Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 8526206, 9 pages
https://doi.org/10.1155/2021/8526206

Research Article

LLSFIoT: Lightweight Logical Security Framework for Internet of Things

Isha Batra,[1] **Hatem S. A. Hamatta**,[2] **Arun Malik**,[1] **Mohammed Baz**,[3] **Fahad R. Albogamy**,[4] **Vishal Goyal**,[5] **and Sultan S. Alshamrani**[6]

[1]*School of Computer Science and Engineering, Lovely Professional University, Phagwara, India*
[2]*Department of Applied Sciences, Aqaba University College, Al Balqa Applied University, Aqaba, Jordan*
[3]*Department of Computer Engineering, College of Computer and Information Technology, Taif University, PO Box. 11099, Taif 21994, Saudi Arabia*
[4]*Turabah University College, Computer Sciences Program, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia*
[5]*Department of Computer Science, Punjabi University, Patiala, India*
[6]*Department of Information Technology, College of Computer and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia*

Correspondence should be addressed to Sultan S. Alshamrani; susamash@tu.edu.sa

Received 28 May 2021; Revised 17 July 2021; Accepted 30 July 2021; Published 18 August 2021

Academic Editor: Dr. Muhammad Shafiq

Current research in Internet of Things (IoT) is focused on the security enhancements to every communicated message in the network. Keeping this thought in mind, researcher in this work emphasizes on a security oriented cryptographic solution. Commonly used security cryptographic solutions are heavy in nature considering their key size, operations, and mechanism they follow to secure a message. This work first determines the benefit of applying lightweight security cryptographic solutions in IoT. The existing lightweight counterparts are still vulnerable to attacks and also consume calculative more power. Therefore, this research work proposes a new hybrid lightweight logical security framework for offering security in IoT (LLSFIoT). The operations, key size, and mechanism used in the proposed framework make its lightweight. The proposed framework is divided into three phases: registration, authentication, and light data security (LDS). LDS offers security by using unique keys at each round bearing small size. Key generation mechanism used is comparatively fast making the compromise of keys as a difficult task. These steps followed in the proposed algorithm design make it lightweight and a better solution for IoT-based networks as compared to the existing solutions that are relatively heavy weight in nature.

1. Introduction

A fresh primitive cryptography known as lightweight cryptography is specifically being put on the market for use as integrated systems in resource-restricted settings like radio frequency identification (RFID) IoT [1]. Lightweight will not be soft in nature, but will not be enforced on many apps. The attacker is restricted by lightweight algorithms with the exposure of only restricted information per key [2]. Lightweight alternatives are used to marinate the necessary trade between efficiency, safety, and assets [3]. The major challenges in IoT are restricted instruments such as RFID and battery-operated detectors. Particular consideration should therefore be paid to limiting the use of its funds and at the same moment to provide safety [4]. Solutions for lightweight cryptography deliver both safety and efficiency [5]. The easiest approach seems to be of IoT resource restrictions. Lightweight alternatives provide safety only through the exposure of restricted operational information [6]. The limitations in existing network are the use of large key size, block size, complex round structure, and the implementation requirements [7]. Being a resource constrained network, security in IoT should be using a security mechanism using less key size, block size, simple round structures, and simple implementation requirements [8, 9].

For Lightweight solutions, the National Institute of Standards and Technology (NIST) sets a minimum key size requirement of 112 bits. Even smaller key sizes are more vulnerable to brute force attack [10]. The following requirement is for a small block size. The lightweight cipher's block size should be smaller than that of conventional cyphers. For instance, if the block size is 64 bits rather than the 128 bits used by AES, a greater number of plaintext blocks can be encrypted [11]. Additionally, memory requirements will decrease. Following that, a simple round structure should be used: The rounds used in lightweight cyphers should be simpler than those used in conventional cryptography [12]. For instance, a round can be simplified by substituting a 4-bit S-Box for an 8-bit S-Box. This also reduces the amount of memory required. Increasing the total number of rounds to be fired may lower the amount of security that can be improvised [13]. The requirements take into account the fact that the device should be capable of either encryption or decryption. Rather than implementing the entire cypher, only required operations should be implemented [14, 15]. This comes out with an issue while implementing lightweight solutions in IoT but once it is implemented, the overall resources and life of network can be improved [16].

The contribution of this research work is to overcome the limitations of existing solutions by making changes in design of the security algorithm. In comparison to conventional block cyphers, the requirements of lightweight security solutions are lower for key size. Existing security solutions like AES, SIMON, and SPECK, they have more key size requirements as compared to the proposed algorithm.

The remaining paper is arranged accordingly. In Section II, work related to IoT security is reviewed. Existing solutions for providing security and authentication are covered. Section III propose the hybrid lightweight security solution for IoT comprising three phases, i.e, registration, authentication, and data security. Later, in Section IV, round key generation schedule of data security mechanism is discussed. Section V discusses the complete one round structure for Data security. Section VI analyzes the proposed algorithm by evaluating the mean and standard deviation. Finally, conclusion states the currents state of art and the benefit of the proposed security framework.

2. Related Work

Light weight means the algorithms that require fewer and optimal performance funds. The lightweight term does not refer to the weakness of the algorithm [17]. As the trend for future appliances with restricted systems has changed, a great deal of effort was produced to optimize AES for these apps. However, adaptation to the requirements of these systems in the AES was not suitable [18]. Although AES has been implemented quickly, however, it is still very complicated and has big codes that do not comply with the needs [19]. In [20], it is mentioned that AES is utilized as validation component in RFID-based frameworks. The AES is used in the application layer as an integrated COAP system. Advanced encryption standards (AES) is an institutionalized symmetrical square figure by NIST. It uses a replacement phase scheme and deals with a 128-bit square-length 4×4 network [21]. Each byte is influenced by the effects of subbytes, row shifting, MIXED COLUMNS, and ADD ROUND KEY. The key size that can be used is 128 bits, 192 bits, or 256 bits. AES is as yet defenseless against man-in-center assault [22, 23].

The author suggested in [24], PRESENT which is SPN based and used as an ultra-light safety calculation. It uses 4-bit info and S-box rates to advance devices at the replacement layer. It has 80 or 128 parts of main size and operates on 64 pieces. PRESENT is listed as a lightweight cryptography scheme in ISO/IEC 29192-2 : 2012 "Lightweight cryptography." On 26 out of 31 rounds [25], PRESENT is indefensible from differential attack. In [26], author referred SIMON $2n$ an N-bit word cipher forming a $2n$-bit block. N can have 16, 24, 32, 48, and 64 values. SIMON $2n$ using key as k -word key (kn-bit) is referred as SIMON $2n$/kn. Therefore, SIMON 96/144 will be working on a block of 96-bit plaintext and using key of 144 bits. SIMON is a member of the block cypher family with varying block sizes. It can support 32, 48, 64, 96, and 128 bits of block size that further can work on varying key sizes.

In [27], author referred SPECK highlighted that SPECK requirements are like SIMON. SPECK 128/128 therefore means the 128 bit file length SPECK block code that sucks the 128-bit button. The SPECK supporting block and key size is identical to that of SIMON. SPECK uses Feistel structure performing bitwise XOR, circular shits, and modular addition in each round at both directions [28]. In [29], TWINE is described as a 64 bit block cipher forming a basic Feistel structure. Feistel functions consist of 16 4-bit subblocks using key addition. Two key sizes 80 and 128 bits are supported by TWINE. TWINE operates on total 36 rounds with same round function. In [30], author mentioned FANTOMAS as an LS-design example (LS consists of L-boxes using bit-sliced looking tables and S-boxes). The block cipher FANTOMAS can be displayed with the $s \times L$ bit array. The $s \times s$ parts are permutation for each matrix row, whereas the permutation for each matrix row is $L \times L$. Consider, for instance, a 128-bit FANTOMAS key and block length. The s-bits are 8, and the L-bits are 16.

3. Proposed Lightweight Logical Security Framework for IoT (LLSFIoT)

The proposed LLSFIoT is divided into three phases: registration, authentication, and LDS. When a new device enters the network, the credentials are first registered with the server using the key sharing mechanism. Once the device has the credentials, mutual authentication between the device and the server will take place before initiating any communication. Using the LDS algorithm, the data transmitted by and from the device is secured. The notations used in the process of registration, authenticationn and data security are shown in Table 1.

Table 1: Notations used in LLSFIoT.

Symbol	Description
D_i	ith device
IS	Information server
ID_D	Identity of device
ID_S	Identity of server
K_S	Key shared between device and server
SN	Sequence number
UID	Unique IDs
K_a	Alternate keys
T_V	Temporary variable
n	Number of bits in each word
$2n$	Block size/number of input bits
SK_i	ith key subblock
S^{-x}	Left rotation by x bits
S^y	Right rotation by y bits

3.1. Phase 1: Registration. Steps corresponding to registration phase are detailed below:

Step 1. Device D_i will initiate a connection that has been established with IS by submitting its IDD to the IS making use of a secure medium.

Step 2. IS following receipt of the connection request from D_i computes a nonce value N_S. This N_S is used to compute a shared key $K_{S,}$ where $K_S = \text{ID}_S \oplus h(\text{ID}_D \| N_S)$.

Step 3. Additionally to this, IS generates a collection of unique IDs, $\text{UID} = \{\text{id}_{1,} \text{id}_{2,} \text{id}_3, \cdots . \text{id}_n \cdot\}$, and set of alternate keys $Ka = \{k_{a1}, k_{a2}, \cdots . k_{an}\}$ in relation to one another $\text{uid}_i \in \text{UID}$.

Step 4. Additionally, IS, a sequence number, is a generated randomly SN. As a result, for each request submitted by the D_i, IS generates K_S, unique IDs, alternate keys, and SN. If D_i makes an additional request to IS, a new SN is generated. The onus of IS is to maintain one copy of SN in database and forward same copy to the D_i. The benefit of using SN is to avoid any replay that the intruder may inject.

Step 5. Before the authentication process actually begins, IS checks to see if the SN sent by D_i matches one already stored in the database. Authentication phase 2 will be active when this match occurs. whereas IS ends connection with D_i and requires D_i to use one UID and Ka couple if match does not occur in SN.

The pair will be used once, and the entry will be removed in both the IS and D_i database. I will send a message at the end D_i encrypted using public key of D_i having a set of values: K_S, $\{\text{id}_i, k_{\text{ai}}\}$, SN, and in its own database keeps the same values as the D_i ID, i.e. IDD.

3.2. Phase 2: Authentication. In the authentication phase, two way mutual authentication is performed between D_i and IS. Steps corresponding to authentication phase are detailed below:

Step 6. D_i by taking a nonce value N_1 generates a variable $V_1 = h(\text{ID}_D \| K_S \oplus N_1)$.

Step 2. Now, D_i creates a message of request having $\{V_1, \text{ID}_D, \text{SN}\}$ to the IS.

Step 3. On the off chance that SN is not accessible with D_i, D_i will use one of the $\{\text{id}_i, \text{k}_{\text{ai}}\}$ pair where k_{ai} can be used in replacement of K_S.

Step 4. On receiving request from D_i, The IS verifies the message's SN or checks that additional parameters are legitimate or not if they match the matching SN of the D_i stored in the database. The value of N1 is later calculated by IS.

Step 5. If all the parameters are validated, then IS after taking a nonce value N_2 will generate a new random sequence number $\text{SN}_{\text{new}=}\ h\ (\text{ID}_\text{D} \| K_S \| N_1) \oplus \text{SN}$ and computes a temporary variable $T_V = h(\text{ID}_D \| K_S \| N_2) \oplus \text{SN}_{\text{new}}$ and computing variable $V\_2 = h(\text{ID}_D \| K_S \| N_1 \| T_V)$.

Step 6. D_i on receiving message containing $\{V_2, \text{SN}_{\text{New}}, T_V\}$ from the IS computes the value $h(\text{ID}_D \| K_S \| N_1 \| T_V)$ and compares it with V_2. If match occurs, D_i computes nonce N $_2$ using $T_V = h(\text{ID}_D \| K_S \| N_2) \oplus \text{SN}_{\text{new.}}$

3.3. Phase 3: Lightweight Data Security (LDS) Algorithm. Once mutual authentication is performed between D_i and IS, the next step is to offer data security using the encryption method. Data is taken in blocks of 64 bits each, and the size of K_S shared between D_i and IS that is 128 bits. To offer security, a lightweight data security (LDS) algorithm is proposed. This algorithm takes of the secure data communication and offers the services for security such as confidentiality of data and integrity of data.

Proposed LDS works on 20 rounds using addition, rotation, and XOR (ARX) operations. This flexibility of choosing the number of rounds lies with the user depending upon the execution time required and also on full diffusion. The three operations ARX are chosen for offering optimum security trading off with lightweight solution considering the IoT application scenario. The reason for choosing only these operations for a round is discussed later in Section 4. The structure of LDS consisting of 20 rounds using ARX operations and a key generation function is represented through Figure 1.

4. Generation of Subkeys for Each Round

For each round, two n-bit subkey bocks are required, considering n as the number of bits in a word. Block size that can be taken as input will be $2n$. Here, block size of 64 bits is assumed; so, value of n is 32. Key size is taken as 128 bits. Therefore, for 20 rounds, 40 key subblocks have each of 32

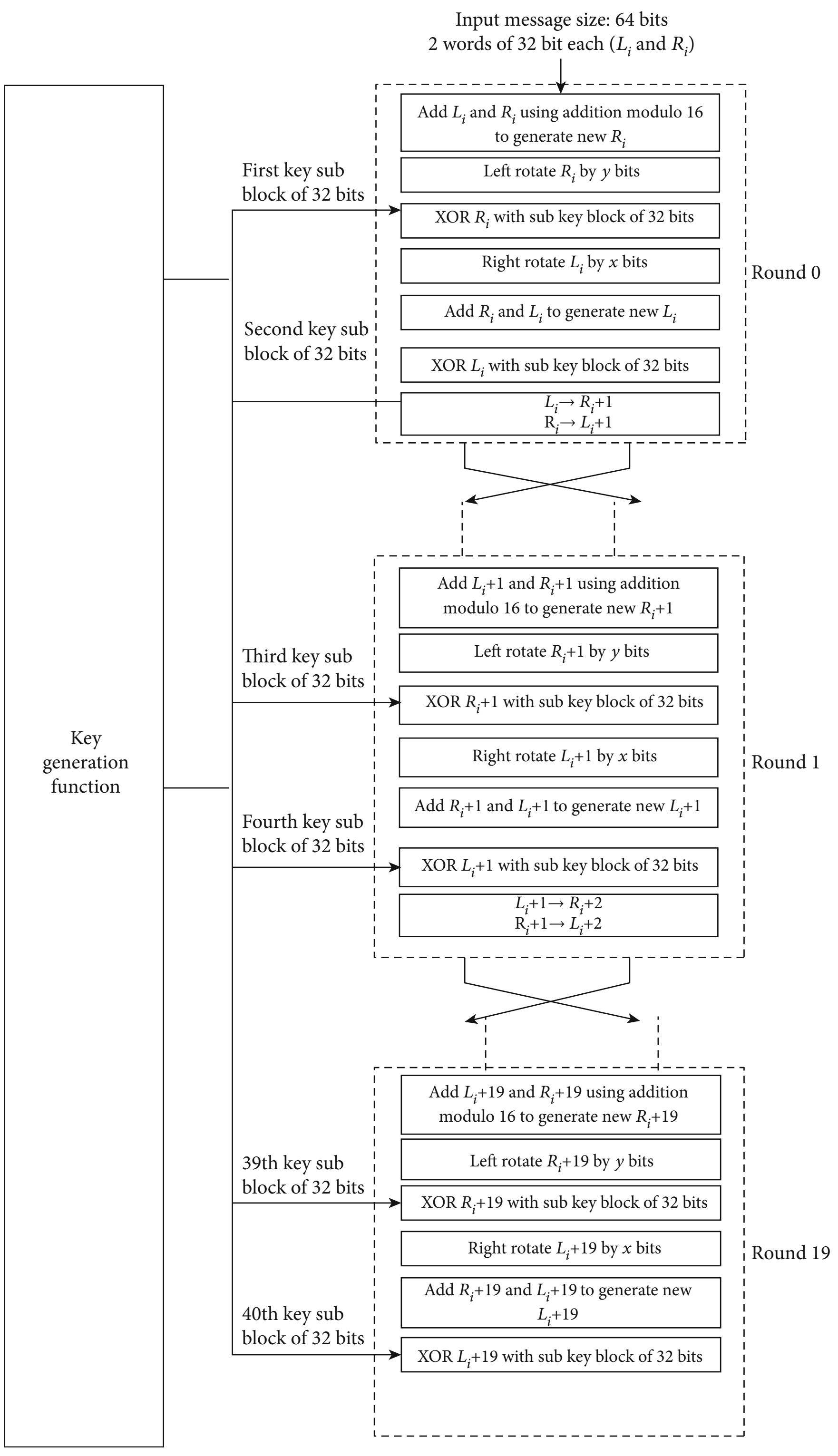

Input message size: 64 bits
2 words of 32 bit each (L_i and R_i)
Add L_i and R_i using addition modulo 16 to generate new R_i
Left rotate R_i by y bits
XOR R_i with sub key block of 32 bits
Right rotate L_i by x bits
Add R_i and L_i to generate new L_i
XOR L_i with sub key block of 32 bits
$L_i \rightarrow R_i+1$
$R_i \rightarrow L_i+1$
Round 0
First key sub block of 32 bits
Second key sub block of 32 bits
Add L_i+1 and R_i+1 using addition modulo 16 to generate new R_i+1
Left rotate R_i+1 by y bits
XOR R_i+1 with sub key block of 32 bits
Right rotate L_i+1 by x bits
Add R_i+1 and L_i+1 to generate new L_i+1
XOR L_i+1 with sub key block of 32 bits
$L_i+1 \rightarrow R_i+2$
$R_i+1 \rightarrow L_i+2$
Round 1
Third key sub block of 32 bits
Fourth key sub block of 32 bits
Key generation function
Add L_i+19 and R_i+19 using addition modulo 16 to generate new R_i+19
Left rotate R_i+19 by y bits
XOR R_i+19 with sub key block of 32 bits
Right rotate L_i+19 by x bits
Add R_i+19 and L_i+19 to generate new L_i+19
XOR L_i+19 with sub key block of 32 bits
Round 19
39th key sub block of 32 bits
40th key sub block of 32 bits

FIGURE 1: LDS structure.

bit out of 128 bit long key. A key generation mechanism is required for getting the key subblocks for each round of operation.

Subkey generation is done in such a manner that key generator gives a unique and random subkey every time it is run. For a good key generator mechanism, if the generated subkey is compromised by cryptanalysis, other subkeys should not be identified. The subkeys are generated from the main key of 128 bits. As stated earlier, each round requires two subkey blocks. The mechanism of key generation function consists of a key generation that divides the keys into subblocks. Key generator generates subkeys for two rounds at a time.

Therefore, for 20 rounds, key generator will work for 10 times and generate 4 subkey blocks each time, making a total of 40 subkey blocks. The whole mechanism of key generator is explained through following steps:

Step 1. The original key (K_i) of 128 bits is given as input to subkey generator.

Step 2. Sub key generator generates 4 sub key blocks of 32 bit each. Two key sub blocks of 32 bits are passed as input to first round and next two key sub blocks of 32 bits are passed as input to second round.

Step 3. Bits in original K_i are processed using a mixing function to generate input for the running the key generator for the next time. From there again, 4 key subblocks are generated for next two rounds.

Step 4. Mixing function takes as input the output of the previous key generator function. For the first time, after the execution of key generator, original K_i consists of 4 key subblocks, let us say, SK_1, SK_2, SK_3, SK_4, each of 32 bits. Mixing function performs the XOR operation in circular rotation. All the bits of SK_1 are XORed with random bits of SK_2, SK_2 is XORed with random bits of SK_3, SK_3 is XORed with random bits of SK_4, and SK_4 is XORed with random bits of SK_1. $\mathrm{SK}_1(0)$ represents the first bit of key subblock SK_1. Figure 2 shows the block diagram for operation of key generator. The sample equations to generate subkey can be represented mathematically in Table 2.

Step 5. Step 3 and step 4 are repeated till all the 40 subkey blocks are generated for all the 20 rounds.

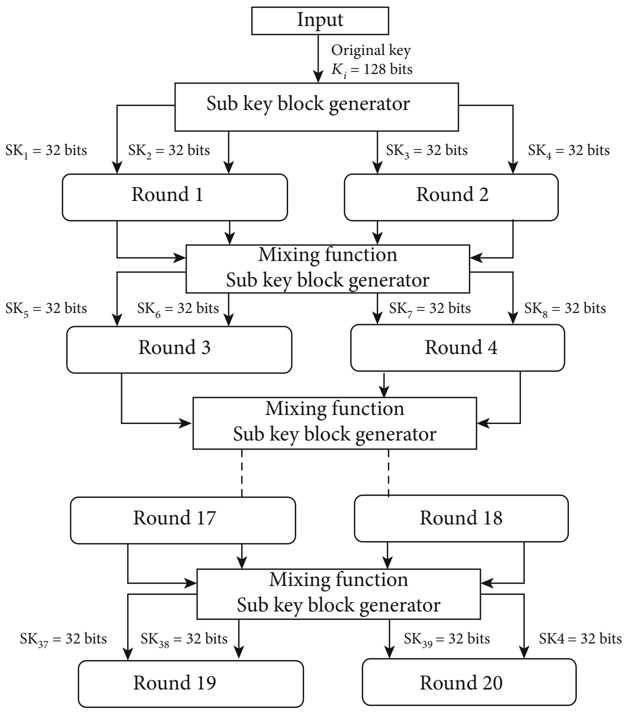
5. Round Function of LDS

LDS framework works on the Feistel-like structure. Operations used during the encryption process of LDS are

(i) Addition modulo 2^n, considering n as the number of bits in a word. If n is 16, block size will be 32 and for n taken as 32, block size will be 64 bits. Addition modulo is preferred over multiplication modulo. There may be multiple reasons for choosing addition over multiplication. First, multiplication require more cycles as compared to addition even with the fastest CPUs. Second, operation of multiplication may lead to timing attacks

(ii) Bitwise XOR, $\oplus$: most block ciphers work using XOR as the basic operation as compared to other operations like AND and OR. Numbers of factors supporting XOR over other operations are first, XOR operation works on reversible procedure. When encryption is performed on original text XOR with key to generate cipher text, same key when operated using XOR with cipher text the resultant will be same original text. Second, XOR can be realized using the NAND gate requiring few transistors as compared to other operations, making its hardware implementation quite easier. Third, in XOR, the output is dependent on both the operands as compared to AND and OR. In AND, if one of the operand is false, second is not evaluated at all. In OR, if one of the operand is true and second is not evaluated at all, whereas, in XOR, if first operand is true or false, second needs to be evaluated for getting the expected output

(iii) R^{-b} and R^{b} are left and right rotations respectively, where b is the number of bits to rotate. Rotations are preferred over shift as rotation when used with the XOR operation that creates maximum diffusion in the resultant output with alteration in a single input bit. On the other hand, when shift is used with the XOR, then diffusion created is less in output with alteration in a single input bit


Input
Original key K_i = 128 bits
Sub key block generator
SK_1 = 32 bits
SK_2 = 32 bits
SK_3 = 32 bits
SK_4 = 32 bits
Round 1
Round 2
Mixing function
Sub key block generator
SK_5 = 32 bits
SK_6 = 32 bits
SK_7 = 32 bits
SK_8 = 32 bits
Round 3
Round 4
Mixing function
Sub key block generator
Round 17
Round 18
Mixing function
Sub key block generator
SK_{37} = 32 bits
SK_{38} = 32 bits
SK_{39} = 32 bits
SK4 = 32 bits
Round 19
Round 20
FIGURE 2: Block diagram for key subblock generator function.

The input block of n bits is divided into two equal halves. For example, if input text is 64 bits long, it will be divided into 32 bits each represented as L_i and R_i. L_i represents the left subblock, and R_i represents the right subblock. The left and the right subblock in a particular round is evaluated as

$$L_i = (S^{-x}L_i + (S^{y}(L_i + R_i) \oplus SK_2)) \oplus SK_1,$$
$$R_i = S^{y}(L_i + R_i) \oplus SK_2. \tag{1}$$

Therefore, the round function of LDS is denoted as

$$F(L_i, R_i) = ((S^{-x}Li + (S^{y}(L_i + R_i) \oplus \mathrm{SK}_2)) \oplus \mathrm{SK}_1, S^{y}(L_i + R_i) \oplus \mathrm{SK}_2), \tag{2}$$

Table 2

$\mathrm{SK}_1(0) = \mathrm{SK}_1(0) \oplus \mathrm{SK}_2(31)$	$\mathrm{SK}_2(0) = \mathrm{SK}_2(0) \oplus \mathrm{SK}_3(0)$
$\mathrm{SK}_1(1) = \mathrm{SK}_1(1) \oplus \mathrm{SK}_2(0)$	$\mathrm{SK}_2(1) = \mathrm{SK}_2(1) \oplus \mathrm{SK}_3(1)$
$\mathrm{SK}_1(2) = \mathrm{SK}_1(2) \oplus \mathrm{SK}_2(1)$	$\mathrm{SK}_2(2) = \mathrm{SK}_2(2) \oplus \mathrm{SK}_3(2)$
$\mathrm{SK}_1(3) = \mathrm{SK}_1(3) \oplus \mathrm{SK}_2(2)$	$\mathrm{SK}_2(3) = \mathrm{SK}_2(3) \oplus \mathrm{SK}_3(3)$
$\mathrm{SK}_1(4) = \mathrm{SK}_1(4) \oplus \mathrm{SK}_2(3)$	$\mathrm{SK}_2(4) = \mathrm{SK}_2(4) \oplus \mathrm{SK}_3(4)$
$\mathrm{SK}_1(5) = \mathrm{SK}_1(5) \oplus \mathrm{SK}_2(4)$	$\mathrm{SK}_2(5) = \mathrm{SK}_2(5) \oplus \mathrm{SK}_3(5)$
$\mathrm{SK}_1(6) = \mathrm{SK}_1(6) \oplus \mathrm{SK}_2(5)$	$\mathrm{SK}_2(6) = \mathrm{SK}_2(6) \oplus \mathrm{SK}_3(6)$
$\mathrm{SK}_1(7) = \mathrm{SK}_1(7) \oplus \mathrm{SK}_2(6)$	$\mathrm{SK}_2(7) = \mathrm{SK}_2(7) \oplus \mathrm{SK}_3(7)$
$\mathrm{SK}_1(8) = \mathrm{SK}_1(8) \oplus \mathrm{SK}_2(7)$	$\mathrm{SK}_2(8) = \mathrm{SK}_2(8) \oplus \mathrm{SK}_3(8)$
$\mathrm{SK}_1(9) = \mathrm{SK}_1(9) \oplus \mathrm{SK}_2(8)$	$\mathrm{SK}_2(9) = \mathrm{SK}_2(9) \oplus \mathrm{SK}_3(9)$
$\mathrm{SK}_1(10) = \mathrm{SK}_1(10) \oplus \mathrm{SK}_2(9)$	$\mathrm{SK}_2(10) = \mathrm{SK}_2(10) \oplus \mathrm{SK}_3(10)$
$\mathrm{SK}_1(11) = \mathrm{SK}_1(11) \oplus \mathrm{SK}_2(10)$	$\mathrm{SK}_2(11) = \mathrm{SK}_2(11) \oplus \mathrm{SK}_3(11)$
$\mathrm{SK}_1(12) = \mathrm{SK}_1(12) \oplus \mathrm{SK}_2(11)$	$\mathrm{SK}_2(12) = \mathrm{SK}_2(12) \oplus \mathrm{SK}_3(12)$
$\mathrm{SK}_1(13) = \mathrm{SK}_1(13) \oplus \mathrm{SK}_2(12)$	$\mathrm{SK}_2(13) = \mathrm{SK}_2(13) \oplus \mathrm{SK}_3(13)$
$\mathrm{SK}_1(14) = \mathrm{SK}_1(14) \oplus \mathrm{SK}_2(13)$	$\mathrm{SK}_2(14) = \mathrm{SK}_2(14) \oplus \mathrm{SK}_3(14)$
$\mathrm{SK}_1(15) = \mathrm{SK}_1(15) \oplus \mathrm{SK}_2(14)$	$\mathrm{SK}_2(15) = \mathrm{SK}_2(15) \oplus \mathrm{SK}_3(15)$
$\mathrm{SK}_1(16) = \mathrm{SK}_1(16) \oplus \mathrm{SK}_2(15)$	$\mathrm{SK}_2(16) = \mathrm{SK}_2(16) \oplus \mathrm{SK}_3(16)$
$\mathrm{SK}_1(17) = \mathrm{SK}_1(17) \oplus \mathrm{SK}_2(16)$	$\mathrm{SK}_2(17) = \mathrm{SK}_2(17) \oplus \mathrm{SK}_3(17)$
$\mathrm{SK}_1(18) = \mathrm{SK}_1(18) \oplus \mathrm{SK}_2(17)$	$\mathrm{SK}_2(18) = \mathrm{SK}_2(18) \oplus \mathrm{SK}_3(18)$
$\mathrm{SK}_1(19) = \mathrm{SK}_1(19) \oplus \mathrm{SK}_2(18)$	$\mathrm{SK}_2(19) = \mathrm{SK}_2(19) \oplus \mathrm{SK}_3(19)$
$\mathrm{SK}_1(20) = \mathrm{SK}_1(20) \oplus \mathrm{SK}_2(19)$	$\mathrm{SK}_2(20) = \mathrm{SK}_2(20) \oplus \mathrm{SK}_3(20)$
$\mathrm{SK}_1(21) = \mathrm{SK}_1(21) \oplus \mathrm{SK}_2(20)$	$\mathrm{SK}_2(21) = \mathrm{SK}_2(21) \oplus \mathrm{SK}_3(21)$
$\mathrm{SK}_1(22) = \mathrm{SK}_1(22) \oplus \mathrm{SK}_2(21)$	$\mathrm{SK}_2(22) = \mathrm{SK}_2(22) \oplus \mathrm{SK}_3(22)$
$\mathrm{SK}_1(23) = \mathrm{SK}_1(23) \oplus \mathrm{SK}_2(22)$	$\mathrm{SK}_2(23) = \mathrm{SK}_2(23) \oplus \mathrm{SK}_3(23)$
$\mathrm{SK}_1(24) = \mathrm{SK}_1(24) \oplus \mathrm{SK}_2(23)$	$\mathrm{SK}_2(24) = \mathrm{SK}_2(24) \oplus \mathrm{SK}_3(24)$
$\mathrm{SK}_1(25) = \mathrm{SK}_1(25) \oplus \mathrm{SK}_2(24)$	$\mathrm{SK}_2(25) = \mathrm{SK}_2(25) \oplus \mathrm{SK}_3(25)$
$\mathrm{SK}_1(26) = \mathrm{SK}_1(26) \oplus \mathrm{SK}_2(25)$	$\mathrm{SK}_2(26) = \mathrm{SK}_2(26) \oplus \mathrm{SK}_3(26)$
$\mathrm{SK}_1(27) = \mathrm{SK}_1(27) \oplus \mathrm{SK}_2(26)$	$\mathrm{SK}_2(27) = \mathrm{SK}_2(27) \oplus \mathrm{SK}_3(27)$
$\mathrm{SK}_1(28) = \mathrm{SK}_1(28) \oplus \mathrm{SK}_2(27)$	$\mathrm{SK}_2(28) = \mathrm{SK}_2(28) \oplus \mathrm{SK}_3(28)$
$\mathrm{SK}_1(29) = \mathrm{SK}_1(29) \oplus \mathrm{SK}_2(28)$	$\mathrm{SK}_2(29) = \mathrm{SK}_2(29) \oplus \mathrm{SK}_3(29)$
$\mathrm{SK}_1(30) = \mathrm{SK}_1(30) \oplus \mathrm{SK}_2(29)$	$\mathrm{SK}_2(30) = \mathrm{SK}_2(30) \oplus \mathrm{SK}_3(30)$
$\mathrm{SK}_1(31) = \mathrm{SK}_1(31) \oplus \mathrm{SK}_2(30)$	$\mathrm{SK}_2(31) = \mathrm{SK}_2(31) \oplus \mathrm{SK}_3(31)$
$\mathrm{SK}_3(0) = \mathrm{SK}_3(0) \oplus \mathrm{SK}_4(1)$	$\mathrm{SK}_4(0) = \mathrm{SK}_4(0) \oplus \mathrm{SK}_1(2)$
$\mathrm{SK}_3(1) = \mathrm{SK}_3(1) \oplus \mathrm{SK}_4(2)$	$\mathrm{SK}_4(1) = \mathrm{SK}_4(1) \oplus \mathrm{SK}_1(3)$
$\mathrm{SK}_3(2) = \mathrm{SK}_3(2) \oplus \mathrm{SK}_4(3)$	$\mathrm{SK}_4(2) = \mathrm{SK}_4(2) \oplus \mathrm{SK}_1(4)$
$\mathrm{SK}_3(3) = \mathrm{SK}_3(3) \oplus \mathrm{SK}_4(4)$	$\mathrm{SK}_4(3) = \mathrm{SK}_4(3) \oplus \mathrm{SK}_1(5)$
$\mathrm{SK}_3(4) = \mathrm{SK}_3(4) \oplus \mathrm{SK}_4(5)$	$\mathrm{SK}_4(4) = \mathrm{SK}_4(4) \oplus \mathrm{SK}_1(6)$
$\mathrm{SK}_3(5) = \mathrm{SK}_3(5) \oplus \mathrm{SK}_4(6)$	$\mathrm{SK}_4(5) = \mathrm{SK}_4(5) \oplus \mathrm{SK}_1(7)$
$\mathrm{SK}_3(6) = \mathrm{SK}_3(6) \oplus \mathrm{SK}_4(7)$	$\mathrm{SK}_4(6) = \mathrm{SK}_4(6) \oplus \mathrm{SK}_1(8)$
$\mathrm{SK}_3(7) = \mathrm{SK}_3(7) \oplus \mathrm{SK}_4(8)$	$\mathrm{SK}_4(7) = \mathrm{SK}_4(7) \oplus \mathrm{SK}_1(9)$
$\mathrm{SK}_3(8) = \mathrm{SK}_3(8) \oplus \mathrm{SK}_4(9)$	$\mathrm{SK}_4(8) = \mathrm{SK}_4(8) \oplus \mathrm{SK}_1(10)$
$\mathrm{SK}_3(9) = \mathrm{SK}_3(9) \oplus \mathrm{SK}_4(10)$	$\mathrm{SK}_4(9) = \mathrm{SK}_4(9) \oplus \mathrm{SK}_1(11)$
$\mathrm{SK}_3(10) = \mathrm{SK}_3(10) \oplus \mathrm{SK}_4(11)$	$\mathrm{SK}_4(10) = \mathrm{SK}_4(10) \oplus \mathrm{SK}_1(12)$
$\mathrm{SK}_3(11) = \mathrm{SK}_3(11) \oplus \mathrm{SK}_4(12)$	$\mathrm{SK}_4(11) = \mathrm{SK}_4(11) \oplus \mathrm{SK}_1(13)$
$\mathrm{SK}_3(12) = \mathrm{SK}_3(12) \oplus \mathrm{SK}_4(13)$	$\mathrm{SK}_4(12) = \mathrm{SK}_4(12) \oplus \mathrm{SK}_1(14)$
$\mathrm{SK}_3(13) = \mathrm{SK}_3(13) \oplus \mathrm{SK}_4(14)$	$\mathrm{SK}_4(13) = \mathrm{SK}_4(13) \oplus \mathrm{SK}_1(15)$
$\mathrm{SK}_3(14) = \mathrm{SK}_3(14) \oplus \mathrm{SK}_4(15)$	$\mathrm{SK}_4(14) = \mathrm{SK}_4(14) \oplus \mathrm{SK}_1(16)$
$\mathrm{SK}_3(15) = \mathrm{SK}_3(15) \oplus \mathrm{SK}_4(16)$	$\mathrm{SK}_4(15) = \mathrm{SK}_4(15) \oplus \mathrm{SK}_1(17)$
$\mathrm{SK}_3(16) = \mathrm{SK}_3(16) \oplus \mathrm{SK}_4(17)$	$\mathrm{SK}_4(16) = \mathrm{SK}_4(16) \oplus \mathrm{SK}_1(18)$
$\mathrm{SK}_3(17) = \mathrm{SK}_3(17) \oplus \mathrm{SK}_4(18)$	$\mathrm{SK}_4(17) = \mathrm{SK}_4(17) \oplus \mathrm{SK}_1(19)$
$\mathrm{SK}_3(18) = \mathrm{SK}_3(18) \oplus \mathrm{SK}_4(19)$	$\mathrm{SK}_4(18) = \mathrm{SK}_4(18) \oplus \mathrm{SK}_1(20)$
$\mathrm{SK}_3(19) = \mathrm{SK}_3(19) \oplus \mathrm{SK}_4(20)$	$\mathrm{SK}_4(19) = \mathrm{SK}_4(19) \oplus \mathrm{SK}_1(21)$
$\mathrm{SK}_3(20) = \mathrm{SK}_3(20) \oplus \mathrm{SK}_4(21)$	$\mathrm{SK}_4(20) = \mathrm{SK}_4(20) \oplus \mathrm{SK}_1(22)$
$\mathrm{SK}_3(21) = \mathrm{SK}_3(21) \oplus \mathrm{SK}_4(22)$	$\mathrm{SK}_4(21) = \mathrm{SK}_4(21) \oplus \mathrm{SK}_1(23)$
$\mathrm{SK}_3(22) = \mathrm{SK}_3(22) \oplus \mathrm{SK}_4(23)$	$\mathrm{SK}_4(22) = \mathrm{SK}_4(22) \oplus \mathrm{SK}_1(24)$
$\mathrm{SK}_3(23) = \mathrm{SK}_3(23) \oplus \mathrm{SK}_4(24)$	$\mathrm{SK}_4(23) = \mathrm{SK}_4(23) \oplus \mathrm{SK}_1(25)$
$\mathrm{SK}_3(24) = \mathrm{SK}_3(24) \oplus \mathrm{SK}_4(25)$	$\mathrm{SK}_4(24) = \mathrm{SK}_4(24) \oplus \mathrm{SK}_1(26)$
$\mathrm{SK}_3(25) = \mathrm{SK}_3(25) \oplus \mathrm{SK}_4(26)$	$\mathrm{SK}_4(25) = \mathrm{SK}_4(25) \oplus \mathrm{SK}_1(27)$
$\mathrm{SK}_3(26) = \mathrm{SK}_3(26) \oplus \mathrm{SK}_4(27)$	$\mathrm{SK}_4(26) = \mathrm{SK}_4(26) \oplus \mathrm{SK}_1(28)$
$\mathrm{SK}_3(27) = \mathrm{SK}_3(27) \oplus \mathrm{SK}_4(28)$	$\mathrm{SK}_4(27) = \mathrm{SK}_4(27) \oplus \mathrm{SK}_1(29)$
$\mathrm{SK}_3(28) = \mathrm{SK}_3(28) \oplus \mathrm{SK}_4(29)$	$\mathrm{SK}_4(28) = \mathrm{SK}_4(28) \oplus \mathrm{SK}_1(30)$
$\mathrm{SK}_3(29) = \mathrm{SK}_3(29) \oplus \mathrm{SK}_4(30)$	$\mathrm{SK}_4(29) = \mathrm{SK}_4(29) \oplus \mathrm{SK}_1(31)$
$\mathrm{SK}_3(30) = \mathrm{SK}_3(30) \oplus \mathrm{SK}_4(31)$	$\mathrm{SK}_4(30) = \mathrm{SK}_4(30) \oplus \mathrm{SK}_1(0)$
$\mathrm{SK}_3(31) = \mathrm{SK}_3(31) \oplus \mathrm{SK}_4(0)$	$\mathrm{SK}_4(31) = \mathrm{SK}_4(31) \oplus \mathrm{SK}_1(1)$

where x and y are the rotation constants. For block size of 64 bits and key size of 128 bits, the value of x is taken as 7, and the value of y is taken as 3. This composition of round function is represented through Figure 3.

6. Evaluating Diffusion Property for LDS Round

Diffusion property of the cryptographic algorithm is focused on incorporating the avalanche effect. It refers to observe the change in number of bits of the output cipher text with a single bit modification in the input original text. With more number of bits affected by diffusion, the cryptographic solution proves to be stronger.

The input original text of 64 bits is divided into 2 subdata blocks of 32 bits each and referred as A and B. The values of rotation constant for creating the diffusion matrix may vary from 0 to 31. Therefore, the total possible combinations of rotation constants x and y each carrying value from 0 to 31 may lead to $32 \times 32 = 1024$ combinations. 1024 combinations can generate 1024 diffusion matrices based on respective designs. A sample diffusion matrix is shown in Table 3 below:

In Table 3, M_{AA} refers to the count of number of bits modified in A by modifying the single bit of A. As there are 32 bits in A, the mean value and the standard deviation are calculated after changing every bit of A and noticing its effect on A and B. Similar effect can be noticed in M_{AB}, M_{BA}, and M_{BB}.

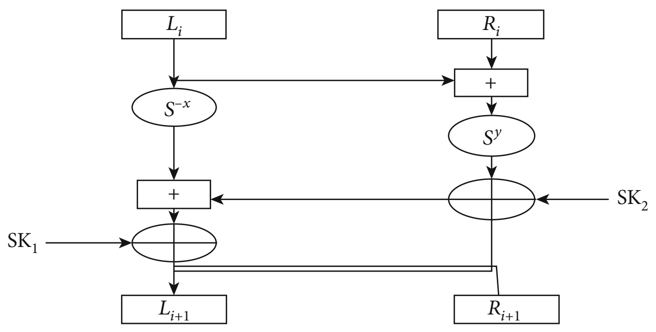


Figure 3: Single round function of LDS.

Table 3: Generalized diffusion table.

Input	Output	
	Left block (A)	Right block (B)
Left block (A)	M_{AA}	M_{AB}
Right block (B)	M_{BA}	M_{BB}

Table 4: Diffusion table considering $x = 7$ and $y = 3$.

Input	Output	
	Left block (A)	Right block (B)
Left block (A)	10.5	12.3125
Right block (B)	6.71815	8.90625
Mean = 9.609, standard deviation = 2.058		

In order to generate the diffusion matrix, certain steps are taken that require the input and rotation constants assumed as x and y here.

Step 1. Input block of 64 bits is divided into two subblocks referred as A and B each of 32 bits.

Step 2. Considering the different combinations of x and y where each can take values from 0 to 31, thus, the overall possible combinations are 1024. Here, the value of x and y is assumed to be fixed; that is, x is taken as 7, and y is taken as 3, while executing the LDS algorithm.

Step 3. Round function of LDS is executed over the data blocks A and B to generate output as A^1 and B^1.

Step 4. Start by modifying one bit of A, then execute LDS over modified A bits and the original B to get output as A^2 and B^2. Compare bits of A^1 with bits of A^2 and calculate the number of bits which have been altered, that will become value of M_{AA}. Similarly, compare bits of B^1 with bits of B^2 and calculate the number of bits which have been altered, that will become value of M_{AB}.

Step 5. Repeat step 3 but this time by modifying one bit of B. Execute LDS over modified B bits and the original A to get output as A^2 and B^2. Compare bits of A^1 with bits of A^2 and calculate the number of bits which have been altered, that will become the value of M_{BA}. Similarly, compare bits of B^1 with bits of B^2 and calculate the number of bits which have been altered, that will become the value of M_{BB}.

Table 5: Diffusion table considering $x = 8$ and $y = 3$.

Input	Output	
	Left block (A)	Right block (B)
Left block (A)	9.375	10.156
Right block (B)	3.156	7.218
Mean = 7.4762, standard deviation = 2.716		

Table 6: Diffusion table considering $x = 7$ and $y = 2$.

Input	Output	
	Left block (A)	Right block (B)
Left block (A)	8.625	11.25
Right block (B)	4.312	6.75
Mean = 7.734, standard deviation = 2.541		

Table 7: Mean and standard deviation with different sets of rotation constants.

Rotation constants	Mean	Standard deviation
$x = 7$ $y = 3$	9.609	2.058
$x = 8$ $y = 3$	7.476	2.716
$x = 7$ $y = 2$	7.734	2.541

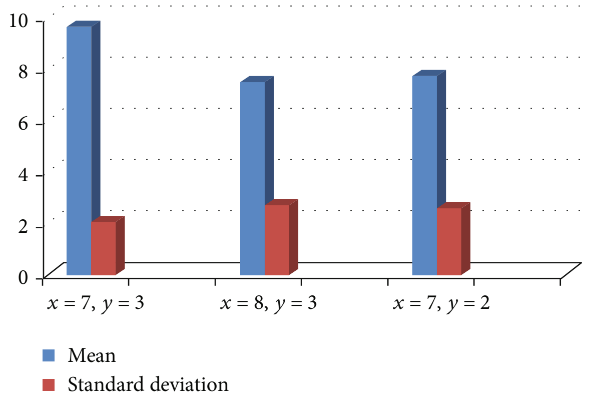


Figure 4: Mean and standard deviation for different sets of rotation constants.

Step 6. Repeat steps 3 and 4 at least 64 times using rotation constant x =7 and $y = 3$ to find the average of each value of matrix M to get diffusion table as shown in Table 4 below.

The possible combinations for x and y can be 1024, but the same steps for creating diffusion table are repeated by considering the most common used rotation constants $x = 8$ and $y = 3$ and second time by considering $x = 7$ and $y = 2$. The diffusion table generated by repeating the steps 64 times for the rotation constants $x = 8$ and $y = 3$ is shown through Table 5 below.

The diffusion table generated by repeating the steps 64 times for the rotation constants $x = 7$ and $y = 2$ is shown through Table 6.

Mean value in all the combinations shows the average number of bits that are affected by changing the individual bits as shown in equation (3)

$$\text{Mean} = (M\text{AA} + \text{MAB} + \text{MBB} + \text{MBA})/4. \tag{3}$$

Standard deviation is calculated by finding the variance after subtracting each data value from the mean and then finding their sum and finally performing square root as shown in equation (4).

$$\text{Standard Deviation} = \sqrt{\sum_{i,j=1}^{2} (\text{Mij} - \text{Mean})}. \tag{4}$$

Rotation constants chosen for creating diffusion matrix and hence calculating mean and standard deviation show the extent of output change by changing input bits. These constants are used to calculate transition probability. For block size of 64 bit block and key size of 128 bits, the mean and standard deviation for three sets of rotation constants are shown in Table 7 and are represented through Figure 4.

Figure 4 clarifies that when same LDS is executed with different combinations of rotation constants, the maximum value and the least standard deviation are observed with rotation constants $x = 7$ and $y = 3$. Therefore, the proposed LDS algorithm chooses the rotation constants $x = 7$ and $y = 3$.

7. Conclusion

Once the data is collected through devices using sensors, the next concern is to offer security to data or devices that play active role in communication. Therefore, this research work proposes a LLSFIoT model consisting of three phases. Phase 1 registers the new devices with the central server and handover the essential credentials to the device. Phase 2 performs mutual authentication between server and the device. Phase 3 proposed a LDS algorithm that offers confidentiality and integrity to data in transit. LDS works as a Feistel structure on 20 rounds of operation using ARX operations: addition, rotation, and XOR. The LDS is evaluated by performing cryptanalysis using diffusion property. Different sets of rotation constants are used to find mean and standard deviation. This research work concludes that LDS works well with constant value as $x = 7$ and $y = 3$ with maximum mean and minimum standard deviation assuring that single change in input will affect more bits in output.

Data Availability

There is no dataset involved in this research paper.

Conflicts of Interest

The authors declare no conflict of interest.

Acknowledgments

This research was supported by the Taif University Research Supporting Project number (TURSP-2020/239), Taif University, Taif, Saudi Arabia.

References

[1] C. Cheng, R. Lu, A. Petzoldt, and T. Takagi, “Securing the Internet of Things in a quantum world,” *IEEE Communications Magazine*, vol. 55, no. 2, pp. 116–120, 2017.

[2] W. Feng, Y. Qin, S. Zhao, and D. Feng, “AAoT: Lightweight attestation and authentication of low-resource things in IoT and CPS,” *Computer Networks*, vol. 134, pp. 167–182, 2018.

[3] J. S. Silva, P. Zhang, T. Pering, F. Boavida, T. Hara, and N. C. Liebau, “People-Centric Internet of Things,” *IEEE Communications Magazine*, vol. 55, no. 2, pp. 18-19, 2017.

[4] C. Verikoukis, R. Minerva, M. Guizani, S. K. Datta, Y. K. Chen, and H. A. Muller, “Internet of Things: part 2,” *IEEE Communications Magazine*, vol. 55, no. 2, pp. 114-115, 2017.

[5] K. R. Choo, S. Gritzalis, and J. H. Park, “Cryptographic solutions for industrial internet-of-things: research challenges and opportunities,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3567–3569, 2018.

[6] L. Ledwaba, G. P. Hancke, H. S. Venter, and S. J. Isaac, “Performance costs of software cryptography in securing new-generation internet of energy endpoint devices,” *IEEE Access*, vol. 6, pp. 9303–9323, 2018.

[7] Y. Harbi, Z. Aliouat, S. Harous, A. Bentaleb, and A. Refoufi, “A review of security in Internet of Things,” *Wireless Personal Communications*, vol. 108, no. 1, pp. 325–344, 2019.

[8] Hong, “Internet of Things is now in sync with real life,” 2016, https://news. http://samsung.com/global/samsung-shows-that-the-internet-of-things-is-now-in-sync-with-real-life.

[9] W. H. Hassan, “Current research on Internet of Things (IoT) security: a survey,” *Computer Networks*, vol. 148, pp. 283–294, 2019.

[10] S. S. Dhanda, B. Singh, and P. Jindal, “Lightweight cryptography: A solution to secure IoT,” *Wireless Personal Communications*, vol. 112, no. 3, pp. 1947–1980, 2020.

[11] Y. Yang, H. Peng, L. Li, and X. Niu, “General theory of security and a study case in internet of things,” *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 592–600, 2017.

[12] P. I. Radoglou Grammatikis, P. G. Sarigiannidis, and I. D. Moscholios, “Securing the Internet of Things: challenges, threats and solutions,” *Internet of Things*, vol. 5, pp. 41–70, 2019.

[13] N. Miloslavskaya and A. Tolstoy, “Internet of things: information security challenges and solutions,” *Cluster Computing*, vol. 22, no. 1, pp. 103–119, 2019.

[14] V. Adat and B. B. Gupta, “Security in Internet of Things: issues, challenges, taxonomy, and architecture,” *Telecommunication Systems*, vol. 67, no. 3, pp. 423–441, 2018.

[15] A. Jan, S. A. Parah, B. A. Malik, and M. Rashid, “Secure data transmission in IoTs based on CLoG edge detection,” *Future Generation Computer Systems*, vol. 121, pp. 59–73, 2021.

[16] M. Rashid and U. I. Wani, “Role of fog computing platform in analytics of Internet of Things- issues, challenges and opportunities,” *Fog, Edge, and Pervasive Computing in Intelligent IoT Driven Applications*, vol. 1, pp. 209–220, 2020.

[17] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, “Consolidate IoT edge computing with lightweight virtualization,” *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[18] H. Hellaoui, M. Koudil, and A. Bouabdallah, “Energy-efficient mechanisms in security of the internet of things: a survey,” *Computer Networks*, vol. 127, pp. 173–189, 2017.

[19] D. He, R. Ye, S. Chan, M. Guizani, and Y. Xu, “Privacy in the Internet of Things for smart healthcare,” *IEEE Communications Magazine*, vol. 56, no. 4, pp. 38–44, 2018.

[20] N. Karthik and V. S. Ananthanarayana, “Trust based data gathering in wireless sensor network,” *Wireless Personal Communications*, vol. 108, no. 3, pp. 1697–1717, 2019.

[21] P. Derbez and P. A. Fouque, “Exhausting Demirci-Selçuk meet-in-the-middle attacks against reduced-round AES,” in *International Workshop on Fast Software Encryption. FSE 2013*, S. Moriai, Ed., vol. 8424 of Lecture Notes in Computer Science, pp. 541–560, Springer, Berlin, Heidelberg, 2014.

[22] W. Li, H. Song, and F. Zeng, “Policy-based secure and trustworthy sensing for internet of things in smart cities,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 716–723, 2018.

[23] Y. Kawamoto, H. Nishiyama, N. Kato, Y. Shimizu, A. Takahara, and T. Jiang, “Effectively collecting data for the location-based authentication in internet of things,” *IEEE Systems Journal*, vol. 11, no. 3, pp. 1403–1411, 2017.

[24] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu, “Security vulnerabilities of internet of things: a case study of the smart plug system,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1899–1909, 2017.

[25] C. Blondeau and K. Nyberg, “Links Between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities,” in *Advances in Cryptology – EUROCRYPT 2014. EUROCRYPT 2014*, P. Q. Nguyen and E. Oswald, Eds., vol. 8441 of Lecture Notes in Computer Science, pp. 165–182, Springer, Berlin, Heidelberg, 2014.

[26] R. Beaulieu, S. T. Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, “The SIMON and SPECK lightweight block ciphers,” in *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6, San Francisco, CA, USA, 2015.

[27] A. V. Duka and B. Genge, “Implementation of SIMON and SPECK lightweight block ciphers on programmable logic controllers,” in *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, pp. 1–6, Tirgu Mures, Romania, 2017.

[28] N. Wang, X. Wang, K. Jia, and J. Zhao, “Differential attacks on reduced SIMON versions with dynamic key-guessing techniques,” *SCIENCE CHINA Information Sciences*, vol. 61, pp. 1–3, 2018.

[29] A. Garcia-de-Prado, G. Ortiz, and J. Boubeta-Puig, “COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things,” *Expert Systems with Applications*, vol. 85, pp. 231–248, 2017.

[30] V. Grosso, G. Leurent, F. X. Standaert, and K. Varıcı, “LS-designs: bitslice encryption for efficient masked software implementations,” in *Fast Software Encryption. FSE 2014*, C. Cid and C. Rechberger, Eds., vol. 8540 of Lecture Notes in Computer Science, pp. 18–37, Springer, Berlin, Heidelberg, 2014.