Engineering and Technology Journal
Journal homepage: https://etj.uotechnology.edu.iq

Proposal Framework to Light Weight Cryptography Primitives

Mustafa M. Abd Zaid, Soukaena Hassan

Computer Sciences Dept., University of Technology-Iraq, Alsina'a street, 10066 Baghdad, Iraq.
*Corresponding author Email: mustafamajeed2014@gmail.com

HIGHLIGHTS

- Light weighting encryption algorithm is the best solution for IoT networks
- All cryptography primitives (block, stream, hashing, asymmetric) must be light weighted.
- Lightweight importance in time will decrease and complexity and randomness still same.
- Lightweights encryption save energy and computation consuming of smart devices.

ARTICLE INFO

Handling editor: Rana F. Ghani

Keywords:
Lightweight
Cryptography
LWAES-128
LWRC4
LWRSA
LWSHA-256
NIST

ABSTRACT

Due to manufacturing cost and portability limitations, the computing power, storage capacity, and energy of the Internet of Things (IoT) hardware are still slowly developing. From above, the proposed security system based on encryption must consider the resources, time, memory used, and the lifespan of related sensors. In addition, some applications need simple encryption, especially after the emergence of IoT and the Web of Things (WoT). Providing solutions suitable for resource-constrained devices can be achieved by using lightweight cryptography. In this paper, building a framework that includes proposals for producing lightweight security algorithms for cryptography primitives was highly recommended. For the block cipher, some suggestions have been applied to an example of block encryption, Advance Encryption Standard 128 (AES-128), to produce lightweight AES-128. For lightweight stream cipher, the system applied the proposals on Ronald Rivest Cryptography algorithms (RC4). Rivest–Shamir–Adleman (RSA) algorithm is used to produce a lightweight asymmetric cipher by key partition and using the Chinese Remainder Theorem (CRT) in the decryption process to produce a lightweight RSA algorithm. Several proposals have been used for hash functions, the most important of which is reducing the number of rounds and simplifying the functions in SHA-256. Depending on the proposed framework, all the produced lightweight algorithms passed the National Institute of Standards and Technology (NIST) statistical tests for test randomness. The produced algorithms showed better processing time than standard algorithms, less memory usage for a lightweight version of each standard algorithm, and higher throughput than standard algorithms.

1. Introduction

The increasing number of people who use internet-accessed devices, especially those with the high enhancement of artificial intelligence, such as smartphones, brings more technical difficulties regarding many issues such as the encryption and decryption processes [1, 2]. LWC (Light Wight Cryptography) is a research field that has developed in recent years to offer such cryptographic security characteristics [3, 4]. LWC works with the constraint potentiality to devices, especially in terms of energy consumption, communication bandwidth, execution time, and RAM size. LWC can define as a trade-off between cryptography and lightweight [5]. Many cryptographers suggest such lightweight characteristics in a lightweight block cipher, stream cipher, specially developed hash functions, and one-pass authenticated encryption [6]. The efficient implementation of Radio-Frequency Identification (RFID) tags, sensors in wireless sensor networks and more specifically, in small to medium internet-enabled appliance that expected to flood the market especially in the IoT era [1, 7]. LWC is affected by three main factors; cost, security [8, 9], and efficiency. The lightweight cipher-based standardization has not been applied in the market; however, ordinary or extraordinary (novel) designs of LWC should consider those practical factors [8]. This paper handles a proposed framework of the primitives regarding the LWC and generates strategies for employing the standardization of LWC algorithms in the current work.

2. Related work

This section discusses similar work that deals with Lightweight cryptography primitives algorithms

http://doi.org/10.30684/etj.v40i4.1679
Received 15 April 2020; Accepted 3 June 2020; Available online 25 April 2022
2412-0758/University of Technology-Iraq, Baghdad, Iraq

2.1 Aes lightweight related work

In 2018, Pratthiba et al. [7], designing of an LWC and securing a suitable IoT-based application substitution box (S-box) were the significant points targeted by the present work. The 4×4 S-box built the structure was sub-designed into two Galois Filed GF-based finite fields (FFs); (2^4) and $((2^2)^2)$. The FF-constructed S-box was developed recognizing inversional multi-processors (IMPs) and follow-up steps of affine transformation (AT). The IMP-structural-enhanced versions performed in the $((2^2)^2)$-based GFs. However, (2^4) -constructed GFs are considered the main fields for the induction of the AT. The (2^4)-$((2^2)^2)$ generated isomorphic mapping maintained in the (2^4)-related primitive elements. It was noticed that sub-pipelining was enhanced by the designed combinational architecture of the finite field S-box.

2.2 Rsa lightweight related work

In 2017, Sahu et al. [10] presented an RSA-algorithm-based system with increased potential security. The security highlight proposed is the standard RSA-algorithm-based end of "n" and insertion of a new number f in the place of n—this replacement is used in both private and public keys. Mathematical factorization attacks are prone to the RSA algorithm, and eliminating n with f makes the process very hard to factorize it and get the original numbers, i.e., p and q. Despite a slight increase in time complexity, this modification makes the algorithm more secure.

2.3 Rc4 lightweight related work

In 2017, Maity et al. [11] proposed a lightweight stream cipher algorithm providing some security levels such as Grain-128. Original RC4 and other types of stream ciphers regarding short-timed session-based wireless applications inserting Lightweight Pseudo-Random Generation Algorithm (LPRGA) instead of RC4. Original PRGA shows a better NIST-based output keystream than that from PRGA for less than 30,000bits. Related keystream sequences enhance Wireless Sensors Networks (WSNs), almost all mobile transactions, and ad hoc networks, plus better security levels. However, a longer processing time than those from RC4-based PRGA was recognized. Interestingly, the cost was less in the proposed system with high-quality performance levels of hardware-constructed wireless devices.

2.4 Sha-256 lightweight related work

In 2016, Bussi et al. [8] proposed a lightweight hash, the simple fundamental thought of LWC, satisfied with 'Neeva-hash' providing a unique security requirement at a collision resistance of 2^{112}. Neeva-hash depends on friendly permutation with sponge cycle mode, which gives required security and extraordinary efficiency in RFID technology and can utilize in many applications. Some factors such as a heuristic proof of differential characteristics, bit variance test, and near-collision resistant test were used to evaluate the system.

3. General description of Proposal

Lightweight algorithms were hypothesized using cryptographic primitives as examples for the proposed framework. The standard algorithms were altered to meet the requirements of constrained devices. A general description of a proposed framework was discussed in this section. An example for each type of cryptography primitives was given to convert it to a lightweight version. This paper proposes a framework to produce lightweight algorithms needed in a constrained environment for all primitives of cryptography. In the evaluation step, statistical tests are done to check if the ciphertext will be random. The statistical tests evaluation step is not enough in deciding if the algorithm is lightweight or not. Other important evaluations must be done, such as measuring time and memory used. The proposed framework is shown in Figure 1.

3.1 Proposal to Lightweight symmetric block cipher (AES)

To overcome the issue of computational overhead and high calculation, we break down AES (Advanced Encryption Standard) and modify it to diminish the algorithm's calculation. So we designed and implemented a lightweight AES-128 algorithm. The essential plan is to alter AES-128 to give better security to the information and less calculation. The modified AES-128 algorithms acclimate to giving the best encryption and decryption speed. The length of the key and block is specified as a standard AES algorithm. To defeat the issue of high computation, we have proposed an alternative lightweight design for both forward and inverse MixColumns operation required in the AES implementation and integrated it with the AddRound key step to reduce the memory used. A modified proposal for ShiftRows step by combining (ShiftRows and ShiftColumns). The three steps used for lightweight AES algorithms included in LWAES algorithms are:

- SubBytes
- Modified ShiftRows
- Mix columns & AddRoundKey

Substitution bytes stay unaffected as it is in the standard AES.

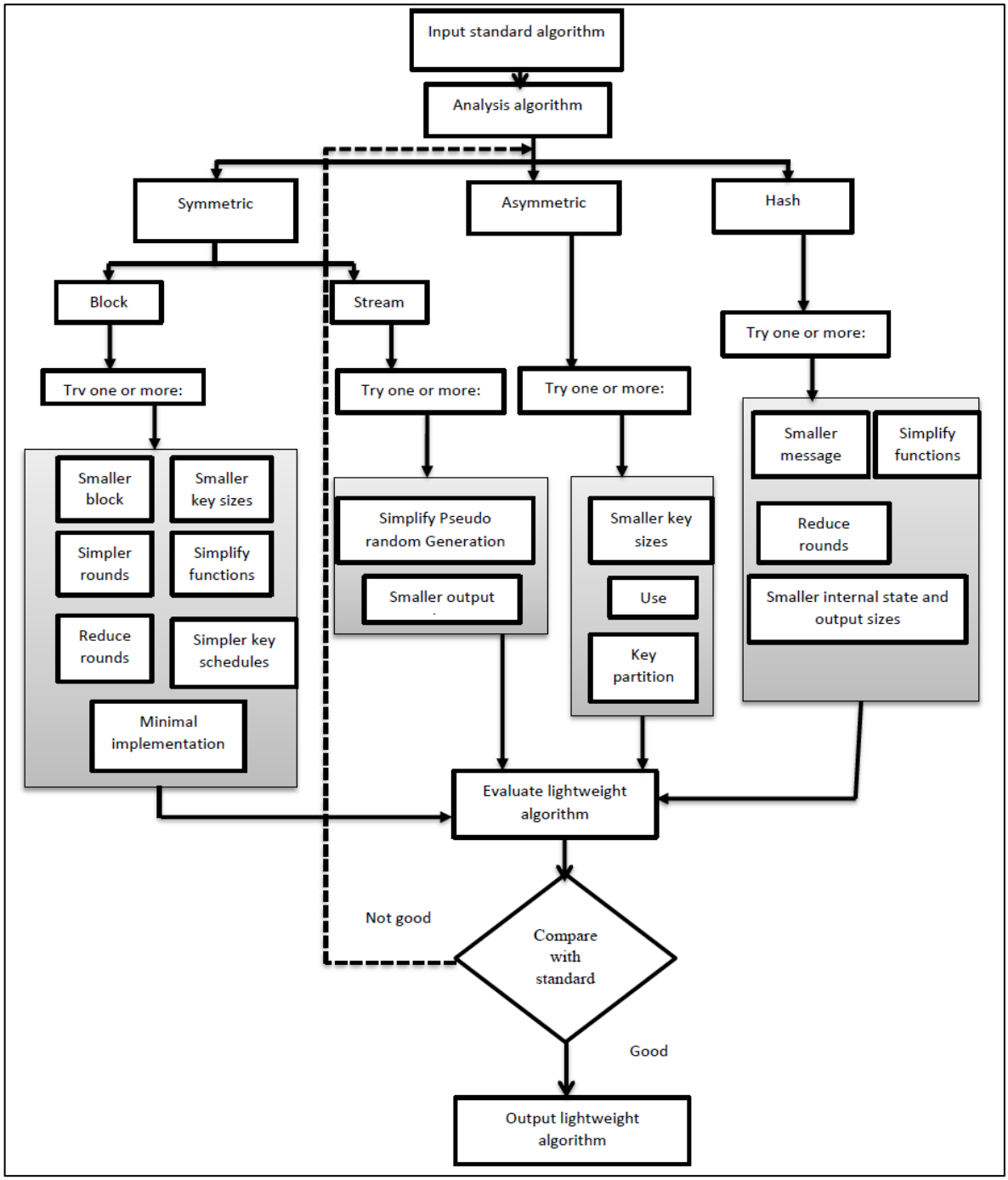

Figure 1: The proposed framework of lightweight cryptography primitives

3.1.1 Modified shiftrows

In the modified ShiftRow cyclical moving process for the key matrix in each row and column, the arrays' rows and columns are rotated by a specific number of byte positions to expand diffusion more than in the standard AES algorithm. Figure2-shows modified ShiftRows operation.

3.1.2 Mixcolumns & addroundkey

In this step, we combine the modified MixColumns operation with the AddRoundKey operation. The bytes of the column are mapped into another esteem in a MixColumn step, which is an element of every one of the four bytes in that column. The following matrix is shown in Figure 3- transformations characterized for each component the amount of the item matrix of one column and one-row parts.

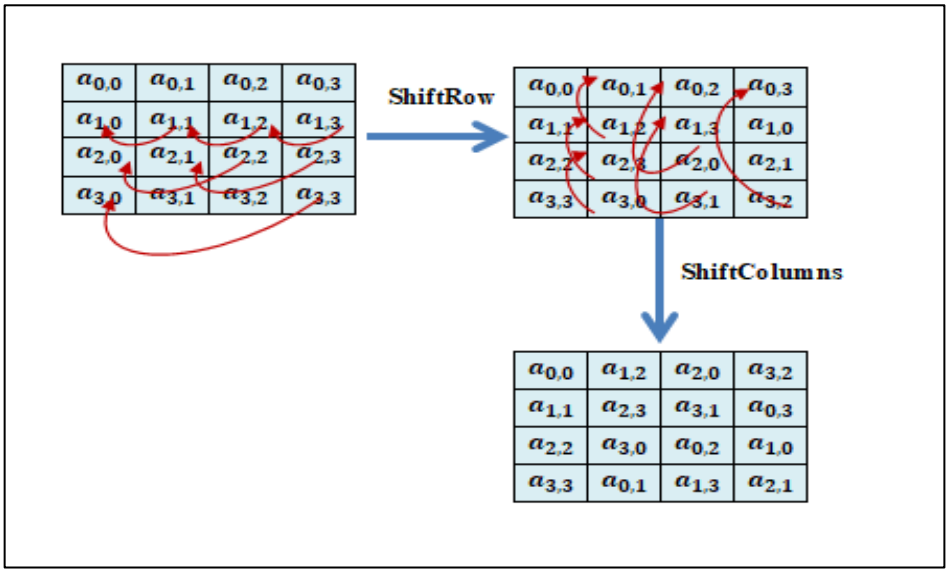

Figure 2: Modified ShiftRows

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 3: MixColumn Matrix

For this situation, the multiplications and individual additions are performed in GF (2^8). The equation above can be implemented more efficiently to eliminate the shifts and conditional XORs.Utilizing the identity $\{03\} \oplus y = (\{02\} \oplus y)\, y$, now MixColumns equation can modify as follows:

$$\text{tmp} = col_{0,j} \oplus col_{1,j} \oplus col_{2,j} \oplus col_{3,j} \quad (1)$$

$$col'_{0,j} = col_{0,j} \oplus \text{tmp} \oplus [(col_{0,j} \oplus col_{1,j}) * 2] \quad (2)$$

$$col'_{1,j} = col_{1,j} \oplus \text{tmp} \oplus [(col_{1,j} \oplus col_{2,j}) * 2] \quad (3)$$

$$col'_{2,j} = col_{2,j} \oplus \text{tmp} \oplus [(col_{2,j} \oplus col_{3,j}) * 2] \quad (4)$$

$$col'_{3,j} = col_{3,j} \oplus \text{tmp} \oplus [(col_{3,j} \oplus col_{0,j}) * 2] \quad (5)$$

We can combine modified MixColumns with the AddRound key to get fewer repetitions. In the decryption process, the inverse MixColumn transformation is used in the decryption process. The sum of products of components of one row and one column forms each component in the product matrix. In this situation, the multiplications and individual additions are generated in GF (2^8). The MixColumns on an individual column-based transformation j ($0 \leq j \leq 3$) of state can be defined as:

Inverse MixColumn equation is formulated to simplify as:

$$\text{tmp} = 09 * (col_{0,j} \oplus col_{1,j} \oplus col_{2,j} \oplus col_{3,j})$$
$$col'_{0,j} = col_{0,j} \oplus \text{tmp} \oplus [(col_{0,j} \oplus col_{2,j}) * 2] * 2 \oplus (col_{0,j} \oplus col_{1,j}) * 2]$$
$$col'_{1,j} = col_{1,j} \oplus \text{tmp} \oplus [(col_{1,j} \oplus col_{3,j}) * 2] \oplus (col_{1,j} \oplus col_{2,j}) * 2]$$
$$col'_{2,j} = col_{2,j} \oplus \text{tmp} \oplus [(col_{0,j} \oplus col_{2,j}) * 2] \oplus (col_{2,j} \oplus col_{3,j}) * 2]$$
$$col'_{3,j} = col_{3,j} \oplus \text{tmp} \oplus [(col_{1,j} \oplus col_{3,j}) * 2] \oplus (col_{3,j} \oplus col_{0,j}) * 2] \quad (6)$$

Finally, we combine the MixColumns operation with the AddRound Key in the same step to reduce the number of repetitions. The decoding method follows the same measures as encoding but uses opposite phase keys. The decryption of the suggested method functions by applying the opposite of all four activities of the earlier mentioned. The InvSubBytes (Inverse SubBytes) step is done on the received data by applying the same steps on the original AES-128, then modified InvShiftRows (Inverse ShiftRows and ShiftColumns) so that each data row returns to its original position. Then InvMixColumns (Inverse MixColumns) step is done. Finally, the result is XORed with the next round's generated key. Notice that the key will be in the inverse mode. In other words, it starts with key[10] and ends with key[0].

3.2 Proposal for Lightweight symmetric stream cipher(RC4)

We produce an efficient stream cipher algorithm that is lightweight compared to the original RC4. The proposed algorithm brings down the cost of computational overhead compared with the conventional stream cipher like RC4. The suggested lightweight algorithm is sufficiently secure for utilization in many low-term wireless application sessions. For the creation of the random initial permutation S, utilizing the algorithm key Stream Algorithm (KSA) (first algorithm) belongs to RC4, but we also supplant the PRGA of RC4 new PRGA (lightweight PRGA). Lightweight PRGA is utilized for keystream creation from the entry permutation (S_{box}) result from Key Stream Algorithm (KSA). The keystreams generated from this LPRGA will be XORed with the successive bits of a plaintext message to produce the required ciphertext message.

3.3 Proposal for Lightweight symmetric stream cipher (RSA)

In this sub-section, we propose a lightweight RSA (LWRSA) algorithm. The proposed scheme includes improving the RSA technique by proposing a technique that has speed enhancement on the RSA key generation/decryption sides. A new component."s_n" was added to increase the complexity of the RSA algorithm. So, the key generation time must be decreased, and the analytical difficulty of the variable "N" must be increased because of the presence of three prime numbers rather than two.

3.4 Proposal for Lightweight hash function (SHA-256)

In this section, a lightweight hash suggests for the SHA-256 algorithm. Many lightweight cryptographic algorithms have been developed, and also existing algorithms are modified in terms of resource constraint environments. The proposed scheme is based on SHA-256, with different structures and some modifications inside the functions. In addition, full adder arrays (FA's) were used to increase the complexity of the output hash. Figure 4 - shows the proposed LWSHA-256.

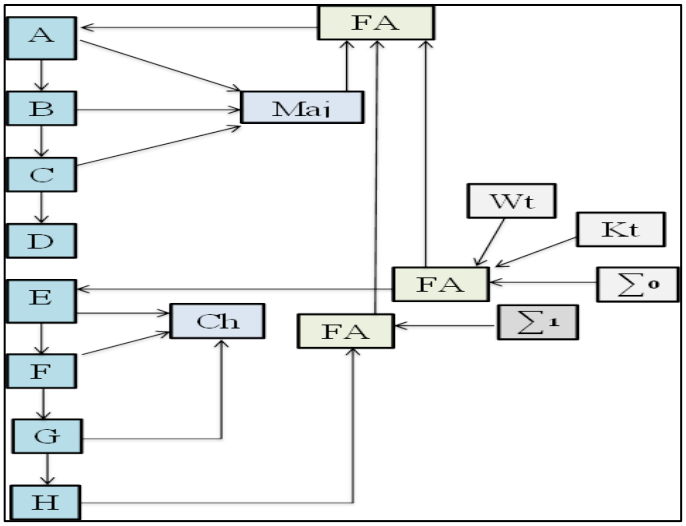

Figure 4: Proposed LWSHA-256

- Reducing the number of steps to 24. Hence, the message expansion becomes

$$Wj = \begin{cases} Mj & for\ 0 \le j < 16 \\ \sigma 1(Wj-2) \oplus Wj-7 \oplus \sigma 0(Wj-15) \oplus Wi-16 & for\ 16 \le j < 24 \end{cases} \quad (7)$$

- Propose SHA-256 uses only 24 sequences of sixty-four constant, K_0^{256}, K_1^{256},… K_{24}^{256}
- Initialize the eight acting factors, A, B, C, D, E, F, G, and H, with the $(j-1)^{st}$ hash.
- Use the same functions in SHA-256 with some modifications to it

Using Ch and Maj as an alternative of SHA1

$$Ch(a,b,c) = (c\hat{}(a\&(b\hat{}c)) \quad (8)$$

$$Maj(a,b,c) = (a \,\&\, b) \mid (\, c \,\&(a|b)) \tag{9}$$

Use Σ1 and Σ0 as in standard SHA-256

$$\sum_0^{256} a \; = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a) \tag{10}$$

$$\sum_1^{256} a \; = ROTR^6(a) \oplus ROTR^{11}(a) \oplus ROTR^{25}(a) \tag{11}$$

Modify α1 and α0 by removing Shift right

$$\sigma_0^{256}(a) = ROTR^7(a) \oplus ROTR^{18}(a) \oplus ROTR^3(a) \tag{12}$$

$$\sigma_1^{256}(a) = ROTR^{17}(a) \oplus ROTR^{19}(a) \oplus ROTR^{10}(a) \tag{13}$$

4. Experiments and Results

In the networking world, more and more internet facilities will connect devices rather than people together. Many of these devices have powerful processors and are fully capable of using a cryptographic algorithm that looks similar to those used in desktop PC's. While other connected devices use tremendously low-power microcontrollers, which can only afford fractions of their power percentage to afford security, thus using similar cryptography algorithms may cause very high-power consumption and incur latency. Therefore, several tests are run on each of the encrypted test cases to ensure that they are strong enough to prevent hacking by ensuring the randomness of its characters instead of the original data. The execution and results of the proposed system will be explained in detail in this section.

4.1 Encryption keys test

We applied NIST statistical randomness tests for two sets of keys, and each set contains six keys. The first set consisted of random keys of size 128-bit, and the other set of keys was size 256-bit. Table1 shows the set of one of the keys used to encrypt AES128 and LWAES128.

Table 1: Two Sets Of Keys

Key number	Set1(128-bit)	Set2(256-bit)
1	MbQeThWmYq3t6w9z	SgVkXp2s5v8y/B?E(H+MbQeThWmZq3t6
2	(G+KbPeShVmYp3s6	aNdRgUkXn2r5u8x/A?D(G+KbPeShVmYq
3	A%D*G-KaPdSgVkXp	F)J@NcRfUjXnZr4u7x!A%D*G-KaPdSgV
4	7w!z%C*F-JaNdRgU	$B&E)H@McQfTjWnZq4t7w!z%C*F-JaNd
5	q3t6w9z$C&F)J@Nc	v8y/B?E(H+MbQeThWmYq3t6w9z$C&F)J
6	VmYq3t6w9z$B&E)H	PeShVmYq3t6w9z$B&E)H@McQfTjWnZr4

A fine random number produced was identified by the *p*-value. If the *p*-value was over 0.01, the procedure was recognized as a random process; otherwise, a non-random procedure was decided. The results of applying the NIST statistical tests on the keys showed the success of almost all tests. The best key applied in NIST statistical tests for set1 is (Key5), which revealed the best randomness. The system used different text, case files of different sizes (plaintext, first one was the short-text (704) bits named TX1, the second one was of medium size about (32768) bits called TX2, and the final one named TX3 which was about (65536) bits (TX3)) utilized for encryption in different variations of the algorithms.

4.2 Nist statistical tests

Statistical test results demonstrated successful output of the two tested algorithms showing adequate statistical tests with good randomness for all tested algorithms. Table 2- shows NIST tests applied to standard and lightweight algorithms using the key5 of the second set. In this sub-section, the statistical tests are used to measure the randomness of ciphertext for each three text files using key5 of set one. The results of statistical tests regarding the ciphertext of the standard and modified algorithms show the success of all tests (where p=0.01), as shown in Table 2, the tests applied to the three different plaintext files.

Table 2: Results of NIST Tests Applied standard and lightweight algorithms

Algorithm	File	Frequency test	Frequency within block	Run test	Longest run of ones in block	Discrete Fourier Transform	Serial Test
AES-128	Tx1	0.508	0.288	0.49	0.72	0.61	0.27
	Tx2	0.04	0.21	0.043	0.55	0.39	0.507
	Tx3	0.2	0.02	0.3	0.31	0.114	0.21
LWAES1-128	Tx1	0.377	0.48	0.15	0.84	0.06	0.85
	Tx2	0.18	0.75	0.25	0.64	0.03	0.58
	Tx3	0.4	0.35	0.31	0.408	0.646	0.69
LWAES2-128	Tx1	0.462	0.18	0.73	0.409	0.48	0.633
	Tx2	0.48	0.105	0.32	0.82	0.78	0.39
	Tx3	0.52	0.15	0.23	0.116	0.355	0.42
RC4	Tx1	0.76	0.85	0.82	0.99	0.23	0.072
	Tx2	0.68	0.9	0.33	0.28	0.93	0.42
	Tx3	0.69	0.67	0.12	0.75	0.62	0.92
LWRC4	Tx1	0.61	0.59	0.26	0.52	0.56	0.64
	Tx2	0.23	0.59	0.49	0.932	0.64	0.47
	Tx3	0.7	0.2	0.32	0.4	0.87	0.49
RSA	64bit	0.282	0.283	0.771	0.08	0.146	0.6
	80bit	0.447	0.07	0.786	0.514	0.245	0.152
	128bit	0.042	0.085	0.311	0.937	0.146	0.695
LWRSA	64 bit	0.949	0.825	0.146	0.455	0.771	0.778
	80 bit	0.527	0.276	0.761	0.535	0.146	0.1
	1024bit	0.569	0.558	0.562	0.84	0.771	0.9
SHA-256	Tx1	0.21	0.44	0.08	0.35	0.038	0.21
	Tx2	0.45	0.73	0.87	0.17	0.13	0.45
	Tx3	0.66	0.47	0.86	0.76	0.28	0.66
LWSHA256	Tx1	0.8	0.58	0.99	0.93	0.73	0.8
	Tx2	0.7	0.25	0.62	0.64	0.81	0.7
	Tx3	0.8	0.59	0.52	0.41	0.7	0.8

4.3 Other performance metrics of Lwaes

Once assured that everything was working fine, encryption time, decryption time, encryption throughput, decryption throughput, avalanche effect, memory used, and correlation coefficient between standard algorithms and Lightweight algorithms were done. As shown in Table3, the lightweight algorithm's encryption and decryption time are less than standard algorithms for the three files. This is evident by increasing file size. These values applied on TX1, TX2, and TX3 are presented in Table 3. The throughput for proposed lightweight algorithms is perfect for TX3 and better for TX2 and TX1 compared to standard algorithms. The avalanche effect for proposed algorithms in TX1 is the best among all tested files. The proposed algorithms show less memory used in comparison with standard algorithms.

Table 3: Comparing lightweight algorithms with standard algorithms

Algorithm	File	Encryption time(second)	Decryption time(second)	Encryption Throughput (KB/sec)	Decryption Throughput (KB/sec)	Avalanche Effect	Memory used	Correlation coefficient
AES-128	Tx1	0.0265	0.0286	3.24	3.004	0.503	32768	0.0303
	Tx2	0.42	0.43	9.52	9.3	0.43	32768	0.016
	Tx3	1.39	1.43	5.75	5.63	0.457	32768	0.0023
LWAES1-128	Tx1	0.0098	0.0202	8.76	4.25	0.662	28672	0.0357
	Tx2	0.157	0.356	25.47	11.23	0.431	28672	0.0002
	Tx3	0.49	1.15	16.32	6.95	0.464	28672	0.0023
LWAES2-128	Tx1	0.005	0.012	17.188	7.161	0.654	8192	0.037
	Tx2	0.098	0.207	40.81	19.32	0.419	8192	0.0096
	Tx3	0.301	0.647	26.57	12.36	0.459	8192	0.008
RC4	Tx1	0.00037	0.0019	232.27	45.231	0.00121	12288	0.016
	Tx2	0.0047	0.0052	851.06	769.23	0.2545	12288	0.0002
	Tx3	0.0116	0.0123	689	650	0.454	12288	0.00236
LWRC4	Tx1	0.00034	0.0004	253	214.85	0.529	4096	0.015
	Tx2	0.0022	0.0023	1818	1739	0.222	4096	0.0044
	Tx3	0.0067	0.0068	1194	1176	0.4615	4096	0.00492
RSA	64bit	1.47	0.088	0.095	0.084	0.514	24576	0.055
	80bit	1.62	0.113	0.1036	0.097	0.50	24576	0.062
	1024bit	2.9	0.147	0.119	1.07	0.5103	24576	0.049
LWRSA	64 bit	0.294	0.087	0.00298	2.68	0.53	12288	0.0197
	80 bit	0.39	0.076	0.0029	3.44	0.5	12288	0.0099
	1024bit	0.375	0.117	0.00307	41.69	0.518	12288	0.0127
SHA-256	Tx1	0.0031	-	30	-	0.654	-	-
	Tx2	0.04	-	39	-	0.6383	-	-
	Tx3	0.118	-	43	-	0.6383	-	-
LWSHA256	Tx1	0.002	-	44	-	0.66	-	-
	Tx2	0.028	-	55	-	0.637	-	-
	Tx3	0.0948	-	55	-	0.655	-	-

5. Conclusions

Cryptography systems never depend on an increase in the number of rounds, size of keys, and size of blocks to influence the randomness. Lightweight cryptography is dangerous if not built to consider all constraints as it must balance the objectives trend to memory, time, and throughput with security. Cryptographic primitives will force the lightweight to consider the way of light-weighting. LWAES provides good security with low time memory usage by modifying the number of rounds, ShiftRows, backward ShiftRows (inverse), and both forward and backward MixColumns (inverse) operations required in the AES. This is combined with an AddRound key operation to reduce the time of encryption and decryption operations. LWRSA has less memory usage compared to other asymmetric algorithms. It also provides speed enhancement in the RSA algorithm's key generation and decryption sides. On the key generation side, using three prime numbers rather than two primes provides N with the same length as the standard RSA but with fewer bits for prime numbers. On the other side, the speed of the decryption face has improved by utilizing the idea of CTR (Chinese remainder theorem). LWRC4 algorithm is efficient; in other words, it is more cost-effective than the standard RC4 and is faster. The generated output sequences of the proposed algorithm have passed the NIST suite of statistical tests. This makes LWRC4, to a great degree, appropriate for actualizing secure correspondence in a wide range of wireless applications like Wi-Fi Protocol Access (WPA), where devices are compelled by either cost, energy, or processing ability. LWSHA-256 reduces the number of rounds used in standard SHA-256, uses different forms for functions, and modifies the original scheme by inserting FAA in the scheme. This gave a different result compared to SHA-256. The results indicated that the proposed algorithm has a better avalanche effect than standard SHA-256. It has passed the NIST test suite and is faster than the traditional one. Therefore, the LWSHA-256 algorithm achieves most of the properties required for lightweight cryptography.

Algorithm 1- Proposed LWAES-128 encryption

Input: plaintext, key
Output: ciphertext

Start
Step1: For each plaintext block do
Step2: state=AddRoundKey (XOR between block and key[0])
Step3: for i=1 to 9 do //can reduce to 6 in some situations.
Step4: SubBytes(state)
Step5: apply modified ShiftRows (state) //see Fig.2.
Step6: apply modified MixColumns& AddRoundKey (state, key[i])
Step7: end for
Step8: SubBytes(state)
Step9: apply modified ShiftRows(state)
Step10: AddRoundKey (state, key[10])
Step11:endfor
End

Algorithm 2- Proposed LWAES-128 decryption

Input: Ciphertext, key
Output: Plaintext

Start
Step1: For each ciphertext block do
Step2: state=AddRoundKey (XOR between block and key[10])
Step3: for i=9 to 1 do //can reduce to 6 in some situations.
Step4: InvSubBytes (state)
Step5: apply modified InvShiftRows(state)
Step6: apply modified InvMixColumns & AddRoundKey (state, key[i])
Step7: end for
Step8: InvSubBytes(state)
Step9: apply modified InvShiftRows(state)
Step10: AddRoundKey (state, key[0])
Step11:end for
End

Algorithm 3 Proposed LWRC4 encryption/decryption

Input: planintext/ciphertext, key
Output: ciphertext/plaintext

Start
// initialize S_{box}
Step1: for i=0 to 255
Step2: $S_{box}[i]$=i
Step3: end for
// KSA
Step4: set j=0
Step5: for i=0 to 255
Step6: j=j+$S_{box}[i]$+key[I mod key_length] mod 255
Step7: swap ($S_{box}[i], S_{box}[j]$)
Step8:end for
//LPRGA
Step9: set j=255;i=0;t=0
Step10: For i = 0 to 255
Step11: t= (S_{box}[i]+ S_{box} [j]+j) mod 256
Step12: j=i
Step13:i=S_{box} [i]
Step14: S_{box} [j] =t
Step15: Key_STREAM=S_{box} [i] XOR S_{box} [j]
Step16: End for
Step17: Ciphertext/Plaintext=Plaintext/Ciphertext ⊕ Key
End

Algorithm 4 Proposed LWRSA encryption/decryption

Input: plaintext(Pm)/ciphertext(Cm), Prime numbers p_n, q_n and s_n
Output: ciphertext(Cm)/plaintext(Pm)

Start
// key generator
Step1: Calculate $N=p_n \times q_n \times s_n$.
Step2: φ(N) = (p_n -1)(q_n -1)(s_n-1).
Step3: Chooses e, 1 < e_x < φ (N) like that gcd (e_x, φ (N)) = 1
Step4: Finds d such that $e_x \times d$ = 1 mod φ (N).
Step5: Finds d_p such that $e_x \times d_p$ = 1 mod (p_n -1), d_q such that $e_x \times d_q$ = 1 mod (q_n -1).
Step6: Finds Q_{in} such that
// $q_n \times Q_{in}$ = 1 mod p_n. if $p_n > q_n$
// $p_n \times Q_{in}$ = 1 mod q_n. if $q_n > p_n$
//Public Key Ku=(e_x, N) and Private key Kr=(Q_{in}, d_p, d_q, p_n, q_n).
// **RSA encryption**
Step7: Cipher text Cm = Me^{e_x} Mod N
// **RSA decryption**
Step8: $M_a = C^{dp}$ mod p_n
Step9: $M_b = C^{dq}$ mod q_n
Step10: h= ($Q_{in} \times (M_a - M_b)$) mod p_n
Step11: Me=M_b+(h×q_n) =plaintext
End

Algorithm 5 Proposed LWSHA-256

Input: Message M(M)
Output: Hash digits (256 bits)

Start
Process:
Step1: For i = 1 to N (N = padded message block numbers)
Step2: starts the eight values; A, B, C, D, E, F, G, and H, with the $(j-1)^{st}$ hash value: Intermediate hash value (= the initial hash value when j = 1)
// Apply the SHA-256 function of compression to upgrade eight registers values
A, B, C, D, E, F, G, and H
Step3: Compute: Wj, $\sigma_0^{256}(a)$, $\sigma_1^{256}(a)$
Step3: For i=1 to nb-rounds do
Compute Ch(e,f , g), Maj(a,b, c), $\sum_1^{256} e$, $\sum_0^{256} a$
Compute:
s,c_in ←FA (h, $\sum_1^{256} e$, ch(e,f,g))
s1,c_in1← FA(k[i],w[i], $\sum_0^{256} a$))
h←g
g←f
f←e
e,a0← FA(s,c_in1, $\sum_0^{256} a$)
d←c
c←b
b←a
a←s1
EndFor
Step7: The jth intermediate hash value H(j) can compute as:
H_0^j=a+H_0^{j-1}
H_1^j=b+H_1^{j-1}
H_2^j=c+H_2^{j-1}
H_3^j=d+H_3^{j-1}
H_4^j=e+H_4^{j-1}
H_5^j=f+H_5^{j-1}
H_6^j=g+H_6^{j-1}
H_7^j=h+H_7^{j-1}
EndFor
Step9: result 256-bits message digest(hash) of the message M which is: $H_0^j, H_1^j, H_2^j, H_3^j, H_4^j, H_5^j, H_6^j, H_7^j$
End

Author contribution

All authors contributed equally to this work.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability statement

The data that support the findings of this study are available on request from the corresponding author.

Conflicts of interest

The authors declare that there is no conflict of interest.

References

[1] Omran, A. Performance Analysis of AES and LWAES Algorithms, Master, thesis, Iraqi Comm. Comput. Inform., 2018.

[2] O. Toshihiko, Lightweight Cryptography Applicable to Various IoT Devices, NEC Tech. J., 12 (2017) 67-71.

[3] R. García, I. A. Badillo, M. Sandoval, C. Feregrino, R. Cumplido, Acompact FPGA-based Processor for the Secure Hash Algorithm SHA-256, Comput. Electr. Eng., 40 (2014)194-202. https://doi.org/10.1016/j.compeleceng.2013.11.014

[4] Ashton, K. 'That Internet of Things, RFID Journal, New York, 22 (2009) 97-114.

[5] Höglund, R. Lightweight Message Authentication for the Internet of Things, Master's Thesis, School of Information and Communication Technology (ICT) KTH Royal Institute of Technology Stockholm, Sweden, 2014.

[6] Cryptrec Lightweight Cryptography Working Group, Lightweight Cryptography, Cryptographic Technol. Guideline, 2017.

[7] A. Prathiba , V. S. K. Bhaaskaran, Lightweight S-Box Architecture for Secure Internet of Things, Info., 9 (2018) 13 https://doi.org/10.3390/info9010013

[8] K. A. Bussi, Lightweight Hash Function, University of Delhi, INDIA, 2016.

[9] E.G. Ahmed, E. Shaaban, M. Hashem, Lightweight Mix Columns Implementation for AES, Proc. Int. Conf. Appl.info. comm., (2013) 1790-5109.

[10] J. Sahu, V. Singh, V. Sahu, C. Chopra, An Enhanced Version of RSA to Increase the Security, J. Netw. Commun. Emerg.Technol.,7 (2017)1-4.

[11] S. Maity, K. Sinha, B.P. Sinha, An Efficient Lightweight Stream Cipher Algorithm for Wireless Networks, IEEE Wirel. Comm. Network. Conf., (2017) 1-6. https://doi.org/10.1109/WCNC.2017.7925562