ORSCA-GPU: One Round Stream Cipher Algorithm for GPU Implementation

Ahmed Fanfakh[1], Hassan N. Noura[2], and Raphaël Couturier[2]

[1]University of Babylon, Iraq
[2]Univ. Bourgogne Franche-Comté (UBFC), FEMTO-ST Institute, CNRS, Belfort, France

Abstract

Data confidentiality is one of the most critical security services. Many encryption algorithms are currently used to provide data confidentiality. That is why there are continuous research efforts on the design and implementation of efficient cipher schemes. For this purpose, different lightweight cipher algorithms have been presented and implemented on GPUs with different optimizations to reach high performance. Some examples of these ciphers are Speck, Simon which both require less latency compared to Advanced Encryption Standard (AES). However, these solutions require a higher number of rounds but with a more simple round function compared to AES. Therefore, in this paper, a new cipher scheme called "ORSCA" is defined which only requires one round with the dynamic key-dependent approach. The proposed cipher is designed according to the GPU characteristics. The proposed one-round stream cipher solution is suitable for the high data rate applications. According to the performance results, it can achieve high data throughput compared to existing ones, with throughput greater than 5 Terabits/s on a Tesla A100 GPU. Thus, this approach can be considered as a promising candidate for real-time applications. Finally, the security level is ensured by using the dynamic cryptographic primitives that can be changed for each new input message (or for a set of messages: sub-session key). Thus, the proposed solution is a promising candidate for high secure GPU cryptographic algorithms.

Keywords— One round GPU stream cipher solution; Security and performance analysis; Parallel computing; Dynamic key dependent cryptographic primitives

1 Introduction

Data security is facing increasing challenges with new security attacks that benefit from the advancement of the attackers computational power. Data security attacks can be either active or passive. The passive attacks can seriously compromise the data

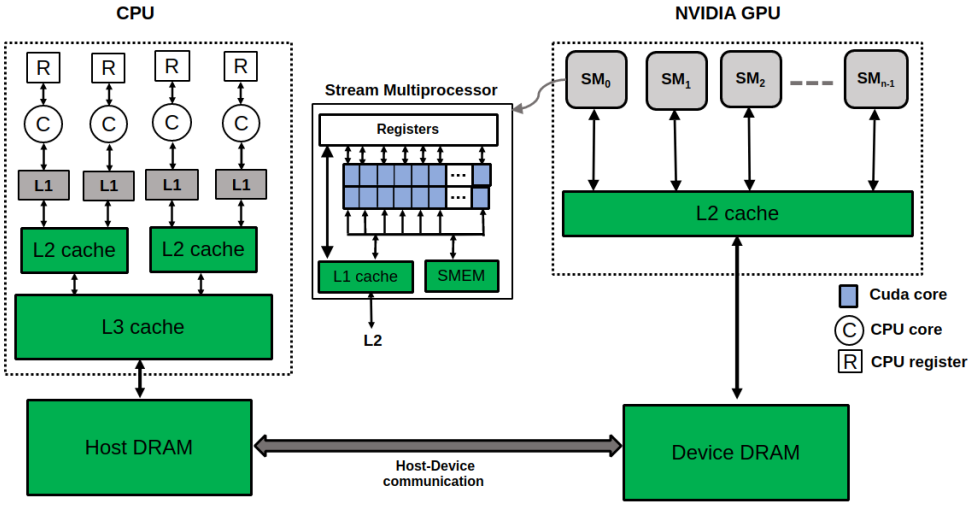
confidentiality, while active attacks can compromise the data authentication, integrity, and availability. Active attackers have the ability to insert, remove, and alter the data content, while passive attackers just intercept the communicated data. Passive attacks are more difficult to detect but they should be taking into account to preserve the data confidentiality.

Data confidentiality can be ensured in general by using cryptographic algorithms, that can be divided into two main classes: symmetric and asymmetric. The symmetric cryptographic algorithms are more efficient in terms of memory and computational overhead compared to the asymmetric ones. Furthermore, symmetric cryptographic algorithms can be block or stream cipher based. Generally, a block cipher [1] uses a round function that can be based on Feistel Network (FN) like Data Encryption Standard (DES) or Substitution-Permutation Networks (SPN) like Advanced Encryption Standard (AES) [2]. Currently, AES is widely used because it is more secure and efficient compared to DES, as indicated in [3]. In addition, SPN lends itself to parallel implementation and requires fewer rounds than FN. Besides, SPN or FN implementations use the concept of multi-rounds, where a round function is iterated for r rounds, which presents a high computation overhead.

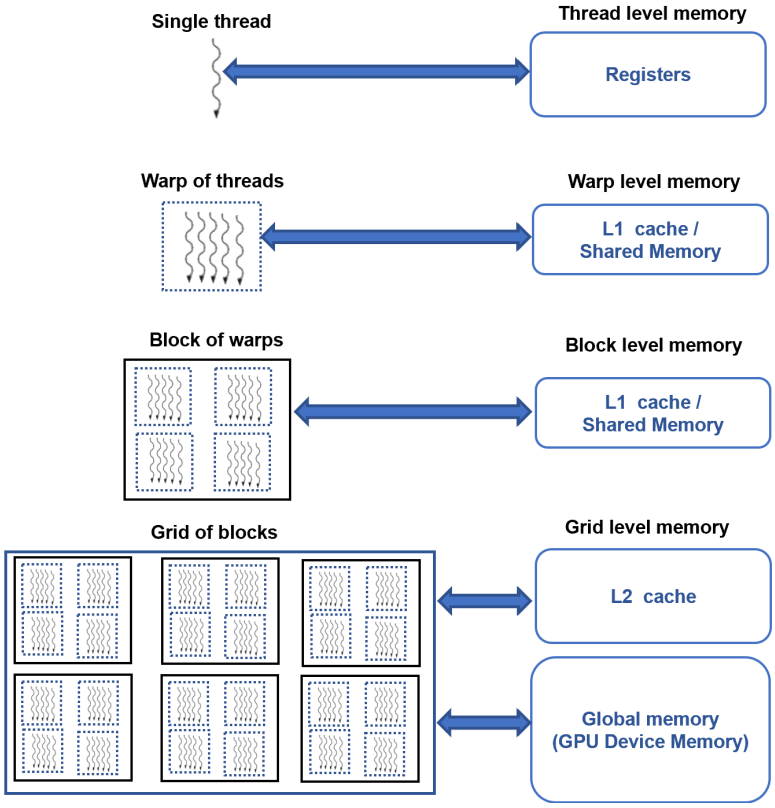
Each generation of the GPU architecture adds new hardware capabilities that boost the computational capability of the GPU. One of the most significant characteristics is the expansion of the GPU's computational cores. Additionally, the computational capabilities per GPU core remain somewhat limited due to the massive parallelism embedded in the GPU device's architecture. However, existing cryptographic algorithms that employ many rounds per core (thread) restrict them from reaching their best performance.

Therefore, in order to solve this challenge and to reach maximum performance, one need to redesign cryptographic algorithms according to the optimized GPU features. Therefore, in this paper, a new lightweight stream cipher scheme with parallel structure is presented which only requires one round with simple operations. The proposed solution is designed according to GPU features to be implemented in an efficient manner on GPU. Therefore, its GPU implementation can be optimized and can achieve high throughput and speedup compared to existing recent cipher algorithms. Therefore, the proposed stream cipher solution with GPU implementation can ensure data confidentiality with minimum latency overhead and consequently can respond better to real-time applications. This work is presented to indicate that GPU can be used to make symmetric cryptographic algorithms more secure and robust.

The rest of the paper is organized as follows: Section 3 describes the necessary background for GPUs and the lightweight cryptographic algorithm implemented over GPUs. Then, the proposed one-round stream cipher algorithm "ORSCA" for GPU implementations is detailed and described in Section 4. The robustness of the proposed stream cipher scheme is described in Section 5. In addition, several performance tests were done and described in Section 6 to confirm the efficiency of the proposed solutions in terms of throughput and speed-up. Finally, a conclusion and future work are presented in Section 7.



(a) CPU-GPU functionality



(b) NVIDIA memory hierarchical

Figure 1: CPU and NVIDIA GPU architecture (a) and NVIDIA GPU memory hierarchical level (b)

2 Related Work

Recently, to achieve better performances, new counter (CTR)-mode-based symmetric ciphers have been designed to run on the Graphics Processing Unit (GPU). This leads to better throughput compared to the CPU implementation. As GPUs are very efficient for parallel computing. Thus, the CTR based encryption approach can take advantage of the hundreds or thousands of core components of a GPU to reduce the cipher execution time. Researchers have already used GPUs for pseudo-random numbers generation, as in [4, 5]. Furthermore, AES has also been implemented on GPUs [6, 7, 8], yielding an impressive speedup [9] over the CPU implementation. Note that the efficient GPU implementation of an algorithm requires making the best use of the GPU architecture in terms of shared memory, registers, and warp [10].

An optimized and efficient implementation of AES on GPU has been presented in [8]. This implementation presents various optimizations over previous related work and thus is considered it as a reference for comparison with our proposed solution. Another recent implementation of AES on GPU, PHAST, has been described in [11]. This implementation in a decrease of approximately 10% of the performance compared to [8].

In addition, another implementation of AES-128 is presented in [12]. This implementation was executed with a RTX 2070 GPU and reaches 878.6 Gbps throughput thanks to an improvement that removes bank conflicts in shared memory accesses. However, the existing ciphers require a higher number of rounds (r) since the fixed cryptographic primitives concept is used [13, 14]. These ciphers are vulnerable to a range of analytical attacks if they are configured with a low number of rounds.

To that purpose, researchers are focusing on developing lightweight round functions that use basic operations, and by reducing the number of rounds by employing a dynamic key-dependent cryptographic approach [15, 16, 17, 18, 19, 20, 21]. For example, a set of the approaches such as [22, 16, 20] requires two rounds and use the chaining operation mode, but the one presented in [15] needs only a single round, and it processes two blocks at once.

This work is presented to improve the last research work on the dynamic cryptographic approach, which is described in [21]. We propose the development of an efficient and resilient stream cipher that can benefit from the parallel computing capability of GPU devices. Compared to [21], different cryptographic primitives are used. In fact, the proposed solution uses a different technique to produce the dynamic cryptographic primitives (without a chaining operation). Additionally, the proposed cipher solution avoids the use of the permutation operation to reach best performance. In addition, in terms of number of operations, the proposed solution uses only one round function without the need of an update function. Therefore, only one substitution and one PRNG operations are required in the proposed solution compared to two in the previous dynamic-key based solution.

In fact, the required latency and resources of existing ciphers depend on the round function structure (confusion and diffusion operations) and the number of rounds r. Therefore, the best performance can be reached by reducing the number of rounds and simplifying the round function. In addition, any efficient cipher scheme should

preserve a high level of security and performance at the same time, which is a hard challenge.

3 Background

This section presents background knowledge related to the concept and characteristics of the stream cipher and GPUs. On the other hand, the list of notations used in this paper is introduced in Table 1.

Table 1: Notations table

Symbol	**Definition**
SK	Shared secret session key
$nonce$	A dynamic Nonce that changes per input message
DK	Dynamic key updated per input message
k_{S1} and k_{S2}	Substitution sub-keys
S_1 and S_2	Substitution tables produced by using k_{S1} and k_{S2}, respectively.
$S(m,\ S_1,\ S_2)$	Substitute the bytes of m with odd and even indices by using first substitution table S_1 and S_2, respectively.
K_{Seed}	the seed sub-key and it is used to produce N seeds.
len	Length of input message
$\lceil x \rceil$	Rounds x to the nearest integer above its current value
$nbth$	Blocks number per input message and is equals to $\lceil \frac{len}{size} \rceil$
$size$	Number of words keystream produced per thread
M	Original message
$nbth$	Total number of threads
N	Length of initial seeds
m_i	i^{th} original plain block
C	Encrypted message
c_i	i^{th} encrypted block
KSA	Key Setup Algorithm of RC4

3.1 GPU Characteristics

The GPU is a processing technology often used to accelerate computations. Nowadays, GPUs are used in a wide range of computing applications and systems, including smart phones, embedded computing, and supercomputers. The GPU architecture is very different from the CPU one. The design of a GPU is tuned to increase the execution throughput of multiple concurrent threads. The number of computing cores within a GPU can range from hundreds to thousands of cores. The hardware is built to handle many threads, even if memory access could be the bottleneck. To make use of the GPU computational capacity, a GPU code needs to use more threads than the number of cores. As a result, while some threads are waiting for data, other threads can compute.

NVIDIA GPUs consist of several Stream Multi-processors (SMs) as in Figure 1.a. According to the GPU type, some characteristics of the SMs may vary. SMs typically

consist of 32 cores which can only carry out one instruction at a time. So, if two threads with two different instructions are run on the same SM, only one instruction gets executed while the second one needs to wait. This is known as threads divergence. All conditional instructions, like "if" and repetitive instructions such as "for" or "while", must thus be prevented as far as possible. A set of 32 threads is executed on an SM. This set is called a warp. Moreover, threads are structured into blocks. The maximum number of threads for each block is restricted to 1,024 according to the GPU architecture.

Typically, a GPU has many types of memory: global memory, which is the slowest; cache memory; texture memory; shared memory; local memory; and a limited set of registers with the quickest access. Figure 1.b presents the memory hierarchy in NVIDIA GPU devices as follows:

- Registers: These are thread-private, which implies that thread-specific registers are not accessible to other threads. The compiler makes judgments on the register use.
- L1/Shared memory (SMEM): Each SM includes an on-chip memory that may be utilized as both the L1 cache and a shared memory. The L1 cache is privately accessible by threads while all threads in a CUDA block can use the shared memory simultaneously.
- L2 cache: It is shared among all SMs and it is accessible to every thread throughout each CUDA block.
- Global memory: This refers to the size of the GPU's frame buffer and the DRAM included within the GPU.

3.2 Dynamic key generation method

The proposed solution is based on the dynamic key dependent approach, where a Dynamic Key DK is used to produce a set of dynamic cryptographic primitives (substitution tables in addition to a set of N seeds, where each seed can be a word of 32 or 64 bits). This dynamic key Dk is obtained by hashing the output of "exclusive or" between a shared secret key (session key SK) and a $Nonce$ that should be updated and unique for each message. This operation is done as illustrated in Figure 2 and described in the following equation:

$$Dk = hash(SK \oplus Nonce) \tag{1}$$

Any secure cryptographic hash function can be used in this step to avoid collisions. We use the SHA-512 as a secure cryptographic function to obtain a dynamic key with 512 bits length. Then, this dynamic key Dk is divided into three sub-keys, where each one of the first two sub-keys (KS_1 and KS_2) has 128 bits length and the third one (K_{Seed}) has 256 bits length.

The description of these sub-keys are presented in the following:

1. KS_1 is the first substitution sub-key and it represents the first 128 Least Significant Bits of DK. This sub-key is used to produce the first substitution table S_1. Any method to produce dynamic-key dependent substitution tables can be used in this step. For example, the Key Setup Algorithm (KSA) of RC4 was used in [15] to construct dynamic substitution tables. In this step, we also use the KSA algorithm of RC4.

2. KS_2 is the second substitution sub-key and it represents the second 128 Least Significant Bits of DK. This sub-key is used to produce the second substitution table S_2 by using the same method uses with KS_1, which is the KSA of RC4.
3. K_{Seed} represents the first 256 Most Significant Bits (MSB) of DK. This sub-key is used as a secret seed with any PRNG to produce a keystream of length N words, where words can have a length of 32 or 64 bits. Therefore, N seeds are obtained from K_{Seed}. Each thread will select one of these produced seeds that are generated in a dynamic pseudo-random manner.

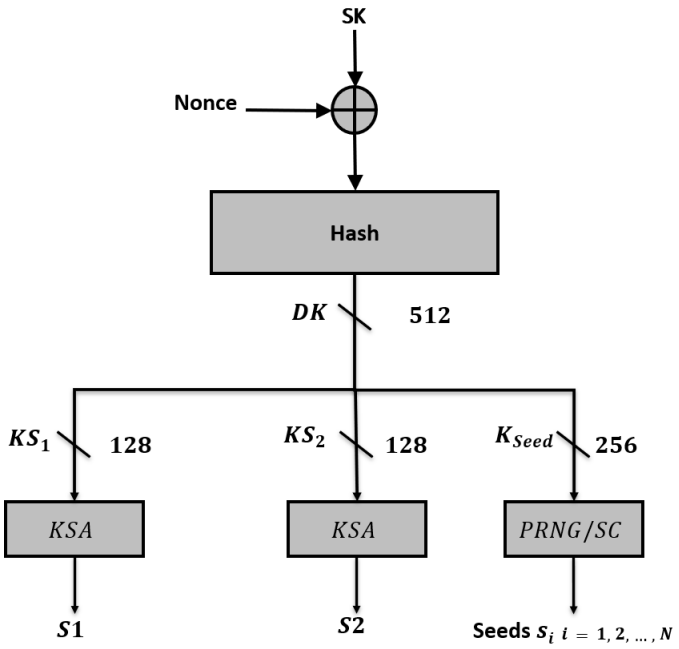


Figure 2: The proposed dynamic key generation and construction cryptographic primitives

Besides, the construction of these dynamic cryptographic primitives are done at the CPU and copied to GPU.

4 Proposed Stream Cipher Scheme

The same keystream block generation procedure is applied on all the threads but with different input seeds (see Figure 3). Each thread selects a seed according to its thread *id* and uses it to produce a given number of words (chosen by the user). This number is denoted *size* in the following. This procedure is based on repeating the round function for *size* times. The main steps of the proposed keystream generation process are described in the following:

1. Iterate the PRNG Splitmix64 with the seed x;

2. Substitute the bytes of the produced word by using two substitution tables that are stored in the shared memory;
3. The output of this iteration becomes the seed for the next iteration and it is considered one of the produced word keystream.

In fact, the round function will be repeated for *size* iterations in a recursive manner. In each iteration, a word keystream is produced and "exclusive or" with a word of the plaintext/ciphertext to obtain ciphertext/plaintext, respectively. Therefore, each thread will be responsible to encrypt/decrypt *size* words. Let us indicate that the generation of the keystream for each thread follows the chaining operation mode.

The entire data stream (plaintext/ciphertext) *in* will be converted into words. We consider that the length of plaintext/ciphertext is *nbth* words that can be encrypted/decrypted in parallel by using the proposed one round stream cipher. The data stream is divided into $N = \frac{nbth}{size}$ blocks, and each block will have *size* words. Moreover, the GPU device has many computing threads that work in parallel. Thus, a high number of threads can be used to produce the keystream and consequently encrypt/decrypt the plaintext/ciphertext, respectively. The encryption/decryption steps to

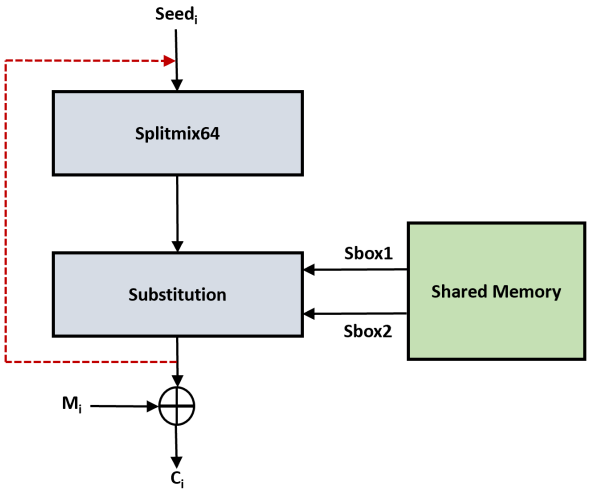


Figure 3: The proposed one round function of ORSCA stream cipher (each thread has a different seed in the GPU implementation)

encrypt (or decrypt) the i^{th} ciphertext block of *size* words, c_i, are illustrated in Figure 3 and described in detail in Algorithm 1. In the following, we describe the different phases of the GPU implementation of the proposed one round function (Kernel).

4.1 Parallel kernel function

The proposed technique to generate the keystream (of *size* words) at each thread is presented in Algorithm 2. In this algorithm, the output ciphertext/deciphertext data stream is stored in the *out* vector and the input data stream in the *in* data vector. These vectors have words elements of 64 bits. In addition, for constant variables, we

Algorithm 1 ORSCA Stream Cipher

Input: Plain-text Message M;
Substitution tables (S_1 and S_2);
N seeds: $Seeds = \{Seed_1,\ Seed_2,\ \dots\ ,\ Seed_N\}$
Output: Cipher-text C

procedure ENCR($M,\ S_1,\ S_2,\ Seeds$)
  $M = m_1\ , m_2,\ \dots\ , m_N$
  for $i = 1 \rightarrow N$ **do**
    $X \leftarrow Seed_i$
    for $j = 1 \rightarrow size$ **do**
      $X \leftarrow Splitmix64(X)$
      $X \leftarrow Substitution(X,\ S_1,\ S_2)$
      $RK_{i,j} \leftarrow X$
    $c_i = m_i \oplus RK_i$
  $C \leftarrow c_1||c_2||\dots||c_N$

use the $\_\_restrict\_\_$ keyword is used to allow the compiler to minimize the access time for these variables. This keyword is likewise used for other unaltered variables when the algorithm is executed. Initially, two substitution tables are used ($S1$ is called $sbox1$ and $S2$ is called $sbox2$ in the following code and each one has a size of 256 bytes length). In addition, they are stored in the shared memory to improve the access speed of parallel threads. In this case, we should use the $\_\_syncthreads()$ instruction to synchronize threads in separate warps. However, the thread synchronization introduces overhead, which consequently reduces the performance. Therefore, to maximize the GPU's occupancy and consequently to achieve the best performance, each thread should produce $size$ keystream words per GPU thread. Thus, the number of threads decreases by increasing $size$ and the best thread granularity can be reached. In the proposed implementation, $nbth$ represents the total number of threads, and it depends on the data size to be encrypted/decrypted.
In more details, the structure of the proposed round function consists of two main operations (PRNG iteration and byte substitution) for each iteration j, where j varies between 0 and $size - 1$. More details are described in the following steps:

1. Iterate an efficient PRNG with a unique seed for each thread that are stored in a variable $v2$. An example of a possible PRNG that can be employed is the splitmix64.
2. The produced output of each PRNG iteration is a word and will be stored in $v2$ (input-output variable). Then, $v2$ will be converted to a set of bytes. For example, for word length of 64 bits, 8 bytes are obtained.
3. Then, the produced output is substituted using two substitution tables ($S1$ and $S2$). The bytes with odd indices are substituted by using the first substitution table $S1$ and the even ones are substituted with the second substitution table $S2$. Two substitution tables are used at this step to increase the security level.
4. Then, a corresponding input plaintext/ciphertext word is XORed with the produced substituted output to provide ciphertext/deciphertext word, respectively.
5. For the next iteration, the seed is updated to be equal to the produced sub-

stituted word. This means that the produced keystream is done in a chaining mode, where the output of the previous iteration becomes the input of the next iteration.

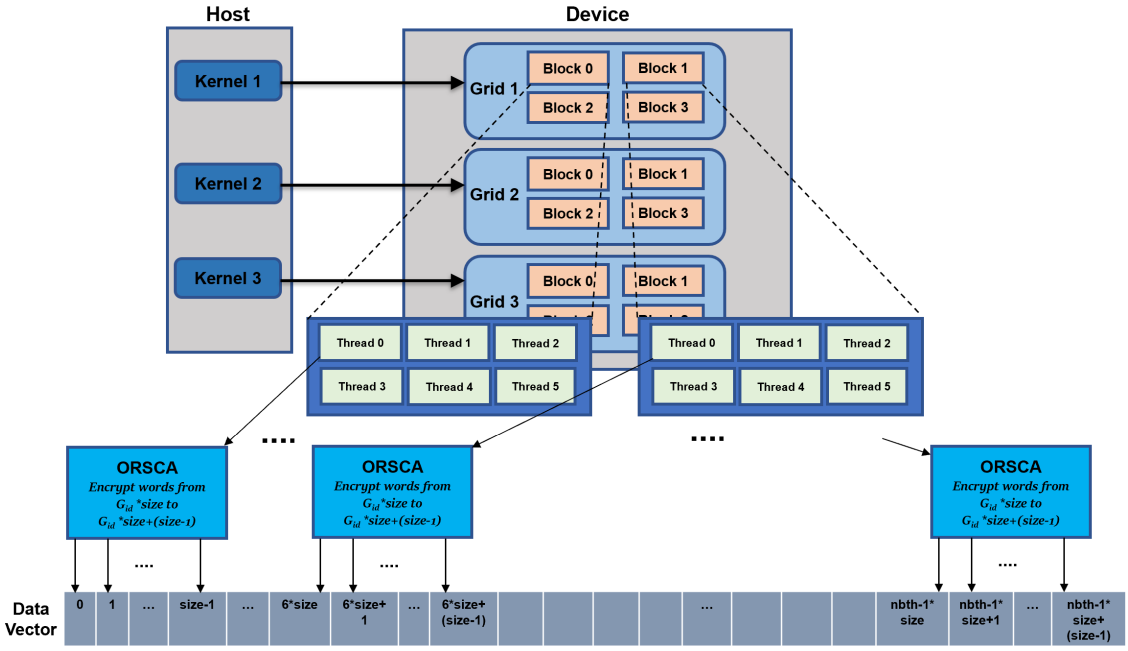


Figure 4: Graphical illustration of the ORSCA stream cipher implementation with GPU

For a further illustration of how ORSCA works, Figure 4 gives a graphical example. It shows that a CUDA block is a set of threads, and a grid of CUDA blocks is used to organize them. Each CUDA block is executed by a single streaming multiprocessor (SM) on the GPU. On the host side, before a kernel is allocated to a grid of thread blocks, input vectors must be copied from host to device. According to the communication overhead between host and device, the size of the initial vector v is reduced by a factor of $\frac{1}{size}$ compared to the plain text size. However, to manipulate a data vector in parallel by a kernel function, in the beginning, the global index G_{id} must be computed by each thread, $G_{id} = blockidx.x \times blockDim.x + threadidx.x$. Then, the algorithm encrypts/decrypts (per thread) a number of words starting from the index $G_{id} \times size$ to the $G_{id} \times size \times (size-1)$ of data vector. Accordingly, the proposed algorithm is portable enough to work with all NVIDIA GPU devices without any further modifications.

4.1.1 Splitmix64

Splitmix64 PRNG uses arithmetical (addition and multiplication) and logical (xor and rotation) operators [23]. Algorithm 3 presents the steps of the splitmixt64. Splitmix64 is not a secure PRNG but it is selected since it is efficient (low execution time) and can

Algorithm 2 ORSCA kernel function

```
__global__
void encrypt(uint64_t*  __restrict__ v, const uint64_t* __restrict__ in, uint64_t* out,
const uchar* __restrict__ box1, const uchar* __restrict__ box2 ,int size, long nbth)
{

 int    Gid=blockIdx.x*blockDim.x+threadIdx.x;
  if(threadIdx.x<256){
    sbox1[threadIdx.x]=box1[threadIdx.x];
    sbox2[threadIdx.x]=box2[threadIdx.x];
  }
  __syncthreads();
  if (Gid<nbth)
  {
   uchar* tt;
   uint64_t v2=v[Gid];
   for(int j=0;j<size;j++)
    {
     v2=splitmix64_stateless(v2);
     tt=(uchar*)&v2;
     tt[0]=sbox1[tt[0]];
     tt[1]=sbox2[tt[1]];
     tt[2]=sbox1[tt[2]];
     tt[3]=sbox2[tt[3]];
     tt[4]=sbox1[tt[4]];
     tt[5]=sbox2[tt[5]];
     tt[6]=sbox1[tt[6]];
     tt[7]=sbox2[tt[7]];
     out[Gid+j*nbth]=v2^in[Gid+j*nbth];
    }
   v[Gid]=v2;
  }
}
```

be implemented easily. The security level of the proposed stream cipher is based on the use of dynamic cryptographic primitives in addition to having high key space. In

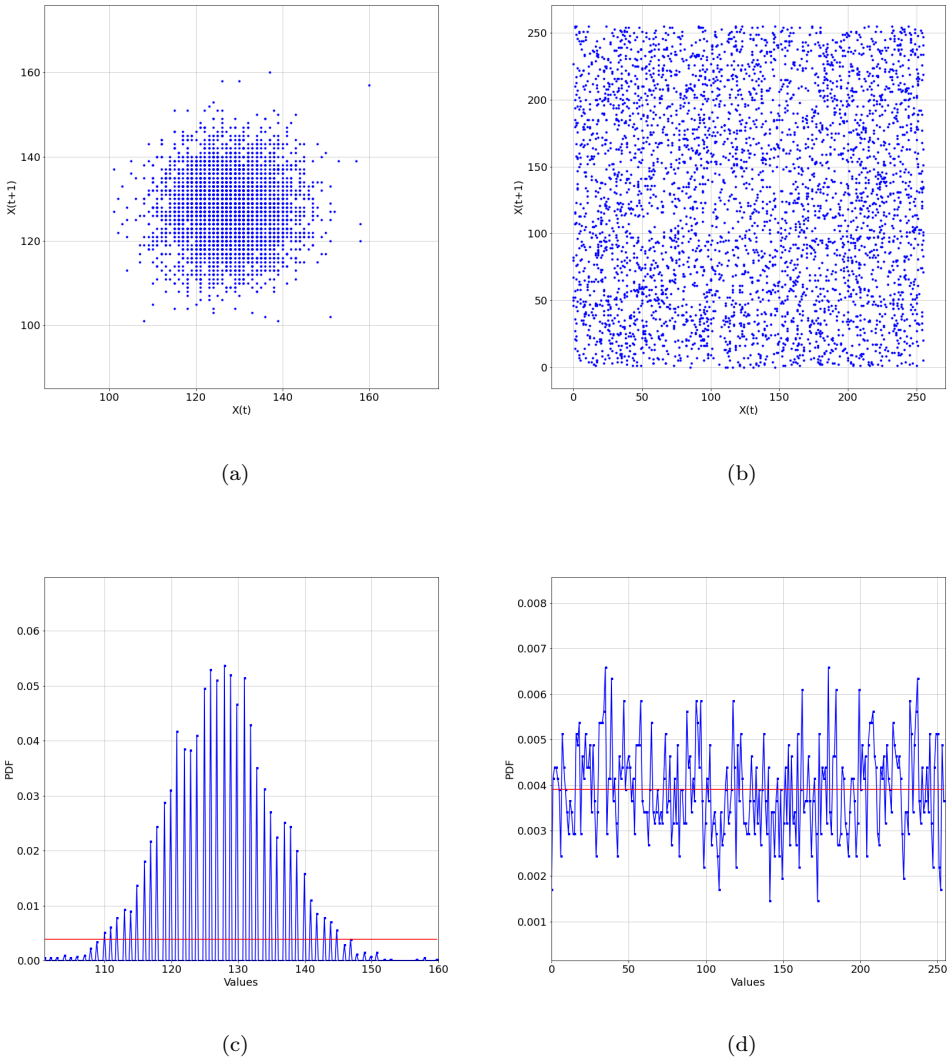
Algorithm 3 Splitmix64 function

```
__device__
uint64_t splitmix64_stateless(uint64_t x) {
  uint64_t z = (x + UINT64_C(0x9E3779B97F4A7C15));
  z = (z ^ (z >> 30)) * UINT64_C(0xBF58476D1CE4E5B9);
  z = (z ^ (z >> 27)) * UINT64_C(0x94D049BB133111EB);
  return z ^ (z >> 31);
}
```

the following section, several security and performance tests are included to validate the robustness and effectiveness of the proposed one round stream cipher compared to other existing ones.

5 Security Analysis

Well-known attacks such as statistical, linear, differential, or brute force attacks [15, 24] are used to test the safety and security of a proposed cipher scheme. This section performs extensive experiments to prove the robustness of the proposed cipher scheme. Note that the proposed encryption scheme can be used for any data type, but in the following, only the results for multimedia contents are provided.



(a) (b)

(c) (d)

Figure 5: The recurrence of the original message (a), ciphertext (b) produced by using the proposed stream cipher for a random dynamic key. The corresponding PDF of the original message (c) and of the produced ciphertext (d).

5.1 Ciphertext Statistical Analysis tests

To be considered as secure against statistical attacks, a cipher must exhibit two essential properties, which are randomness and uniformity [25]. To check the randomness degree, the following statistical security tests are performed: (a) Probability Density Function (PDF) analysis, (b) entropy analysis, and (c) correlation between original and encrypted messages.

5.1.1 Uniformity Analysis

The most significant test is the probability density function (PDF) of the encrypted message, which must be uniform. Each symbol in the produced ciphertext has a probability occurrence close to $\frac{1}{n}$, where n is the number of symbols. The original PDF and their corresponding encrypted messages are shown in Figure 5. The PDFs of the encrypted messages can be considered to be close to the uniform distribution, with a value close to 0.039 ($\frac{1}{256} = 3.9 \times 10^{-3}$) for all ciphertext symbols.

5.1.2 Information Entropy Analysis

The information entropy of a given message M, is a metric that measures the level of uncertainty of a random variable [26], and can be defined as:

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

Entropy is expressed in bits, and $p(m_i)$ is the probability of occurrence of symbol m_i, and NS is the total number of symbols. The entropy of the ciphertext, if equal or close to $log_2(NS)$, can be interpreted as a true random source with a uniform distribution.

The analysis of the entropy of the ciphertext (encrypted messages) at the submatrix level with a dimension of 16× 16 (256 elements), and by using a random dynamic key, is shown in Figure 6. Furthermore, the obtained results indicate that the produced ciphertexts have entropy equals to the desired value, which is 8. Thus, the proposed cipher scheme is sufficiently secure against any given entropy attack.

5.1.3 Statistical tests with TestU01 and Practrand

As explained earlier, the proposed stream cipher was tested with 100 seeds using TestU01 [27] and Practrand [28], and it successfully passed all the tests. Practically, a message of size 512*512 is simply used with all elements set to zero, and the key is initialized only once, at the beginning. All other variables are also initialized once. The obtained ciphertexts were tested using the PractRand and TestU01 statistical tests, which are considered as the most difficult ones. These tests can verify if the generated key-stream satisfies the appropriate randomization and uniformity properties.

5.1.4 Independence

The elimination of any correlation between the elements sequence is essential to ensure the robustness of the proposed encryption scheme [15]. If the correlation coefficient is close to zero, it means that the cipher scheme has a high degree of randomness. The

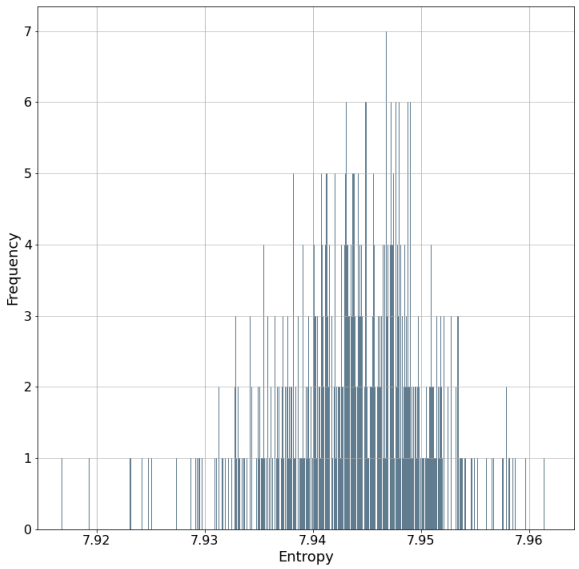


Figure 6: Entropy analysis of produced ciphertext versus 1,000 random secret keys at the sub-matrix level. Ciphertext is divided into a set of sub-matrices of size 16×16 and $NS = 256$ bytes (mean equals to 7.175).

correlation test is done by randomly picking adjacent pixels from an original message and its corresponding encrypted ones. The correlation can be done in the horizontal, vertical and diagonal directions. The correlation coefficient r_{xy} is calculated by using the following equation:

$$r_{xy} = \frac{cov(x,y)}{\sqrt{D(x) \times D(y)}} \tag{3}$$

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 results of the correlation test between original and encrypted messages are provided in Figure 7 for one random key for each time, and for 1,000 random keys in

total. The results clearly show that the correlation coefficient is very low, close to 0, which confirms the randomness property of the produced ciphertext and consequently its independence. On the other hand, the ciphertext must be very different from the

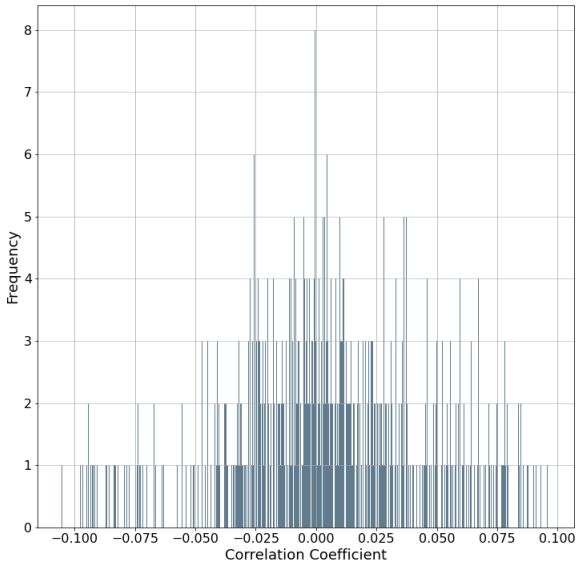


Figure 7: Probability density function of the correlation coefficient between original and encrypted messages for 1,000 random dynamic keys.

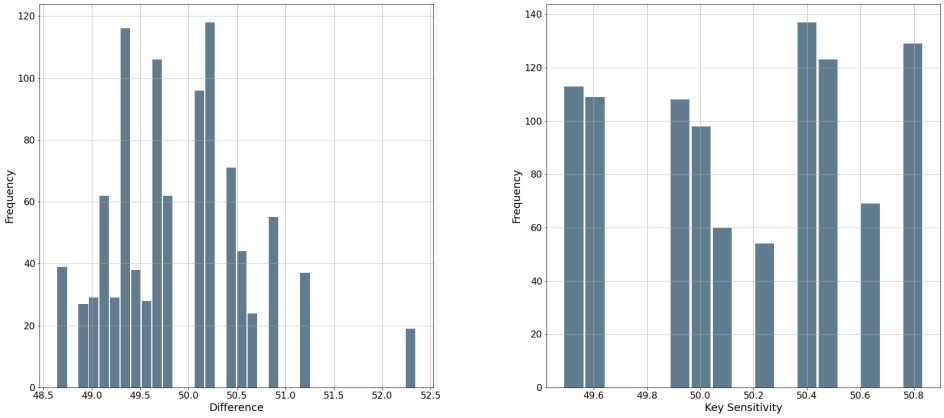
original one, with a difference of at least 50%, at the bit level. From the results shown in Figure 8-(a), the proposed cipher scheme satisfies the desired difference results, with a percentage of at least 50% between the plaintext and the encrypted ones.

5.2 Sensitivity Tests

The differential attacks focuses on analyzing the relationship between two encrypted messages resulting from a small change, such as a one-bit difference, between two original messages. The sensitivity tests must confirm that a small change in the plaintext or in the key affects the ciphertext and produces a different one. The larger the difference, the better the sensitivity of the proposed cipher scheme.

5.2.1 Key Avalanche Effect

One of the most important test is the key avalanche test. This test quantifies the sensitivity of the ciphertext (or the sensitivity of keystream in the case of stream cipher) to a slight change in the secret key. The proposed key derivation function can ensure a high level of sensitivity of the secret key and the nonce since it is based on them. To further examine the sensitivity of the dynamic key, two dynamic keys, DK_1 and DK_2, which differ by a single random bit, are used. Then, the same plaintext



(a) (b)

Figure 8: (a) The different variations between plain and ciphered messages (percentage of the Hamming distance) and (b) key sensitivity against 1,000 random keys.

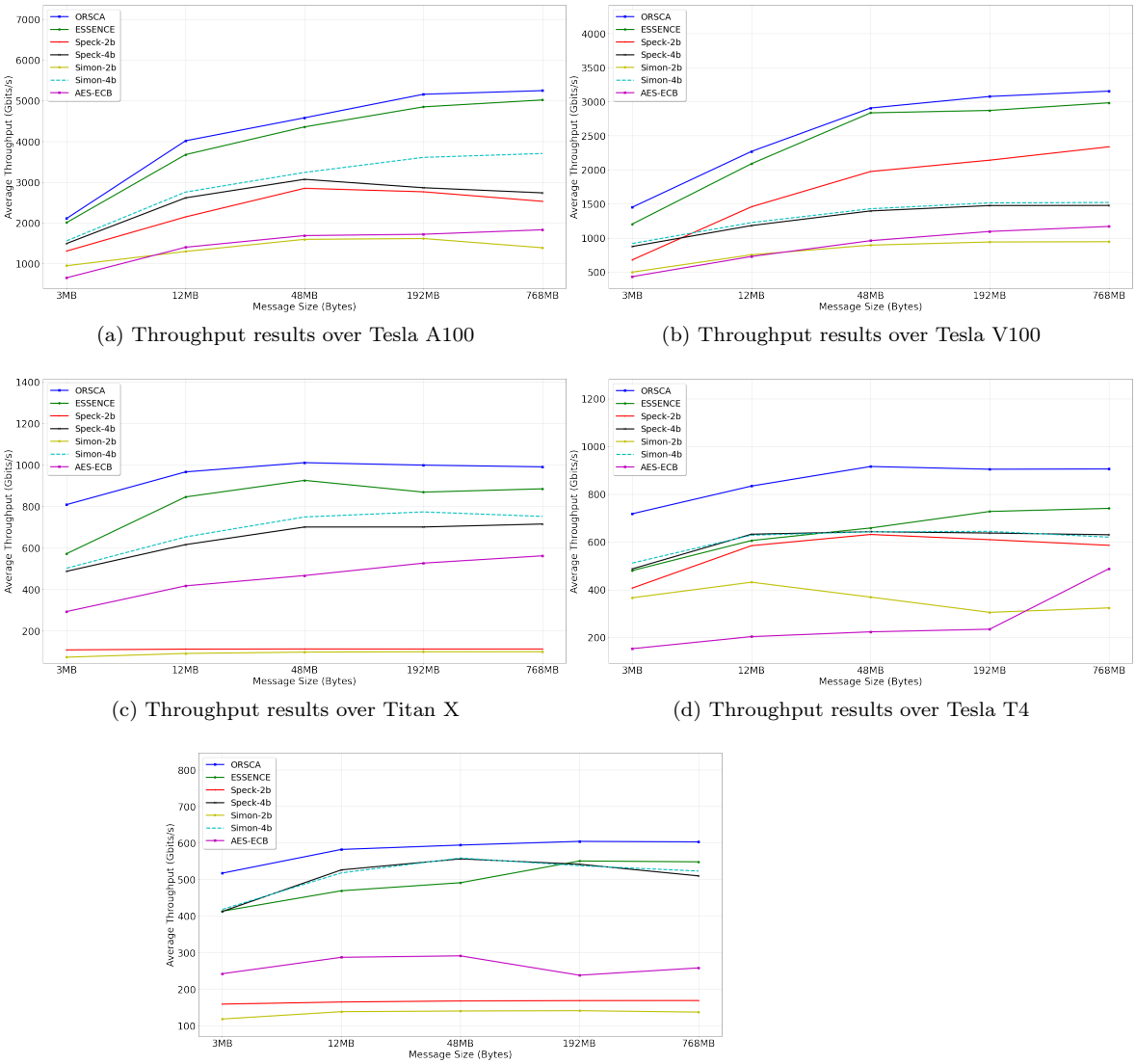
is encrypted separately, and the Hamming distance of the corresponding ciphertexts, C_1 and C_2, is computed and shown in Figure 8-(b) against 1000 random dynamic keys. It is clear that the majority of the values are close to the optimal value (50%) with a mean equals to 49.99% (see Table 2). The obtained results confirm the high key sensitivity of the proposed cipher scheme. Moreover, the obtained results are acceptable compared to those of AES.

5.2.2 Message Avalanche Effect(Sensitivity)

As a different dynamic key is used for each input message, the algorithm produces a completely different ciphertext for the same plaintext input. Consequently, the proposed cipher scheme successfully satisfies the avalanche criterion.

Table 2: Statistical results of the proposed stream cipher for 1,000 random keys.

Security Test	*Min*	*Mean*	*Max*	*Std*
Dif	48.6419	49.9029	52.3468	0.71019
KS	49.4873	50.179	50.836	0.4277
H_E	7.91	7.943	7.9618	0.00654
ρ	-0.1051	0.00499	0.0966	0.038



(a) Throughput results over Tesla A100

(b) Throughput results over Tesla V100

(c) Throughput results over Titan X

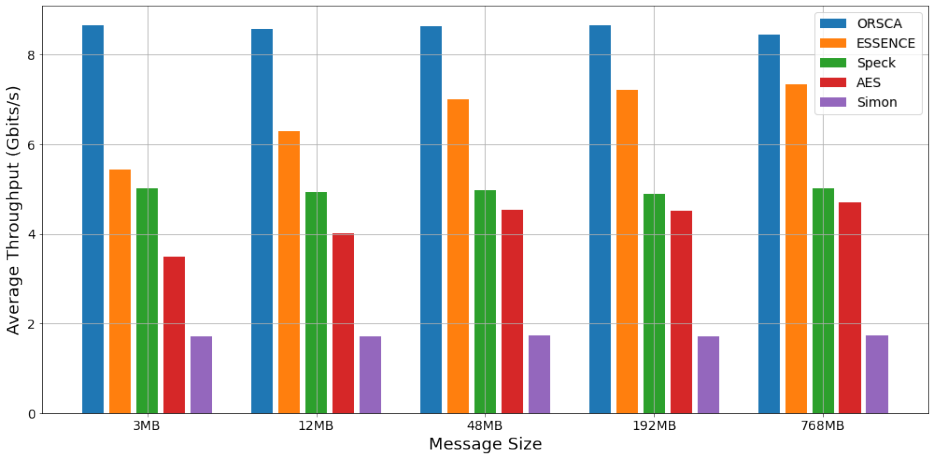
(d) Throughput results over Tesla T4

(e) Throughput results over GTX1060

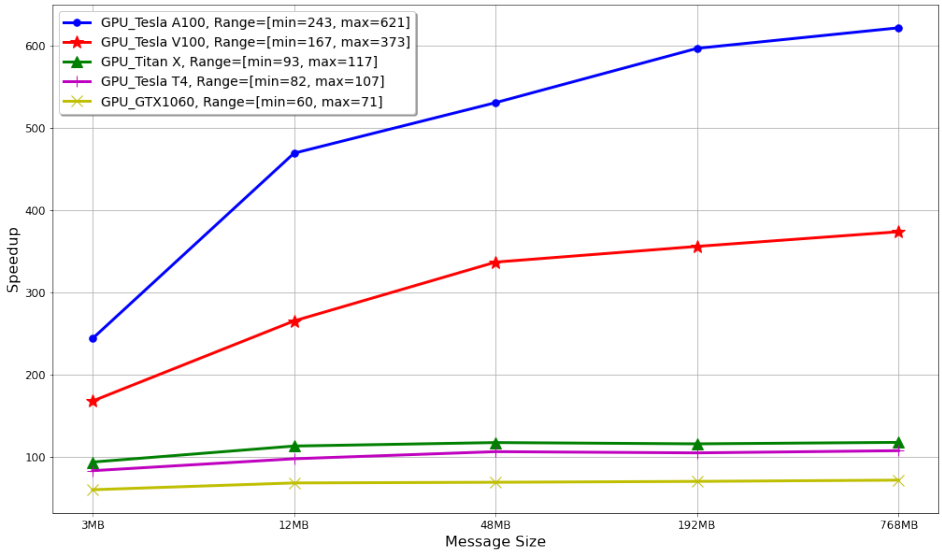
Figure 9: Throughput analysis for the proposed cipher and related ones over different GPU devices

5.3 Cryptanalysis: Resistance Against Well-known Types of Attacks

Different from the majority of existing cipher solutions, the proposed scheme is based on a dynamic key approach, with dynamic substitution and permutation primitives



(a)



(b)

Figure 10: The throughput results (a) of the proposed cipher and others over the CPU. In addition, the speedup ratio (b) comparison of the proposed cipher over GPUs to CPU speed.

for every input message. Previous statistical tests (entropy analysis, probability density function, and correlation tests) confirmed the strength of the proposed cipher scheme against statistical attacks (see Table 2). In addition, the key sensitivity analysis demonstrated a high sensitivity and consequently resistance against key-related attacks. These results are sufficient to infer that no useful information can be derived

from the produced ciphertext. On the other hand, the resistance to chosen/known plaintext attacks can be confirmed by using the dynamic key approach, which greatly complicates the attacker's task. Accordingly, the effect of single message failure and accidental key disclosure are also minimized since a dynamic key is produced for each input message. Therefore, differential and linear attacks are not feasible since any change in the dynamic key results in a significant difference in the produced cryptographic primitive produced and thus in the ciphertext. Furthermore, the key space of the secret key is on the order of 2^{128}, which is large enough to make brute force attacks infeasible. The same is true for the dynamic key space, which is 2^{512}. One should takes into account the fact that the difficulty of the ciphertext-only attack is equivalent to one of the brute force attacks and in the proposed solution a large secret key and a large dynamic key are used. Therefore, in our case, a ciphertext-only attack cannot recover useful information from the ciphertext.

6 Performance Analysis

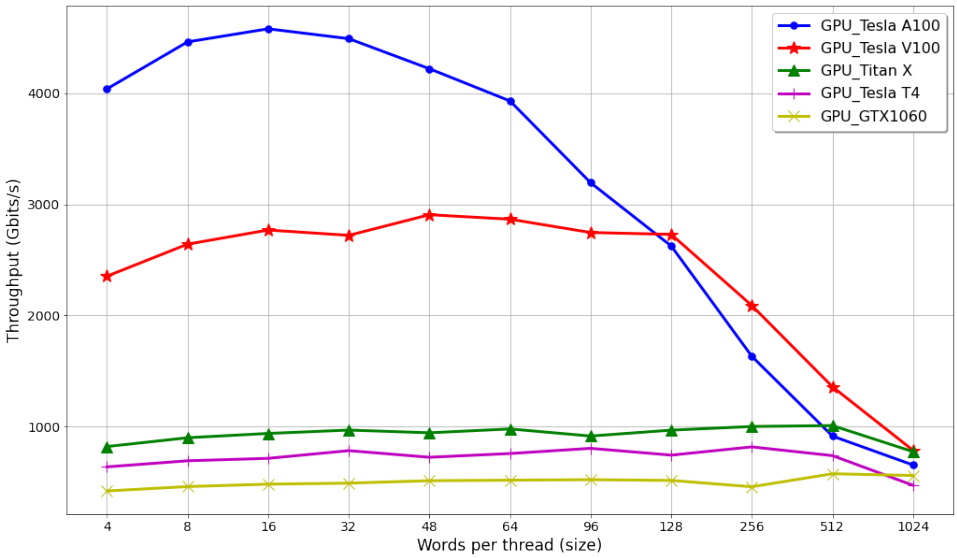
In this section, the proposed method is evaluated and compared against various encryption algorithms using both GPU and CPU devices. This evaluation was conducted on a Linux/Debian system. CUDA version 11.3 is used to program and implement all cryptographic algorithm kernels.

6.1 Experimental results over GPUs

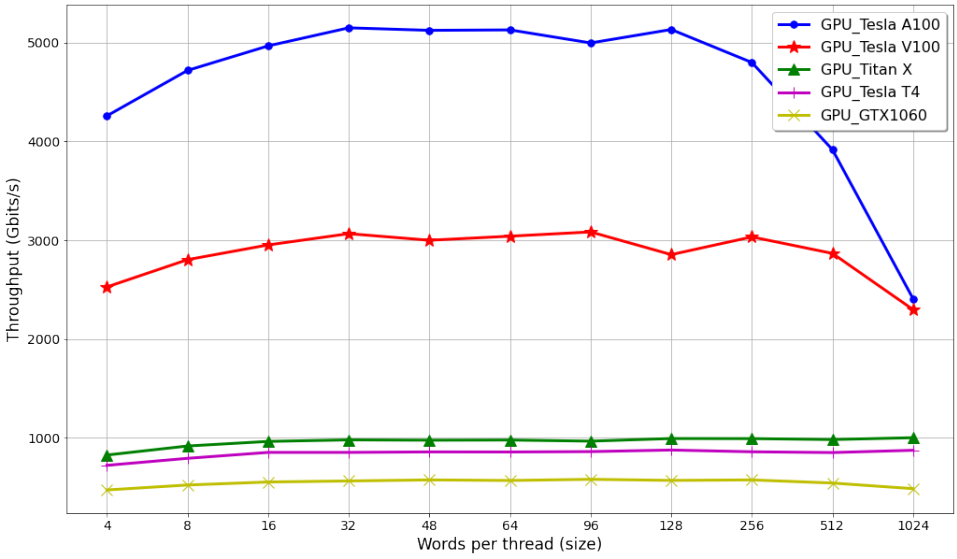
In this section, the required latency of all tested ciphers are quantified to assess their performance. Therefore, these ciphers were evaluated on different GPU devices that are listed and detailed in Table 3. In addition, four different cipher algorithms are applied in this test, which are Speck [29], Simon [29] and AES [8] in addition to a recent stream cipher based on GPU, called ESSENCE [30] are compared with the proposed one. It has one round function to encrypt or decrypt the data stream vector. However, these ciphers were implemented in the CTR mode (which can be considered a stream cipher) to be processed in parallel, and thus they are comparable to the proposed method.

The throughput results over five GPU devices are presented in Figure 9. The obtained results indicate that the proposed stream cipher scheme is faster compared to recent and standard ciphers on GPU. Different throughput ratios are obtained depending on the message size and the computing capabilities of the GPU devise executing the cipher algorithm. Table 4 and Table 5 show the highest throughput and execution time values acquired over Tesla A100 GPU.

Moreover, the average speedup ratios of the proposed cipher and the considered ciphers for comparison, obtained over all GPU devices, are presented in Table 6. As these results show, the proposed stream cipher is more efficient than the existing ones and consequently it is more suitable for real-time applications. Furthermore, the proposed cipher requires a low computation complexity due to its simple implementation (one round with simple operations such as PRNG and substitution). The proposed solution can be considered as an enhancement of the recent ESSENCE cipher and it reaches better throughput.



(a) message size = 48 MB



(b) message size = 192 MB

Figure 11: The throughput results of different word lengths per thread: (a) message size = 48 MB, (b) message size = 192 MB.

On the other hand, recent lightweight ciphers such as Speck, Simon, and AES except ESSENCE apply several rounds and use several operations per round. The round number is 32, 68, and 10 for Speck, Simon, and AES respectively for a secret key size

equal to 128 bits. Both Speck and Simon ciphers are based on quick and simple procedures that are popular in a wide variety of computer systems (AND, rotate, XOR, and modular addition). Both Speck and Simon 2B work on 2 blocks per thread, while the 4B versions work on 4 blocks per thread. Moreover, for more details about their implementation, the reader can refer to [29]. However, Simon is dedicated to hardware implementation, its results present different throughput speeds according to the GPU hardware used.

Moreover, Figure 11 presents the throughput versus number of words per thread ($size$) for two message lengths and for different GPU devices. The obtained results indicate that for a low number of words per length, a high throughout is achieved by the proposed solution. However, for a higher number of words per thread, a lower throughput is obtained. Thus, the optimal number of words per length can be varied between 16 and 64.

6.2 Experimental results over CPU

The proposed ORSCA and the other recent cipher algorithms are evaluated on an Intel (R) i7-7700HQ CPU to determine their performance. Each core of this CPU operates at a frequency of 2.80 GHz. The sequential versions of all ciphers are implemented on this CPU and their throughput results are presented in Figure 10a. The obtained results indicate that the sequential algorithm of the proposed ORSCA has the highest throughput compared to all other cipher algorithms for different message sizes, especially compared to ESSENCE. Moreover, the speedup ratios of the proposed ORSCA are computed between its execution time over all five GPU devices and its execution time over the CPU, as in Figure 10b. These results indicate clearly that by using a powerful GPU device, better speedup can be achieved, where the best speed is achieved with the Tesla A100.

7 Conclusion

In this paper, an efficient optimized one-round stream cipher scheme is proposed and it is called "ORSCA". It is designed to target the GPU architecture. The proposed cipher outperforms the most optimized implementations of AES, SIMON, and Speck on GPU, which makes it more suitable for real-time applications. Moreover, the implementation of the proposed cipher is very simple compared to other existing cipher schemes. Furthermore, the robustness of the proposed stream cipher scheme is based on the use of dynamic cryptographic primitives that can be changed for each input message. In addition, the resistance against attacks has been assessed and confirmed via cryptanalysis. Also, different benchmark tests were done to prove the efficiency of the proposed cipher. In future work, the design of an efficient parallel dynamic key-dependent hash algorithm for GPU will be investigated.

Acknowledgement

This paper is partially funded by the Ministry of Higher Education and Scientific Research of Iraq. It was partially supported 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] Christof Paar and Jan Pelzl. *Understanding Cryptography: a Textbook for Students and Practitioners*. Springer Science & Business Media, 2009.

[2] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.

[3] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Upper Saddle River, NJ, 2017.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Raphaël Couturier. *Designing Scientific Applications on GPUs*. Numerical Analysis & Scientific Computating. Chapman & Hall/CRC, 2013.

[10] Nvidia, CUDA. A Cuda Programming Guide, version 9.0. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

[11] 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.

[12] Cihangir Tezcan. Optimization of advanced encryption standard on graphics processing units. *IEEE Access*, 9:67315–67326, 2021.

[13] Hassan N Noura, Ola Salman, Nesrine Kaaniche, Nicolas Sklavos, Ali Chehab, and Raphaël Couturier. Tresc: Towards redesigning existing symmetric ciphers. *Microprocessors and Microsystems*, page 103478, 2020.

[14] Hassan N Noura, Ali Chehab, and Raphaël Couturier. Overview of efficient symmetric cryptography: dynamic vs static approaches. In *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6. IEEE, 2020.

[15] 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*, 77(14):18383–18413, 2018.

[16] 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.

[17] Hassan Noura, Ali Chehab, Mohamad Noura, Raphaël Couturier, and Mohammad M Mansour. Lightweight, dynamic and efficient image encryption scheme. *Multimedia Tools and Applications*, 78(12):16527–16561, 2019.

[18] Hassan Noura, Raphaël Couturier, Congduc Pham, and Ali Chehab. Lightweight stream cipher scheme for resource-constrained iot devices. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–8. IEEE, 2019.

[19] Hassan Noura, Ali Chehab, and Raphael Couturier. Lightweight dynamic key-dependent and flexible cipher scheme for iot devices. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–8. IEEE, 2019.

[20] Hassan N Noura, Ali Chehab, and Raphael Couturier. Efficient & secure cipher scheme with dynamic key-dependent mode of operation. *Signal processing: Image communication*, 78:448–464, 2019.

[21] Hassan N. Noura, Ola Salman, Raphaël Couturier, and Ali Chehab. Lorca: Lightweight round block and stream cipher algorithms for iov systems. *Vehicular Communications*, page 100416, 2021.

[22] 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.

[23] Guy L. Steele, Doug Lea, and Christine H. Flood. Fast splittable pseudorandom number generators. OOPSLA '14, page 453–472, New York, NY, USA, 2014. Association for Computing Machinery.

[24] 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.

[25] 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.

[26] Guoji Zhang and Qing Liu. A Novel Image Encryption Method Based on Total Shuffling Scheme. *Optics Communications*, 284(12):2775–2780, 2011.

[27] 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.

[28] C. Doty-Humphrey. Practrand. `http://pracrand.sourceforge.net`, 2014.

[29] Wai-Kong Lee, Bok-Min Goi, and Raphael C.-W. Phan. Terabit encryption in a second: Performance evaluation of block ciphers in GPU with kepler, maxwell, and pascal architectures. *Concurr. Comput. Pract. Exp.*, 31(11), 2018.

[30] Raphael Couturier, Hassan Noura, and Ali Chehab. ESSENCE: GPU-based and dynamic key-dependent efficient stream cipher for multimedia contents. *Multimedia Tools and Applications*, 79(19-20):13559 – 13579, 2020.

Table 3: Employed GPU devices during bench-marking

GPU Device	**Characteristics**
Tesla A100	• Compute capability: 8.0 • Global memory: 40,000 MB • GPU frequency: 1.41 GHz • Memory frequency: 1,215 MHz • Number of CUDA cores: 6,912
Tesla V100	• Compute capability: 7.0 • Global memory: 16,152 MB • GPU frequency: 1.53 GHz • Memory frequency: 877 MHz • Number of CUDA cores: 5,120
Titan X GPU	• 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
Tesla T4 GPU	• Compute capability: 7.5 • Global memory: 15,110 MB • GPU frequency: 1.59 GHz • Memory frequency: 5,001 MHz • Number of CUDA cores: 2,560
GTX1060 GPU	• Compute capability: 6.1 • Global memory: 6,078 MB • GPU frequency: 1.34 GHz • Memory frequency: 4004 MHz • Number of CUDA cores: 1,280

Table 4: Throughout comparison of ORSCA ($size$=32) and with other ciphers on Tesla A100 GPU

Message size	Throughput (in Gbits/s)						
	ORSCA	ESSE.	Speck 2B	Speck 4B	Simon 2B	Simon 4B	AES
$1024^2 \times 3$	2109.4	2008.4	1308.7	1494.4	948.8	1560.3	653.0
$2048^2 \times 3$	4016.0	3676.5	2147.3	2615.3	1298.7	2756.7	1402.4
$4096^2 \times 3$	4580.2	4357.2	2849.4	3070.6	1596.6	3239.2	1689.1
$8192^2 \times 3$	5158.2	4849.3	2763.7	2863.4	1616.4	3608.4	1721.4
$16384^2 \times 3$	5247.8	5018.2	2529.8	2738.0	1390.0	3703.9	1831.0

Table 5: Execution time comparison of ORSCA ($size$=32) and with other ciphers on Tesla A100 GPU

Message size	Execution time (millisecond)						
	ORSCA	ESSE.	Speck 2B	Speck 4B	Simon 2B	Simon 4B	AES
$1024^2 \times 3$	0.012	0.014	0.019	0.017	0.027	0.016	0.039
$2048^2 \times 3$	0.024	0.027	0.047	0.038	0.078	0.037	0.072
$4096^2 \times 3$	0.088	0.092	0.141	0.131	0.252	0.124	0.238
$8192^2 \times 3$	0.312	0.332	0.583	0.562	0.996	0.446	0.936
$16384^2 \times 3$	1.228	1.284	2.547	2.353	4.635	1.739	3.519

Table 6: Speedup ratio comparison of the proposed ORSCA to all other methods over all five GPU devices

Message size	Average speedup ratio compare to					
	ESSE.	Speck 2B	Speck 4B	Simon 2B	Simon 4B	AES
$1024^2 \times 3$	1.35	1.54	1.42	2.42	1.42	3.73
$2048^2 \times 3$	1.18	3.49	1.14	4.29	1.14	2.22
$4096^2 \times 3$	1.18	8.57	1.49	10.45	1.41	2.18
$8192^2 \times 3$	1.09	1.59	1.97	3.15	1.91	3.00
$16384^2 \times 3$	1.06	1.81	1.63	3.03	1.41	2.93