part of plaintext block
        Set  block K[0] =left part of key, K[1]= right part of key
Step2:- Define Round (x,y,k) taking taking 3 blocks x, y and k as
        1. x=x>>8 | x<< 56
        2. x+=y
        3. x= x ⊕ k
        4. y=y<<3 | x>>61
        5. y= y ⊕ x
Step 3 :- Set K0=K[1]
Step 4 :- For ( i =1;i< 31;i++){
                Call Round(K[0],K[1],i)
                Set round keys Ki =K[1] which is done after each step of iteration
         }
Step 5:- For ( i =0;i< 31;i++){
```

```
                Call Round(PT[0],PT[1],Ki)
        }
Step 6:- After 32 iterations
        Set CT[0]= PT[0]
        Set CT[1]=PT[1]
Step 7:- Set Cipher text block by combing CT[0] and CT[1]
Step 8 :- For ( i=1;i< n;i++){
                Each plaintext block is encrypted to corresponding cipher text block
        }
```

Figure 5.7 : Pseudocode for Speck algorithm

As per algorithm 5, size of input key is 64 bit. Input key and plaintext block are split into 2 parts. Key for 32 rounds is generated by passing key blocks into 4 step round operation. Plaintext blocks are then passed with key into round operation to obtain cipher text..

5.3.1.2 INTEGRITY ALGORITHMS

Algorithm 1: Blake 2s Algorithm

```
Step1 :- Set 256 bit key and plaintext
        Set key size=length(key)
        Set plaintext size= length(plaintext)
        Set hash size =256
Step 2:- For ( i=0 ;i< 7;i++){
                Set  IV[i] = floor(2^32 X frac(sqrt(prime(i+1))))
        }
        Set combtext= key|| plaintext
        Set  number of block= ceil(key size/512)+ceil(plaintext size/512)
Step 3 :- Create initial state h[0..7]
        Set h[0...7] = IV[0…7]
```

Set h[0]= h[0] = h[0] ⊕ 0x01010000 ⊕ (key size << 8) ⊕ hash size to indicate key and output hash size

Step 4 :- Define Mix Function Mix(v[0..15], a, b, c, d, x, y) using 6 blocks a,b,c,d,x,y and local working state v[0..15] as

- v[a] = (v[a] + v[b] + x) mod 232
- v[d] = (v[d] ⊕ v[a]) >>> 16
- v[c] = (v[c] + v[d]) mod 2^{32}
- v[b] = (v[b] ⊕ v[c]) >>> 12
- v[a] = (v[a] + v[b] + y) mod 2^{32}
- v[d] = (v[d] ⊕ v[a]) >>> 8
- v[c] = (v[c] + v[d]) mod 2^{32}
- v[b] = (v[b] ⊕ v[c]) >>> 7

Step 5 :- Define Compression Function Comp(h[0..7], m[0..15], t, f) using initial state h, message block m, counter t and flag to indicate last block f

Create local working state v[0..15]

Set 16x16 S box matrix SIGMA

Set v[0..7]= h[0…7]

Set v[8…15]=IV[0..7]

Set v[12] = v[12] ⊕ (t mod 2^{32})

Set v[13] = v[13] ⊕ (t >> 32)

If (f = true) {

Set v[14] = v[14] ⊕ 0xFF..FF

}

For (i=1 ;i< 9;i++){

Set s[0..15] = SIGMA [i mod 10][0..15]

Define 8 mix rounds

- v = Mix(v, 0, 4, 8, 12, m[s[0]], m[s[1]])
- v =Mix(v, 1, 5, 9, 13, m[s[2]], m[s[3]])
- v = Mix(v, 2, 6, 10, 14, m[s[4]], m[s[5]])
- v = Mix(v, 3, 7, 11, 15, m[s[6]], m[s[7]])
- v = Mix(v, 0, 5, 10, 15, m[s[8]], m[s[9]])
- v =Mix(v, 1, 6, 11, 12, m[s[10]], m[s[11]])

```
        ▪ v = Mix( v, 2, 7,  8, 13, m[s[12]], m[s[13]] )
        ▪ v = Mix( v, 3, 4,  9, 14, m[s[14]], m[s[15]] )
    }
    For (i = 0 ;i< 7;i++){
        Set h[i] = h[i] ⊕ v[i] ⊕ v[i + 8]
    }
Step 6 :-  For ( i=0 ; i<number of block-2;i++){
        Call Comp( h[0..7], combtext[i], i, False)
     }
Step 7 :- For final block
    If (key size=0){
        h = Comp( h, combtext[number of block - 1], l, TRUE )
    }
    Else{
        h = Comp( h, combtext[number of block - 1], l + 64, TRUE )
    }
Step 8 :- Set outputhash=h[0…hashsize]
```

Figure 5.8: Pseudocode for Blake2s algorithm

In algorithm 1, size of key is 256 bit key and size of hash is 256 bit . Initialization vector is of size 8 byte and is calculated by using i th prime. IV is assigned to state h and passed through compression function where mixing of local working state v[0..15] and message block occurs and it is xor ed with h to obtain final hash value

Algorithm 2: SipHash Algorithm

```
Step 1: Set 128 bit Key and plaintext
    Set k1=key[0..63]
    Set k2=key[64..127]
Step 2: Initialise 64 bit internal states
```

- v0=k[0] ⊕ 736f6d6570736575
- v1=k[1] ⊕ 646f72616e646f6d
- v2=k[0] ⊕ 6c7967656e657261
- v3=k[1] ⊕ 7465646279746573

Step 3: Define SipRound (v0,v1,v2,v3) where v0,v1,v2,v3 are internal states

- v0+=v1
- v1<<=13
- v1 ⊕ =v0
- v0<<32
- v2+=v1
- v1<<=17
- v1 ⊕ = v2
- v2<<=32
- v2+=v3
- v3<<=16
- v3 ⊕ =v2
- v0+=v3
- v3<<=21
- v3 ⊕ =v0

Step 4 :- Set plaintext length=length(plaintext)

For (i =1 ; i<(plaintext length+1)/64;i++){

Set v3 ⊕ =plaintext[i]

}

Set no of compress rounds

Call siproud(v0,v1,v2,v3) no of compress round times

Set v0 ⊕ =plaintext[i]

Step 5:- Set no of diffusion rounds

Set v2 ⊕ =0xFF

Call siproud(v0,v1,v2,v3) no of diffused round times

Step 6:- Set outputhashvalue= v0 ⊕ v1 ⊕ v2 ⊕ v3

Figure 5.9: Pseudocode for SipHash algorithm

In algorithm 2, size of key is 128 bits and it is split into 2 parts. Size of hash is 64 bit.4 internal states are created using the key and values of internal states are updated either by xor ing an internal state with 64 bit plaintext block or by using compression and diffusion sip rounds. Finally, internal states are xor ed to obtain final hash value

Algorithm 3 : Poly1305 Algorithm

```
Step 1:- Set plaintext and right side of key r
        Set r &= 0x0ffffffc0ffffffc0ffffffc0fffffff
        Set prime p=(1<<130)-5
        Set accumulator a=0
Step 2:- Set left side of key k and nonce
        Call aes128 algorithm to encrypt k with nonce
        Set s= aes128(k,nonce)
Step 3:- Set no of plaintext block= length(plaintext)/128
        For ( i=0 ;i< no of plaintext block;i++){
                Set n = plaintext[((i-1)X16)..(iX16)] | [0x01]
                Set a += n
                Set a = (r X a) % p
         }
Step 4:- Set  a += s
        Set outputhashvalue=a
```

Figure 5.10: Pseudocode for Poly1305 algorithm

In algorithm 3, two keys r and k of size 128 bit are taken and nonce of size 128 bit is taken. Size of hash is 128 bit. r is modified to meet certain conditions .AES algorithm is used to encrypt nonce with k to produce s and then (r,s) are taken as key pairs for algorithm. With (r,s) key pair and a constant prime p, output hash value is obtained

Algorithm 4: SHA-256 Algorithm

Step 1:- Set x,y,z as states

Define Ch(x,y,z)=(x^y)⊕ (~x^z)

Define Maj(x,y,z)=(x^y) ⊕ (x^z) ⊕ (y^z)

Define ROTRi(x)= (x>>n) | (x<<w-n)

Define Sigma0(x)=ROTR7(x) ⊕ ROTR18(x) ⊕ x>>3

Define Sigma1(x)= ROTR17(x) ⊕ ROTR19(x) ⊕ x>>10

Define Epsilon0(x)= ROTR2(x) ⊕ ROTR13(x) ⊕ ROTR22(x)

Define Epsilon1(x)= ROTR6(x) ⊕ ROTR11(x) ⊕ ROTR25(x)

Step 2:- Set plaintext

Set 64 constants CONST0,…CONST63

Set ksize =448-(length(plaintext)+1) mod 512

Set plaintextcomb= plaintext||ksize X (0)

Set n=length(plaintextcomb)/512

Set plaintextcomb1= plaintextcomb[0..511],….plaintextcombN=plaintextcomb [n X 5 12 ….(n X 512)+511]

Set initial state h0[0..7]={6a09e667,bb67ae85,3c6ef372,a54ff53a,510e527f,

9b05688c,1f83d9ab,5beocd19}

Step 3:- For (i=0 ;i<15;i++){

Set Mi= plaintextcomb1[iX32 ...iX32+31]

}

For(i=0;i<15;i++){

Set Wi=Mi

}

For(i=16;i<63;i++){

Set Wi=Sigma1(Wi-2)+Wi-7+Sigma0(Wi-15)+Wi-16

}

Step 4:- Set a=h0[0],b=h0[1],c=h0[2],d=h0[3],e=h0[4],f=h0[5],g=h0[6],h=h0[7]

For (i=0 ;i< 63;i++){

Set T1=h+Epsilon1(e)+Ch(e,f,g)+CONSTi+Wi

Set T2=Epsilon0(a)+Maj(a,b,c)

```
                Set h=g
                Set g=f
                Set f=e
                Set e=d+T1
                Set d=c
                Set c=b
                Set b=a
                Set a=T1+T2
        }
Step 5:-Set h1[0]=a+h0[0]
        Set h1[0]=a+h0[0]
        Set h1[1]=b+h0[1]
        Set h1[2]=c+h0[2]
        Set h1[3]=d+h0[3]
        Set h1[4]=e+h0[4]
        Set h1[5]=f+h0[5]
        Set h1[6]=g+h0[6]
        Set h1[7]=h+h0[7]
Step 6 :- Steps 3 to 5 are repeated for N plaintextblocks
Step 7 :- Set outputhash by combining hn[0] to hn[7]
```

Figure 5.11: Pseudocode for SHA-256 algorithm

In algorithm 4, size of hash is 256 bit. Value for initial state h0 is assigned. Plaintext is padded and taken as a 512 bit block. Plaintext blocks are split into 16 parts and assigned to 64 words Wi of size 32 bits. 64 rounds of operation take place and Wi along with constants CONSTi are used to change initial state h0 to h1.It is repeated for N blocks of plaintext and output hash is obtained.

Algorithm 5 : Tiger Algorithm

Step 1:- Set plaintext and split it into 512 bit plaintext blocks

Split 512 bit plaintext blocks into eight 32 byte msgblocks (msgblocks[0]… msgblocks[7])

Create initial state h of size 192 bit

Split h into three 64 bit blocks(h[0],h[1],h[2])

Set h[0]=0X0123456789ABCDEF

Set h[1]=0XFEDCBA9876543210

Set h[2]=0XF096A5B4C3D2E187

Step 2:- Set a=h[0],b=h[1],c=h[2]

Set x0..7=msgblocks[0..7]

Set aa=a, bb=b, cc=c

Step 3:- Set four S box matrices t1,t2,t3,t4 having 256 values of size 64 bit

Define round(a,b,c,x,mul) where a,b,c,x are states of hash and mul is the multiple taken

- Set c⊕=x
- Set a-=t1[c&0xFF] ⊕t2[c>>16&0xFF] ⊕t3[c>>32&0xFF] ⊕t4[c>>48&0xFF]
- Set b+= t4[c>>8&0xFF] ⊕t3[c>>24&0xFF] ⊕t2[c>>40&0xFF]^ t1[c>>56&0xFF]
- Set b X=mul

Step 4 :- Define pass(a,b,c,mul) where a,b,c are states of hash and mul is the multiple taken

- round(a,b,c,x0,mul)
- round(b,c,a,x1,mul)
- round(c,a,b,x2,mul)
- round(a,b,c,x3,mul)
- round(b,c,a,x4,mul)
- round(c,a,b,x5,mul)
- round(a,b,c,x6,mul)
- round(b,c,a,x7,mul)

Step 5:- Define Keyschedule() as

- x0-=x7⊕0XA5A5A5A5A5A5A5A5
- x1⊕=x0
- x2+=x1

- x3-=x2⊕ (~x1<<19)
- x4⊕=x3
- x5+=x4
- x6-=x5⊕ (~x4<<23)
- x7⊕=x6
- x0+=x7
- x1-=x0⊕ (~x7<<19)
- x2⊕=x1
- x3+=x2
- x4-=x3⊕ (~x2<<23)
- x5⊕=x4
- x6+=x5
- x7-=x6⊕0x0123456789ABCDEF

Step 6:- Call pass(a,b,c,5)

Call Keyschedule()

Call pass(c,a,b,7)

Call Keyschedule()

Call pass(b,c,a,9)

Set a⊕=aa,b-=bb,c+=cc

Set h[0]=a, h[1]=b, h[2]=c

Set outputhash by combining h[0],h[1] and h[2]

Figure 5.12: Pseudocode for Tiger algorithm

In algorithm 5, size of hash is 192 bits and value for initial state h of size 192 bits has been assigned. Initial state h is split into 3 blocks and values are saved .24 Rounds of operation take place where initial state h is modified using round key of size 64 bits and s box matrices . Round keys are obtained by performing operations on plaintext . Final state is split into 3 blocks and it is either xor ed or added with initial state blocks to obtain output hash value.

5.4 COMPARISON OF VARIOUS PERFORMANCE METRICS FOR CONFIDENTIALITY AND INTEGRITY ALGORITHMS

Time=3s, CPU Frequency= 2.3GHz=$2.3X10^9$ GHz

The efficiency and capacity of machine used for execution of security algorithms is as follows -

S.No	Confidentiality Algorithm	Elapsed Time(s)	Bytes Processed	Speed (Mibs)	Latency (Cycles / byte)	Space occupied (kb)	Software efficiency (Mibs/ kb)
1	AES-CTR(Default)	5.76	8589934592	1422.22	1.54227	2.7	526.748
2	AES-ECB(Default)	5.35	8589934592	1531.21	1.43249	2.2	696.004
3	Simon	4.24	2147483648	483.019	4.54113	2.2	219.55
4	Speck	5.96	8589934592	1374.5	1.59582	2.4	572.708
5	Rabbit	4.44	4294967296	922.523 (IV)	2.37767	1.5	615.01533
		4.41	4294967296	928.798 (WO IV)	2.3616	1.3	714.46
6	Chacha	3.65	171798691844	4488.77	0.4886	1.6	2805.48125
7	Salsa	5.82	171798691844	2815.12	0.7791	1.6	1759.45

Table 5.1 : Comparison of various performance metrics for confidentiality algorithms. in Linux OS using CryptoPP library

The confidentiality algorithms for resource constrained LoRaWAN expects lower latency and high speed and also it must occupy lesser space. Cipher is said to be efficient in software if it maintains balance between speed and minimum space . In order to improve the confidentiality of LoRaWAN the existing confidentiality algorithms (AES ECB, AES CTR) are needed to be compared with alternative confidentiality algorithms. Metrics considered for comparison are speed latency , space occupied and software efficiency .The alternative confidentiality algorithms suggested for LoRaWAN are Simon , Speck, Rabbit, Chacha and Salsa. Values are measured and tabulated for various performance metrics in Table 5.1. From the table, it is observed that Chacha8 confidentiality algorithm has highest speed and lowest latency. ChaCha cipher has also been identified more software efficient confidentiality algorithm even though Rabbit algorithm occupies lesser space.

S.No	**Integrity Algorithm**	**Elapsed Time(s)**	**Bytes Processed**	**Speed (Mibs)**	**Latency (Cycle/ byte)**	**Space Occupied (kb)**	**Software Efficiency (Mibs/kb)**
1	AES-CMAC (Default)	5.2	4294967296	787.692	2.78465	1.4	562.637
2	SipHash	5.57	2684355456	45.9605	47.7247	1.3	35.354
3	SHA 256	5.64	2147483648	363.121	6.04056	1.4	259.37
4	Tiger	3.01	2147483648	680.399	3.22377	0.69	986.085
5	Blake 2s	5.8	4294967296	706.207	3.10596	0.69	1023.488
6	Poly 1305	3.09	4294967296	1325.57	1.65473	1.6	828.481

Table 5.2 :- Comparison of various performance matrix for integrity algorithms in Linux OS CryptoPP library

The integrity algorithms for resource constrained LoRaWAN expects lower latency and high speed and also it must occupy lesser space. Hash is said to be efficient in software if it maintains balance between speed and minimum space . In order to improve the integrity of LoRaWAN the existing integrity algorithm (AES CMAC) is needed to be compared with alternative integrity algorithms. Metrics considered for comparison are speed latency , space occupied and software efficiency .The alternative integrity algorithms suggested for LoRaWAN are Blake2s, SipHash, Poly1305,Tiger and SHA-256. Values are measured and tabulated for various performance metrics in Table 5.2. From the table, it is observed that Poly1305 integrity algorithm has highest speed and lowest latency. But Blake2s algorithm has been identified as more software efficient integrity algorithm as it occupies comparatively lesser space.

5.5 EXISTING SECURITY ATTACKS TO CONFIDENTIALITY AND INTERGRITY ALGORITHM OF LORAWAN

S.No	Confidentiality Algorithm	Cryptanalysis
1	AES-CTR(Default)	Side channel attack(Persistent Fault Analysis) ,Related key attack, Man in the middle attack
2	AES-ECB(Default)	Side channel attack(Persistent Fault Analysis) ,Related key attack, Man in the middle attack
3	Simon	Differential cryptanalysis ,Cube cryptanalysis , related key attack, Side Channel attack(Differential Fault analysis)
4	Speck	Differential cryptanalysis ,related key attack, Side Channel attack(Differential Fault analysis)
5	Rabbit	Distinguishing attack ,side channel attack(Correlated Power analysis)
6	Chacha	Differential cryptanalysis (only below 7 rounds),side channel attack(Fault injection attack)
7	Salsa	Differential cryptanalysis (only below 8 rounds),side channel attack(Fault injection attack)

Table 5.3 : Existing Security Attacks on lightweight ciphers considered

From the literature, it is observed the confidentiality algorithm AES is susceptible to attacks namely Man in the Middle Attack (MITM) attack[14] , Related Key attack[15] and Side Channel attack[16] . Computational requirement for MITM attack in AES is 2^{80} no of logical combinations. Computational requirement for related key attack in AES is 2^{38} no of logical combinations. Side channel attack for AES requires minimum size of intercepted cipher text combinations as 1641 in order to obtain key. In Simon algorithm, Differential Cryptanalysis[17] needs 18 out 32 rounds with complexity of 2^{31} .Cube attacks[18] in Simon algorithm have succeeded on 17 out of 32 rounds. Cube attacks on Simon algorithm occur with space complexity of 2^{31}. Related Key attacks[19] in Simon algorithm have succeeded on 10 out of 32 rounds with complexity of 2^{14}. In Speck algorithm, Differential cryptanalysis[17] needs 10 out of 22 rounds with complexity of 2^{29}. Related Key attacks[20] in Speck algorithm have succeeded on 12 out of 22 rounds .Side Channel attacks[21] can be done on both Simon and Speck ciphers. In Rabbit algorithm, Distinguishing attack[22] can be implemented for 2^{25} no of logical combinations .Side channel attacks[23] can be done on Rabbit algorithm .Chacha algorithm[11] is susceptible to Differential Cryptanalysis for 7 round implementation .Salsa algorithm[24] is susceptible to Differential Cryptanalysis for 8 round implementation. Side Channel attack[25] can be done on both Chacha and Salsa algorithm

Chacha8 cipher is more preferrable as it has no attacks other than side channel attacks for 8 round implementation

S.No	**Integrity Algorithm**	**Cryptanalysis**
1	AES-CMAC (Default)	Side channel attack (Persistent Fault Analysis),Related key attack, Man in the middle attack
2	SipHash	Differential cryptanalysis, side channel attack(Correlation Power analysis),rotational XOR cryptanalysis
3	SHA 256	Collision attack, Meet in the middle preimage attack
4	Tiger	Pseudo near Collision attack, Meet in the middle preimage attack

5	Blake 2s	Collision attack
6	Poly 1305	Side channel attack ,Related key attack, Man in the middle attack

Table 5.4 : Existing Security Attacks on lightweight hashes considered

SipHash is susceptible to attacks namely Differential Cryptanalysis[26], side channel attack[27] and Rotational XOR Cryptanalysis[28]. In SHA-256 algorithm, Meet in the Middle Preimage attack[12] can be done with time complexity of 2^{254}. Collision attack in SHA-256 algorithm have succeeded on 21 out of 64 rounds at complexity of 2^{19} [29] .In Tiger algorithm, Meet in the Middle Preimage attack[12] can be done with time complexity of 2^{188}. Tiger algorithm is also susceptible to pseudo near collision attack[30] with complexity of 2^{47}. .Blake2s is susceptible only to collision attack at complexity of 2^{64} [10]. Poly 1305 is susceptible to all attacks of AES as it uses AES for key generation.

Blake2s is more preferrable as it only suffers collision attack.

CHAPTER 6

SIMULATION OF EFFICIENT SECURITY ALGORITHM IN LORAWAN NETWORK

This chapter presents the implementation of efficient security algorithm finalized in Chapter 5 on LoRaWAN Network.

6.1 INTRODUCTION

Initial implementation of LoRaWAN Network in NS3 doesn't the include security features of LoRaWAN. Implementation of security algorithms will be carried out using CryptoPP library which is integrated into NS3 using Cmake FindPackage method. This chapter deals with implementation of efficient security mechanism for LoRaWAN .

6.2 INTERPRETOR HIERARCHY FOR SECURITY IMPLEMENTATION IN LORAWAN

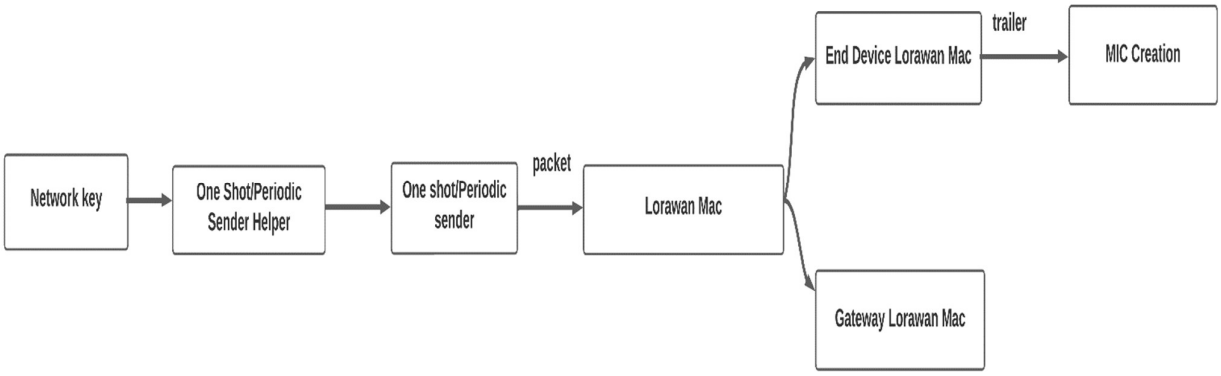

Figure 6.1: C++ Interpreter hierarchy for security in End Device

In order to secure LoRaWAN network, encryption of input is done for confidentiality of data using AppSkey and NwkSkey as variables in NS3. LoRaWAN End device executes encryption algorithm using the assigned AppSkey variable. Using the same AppSkey variable, decryption of received data has been done in Network Server of LoRaWAN network environment. Integrity of messages from LoRaWAN End Devices are ensured using MIC (Message Integrity Code) generation with NwkSkey variable in NS3 script. LoRaWAN Network Server does the decryption process only after verifying the integrity code hence original user message is only allowed to be decrypted at the Network Server. This reduces time requirement for security in LoRaWAN..

The class **'Network Server Helper'** in NS3 uses declared variable AppSkey and IV to trigger decryption process in Network Server. The class **'EndDeviceLorawan Mac'** on End Device uses NwkSkey variable for MIC creation and **'One shot Sender'** or **'Periodic Sender'** class functions to generate packets with cipher text. Now, the End Device generates MIC code of generated messages and it is attached to trailer of the packet. The variable NwkSkey which is used in End Device is also sent to Network Server to verify integrity of received messages on the Scheduler of the Network Server. Now, the received packet verification for integrity result decides to do decryption of received packet

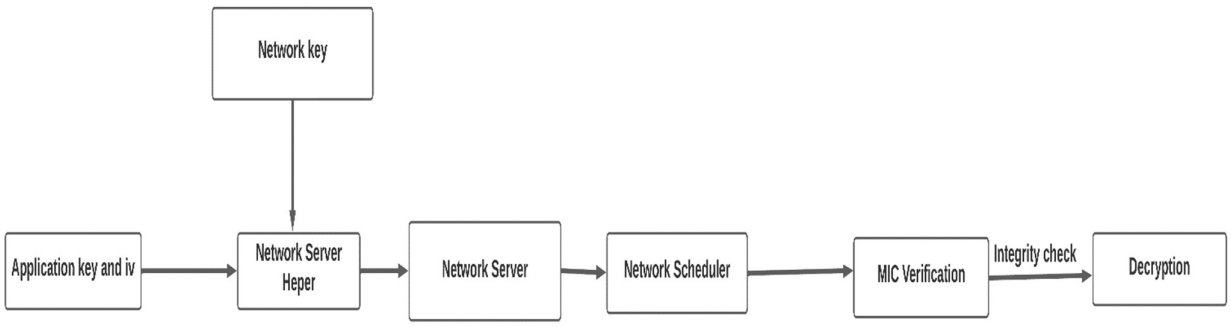

Figure 6.2: C++ Interpreter hierarchy for security in Network Server

6.3 ALGORITHM FOR EFFICIENT AND SECURE LORAWAN

```
Lorawan
{
        //Creation of Cipher
        Set AppSkey,NwkSkey,IV and Msg
        Set Cipher=Chacha8( Msg ,AppSkey,Iv)

        // Creation of EndDevice and Gateway Node
        Set EndDevice=Create(1,Position(x1,y1,z1))
        Set Gateway= Create(1,Position(x2,y2,z2))

        // Channel Creation
        Set  Refdist,Refpathloss and Pathlossexp for LoRa Channel
        Set Pathloss= LogDistancePropogationDelayModel(Refdist,Refpathloss and Pathlossexp)
        Set PropDelay = ConstantSpeedPropagationDelayModel()
        Set Channel= Create(PathLoss, PropDelay)

        // Assign End Device Properties
        Set Nwkid and Nwkaddr for LoRa end device
        Set DeviceAddress = Nwkid | Nwkaddr
        Create phylayer() and Maclayer() function
        Call phylayer.Set(Channel ,Enddevice)
        Call Maclayer.Set (DeviceAddress, MessageType, RegionofOperation,EndDevice)

        //One Shot Sender Creation
        Set txSendtime of the packet
        Create OneShotSender()
        Call OneShotSender.Set(txSendtime)
        Call OneShotSender.Send(NwkSkey)

        // Assign Gateway Properties
        Call Phylayer.Set(Gateway)
        Call Maclayer.Set(Gateway)
        Call Gateway.Install(Forwarder)

        // Set Spreading Factor
        Set Spreadfactor= Set(EndDevice,Gateway,Channel)

        //Creation of Network Server
        Set Nwksvr=Create(1,Position(x3,y3,z3))
        Call Nwksvr.Set(Gateway, EndDevice)
        Call Nwksvr.Send(AppSkey,IV)
```

```
        //Packet Sending in EndDevice and Gateway
        Call OneShotSender.SendPacket(cipher,size(cipher), EndDeviceLorawanMac)
        Set MIC = Blake2s(cipher,NwkSkey)
        Call Packet.Add(FrameHdr,MacHdr,MIC)
        Call EndDeviceLorawanMac.SendPacket(EndDeviceLoraPhy)
        Call EndDeviceLoraPhy.SendPacket(Gateway)
        Call GatewayLoraPhy.SendPacket(GatewayLorawanMac)
        Call Forwarder.SendPacket(Nwksvr)

        //Packet Receiving in Network Server
        Set receivedpacket=Nwksvr.Receive(Gateway)
        Call Nwkscheduler.CheckMIC(Packet)
        Set ReceivedMsg=Packet.Data()
        Set ReceivedMIC= Packet.Trailer()
        Set MICcalc=Blake2s(ReceivedMsg,NwkSkey)
        If (ReceivedMIC= MICcalc)
        {
                Set decryptedmessage= Chacha8(ReceivedMsg,AppSkey,IV)
        }

}
```

Figure 6.3 : Procedure for development of efficient and secure LoRaWAN Network

In order to secure LoRaWAN , security key AppSkey of size 32 byte, NwkSkey of size 32 byte and Initial Vector (IV) of size 8 byte are used in Confidentiality and integrity algorithm. Encryption of input message of N bytes is done by ChaCha confidentiality algorithm with 32 byte key AppSkey and 8 byte Initial Vector(IV) . Now, from the simulation environment, LoRaWAN End Devices are created and its positions have been configured using NS3 class functions. Additionally LoRaWAN Gateways are also created and configured on the simulated environment.

At the LoRaWAN End Device, wireless channel quality has been calculated using NS3 class functions for LoRaWAN channel and values for reference distance, path loss at reference distance and path loss exponent .Path loss estimation is done using wireless channel model called Log distance propagation model with formula

$L=L0+10 \text{ X } n \text{ X } \log_{10}(d/d0)$ dB ->**Eq5**

Where n = path loss exponent

L0= path loss at reference distance

d0= reference distance

d= distance between gateway and lorawan end devices

Wireless channel propagation delay is calculated by using the model called Constant Speed Propagation Delay Model with formula

Tp=(d/c) s-> **Eq6**

Where d= distance between gateway and lorawan end devices

c= speed of light

Now , the LoRaWAN wireless channel finally has been created using path loss estimation and propagation delay estimation based NS3 class functions. Then LoRaWAN device address is calculated by combining network address (Nwkaddr) and network id (Nwkid) class functions. At the LoRaWAN End Device node, physical layer properties of wireless channel are assigned to the node using NS3 class functions and other physical layer properties of End Device namely transmission power, size of interference matrix, data rate of node, spreading factor of node are also assigned to the node . Now at the LoRaWAN End Device node, Mac layer properties are namely device address, region of operation, type of message, specific region of operation , frequency of operation are assigned using NS3 class functions.

At the LoRaWAN End Device, class function **'One Shot Sender'** is used to create form packet with the cipher . Message Integrity code is generated for the cipher in LoRaWAN End Device using NwkSkey. Now packets are encrypted and attached with headers and the generated Message integrity code on trailer of node PDU data. Spreading factor of modulation is decided based on the distance between end device and gateway thereby in order to estimate maximum data transfer possible. Network Server is created and its position is configured. End Devices and Gateways are assigned

to Network Server. Packet is sent from LoRaWAN End Devices to Gateway and Gateway to Network Server. AppSkey and counter are also sent to Network Server through Network Server Helper to enable decryption in network server. NwkSkey is also sent to Network Server through which packet arriving at Network Server is checked for integrity in Network Scheduler. Only when integrity is verified , Decryption of packet is carried out in the Network Server.

CHAPTER 7

SIMULATION RESULTS OF PERFORMANCE MEASUREMENT FOR SECURED LORAWAN NETWORK

This chapter deals with simulation of confidentiality and integrity services to LoRaWAN network and performance measurement of secure LoRaWAN Network

7.1 SIMULATION RESULTS FOR THE CONFIDENTIALITY AND INTEGRITY ALGORITHM

Simulation of confidentiality service to LoRaWAN using Chacha8 algorithm has been completed and simulation of integrity service for messages of LoRaWAN devices has been done using Blake2s algorithm.

Message Encryption:-

Input message “Abhinav” with size of 7 bytes is considered as plaintext to the encryption algorithm . The size and Value of the key shared between End Device and Network Server are 16 bytes and “sgggsffscdcadada” respectively and it is named as AppSkey. Value of the Initial Vector is “00000001” with 8 bytes size. Cipher is generated with size of 7 bytes and in Hexadecimal as “B99D94AC241B9C” using Chacha algorithm at LoRaWAN End Device.

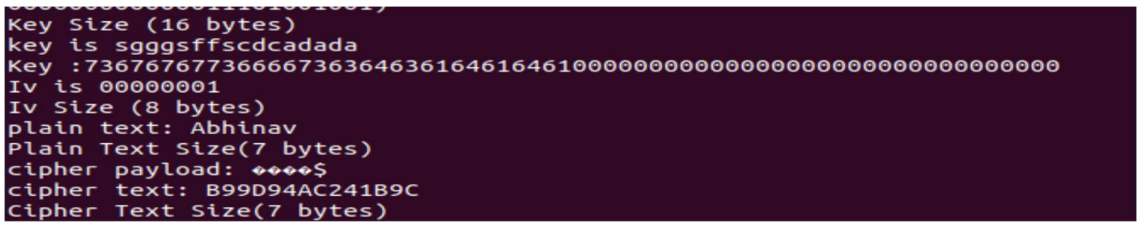

Figure 7.1 : Result of Encryption in LoRaWAN End Device using Chacha8 algorithm

Figure 7.1 shows the generated key in hexadecimal format. It also displays size of key as 16 bytes and size of iv as 8 bytes. Encrypted plaintext called ciphertext is generated and is displayed as “B99D94AC241B9C”in hexadecimal format.

Message Integrity Code creation :-

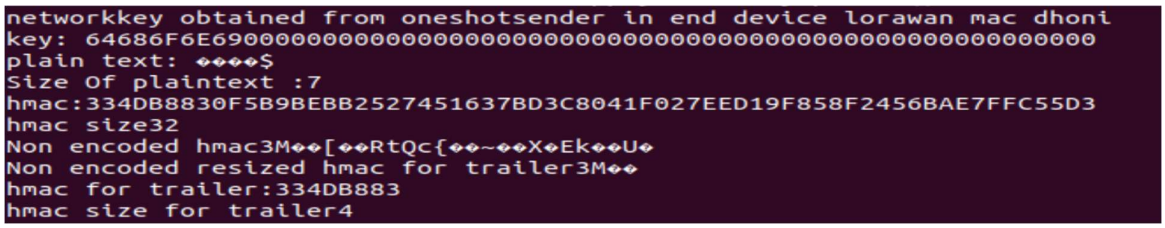

Figure 7.2 : MIC Creation results in End Device using Blake2s algorithm with NwkSkey

Figure 7.2 shows results of message integrity code generation at LoRaWAN end device. The Received ciphertext “B99D94AC241B9C” is used in order to generate message integrity code with secret key NwkSkey whose value is “dhoni”. Integrity code for message is generated as “C0F5B9BEBB2527451637BD3C8041F027 EED19F858F24dh56BAE7FFC55D3” with size of 32 bytes. LoRaWAN requires 4 bytes only therefore final MIC is “334DB883”

Verification of Message Integrity Code at destination:-

Figure 7.3: Message Integrity verification results in Network Scheduler using MIC received from trailer of the packet and MIC calculated with NwkSkey using Blake2s algorithm

The Received cipher text at network server is “B99D94AC241B9C”. Now, value of message integrity code “334DB8830F5B9BEBB2527451637BD3C8041F027EED19F858F2456BAE7FFC55D” is generated using NwkSkey by Blake2s algorithm. But the trailer of Lorawan packet considers first 4 bytes of generated integrity code as “334DB883”. Results shown in the Figure 7.2 are verified in Figure 7.3 for message integrity code. Hence the message integrity of lorawan end device is preserved in the simulated network environment.

Message Decryption:-

Figure 7.4: Decryption results in Network Server using Chacha8 algorithm with AppSkey and IV after message integrity verification

The Received cipher text at network server is “B99D94AC241B9C”. As the packet has not been tampered, Cipher is decrypted in Network Server using AppSkey

and IV by Chacha algorithm. Recovered text after decryption is “Abhinav” with size of 7 bytes. Figure 7.4 shows message decryption at network server. This figure shows key, iv and value of the recovered text

.

7.2 PERFORMANCE MEASUREMENT OF CONFIDENTIALITY AND INTEGRITY ALGORITHMS FOR SIMULATED SECURE LORAWAN NETWORK BY THROUGHPUT CALCULATION

In order to measure performance of secured LoRaWAN network following parameters namely Packet Delivery Rate, Network Traffic and Throughput have been considered. Total simulation time taken is 600 seconds and On air time of packet between End Device and gateway is 0.056576. In the simulation work, each device sends a packet and number of devices is varied in order to find Network Throughput.

1. Packet Delivery Rate(PDR)=Packets received / packets sent ->**Eq 7**
2. Network Traffic(G) = Packets sent X OnairTime/time ->**Eq 8**
3. Throughput(S)=G X PDR ->**Eq 9**

Figure 7.5 : Formulas for measurement of performance of secured LoRaWAN Network

Following parameters are fixed for the simulation work in NS3 simulation Time=600s,Packets per device=1,On air Time=0.056576

S.No	No of Devices	Packets sent	Packets received	Packet Delivery Rate(PDR)	Network Traffic(G)	Throughput (S)
1	200	200	196	0.98	0.01855	0.018473
2	300	300	276	0.92	0.02828	0.026017

3	600	600	506	0.84	0.05657	0.047518
4	1000	1000	822	0.822	0.09429	0.077506
5	2000	2000	1369	0.6845	0.18858	0.129083
6	3000	3000	1721	0.5736	0.28288	0.162259

Table 7.1 : Performance measurement using metrics namely Packet Delivery Rate, received network traffic and Throughput

From table 7.1, it can be noted that as number of devices sending packets increase Packet Delivery Rate (PDR) decreases. But as Network Traffic increases ,Throughput of the network increases

CHAPTER 8

CONCLUSION

LoRaWAN is a type of long range wide area network . Resource constraints of LoRaWAN network provide the need for light weight security algorithms for ensuring security. But challenges and issues faced by LoRaWAN motivates protection of data as an important task of the network. In this project work, performance measurement of various confidentiality and integrity services for LoRaWAN network security has been simulated. List of confidentiality algorithms namely Chacha, Salsa, Rabbit, Simon and Speck and integrity algorithms namely Blake2s, SHA256, SipHash, Tiger and Poly1305 are compared and based on software efficiency and study on existing attacks Chacha and Blake2s algorithm have been considered for confidentiality and integrity. From the simulation work , Input message ("Abhinav") of size 7 bytes is considered as plaintext to the encryption algorithm. Cipher ("B99D94AC241B9C") of size 7 bytes is generated and using Chacha algorithm and a packet is created in LoRaWAN End Device. Message Integrity Code or MIC is created using Blake2s algorithm at LoRaWAN End Device and its value is "334DB883". Message Integrity code is verified in Network Server and decryption of packet is done. In order to measure performance of secured LoRaWAN network, number of devices sending a packet are varied and Network Throughput is found .

REFERENCES

[1] S.Devalal. A.Karthikeyan (2018), "**LoRa technology-an overview**", In Proceed ings of the 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 29–31 March 2018; pp. 284–290.

[2] A.Augustin, J.Yi, T.Clausen, W.M.Townsley (2016), **"A Study of LoRa: Long Range & Low Power Networks for the Internet of Things"** ,Sensors 2016, *16*, 1466. https://doi.org/10.3390/s16091466

[3] J.Silva, D.Flor, V.A. de Junior, N.Bezerra, A.Medeiros (2021), **"A Survey of LoRaWAN Simulation Tools in ns-3"** , Journal of Communication and Information Systems, 36(1), 17-30. https://doi.org/10.14209/jcis.2021.2

[4] E.Aras, G.S.Ramachandran, P.Lawrence, D.Hughes (2017), **"Exploring the security vulnerabilities of LoRa(2017)"** ,In Proceedings of the 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), Exeter, UK, 21–23 June 2017; pp. 1–6.

[5] X.Yang, E.Karampatzakis, C.Doerr, F. Kuipers (2018), **"Security Vulnerabiliti es 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] X.Chen, M.Lech, L.Wang (2021), **"A Complete Key Management Scheme for LoRaWAN v1.1"** ,Sensors 2021, 21, 2962. https://doi.org/10.3390/s21092962

[7] S.A.A.Hakeem, S.M.A.El-Kader, H.Kim (2021), **"A Key Management Protocol Based on the Hash Chain Key Generation for Securing LoRaWAN Networks"** ,Sensors 2021, 21, 5838. https://doi.org/10.3390/s21175838

[8] I.Bhardwaj, A.Kumar, M.Bansal (2017), **"A Review on Lightweight Cryptography Algorithms for Data Security and Authentication in IoTs"** , 4th International Conference on "Signal Processing, Computing and Control"At: Jaypee University of Information Technology, Waknaghat, India, doi: 10.1109/ISPCC.2017 .8269731

[9] V.Thakor, M.A.Razzaque, M.Khandaker (2021), **"Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities".** IEEE Access. 9. 28177-28193. 10.1109/ACCESS.20 21.3052867.

[10] J.Guo, P.Karpman, I.Nikolić, L.Wang, S.Wu (2014), **"Analysis of BLAKE2"** ,In: Benaloh, J. (eds) Topics in Cryptology – CT-RSA 2014. CT-RSA 2014. Lecture Notes in Computer Science, vol 8366. Springer, Cham. https://doi.org/10.1007/978-3-319-04852-9_21

[11] M.Shotaro, I.Ryoma, M.Atsuko (2022), **"PNB-Focused Differential Cryptana lysis of ChaCha Stream Cipher"** ,In Information Security and Privacy: 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28–30, 2022, Proceedings. Springer-Verlag, Berlin, Heidelberg, 46–66. https://doi.org/10.1007/978-3-031-22301-3_3

[12] S.Ling, J.Guo, C.Rechberger, H.Wang (2010), **"Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2".** In: Abe, M. (eds) Advances in Cryptology - ASIACRYPT 2010. ASIACRYPT 2010. Lecture Notes in Computer Science, vol 6477. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17373-8_4

[13] D.Magrin, M.Centenaro, L.Vangelista (2017), **"Performance evaluation of LoRa networks in a smart city scenario"** ,2017 IEEE International Conference on Communications (ICC), Paris, France, 2017, pp. 1-7, doi: 10.1109/ICC.2017.7996384.

[14] A.Bogdanov, D.Khovratovich, C.Rechberger (2011), **"Biclique Cryptanalysis of the Full AES"** ,In: Lee, D.H., Wang, X. (eds) Advances in Cryptology – ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol 7073. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25385-0_19

[15] A.Biryukov, D.Khovratovich (2009), **"Related-Key Cryptanalysis of the Full AES-192 and AES-256"** ,In: Matsui, M. (eds) Advances in Cryptology – ASIACRYPT 2009. ASIACRYPT 2009. Lecture Notes in Computer Science, vol 5912. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10366-7_1

[16] F.Zhang, Y.Zhang, H.Jiang, Huilong & X.Zhu (2020), **"Persistent Fault Attack in Practice"** ,IACR Transactions on Cryptographic Hardware and Embedded Systems. 172-195. 10.46586/tches.v2020.i2.172-195.

[17] F.Abed, E.List, S.Lucks, J.Wenzel (2015), **"Differential Cryptanalysis of Rou nd-Reduced SIMON and SPECK"** ,In: Cid, C., Rechberger, C. (eds) Fast Software Encryption. FSE 2014. Lecture Notes in Computer Science(), vol 8540. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-46706-0_27

[18] R.Rabbaninejad, Z.Ahmadian, M.Salmasizadeh, M.R.Aref (2014), **"Cube and dynamic cube attacks on SIMON32/64"** ,2014 11th International ISC Conference on Information Security and Cryptology, Tehran, Iran, 2014, pp. 98-103, doi: 10.1109/ISCISC.2014.6994030.

[19] J.Lu, Y.Liu, T.Ashur, B.Sun, C.Li (2020), **"Rotational-XOR Cryptanalysis of Simon-Like Block Ciphers"** ,In: Liu, J., Cui, H. (eds) Information Security and Privacy. ACISP 2020. Lecture Notes in Computer Science(), vol 12248. Springer, Cham. https://doi.org/10.1007/978-3-030-55304-3_6

[20] Y.Liu, G.Witte, A.Ranea, T.Ashur (2017), **"Rotational-XOR Cryptanalysis of Reduced-round SPECK"** ,IACR Transactions on Symmetric Cryptology. 24-36. 10.46586/tosc.v2017.i3.24-36.

[21] H.Tupsamudre, S.Bisht, D.Mukhopadhyay (2014), **"Differential Fault Analysis on the Families of SIMON and SPECK Ciphers"** ,2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, Busan, Korea (South), 2014, pp. 40-48, doi: 10.1109/FDTC.2014.14.

[22] N.R.Darmian (2013), **"A Distinguish attack on Rabbit Stream Cipher Based on Multiple Cube Tester"** ,IACR Cryptol. ePrint Arch. 2013: 780.

[23] B.KiSeok, A.MahnKi, L.HoonJae, H.JaeCheol, M.SangJae (2011), **"Power analysis attack and countermeasure on the Rabbit Stream Cipher"** ,In Proceedings of the 7th International Workshop on Software Engineering for Secure Systems (SESS '11). Association for Computing Machinery, New York, NY, USA, 50–56. https://doi.org/10.1145/1988630.1988640

[24] J.P.Aumasson, S.Fischer, S.Khazaei, W.Meier, C.Rechberger (2008), **"New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba"** ,In: Nyberg, K. (eds) Fast Software Encryption. FSE 2008. Lecture Notes in Computer Science, vol 5086. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71039-4_30

[25] K.Fukushima, R.Xu, S.Kiyomoto, N.Homma (2017), **"Fault Injection Attack on Salsa20 and ChaCha and a Lightweight Countermeasure"** ,2017 IEEE Trustcom /BigDataSE/ICESS, Sydney, NSW, Australia, 2017, pp. 1032-1037, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.348.

[26] C.Dobraunig, F.Mendel, M.Schläffer (2014), **"Differential Cryptanalysis of SipHash"** , IACR Cryptol. ePrint Arch. 2014 (2014): 722.

[27] M. Olekšák, V. Miškovský (2022), **"Correlation Power Analysis of SipHash"** ,2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Prague, Czech Republic, 2022, pp. 84-87, doi: 10.1109/DDECS54261.2022.9770139.

[28] W.Xin, Y.Liu, B.Sun, C.Li (2019), **"Improved Cryptanalysis on SipHash"** .In Cryptology and Network Security: 18th International Conference, CANS 2019, Fuzhou, China, October 25–27, 2019, Proceedings. Springer-Verlag, Berlin, Heidelberg, 61–79. https://doi.org/10.1007/978-3-030-31578-8_4

[29] I.Nikolić, A.Biryukov (2008), **"Collisions for Step-Reduced SHA-256"** ,In: Nyberg, K. (eds) Fast Software Encryption. FSE 2008. Lecture Notes in Computer Science, vol 5086. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71039-4_1

[30] F.Mendel, V.Rijmen (2007), **"Cryptanalysis of the Tiger Hash Function"** ,In: Kurosawa, K. (eds) Advances in Cryptology – ASIACRYPT 2007. ASIACRYPT 2007.

Lecture Notes in Computer Science, vol 4833. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-76900-2_33

[31] D.Magrin, M.Capuzzo, S.Romagnolo, M. Luvisotto (2017), LoRaWAN ns-3 module [Online]. Available: https://github.com/signetlabdei/lorawan

[32] Semtech, AN 120022 (2015), LoRa Modulation Basics, May 2015. http://www.semtech.com/images/datasheet/an1200.22pdf.

[33] LoRa Alliance Technical Committee (2016), LoRaWAN Specification; Tech. rep. Version 1.0.2; LoRa Alliance Technical Committee:Fremont, CA, USA, 2016, https://resources.lora-alliance.org/document/lorawan-specification-v1-0-2

[34] J.P.Aumasson, S.Neves, Z.Wilcox-O'Hearn, C.Winnerlein (2013), "**BLAKE2: simpler, smaller, fast as MD5"** , In Proceedings of the 11th international conference on Applied Cryptography and Network Security119-135. doi: 10.1007/978-3-642-38980-1_8.

[35] D.J.Bernstein (2008), **"ChaCha, a variant of Salsa20"** ,Workshop Record of SASC 2008: The State of the Art of Stream Ciphers

[36] Crypto++ Library 8.7 (2023), Crypto++: A Free C++ Class Library of Crypto graphic schemes. Available : https://github.com/weidai11/cryptopp