International Journal of Safety and Security Engineering
Vol. 14, No. 3, June, 2024, pp. 843-852
Journal homepage: http://iieta.org/journals/ijsse

Parallel Multi-core Implementation of the Optimized Speck Cipher

Ahmed Fanfakh[1*], Nihad Abduljalil[2], Ali Kadhum M. Al-Qurabat[3]

[1] Department of Computer Science, University of Babylon, Babylon 51002, Iraq
[2] Department of Air Conditioning and Refrigeration, University of Warith Al-Anbiyaa, Karbala 56001, Iraq
[3] Department of Cyber Security, College of Science, Al-Mustaqbal University, Babylon 51001, Iraq

Corresponding Author Email: ahmed.fanfakh@uobabylon.edu.iq

https://doi.org/10.18280/ijsse.140316

Received: 26 February 2024
Revised: 1 May 2024
Accepted: 8 May 2024
Available online: 24 June 2024

Keywords:
speck cryptography, parallel computing, multi-core CPU, rounds reduction

ABSTRACT

Lightweight cryptographic algorithms like Speck, which are a family of block ciphers developed by the US National Security Agency (NSA), have become popular because of their efficient performance and small operational size. This paper introduces the execution on a parallel multi-core processor of the optimized version of the Speck cipher. However, this proposition fulfils the increased demand for developing quick and ultra-lightweight ciphers. In this work, this is addressed by optimizing the speck128/128 cipher by reducing its number of rounds to five. The optimization is accomplished by adding the dynamic substitution layer to increase the randomness of the cipher, which allows us to reduce the speck rounds. We conducted tests such as statistical, randomness, and cryptanalysis tests for linear and differential attacks on the optimized speck. The security results show that the optimized speck overcomes the original speck security level. The conducted experiments show that the new version of the speck runs faster than the original one in terms of execution time and throughput. The parallel execution over a multicore processor is applied, and its speedup ratio is equal to 2.64 when it's compared to the parallel execution of the original speck. Different message sizes and thread configurations are used in this work. The sequential execution of both speck ciphers is computed in terms of execution time and throughput, and the acceleration ratio of the optimized speck in this case is equal to 2.63.

1. INTRODUCTION

Given the availability of resource limited devices, including those that are connected to Internet of Things (IoT), lightweight cryptography has become an important research topic. Furthermore, when it comes to these low-end devices with limited processing power and memory, implementation of any robust cryptosystems is often hampered due to their strong security [1].

Recently, a lot of attention has been focused on Speck; a lightweight block cipher because it operates efficiently in this context and can be easily implemented. In 2013, the United States' National Security Agency (NSA) introduced Speck cipher as an alternative to AES which was more affordable. These are classes of block ciphers that work on 32- or 64-bit words depending on the key sizes and use fixed number of cycles. For instance, if one needs secure communication but has devices having limitations in resources, then he or she will definitely choose Speck since its simplicity and efficacy appear to be very appealing [2].

However much beneficial Speck maybe but there is more need for enhancing its operations especially through making them more efficient so as to meet the growing demand for cryptographic algorithms. Parallel processing is a computational methodology utilized to partition the running of an algorithm for cryptography into more manageable tasks. These tasks are then executed concurrently on several computing resources, including multi-core central processing units (CPUs).

In this work, we proposed new configured speck cipher to increase its performance capabilities. The approach prioritizes the reduction of round count and is executed on multicore CPUs with the intention of enhancing performance. In addition, the parallel implementation was devised strategically to take advantage of the capabilities of the multi-core architecture of modern CPUs. By utilizing the CTR mode of operation, the proposed parallel implementation of Speck achieves a high degree of parallelism and enables efficient data encryption and decryption. This work examines the outcomes of the security analysis and assesses the performance of the parallel implementation in relation to the original Speck algorithm. It evaluates the implementation's efficacy in terms of throughput and speedup.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the related works. Section 3 describes the background of Speck encryption. Section 4 describes the proposed optimized Speck algorithm. Sections 5 demonstrates the parallel implementation of the optimized Speck and its original version on multi-core CPU. Section 6 validates and compares the security results of the proposed optimized Speck cipher. Section 7 presents the experimental results of sequential and parallel implementations. Section 8 concludes the proposed work and summarizes the obtained results.

2. RELATED WORKS

The two primary categories of encryption methods are symmetric and asymmetric. The symmetric-key strategy is favoured in practical implementations because it requires less computational complexity, memory usage, and resources than the asymmetric key scheme. Stream ciphers and block ciphers are the two main types of symmetric ciphers. The Counter mode (CTR) variation of AES is one of the most common implementations of this widely-used algorithm [3, 4]. The block cipher functions as a stream cipher here because the ciphering process is decoupled from the plaintext. To achieve the appropriate degree of security, however, block ciphers often need numerous rounds. Feistel or Substitution-Permutation Networks (FN or SPN) may form the basis goal of keeping the confusion and diffusion features intact by increasing the number of rounds. Several notable research endeavours have been dedicated to exploring innovative techniques and optimizations for reducing the number of rounds in Speck encryption, aiming to enhance its performance without compromising security. In previous research, researchers of Ren and Chen [5], and Gohr [6] utilized deep learning and cryptanalysis approaches on reduced Speck to achieve reductions to 11 rounds. In contrast, researchers of Sleem and Couturier [7], Yue and Wu [8] focused on reducing the number of rounds of Speck to 7 for lightweight cryptographic schemes, with the former proposing an ultra-lightweight cryptographic scheme for IoT and the latter presenting an improved neural differential distinguish model for the lightweight cipher Speck.

Cryptanalysis studies have been made in the state-of-the-art researches to determine the resistance of speck cipher when reducing its number of rounds. A new method for developing lightweight and universally applicable deep learning-aided differential distinguishers is presented by Liu et al. [9]. The work of differential cryptanalysis against the NSA block cipher SPECK32/64 demonstrates how traditional approaches in cryptanalysis have been advanced by the inclusion of deep learning techniques when the speck rounds are reduced.

While demonstrating the effectiveness of deep learning in cryptanalysis, the work of Deng et al. [10] break new ground by incorporating attention processes into differential cryptanalysis on SPECK. Their results outperform those of classic residual networks in terms of accuracy and interpretability, uncovering possible vulnerabilities in the process.

An enhanced method of differential-neural cryptanalysis for round-reduced is introduced by Zhang et al. [11]. They outperform DDT-based approaches in certain rounds after using a neural network to improve the accuracy of neural distinguishers. They accomplish realistic key recovery attacks by enhancing both classical differentials and neural distinguishers; this raises the attack threshold by two rounds, allowing assaults on up to 17 rounds.

By continually striving to reduce the number of rounds in Speck encryption, this work presents an innovative approach that significantly minimizes the number of Speck128/128 rounds to 5. The proposed work shows a strong design that is specifically made for the parallel architecture of multi-core CPUs.

3. SPECK CRYPTOGRAPHY

Speck, a symmetric key block cipher, is an algorithm that provides secure and efficient methods of encoding or decoding data. The lightweight cipher Speck was introduced by the National Security Agency (NSA) of the United States in 2013. It belongs to a family of ciphers named ARX (Addition/ Rotation/ XOR) which are renowned worldwide. Speck is a suitable encryption technique for many applications that require data integrity and confidentiality because it is very efficient and straightforward in its operation. Speck works on blocks of data with predetermined length using a single key for both encryption and decryption processes. Maximum quantity of data that can be processed in a single operation and level of security are directly affected by the block and key sizes.

In this work, a Speck cipher with a block size of 128 bits and a key size of 128 bits are used, ensuring strong cryptographic capabilities. The speck function in this configuration takes 32 rounds. The Speck round function can be broken down into three distinct operations: XOR, modulo addition, and rotation, as shown in Eq. (1).

$$Li+1 = ((Li \gg \alpha) \boxplus R) \oplus k_i \; Ri+1 = (Ri \ll \beta) \oplus Li+1 \tag{1}$$

In this case, *Li* and *Ri* denoted the left and right halves of the data at round *i*, respectively. k_i denotes the round key at round *i*, and *α* and *β* are the rotation constants. Figure 1 shows the CTR Speck round function.

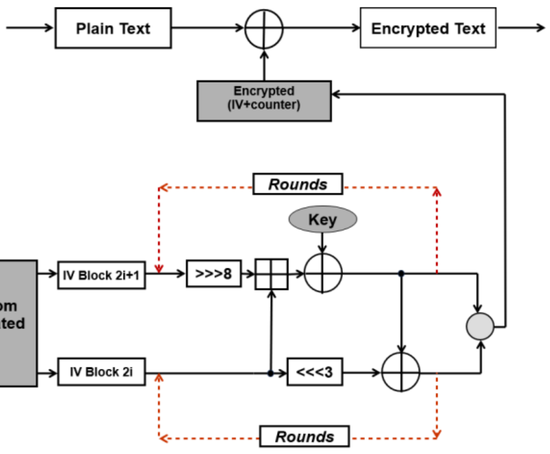


Figure 1. Speck encryption function

4. THE PROPOSED OPTIMIZED SPECK CIPHER

The proposed optimized SPECK algorithm employs a substitution table to reduce the number of SPECK rounds, thereby improving the efficiency of the encryption process.

We started with iterative testing and experimentation to see how various changes affected the cipher's efficiency and security. To assess the cryptographic properties, the fine-tune settings like the number of rounds, the size and structure of the S-box is used. Starting with a single Speck round, we then applied randomization and statistical testing as part of our optimization method and so on till reaching to the acceptable security level. Using this strategy, we were able to determine the optimization threshold at which we could achieve our desired degree of security. The *Speck-Encrypt* algorithm accomplishes this optimization by accepting the key schedule, input data, output buffer, initialization vector (IV), and substitution box (Sbox) as parameters. The optimized Speck encryption function, illustrated in Figure 2, incorporates the substitution box subsequent to every Speck round.

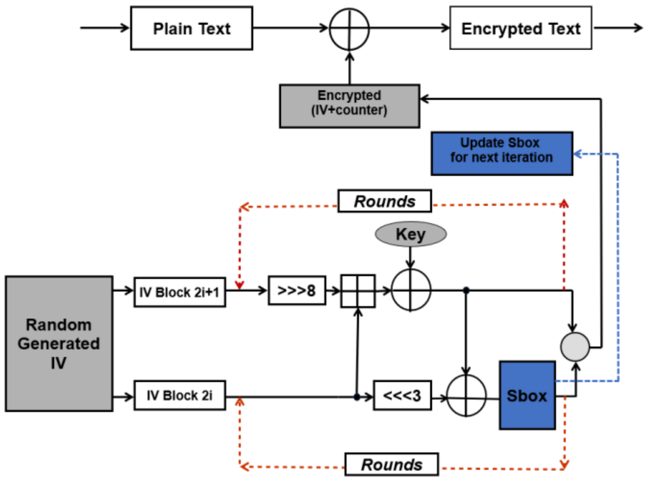


Figure 2. The proposed optimized speck encryption function

Algorithm 1 The Optimized SPECK Encryption Algorithm

```
1: procedure Speck-Encrypt (keySchedule, in,
                     out, iv, Sbox, messageSize)
2:  Sbox ← RC4(key_schedule, key_size)
3:  for i = 0 → message size do
4:  iv[0] ← iv[0] + i
5:  crypted_iv[0] ← iv[0]
6:  crypted_iv[1] ← iv[1]
7:  for j = 0 → 5 do
8:  SpeckRound(cryptediv[1], cryptediv[0],
    keySchedule[j])
9:  cryptediv[0] ← Substitution(cryptediv[0],
    Sbox)
10: out[i] ← crypted_iv[0] ⊕ in[i]
11: out[i + 1] ← crypted_iv[1] ⊕ in[i + 1]
12: Update Sbox ← ROTL(Sbox,
     crypted_iv[0]&8)
```

Algorithm 1 commences by initializing the *cryptediv* array containing the current version of IV, which is updated for every data block in order to guarantee the encryption process's uniqueness. Each byte of the *cryptediv* array undergoes a substitution operation via the substitution box (Sbox). This operation contributes to the improvement of the encryption's confusion and diffusion properties, thereby enhancing its security. Within the optimization process, five rounds of SPECK encryption are executed within the inner iteration, in contrast to the conventional 32 rounds. The significant reduction in computational expense facilitates encryption to execute at an accelerated pace, with only a marginal compromise on security. Each iteration of the encryption process involves calling the *SpeckRound* function, which applies the speck basic round function to the *cryptediv* array in order to update it. In Algorithm 2, the details of the speck round function are illustrated. Furthermore, the algorithm applies a bitwise left rotation to the solitary byte of the substitution box (*Sbox*) by utilizing the least significant bits of the byte in *cryptediv* that corresponds to the rotator (Sbox). This operation provides diffusion further and strengthens the encryption's cryptographic integrity.

Notably, the proposed optimization makes use of the widely implemented substitution concept in block ciphers, which has been implemented to enhance performance while maintaining a solid security foundation. Nevertheless, a comprehensive security analysis and performance evaluation of the optimized SPECK algorithm that has been proposed is imperative to ascertain its robustness against prospective attacks and its appropriateness for particular use cases. Indeed, the RC4 stream algorithm (Algorithm 3) will be utilized to create the substitution *Sbox*.

Algorithm 2 The round function of speck128/128

```
void SpeckRound(uint64_t* x1, uint64_t* x2, const
uint64_t key)
{
   *x1 = (*x1 >> 8) | (*x1 << (8 * sizeof(*x1) - 8));
   *x1 += *x2;
   *x1 ^= key;
   *x2 = (*x2 << 3) | (*x2 >> (8 * sizeof(*x2) - 3));
   *x2 ^= *x1;
}
```

Algorithm 3 RC4 Key Scheduling Algorithm (KSA)

```
1:  Procedure Rc4_KeySchedule (K, L)
2:  S ← Array of size 256 initialized with 0 to 255
3:  for idx1 ← 0 to 255 do
4:  S[idx1] ← idx1
5:  idx2 ← 0
6:  for idx1 ← 0 to 255 do
7:  idx2 ← (idx2 + S[idx1] + K [idx2 mod L]) mod
    256
8:  Swap S[idx1] and S[idx2]
9:  Return S
```

To create a single reliable *Sbox*, iterations are performed based on the previously generated DK. RC4 is employed because of its popularity and ease of implementation in both hardware and software. The dynamic *Sbox* is generated during the Key Setup Algorithm (KSA) step of RC4 setup.

5. THE PARALLEL EXECUTION OF THE OPTIMIZED SPECK OVER MULTICORE PROCESSOR

The parallel message-passing execution over a multicore processor for both the optimized and original Speck ciphers is demonstrated in this section. Algorithm 4 of the optimized speck has various input data, such as the key, the plain message (in), the substitution Sbox, the initial vector (IV), the block size, and the process ID to output the encrypted message.

The algorithm encrypts two blocks per iteration, and at the ends of the iteration, the Sbox is rotated. Algorithm 5 demonstrates the steps of the original speck algorithm over a multicore platform. Both algorithms are called using the main function that is presented in Algorithm 6 to perform MPI communication routines. The algorithm applies encryption using the Speck cipher in a parallelized manner using MPI (Message Passing Interface) through scatter and gather operations. The code employs non-blocking scatter and gathers operations (MPI_Iscatter and MPI_Igather), which overlap communication and computation. This allows the encryption process to proceed on each process while data is being exchanged, maximizing computational efficiency. Moreover, the unique index of the block, *taskid ∗ blocksize*, is used as a counter (CTR) that updates the first block of the initial vector.

Algorithm 4 The Optimized SPECK Encryption Function for Multi-core CPU

```
void SpeckEncrypt (uint64_t∗ keySchedule, uint64_t∗ in,
uint64_t∗ out, uint64_t∗ IV, uchar∗ Sbox, int blocksize, int
taskid)
{
   uint64_t crypted_iv[2];
   uint64_t iv_block[2];
   uchar∗ tt; int k = taskid∗blocksize;
 for(int i=0; i<blocksize; i+=2)
 {
     IV[0] + = k;
     crypted_iv[0] = IV[0];
     crypted_iv[1] = IV[1];
     for(int j=0; j<5; j++)
     {
        SpeckRound(&crypted_iv[1], &crypted_iv[0],
        &keSchedule[j]);
        tt = (uchar∗)&crypted_iv[0];
        tt[0] = Sbox[tt[0]];
        tt[1] = Sbox[tt[1]];
        tt[2] = Sbox[tt[2]];
        tt[3] = Sbox[tt[3]];
        tt[4] = Sbox[tt[4]];
        tt[5] = Sbox[tt[5]];
        tt[6] = Sbox[tt[6]];
        tt[7] = Sbox[tt[7]];
      }
    out[i] = crypted_iv[0] ^ in[i];
    out[i+1] = crypted_iv[1] ^ in[i+1];
    rotl8(Sbox[tt[0]],  tt[7]&8 );  //updating Sbox
    k++;
  }
}
```

Algorithm 6 The proposed MPI Speck Encryption Function for Multi-core CPU

```
void speck_ctr_encrypt(uint64_t∗  key_schedule,
       uint64_t∗ in, uint64_t∗ out,
        uint64_t∗ iv, uchar∗ Sbox, int n)
{
    int taskid, blocksize, numtasks;
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Comm_size(MPI_COMM_WORLD,
&numtasks);
    blocksize = n / numtasks;
    uint64_t∗ local_in = (uint64_t∗)
    malloc(blocksize * sizeof(uint64_t));
    uint64_t∗ local_out = (uint64_t∗)malloc
                      (blocksize * sizeof(uint64_t));
    MPI_Request scatterRequest, gatherRequest;
    MPI_Datatype datatype = MPI_UINT64_T;
    MPI_Datatype blocktype;
    MPI_Type_contiguous(blocksize, datatype,
&blocktype);
    MPI_Type_commit(&blocktype);
    MPI_Iscatter(in, blocksize, datatype, local_in,
                    blocksize, datatype, 0,
    MPI_COMM_WORLD, &scatterRequest);
    speck_encrypt(key_schedule, local_in, local_out,
                     iv, Sbox, blocksize, taskid);
    MPI_Igather(local_out, blocksize, datatype, out,
                    blocksize, datatype, 0,
    MPI_COMM_WORLD, &gatherRequest);
    MPI_Wait(&scatterRequest,
                MPI_STATUS_IGNORE);
    MPI_Wait(&gatherRequest,
                MPI_STATUS_IGNORE);
    MPI_Type_free(&blocktype);
}
```

Algorithm 5 The Original SPECK Encryption Function for Multi-core CPU

```
void SpeckEncrypt (uint64_t∗ keySchedule, uint64_t∗ in,
uint64_t∗ out, uint64_t∗ IV, uchar∗ Sbox, int blocksize, int
taskid)
{
   uint64_t crypted_iv[2];
   uint64_t iv_block[2];
   int k=taskid∗blocksize;
 for(int i=0; i<blocksize; i+=2)
 {
     IV[0] += k;
     crypted_iv[0] = IV[0];
     crypted_iv[1] = IV[1];
     for(int j=0; j<32; j++)
     {
        SpeckRound(&crypted_iv[1], &crypted_iv[0],
        &keySchedule[j]);
     }
   out[i] = crypted_iv[0] ^ in[i];
   out[i+1] = crypted_iv[1] ^ in[i+1];
    k++;
 }
}
```

6. ANALYSIS OF THE SECURITY RESULTS

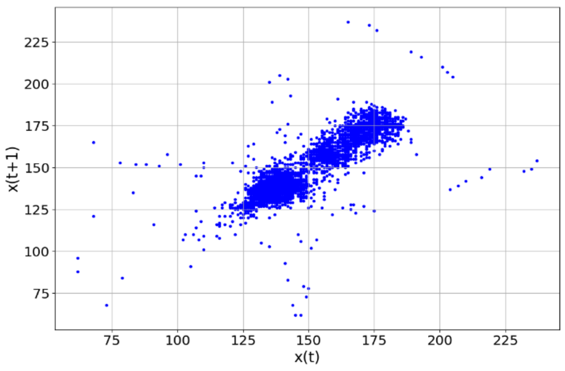
A proposed encryption scheme's security and safety are evaluated using well established techniques including statistical, linear, differential, or brute force attacks [12, 13]. Here, we conduct rigorous tests to demonstrate the security of the proposed optimized Speck encryption. While the proposed encryption system may be used for any kind of data, the results for multimedia content are.

6.1 Statistical analysis tests

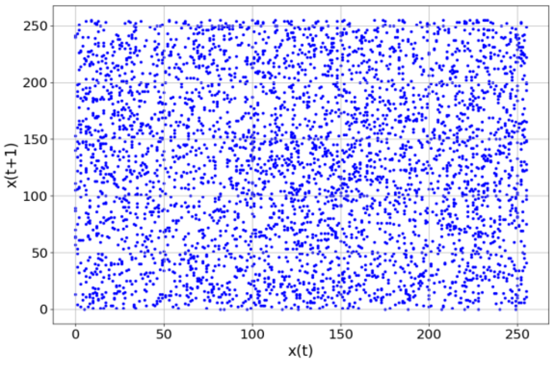
Randomness and uniformity are two features a cipher must have in order to be regarded as safe against statistical assaults [14, 15]. The following statistical integrity tests are carried out to evaluate the level of randomness: Analysis of the entropy of the data, the histogram of the plain and encrypted messages, the correlation between the original and encrypted messages, and the Probability Density Function (PDF).

6.1.1 Uniform analysis test

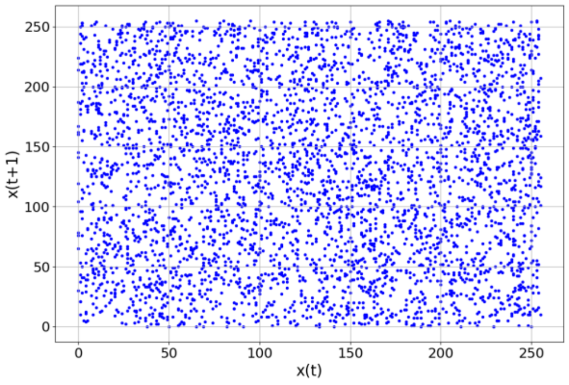
The most important test is the uniformity of the encrypted message's probability density function (PDF). The possibility of seeing any given symbol in the resulting ciphertext is about 1/n, where n is the total number of symbols. In Figure 3, the original PDF and the encrypted messages using both speck versions are shown. For all ciphertext symbols, the PDFs are close to 0.039 ($1/256 = 3.9 \times 10^{-3}$), which is consistent with a uniform distribution.



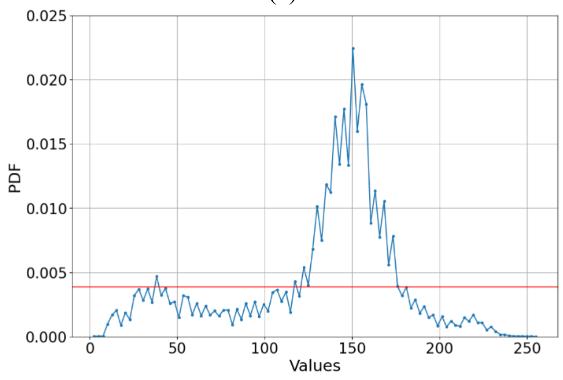
(a)



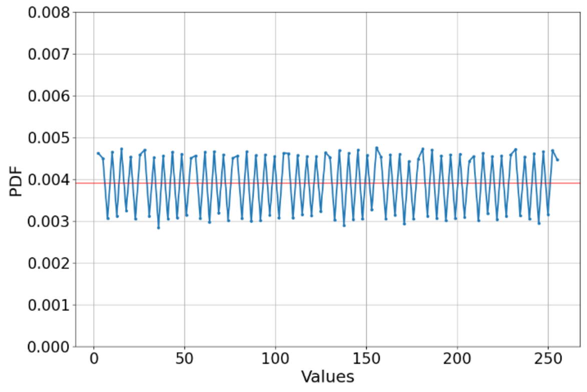
(b)



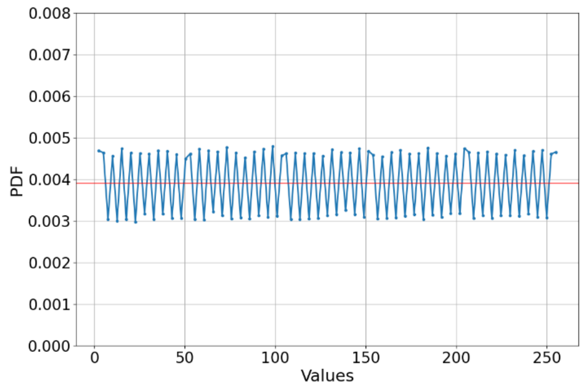
(c)



(d)



(e)

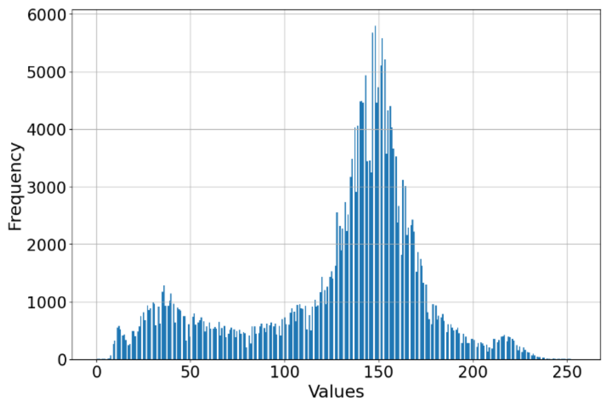


(f)

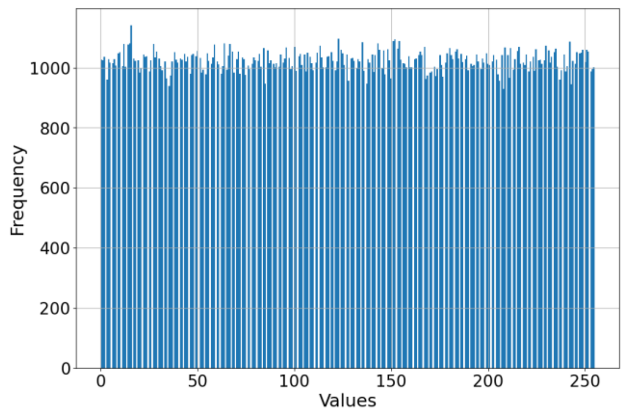
Figure 3. The recurrence of the original message (a), cipher using optimized speck (b), and the cipher using original speck (c). The related PDF of the original message (d), the produced cipher of the optimized speck (e), and the original speck (f)

6.1.2 Histogram analysis test

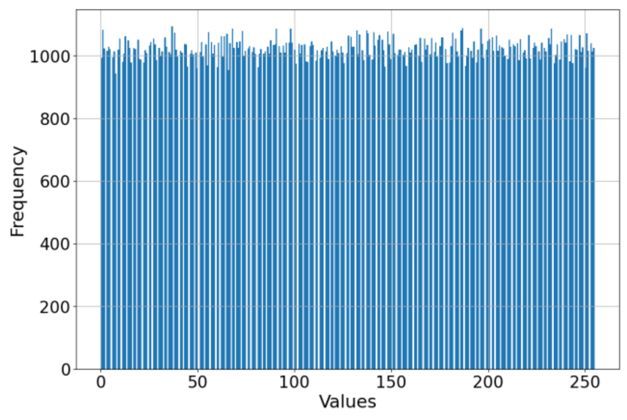
In this section, the optimized and original speck ciphers are evaluated and compared to histogram analysis tests. When the histogram of the encrypted image is normally distributed, can we say that this encryption successfully meets the uniformity criteria. This suggests that the occurrence of each symbol in the message is proportionate to the number of symbols in it. In other words, it should be somewhat close to the message size/character count. A histogram is shown in Figure 4, contrasting the 512-by-512-pixel plain images with their cipher image equivalents. It is shown that the encrypted image of both speck versions has a histogram very close to the uniform distribution given by (512*512) /256 =1024).



(a)



(b)



(c)

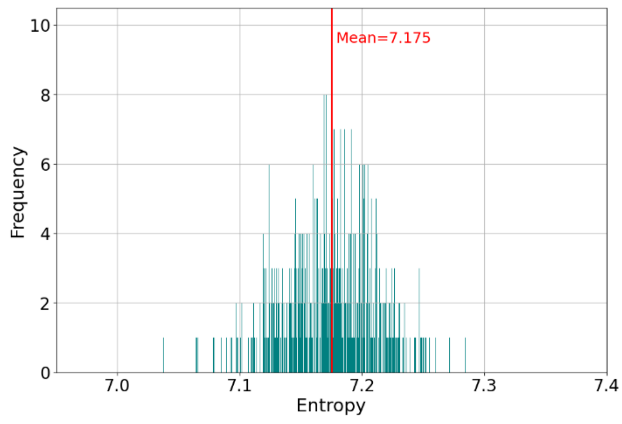
Figure 4. The histogram analysis of (a) plain message, (b) optimized, and (c) original speck

6.1.3 Entropy analysis

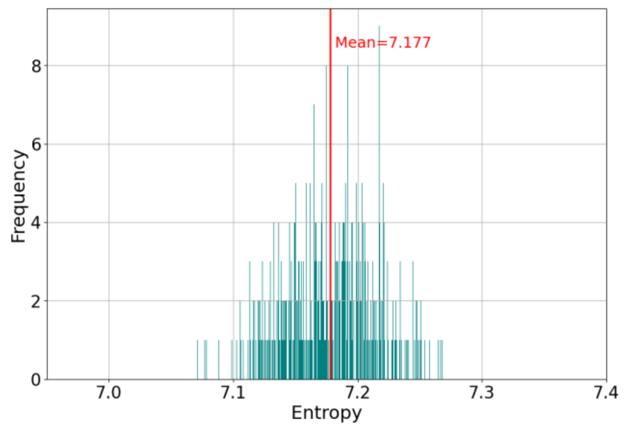
The entropy of information in a message *M*, which stands for the measure of dispersion [14], may be described as the following when applied to a random variable:

$$H(m) = -\sum_{i=1}^{h2} pr(mi)\log_2 \frac{1}{pr(mi)} \qquad (3)$$

Entropy is measured in bits, and the probability of observing a given symbol is denoted by *pr(mi),* where *mi* is an integer between *1* and *NS*. If the ciphertext's entropy is the same as or near to *log2(N S)*, then the ciphertext represents a truly random source with a normal distribution.



(a)



(b)

Figure 5. Entropy analysis for the encrypted message using both optimized and original speck in (a) and (b) respectively

In Figure 5, the comparison of the encrypted messages' entropy at the 16×16 (256-element) sub-matrix level using a random dynamic key is presented. The entropy of the resulting ciphertexts also matches the target value of 8 as shown by the results of the analysis. As a result, the proposed optimized speck cipher system is sufficiently safe against any given entropy attack.

6.1.4 Correlation analysis

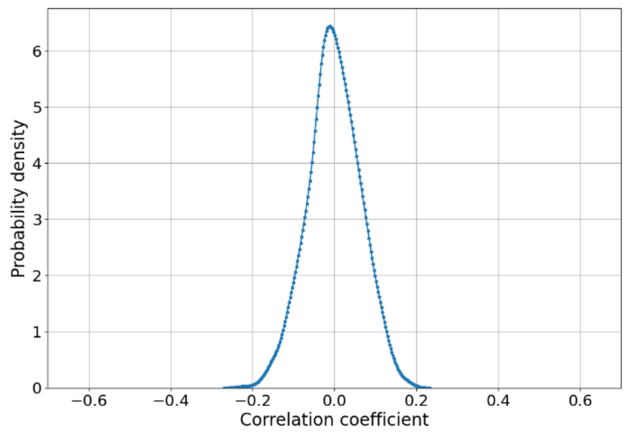
To make sure that the proposed encryption system holds up in practice, it is crucial to get rid of any relationship between the sequence of the components [15, 16]. When the correlation coefficient is small (close to zero), it indicates that the cipher scheme is actually random. To perform the correlation test, we choose pairs of adjacent pixels from the original message and the encrypted version at random. Correlation may be performed in any of the three possible dimensions (horizontal, vertical, and diagonal). The following equation is used to determine the value of the correlation coefficient r_{xy}:

$$r_{xy} = \frac{\text{cov(x,y)}}{\sqrt{D(x)\times D(Y)}}$$
$$\text{cov(x,y)}= \frac{1}{n}\sum_{i=1}^{n}(x_i - E(x))(y_i - E(y)) \; E_{x=}\frac{1}{n}\times \sum_{i=1}^{n} x_i \qquad (4)$$
$$D_{x=}\frac{1}{n}\times\left(\sum_{i=1}^{n}(x_{i\_}\,\text{E(x)})\right)^2$$

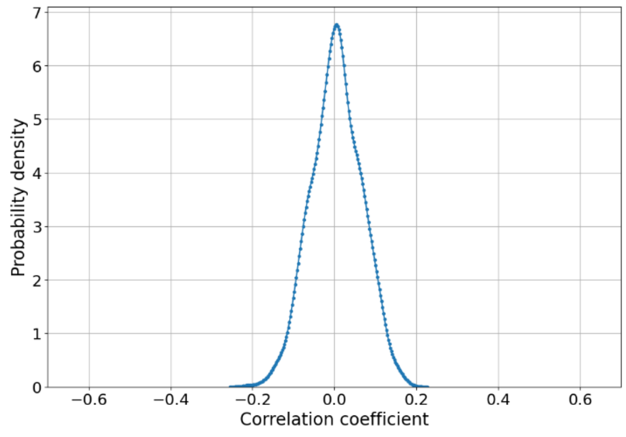
The correlation test between the original and encrypted messages of both speck versions is shown for one random key at a time and for a total of 1000 random keys in Figure 6. The results indicate that the correlation coefficient is extremely small, very near to 0, which substantiates the ciphertext's randomness and, by extension, its own independence.

6.1.5 Practrand randomness test

To ensure PRNGs are random, the PractRand testing suite uses statistical methods [17]. As was previously said, the proposed optimized Speck cipher was tested with 64 seeds under Practrand, and it succeeded in every case. The created sequence is evaluated by PractRand, and a report is generated indicating whether or not the sequence passes the tests. The encrypted message was then subjected to PractRand, which is one of the most difficult statistical tests available. This verification ensures that the generated cipher of the optimized speck algorithm is sufficiently random and reliable.



(a)



(b)

Figure 6. The PDF distribution of the correlation coefficient between the plain and encrypted message using (a) the optimized and (b) the original speck

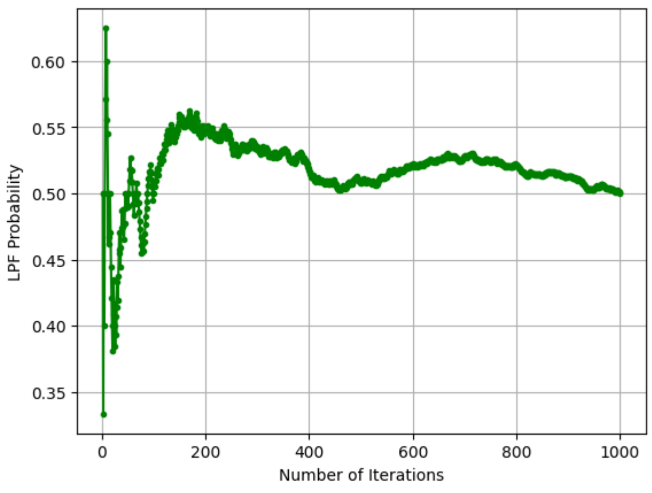
6.2 Linear and differential analysis tests

When designing cryptographic algorithms, it is essential that they be resistant to linear and differential attacks. This is particularly true for symmetric key block ciphers [18]. These assaults show two common ways that cryptanalysts attack the algorithms of cryptography and get private keys or plaintext.

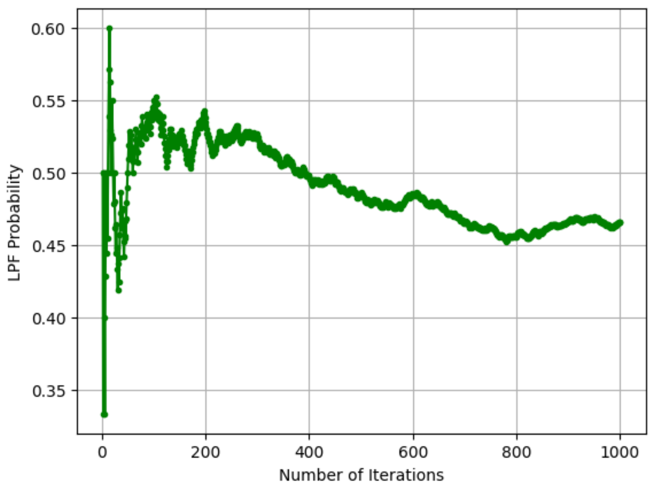
6.2.1 The linear analysis test

In the study of Aly et al. [18], the idea of linear probability approximation (LPF) was initially proposed as a strategy for linear cryptanalysis of the DES block cipher, which is used in this test. The main idea is to find a linear connection or approximation that links certain parts of the plaintext with their corresponding ciphertext counterparts. Establishing a linear relationship between the plaintext and the ciphertext makes the key more sensitive to extraction.

We compute the linear probability value for both optimized and original speck ciphers for each discrete set of 16-byte plaintext and ciphertext blocks with 1000 iterations. See Figure 7 for the test results. For optimized speck ciphers, the mean of LPF is 0.517, whereas for original speck ciphers, it is 0.479. To compute the resistance to the linear attack, the study of Matsui [19] computed the absolute difference of the desired probability 0.5 with the computed linear probability *P*. If the resulted value is close to zero, then the cipher has more resistance to liner attacks. According to this proposition, the optimized speck gives a bias of 0.017, whereas the original speck bias is equal to 0.021. Thus, the optimized speck is more resistance to the linear attack compared to its original version.



(a)

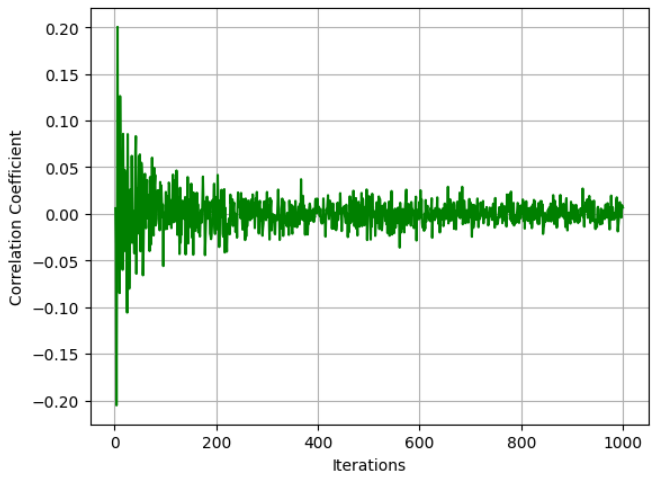


(b)

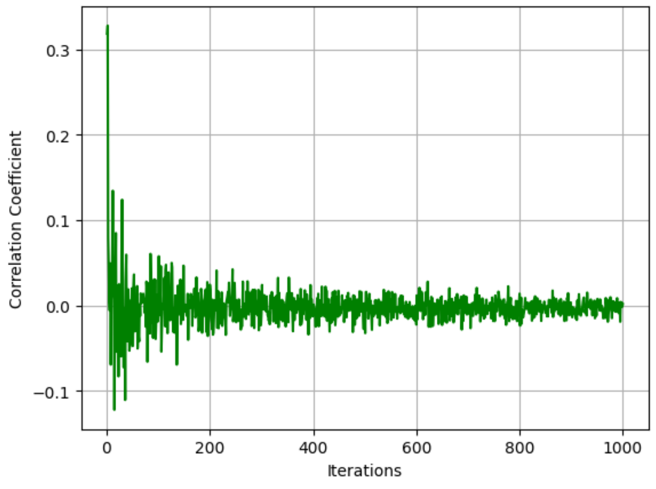
Figure 7. The linear attack analysis for (a) the optimized and (b) the original speck

6.2.2 The differential attack test

An efficient way to evaluate the safety of cryptographic algorithms is differential cryptanalysis, which involves looking at how small changes in the input data affect the output data [20]. In this test, the correlation coefficient between two sets of encrypted texts that represent two plaintexts that varied by one bit every block is computed. To evaluate the resistance of differential attacks on cipher messages, the correlation coefficient is computed and must give a lower value. In this test, 1000 sub-matrices of size 16 bytes are randomly selected from two different ciphertexts. The average of the correlation coefficient is computed for all sub-metrics of the two ciphertexts. For the optimized speck cipher, the average correlation coefficient is equal to 0.0003 and is equal to -0.0018 for the original one. Accordingly, the optimized speck has more resistance to the differential attack than the original version. Figure 8 demonstrates the computed correlation coefficient of all selected cipher blocks.



(a)



(b)

Figure 8. The analysis of differential attack for (a) the optimized and (b) the original speck

7. PERFORMANCE RESULTS AND COMPARISON

This section presents the performance of the proposed optimized speck cipher that uses two scenarios. In the initial scenario, the optimized and original Speck ciphers are executed sequentially on a single CPU core. The execution of message passing algorithms for both ciphers on a multicore CPU constitutes the second scenario. This section presents a comparison of the speedup, encryption time, and throughput metrics for both instances of the Speck crypto algorithm.

7.1 Results of sequential execution

The calculated performance comparison of the optimized and original Speck ciphers was performed on one core of the Intel(R) i7-7700HQ processor. The CPU utilizes a frequency speed of 2.80 GHz for all its processors. The assessment results of the execution time and throughput of the single-thread optimized and original ciphers on the processor are as demonstrated in Table 1.

Upon analysing the data presented, it is evident that the optimized Speck algorithm features considerable improvements in the execution time and throughput compared to the initial Speck algorithm. The average execution time and throughput speedup ratio equal to about 2.58, which means that the Speck algorithm optimized performs cryptographic operations faster than the original one. In other words, the operations it performs cope with larger amounts of transmissions at a time, boosting the efficacy of data processing.

Table 1. The sequential execution results comparison of optimized and original Speck

Message Size	Optimized Speck		Original Speck	
	Exe. Time (s)	Throughput (Gbits/s)	Exe. Time (s)	Throughput (Gbits/s)
4	0.032	1.04	0.086	0.39
8	0.062	1.08	0.172	0.38
16	0.124	1.08	0.329	0.34
32	0.243	1.10	0.540	0.49
64	0.486	1.11	1.041	0.51
128	0.982	1.09	2.602	0.41
256	1.999	1.07	5.354	0.40
512	3.933	1.09	10.151	0.42

7.2 Parallel execution results over multicore processor

Experiment of this subsection was performed on multicore processor Intel i7-7700HQ and its purpose was to determine the efficiency of the optimized Speck algorithm compared to the initial one. Algorithms of this subsection were developed using message passing interface parallel primitives MPI. To show the improvement of Speck algorithm, four different combinations of threads 2, 4, 6, 8 are used for each of the executions on the multicore processor. Figure 9 shows all results of the execution time, throughput, and speedup comparing with the single core of Speck algorithms. The mean throughput of the optimized Speck algorithm is 3.83 gigabits per second. In order to determine the speedup ratio, the sequential execution time of the optimized Speck cipher is compared to the parallel execution time. The mean speedup ratio is 2.63 on average. The mean acceleration obtained by the two Speck parallel algorithms is 2.64. Table 2 shows the average throughput and execution time for all thread counts. The advantage of the optimized algorithm relative to the original algorithm is immediately obvious. At various message sizes and thread configurations, it completes cryptographic operations much faster.

Table 2. Comparison of the parallel average results of all thread configurations for optimized and original speck

Message Size	Optimized Speck		Original Speck	
	Exe. Time (s)	Throughput (Gbits/s)	Exe. Time (s)	Throughput (Gbits/s)
4	0.012	2.83	0.029	1.25
8	0.018	4.00	0.052	1.40
16	0.038	3.75	0.099	1.43
32	0.073	3.96	0.180	1.62
64	0.140	4.14	0.368	1.59
128	0.284	4.08	0.738	1.51
256	0.627	4.00	1.681	1.43
512	1.242	3.98	3.064	1.54

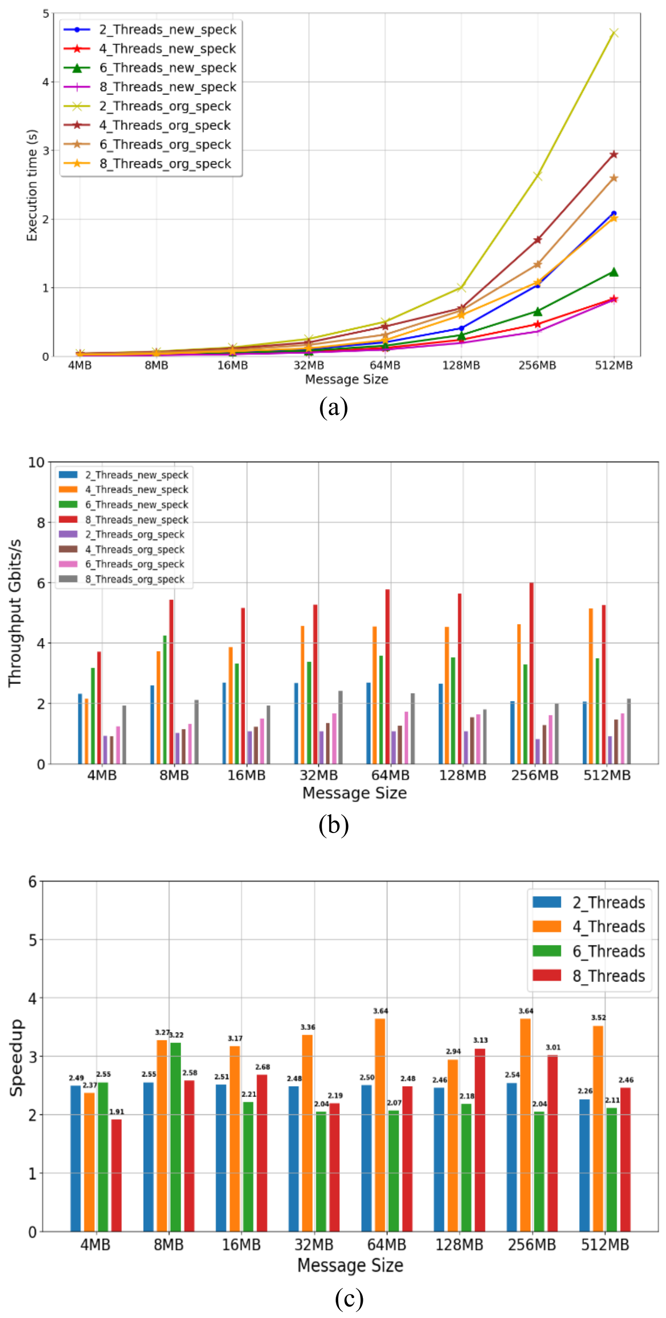


Figure 9. The experiment results over multicore: (a) execution time, (b) throughput, and (c) speedup

8. CONCLUSION

This work presents a new version of the Speck cipher which also examines its multi-core processor effectiveness by exploiting parallel processing capabilities of the CPUs. The dynamic, Randomized S-box layer of optimized speck cipher enhances its security and enables reduction in rounds. Statistical tests, randomness and immunity to linear or differential attack for both the original as well as the optimized algorithm are compared and evaluated. Consequently, security obtained indicates that the optimized speck can retain the same level of safety and face attacks. It is indicated through experimental findings that the optimized Speck algorithm has superior performance in terms of execution time, throughput, and speedup compared with its original version. The suggested optimization method integrates substitution-based adjustments with a decreased round count in order to improve encryption efficiency while maintaining security levels at an acceptable level. By average sequential execution performance increase, it has been established that the optimized Speck algorithm was 2.58 times faster than its previous generation. When executing on multi-core processor, the new Speck cipher manages larger data more efficiently by running 2.64 times faster in comparison to the original speck.

In the future, we are going to acknowledge the significance of investigating key sizes over 128 bits, such as 192 and 256 bits. The objective is to examine the effects of varying key sizes on the optimized Speck algorithm. To accommodate larger key sizes, we may contemplate making modifications to security primitives or employing round configurations. Moreover, it is interesting to implement the proposed cipher over GPUs to show its performance and efficiency.

ACKNOWLEDGMENT

The authors would like to thank the universities of Babylon, Warith Al-Anbiyaa, and Al-Mustaqbal for supporting this work.

REFERENCES

[1] Rana, M., Mamun, Q., Islam, R. (2022). Lightweight cryptography in IoT networks: A survey. Future Generation Computer Systems, 129: 77-89. https://doi.org/10.1016/j.future.2021.11.011

[2] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L. (2013). The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, 404. https://ia.cr/2013/404.

[3] Singh, S., Sharma, P.K., Moon, S.Y., Park, J.H. (2024). Advanced lightweight encryption algorithms for IoT devices: Survey, challenges and solutions. Journal of Ambient Intelligence and Humanized Computing, 15: 1625-1642. https://doi.org/10.1007/s12652-017-0494-4

[4] Daemen, J., Rijmen, V. (2020). The Design of Rijndael. The Advanced Encryption Standard (AES), Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-60769-5

[5] Ren, J.J., Chen, S.Z. (2019). Cryptanalysis of Reduced-Round SPECK. IEEE Access, 7: 63045-63056. https://doi.org/10.1109/ACCESS.2019.2917015

[6] Gohr, A. (2019). Improving attacks on round-reduced speck32/64 using deep learning. Advances in Cryptology – CRYPTO 2019, 150-179. https://doi.org/10.1007/978-3-030-26951-7_6

[7] Sleem, L., Couturier, R. (2021). Speck-R: An ultra light-weight cryptographic scheme for internet of things. Multimedia Tools and Applications, 80: 17067-17102. https://doi.org/10.1007/s11042-020-09625-8

[8] Yue, X.T., Wu, W.Q. (2023). Improved neural differential distinguisher model for lightweight cipher speck. Applied Sciences, 13(12): 6994. https://doi.org/10.3390/app13126994

[9] Liu, J., Ren, J., Chen, S. (2023). A deep learning aided differential distinguisher improvement framework with more lightweight and universality. Cybersecurity, 6(47): 2023. https://doi.org/10.1186/s42400-023-00176-7

[10] Deng, H.R., Cao, X.H., Cheng, Y. (2023). Attention in differential cryptanalysis on lightweight block cipher

SPECK. In 2023 20th Annual International Conference on Privacy, Security and Trust (PST), Copenhagen, Denmark, pp. 1-9. https://doi.org/10.1109/PST58708.2023.10320201

[11] Zhang, L., Lu, J.Y., Li, C. (2023). Improved differential-neural cryptanalysis for round-reduced SIMECK32/64. Frontiers of Computer Science, 17: 176817. https://doi.org/10.1007/s11704-023-3261-z

[12] Fanfakh, A., Noura, H., Couturier, R. (2022). ORSCA-GPU: One round stream cipher algorithm for GPU implementation. The Journal of Supercomputing, 78: 11744-11767. https://doi.org/10.1007/s11227-022-04335-4

[13] Hamiza, H.J., Fanfakh, A. (2024). Parallel lightweight block cipher algorithm for multicore CPUs. Baghdad Science Journal. https://doi.org/10.21123/bsj.2024.9052

[14] Xu, S.J., Wang, Y.L., Wang, J.Z., Tian, M. (2008). Cryptanalysis of two chaotic image encryption schemes based on permutation and XOR operations. In 2008 International Conference on Computational Intelligence and Security, Suzhou, China, pp. 433-437. https://doi.org/10.1109/CIS.2008.146

[15] Alamari, Y.A., Fanfakh, A., Hadi, E. (2023). Parallel message authentication algorithm implemented over multicore CPU. International Journal of Intelligent Engineering & Systems, 16(4): 2023. https://doi.org/10.22266/ijies2023.0831.52

[16] Zhang, G.J., Liu, Q. (2011). A novel image encryption method based on total shuffling scheme. Optics Communications, 284(12): 2775-2780. https://doi.org/10.1016/j.optcom.2011.02.039

[17] http://pracrand.sourceforge.net, accessed on Nov. 2, 2023.

[18] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A. (2020). Design of symmetric-key primitives for advanced cryptographic protocols. IACR Transactions on Symmetric Cryptology, 2020(3): 1-45. https://doi.org/10.13154/tosc.v2020.i3.1-45

[19] Matsui, M. (1993). Linear cryptanalysis method for DES cipher. Advances in Cryptology - EUROCRYPT '93, 386-397. https://doi.org/10.1007/3-540-48285-7_33

[20] Biham, E., Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology, 4(1): 3-72. https://doi.org/10.1007/BF00630563