ESSENCE: GPU-based and dynamic key-dependent efficient stream cipher for multimedia contents

Raphael Couturier, Hassan Noura, Ali Chehab

Noname manuscript No.
(will be inserted by the editor)

ESSENCE: GPU-Based and Dynamic Key-Dependent Efficient Stream Cipher for Multimedia Contents

Raphaël Couturier* · Hassan N. Noura · Ali Chehab

Received: date / Accepted: date

Abstract Data Confidentiality (DC) is considered one of the most important security services. Currently, a set of existing cipher algorithms is being used to ensure DC. However, researchers constantly investigate the design and implementation of more efficient cipher schemes. To this end, different versions of AES have been implemented efficiently on GPUs to increase the efficiency over big data. However, AES implementation on GPU exhibits limitations in terms of latency and hence, it might not be a suitable solution for high data rates in modern systems and applications. This often leads to a trade-off between system performance and security level. To address these challenges, we propose "ESSENCE", a lightweight stream cipher scheme, which combines two different Pseudo-Random Number Generators (PRNG), and based on a dynamic key approach. The scheme achieves a high level of security with minimal latency and required resources when compared to existing cipher standards such as AES. Moreover, the implementation of the proposed dynamic key-dependent cipher scheme on GPU is more efficient compared to all existing AES implementations on GPUs. Experimental results indicate that the proposed cipher is highly efficient with a throughput more than 115 GB/s on a Titan X GPU, and more than 372 GB/s on a Titan V100 GPU. Thus, ESSENCE can be considered as a promising stream cipher candidate with high randomness degree (BigCrush of TestU01), periodicity, and key sensitivity.

Keywords Lightweight GPU Stream cipher solution; Security and Performance Analysis; parallel computing; Dynamic Key dependent cryptographic primitives; Cryptanalysis

Raphaël Couturier
Univ. Bourgogne Franche-Comté (UBFC), CNRS, FEMTO-ST Institute, France
E-mail: raphael.couturier@univ-fcomte.fr

Hassan N. Noura and Ali Chehab
American University of Beirut
Department of Electrical and Computer Engineering, Beirut, Lebanon.
E-mail: {hn49,chehab}@aub.edu.lb

1 Introduction

Security has become the most important armor responsible for protecting all kinds of resources and data from various types of threats targeting security services such as data confidentiality, integrity, and source authentication. These security services are typically ensured by resorting to cryptographic solutions, which are essential to overcome and limit such threats. Existing attacks can be either active or passive, where passive attacks can seriously impair the Data Confidentiality (DC) and privacy of the system, while active attacks can compromise its authentication (source, user, device), integrity, and availability. Moreover, the nature of passive attacks makes them very difficult to detect compared to active ones. An active attacker may insert, delete or modify data contents. Encrypting communicated or stored data can solve all problems related to passive attacks. However, this requires a distributed scheme and a robust key exchange mechanism. Typically, symmetric-key schemes are used for data encryption, especially since they are more efficient in terms of memory and computational complexity compared to asymmetric-key ones. Furthermore, conventional symmetric key ciphers are either block-based or stream-based. Several standardized cipher algorithms that ensure DC already exist, including the stream cipher RC4 [1], and the block cipher AES [2].

In general, a block cipher [3] uses a round function that can either be based on a Feistel Network (FN) such as DES, or on Substitution-Permutation Networks (SPN) such as AES. SPN lends itself to parallel implementation and requires a lower number of rounds compared to FN and hence, SPN exhibits lower latency and requires fewer resources than FN.

1.1 Related Work

AES [4] is a block cipher that processes data in blocks of size 128 bits (16 bytes), and it uses keys of size 128, 192 and 256 bits. The design of AES depends on the SPN principle. It includes a *round* function, which consists of diffusion and substitution operations, and it is iterated r times, depending on the size of the secret key. The number of rounds, r, is equal to 10, 12, and 14 for a secret key of size 128, 192, and 256 bits, respectively.
Each round, except for the last, includes four operations:

- RoundKeyAddition: it mixes the plain input block with the specific round key.
- ByteSubstitution: the operation employs a substitution table, S-Box, to ensure the confusion property.
- ShiftRows and MixColumn operations are used to ensure the diffusion property. Note that the MixColumn operation is eliminated in the last round.

1.2 Problem Formulation

The security level of existing symmetric ciphers, against analytic attacks, depends on the number of rounds r, which leads to a trade-off between the security level and the required latency and resources. Ciphers that are based on a static structure have proven their resistance against analytic cryptanalysis. However, the static structure of the round function represents the main security issue. Moreover, since the cipher primitives are static, the required number of rounds r is high, where different substitution and diffusion operations are performed within each round [5, 6, 7].

Fixed cipher structures lend themselves to future potential attacks [8, 9], which would benefit from the fixed structure (substitution and diffusion primitives) to recover the secret key [10]. Examples of such attacks include implementation attacks such as side-channel attacks and fault attacks [10]. Hence, countermeasures against implementation attacks are required, which would increase the latency and required resources. This, in turn, reduces their performance and makes them not suitable for some of the future systems and applications [11].

1.3 Motivation

To overcome these limitations, our approach uses the dynamic key-dependent structure as in [5, 6] to reduce the required number of rounds and operations. This leads to a good balance between efficiency and security level, as well as offering a simple solution to prevent certain implementation attacks.

To reduce the execution time of the existing cryptographic algorithms, GPU (Graphic Processing Unit) implementations are being adopted. A GPU is useful for cryptographic algorithms, which can benefit from the hundreds and even thousands of cores in a GPU. Researchers use GPUs to generate pseudo-random numbers such as in [12, 13]. Also, standard cryptographic algorithms have been implemented on GPUs such as AES [14, 15, 16], which resulted in an impressive speed-up [17] compared to the CPU implementation. It is worth noting that the efficient implementation of an algorithm on a GPU requires the expertise to optimize the use of the GPU architecture in terms of shared memory, registers, and warp [18].

Recently, an optimized and efficient implementation of AES on GPU was presented in [16]. It achieved an excellent performance and the authors made various optimization compared to the previous related works. Accordingly, this implementation is selected as the reference for comparison against the proposed cipher solution. There is another recent implementation of AES on GPU, PHAST, which was described in [19]. This implementation is more generic and it resulted in about 10% decrease in performance as compared to [16].

1.4 Contributions

The proposed cipher solution follows the recent dynamic key-dependent approach of [5, 7, 20]. In contrast to these related solutions, no integer diffusion operation is used in the proposed dynamic key-dependent stream cipher. This operation is eliminated without weakening the cipher security level since the cryptographic primitives are updated for each new input data. Moreover, the proposed solution does not require the avalanche effect, but it is based on high key sensitivity.

The proposed cipher scheme uses an efficient and simple key-stream generation algorithm that uses dynamic permutation and substitution tables in addition to two different PRNGs with a large number of seeds. To the best of our knowledge, the proposed solution is the first dynamic key-dependent stream cipher algorithm with dynamic seeds and substitution/permutation tables.

Next, we list the technical contribution of this paper as compared to the existing cipher solutions:

- The proposed cipher is based on a dynamic key-dependent approach, and it is based on a simple key derivation function that uses a variable session key and a Nonce, which change for each new input message, making it highly resistant against attacks.
- The permutation table is used as a perturbation technique to modify the internal state, which increases the periodicity of the employed PRNGs.
- The proposed solution uses a dynamic substitution process to increase the nonlinear degree of the generated key-stream and to achieve higher key sensitivity.
- The proposed cipher exhibits a high level of randomness, which was verified using the "BigCrush" of "TestU01" [21] statistical suite tests on the generated key-stream.
- The proposed cipher scheme uses lightweight PRNGs and simple operations, which minimizes the latency and required resources, and leads to a simple software implementation.

In summary, The proposed cipher satisfies the desirable cryptographic characteristics such as long periodicity, high level of key sensitivity, and high level of randomness and thus, higher resistance against attacks, with low latency and overhead.

1.5 Organization

The remainder of the paper is organized as follows. Section 2 describes and analyzes existing GPU cipher implementation. In Section 3, the proposed dynamic key derivation is presented. While in Section 4, the employed cipher primitives construction techniques are described. Then, in Section 5, we introduce and describe in details the proposed stream cipher algorithm, along with

the functionality of each operation. In Sections 6 and 7, we respectively assess the robustness of the proposed cipher scheme and its performance. Finally, in Section 8, a conclusion and future directions are presented.

2 Existing GPU Cipher Algorithms and Their Corresponding Implementations

A GPU (Graphic Processing Unit) is a commonly used architecture to accelerate computations. GPUs are used in many computing applications and systems ranging from smart-phones, embedded computing, to supercomputers. The architecture of a GPU is quite different from that of a CPU. In a GPU, the architecture is optimized to maximize the execution throughput of many simultaneous threads. The number of computing cores inside a GPU ranges from hundreds to even thousands. The hardware is designed to execute many threads, even if the bottleneck is the memory access itself. To benefit from the GPUs computing power, users need to use a number of threads that exceeds the number of cores. Hence, while some threads are waiting for their data, other threads are capable of executing. Typically, there are many kinds of memory in a GPU: global memory which is the slowest one, cache memory, texture memory, shared memory, local memory, and a limited set of registers having the fastest access. Consequently, memory management is critical within GPUs.

GPUs are composed of streaming multiprocessors (SMs), and their number varies for different GPU types. Typically, SMs are composed of 32 cores that can execute only a single instruction at a time. So, if two threads are executed on the same SM, one instruction will be executed, while the second one would have to wait. This is called thread divergence. Consequently, "IF" instructions and "WHILE" conditions must be avoided, whenever possible. Threads are scheduled by groups of 32 on an SM, and they are referred to as warps. In practice, threads are organized into blocks; depending on the GPU architecture, the maximum thread number per block is limited to 1024. Hence, it is important to keep in mind that GPUs technical details are constantly changing with every new generation.

3 Dynamic Key Derivation Function

In this section, the proposed dynamic key generation function and the corresponding sub-keys generation schemes are presented and illustrated in Fig. 1. The cipher primitives (seeds and permutation boxes) are dynamic and they change based on this set of sub-keys. The specific secret key, SK, is mixed with a NONCE N_o (unique for each new input) to produce a dynamic secret, O. Then, the new dynamic key (DK) is obtained by hashing O using a secure cryptographic hash function. To ensure that a different DK is produced for each different input message or session, the SHA-512 hash function is chosen

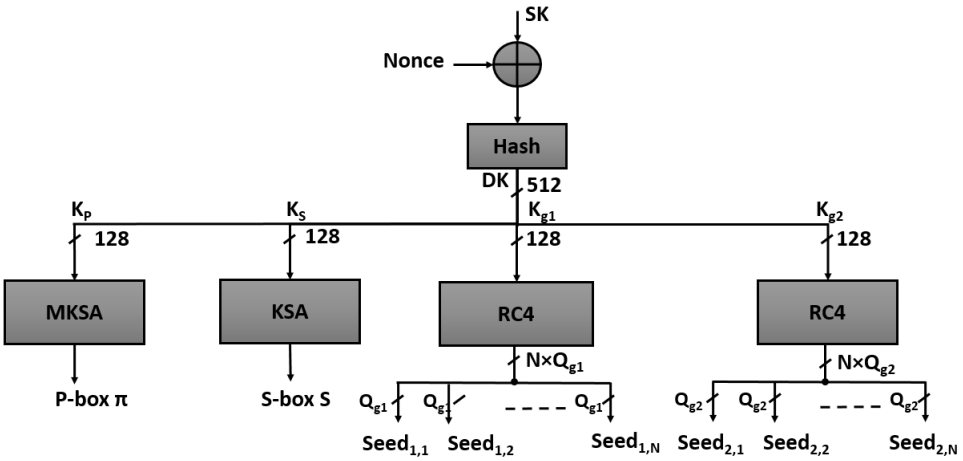


Fig. 1: The proposed Key derivation function and its corresponding construction of cipher primitives.

and it is known for its high resistance degree against collisions. The dynamicity introduces robustness against powerful attacks. The dynamic key, DK, is used to generate the required sub-keys as explained next.

- **Master Secret Key** K: It is shared between both legal entities to provide enhanced security. It allows the symmetric secret key to be renewed after each periodic interval, depending on the application itself. For example, Elliptic Curve Diffie Hellman (ECDH) protocols can be selected for this specific task.
- **Nonce** N_o: Each Nonce will be used only once; it is updated for every input image or session. Two possible Nonce generation techniques can be adopted i) generated by the sender and transmitted to the receiver in an encrypted form, by either employing a secret key or by employing the receiver public key; ii) producing the Nonce at the sender and receiver in a synchronized manner, through the use of a deterministic pseudo-random generator.
- **Dynamic Key** DK: The master secret key K is XORed with N_0, and the output is hashed using SHA-512. This generates the dynamic key, DK, which represents the MAC value with a size of 512 bits. Then, DK is divided into 4 main sub-keys $\{K_P,\ K_S\ \ Kg_1,\ Kg_2\}$ each with a size of 16 bytes (128 bits).

4 Construction of Dynamic Cipher primitives

The sub-keys are used to generate the required cipher primitives, as described below.

– **Permutation sub-key** K_P: it consists of the most significant 16 bytes of DK, and it is used to produce a set of permutation tables (32 $P-boxes$) that can be employed during the selection process. In this solution, any key-dependent permutation generation algorithm can be employed such as the ones in [7, 22]. The selection of the Modified Key Setup Algorithm (MKSA) of [22] is used to construct the required dynamic key-dependent permutation tables. In fact, MKSA is selected due to its simple hardware and software implementations. To ensure that a P-box has a good cryptographic performance, MKSA should be always iterated with a different key in order to produce different P-boxes. Therefore, K_P is used as a seed for the RC4 just to generate a set of permutation sub-keys, and each sub-key is used as a seed for the MKSA to produce a different permutation table. On the other hand, the weaknesses of RC4, as reported in [23, 24, 25] do not affect the proposed solution, which is based on a dynamic key-dependent structure.

  RC4 will be iterated to generate a byte vector of length equals to $Np \times lp$. Then, the output is reshaped to form a matrix with a size of $Np \times lp$, where each row represents one of the dynamic permutation keys with a length of $Qp = lp \times 8$ bits, to be used as a permutation table.

  Note that $RC4$ is iterated with a dynamic sub-key to avoid any weakness and to achieve a high level of security.
– **Substitution sub-key** K_S: it represents the second set of the 16 most significant bytes, and it is used to produce a set of substitution sub-keys, where each sub-key is used to produce a dynamic substitution table (S-box). Any key-dependent algorithm could be used for the generation of the substitution tables. We adopt the simple technique used in [22], which is based on the Key Setup Algorithm (KSA) of RC4. The output of the original KSA, for any input key, is a substitution table that is used as a dynamic S-box. RC4 is iterated to form a byte vector of length equals to $Ns \times ls$. Then, the output is reshaped to form a matrix of size $Ns \times ls$; each row represents one of the dynamic substitution keys, with a size of $Qs = ls \times 8$ bits, and used as a key-dependent substitution table.

– **First PRNG seed** Kg_1: it represents the third most significant 16 bytes of DK and it is used to produce a set of seeds of length $lg1$, one of which is selected for each thread. Also, in this step, RC4 is selected and it is iterated for $\frac{lg1 \times Qg1}{8}$ times to generate different N seeds, where N represents the possible number of threads, and $Qg1$ represents the precision of the first generator, which can be equal to 32, 64 or 128. The output key-stream is reshaped to form a byte matrix of size $N \times \frac{Qg1}{8}$. Each row of this matrix represents one of the seeds, and it has a length equals to $Qg1$ bits. Any repeated row (seed) is eliminated from this list and RC4 is re-iterated to produce a new seed.

– **Second PRNG seed** Kg_2: It represents the fourth most significant 16 bytes of DK and it is used to produce a set of seeds of length N, one of which is selected for each thread. Similarly, RC4 is selected and it is iterated for $\frac{N \times Qg2}{8}$ times to generate different N seeds. Besides, $Qg2$ represents the precision of the second generator. The output key-stream is reshaped to form a byte matrix of size $N \times \frac{Qg2}{8}$. Each row has a size of $Qg2$ bits, and represents one of the seeds. Any repeated row (seed) is also eliminated from this list, and RC4 is re-iterated to produce a new seed.

All notations are shown in Table 1. These steps guarantee a high level of sensitivity, where any tiny change in the dynamic key would result into a completely different cipher primitive in the generation process; such a change was proven in Section 6.2. The parameters' derivation is illustrated in Fig. 1.

Table 1: Table of notations

Notation	Definition
K	Secret key
N_o	Nonce
DK	Dynamic Key
K_P	Permutation sub-key
K_S	Substitution sub-key
K_g1	First PRNG sub-key
K_g2	Substitution sub-key
$P-box$	A dynamic produced permutation box
$S-box$	A dynamic produced substitution table
$Seed_1$	A dynamic set of seed for the first generator
$Seed_2$	A dynamic set of seed for the second generator
$Seed_{1,i}$	The i^{th} seed for the first generator
$Seed_{2,i}$	The i^{th} seed for the second generator
N	Number of possible threads
Q_g1	Precision of the first generator that can be 32, 64 or 128.
Q_g2	Precision of the second generator that can be 32, 64 or 128.
l	Number of bytes of the input message
nb	Number of blocks in an input message.
M_i	The i^{th} block of plain message
C_i	The i^{th} block of encrypted message

5 Proposed Stream Cipher Algorithm

This section describes the proposed stream cipher, "ESSENCE", which is designed with a single round to outperform AES. The main properties of the proposed solution are: high-security level, reduced computational complexity, and simple and parallel hardware and software implementations.

5.1 Basic Concepts

The proposed scheme is based on 3 main concepts:

– **Parallel Computing:** This algorithm is designed to run in parallel. All the threads are independent of each other and they could be all executed in parallel (see Fig. 2), even if it is not possible to schedule all of them at the same time.
Multi-streaming multiprocessors (SM) contain each 32 syn-chrome threads and shared memory and hence, the same operation is applied on these syn-chrome threads but with different inputs. In the proposed scheme, every 32 threads are iterated to perform the same function. For example, the first PRNG with different seeds is iterated to produce 32 outputs, each represented by 32 bits.

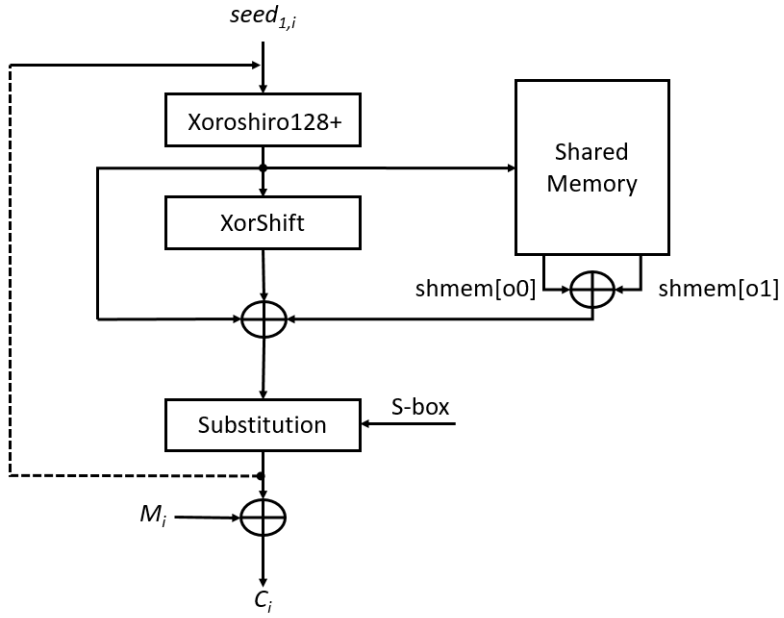


Fig. 2: Scheme of the proposed lightweight stream cipher algorithm for the i^{th} thread.

– **Flexible Structure:** The structure of the proposed stream cipher allows for any pair of outputs of the efficient PRNGs to be used [26]. Therefore, any PRNG that exhibits very good performance and satisfies the randomness properties could be used. For example, in this paper, we use "xoroshiro128plus" and "xorshift", which were selected due to their simplicity and efficiency. The proposed solution uses both PRNGs since TestU01 can detect the link between all the threads, if only one PRNG is used.

- **Efficient & Lightweight Combination of both PRNGs** The selected pairs of PRNGs are combined in an efficient manner (diffusion operation) to produce a key-stream with high periodicity, and a stable randomness degree. The proposed technique benefits from the shared memory of GPU, whereby the output of the first PRNG is stored in the shared memory. Then, the output of the second PRNG is mixed with two different outputs of the first PRNG.
- **Dynamic Selection of Shared Memories**. These shared memories (O_0, and O_1) are selected according to dynamic permutation tables (32 different P-boxes).

In the following, we describe a set of possible pseudo-random generators that can be used in the proposed stream cipher.

5.2 xoroshiro128+ (XOR/rotate/shift/rotate)

It is a successor to Xorshift (implementation at xorshift128+). It uses a carefully handcrafted shift/rotate-based linear transformation, as shown in Algorithm 1. This PRNG ensures a significant reduction in latency and the corresponding resources. Also, this PRNG reaches a high level of randomness. It has a repetition period of (2^{128}-1), which is not long enough for cryptographic algorithms. Therefore, the proposed stream cipher scheme uses a higher number of threads and for each thread, a xoroshiro128+ PRNG is used. Also, a diffusion operation is applied to the output of three different xoroshiro128+ PRNGs (different for each iteration) to increase the periodicity of the proposed key-stream. The xorshift64 algorithm is presented in Algorithm 2.

Algorithm 1 xoroshiro128plus code

```
__device__ inline
ulong xoroshiro128plus(ulong2* rng) {
  const ulong s0 = rng->x;
  ulong s1 = rng->y;
  const ulong result = rng->x + rng->y;
  s1 ^= s0;
  rng->x = rotl(s0, 24) ^ s1 ^ (s1 << 16);
  rng->y = rotl(s1, 37);
  return result;
}
```

5.3 Xorshift

Xorshift belongs to a class of PRNGs that is based on linear-feedback shift registers (LFSRs), which is described in Algorithm-2. Xorshift allows for an

Algorithm 2 xorshift64 code

```
__device__ inline
ulong xorshift64(ulong t)
{
  ulong x = t;
  x ^= x >> 12;
  x ^= x << 25;
  x ^= x >> 27;
  return x;
}
```

efficient implementation without the need of excessively using sparse polynomials. This makes them extremely fast on any modern computer architecture. Similar to LFSRs, the available parameters must be chosen with extreme caution in order to achieve a long period [27]. However, xorshift generators do not have non-linear steps. This makes them fail some statistical tests [27]. However, Xorshift generators do have numerous advantages including low execution time as well as a simple implementation.

5.4 Proposed Encryption Algorithm

Below, we describe the various steps of the proposed algorithm, as illustrated in Algorithm-3:

Note that the input data is stored in the d_input table, and the encrypted data (output) is stored in the d_output table. As the input is not changed, the keyword __restrict__ allows the compiler to optimize the variable's access, which reduces the memory access time. Other unchanged variables also have this keyword during the execution of the algorithm.

The variable d_xoro is used to store the required internal values for the "xoroshiro128plus" PRNG [28]. Each thread has a different value. To improve the performance, in many GPU algorithms, one is advised not to compute more than a single value per thread. Consequently, in our algorithm, the variable nb represents the number of elements that each thread is responsible for. The use of the loop is essential to reduce the number of threads used in the code to maximize the GPU's occupancy. Without the loop, the performance would be diminished.

In the main loop, the xoro variable is used to select 2 permutation tables from the 32 generated ones. Note that we could have chosen bigger permutation tables. However, in this case, we would need to use the __syncthreads() instruction to synchronize threads on different warps. However, such an instruction reduces the performance significantly. These permutation tables are obtained using 32 P-boxes generated with the initial key provided to the proposed ESSENSE PRNG. So, the variable d_pbox contains 32 random permutations tables of size 32. Variable shmem is the shared memory that allows threads to exchange their values. It should be noted that each thread will have

Algorithm 3 ESSENCE kernel

```
__global__
void essence_kernel(ulong2 *d_xoro, uchar * __restrict__ d_pbox,
uchar * __restrict__ d_sbox, ulong *d_output,
const ulong * __restrict__ d_input, int nb_ele,int nb) {
  uint i = blockIdx.x*blockDim.x + threadIdx.x;
  if(i<nb_ele) {
    ulong res,res2,res3;
    uchar *resc;
    ulong2 xoro=d_xoro[i];
    unsigned offset=threadIdx.x & 31;
    unsigned base=threadIdx.x-offset;
    for(int j=0;j<nb;j++) {
      int o0=base+d_pbox[32*(xoro.x&15)+offset];
      int o1=base+d_pbox[32*(16+xoro.y&15)+offset];
      res=xoroshiro128plus(&xoro);
      shmem[threadIdx.x]=res;
      res2=xorshift64(res);
      res2=res2^shmem[o0]^shmem[o1];
      res3=res^res2;
      resc=(uchar*)&res3;
      resc[0]=d_sbox[resc[0]];
      resc[1]=d_sbox[resc[1]];
      resc[2]=d_sbox[resc[2]];
      resc[3]=d_sbox[resc[3]];
      resc[4]=d_sbox[resc[4]];
      resc[5]=d_sbox[resc[5]];
      resc[6]=d_sbox[resc[6]];
      resc[7]=d_sbox[resc[7]];
      d_output[i+j*nb_ele]=d_input[i+j*nb_ele]^res3;
    }
    d_xoro[i]=xoro;
  }
}
```

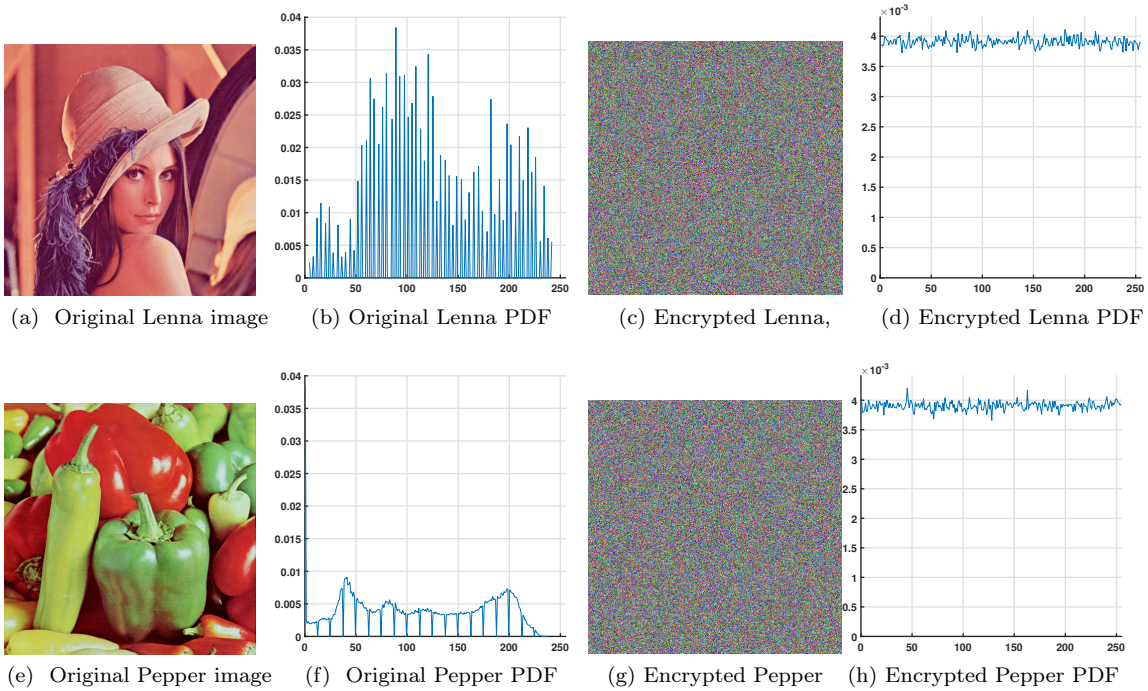
values coming from different permutation tables. For example, thread 0 will xor its result with threads 2 and 10, while thread 1 will xor its result with threads 3 and 8, and thread 2 will xor its results with threads 31 and 9, and so on. Moreover, at each iteration of the loop, the values of o0 and o1 change.

Then, the algorithm calls the xoroshiro128plus function, which changes the variable xoro, and puts the result into the variable res. Then, the shared memory is used to save this variable before xoring it with 2 other numbers generated by 2 other threads (according to the two permutation tables, as previously mentioned). The variable res is used as input to the second PRNG (xorshift64), and the result is saved in res2. Then, res2 is xor-ed with two other values coming from two other threads (in the same warp). Next, res and res2 are also xor-ed in order to obtain res3. Finally, a substitution table d_sbox is used to substitute 4 or 8 different bytes of res3 for an output of 32 or 64 bits, respectively. Note that the output is converted to an unsigned char table before applying the substitution operation on each element of the table. At the end

of the loop, the internal value of xoro is saved for the next call of the function. Finally, it should be noted that nb_ele is the total number of threads, which depends on the size of the data to encrypt.

5.5 Proposed Decryption Algorithm

A legitimate receiver will use the same steps for decryption as the ones for encryption, and the same secret and Nonce to produce the specific dynamic key. This allows for the generation of the required cipher primitives. Then, the decryption algorithm proceeds in a similar manner to the encryption algorithm.



(a) Original Lenna image (b) Original Lenna PDF (c) Encrypted Lenna, (d) Encrypted Lenna PDF

(e) Original Pepper image (f) Original Pepper PDF (g) Encrypted Pepper (h) Encrypted Pepper PDF

Fig. 3: (a) and (e) show original images; (b) and (f) show their corresponding PDF; (c) and (g) show their corresponding encrypted images; (d) and (h) show the PDF of encrypyed images. In (b), (f), (d) and (h), the x-axis and y-axis represent the symbol values and their corresponding probability values.

6 Security Analysis

An efficient encryption algorithm should be able to resist the most known types of attacks such as statistical, differential, chosen/known plain-text, and brute-force attacks [7, 22]. Extensive experiments are performed in this section to demonstrate the efficiency and security level of the proposed scheme against such attacks. Note that the proposed solution can be used for any kind of data (structured or unstructured), but the following results are provided for multimedia image contents.

6.1 Statistical Analysis

To guard against statistical attacks, a cipher must exhibit a high degree of randomness and uniformity [29]. To test the randomness degree, the following statistical security tests were carried out, (a) Probability Density Function (PDF) analysis, (b) Entropy analysis and (c) Correlation between plain and encrypted images.

6.1.1 ***Uniformity Analysis***

The most important test is the probability density function(PDF) of the encrypted image, which must be uniform; every symbol has a probability occurrence close to $\frac{1}{n}$, where n is the number of symbols. The PDFs of two original plain-images and their corresponding cipher images are shown in Figure 3. It can be seen that the PDFs of the encrypted images are close to a uniform distribution, with a value close to 0.039 that is $\frac{1}{256} = 3.9 \times 10^{-3}$.

6.1.2 ***Information Entropy Analysis***

The information entropy, of a given image M, is a parameter that measures the uncertainty level in a random variable [30], and it is defined by:

$$H(m) = -\sum_{i=1}^{h^2} p(m_i) \log_2 \frac{1}{p(m_i)} \tag{1}$$

The entropy is expressed in bits, and $p(m_i)$ indicates the occurrence probability of symbol m_i, and NS the total number of symbols. If the entropy of the encrypted data is either equal to or close to $log_2(NS)$, it can be considered as a true random source with a uniform distribution.

The Entropy analysis of the encrypted Lenna image, at the sub-matrix level with a dimension of 16×16, and by using a random dynamic key, is shown in Fig. 4. The results indicate that the encrypted images have an entropy similar to the desired value of 8. As such, the proposed cipher is sufficiently secure against any given entropy attack.

6.1.3 ***Independence***

Removing any correlation between the sequence of elements is highly essential to ensure the robustness of the proposed cipher scheme [22]. Having a correlation coefficient close to zero means that the cipher scheme exhibits a high randomness degree. The correlation test is performed by randomly taking adjacent pixels from an original image and its corresponding encrypted image. This correlation can be done in horizontal, vertical and diagonal directions. The correlation coefficient r_{xy} is calculated using the following equation:

$$r_{xy} = \frac{cov(x,y)}{\sqrt{D(x) \times D(y)}} \tag{2}$$

where :

$$cov(x,y) = \frac{1}{N} \times \sum_{i=1}^{N} (x_i - E(x))(y_i - E(y))$$

$$E_x = \frac{1}{N} \times \sum_{i=1}^{N} x_i$$

$$D_x = \frac{1}{N} \times \sum_{i=1}^{N} (x_i - E(x))^2$$

The correlation results of the original and encrypted images, and for (2,000 pairs of adjacent pixels), are shown in Fig. 5 and Fig. 6, for one random key, and for 1,000 random keys, respectively. The results clearly show that the adjacent pixels of the plain image have a high correlation, close to 1. However, the coefficient correlation of the encrypted images tends is very low, close to 0, confirming the randomness property of the proposed cipher.

6.1.4 ***Plain Data vs. Encrypted Data***

The encrypted data should be very different from the original one, with a difference of at least 50%, at the bit level. According to the obtained result in Fig. 7-a), the proposed cipher scheme satisfies the desirable difference results, with a percentage of at least 50% between the plain and the encrypted Lenna images.

6.2 Sensitivity Tests

Differential attacks are based on studying the relation between two encrypted messages resulting from a slight change, such as a one-bit difference, between two original messages. The sensitivity tests must confirm that a small change in the plain-image or in the key affect the cipher image and generate a different one. The higher the difference, the better is the sensitivity of the encryption algorithm.

6.2.1 ***Key Sensitivity test***

This is one of the most important tests, and it quantifies the sensitivity against a slight change in the secret key. The proposed key derivation function is based on a secret key and a Nonce. To further study the key sensitivity, two dynamic keys are used, DK_1 and DK_2, which differ by a single random bit. The two plain-images are then encrypted separately, and the Hamming distance of the corresponding encrypted images, C_1 and C_2, is computed and illustrated in Fig. 7-(b) against 1,000 random dynamic keys. We can see that the majority of values are close to the optimal one (50 %). This confirms the high key sensitivity of the proposed cipher algorithm. Additionally, the obtained results of 49.9970 are acceptable when compared to the reported ones of AES.

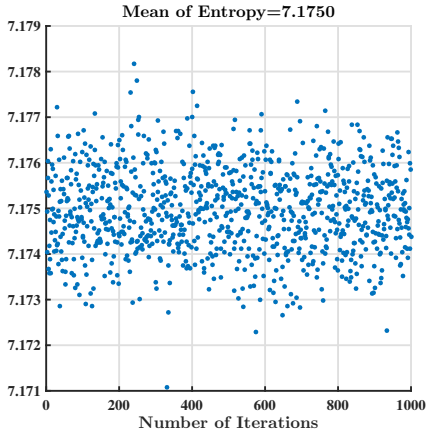


Fig. 4: Entropy analysis of encrypted Lenna versus 1,000 random secret keys at the sub-matrix level. Encrypted image is divided into a set of sub-matrices of size 16×16 and $NS = 256$ bytes (mean equal to 7.175).

6.2.2 Plain-text Sensitivity

Since a different dynamic key is being used for each input image, the algorithm produces a completely different cipher image for the same plain image. Hence, the proposed cipher successfully satisfies the avalanche criteria.

6.3 **Visual Degradation**

This test is restricted to image and video contents, and it quantifies the visual degradation associated with the output of a cipher scheme. Two popular pa-

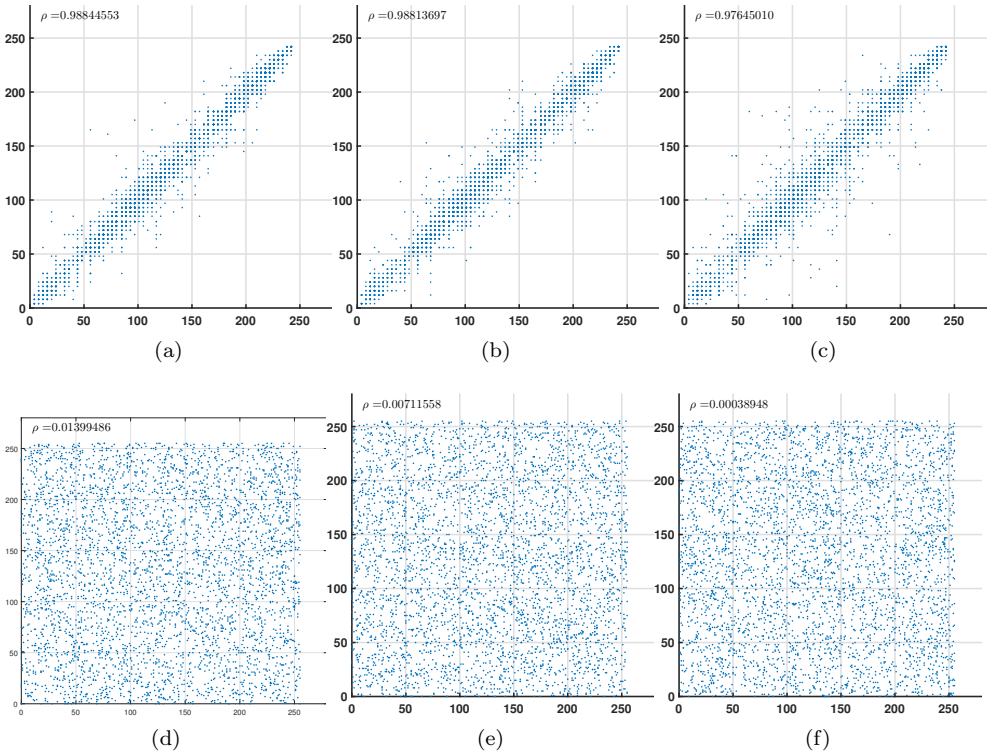


Fig. 5: Correlation distribution in adjacent pixels (2,000 pairs) in original Lena: (a) horizontally, (b) vertically and (c) diagonally.
Correlation in adjacent pixels in ciphered Lena:(d) horizontally, (e) vertically and (f) diagonally.

rameters are assessed to measure the visual quality: the Structural SIMilarity index (SSIM) [31], and the Peak Signal-to-Noise Ratio (PSNR) [32].

The PSNR is derived from the Mean Squared Error (MSE), which represents the cumulative squared error between the encrypted and original images. A low PSNR value indicates a high difference between the cipher and original images. On the other hand, SSIM lies in the [0,1] interval, where 0 means the absence of correlation between original and cipher images, while a value close to 1 indicates a high correlation between the original and cipher images. We measured $PSNR$ and $SSIM$ between the original and encrypted Lenna images for $1,000$ random keys. The results are presented in Fig. 8-(a) and (b), respectively. It can be seen that the value of the PSNR is 9.23 dB, which is a low value and confirming the high difference between the original and encrypted images. Also, the $SSIM$ values are always close to zero, which confirms that a high and hard visual distortion is achieved by the proposed cipher algorithm.

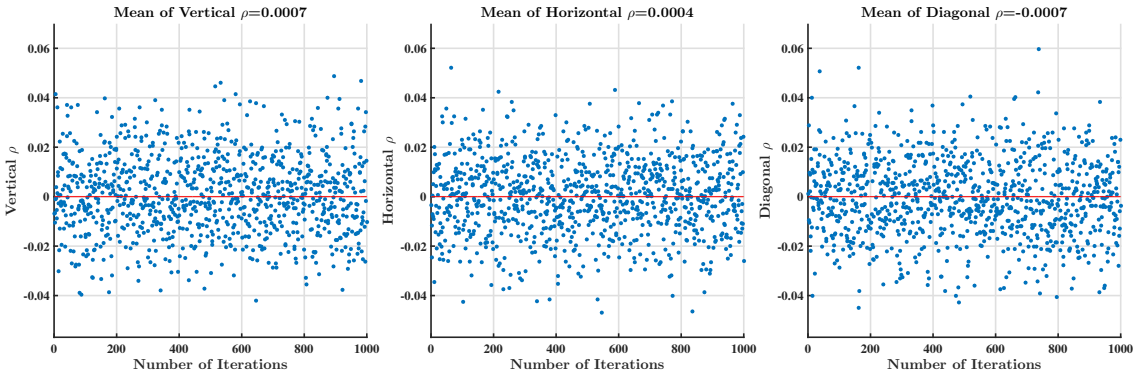


Fig. 6: The variation of the correlation coefficient for adjacent pixels in ciphered Lenna image versus 1000 random keys: (a) horizontally, (b) vertically and (c) diagonally.

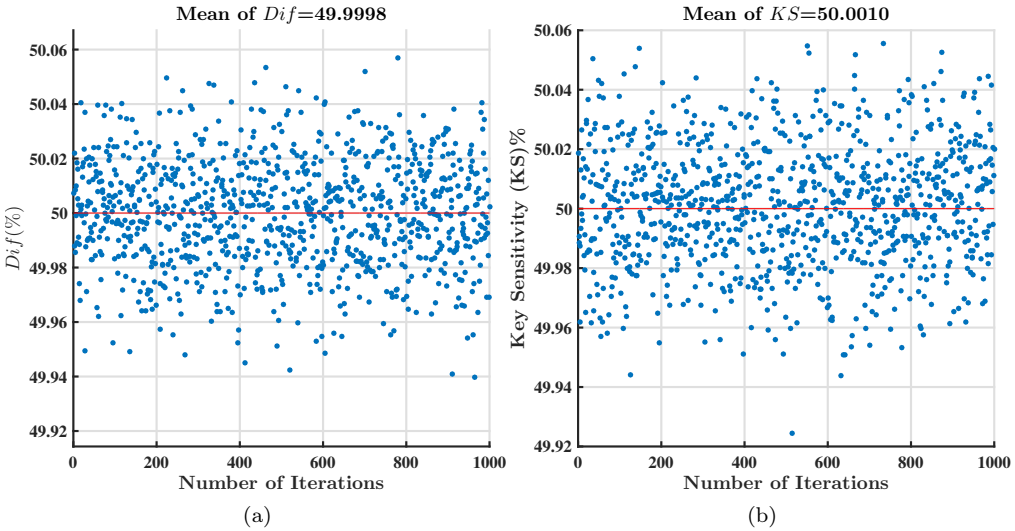
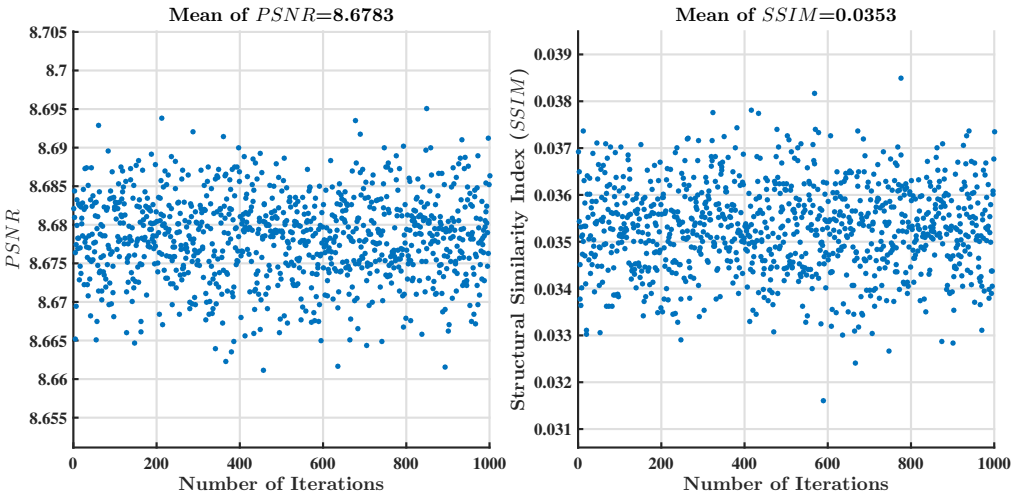


Fig. 7: (a) The different variation between plain and ciphered Lenna image (percentage of the Hamming distance) and (b) key sensitivity against 1,000 random keys.

6.4 Cryptanalysis: Resistance Against Well-known Types of Attacks

In contrast to the majority of existing cipher solutions, our scheme is based on a dynamic key approach, with dynamic substitution, permutation and diffusion layers for each input data. Previous statistical tests (entropy analysis,



(a) (b)

Fig. 8: Variation of $PSNR$ and $SSIM$ between the original and encrypted Lenna image versus $1,000$ random keys.

probability density function, correlation tests) have confirmed the robustness of the proposed cipher scheme and its high resistance against statistical attacks. Moreover, the key sensitivity analysis demonstrated a high sensitivity against key-related attacks. These results are sufficient to infer that no useful information can be inferred from the encrypted data. On the other hand, the resistance against chosen/known plain-text attacks is verified due to the dynamic key approach, which drastically complicates the attacker's task. As such, the problems of a single message failure and accidental key disclosure are avoided. Furthermore, differential and linear attacks are ineffective since any change in the dynamic key leads to a significant difference in the produced cipher primitive and in the encrypted message as well. Also, the key space of the secret key is of the order of 2^{128}, 2^{192} or 2^{256}, which is sufficiently large to make brute-force attacks unfeasible. The same is true for the key space of the dynamic key, which is 2^{512}. Note that a large secret key and a large dynamic key are being used since the difficulty of cipher-text-only attack is equivalent to one of the brute force attacks, making it impossible for a cipher-text-only attack to retrieve useful information from the cipher image.

6.5 **Statistical tests with TestU01**

As previously explained, ESSENCE has been tested with more than 100 seeds via TestU01 [21], and it successfully passed all the tests. In practice, a message of size 512*512 is typically used with all elements set to zero, and the key is initialized only once, at the beginning. All the other variables are also

initialized once. Since TestU01 uses many pseudo-random numbers, the same message is used repeatedly over a very large number of iterations, with a single difference between iterations, the different numbers generated by the PRNGs.

7 Performance Analysis

In this section, the cipher latency is quantified to assess the performance of the proposed cipher.

7.1 Experiments

To measure the performance of the proposed cipher, ESSENCE is evaluated on a Titan X GPU, which has the following characteristics:

- Compute capability: 5.2
- Global memory: 12,207 MB
- GPU frequency: 1.25 GHz
- Memory frequency: 3,505 MHz
- Number of CUDA cores: 3,072

and on a Tesla V100 with the following characteristics:

- Compute capability: 7.0
- Global memory: 16,152 MB
- GPU frequency: 1.53 GHz
- Memory frequency: 877 MHz
- Number of CUDA cores: 5,120

To compare the performance against the best AES implementation, we selected the implementation of [16], which uses the ECB operation mode, and we shall refer to it as AES-ECB. The performance tests are based on different 8-bit color images. Note that the throughput of AES-ECB is very close to the result in [19], 570.72 Gbps, which corresponds to 71.3 GBps.

Table 2: Throughput of ESSENCE and AES-ECB on a Titan X GPU

Image size	ESSENCE Throughput (in GB/s)	AES-ECB Throughput (in GB/s)
512x512x3	35.1	20.3
1024x1024x3	71.5	36.6
2048x2048x3	105.7	52.1
4096x4096x3	115.7	58.3
8192x8192x3	108.6	65.8
16384x16384x3	110.6	70.2

The execution time of the encryption algorithm is the same as the one of the decryption algorithm (stream cipher). Note that our implementation is

Table 3: Throughput of ESSENCE and AES-ECB on a Volta V100 GPU

Image size	ESSENCE Throughput (in GB/s)	AES-ECB Throughput (in GB/s)
512x512x3	53.5	22.9
1024x1024x3	150.5	54.1
2048x2048x3	261.1	91.0
4096x4096x3	354.4	120.0
8192x8192x3	358.8	136.9
16384x16384x3	372.8	146.1

highly optimized, and the kernel operations of reading and writing an image take approximately the same time. The speed-up of ESSENCE compared to AES-ECB is shown in Tables 2 and 3, and in Fig 9.

The obtained results indicate that the proposed cipher scheme is faster compared to AES-ECB, and the ratio varies between 1.4 and 2 depending on the message length on the Titan X, and between 2.4 and 2.9 for the Tesla V100. Therefore, the proposed cipher scheme is more suitable for real-time applications.

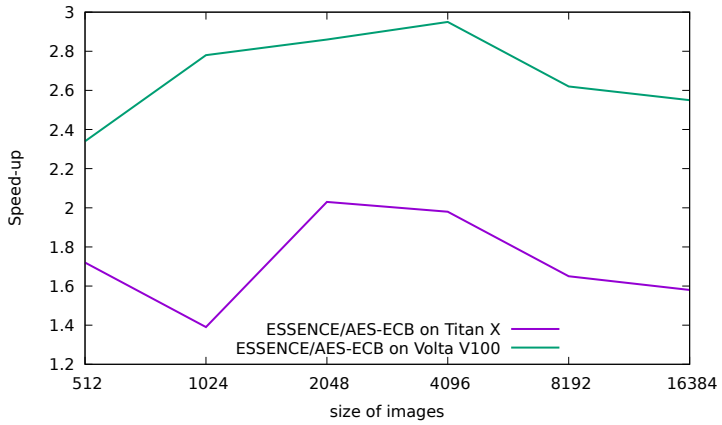


Fig. 9: Speed-up (execution time ratio) of ESSENCE compared to AES-ECB on a Titan X GPU and on Tesla V100

8 Conclusion

In this paper, we presented ESSENCE, a new dynamic, key-dependent, one-round stream cipher scheme with an efficient, parallel, and dynamic key-dependent structure, and which was designed targeting a GPU implementation. ESSENCE outperformed the most optimized implementation of AES

on GPU, which makes it preferable for real-time applications. Moreover, the proposed cipher scheme offers a high degree of randomness, which was validated by quantifying the produced key-stream, which successfully passed the statistical tests of TestU01. Also, ESSENCE has a high periodicity since it combines the threads' results of two PRNGs, which are then dynamically xored based on 32 permutation tables, which are also generated and related to the dynamic key. Moreover, the implementation of ESSENCE is very simple compared to other existing cipher schemes. Equally important, the robustness of ESSENCE has been assessed and confirmed via cryptanalysis along with different benchmark tests. Note that other existing cryptanalysis techniques are designed to target static structures, which is not the case of the proposed scheme. In future work, the design of an efficient parallel dynamic key-dependent hash function for GPU will be investigated.

Acknowledgement

This paper is partially funded by the Maroun Semaan Faculty of Engineering and Architecture at the American University of Beirut and by the EIPHI Graduate School (contract "ANR-17-EURE-0002"). We also thank the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

References

1. Goutam Paul and Subhamoy Maitra. *RC4 Stream Cipher and Its Variants*. CRC Press, 2011.
2. Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.
3. Christof Paar and Jan Pelzl. *Understanding Cryptography: a Textbook for Students and Practitioners*. Springer Science & Business Media, 2009.
4. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer Science & Business Media, 2013.
5. Hassan N Noura, Mohamad Noura, Ali Chehab, Mohammad M Mansour, and Raphaël Couturier. Efficient and Secure Cipher Scheme for Multimedia Contents. *Multimedia Tools and Applications*, pages 1–30, 2018.
6. Hassan Noura, Ali Chehab, Mohamad Noura, Raphaël Couturier, and Mohammad M Mansour. Lightweight, Dynamic and Efficient Image Encryption Scheme. *Multimedia Tools and Applications*, pages 1–35, 2018.
7. Hassan Noura, Lama Sleem, Mohamad Noura, Mohammad M Mansour, Ali Chehab, and Raphaël Couturier. A New Efficient Lightweight and Secure Image Cipher Scheme. *Multimedia Tools and Applications*, 77(12):15457–15484, 2018.
8. Like Chen and Runtong Zhang. A Key-dependent Cipher DSDP. In *Electronic Commerce and Security, 2008 International Symposium on*, pages 310–313. IEEE, 2008.

9. Runtong Zhang and Like Chen. A Block Cipher using Key-dependent S-box and P-boxes. In *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, pages 1463–1468. IEEE, 2008.
10. William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Upper Saddle River, NJ, 2017.
11. Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The simon and speck lightweight block ciphers. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.
12. Jacques Bahi, Raphaël Couturier, Christophe Guyeux, and Pierre-Cyrille Héam. Efficient and Cryptographically Secure Generation of Chaotic Pseudorandom Numbers on GPU. *The Journal of Supercomputing*, 71(10):3877–3903, 2015.
13. Wai-Kong Lee, Hon-Sang Cheong, Raphael C-W Phan, and Bok-Min Goi. Fast Implementation of Block Ciphers and PRNGs in Maxwell GPU Architecture. *Cluster Computing*, 19(1):335–347, 2016.
14. Qinjian Li, Chengwen Zhong, Kaiyong Zhao, Xinxin Mei, and Xiaowen Chu. Implementation and Analysis of AES Encryption on GPU. In *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, pages 843–848. IEEE, 2012.
15. Guang-liang Guo, Quan Qian, and Rui Zhang. Different Implementations of AES Cryptographic Algorithm. In *High Performance Computing and Communications (HPCC), IEEE 7th International Symposium on Cyberspace Safety and Security (CSS)*, pages 1848–1853. IEEE, 2015.
16. Rone Kwei Lim, Linda Ruth Petzold, and Çetin Kaya Koç. Bitsliced High-performance AES-ECB on GPUs. In *The New Codebreakers*, pages 125–133. Springer, 2016.
17. Raphaël Couturier. *Designing Scientific Applications on GPUs*. Numerical Analysis & Scientific Computating. Chapman & Hall/CRC, 2013.
18. Nvidia, CUDA. A C Programming Guide, version 9.0. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.
19. Biagio Peccerillo, Sandro Bartolini, and Çetin Kaya Koç. Parallel Bitsliced AES through PHAST: a Single-Source High-Performance Library for Multi-Cores and GPUs. *Journal of Cryptographic Engineering*, pages 1–13, 2017.
20. Zeinab Fawaz, Hassan Noura, and Ahmed Mostefaoui. An Efficient and Secure Cipher Scheme for Images Confidentiality Preservation. *Signal Processing: Image Communication*, 42:90–108, 2016.
21. Pierre L'Ecuyer and Richard J. Simard. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Trans. Math. Softw*, 33(4), 2007.
22. Hassan Noura, Ali Chehab, Lama Sleem, Mohamad Noura, Raphaël Couturier, and Mohammad M Mansour. One Round Cipher Algorithm for Multimedia IoT Devices. *Multimedia Tools and Applications*, pages 1–31, 2018.

23. Andreas Klein. Attacks on the rc4 stream cipher. *Designs, codes and cryptography*, 48(3):269–286, 2008.
24. Scott Fluhrer, Itsik Mantin, Adi Shamir, et al. Weaknesses in the Key Scheduling Algorithm of RC4. Springer.
25. Itsik Mantin and Adi Shamir. A practical attack on broadcast rc4. In *International workshop on fast software encryption*, pages 152–164. Springer, 2001.
26. Chris Wellons. Finding the Best 64-bit Simulation PRNG « null program. `https://nullprogram.com/blog/21/09/2017`, September 2017.
27. François Panneton and Pierre L'ecuyer. On the xorshift Random Number Generators. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 15(4):346–361, 2005.
28. David Blackman and Sebastiano Vigna. Scrambled Linear Pseudorandom Number Generators. *CoRR*, abs/1805.01407, 2018.
29. Shujiang Xu, Yinglong Wang, Jizhi Wang, and Min Tian. Cryptanalysis of Two Chaotic Image Encryption Schemes Based on Permutation and XOR Operations. In *Computational Intelligence and Security, 2008. CIS'08. International Conference on*, volume 2, pages 433–437. IEEE, 2008.
30. Guoji Zhang and Qing Liu. A Novel Image Encryption Method Based on Total Shuffling Scheme. *Optics Communications*, 284(12):2775–2780, 2011.
31. Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image Quality Assessment: from Error Visibility to Structural Similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.
32. Quan Huynh-Thu and Mohammed Ghanbari. Scope of Validity of PSNR in Image/Video Quality Assessment. *Electronics letters*, 44(13):800–801, 2008.