# Simultaneous Encryption and Authentication of Messages over GPUs

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

[1]University of Babylon, Hillah, Iraq
[2]Université de Franche-Comté, CNRS, institut FEMTO-ST, F-90000 Belfort, France
[*]Corresponding author

**Abstract**

There has recently been a rising interest in inventing new efficient cryptographic algorithms, thanks to advances in the field of Graphics Processing Unit (GPU) technology. Current cryptographic algorithms have been implemented with GPUs, including the Advanced Encryption Standard algorithm (AES) and the Secure-Hash Algorithm 3 (SHA3). However, the currently available cryptographic approaches cannot fully benefit from the GPU's capabilities, as they are not designated in accordance with the GPU characteristics. Therefore, they are not carried out in an efficient manner. Thus, the need to design new cryptographic algorithms that can achieve the best performances without degrading the security level. In this work, a new message Encryption and Authentication Algorithm (MEAA) is specifically proposed for graphics processing units (GPUs). It consists of one-round encryption and authentication functions that are based on the dynamic key-dependent scheme. Experimental results indicate that the proposed approach reaches a throughput of over 580 GB/s over the GPU Tesla A100. Additionally, it demonstrates that the performance improvement ratio is better than that of existing methods. On the other hand, the proposed MEAA is impervious to well-known cryptanalysis attacks since it is based on the dynamic key strategy, and different primitives of cryptography, which are employed for each new input message.

## 1 Introduction

GPUs (Graphics Processing Units) are used in big data to accelerate the processing of large amounts of data. Traditional CPUs (Central Processing Units) are designed to handle a wide variety of tasks, but they are not optimized for parallel processing of large amounts of data. In contrast, GPUs are designed specifically for parallel processing and can handle a large number of calculations simultaneously. This makes GPUs particularly useful for big data tasks such as data processing, protection, and analysis, which involve processing large amounts of data in parallel. As a result, GPUs have become an increasingly popular tool for big data processing and analysis, especially in fields such as finance, healthcare, and e-commerce, where large amounts of data are generated and analyzed on a daily basis.

However, it is critical for any application to implement robust security measures to minimize the risk of attacks and mitigate their impact. Therefore, several security services have to be ensured in this context to prevent attacks such as confidentiality, integrity, authentication, and availability. This can be done by using cryptographic [1] and non-cryptographic solutions [2, 3, 4, 5, 6].

Therefore, without adequate security measures, big data systems can be vulnerable to data breaches (where attackers gain unauthorized access to sensitive data stored in big data systems, resulting in the exposure of confidential information), cyber-attacks, and other security threats that can have serious consequences for individuals and organizations. Furthermore, data breaches are considered one of the biggest risks associated with attacks on big data. Breaches can result in the theft or exposure of sensitive data, which can lead to identity theft, financial fraud, or other forms of cybercrime. Consequently, the security of big data is mandatory because of the large volumes of sensitive data that are collected, stored, and analyzed in big data systems such as personal data, financial information, and confidential business data. Therefore, to ensure the

security of big data systems, several security measures have to be introduced such as implementing secure message authentication and encryption of data both in transit and at rest, which is the main goal of this paper.

Big data can be based on real-time applications that generate large volumes of data that need to be secured in real-time. However, it is equally important to ensure that these security measures do not compromise the main functionality of the real-time applications. Therefore, new security solutions are required that can reach a high level of security while minimizing the required delay to preserve the main functionality of these real-time applications. Indeed, the primary objective of this study is to verify whether designing a cryptographic algorithm such as message authentication encryption that takes advantage of GPU characteristics can enhance performances by leveraging parallel processing to efficiently handle large volumes of data. A significant hurdle in achieving this objective is developing cryptographic operations, or round functions, that can be parallelized. This approach enables the GPU-based algorithm to outperform conventional methods by executing substitution and diffusion operations more quickly than a traditional CPU.

## 1.1 Related Works

In general, data confidentiality can be achieved by employing symmetric cryptographic algorithms, which can be categorized into two primary types: stream ciphers and block schemes. On the other hand, message authentication algorithms can be used to overcome the difficulties associated with message authentication attacks. As a result, the use of a scheme distribution and a reliable key exchange mechanism is required.

When it comes to both data integrity and message authentication, two possible methods can be used,: symmetric message authentication algorithms that can be based on keyed-hash function or block cipher; and digital signatures, which employs asymmetric cryptographic algorithms such as RSA or Elliptic curve. However, both integrity and data confidentiality can be satisfied by combining together the encryption and the authentication of messages using the same cryptographic method.

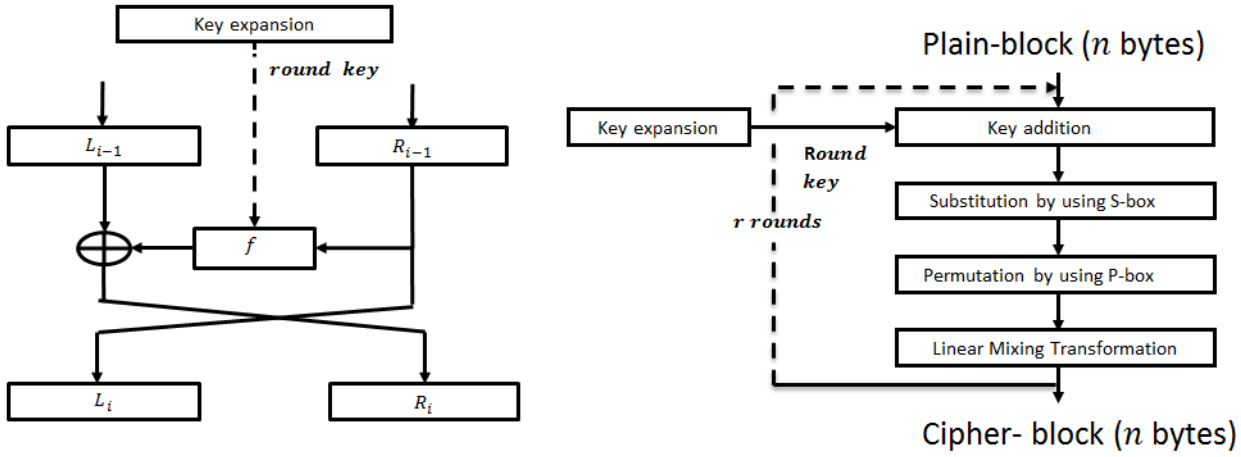
Examples of current MAA that use block ciphers such as AES [7] are based on the authentication operating mode which includes either the Ciphered Message Authentication Code (CMAC) or the Galois Message-Authentication Code (GMAC). On the other hand, Hash-Message Authentication Code (HMAC) is based on the Secure-Hash Algorithm (SHA) family.

The round functions used by the MAA are either based on Feistel-Network (FN), such as the recent lightweight ciphers SIMON and SPECK or based on the Substitution Permutation Network (SPN), which is similar to the work of AES in [1, 8]. The listed round function structures are illustrated in Figure 9. In fact, there are fewer rounds required with SPN in comparison to FN. This should result in reduced latency and less resources required with SPN over FN.

The use of a Graphics Processing Unit (GPU) is being adopted to accelerate the execution of current encryption algorithms. As a result of the hundreds or even thousands of cores in a GPU, cryptographic algorithms can be parallelized to accelerate its performance and reduce the required computation delay consequently. For example, AES implemented over GPUs in [8, 9, 10], and the speedup relative to the CPU version [11] have been remarkable. Notice that the performance of an algorithm can be maximized by optimizing shared memory, warp using, and registers [12]. On the other hand, examples of constructed pseudo-random number generators with GPU technology are presented in [13] and [14].

There has been a recent presentation of an optimized and interesting application of AES over GPU in [10]. This approach obtained impressive performance, and the creators made significant improvements over earlier versions of the software. The work in [15] presents an optimized version of AES over GPU. This approach is more general and has improved the performance by around 10% compared to the implementation in [10].

AES or HMAC (applying SHA-2 or SHA-3), for example, are vulnerable to analytic attacks, but the number of rounds (r) determines the security levels. That is why the level of security must be balanced with resource needs. Moreover, existing cryptographic algorithms based on static structures have been proven to have anti-cryptanalytic properties as in [1, 16, 17].




(a) FN (b) SPN

Figure 1: The round function of DES and AES uses different structures: DES uses the Feistel-Network (a), while AES employs the Substitution Permutation Network (SPN) (b).

These algorithms, however, suffer from a variety of security flaws due to the round function's static nature [18]. Accordingly, future attacks [16, 17] that aim of taking advantage of the static structure (i.e diffusion primitives and substitution) in order to obtain the secret-key in [1] are more likely to occur. Instances of such attacks contain applied attacks such as fault and side-channel attacks as described in [1]. To counteract these implementation attacks, higher latency and more resource overhead are necessarily required. Thus, certain applications and future systems may be rendered unusable due to a decrease in the algorithm's performances [19].
Since the cryptographic primitives are fixed, the round number $r$, which is necessary to obtain the desired cryptographic characteristics should be large. Most recently, various diffusion operations and substitutions are made inside each input message of a dynamic pseudo-random way [20, 21, 22, 23] to minimize the number of rounds to 1 or 2 respectively. Moreover, a dynamic cryptographic method presents the advantage of the benefit of providing a higher resilience to implementation attacks.

## 1.2 Problem Definition

As previously stated, current message encryption and authentication methods were not intended for GPU implementation. Therefore, they are unable to make use of GPU features in terms of parallel processing policies. Additionally, these methods need a large number of rounds, which results in a significant level of computing complexity and delay. Moreover, the application of encryption and authentication will simultaneously increase the computational complexity. As a result, for a collection of big-data applications that employ GPUs, one needs to develop a robust and efficient message encryption and authentication method that is optimized for GPU performance. To address these constraints, the suggested parallel message encryption and authentication method depends on the dynamic key-dependent method described in [20, 24], which reduces the number of operations and needed rounds. Therefore, the proposed approach may strike a favorable balance between security and efficiency, while also providing a straightforward method for preventing specific implementation attacks.

## 1.3 Contribution

Based on the in which approaches presented in [20, 22, 25], a simultaneous method for message encryption and authentication is proposed, where the primitives of cryptography are changed for every new input message, and the key-avalanche effect is used to create the message-avalanche effect.

Two PRNGs with a high number of seeds are used in the proposed message encryption-authentication algorithm in addition to an efficient and simplified compression mechanism. A message encryption-authentication algorithm built to operate on a GPU has also been proposed for the first time, to the best of the author's knowledge.

# 2 Background & Methodology

In this part, the required background and preliminaries for message authentication and encryption and the employed cryptographic round functions are discussed in addition to the GPU's architecture. Moreover, the used notations are listed in Table 1.

Table 1: Table of notations

| **Symbol** | **Definition** |
|---|---|
| $k$ | the secret key |
| $SK$ | Shared key of secret session |
| $nonce$ | A dynamic Nonce that updates per input message |
| $DK$ | Dynamic key changed with each input message |
| $k_{S1}$ and $k_{S2}$ | Substitution sub-keys |
| $N_{seeds}$ | Total number of seeds |
| $IV$ | Initial vector of seeds |
| $K_{rg}$ | Seed generation of sub-key |
| $b_i$ | $i^{th}$ plain-text block |
| $C$ | Encrypted message |
| $c_i$ | $i^{th}$ encrypted block |
| $H_i$ | $i^{th}$ authenticated block |
| $KSA$ | Key Setup Algorithm of RC4 |
| $\&$ | The logical AND operation |
| $\oplus$ | The logical "exclusive or" operation |
| $\|\|$ | Concatenation operation |
| $x >>> y$ | Returns x with the bits shifted to the right by y places |

## 2.1 Message Authentication Versus Encryption

Security is required to protect all types of resources and data from a variety of security risks. Numerous security services, including confidentiality, integrity, authentication, and availability must be guaranteed. Unfortunately, privacy and integrity are often misunderstood and needlessly entwined, thus encryption does not offer any integrity (in general), and should never be used for message authentication. Obviously, encryption may be used in combination with other methods to authenticate messages. Cryptography and non-cryptography solutions may be used to guarantee security services. The purpose of a Message Authentication Code (MAC) is to prevent an adversary from altering a message transmitted between two parties without the parties' knowledge. As with encryption, this is only feasible if the communicating parties possess a secret that the opponent is unaware of (otherwise, nothing prevents an adversary from impersonating the sender's message).

For example, if two users want to securely communicate on a message encryption-authentication system, they must begin by creating and exchanging a secret key, $K$ first, as seen in Figure 2. If one party wishes to send a message, the message must be encrypted first and then a MAC must be computed depending on the cipher message. Consequently, the sender will share the key and transmits the cipher message along with its MAC. When the second party receives the cipher message and its MAC, the MAC value is computed, and its value is checked for the cipher message in terms of the shared key.

Algorithms for MAC are a critical component of the majority of internet protocols. They certify the message's integrity involving two or more participants to the transaction.

Moreover, there are predetermined algorithms that need a secret key to govern the input-output mapping which is known as the tag. However, MAC algorithms do not conduct mapping on a fixed input size basis, which makes them similar to hash functions. Although MAC functions take huge input message and output of a constant size, they are not as secure as Hash MAC functions (HMAC). In this case, unlike MAC, HMAC uses a hashed key. In order for HMAC to be more secure, it must use a cipher which is known as CMAC. In order to compare HMAC to CMAC, the issue presented in [26] must be solved first.

CMAC is a relevant choice if the cipher already exists according to privacy. CMAC employs block ciphers, which have smaller inputs but may have lower latency than hashes. For shorter connections, CMAC with AES is usually faster with less delays than HMAC. HMAC begins quicker with larger messages since it takes less cycles per byte. Additionally, hashes provide longer outputs than ciphers, allowing for larger MAC tags. This work follows the principles of CMAC to design GPU multi-threaded message encryption-authentication algorithms. CMAC fulfills our goal to obtain an encrypted message and its MAC simultaneously. Furthermore, the encryption-authentication mode is better to use in this way to maintain the randomness of the message to be authenticated.

## 2.2 Round Function Of Block Stream Ciphers

The avalanche effect is a critical feature of any block cipher since it requires that every change in the input must change every bit of the output [27]. Round function in a symmetric stream cipher can be used in a different way to increase the security level. To satisfy the avalanche effect, a round function is applied several times. Different cryptographic operations may be employed inside the round function such as key mixing, rotation, XORing, substitutions and etc. Moreover, each round's output is used as the input for the subsequent round. The National Security Agency (NAS) created Speck and Simon which used multi-round approaches to increase the security level [28]. Speck has a key size of 64-256 bits, a block size of 32-128 bits, and operates for 22-34 rounds. Simon makes use of a 64-256 bit key, a 32-72 bit internal state block size, and a round number that varies between 32 and 72.
AES [29] is a block cipher that operates on blocks of data of 128 bits in size and employs keys of 128, 192, or 256 bits in length. AES is based on the SPN concept in its architecture. It has a round function that iterates $r$ times, based on the size of the secret key. For a secret key with a size of 128, 192, or 256 bits, the number of rounds, $r$, is equal to 10, 12, or 14. Whereas in the last iteration, this round function has four actions.

Recent works have attempted to strike a balance between the levels of security and the computational cost of the encryption method. The number of rounds used in encryption methods has been decreased in order to shorten the algorithm's execution time, which is particularly important when the algorithm is implemented on restricted hardware devices, see [18, 30]. Whereas for large communications, it is necessary to use fast encryption methods.

## 2.3 GPU's Architecture

The GPU is a type of computing processing technology that is often used to speed up calculations. GPUs are employed in a broad variety of systems and computing applications today, including smartphones, embedded computing, and supercomputers. The GPU architecture is somewhat distinct from the CPU architecture. A GPU's architecture is improved to maximize the throughput of several concurrent threads, which might vary from hundreds to thousands.

NVIDIA GPUs consist of many Stream Multiprocessors (SMs), and their number varies according to the GPU model, as seen in Figure 3 a. In comparison to the CPU design, the GPU architecture supports a large number of concurrent threads to optimize efficiency (i.e the throughput). A GPU has hundreds or thousands of computing cores. The hardware of the GPU is intended to run a large number of threads concurrently, regardless of whether the memory access is the bottleneck. Users may take advantage of the GPU's processing capability by using a large number of threads, which may exceed the number of cores.

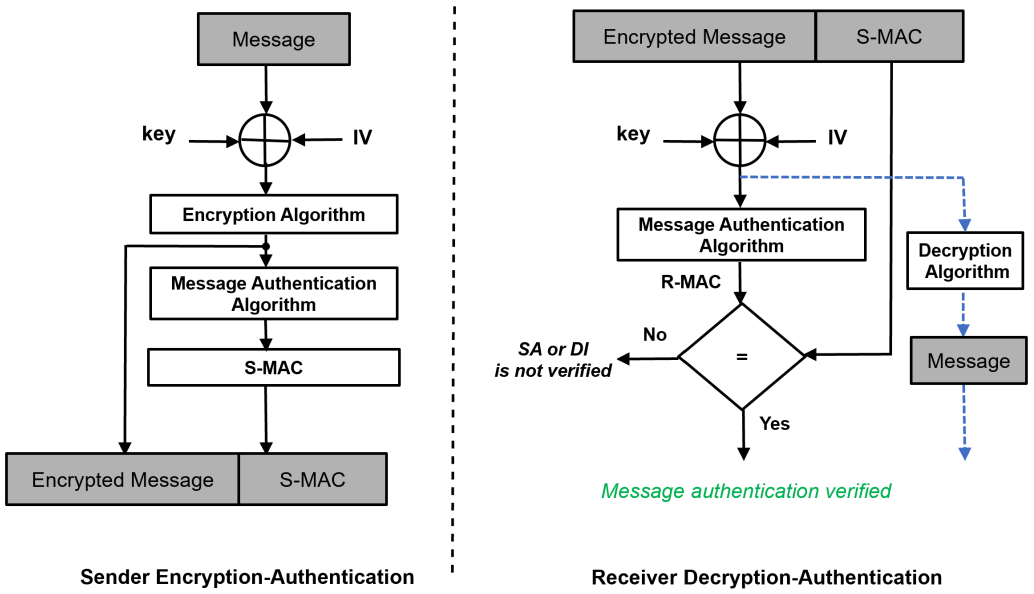



Figure 2: Procedure of Message Authentication-Encryption Algorithm.

As a result, whilst some threads await their information, others may execute. Threads are scheduled in 32-thread groups on an SM referred to as a warp. Each warp is capable of performing only one instruction at a time. However, if there are two threads that are concurrently executing on the same warp, the first command is executed while the second one waits. This is referred to as a thread divergence [31]. As a result, "IF condition" and "WHILE Loop" situations should be avoided wherever possible.

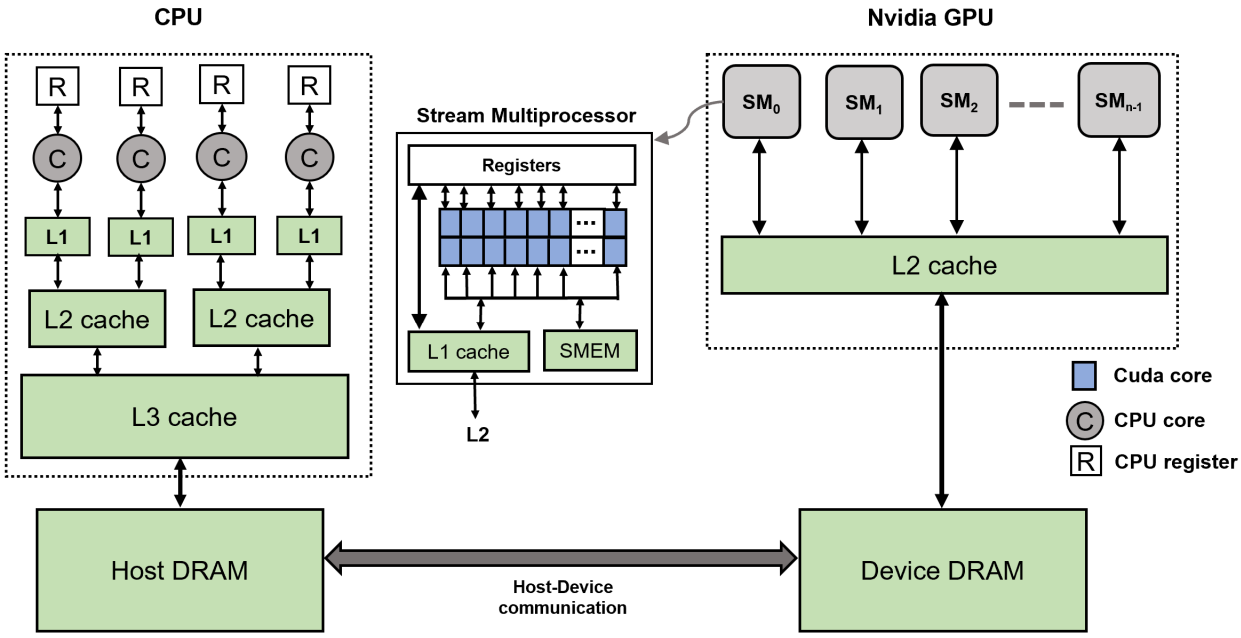
Indeed, threads are organized in blocks. The maximum threads number can reach up to 1024 depending on the GPU's design architecture. Let's note that the technical specifications of the GPU are continuously changing with each new generation. According to the design determinants of GPUs, this paper presents a solution to design both message encryption and authentication, thus fitting the GPU hardware features.

Cache, texture, shared, private and quick access registers are among the various types of memory that a GPU normally provides. As a result, memory management in GPUs is critical and essential. Figure 3 b shows the memory organization of Nvidia GPU devices.

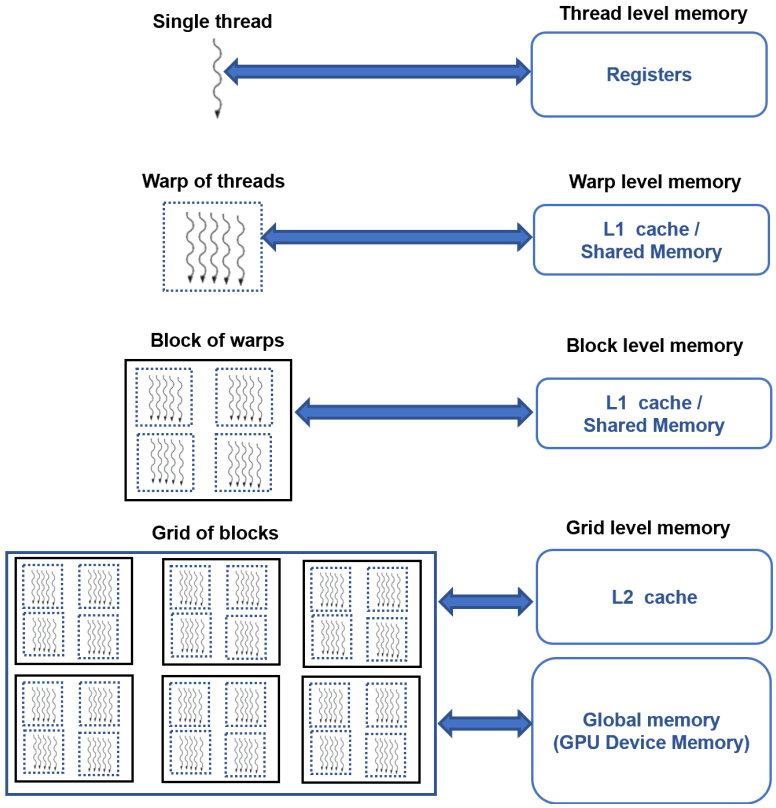
- Thread-private registers: These are thread-specific registers that are not available to other threads. Therefore, the compiler takes more decisions based on the register use.
- L1 cache and Shared memory (SMEM): Each SM has an on-chip memory that can function as both an L1 cache and a shared memory. Threads have private access to the L1 cache, while all threads inside a CUDA block may access the shared memory concurrently.
- L2 cache: This cache is shared by all SMs and is available to all threads in each CUDA block.
- Global memory: This refers to the combined size of the GPU's frame buffer and DRAM.

# 3 Dynamic Key Derivation Method

An illustration of the proposed dynamic-key generation and the primitives of cryptography building methods are presented in Figure 4. The cryptographic primitives (such as seeds and two sub-keys) are also required since the dynamic key is a function of them. It is necessary to combine the secret session-key $K$




(a) CPU and GPU architecture




(b) NVIDIA memory hierarchy

Figure 3: CPU and NVIDIA GPU architectures (a), as well as the hierarchical level of NVIDIA GPU memory (b).

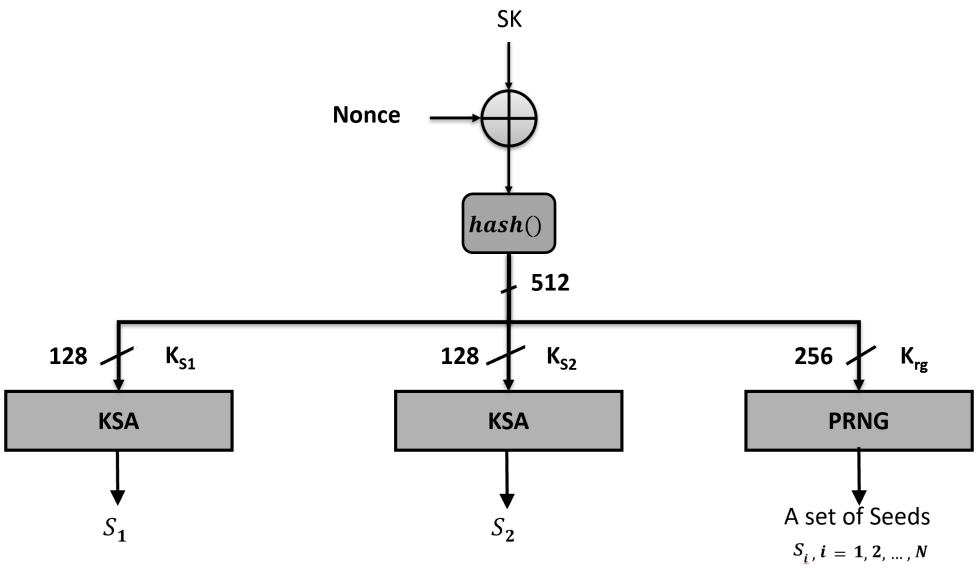



Figure 4: The Proposed dynamic key deviation schema and cipher primitive construction techniques.

with a $Nonce$ (which is unique for each and any input message) $N_o$ in order to generate a dynamic secret block $O = K \bigoplus N_o$, that will be hashed using a safe cryptography hash function to acquire the dynamic key ($DK$). For the purposes of this study, the hash function SHA-512 is used, since among other reasons it provides excellent collision resistance and may be used for short messages (blocks of length equal to 512 bits).

As a consequence, each new input message results in a distinct $DK$ being generated. Because of the dynamic nature of the proposed MEAA algorithm, it is more resistant to powerful message authentication attacks than other algorithms. Then, the dynamic-key $DK$ of length equal to 512 bits, is split into three sub-keys, denoted by the symbols $K_{S1}$, $K_{S2}$, and $K_{rg}$.

Each sub-key is 128 bits (16 bytes) in length, whereas the seeds generation sub-key $K_{rg}$ is 256 bits in length (32 bytes). $K_{S1}$ and $K_{S2}$ are used to generate two shift sub-keys ($DK_1$ and $DK_2$), respectively, while $K_{rg}$ is used as a seed for the PRNG to generate $N$ distinct seeds. The outline of these three sub-keys is described as follows:

1. $K_{S1}$ **and** $K_{S1}$ **sub-keys:** represent the first and second 16 most-significant bytes, respectively, which are used to generate the dynamic shift sub-keys $DK_1$ and $DK_2$. The authors would like to emphasize that the encryption procedure occurs at the byte level, as opposed to other processes that occur at the word level. The Key-Setup Algorithm (KSA) of RC4 was used to produce these shift sub-keys.

2. **Seeds sub key** $K_{rg}$**:** is used to generate the first 32 least-significant bytes of $DK$. In this phase, RC4 is iterated $\frac{N \times Qg}{8}$ times to produce $N$ unique seeds, where $N$ is the maximum number of threads and $Qg$ is the precision of the generator used, which may be 32, 64, or 128. The output key-stream is rearranged to create a matrix of size $N \times \frac{Qg}{8}$ with a single byte as an element. Each row of this matrix is transformed into a binary sequence of length $Qg$ bits. This binary sequence represents one of the $N$ seeds by forming a word of $Qg$ bits. In this instance, any duplicate row of seeds is removed from the list and the RC4 is repeated to generate a new seed. Notably, RC4 is iterated in order to generate the dynamic primitives of cryptography with a high degree of security, where $N$ is chosen according to the required level of security.

These procedures provide a high degree of sensitivity, since every slight modification to the dynamic key results in a whole new set of cryptography primitives.

# 4 The Proposed Message Encryption-Authentication Algorithm (MEAA)

This section presents the proposed MEAA, which is a keyed hash function that employs a single-round function for encryption and authentication in contrast to current symmetric MAA, such as AES and HMAC, which use multi-round cryptographic techniques.

## 4.1 Fundamental Details

The proposed solution has primary characteristics which include: a high degree of security, reduced computing, and simple and parallel hardware and software implementations.

There are three key ideas in the MEAA scheme that help it to achieve these properties:

1. Parallel computation: This method is intended to be executed in many threads at the same time. Although they are completely independent of one another, all of the threads can run concurrently (as seen in Figure 5), since it is not feasible to manage all of them at the exact same time. Each of the SMs has 32 synchronous threads, as well as a shared memory. Since these threads are synchrome, the same operation (encryption function) may be performed on them with various inputs. As a result, the 32 threads are iterated over and over again to execute the authentication function.
2. Modular Design: The modular design of the proposed MEAA allows for input blocks of any size (256, 512, and 1024). Furthermore, any PRNG that shows excellent performance while still meeting the randomization requirements may be utilized in this application. For example, in this work, use of the "Splitmix-64" algorithm is used, which was chosen for its simplicity and speed when dealing with words with a precision of 64 bits.
3. Cryptographic primitives combination in an efficient and lightweight manner: The chosen PRNG (based on word accuracy) is effectively coupled with two dynamic shift sub-keys and an authentication function to create the compressed blocks. On the other hand, when the proposed stream cipher is used, the key-stream that is generated has a steady randomness and uniformity degree, as well as a high degree of periodicity.

## 4.2 The Proposed Parallel MEAA On GPU

The novelty of this paper is that the proposed MEAA is tailored to the features of the GPU, in comparison to current MAA systems. Additionally, the proposed method fulfills the message and key avalanche characteristics, since a new dynamic key and the primitives of cryptography are created each time for a new input message. Additionally, the input message may be used with any data type, including pictures, videos, and texts. The proposed MEAA method just needs a single iteration of the proposed authentication function by authenticating $n$ sub-blocks concurrently. In the proposed MEAA, the Merkle-Damgrad (MD) structure is altered to enable the parallel operation, as shown in Figure 5.

Thus, after producing the primitives of cryptography, the input message $M$ has been padded to a length divided by the data block's size, Tb, which may be set to 128, 256, 512, or 1024 bits. Let us assume that $Tb$ is a multiple of $Qg$ (e.g., $n \times Qg$, where $Qg$ may be equal to 64 or 128). After that, $M$ is partitioned into $nb$ blocks $(B_1, B_2, ..., B_nb)$, with $nb$ equal to $\frac{|D|}{Tb}$). The selection of $Tb$ is determined by the desired degree of security. Following that, each block of data $B_i$, $i = 1,\ 2,\ \ldots,\ nb$, is compressed using the $cf$ function (see Figure 6).

As a result when the processing of all the blocks, a final mixing by the "exclusive or" operation is performed between all blocks $(C_1,\ C_2,\ \ldots,\ C_{nb})$. The mixed block is then compressed using the proposed compression algorithm, using the MAC value as its output. Therefore, the MAC may be computed using the equation (eq. 1) below:

$$MAC = cf(\bigoplus_{i=1}^{nb} C_i), \quad i = 1, \ldots, nb \tag{1}$$

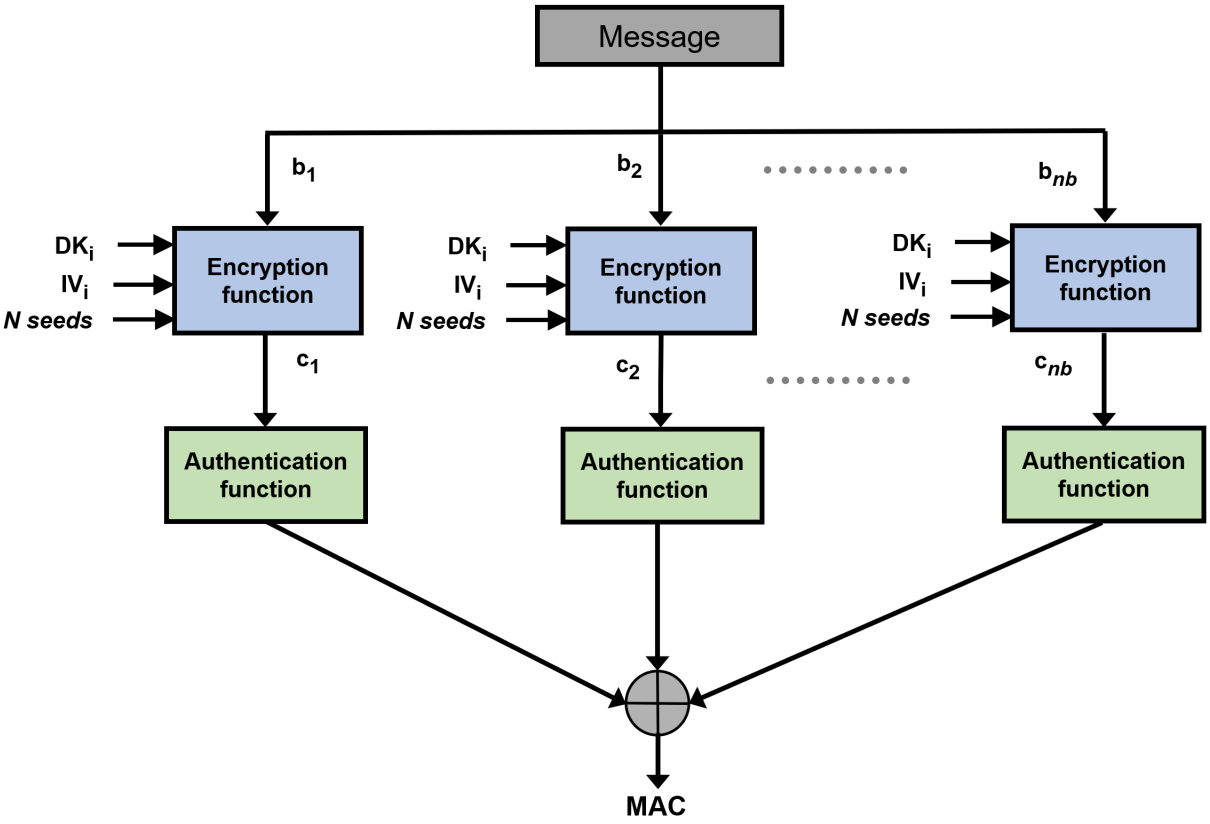



Figure 5: The general structure of proposed MEAA, where the encryption process is applied first and followed by the authentication process.

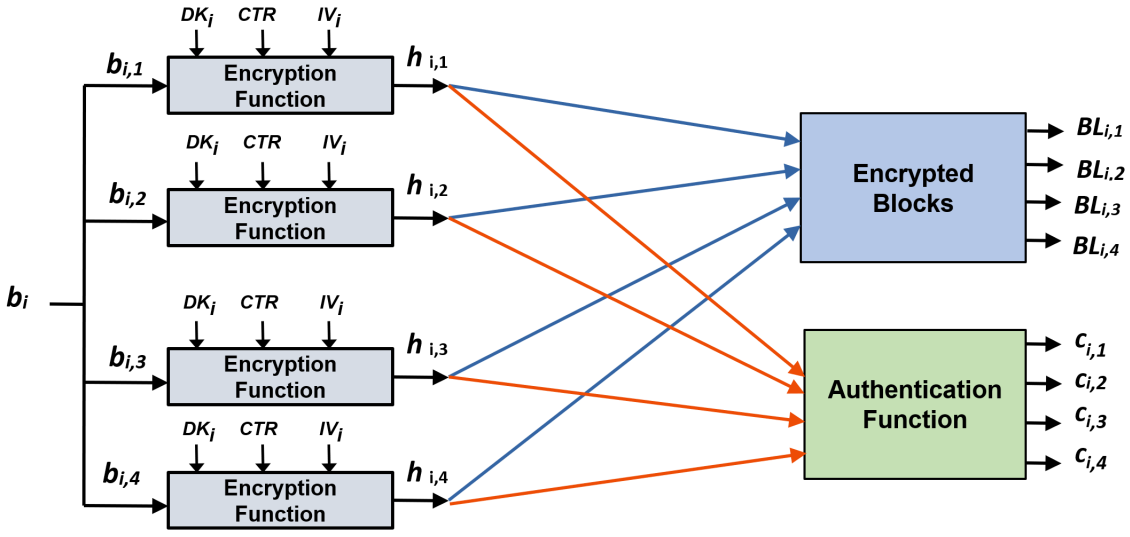



Figure 6: The proposed compression function for the $i^{th}$ block

where $C_i$ represents the $i^{th}$ encrypted block, $\bigoplus$ represents the logical "exclusive or" operation and $nb$ is the number of blocks per message.

The definitions of sustainability in the proposed MEAA's encryption, and authentication functions are

demonstrated as follows:

- **Compression function (cf):** It compresses each data block $B_i$, $1 \leqslant i \leqslant nb$, using the function $cf$ (see Figure 6). The proposed compression function divides the input message blocks into $n$ words of length $Qg$ bits each $(sb_{i,1}, sb_{i,2}, ..., sb_{i,n})$. Indeed, the compression function is intended to be implemented in parallel, as shown in Figure 6, by using the proposed encryption function seen in Figure 7.

  After processing all sub-blocks, the processed sub-blocks are subjected to an authentication function (Figure 8), which results in the compressed block $(C = c_{i,1}, c_{i,2}, ..., c_{i,n})$. This function is used to increase the security of reduced blocks by the compression function. Moreover, the encrypted blocks are used to store each encrypted block $Bl_i$ by the $i^{th}$ thread.

- **Encryption Function:** The proposed encryption function is applied to each sub-block (word level), and it makes use of two dynamic shift sub-keys, as well as a seed that is dynamically selected from a set of produced seeds (N) depending on the block and sub-block numbers, as shown in Figure 7. This function mixes the $i^{th}$ thread index with dynamic key value $Dk_i$. After that, this value will be divided into two sub-words of 32 bits each, denoted by $r1$ and $r2$ respectively, $0 \leqslant (r1\ or\ r2) \leqslant 511$. Similarly, the initial vector $IV_i$'s $i_{th}$ word is divided. The left sub-word of the dynamic key value $DK_{ir1}$ rotates to the left the sub-word of the $IV_i$, while the dynamic key value $DK_{ir2}$ rotates right the sub-word of $IV_i$, $0 \leqslant (DK_{ir1} or DK_{ir2}) \leqslant 63$.

- **Authentication function:** It is used to mix four rounded blocks using three operations: addition, XOR, and rotation, its primitives taken from [32]. When four threads are used to create four rounded blocks, one of them is dedicated to performing this function. Figure 8 indicates that the first block is encrypted by adding its value to both the second block and $DK_i$. The second block is XORing with the third block and then shifts to the right by 24. The third block is encrypted by adding it to both $DK_i$ and to the fourth block. The fourth block XORs with the first block and then shifts to the right by 32. This function is used to increase the security level when authenticating the messages.

# 5 Analysis Of The Proposed MEAA's Security

Numerous security tests, including randomization, uniformity, and sensitivity tests, were conducted to evaluate the proposed MEAA's security [22, 33]. It should be noted that the proposed MEAA is adaptable in relation to the size of $Tb$. For the security analysis, a $Tb$ equal to 256 to provide the results of randomization and tests of uniformity was used. Additionally, a comparison is made between the proposed method and other well-known solutions such as CMAC and HMAC.

## 5.1 Uniformity & Randomness Tests

The message and the cryptographic primitives in this test are used to assess the MAC values' randomness and uniformity. The ciphertext and MAC values are assessed by conducting the proposed MEAA with 1,000 distinct input messages and dynamic keys.

Figure 9 shows the recurrence, the Probability Density Function (PDF) of an original message, and its corresponding encrypted messages. The original message as shown follows the normal distribution, while the ciphertext corresponds to the uniform distribution, and all symbols have a probability of occurrence close to $\frac{1}{256} = 0.039$, as shown in Figure 9, with a random session key. In addition, the recurrence of the encrypted message indicates a high randomness degree of ciphertext. Furthermore, the entropy of ciphertexts is computed and their corresponding PDF is shown in Figure 11-a. These results indicate clearly that the entropy of ciphertext is always high and close to the ideal value, which is 8 at the byte level. These results are validated by the high levels of randomness and uniformity of the proposed message encryption algorithm. The entropy values reveal that the ciphertext adheres to the uniform distribution shown in Figure 9-d.

To validate the independence between plaintext and ciphertext, the correlations between them were computed for 1,000 random dynamic keys. The corresponding PDF of the correlation coefficient between

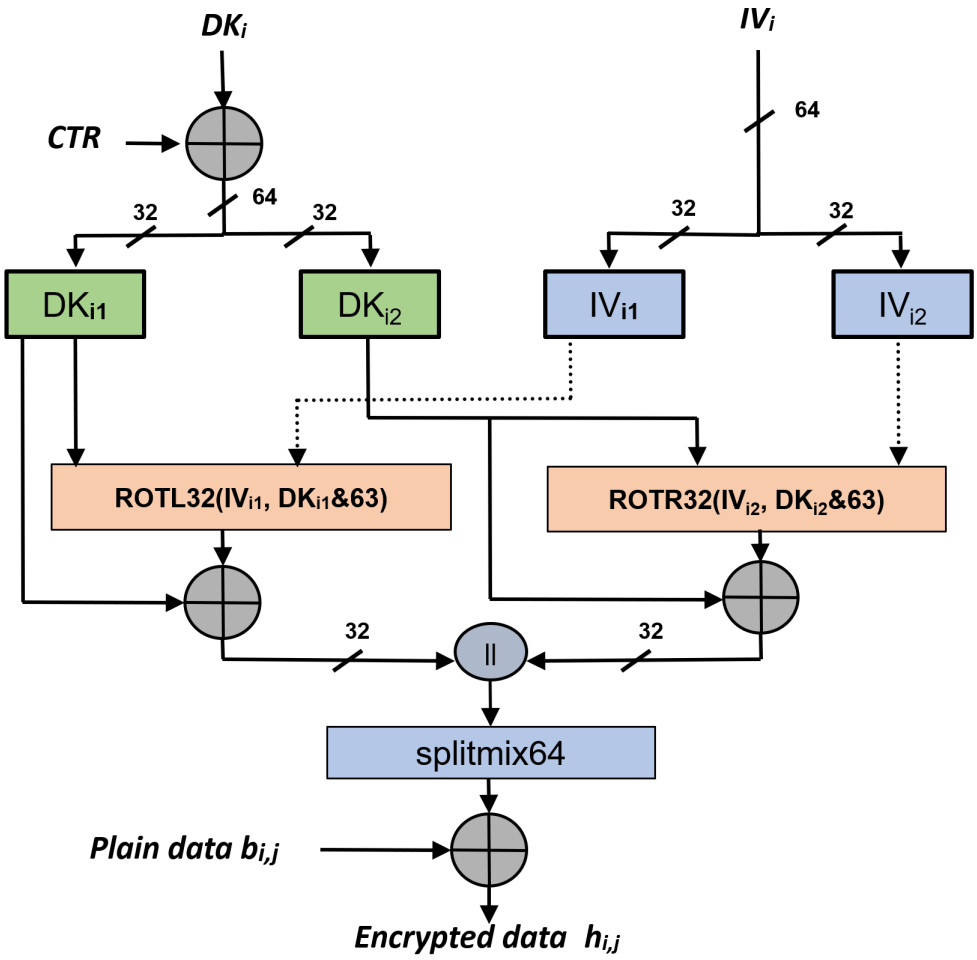



Figure 7: The proposed one-round encryption function

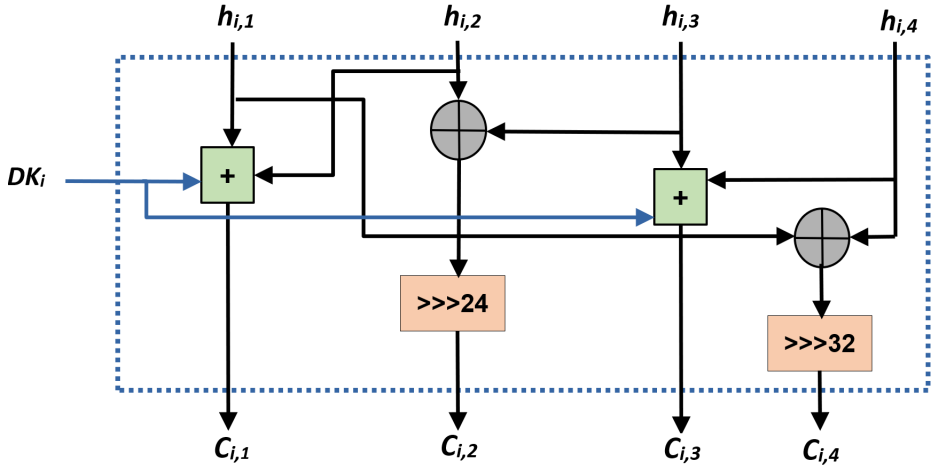



Figure 8: The proposed one-round message authentication function

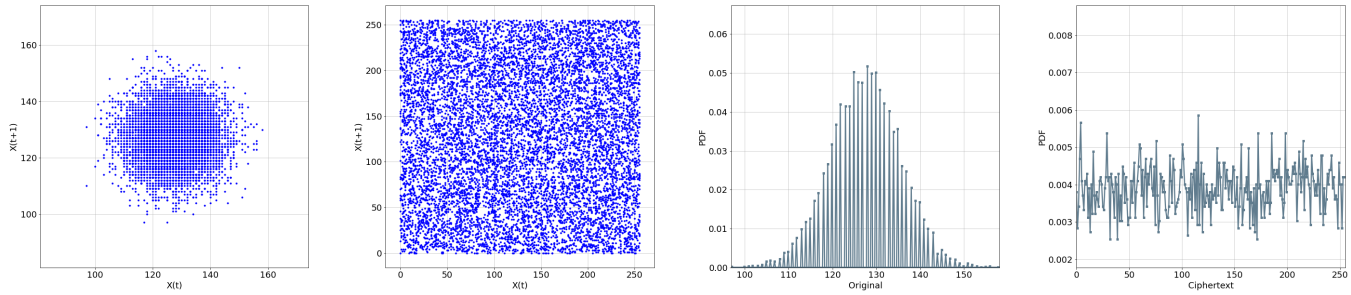
plaintext and ciphertext is shown in Figure 12-(a) and the obtained values are always close to 0. In addition, the difference test, which evaluates the difference in percentage between plaintext and ciphertext, should be close to 50% at the bit level. Figure 12-(b) shows this difference for the proposed message encryption

algorithm. The obtained results for both tests confirm the high level of randomness between the plaintext and ciphertext, respectively. Consequently, these results confirm that the produced ciphertexts are independent of the original ones.

On the other hand, the MAC values for 1,000 random dynamic keys and original messages are computed by using the proposed message authentication algorithm. Then, their values are converted to hexadecimal to produce 64 digits for a MAC length equal to 256 bits. The recurrence and histogram of their values are shown in Figure 10, respectively. The hex values (0 to 15) are equally randomly distributed, which confirms their high level of uniformity and randomness. We also ran a test to determine the number of unique bytes within each MAC value. These tests were applied 1,000 times, each time with a new dynamic key and messages. The randomness and uniformity test results were done with a MAC length equal to 256 bits. The results for the probability of unique byte elements of the proposed message encryption-authentication algorithm are shown in Figure 11-b.

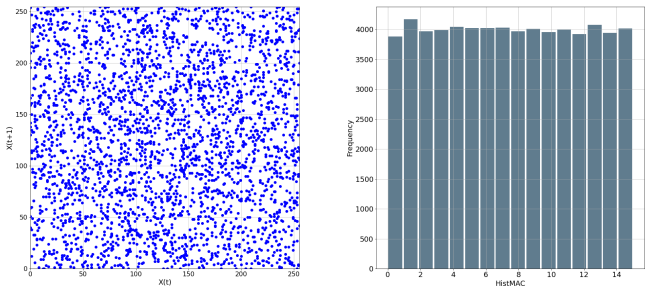
On the other hand, Table 2 provides the numerical statistical (minimum, mean, maximum, and standard deviation) results of all the listed tests for the proposed MEAA. The low standard deviation of all these tests indicates that the obtained results versus the dynamic key are very close to the mean value, which is also close to the ideal value.

The increase in Tb, on the other hand, improves the unpredictability and consistency of test results while also assuring a higher degree of collision resistance. But on the other side, the uniformity and unpredictability of the MAC values can confirm consequently the uniformity and randomness of the produced keystreams. Allow us to mention that the generated key-stream(s) was evaluated using TestU01 and Practrand [34].



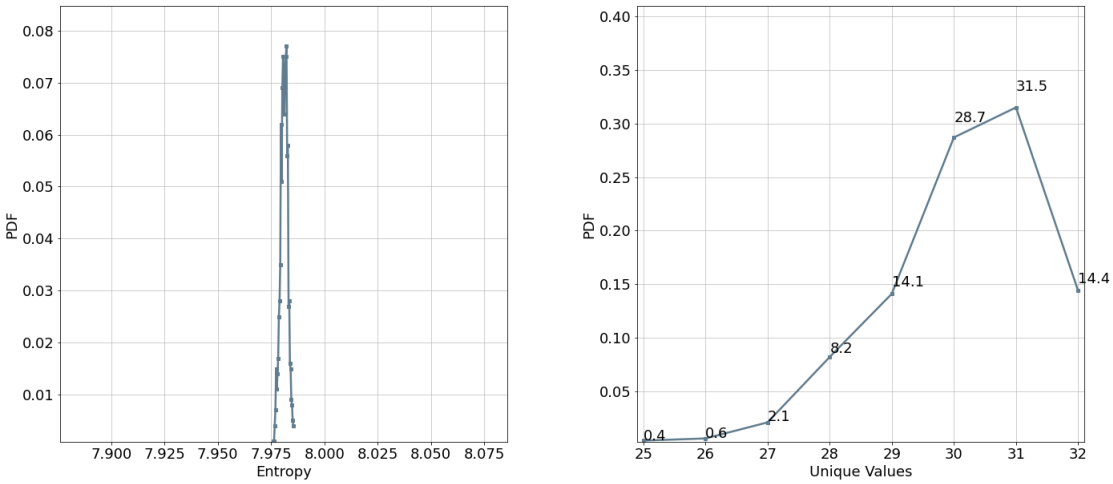

(a) (b) (c) (d)

Figure 9: The recurrence of the original message (a), ciphertext (b) produced by using the proposed MEAA for a random dynamic key (4 words per thread). The corresponding Probability Density Function (PDF) of the original message(c) and of the ciphertext (d).



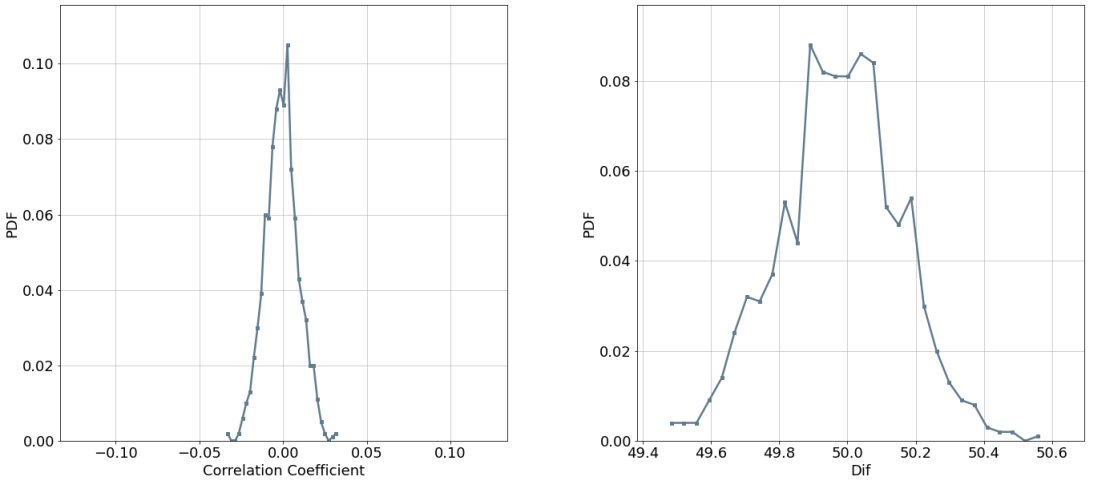

(a) (b)

Figure 10: Recurrence and histogram distribution of the whole MAC values (1000 MACs) in hexadecimal format for the proposed MEAA (a)-(b), respectively.




(a) (b)

Figure 11: PDF of the ciphertext entropy for 1000 times using the proposed third variant (a) at the message level and for a random dynamic key. PDF of the varying ASCII characters' number (in bytes) for 1000 times for the produced MACs (length equal to 256 bits) with the proposed MEAA. Also, percentage values are introduced in this figure.




(a) (b)

Figure 12: PDF of the correlation coefficient (a) and the percentage of the difference between original and encrypted messages (b) for 1,000 random dynamic keys for the proposed MEAA, respectively.

## 5.2 Tests Of Sensitivity

The sensitivity of the message and keys (avalanche effect) tests are used to verify that various MAC values are achieved whenever the message or secret key is slightly modified.

### 5.2.1 Sensitivity To Message Avalanche Effect

Two messages are considered in this test to be authenticated if they are identical except for a single bit. Given that the proposed approach is based on the notion of unique cryptographic primitives for each new

message, the key avalanche effect and the substitution diffusion cryptographic structure ensure the MAC value's sensitivity. The sensitivity test is verified by performing the proposed MEAA on message $M$ for 1,000 randomly chosen keys that measure the Hamming distance between the resultant MAC values when the same key is applied for two messages that vary by one bit. This distance is calculated as follows:

$$MS_w = \frac{\sum_{i=1}^{T} MEAA_{DK_w}(M_w) \bigoplus MEAA_{DK_w}(M'_w)}{Tb} \times 100\% \tag{2}$$

where $MEAA$ represents the proposed cryptographic solution, $MS_w$ is the acquired percentage of Hamming distance among the produced MAC (or ciphertext $C$) value by using the $w^{th}$ message $M_w$, $M'_w$ is the $w_{th}$ modified message, $DK_w$ is the employed dynamic-key; $Tb$ is the bit count of the MAC value for the authentication process and it is equal to the ciphertext bit length for the encryption process. In addition, a randomly selected byte's Least-Significant-Bit (LSB) was modified between $M_w$ and $M'_w$. The results of the message sensitivity test are similar to the ones shown in Figure 13. The key sensitivity and the statistical data are included in Table 2. The obtained results are comparable to those obtained using CMAC and HMAC. $M_S$ is roughly 50% of the time and follows a normal distribution, as shown by the plotted data with regard to 1,000 random keys. As a result, it is obvious that any slight alteration to the original data message will result in a significantly changed MAC value of roughly 50%.

#### 5.2.2 Sensitivity of Key Avalanche Effect

This test is used to determine the impact of a tiny variation in the dynamic-key $DK$ on the MAC values. All cryptographic primitives in the proposed approach are produced from $DK$. The expected behavior is for a single bit change in $DK$, which would result in a new collection of cryptographic primitives and, therefore, in a different MAC value. To verify this, a sensitivity test was performed on $DK$ using 1,000 randomly selected keys, measuring the Hamming distance between the produced MAC values for the same message when two keys differed by only one bit. This distance is calculated in the following manner:

$$KS_w = \frac{\sum_{i=1}^{T} MEAA_{DK_w}(M) \bigoplus MEAA_{DK'_w}(M)}{Tb} \times 100\% \tag{3}$$

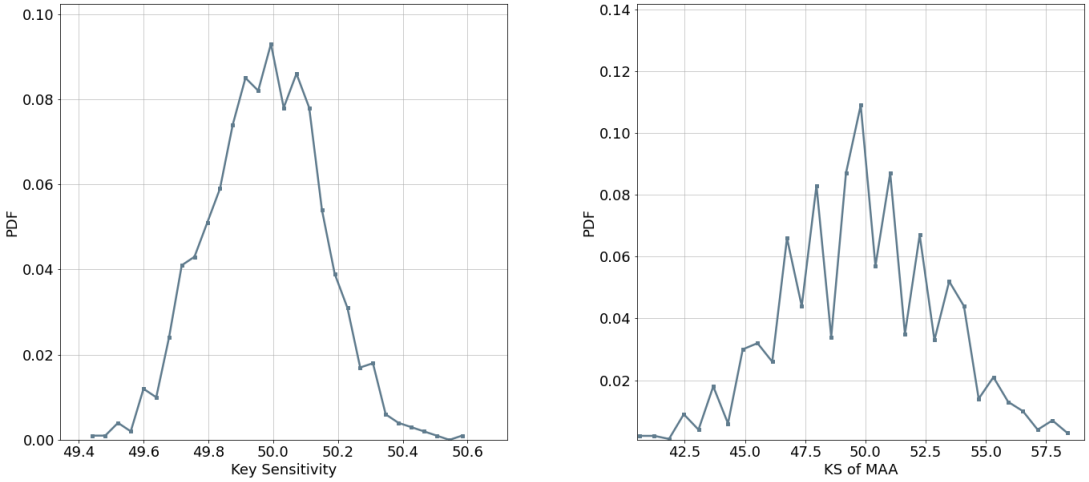
where $KS_w$ represents the ratio of Hamming distance acquired between the MAC value obtained using the $w_th$ dynamic key called $DK_w$ and the $w_{th}$ modified dynamic key $DK'_w$ with the same input message $M_w$. Furthermore, $MAC_w = MEAA_{DK_w}(M_w)$ and $MAC'_w = MEAA_{DK'_w}(M_w)$ are the MAC values which are computed using the dynamic keys $DK_w$ and $DK'_w$, respectively.

The difference between $DK_w$ and $DK'_w$ was examined in a randomly selected byte's Least-Significant-Bit (LSB). The results of the critical sensitivity tests are shown in Figure 13, and the statistical data are reported in Table 2. The achieved results are comparable to those obtained using CMAC and HMAC. The displayed results of $KS$ against 1,000 random keys indicate that KS is roughly 50% and has a normal distribution. As a consequence of the results, it is obvious that even a slight modification in the secret key or nonce will produce a very different MAC value for an identical message, with a difference of roughly 50%. On the other hand, the produced keystream reaches the sensitivity of the secret key, where comparable results are produced as in the case of MEAA.

### 5.3 The Resistance Of Collision

For this essential property, the possibility of two separate input data producing the same MAC value is determined. This test was run 1,000 times with 1,000 different dynamic keys, and each time a different bit in the message was changed. Then, for each variant, the MAC values of the original and modified messages were computed and compared. Also, the number of identical ASCII characters at the same locations was checked using the following equation:

$$Difference = \sum_{j=1}^{TB} DF\{MAC(j), MAC'(j)\}, \tag{4}$$




(a) (b)

Figure 13: PDF of the key sensitivity ($KS$) of the produced message encryption (a) and authentication (b) produced with 1000 random dynamic keys (after changing a random bit of the secret key).

Table 2: Statistical results for 1,000 random keys using the proposed MEAA

| **Proposed Message Encryption Algorithm** | | | | |
|---|---|---|---|---|
| | *Min* | *Mean* | *Max* | *Std* |
| $Dif$ | 49.483643 | 49.994884 | 50.594482 | 0.17627896 |
| $KS$ | 49.44214 | 49.99309 | 50.62256 | 0.17250678 |
| $H_E$ | 7.9760695 | 7.9812775 | 7.9858017 | 0.0016981743 |
| $\rho$ | -0.03311749 | 0.0004095363 | 0.033906624 | 0.009799017 |

| **Proposed Message Authentication Algorithm** | | | | |
|---|---|---|---|---|
| | *Min* | *Mean* | *Max* | *Std* |
| $KS$ | 40.625 | 50.190624 | 58.984375 | 3.1805573 |
| $H_{MAC}$ | 4.5 | 4.8850837 | 5.0 | 0.085197344 |
| $UV$ | 25 | 30.191 | 32 | 1.307 |

where $DF(x, y) = 1$ if $(x = y)$, else 0.

The MAC values of the original and changed messages are represented by $MAC$ and $MAC'$, respectively. Also, $MAC(j)$ corresponds to the $j^{th}$ ASCII character of $MAC$. The results are shown in Table 3, and it can be seen that only three characters at most are equal, for $Tb = 256$. This confirms that the proposed message authentication algorithm is resistant to collision and analytic attacks, including birthday attacks, meet-in-the-middle attacks, and differential attacks, according to [35].

# 6 Cryptanalysis

The preceding part has evaluated and described the proposed MEAA and stream-cipher in terms of high sensitivity, randomness, uniformity, and collision resistance. The cryptanalysis described in this part is used to demonstrate that the proposed method is robust to well-known message-authentication threats. This section discusses the cryptanalysis of the proposed MEAA by examining key and MAC-based attacks.

Table 3: Percentage of distribution of ASCII codes with the exact value at the same position in the MAC value for random LSB bit of the secret key when using $Tb$ =256 for the proposed message authentication algorithm.

| **Number of Hits** | | | | | |
|---|---|---|---|---|---|
| hits / $Tb$ | 0 | 1 | 2 | 3 | 4 |
| 256 | 87.6% | 11.2% | 1.2% | - | - |

## 6.1 Resistance To Statistical-Attacks

Unlike most of the current MAA solutions, the proposed MEAA is using the dynamic key approach, which incorporates both dynamic replacement and seeding for each input data set. Previous statistical tests, most notably "TestU01" and "Practrand", have proven that the proposed MEAA and the stream cipher are resilient to resistant to statistical attacks.

## 6.2 Resistance Against Brute-Force Attacks

The proposed authentication algorithm or stream cipher's secret-key space is very large, $2^{Tb}$, with the Tb being equal to at least 256 bits and perhaps 512 or 1024 bits. According to Schneier [1], to withstand brute force attacks, the key space should be greater than $2^{192}$, which is the case with the proposed approach. Due to the large space, brute-force attacks are impractical. This is also the same case with the dynamic key's key space, which is $2^{512}$. As a result, the proposed MEAA has along with its stream cipher a very large key space, which makes them also resistant to brute-force attacks.

## 6.3 Resistance To Pseudo-Collision

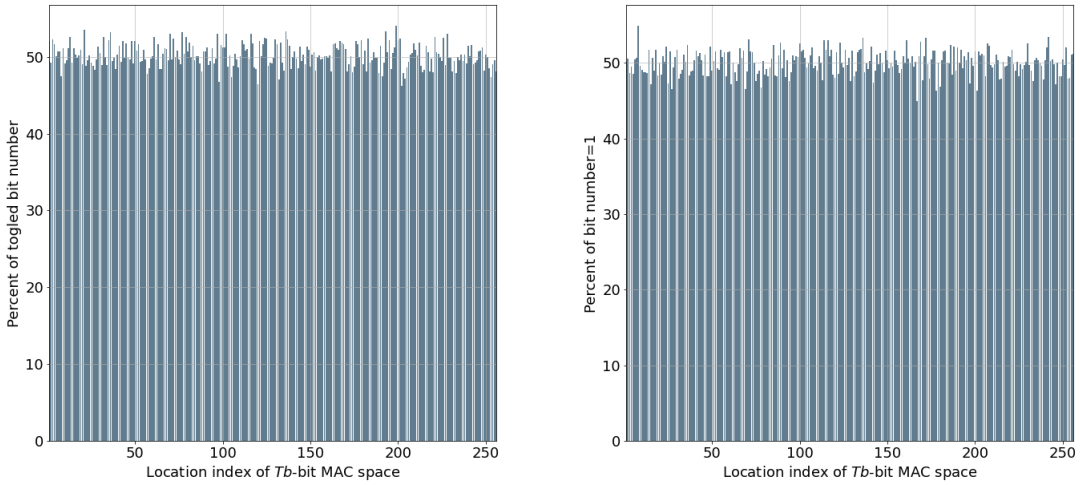
In the case of a pseudo collision attack, the attacker may sometimes attempt to change the message and the corresponding MAC value by exploiting the compression function vulnerabilities [36, 37]. Therefore, the proposed MEAA, which depends on the dynamic-key cryptographic primitives, includes a parallel encryption-authentication algorithm for the message blocks. This will ensure the avalanche effect. Additionally, the compression function is nonlinear in nature. This prevents attackers from obtaining meaningful details about a message using the received MAC, or from obtaining another message with the same MAC. Additionally, each new input message generates a unique key-stream sequence.

## 6.4 Resistance To Birthday Attacks

The Birthday attack is a classic attack that targets the MAA with the goal of locating two messages with identical MAC values N in less than $2^{\frac{N}{2}}$ trials (where $N$ equals $Tb$, the MAC's size) [38]. In the proposed MEAA, the lowest MAC size is 256, which requires brute force attacks to do $2^{128}$ attempts. As stated in [39], "if a suitable padding scheme is used and the compression function is collision-resistant, the hash function will be as well." In fact, the proposed MEAA is not only resistant to collision, but also immune to birthday attacks due to its huge MAC value.

## 6.5 Counter-Meet In The Middle-Attacks

The meet-in-the-middle attack's objectives are to identify a block $m_i$ that may/should replace the $nb$ blocks without affecting the produced MAC value [40, 41]. This is done with a data structure consisting of $nb$ blocks, $M = (m_1,\ m_2, \ldots,\ m_{nb1},\ m_{nb})$. Due to the fact that the proposed MEAA method employs changeable cryptographic primitives for each input data, such an attack is unfeasible. To verify this, a random data block $m_i$ was substituted for a random data block and the MAC values were calculated. This experiment was conducted with N = 1,000 times. The results, shown in Figure 14, indicate that the difference between the resulting MAC values is at least $\frac{Tb}{2}$ bits (50 percent of the MAC bits are adjusted) for $Tb = 256$ and that comparable results were achieved for larger values of $Tb$.




(a) (b)

Figure 14: The distribution percentage of the variation in bit number between $H_n^{''}$ and $H$ versus 1000 tests(a), (c) and percentage of each bit is set to be equal to 1 versus its corresponding index in MAC (b) and (d) for the proposed MEAA, respectively.

### 6.6 Selected/Known Attackers Using Plain-text/Cipher-text

The resistance to pre-selected or known message attacks is shown by the use of a dynamic key method, which significantly confuses the attacker's work. Consequently, the risks associated with a single message failure and inadvertent key exposure are eliminated. Additionally, contemporary attacks are ineffective since any modification to the dynamic key would result in a considerable variation in the generated cryptographic primitives and MAC. Additionally, the key sensitivity study revealed a significant level of susceptibility to key-related attacks. The results indicate that no valuable information can be deduced from the MAC or the generated key stream.

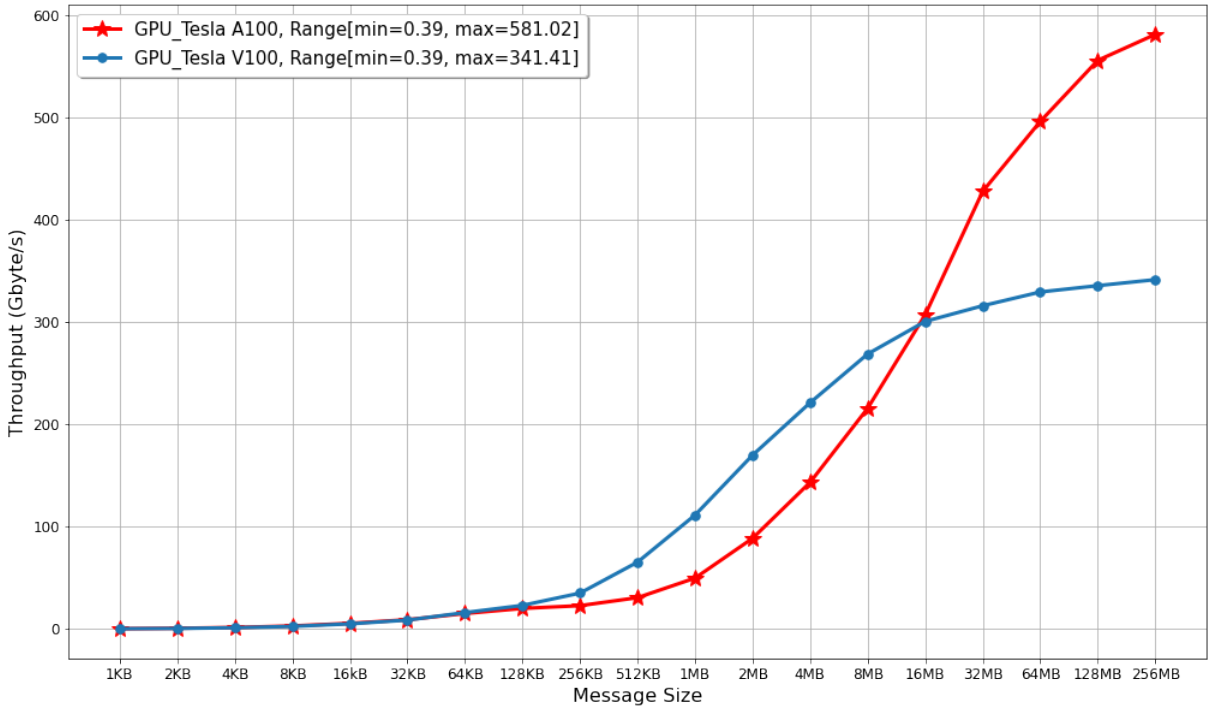
## 7 Performance Results

This section evaluates and compares the suggested approach to a variety of encryption methods which mainly use GPU and CPU devices. The experiments were performed on a Linux/Debian system. To develop and implement the cryptographic algorithm kernels, CUDA version 11.3 is utilized.

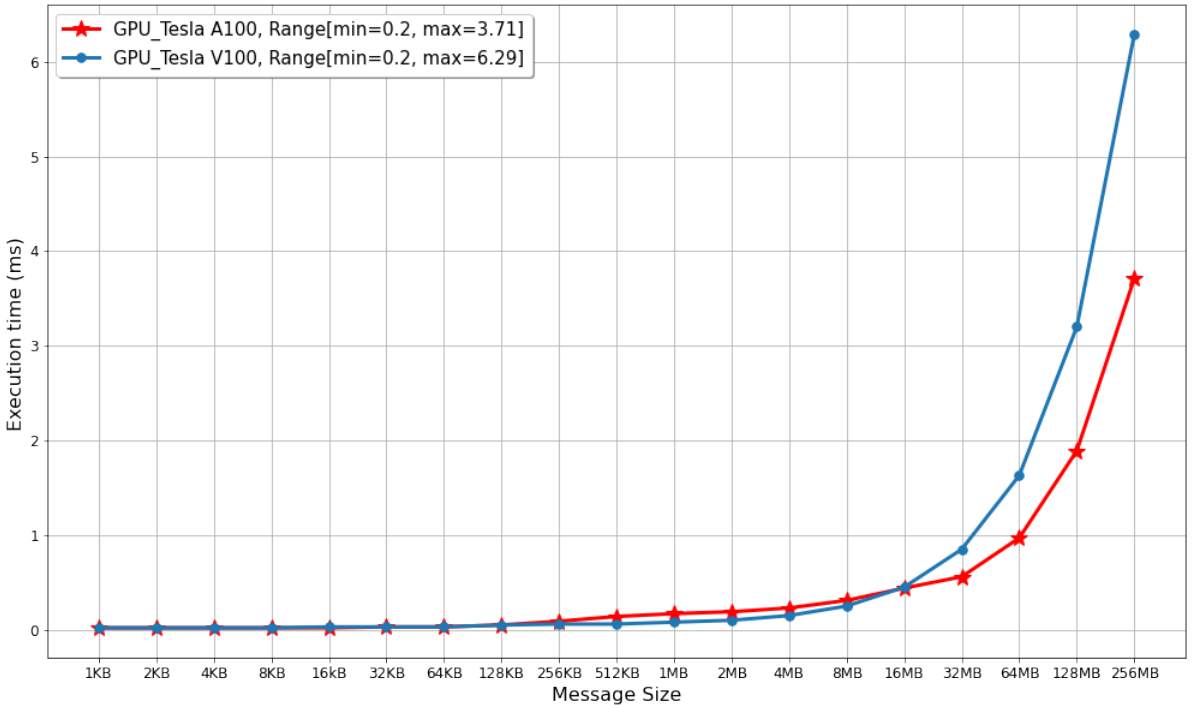
### 7.1 Performance Evaluation Of Proposed MEAA Over GPUs

To acquire the necessary cryptographic features, such as message and key avalanche, the previous MAA methods have used a multiple-round compression mechanism along with several operations per round. Existing MAAs include the CMAC and CCM operational modes, which depend on the AES encryption method [29], which needs a minimum of ten rounds for a secret key size of 128 bits. Another example of MAA is a keyed hash function, such as HMAC, which employs SHA-variants and hence requires a large number of rounds (which are variant dependent).

Additionally, these techniques take advantage of the chaining operation mode, which constrains GPU's parallelism. As a result, AES and HMAC have a longer latency than the proposed MEAA method. Furthermore, the proposed MEAA method depends on the dynamic-key technique, which is known to be resistant to attacks. This may help in reaching the key avalanche effect, which in turn will help with achieving the message avalanche effect and all other essential cryptographic features, all in a single round. In brief, the unique aspect of this study is the straightforward architecture of the proposed MEAA, which was optimized for

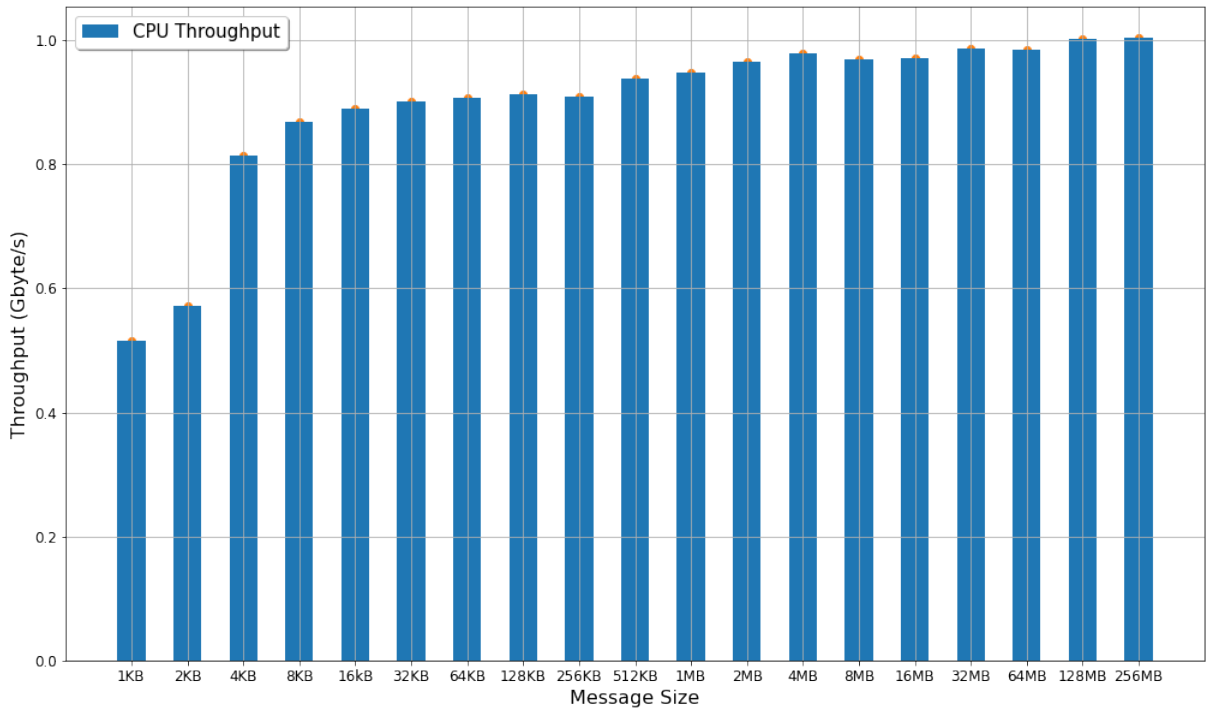



(a) Throughput results over GPUs

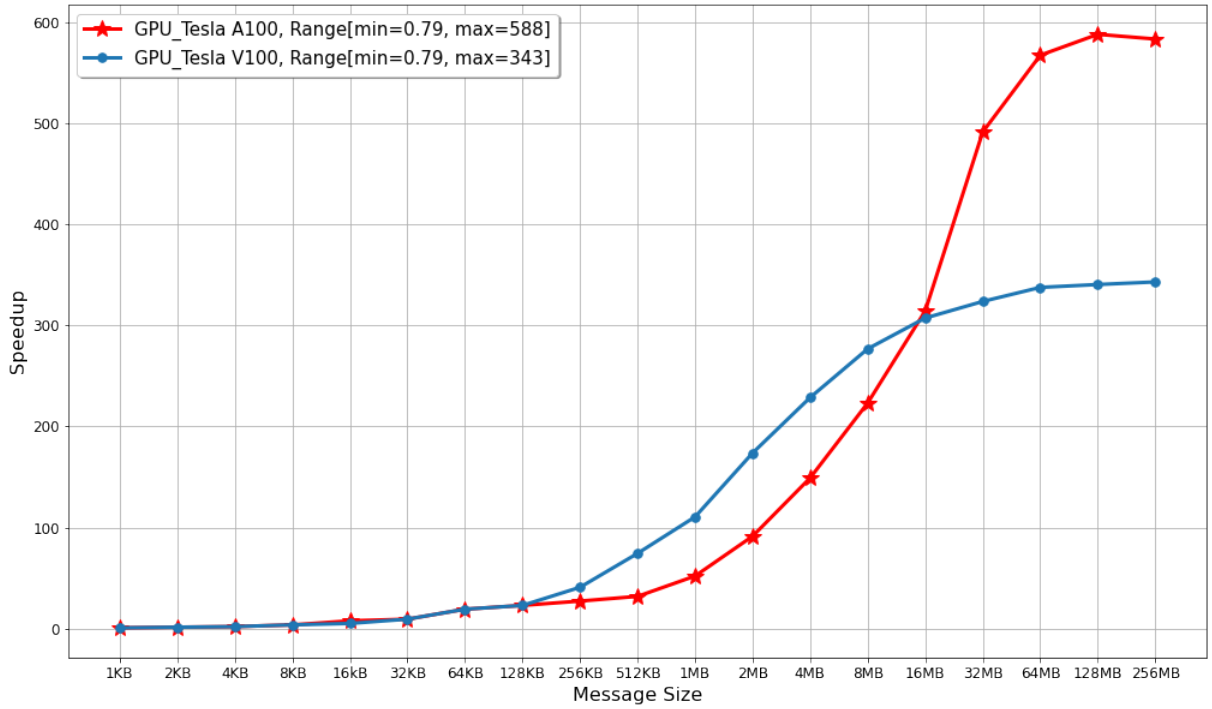



(b) Execution time results over GPUs

Figure 15: Throughput and execution time analysis for the proposed MEAA over two GPU devices




(a) Throughput results over CPU




(b) speedup of the execution time compassion between GPUs and CPU

Figure 16: The results of the proposed MEAA over CPU device (Intel CPU i7-7700HQ)

Table 4: GPU devices characteristics

| GPU Device | Characteristics |
|---|---|
| Tesla A100 | • Capability of computing: 8.0<br>• Global memory: 40000 MB<br>• GPU core frequency: 1.41 GHz<br>• Frequency of memory: 1215 MHz<br>• CUDA cores' numbers: 6912 |
| Tesla V100 | • Capability of computing: 7.0<br>• Global memory: 32510 MB<br>• GPU core frequency: 1.53 GHz<br>• Frequency of memory: 877 MHz<br>• CUDA cores' numbers: 5120 |

optimal throughput using GPU features. In the following, the execution time latency and throughput of the proposed MEAA as a function of message length are presented. The proposed MEAA's performance is examined using a Tesla V100 and Tesla A100 GPUs. Table 4 demonstrates these GPUs computing characteristics.

The proposed MEAA's average throughput and execution time are shown in Figure 15. The captured results show that the proposed MEAA is much quicker than currently used MAAs. As a consequence, the proposed MEAA is more suited to real-time and high-data-rate applications where GPU computing is conceivable.

## 7.2 Comparison of The Proposed Encryption Method To Other Ciphers

The performance of the proposed encryption method, illustrated in Algorithm 2, is compared to other stream ciphers implemented over GPUs. For a fair comparison between existing stream ciphers and the one proposed in this study, only the ciphers using CTR mode and implemented on GPU devices were selected in this work. In order to make a fair comparison with other algorithms, it should be noticed that currently, to the best of our knowledge, there are no other existing GPU algorithms that are able to simultaneously encrypt and authenticate messages. That is why the comparison focuses on other existing algorithms to encrypt messages on GPUs. In the following, some recent works are explained and the comparison between our work and other existing works is illustrated.

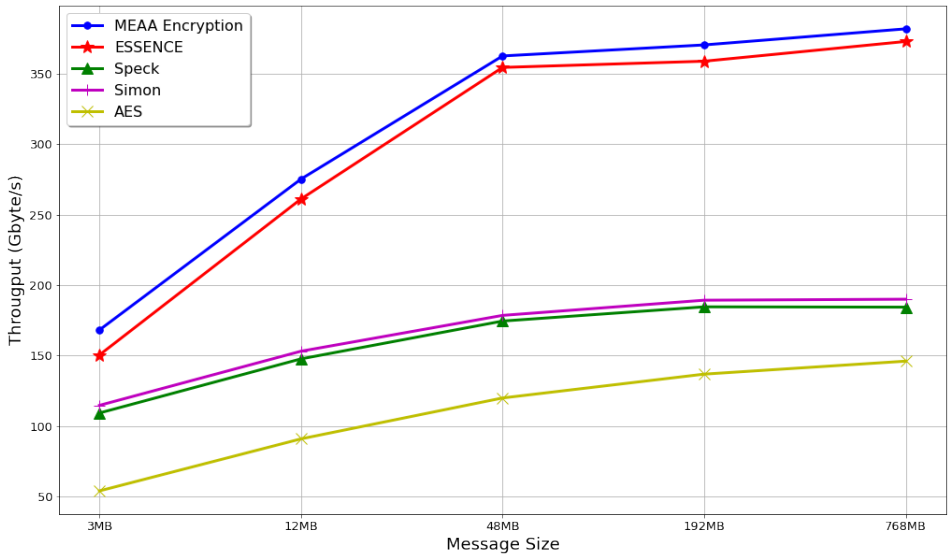
The recent work which is called ESSENCE [42] is used in this comparison since it is designated for GPU devices. ESSENCE uses one round function to encrypt and decrypt the plain text message by applying the substations tables, xorshift128+ and xorshift64 PRNGs. Moreover, ESSENCE is tested with different message sizes, from 3 MB to 768 MB by an increasing factor 4. Other stream-ciphers that apply the CTR mode and are implemented over GPUs are also used in this study such as Speck [43], Simon [43] and AES [10].

Speck, Simon, and AES process four blocks per thread in their GPU implementation. By contrast, these lightweight ciphers use many rounds and multiple operations per thread. For a secret key size of 128 bits, the round numbers are 32, 68, and 10 for Speck, Simon, and AES, respectively. Speck and Simon ciphers are both based on fundamental processes that are widely used in a number of computer systems (AND, rotate, XOR, and modular addition). However, since Simon is a hardware implementation, its findings exhibit varying throughput rates depending on the employed GPU hardware.

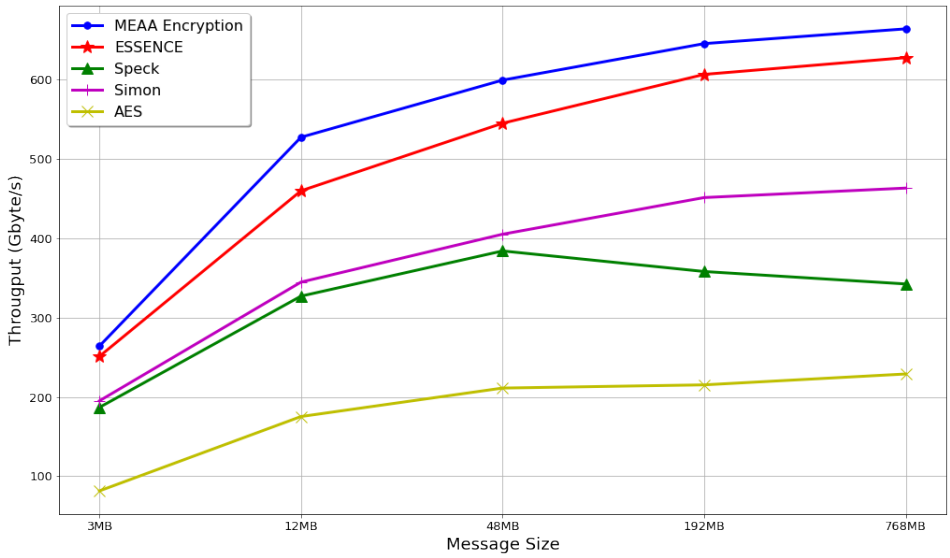
Figure 17 shows the throughput comparison results of the proposed MEAA's encryption to all other

ciphers over NVIDIA Tesla V100 and A100 GPUs. Thus, the proposed method, which is constructed in accordance with the GPU's specifications, provides the highest throughput across all GPU devices used and is more responsive to real-time applications than current ciphers, which is the primary objective of this study. Moreover, tables 5 and 6 present the speedup ratios of the proposed cipher in comparison with others.

Additionally, the grain size of the proposed one-round function of MEAA that encrypts or decrypts a number of words per thread is investigated in this section. Therefore, the best grain size that gives the best throughput is equal to 64 words per thread as shown in Figure 18. This grain size is used in all the comparison results as previously explained in this section. These results can be seen in Figure 17.




(a)




(b)

Figure 17: The throughput results of the proposed cipher and other existing algorithms (a) over GPU Tesla V100 and (b) over GPU Tesla A100

Table 5: Speedup comparison between the proposed solution and related ones over GPU Tesla V100

| Message size | Speedup ratio compare to | | | |
|---|---|---|---|---|
| | ESSENCE | Simon | Speck | AES |
| 3 MB | 1.12 | 1.47 | 1.54 | 3.11 |
| 12 MB | 1.05 | 1.80 | 1.86 | 3.02 |
| 48 MB | 1.02 | 2.03 | 2.08 | 3.02 |
| 192 MB | 1.03 | 1.96 | 2.01 | 2.71 |
| 768 MB | 1.02 | 2.01 | 2.07 | 2.61 |

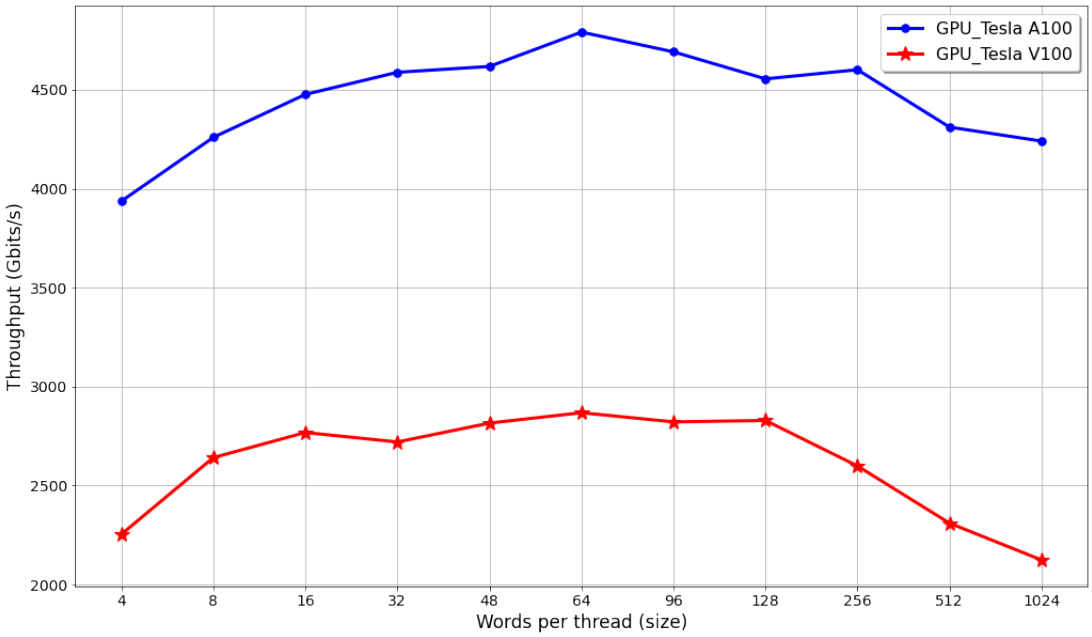



Figure 18: The throughput results of words per thread for a message size of 48 MB (64 words per thread is the best) for the proposed MEAA.

Table 6: Speedup comparison over NVIDIA GPU Tesla A100

| Message size | Speedup ratio compare to | | | |
|---|---|---|---|---|
| | ESSENCE | Simon | Speck | AES |
| 3 MB | 1.05 | 1.35 | 1.41 | 3.23 |
| 12 MB | 1.23 | 1.65 | 1.73 | 3.23 |
| 48 MB | 1.10 | 1.48 | 1.56 | 2.84 |
| 192 MB | 1.06 | 1.43 | 1.80 | 3.00 |
| 768 MB | 1.06 | 1.43 | 1.94 | 2.90 |

### 7.3 Performance Evaluation Of Proposed MEAA Over CPU

This section examines the sequential execution of the proposed MEAA on the Intel CPU i7-7700HQ, which operates at a 2.8 GHz frequency and has 16 GB of main memory. The throughput is measured sequentially for each instance of message size on the CPU, as shown in Figure 16-(a). Moreover, it indicates that the throughput ranged from 0.5 to 1 Gbytes per second for all message sizes. Additionally, the speedup ratio is calculated by comparing the CPU execution time to the GPU execution time when the proposed MEAA is executed, as shown in Figure 16-(b). It demonstrates that the GPU Tesla A100 outperforms the CPU by 583 times when applying MEAA to a 250 MB message, while the Tesla V100 GPU is 343 times faster compared to the previous instance.

## 8 Conclusion

As mentioned previously, existing message encryption and authentication methods were not designed for GPU utilization, which makes it impossible to benefit from the parallel processing capabilities of GPUs. Furthermore, these solutions require a large number of operations and rounds, resulting in increased computational complexity and delays. As a result, in this work, a robust and efficient parallel message encryption and authentication method is designed according to the GPU characteristics to effectively address the demands of big data applications. Moreover, the proposed solution relies on the dynamic key-dependent approach to prevent implementation attacks and to reduce the required number of operations and rounds, which will achieve a more favorable balance between security and efficiency. Moreover, the originality of this work is validated as, to the best of our knowledge, this is the first time an MEAA solution has been specifically designed for GPU implementation. In addition, the proposed MEAA beats the most optimized AES implementations on GPU, making it a better choice for real-time applications. Additionally, the proposed scheme provides a high level of randomness, which was shown by computing the MAC (or keystream, in the event of using the proposed MEAA as a stream-cipher) that passed the “TestU01" and the “Practrand" statistical tests.

On the other hand, cryptanalysis and benchmark tests have been done to validate the proposed solution's robustness and efficiency, respectively. Notably, other cryptanalysis approaches are targeted at static structures, while the proposed one is based on the variable structure. Finally, performance experiment results demonstrate that the proposed MEAA solution when compared to related existing ones, has the lowest execution time costs of message authentication and encryption and consequently the maximum throughput. This validates our main idea that we achieve optimal performance by exploiting better the parallel processing capabilities of the GPU. Furthermore, the design of an efficient and robust multimedia crypto-compression scheme over GPU will be our future work.

## Compliance with Ethical Standards


- **Funding:** This paper is partially supported 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 would also like to thank the Mésocentre de Calcul de Franche-Comté and the GENCI-IDRIS (Grant 20XX-AD011012913) for their supercomputer resources.


- **Conflict of interest/Competing Interests:** The authors declare that they have no conflict of interest or competing interests.
- **Ethical approval:** This article does not contain any studies with human participants or animals performed by any of the authors.

## References


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

[2] Aakanksha Tewari and Brij B Gupta. Secure timestamp-based mutual authentication protocol for iot devices using rfid tags. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 16(3):20–34, 2020.

[3] Nag Mani, Melody Moh, and Teng-Sheng Moh. Defending deep learning models against adversarial attacks. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 13(1):72–89, 2021.

[4] Brij B Gupta, Kuan-Ching Li, Victor CM Leung, Kostas E Psannis, Shingo Yamaguchi, et al. Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system. *IEEE/CAA Journal of Automatica Sinica*, 8(12):1877–1890, 2021.

[5] Zhili Zhou, Yuecheng Su, Yulan Zhang, Zhihua Xia, Shan Du, Brij B Gupta, and Lianyong Qi. Coverless information hiding based on probability graph learning for secure communication in iot environment. *IEEE Internet of Things Journal*, 9(12):9332–9341, 2021.

[6] Ivan Cvitić, Dragan Peraković, Marko Periša, and Brij Gupta. Ensemble machine learning approach for classification of iot devices in smart home. *International Journal of Machine Learning and Cybernetics*, 12(11):3179–3202, 2021.

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] Like Chen and Runtong Zhang. A Key-dependent Cipher DSDP. In *Electronic Commerce and Security, 2008 International Symposium on*, pages 310–313. IEEE, 2008.

[17] 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.

[18] Hassan N. Noura, Ali Chehab, Lama Sleem, Mohamad Noura, Raphaël Couturier, and Mohammad M. Mansour. One round cipher algorithm for multimedia iot devices. *Multim. Tools Appl.*, 77(14):18383–18413, 2018.

[19] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. Simon and speck: Block ciphers for the internet of things. *IACR Cryptology ePrint Archive*, 2015:585, 2015.

[20] 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.

[21] Hassan N 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.

[22] 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.

[23] Ahmed Fanfakh, Hassan N. Noura, and Raphaël Couturier. ORSCA-GPU: one round stream cipher algorithm for GPU implementation. *J. Supercomput.*, 78(9):11744–11767, 2022.

[24] 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.

[25] 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.

[26] Tom St Denis. *Cryptography for Developers*. Syngress, 2007.

[27] Yehuda Lindell Jonathan Katz. *Introduction To Modern Cryptography*. Chapman & Hall/CRC Cryptography And Network Security. CRC Press/Taylor & Francis Group, 3rd edition edition, 2021.

[28] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. 2015 52nd acm/edac/ieee design automation conference (dac). pages 1–6, 2015.

[29] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[30] Hassan N. Noura, Ola Salman, Raphaël Couturier, and Ali Chehab. Novel one round message authentication scheme for constrained iot devices. *Journal of Ambient Intelligence and Humanized Computing*, 2021.

[31] Tolga Soyata. *GPU Parallel Program Development Using CUDA*. Chapman & Hall/CRC Computational Science. Chapman and Hall/CRC, 1 edition, 2018.

[32] Subhamoy Maitra. Chosen iv cryptanalysis on reduced round chacha and salsa. *Discrete Applied Mathematics*, 208:88–97, 2016.

[33] 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.

[34] Lama Sleem and Raphaël Couturier. Testu01 and practrand: Tools for a randomness evaluation for famous multimedia ciphers. *Multimedia Tools and Applications*, 79(33):24075–24088, 2020.

[35] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *In EUROCRYPT*. Springer-Verlag, 2005.

[36] Amir Akhavan, Azman Samsudin, and Afshin Akhshani. A novel parallel hash function based on 3d chaotic map. *EURASIP Journal on Advances in Signal Processing*, 2013(1):1–12, 2013.

[37] B. Yang, Z. Li, S. Zheng, and Y. Yang. Hash function construction based on coupled map lattice for communication security. In *Global Mobile Congress 2009*, pages 1–7, Oct 2009.

[38] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[39] Ivan Damgård. A design principle for hash functions. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 416–427, London, UK, UK, 1990. Springer-Verlag.

[40] Mohamed Amin, Osama S Faragallah, and Ahmed A Abd El-Latif. Chaos-based hash function (cbhf) for cryptographic applications. *Chaos, Solitons & Fractals*, 42(2):767–772, 2009.

[41] A Kanso and M Ghebleh. A structure-based chaotic hashing scheme. *Nonlinear Dynamics*, 81(1-2):27–40, 2015.

[42] 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.

[43] 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.

# A Appendix

## A.1 The Proposed MEAA Implementation On The GPU

The proposed MEAA makes use of multi-block Cooperative Groups for a single-pass reduction (CG). This new functionality was added by Nvidia in Cuda 9.0. CG to provide a scalable collaboration between groups of threads. The next sections describe how CG are used to build an efficient GPU version of the proposed MEAA. It is worth noting that only the version with a 256-bit input block is presented. The following are the main stages of the proposed MEAA, as shown in Algorithm 2:

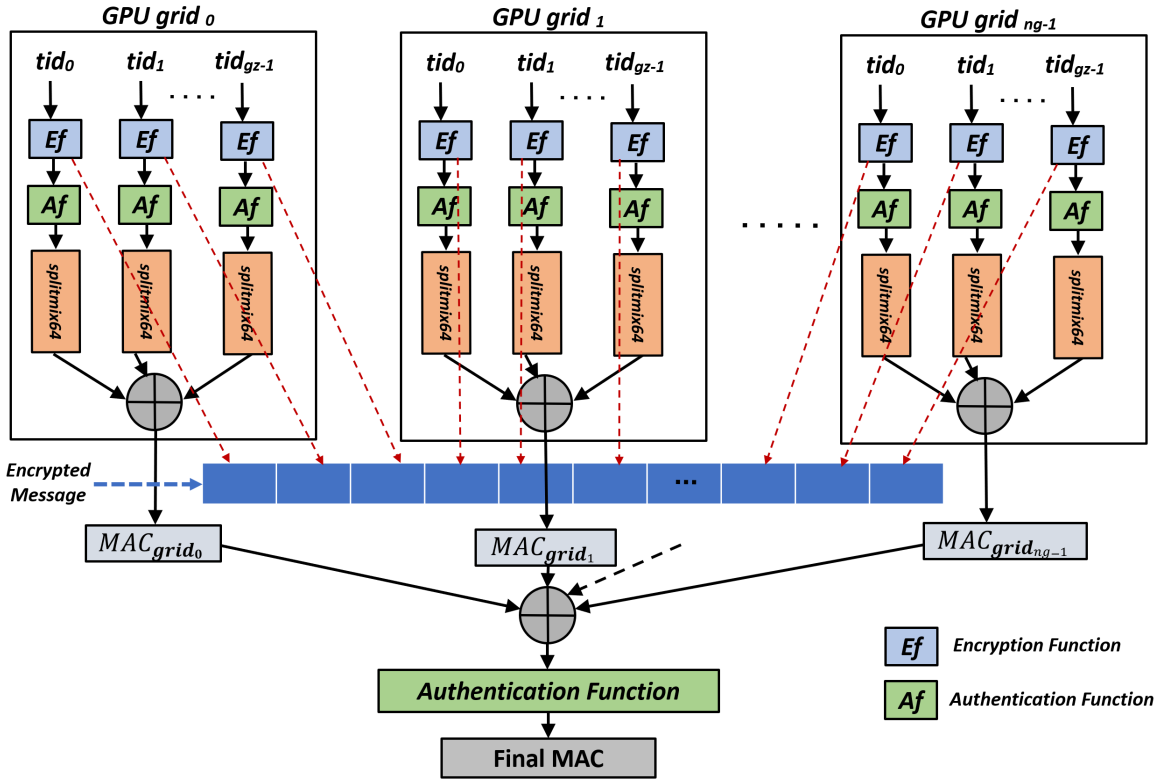



Figure 19: The corresponding GPU implementation of the proposed MEAA

- Two variables of the shared memory are used for each block. Each one has the same size as the number of threads for each block.
- Line 11 initializes the shared arrays $svector1$ and $svector2$.
- Lines 12 to 13 parallelize the copy of $DK$ variable values into the shared memory array $SharedDK$. It is critical to use shared memory for this variable in order to speed up memory accesses operations.
- All block threads are synchronized in line 14.
- Line 16 starts the first loop. It iterates over all of the items contained in the global variable $g\_ivector$. The CG is used here to assign each thread of each block to a separate section of this array. The increment value for each iteration of this loop is the size of a grid where the total number of threads is utilized (the number of blocks multiplied by the number of threads per block).
- In Line 17, the encryption kernel function is called. The algorithm 2 encrypts each word of the initial vector $IV$ with a dynamic key $DK_i$. The algorithm begins by XORing the thread $id$ with the $sharedDK$ value, which is then cast to 64 bits and saved in variable $r1$. The array $rr$ stores the first

and second 32 bits of the variable $r1$. The initial vector value is divided into two parts of 32 bits each and stored in the array $rr2$. The dynamic key $K[rr[0]\&511]\&63$ rotates the first element of the array $rr2$ to the left (applying 64 bits). The result of this step is XORed with the first element of the array $rr$. The dynamic key $K[rr[1]\&511]\&63$ rotates the second element of the array $rr2$ to the left (using 64 bits) and XORs it with the second element of the array $rr$. Finally, the obtained word of 64 bits is sent to "*Splitmix-64*" PNRG algorithm 5 to get a new number.

- Line 18, each thread will save the encrypted word which resulted from the EncryptFun function in $g\_encvector$ global array.
- The If statement is executed in lines 20-23 to indicate that the algorithm works to encrypt-authenticate (if *dec* variable is false) or decrypt-authenticate (otherwise).
- One thread out of four runs the block beginning with line 24. Due to the inherent synchronization of threads on the same warp, all groups of four successive threads are synchronized. The authentication function is called to apply over the four blocks (see algorithm 3).
- Explicit synchronization to warp threads is required to be implemented at line 29.
- Line 30 calls the PRNG "*Splitmix-64*" to generate a new 64-bit value for the shared memory array $svector2$. As a result, each block thread will XOR the updated PRNG value into the shared memory array $svector1$.
- The seed is modified for the next iteration to be equal to the generated replacement word in line 32. This implies that the keystream is generated in a chaining fashion, with the previous iteration's result. Thus, the keystream becomes the input for the next iteration.
- Beyond the first loop, it is necessary to synchronize all threads inside a block as in line 33. The kernel "*BlockReduce*" function is then invoked. The following steps provide an explanation for the "*BlockReduce*" kernel provided in Algorithm 4.
- After this kernel, lines 36 to 39 are executed by the first thread in each block. These lines are used to store in the global memory the four 64-bit values corresponding to the message-authentication code for the block-related content, at the place specified by the block ID (Identification).
- Line 40 shows the synchronization of the whole grid. The second loop is then executed by the first thread only in the whole grid. This loop attempts to XOR all of the blocks' four 64-bit integers together. As a result, the whole text's message authentication is calculated (see Figure 19).
- While threads on the same grid are necessarily synchronized, each set of four successive threads is synchronized. Lines 43 to 46 are used to xor each global memory array element $g\_ovector$.
- Line 48 calls the authentication function to mix four blocks of shared memory array $g\_ovector$.
- The step in line 50 is taken to decrease the necessary amount of XORs (optimization). This step's output is saved to the shared memory $svector1$ to fulfill the compression operation.

Algorithm 4 tries to compute the encrypted message and the message authentication code for a single block based on the text message contained inside this block. The kernel's premise is to xor all four 64-bit values. The first loop (as in line 8) is run concurrently on all 32 threads which is the exact size of a wrap. The number of "XOR" operations is divided into two for each iteration in the loop. Because certain threads fail to perform the condition in line 9, synchronization is necessary (line 14). The XOR between the elements $threadid$ and $threadid + i$ in lines 10 to 12 were calculated. Line 16 synchronizes all the block's threads. Then, only the block's first four threads are used to perform the XOR operation on all 32 items. Finally, the obtained result is subjected to the "*Splitmix-64*" PRNG.

The provided MEAA technique here is able to create a stream cipher capable of producing a keystream with a high degree of unpredictability and uniformity. Figure 19 illustrates the proposed parallel encryption and authentication of messages over GPUs. The round function is the sole difference between this system

**Algorithm 1** The proposed MEAA kernel function

```
__global__ void MEAACG(const ulong * __restrict__ g_ivector, ulong * __restrict__ g_encvector,
        ulong * __restrict__ g_v, ulong * __restrict__ g_ovector, unsigned int n,
        uchar * __restrict__ DK, bool dec)
{
    cg::thread_block block = cg::this_thread_block();
    cg::grid_group grid = cg::this_grid();
    __shared__ uchar sharedDK[512];
    extern ulong __shared__ svector[];
    ulong *svector1 = (ulong*)svector;
    ulong *svector2 = (ulong*)&svector[block.size()];
    svector1[block.thread_rank()] = 0; svector2[block.thread_rank()] = 0;
     if(block.thread_rank()<512)
         sharedDK[block.thread_rank()]=DK[block.thread_rank()];
    __syncthreads();
    ulong v2=g_v[block.thread_rank()];
    for (int i = grid.thread_rank(); i < n; i+= grid.size()) {
      ulong r1 = EncryptFun(sharedDK, v2, i);
      r1=r1^ g_ivector[i];
      g_encvector[i] = r1;
     if (dec)
        svector2[block.thread_rank()]= g_ivector[i];
     else
        svector2[block.thread_rank()]= r1;
     if ((block.thread_rank()&3) == 0) {
       int bl_id=block.thread_rank();
       AuthFun(&svector2[bl_id],&svector2[bl_id+1],&svector2[bl_id+2],
       &svector2[bl_id+3], i, sharedDK);
     }
     __syncthreads();
     svector1[block.thread_rank()]^=splitmix-64(svector2[block.thread_rank()]);
    }
    g_v[block.thread_rank()]= v2;
    cg::sync(block);
    BlockReduce(svector1, block);
    if (block.thread_rank() == 0) {
      g_ovector[blockIdx.x*szblock] = svector1[0];
      g_ovector[blockIdx.x*szblock+1] = svector1[1];
      g_ovector[blockIdx.x*szblock+2] = svector1[2];
      g_ovector[blockIdx.x*szblock+3] = svector1[3]; }
    cg::sync(grid);
    if (grid.thread_rank() == 0) {
     for (int i = 1; i < gridDim.x; i++) {
            g_ovector[0] ^= g_ovector[i*4];
            g_ovector[1] ^= g_ovector[i*4+1];
            g_ovector[2] ^= g_ovector[i*4+2];
            g_ovector[3] ^= g_ovector[i*4+3]
            }
        AuthFun(&g_ovector[0],&g_ovector[1],&g_ovector[2],
        &g_ovector[3],1, sharedDK);
        svector1[0]=g_ovector[0]^g_ovector[1]^g_ovector[2]^g_ovector[3];
        g_ovector[0]=g_ovector[0]^svector1[0];
        g_ovector[1]=g_ovector[1]^svector1[0];
        g_ovector[2]=g_ovector[2]^svector1[0];
        g_ovector[3]=g_ovector[3]^svector1[0];
    }
 }
```

and the planned MEAA scheme. The proposed stream cipher has the advantage of being well-designed to make use of the GPU's parallel structure while still requiring just one round iteration with basic operations. This results in a higher key-stream throughput with reduced latency as compared to previous methods such as AES-CTR, which need multiple rounds.

**Algorithm 2** The proposed encryption kernel function

```
__device__ ulong EncryptFun(uchar *K, ulong v, int id)
{
        ulong r1=((id)^(((ulong*)K)[id&63]));
        uint32_t  *rr=(uint32_t*)&r1;
        ulong  r2 = v;
        uint32_t  *rr2=(uint32_t*)&r2;
        rr[0]^=ROTL32( rr2[0], K[rr[0]&511]&63 );
        rr[1]^=ROTR32( rr2[1], K[rr[1]&511]&63 );
        r1=splitmix64(r1);
        return r1;
}
```

**Algorithm 3** The proposed authentication kernel function

```
__device__ void  AuthFun(ulong *s1, ulong *s2, ulong *s3,
ulong *s4, int id, uchar *DK)
{
  *s1+=*s2+DK[id&63];
  *s4=ROTR64(*s4^*s1,32);
  *s3+=*s4+DK[(id+10)&63];
  *s2=ROTR64(*s2^*s3,24);
}
```

## A.2 Splitmix64

"*Splitmix-64*" PRNG provides a number of benefits, including a very small execution time and a straightforward implementation. Algorithm 5 presents the $SplitMix64$ algorithm, which is a fast splittable PRNG. It features a 64-bit input and output state. The efficiency of $SplitMix64$ for time-critical applications has been shown using two common statistical randomized test suites (DieHarder and TestU01). However, it is not recommended for cryptography or security applications since the sequences of pseudo-random values generated are predictable due to the invertibility of the mixing functions and the fact that two consecutive outputs have enough information to rebuild the internal state. $SplitMix64$ is employed with dynamic seeds in the proposed solution which builds a linear transformation based on shift/rotate. This results in a large decrease in latency and a great degree of randomization in the relevant resources. As a result, the proposed MEAA system employs a large number of threads, with each thread iterating a "*Splitmix-64*" PRNG with a distinct seed.

**Algorithm 4** The proposed block reduction kernel function("*BlockReduce*")

```
__device__ void BlockReduce(ulong *svector, const cg::thread_block &blk)
{
    const unsigned int thread_id = blk.thread_rank();
    cg::thread_block_tile<32> tile32 = cg::tiled_partition<32>(blk);
    ulong bt  = svector[thread_id];
    ulong tmp;

    for (int i = tile32.size()/ 2; i >= szblock; i >>= 1) {
        if (tile32.thread_rank() < i) {
            tmp       = svector[(thread_id+i)];
            bt       ^= tmp;
            svector[thread_id] = bt;
        }
        cg::sync(tile32);
    }
    cg::sync(blk);
    if (blk.thread_rank() <4) {
      bt  = 0;
      for (int i = thread_id; i < blockDim.x; i += tile32.size()) {
        bt  ^= svector[i];
      }
      svector[thread_id] = bt;
    }
    cg::sync(blk);
}
```

**Algorithm 5** The Kernel Function of the *SplitMix-64* PRNG

```
__device__ static  ulong splitmix-64(ulong idx) {
  ulong H = (idx + UINT64(0x9E3779B97F4A7C15));
  H = (H ^ (H >> 30)) * UINT64(0xBF58476D1CE4E5B9);
  H = (H ^ (H >> 27)) * UINT64(0x94D049BB133111EB);
  return H ^ (H >> 31);
}
```