Journal of International Commercial Law and Technology
Print ISSN: 1901-8401

Website: https://www.jiclt.com/

Article

Low-footprint, Energy-Efficient Memory Cryptographic Algorithm For Protecting Data in an Internet Of Things Setting

Article History:

Name of Author:
Shallu Kinger[1], Raj Kumar[2], Gurdas Singh[3]

Affiliation:
[1]Research Scholar, RIMT University, Mandi Gobindgarh, Punjab, India
EmailID: shallukinger1985@gmail.com
[2]Professor and HOD , Department of Computer Applications, RIMT University, Mandi Gobindgarh, Punjab, India.
EmailID: raj.kumar@rimt.ac.in
[3]Assistant Professor, Department of Computer Science, Gujranwala Guru Nanak Khalsa College, Ludhiana, India.
EmailID: ggurdas@gmail.com

***Corresponding author*:**
shallukinger1985@gmail.com

Abstract: The rapid growth of Internet of Things (IoT) applications has intensified the demand for secure data communication under strict resource constraints such as limited memory, low computational power, and restricted energy availability. Conventional cryptographic algorithms are often unsuitable for IoT devices due to their high complexity and resource consumption. This work proposes an energy-efficient and low-footprint lightweight cryptographic algorithm specifically designed for IoT environments. The proposed scheme combines features of the LESED lightweight block cipher and the RC4 stream cipher to enhance security while maintaining simplicity. The algorithm operates on 64-bit plaintext with an 80-bit working key and a 7-bit constant, utilizing only three encryption rounds and simple logical operations such as XOR, XNOR, and bit shifting. Key scheduling is strengthened using an RC4-based mechanism to increase complexity and resistance against attacks. The algorithm is implemented in C and evaluated using the FELICS benchmarking framework on AVR architecture. Experimental results demonstrate that the proposed cipher achieves significantly lower execution time, reduced memory consumption, and minimal energy usage compared to several well-known lightweight cryptographic algorithms. Furthermore, security analysis shows strong confusion and diffusion properties, high key sensitivity, a large key space, and compliance with the Strict Avalanche Criterion. These results confirm that the proposed algorithm is well-suited for secure and efficient data protection in resource-constrained IoT devices.

Keywords Lightweight cryptography; Internet of Things (IoT); Energy-efficient encryption; Low memory footprint; SHSED; RC4; Key scheduling; FELICS evaluation; Embedded security; Resource-constrained devices

How to cite this article: Shallu Kinger, Raj Kumar, Gurdas Singh, Low-footprint, Energy-Efficient Memory Cryptographic Algorithm For Protecting Data in an Internet Of Things Setting, *J Int Commer Law Technol.* 2025;6(1): 1390-1399.

INTRODUCTION

The Internet of Things (IoT) has several applications in many facets of human existence today. IoT examples include wearable technology, smart homes, and car tracking systems. Kevin Ashton coined the term "Internet of Things," which was first used in an RFID experiment in 1999. Then, in 2009, the phrase "Internet of Things" was introduced to describe a group of connected devices that could link wirelessly . Technology is currently advancing at an accelerating rate, with networks essentially utilizing the Internet of Things.

However, there are certain restrictions in the realm of IoT

devices. For instance, a low processing speed, inadequate memory, a constrained computational capability, etc. IoT defense entails a number of responsibilities, including creating access control methods and utilizing security measures to preserve keys. [1]. Since IoT devices exchange sensitive data, data security and integrity are the most crucial considerations when sharing data. Because IoT devices have limited resources, traditional encryption, which offers data security and integrity, is ineffective in this context. Since lightweight encryption offers adaptable algorithms to handle devices with constrained resources, it is ideal for data security and integrity in the Internet of Things. Numerous researchers have proposed either hardware-based or software-based lightweight ciphers[2]. Unlike ciphers that rely on hardware, software-oriented to have comparable reduced manufacturing and maintenance costs and greater scalability [3]. Additionally, channel attacks are not possible with software-directed ciphers.

1.Literature Review

The rapid expansion of the Internet of Things (IoT) has led to the deployment of billions of interconnected devices in areas such as smart homes, healthcare, industrial automation, and intelligent transportation systems. While IoT technology enhances automation and data-driven decision making, it also introduces serious security and privacy challenges due to the constrained nature of IoT devices[4]. Limited memory, low computational capability, and restricted energy supply make it difficult to implement conventional cryptographic solutions in IoT environments.

Traditional cryptographic algorithms such as AES, DES, and RSA provide strong security guarantees; however, their high computational complexity, large key sizes, and significant memory requirements make them unsuitable for resource-constrained IoT devices[5]. Several studies have shown that applying standard encryption techniques on IoT nodes leads to increased execution time, higher energy consumption, and excessive memory usage, which ultimately reduces device lifetime and system efficiency. As a result, lightweight cryptography has emerged as a promising solution to address IoT security requirements while maintaining acceptable performance.

Lightweight cryptographic algorithms are specifically designed to minimize computational overhead, memory footprint, and power consumption while preserving adequate security levels[6]. These algorithms are broadly categorized into lightweight block ciphers and lightweight stream ciphers. Block ciphers such as PRESENT, HIGHT, LBLOCK, LED, TWINE, and PRINCE have been widely studied for IoT and embedded systems. PRESENT, for example, is known for its small hardware footprint, while HIGHT offers reasonable security with moderate performance. However, many of these algorithms still require multiple rounds and complex key scheduling, which can increase execution cycles and energy usage in software implementations.

Stream ciphers have also been explored for lightweight applications due to their simple structure and low latency. RC4 is one of the earliest and most widely used stream ciphers because of its simplicity and speed. It operates on bytes or bits sequentially without requiring padding or block alignment. Although RC4 has known weaknesses when used improperly, researchers continue to adapt and enhance its key scheduling and diffusion mechanisms for lightweight and hybrid encryption schemes[7]. Modified versions of RC4 have been proposed to improve randomness, increase resistance to attacks, and strengthen key generation processes in constrained environments.

Hybrid cryptographic approaches that combine block ciphers and stream ciphers have gained attention in recent years. These approaches aim to exploit the strengths of both techniques, such as the strong diffusion of block ciphers and the speed and simplicity of stream ciphers. Several researchers have proposed combining lightweight block cipher structures with stream-based key scheduling to improve security without significantly increasing resource consumption[8]. Such designs often rely on simple logical operations like XOR, XNOR, modular addition, and bit shifting to reduce computational complexity.

The LESED (Light and extremely safe for encryption/decryption) algorithm is a lightweight block cipher inspired by the IDEA encryption scheme. SHSED uses simple arithmetic and logical operations and supports efficient encryption and decryption with reduced memory requirements. Previous studies have demonstrated that LESED provides a good balance between security and performance, making it suitable for cloud and lightweight environments[9]. However, when used independently, LESED may still require enhancements in key scheduling and diffusion to strengthen resistance against cryptanalytic attacks in IoT applications.

Performance evaluation of lightweight cryptographic algorithms is commonly conducted using benchmarking frameworks such as FELICS (Fair Evaluation of Lightweight Cryptographic Systems)[10]. FELICS provides a standardized environment to compare encryption and decryption cycles, code size, RAM usage, and energy consumption on embedded platforms such as AVR microcontrollers. Comparative studies using FELICS indicate that algorithms with fewer rounds, simpler operations, and optimized key schedules consistently outperform traditional and even some lightweight ciphers in IoT scenarios.

Despite the availability of numerous lightweight encryption schemes, there remains a need for cryptographic algorithms that achieve high security with minimal execution cycles, ultra-low memory usage, and reduced power consumption[11]. Existing solutions often involve trade-offs between security strength and resource efficiency. Therefore, designing an encryption algorithm that combines efficient key scheduling, strong confusion and diffusion properties, and minimal computational overhead continues to be an active research area.

In response to these challenges, recent research focuses on developing novel lightweight cryptographic algorithms or enhancing existing ones through hybrid designs, simplified round structures, and optimized key generation techniques. Such approaches aim to ensure data confidentiality, integrity, and resistance to brute-force and known-plaintext attacks while meeting the strict constraints of IoT devices.

1.2. Overview of LESED

(Light and extremely safe for encryption/decryption) is a low-power encryption and decryption algorithm that is based on the cloud computing IDEA encryption algorithm [12]. It employs a 256-bit key for 64-bit data sizes for both encrypted and decrypted data, as well as a constant 7-bit variable called CST. It codes and decodes in 31 rounds.

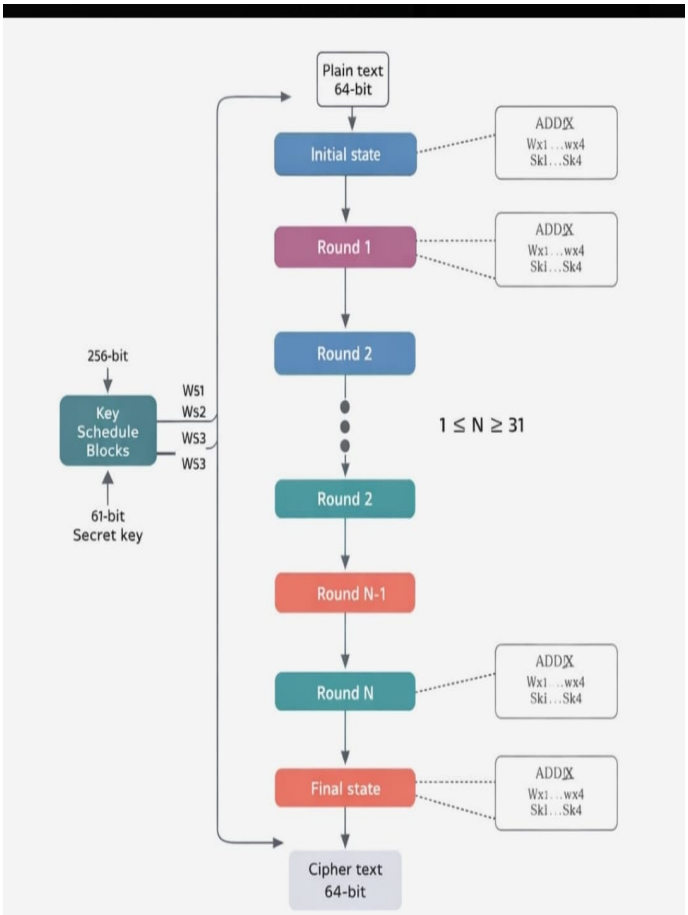
The coding, decoding, and key generation phases are the three stages of the LESED algorithm's execution. Three execution functions—the Initial state, the Round state, and the Final state—are used during the coding step[13].

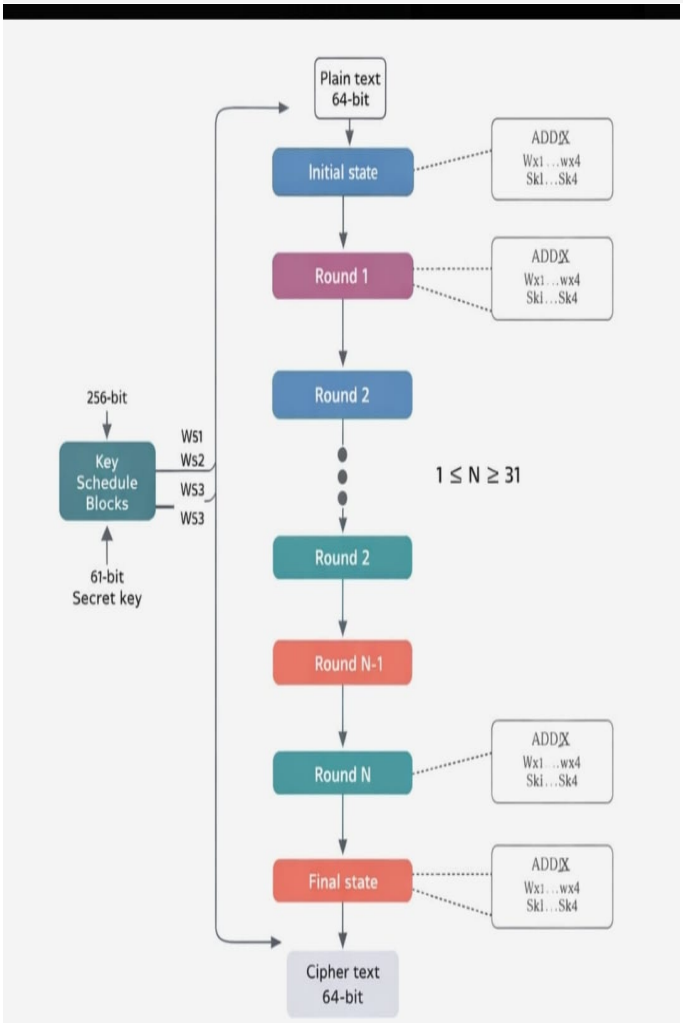
As seen in Figure 1.1, the mathematical operations of addition and XOR are employed in the initial and final cases, while the same mathematical functions are also used in the round case, along with the shift operation. In contrast to the encryption phase, the decoding phase involves mathematical operations (XOR and subtraction).

In the key generation phase, the algorithm generates two keys: a sub key (SK), which is a key produced by the algorithm, and a work key (WK), which is a key entered by the user[6]. In the first and final stages of the cryptographic procedure, the working key is utilized. In the round state, the sub-key is utilized between the Initial as well as the final condition[14].

1.3. Rivest Cipher 4 (RC4) Overview

A form of cryptographic algorithm known as a stream cipher is RC4. Because it is a stream cipher, it processes the incoming data simultaneously, regardless of whether it is in the form of a bit sequence or bytes[15]. The element's range will be subjected to encryption or decryption throughout this scenario. This approach eliminates the need to add extra bytes for encryption or wait for a predetermined quantity of data supplied before it is interpreted. The RS4 algorithm's key creation process is depicted in Figure 1.2. initializing the S-box, generating an array of data to organize the password, and flipping the matrix to speed up the process as well as the matrix flipping procedure, which intensifies the diffusion and confusion process. An S-Box, S [1], S [2],..., S [256], which is part of RC4 encryption, contains a permutation of the digits 1 through 256, where the permutation is a function. key , having a useful length. according to Figure 1.3.





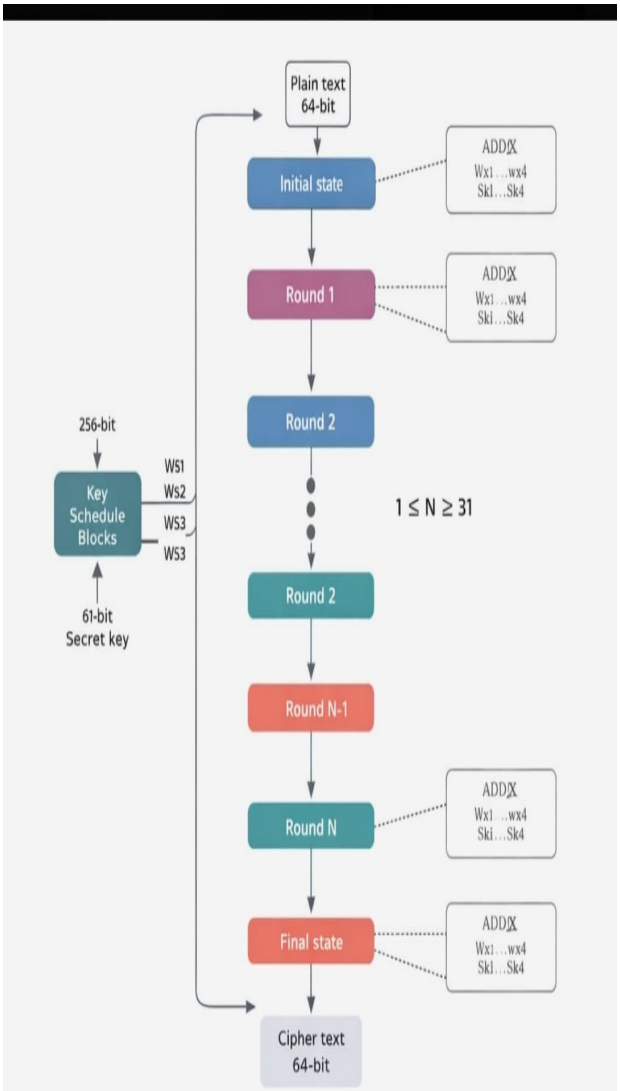


Fig 1.1 Structure of LESED Cryptosystem

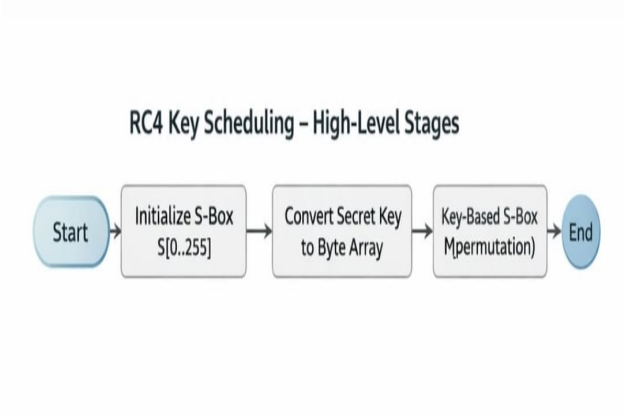


Fig.1.2 RC4 stages of key

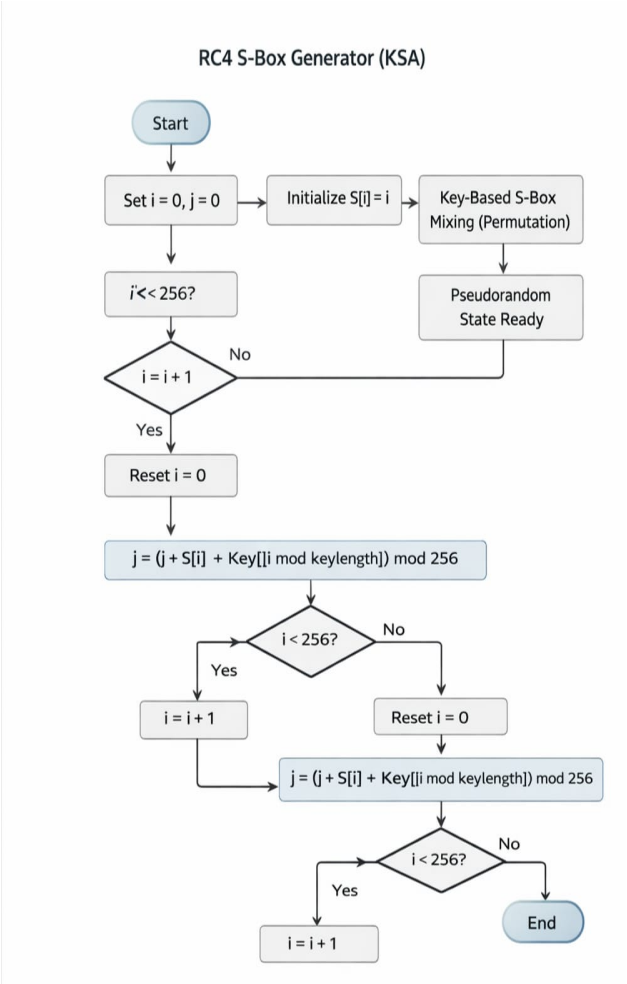


Fig 1.3 S-Box Generator of RC4

1.4. The Proposed Algorithm

The suggested method in this study paper suggests a newly designed lightweight encryption algorithm based on the LESED and RC4 cryptosystem. The algorithm's encryption relies on just three rounds[16]. The algorithm uses 64-plaintext, an 80-bit key, and a 7-bit constant known as CST for encryption and decryption. For encoding and decoding, the technique makes use of basic mathematical operations called XOR, XNOR, and shifting. Initial_ State, Round_ State, and Final _State are the three functions that the algorithm uses[17]. The technique uses two different kinds of keys: sub-keys (SK) and work-keys (WK). As a result, for devices with restricted resources, the algorithm uses less memory and energy while achieving excellent security and speed[18]. From the Figure 5.4 shows the current algorithm's three functions, which are as follows:

Important scheduling feature

The encryption feature

The decryption function

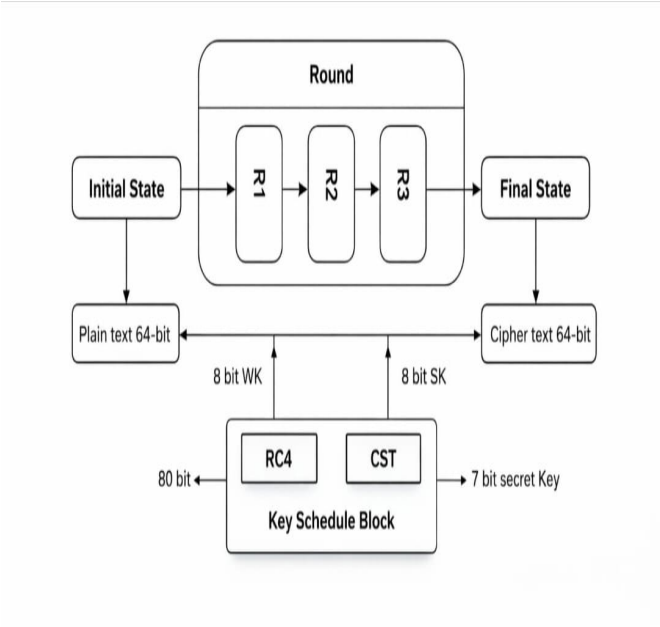


Fig.1.4 Structure of current cryptosystem

1.4.1. Crucial Scheduling Role

Two keys are generated for the encryption and decryption procedures in the LESED algorithm. The user enters the first key, which is known as the work-key (WK). As demonstrated by Algorithm 1.1(CST), the other key, known as a sub-key (SK), is created by a constant in the algorithm that consists of seven bits to produce a sequence of bits, after which the key is extracted from the series[19]. As demonstrated by Algorithm 1.2 (WK_RS4), the work-key (WK) in the current algorithm is generated using the RS4 algorithm to increase complexity and make the key strong and challenging for attackers to guess[20].

Algorithm 1.1: Algorithm of CST

Input: Non

Output :SK[12]//96bitkey

1. CST11[1101100]//constant7bit
2. CST12[1...96]
3. **Begin**
4. For i=1to7
5. CST12[i]=CST11[i]
6. **Next i**
7. For i=1to89
8. CST12[7+i]=XOR(CST12[i+3],CST[i])
9. **Next i**
10. SK[1]=Convert To Hex(CST12[1..8])
11. SK[2]=Convert To Hex(CST12[9..17])
12. .
13. .
14. SK[12]=Convert To Hex(CST12[88..96])

Return SK

Algorithm1.2:The algorithm of generated WK_RS4

Input: Key[10]//80bits Key.

Output: WK[10]//80bitsKey

1. S[1…256]
2. K[1…256]
3. j=1
4. **Begin**
5. For i=1to256
6. S[i]=S[i]-1
7. **Next i**
8. For i=1to256
9. j=(j+ S[i]+Key[I mod(Key. length)])mod256
10. swap S[i],S[j]
11. **Next i**
12. **i=j=1**
13. ForK1=1toKey_length
14. Get position I by(i+1)
15. Get position j by j=(j+ S[j])
16. Swap S[i],S[j]
17. Get value t=S[i]+S[j]
18. WK[K1]=t
19. **Next K1**
20. **Return WK**

1.4.2. Function of Encryption

Encryption_ Block is the encryption function that is displayed in Algorithm 1.3. Three algorithms—1.4, 1.5, and 1.6—are called by the function. Using the encryption key, the encryption block transforms the plain text into an unknown text[21]. The Initial_ State and Final_ State functions use the working_ key (WK) in conjunction with the basic logic operations XOR and XNOR[22]. As seen in Figure 1.5, the sub-key (SK) is utilized with XOR, XNOR, and circular shift in the Round_ State/

Algorithm1.3: Algorithm of Encryption_ Block

Input: Key[10],State[8]//Input Key, plain_ text

Output: Cipher_ Text[8]//cipher_ text

1. SK[12]//Sub_ Key
2. WK[10]//Work_ Key
3. **Begin**
4. WK=**WK_RS4**(Key)
5. SK=**CST**()
6. State=**Initial_ State**(State, WK)
7. Index=1
8. For i=1to3
9. State=**Round_ State**(State, SK, Index)
10. Index=Index+4

11. Next i
12. State=**Final_ State**(State, WK)
13. For i=1to8
14. **Cipher Text**[i]=State[i]
15. Next i
16. **Return Cipher Text**

Algorithm1.4: Algorithm of Initial_ State

Input: WK[10],State[8]//Input Key ,plain _text

Output: Cipher_ Text[8]//cipher_ text

X[10]// template

Begin

1. X[1]=XOR(State_ Text[1],WK[1])
2. X[2]=State_ Text[2]
3. X[3]=XOR(State _Text[3],WK[2])
4. X[4]=State_ Text[4]
5. X[5]=XOR(State _Text[5],WK[3])
6. X[6]=State_ Text[6]
7. X[7]=XOR(State_ Text[7],WK[4])
8. X[8]=XNOR(State _Text[8],WK[8])
9. For i=1to8
10. Cipher_ Text[i]=X[i]
11. Next i
12. **Return Cipher_ Text**

Algorithm1.5: Algorithm of Round_State

Input: SK[10],State_ Text[8]//Sub Key ,State_ Text

Output: Cipher_ Text[8]//cipher_ text

1. X[10]// template
2. **Begin**
3. X[1]=XOR(XOR(State_ Text[7],SK[Index]),State_ Text[8])
4. X[2]=State_ Text[1]
5. Index=Index+1
6. X[2]=XOR
7. (XNOR(State_ Text[1],SK[Index]),State_ Text[2])
8. X[3]=State_ Text[2]
9. Index=Index+1
10. X[4]=XOR(XOR(State_ Text[3],SK[Index]),State_ Text[4])
11. X[5]=State_ Text[4]
12. Index=Index+1
13. X[7]=XOR(XOR(State_ Text[5],SK[Index]),State_ Text[6])
14. X[8]=State_ Text[7]
15. For i=1 to 8
16. Cipher _Text [i]=X[i]
17. Next i

Return Cipher _Text

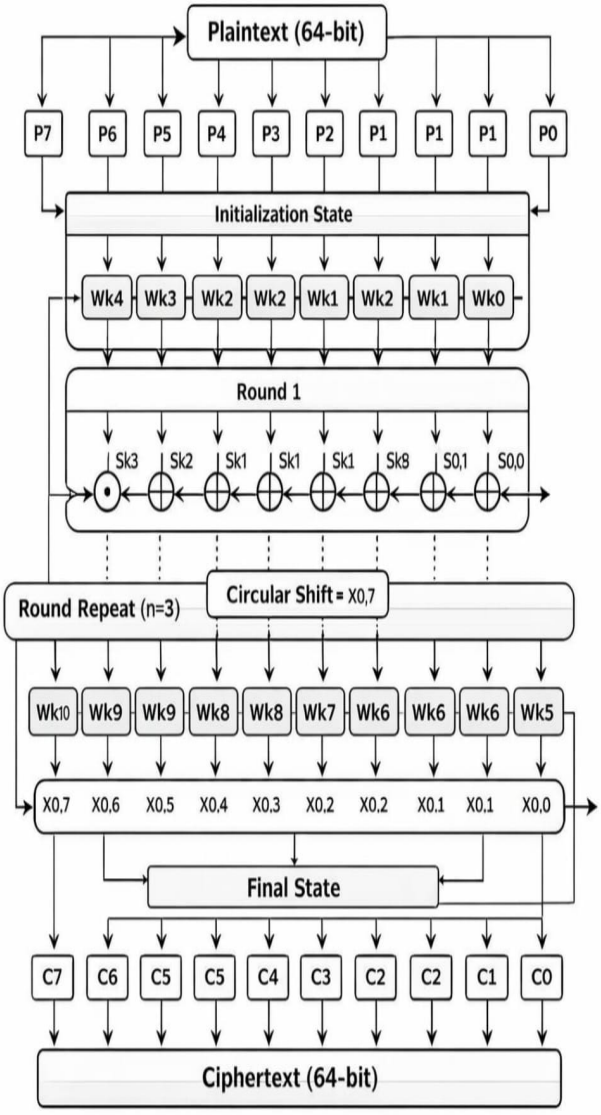


Figure1.5.Structure of Encryption Function for the Proposed Algorithm.

1.4.3. Function of Decryption

The algorithm's decryption approach, which is the opposite of the encryption method, is depicted in Figure 1.6..

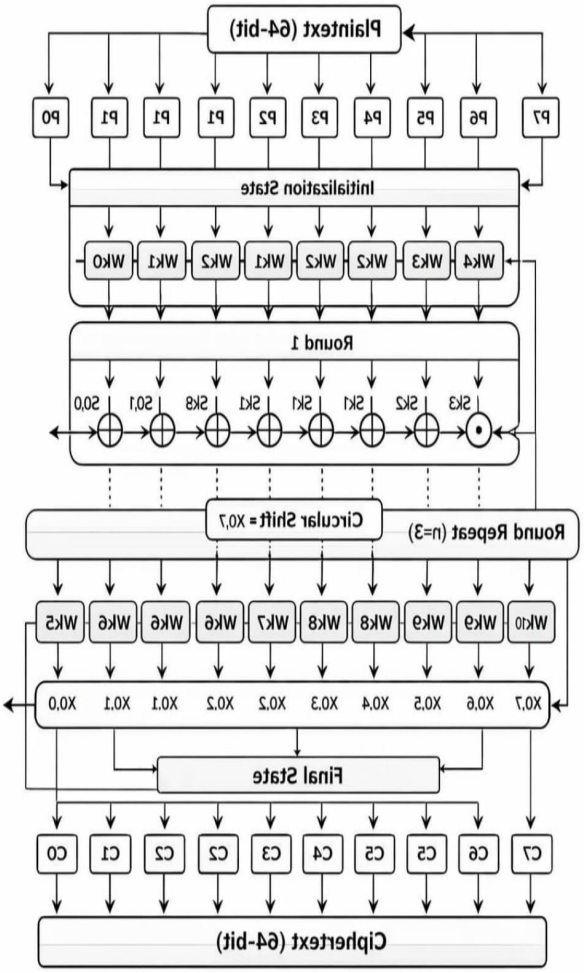


Figure1.6: Structure of Decryption Function for the Proposed Algorithm.

1.5. Algorithm Simulation and Implementation

The suggested method is made for Internet of Things devices and is lightweight. The C programming language is used to write the algorithm[23]. A Linux operating system application called FELICS (Fair Evaluation of Lightweight Cryptographic Systems) also simulates the method[24]. As seen in Figure 1.7, the program first analyzes the algorithm in terms of encoding and decoding (whether it is proper or not).

The current algorithm has extremely high levels of confusion and diffusion processes. Any encryption mechanism must be tested using these procedures. The absence of a partial connection between the encrypted and original text causes confusion[25]. The process of changing important parts of the original text so that the statistical structure of the original text is obscured is called diffusion[26].

The tool's primary metrics for evaluating lightweight encryption methods are speed (execution time), memory usage, and program size (FELICS). The following parameters are used to measure the performance of the current method and obtain an edge over other algorithms:

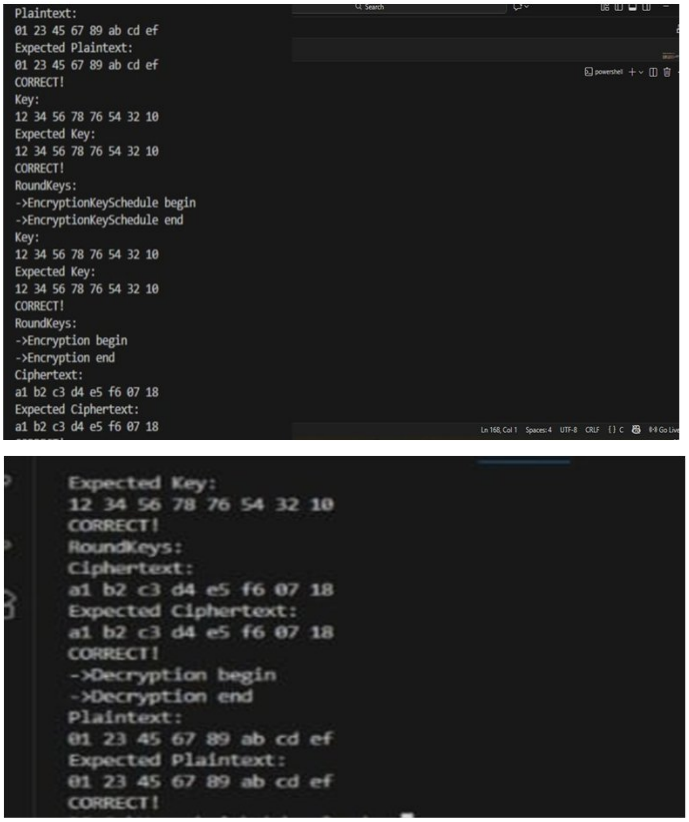


Figure1.7:Implementation Test of Proposed Cipher on FELICS.

1.5.1. Encryption/decryption execution time

The time required to execute the algorithm is the amount of time spent implementing the encoding and decoding of certain data. As demonstrated in Table 1.1 and Figure 1.8, the algorithm's execution time is crucial for measuring lightweight algorithms that operate on Internet of Things applications (with constrained resources)[27]. It is also important to note the correlation between execution time and hardware power consumption, where the longer the execution time, the more energy is consumed. Consequently, the best algorithms are those that require less iterations to implement for both encoding and decoding[28]. Thus, of the techniques displayed in Figures 1.9 and 1.10, our algorithm has the fewest execution cycles.

Results for the Suggested Cipher with Typical Cipher Implementations on AVR Architecture in FELICS Tools are shown in Table 1.1.

The following equation (1.1) quantifies the algorithm's power consumption:

$$E=I*VCC*ls*N \quad (1.1)$$

In this case, VCC stands for the system's supply voltage. T is consumed in seconds, while I is the average current in amperes. Clock time is represented by–. It is the clock cycle number.Thus, the clock period is equal to 1/fSec/Cycle. The Atmel Atmega128 normally operates at 16 MHz, accepts voltage between 2.7 and 5.5, and draws an average of 20 mA of electricity. The power consumption of the recommended cipher and the existing ciphers is contrasted in Figure 1.11.

Table1.1:Results for Proposed Cipher with Common Cipher Implementations on AVR Architecture in FELICS Tools.

Cipher	Device	Block Size (bit)	Key Size (bit)	RAM (byte)	Code Size (byte)	Encrypted-Key Schedule	Encryption (cycles)	Decryption (cycles)
SIT ALGORITHM	AVR	64	64	22	826	2130	3006	2984
ALGORITHM 1	AVR	64	80	18	1354	1407	3359	3434
High-Performance ALGORITHM	AVR	64	80	18	1256	1309	2874	2996
NLBCIT Algorithm	**AVR**	**64**	**64**	**16**	**2094**	**1218**	**1401**	**918**
SBCD Algorithm	**AVR**	**64**	**64**	**16**	**1850**	**950**	**558**	**528**
Proposed Algorithm	AVR	64	80	18	1054	24	282	264

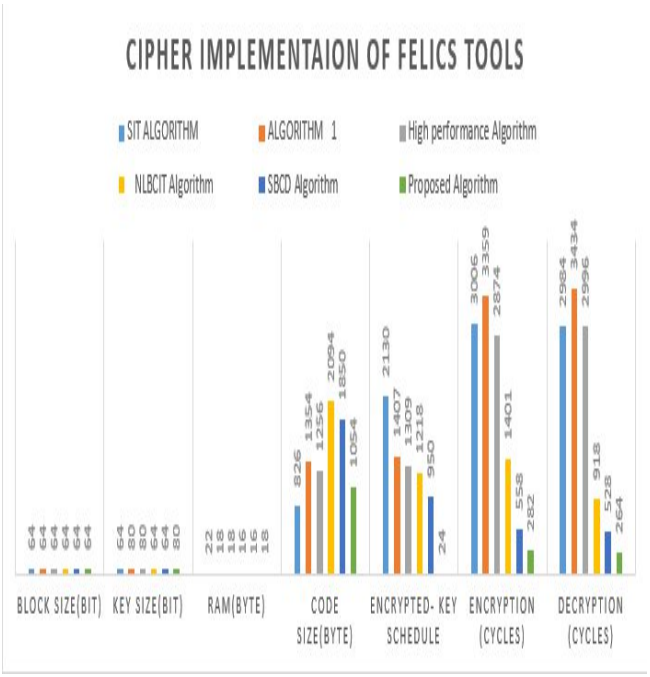


Figure1.8:Comparison Analysis in Terms of Code Size, RAM, Encryption/Decryption (Cycles).

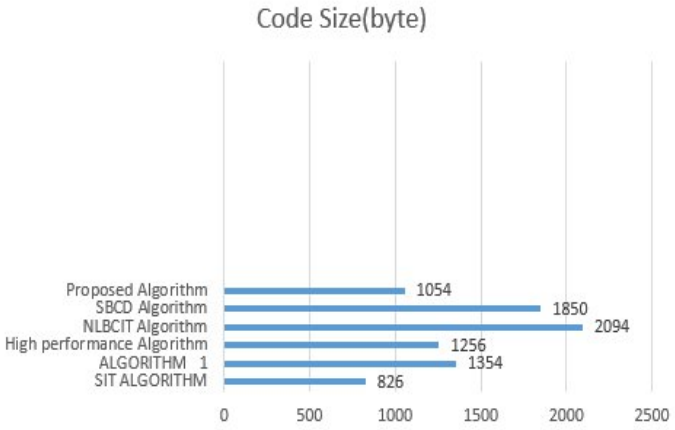


Figure1.9:Code Size Comparisons Algorithms

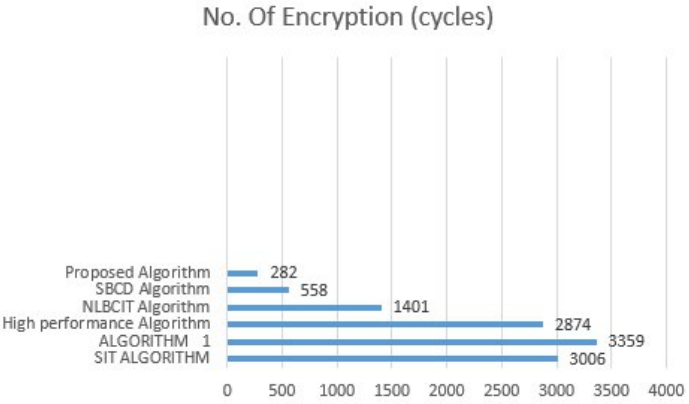


Figure1.10: Number of Cycle Encryption (Execution Time).

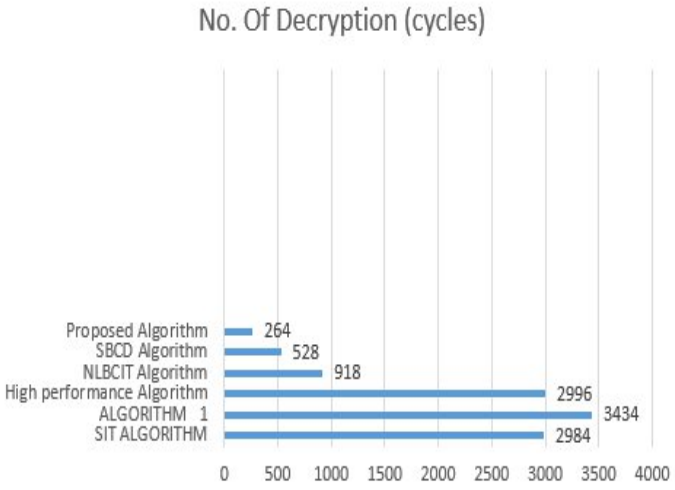


Figure1.11 Number o f Cycle Decryption (Execution Time).

1.5.2. Usage of Memory

One of the most crucial requirements for a lightweight algorithm operating on IoT devices with constrained resources is memory. These devices have a restricted amount of memory because they are tiny[29]. Thus,

algorithms that use less memory are the ones that ought to function on Internet of Things apps and devices. Table 1.1 indicates that the current algorithm uses the least amount of memory.

5.5.3. Space for Keys

The key space is the collection of all potential keys that can be utilized to create a key for any cryptosystem's encoding procedure. One of the most important characteristics that underpins the vastness and resilience of a cryptosystem's algorithms is the key space. Wide key space techniques in the cryptosystem effectively thwart attackers and offer resiliences against brute force assaults, selected cipher text, and known plain text[30]. Two keys are produced by the suggested technique; the first key is 80 bits and is based on the RC4 encryption algorithm, while the second key (SK) is produced automatically by the algorithm using a predetermined bit number that is found to produce 96 bits. There is a great deal of space created by the two keys, and it is secure enough to prevent hackers from conducting a comprehensive search of the keys. There are no known plaintext attacks against the suggested approach. Additionally, the suggested approach has the greatest key space (2^176–2^528), eliminating the possibility of a brute force assault.

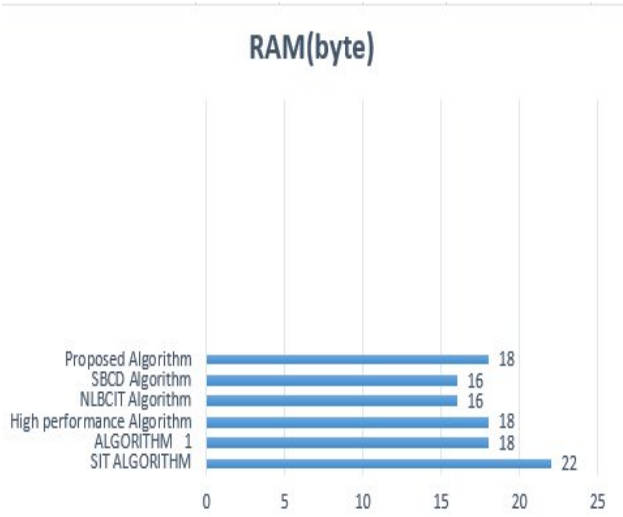


Figure1.12:RAM Consumption Comparison of Algorithms.

1.5.4. Sensitivity of Keys

Key sensitive algorithms are those used in encryption and decryption that cannot recover the original data from the encrypted data in the event that one or more bits of the encryption key change. After encryption, the Strict Avalanche Criteria (SAC) standard is used to measure data changes{31]. If there is a 50% change in the encrypted text's bits, the encryption is deemed successful. Additionally, we observe how sensitive the key is to changes in just one bit. Furthermore, the algorithm excels at confusion and diffusion, which refer to the absence of a connection between the plaintext and the encrypted content), which strengthens the algorithm's resistance to many kinds of attacks[32].

1.6. Summary

A lightweight algorithm for Internet of Things devices is suggested in this chapter. The algorithm utilizes only three rounds of basic mathematical operations. The algorithm's accuracy in encoding and decoding is checked using the FELICS tool, which also measures the memory usage and execution time (cycles) required for data encoding and decoding. The algorithm's optimal use of memory, reduced execution time, and lower energy consumption have all demonstrated its effectiveness. Using key space, key sensitivity, and the SAC standard, the algorithm also offers robust data security.

REFERENCES

1. D. Sehrawat, and N. S. Gill, ―Security Requirements of IoT Applications in Smart Environment‖,in:Proc.of2ndInternationalConferenceonTrendsinElectronics and Informatics (ICOEI 2018), pp. 324-329, 2018. Available: https://doi.org/10.1109/ICOEI.2018.8553681.
2. D. Sehrawat and N. S. Gill, ―Ultra bright: A tiny and fast ultra lightweight block cipher for iot,‖ Int. J. Sci. Technol. Res., vol. 9, no. 2, pp. 1063–1068,2020.
3. L.M.Alramini,―Implementationofproposedlightweightcryptosystemforuse inCloudComputingSecurity‖(MasterDegree,MiddleEastUniversity),June,2018
4. L.M.Alramini,―Implementationofproposedlightweightcryptosystemforuse inCloudComputingSecurity‖(MasterDegree,MiddleEastUniversity),June,2018
CloudComputingSecurity‖(MasterDegree,MiddleEastUniversity),June,2018.
5. Anjum Sheikh, "IoT Based Security System for Smart Homes", International Journal of Computer Sciences andEngineering, Vol.07, Special Issue.11, pp.35-38, 2019.
6. G.Zhouetal.,―Smartsavingsonprivatecarpoolingbasedoninternetofvehicles,‖ J. Intell. Fuzzy Syst., vol. 32, no. 5, pp. 3785–3796, Apr. 2017, doi: 10.3233/JIFS- 169311
7. A. Majumdar, T. Debnath, S. K. Sood, and K. L. Baishnab, ―Kyasanur ForestDisease Classification Framework Using Novel Extremal Optimization Tuned NeuralNetworkinFogComputingEnvironment,‖J. Med.Syst.,vol.42,no.10, 2018, doi: 10.1007/s10916-018-1041-3.
8. T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann,

and L. Uhsadel, ―A survey of lighweight-cryptography implementations,‖IEEE Des. Test Comput., vol. 24,no.99, p. x9, 2008, doi: 10.1109/mdt.2007.4343586.
9. A. Biswas, A. Majumdar, S. Nath, A. Dutta, and K. L. Baishnab, ―LRBC: a lightweight block cipher design for resource constrained IoT devices,‖ J. Ambient Intell. Humaniz. Comput., Jan. 2020, doi: 10.1007/s12652-020-01694-9.
10. D.Dinu,A.Biryukov,J.Großschädl,D.Khovratovich ,Y.LeCorre,andL.Perrin,FELICS– Fair EvaluationofLightweightCryptographicSystems,‖ NISTWork. Light.Cryptogr.,vol.128,NIST,2015,
11. E. Biham, ―New types of cryptanalytic attacks using related keys,‖ Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 765 LNCS, pp. 398–409, 1994, doi: 10.1007/3-540-48285- 7_34.
12. L. Khelladi, Y. Challal, A. Bouabdallah, and N. Badache, ―On security issues in embeddedsystems:Challengesandsolutions,‖Int.J.I nf.Comput.Secur.,vol.2, no. 2, pp. 140–174, 2008, https://doi.org/10.1504/IJICS.2008.018515.
13. J. Daemen, L. Knudsen, and V. Rijmen, ―The block cipher SQUARE,‖ Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 1267, pp. 149–165, 1997, doi: 10.1007/bfb0052343.
14. M. Ebrahim and Chai Wai Chong, ―Secure Force: A low-complexity cryptographic algorithm for Wireless Sensor Network (WSN),‖ in 2013 IEEE International Conference on Control System, Computing and Engineering, Nov. 2013, pp. 557–562, doi: 10.1109/ICCSCE.2013.6720027.
15. B.Schneier,―Descriptionofanewvariable-lengthkey,64-bitblockcipher(blow- fish),‖ in International Workshop on Fast Software Encryption. Springer, 1993, pp. 191–204.
16. 16.M. Usman, I. Ahmed, M. I. Aslam, S. Khan, and U. A. Shah, ―SIT: A Lightweight EncryptionAlgorithmforSecureInternetofThings,‖I nt.J.Adv.Comput.Sci. Appl., vol. 8, no. 1, pp. 402–411, 2017, doi: 10.14569/IJACSA.2017.080151.
17. G. A. Al-rummana, G. N. Shinde, and A. H. A. Al-ahdal, ―MapReduced Based: A NewStreamCipherTechniqueforDataEncryption,‖I nt.J.Eng.Adv.Technol., vol. 9, no. 5, pp. 763–769, 2020, doi: 10.35940/ijeat.e9394.069520.
18. A. F. Webster and S. E. Tavares, ―On the Design of S-Boxes,‖ Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 218 LNCS, pp. 523–534, 1986, doi: 10.1007/3-540-39799-X_41.
19. S. Somaraj and M. A. Hussain, ―Performance and security analysis for image encryption using key image,‖ Indian J. Sci. Technol., vol. 8, no. 35, 2015, doi: 10.17485/ijst/2015/v8i35/73141.
20. A. Kanso and M. Ghebleh, ―An efficient lossless secret sharing scheme for medical images,‖ J. Vis. Commun. Image Represent., vol. 56, no. September, pp. 245–255, 2018, doi: 10.1016/j.jvcir.2018.09.018.
21. S. Kandar, D. Chaudhuri, A. Bhattacharjee, and B. C. Dhara, ―Image encryptionusingsequencegeneratedbycyclicgroup, ‖J.Inf.Secur.Appl.,vol.44,pp.117– 129, 2019, doi: 10.1016/j.jisa.2018.12.003.
22. M. E. Hodeish, L. Bukauskas, and V. T. Humbe, ―A new efficient TKHC-based image sharing scheme over unsecured channel,‖ J. King Saud Univ. - Comput. Inf.Sci., no. xxxx, 2019, doi: 10.1016/j.jksuci.2019.08.004.
23. B.Schneier,―Descriptionofanewvariable-lengthkey,64-bitblockcipher(blow- fish),‖ in International Workshop on Fast Software Encryption. Springer, 1993, pp. 191–204.
24. D.Coppersmith,―DataEncryptionStandard(DES)a nditsstrengthagainstattacks,‖
25. IBMJ.Res.Dev.,vol.38,no.3,pp.243–250,1994,doi: 10.1147/rd.383.0243.
26. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, ―The 128-Bit BlockcipherCLEFIA(ExtendedAbstract),‖pp.181–195,2007,doi:10.1007/978- 3-540-74619-5_12.
27. C. De Cannière, O. Dunkelman, and M. Knežević, ―KATAN and KTANTAN - A familyofsmallandefficienthardware-orientedblockciphers,‖Lect.Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5747 LNCS, pp. 272–288, 2009, doi: 10.1007/978-3-642- 04138-9_20.
28. C. DeCanni,―Trivium,‖pp.244–266,2008.
29. W. Wu and L. Zhang, ―LBlock: A lightweight block cipher,‖ Lect. Notes Comput.Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6715 LNCS, pp. 327–344, 2011, doi: 10.1007/978-3-642-21554-4_19.
30. T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, ―Twine: A lightweight, versatile block cipher,‖ECRYPT Work. pnLight. Cryptogr. LC11, pp. 146–169,2011, [Online].Available: http://www.nec.co.jp/rd/media/code/research/images/twine_LC11.pdf.
31. R. BeaulieuandS.Treatman-clark,―TheSimonandSpeckFamiliesofLightweight
32. BlockCiphers,‖2013

