Cryptography Using GPGPU

Swati Jadhav, Uttkarsh Patel, Atharv Natu, Bhavin Patil, and Sneha Palwe

Abstract Today, with an ever-increasing number of computer users, the number of cyberattacks to steal data and invade privacy is of utmost importance. A group of applications uses the Advanced Encryption Standard (AES) to encrypt data for security reasons. This mainly concerns enterprises and businesses, which ultimately handle user data. But many implementations of the AES algorithm consume large amounts of CPU horsepower and are not up to the mark in terms of throughput. To tackle this problem, the proposed system makes use of GPUs, which are targeted for parallel applications. These enable parallel operations to be performed much faster than the CPU, ultimately increasing throughput and reducing resource consumption to some extent. The vital aspect of this approach is the speedup that is achieved due to massive parallelism. This research aims to implement AES encryption and decryption using CUDA and benchmark it on various compute devices.

Keywords Cyberattacks · AES · CPU · Throughput · GPUs · SIMD · Parallel · Massive parallelism · CUDA · Compute devices

S. Jadhav (✉) · U. Patel · A. Natu · B. Patil · S. Palwe
Vishwakarma Institute of Technology, Pune, Maharashtra, India
e-mail: swati.jadhav@vit.edu

U. Patel
e-mail: patel.uttkarsh21@vit.edu

A. Natu
e-mail: atharv.natu21@vit.edu

B. Patil
e-mail: bhavin.patil21@vit.edu

S. Palwe
e-mail: palwe.sneha21@vit.edu

G. Rajakumar et al. (eds.), *Intelligent Communication Technologies and Virtual Mobile Networks*, Lecture Notes on Data Engineering and Communications Technologies 171,
https://doi.org/10.1007/978-981-99-1767-9_23

1 Introduction

Cybersecurity is critical these days, with cybercrime and cyberattacks carried out by malicious users and hackers on the rise. To protect our data, encryption algorithms are used, like Advanced Encryption Standard (AES), which was developed to tackle security problems found in Data Encryption Standard (DES) in 2001 by the National Institute of Standards and Technology (NIST). AES is a block cipher-based encryption technique. Even after 21 years, AES still withstands all cyberattacks and is also considered to be arguably the most used encryption algorithm. The widespread use of AES led to the development of many optimized implementations for a variety of CPU architectures.

Traditionally, graphical processing units (GPUs) are used by enthusiasts or developers for game playing or development, respectively, video rendering, and all operations that require a considerably large amount of video memory and dedicated processing. GPU architectures work on single instruction multiple data (SIMD) which allows them to execute the same instruction on multiple data streams in parallel fashion. This is also known as "massively accelerated parallelism" [1–5].

The main motive behind adapting this research topic is to enhance data security using encryption algorithms that allow for minimal power consumption, high throughput, and low latency. This includes exploring the field of computation to extract all the benefits we gain from general-purpose computing. Encrypting large files on a CPU tends to take a lot of time as they sequentially perform each calculation, while offloading this computation to a GPU drastically reduces the time taken as it parallelly performs the same calculations [5–10]. This means that many similar calculations are performed concurrently, resulting in a faster result. When GPUs are used to perform general tasks rather than video processing, they are known as general purpose graphical processing units (GPGPUs). GPGPUs are used for tasks that are generally performed by CPUs, such as mathematical equations and cryptography, as well as to create cryptocurrency. These GPGPUs are accessible by making use of parallel platforms like OpenCL or CUDA [10–15]. The proposed project makes use of Compute Unified Device Architecture (CUDA). It is a NVIDIA-exclusive technology that will be available on select NVIDIA compute devices. This compatibility can be checked on NVIDIA's official website.

The proposed study seeks to show the potential speedup and advantage of using a GPU to encrypt files using the AES algorithm. Despite making a significant improvement in performance, this speedup is not directly beneficial to end users. Large corporations can truly harness this power as they have to continuously encrypt a large number of files while being confined by time. As a consequence, end users benefit indirectly since it takes less time to respond to their requests. This technique not only saves a lot of time, but also power if the resources are used efficiently. This will save money not only by lowering electricity consumption but also by lowering the cost of cooling the machines. The applications of using GPUs for general-purpose workloads are limitless; encryption is just one of the many others.

2 Related Work

Survey of related works is shown in Table 1.

3 Proposed Work

(A) AES Algorithm

The AES block cipher works with 128 bits, or 16 bytes, of input data at a time. The substitution-permutation network principle is used in AES, which is an iterative algorithm (Fig. 1). The total number of rounds needed for the encryption or decryption process is determined by the size of the cryptographic key employed. AES's key length and number of rounds is shown in Table 2.

The input is represented as a 4 × 4 bytes grid or matrix in a column major arrangement, in contrast to traditional row major arrangement followed in system programming. The below equation shows the AES 16-byte matrix of 4 rows and 4 columns, which will be mapped as an array for converting plaintext to ciphertext.

$$\begin{bmatrix} b0 & b4 & b8 & b12 \\ b1 & b5 & b9 & b13 \\ b2 & b6 & b10 & b14 \\ b3 & b7 & b11 & b15 \end{bmatrix}$$

Round is composed of several processing steps, including substitution, transposition, and mixing of the input plain text to generate the final output of cipher text. Each round is divided into 4 steps—

(1) SubBytes—Input 16 bytes are substituted by looking up the S-Box.
(2) ShiftRows—Each of the four rows of the matrix is shifted to the left.
(3) MixColumns—Each column of four bytes is transformed using a special mathematical function. This operation is not performed for the last round.
(4) Add Round Key—The 128 bits of the matrix are XORed to the 128 bits of the round key. If the current round is the last, then the output is the ciphertext.

(B) CUDA Implementation

The proposed project consists of a parallel implementation for

- AES-128-bit encryption
- AES-192-bit encryption
- AES-256-bit encryption.

The proposed implementation is developed using Compute Unified Device Architecture (CUDA). It is a parallel computing platform and programming model created by NVIDIA for general computing on NVIDIA GPUs only. CUDA is not just an API,

Table 1 Literature survey

Authors	Year	Parallel platform	Research
Tezcan [16]	2021	CUDA	• Aims to use a single GPU as a cryptographic coprocessor, with optimizations that can help with cryptanalysis • Optimized version of AES providing way faster and higher throughput levels than that of CPU AES instructions • Benchmarked on multiple devices, from one of the entry-level GPUs (NVIDIA GeForce MX 250) to one of the most powerful GPUs (NVIDIA GeForce RTX 2070)
Sanida et al. [17]	2020	OpenCL	• Demonstrate the AES algorithm's implementation in the CTR and XTS parallel operating modes • Portable implementation with key length options of AES-128, AES-192, and AES-256 • File encryption and decryption from range of sizes starting with 512B to 8 MB • Throughput achieved is 12.53 and 14.71 Gbps for XTS and CTR, respectively
Bharadwaj et al. [18]	2021	CUDA	• GPU-accelerated implementation of an image encryption algorithm • To encrypt and decrypt the images, a customized XOR cipher was used with an encryption pad generated using the shared secret key and initialization vectors
Wang and Chu [19]	2019	CUDA	• Benchmarking approach for the GPU-based AES algorithm • Adapts the electronic code book (ECB) cipher mode for cryptographic transformation and the T-box scheme for the purpose of fast lookups • Follows a single thread per state granularity that basically means that every thread is mapped to an AES state for scheduling threads on the GPU • Makes efficient use of GPU hardware in terms of memory allocation and register allocations to increase the overall efficiency and throughput of the operation
Inampudi et al. [20]	2018	OpenCL	• Parallel OpenCL implementation of AES-256-bit algorithm • By using 256 work items, which simultaneously assign the elements to various threads and GPU cores for execution, data parallelism is achieved • Testing done on AMD Radeon 8550 M and 8570G GPU • Compared to sequential implementation, 1024 work items were sped up by 99.8%

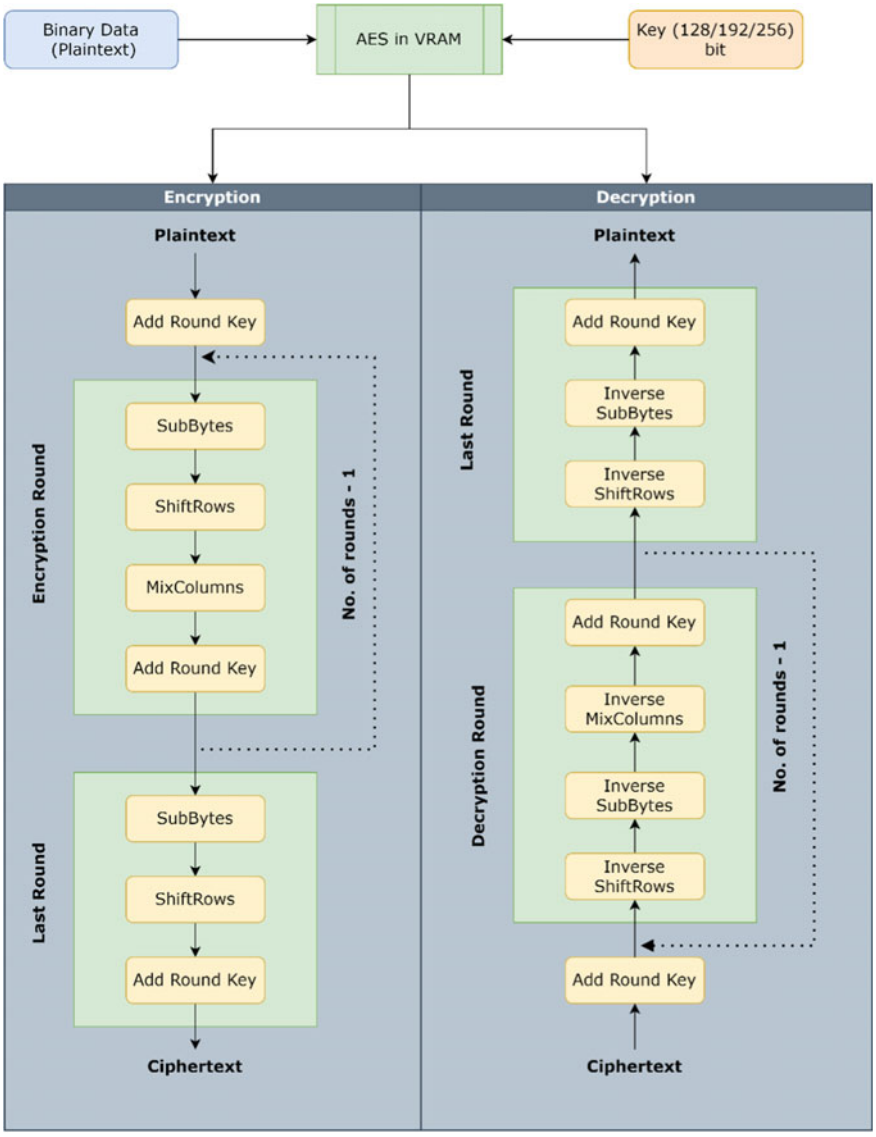

Fig. 1 AES algorithm

Table 2 AES's key length and number of rounds

Key length	Number of rounds
AES-128	10
AES-192	12
AES-256	14

programming language, or SDK. It is mainly a hardware component, allowing drivers and libraries to run on top of it.

The AES algorithm is divided into two parts: one that runs on the CPU and another that runs on the GPU. The calculations performed in AES are performed on the GPU side, and the results are stored back in system memory. The CPU handles reading binary data from images and videos and creating new binary streams after encryption and decryption by the GPU.

Figure 2 presents the NVIDIA CUDA Compiler (NVCC) trajectory. It includes transformations from source to executable that are executed on the compute device. As a result, the.cu source will be divided into host and guest parts and execute on different hardware. This is called hybrid code, which then implements parallel working. We make use of CUDA specifiers like **__global**, which denote that the code runs on the device and is called from the host. The next specifier used is **__device**, which runs and calls on the device itself. Along with this, there is another specifier known as **__host**, which runs and makes calls on host, just like other library APIs or user-defined functions are called.

The key for encryption and decryption will be stored in a text file and can be 128, 192, or 256 bits in length. The binary data, key, and number of threads are passed as command line arguments to the program. The binary data can be a text file, a video, or an image to be encrypted given as a relative path. Number of threads is considered for benchmarking the performance of GPUs to measure their potential. After the execution starts, the data stored in the form of blocks and the key will be copied from system memory, i.e., RAM to the GPU Video RAM (VRAM), using arrays. The operations will be performed based on the number of rounds which is determined by the key length. After encryption and decryption operations are completed in VRAM, the results will be copied back to RAM, and the time required for the calculation will be displayed.

Figure 3 depicts a sample code snippet using CUDA specifiers. It includes the byte substitution process, which replaces the state array with the respective S-Box values and addition of a round key which comprises Binary XOR operation. In this manner, all the AES encryption and decryption operations are implemented as C language functions with modified extensibility using CUDA to implement parallelism.

4 Result Analysis

(A) Evaluation Environment

For the purpose of evaluating the performance of our proposed algorithm, we have used a set of hardware and software components specified in Table 3.

Figure 4 demonstrates the use of our implementation. All of the samples are bitmap images having.bmp file extension. After the program has finished the execution, in the root directory of the application, 2 bitmap images will be generated, **EncryptedImage.bmp** and **DecryptedImage.bmp**. Above figure is a screenshot

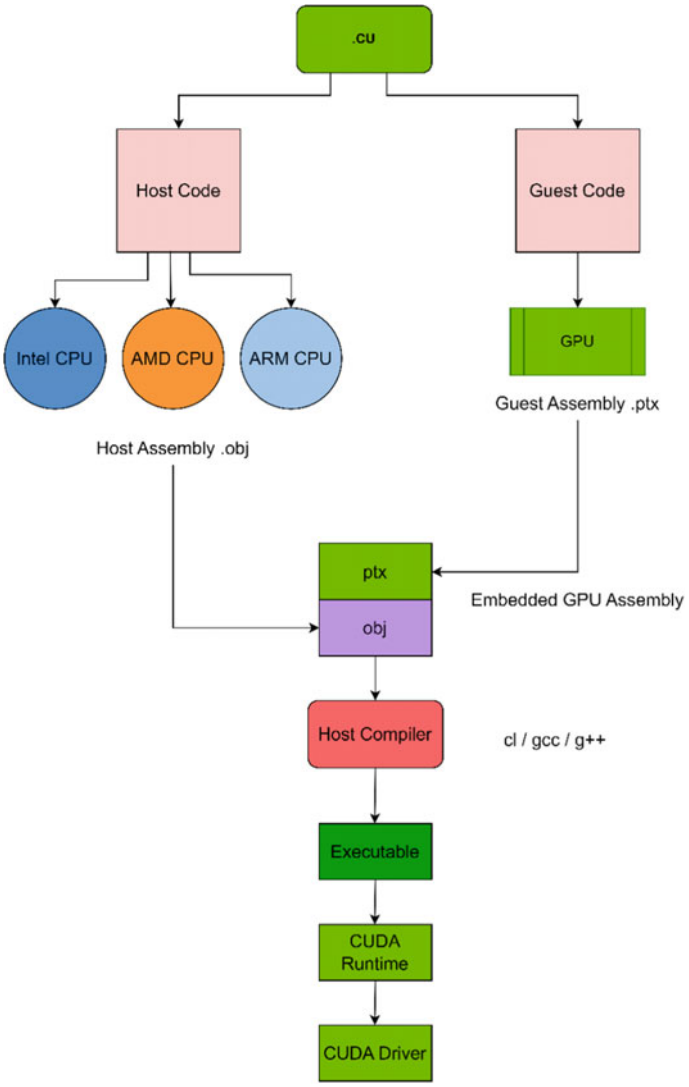

Fig. 2 NVCC trajectory

which combines both the specified files. It has two parts, the left part of the image contains the encrypted file, which the default image application is unable to open, and on the right side you can see the decrypted image.

Figure 5 describes the CUDA information about the computer system you are using to run the program. This uses the APIs from **cuda.h**, which includes **cudaGetDeviceCount()** and **cudaGetDeviceProperties()**. The summary lists out different parameters like—

Fig. 3 Sample code

```
__device__ void sub_bytes(BYTE state[], BYTE sbox[])
{
   for (int i = 0; i < LENGTH; i++)
      state[i] = sbox[state[i]];
}


__device__ void add_round_key(BYTE state[], BYTE round_key[])
{
   for (int i = 0; i < LENGTH; i++)
      state[i] = state[i] ^ round_key[i];
}
```

Table 3 Hardware and software technical specifications

Component	Description	Name
Hardware	CPU	AMD Ryzen 5 5600H
		Intel i7-12700 K
	GPU	NVIDIA GeForce MX 450 Mobile
		NVIDIA GeForce GTX 1650 Mobile
		NVIDIA GeForce GTX 1660 Super
		NVIDIA GeForce RTX 3060
	Operating system	Microsoft Windows 11 64-bit
		Manjaro 22.0 KDE Plasma
		Kubuntu 22.04 LTS
	Drivers	Nvidia Game Ready Driver 528.24
Software	Toolkit	CUDA Toolkit Version 12.0
	IDE/text editor	Microsoft Visual Studio 2022, Microsoft Visual Studio Code
	Compiler	Microsoft CL, GNU GCC
	Library	Helper Timer by Nvidia

(1) Total Number of CUDA Supporting GPU Device/Devices on the System
(2) CUDA Driver and Runtime Information

   a. CUDA Driver Version
   b. CUDA Runtime Version

(3) GPU Device General Information

   a. GPU Device Number
   b. GPU Device Name
   c. GPU Device Compute Capability
   d. GPU Device Clock Rate
   e. GPU Device Type—Integrated or Discrete

(4) GPU Device Memory Information

   a. GPU Device Total Memory

Fig. 4 Image encryption and decryption

```
Developer Command Prompt
CUDA INFORMATION :

**********************************************************************************************
Total Number Of CUDA Supporting GPU Device/Devices On This System : 1

==============================================================================================
***** CUDA DRIVER AND RUNTIME INFORMATION *****
==============================================================================================
CUDA Driver Version                                 : 12.0
CUDA Runtime Version                                : 11.7

==============================================================================================
***** GPU DEVICE GENERAL INFORMATION *****
==============================================================================================
GPU Device Number                                   : 0
GPU Device Name                                     : NVIDIA GeForce GTX 1650
GPU Device Compute Compatibility                    : 7.5
GPU Device Clock Rate                               : 1515000
GPU Device Type                                     : Discrete (Card)

==============================================================================================
***** GPU DEVICE MEMORY INFORMATION *****
==============================================================================================
GPU Device Total Memory                             : 4 GB = 4096 MB = 4294639616 Bytes
GPU Device Constant Memory                          : 65536 Bytes
GPU Device Shared Memory Per SMProcessor            : 49152

==============================================================================================
***** GPU DEVICE MULTIPROCESSOR INFORMATION *****
==============================================================================================
GPU Device Number Of SMProcessors                   : 14
GPU Device Number Of Registers Per SMProcessor      : 65536

==============================================================================================
***** GPU DEVICE THREAD INFORMATION *****
==============================================================================================
GPU Device Maximum Number Of Threads Per SMProcessor    : 1024
GPU Device Maximum Number Of Threads Per Block          : 1024
GPU Device Threads In Warp                              : 32
GPU Device Maximum Thread Dimensions                    : 1024, 1024, 64
GPU Device Maximum Grid Dimensions                      : 2147483647, 65535, 65535
```

Fig. 5 CUDA device properties

b. GPU Device Constant Memory
c. GPU Device Shared Memory per SMProcessor

(5) GPU Device Multiprocessor Information

a. GPU Device Number of SMProcessors
b. GPU Device Number of Registers per SMProcessor

(6) GPU Device Thread Information

a. GPU Device Maximum Number of Threads Per SMProcessor
b. GPU Device Maximum Number of Threads Per Block
c. GPU Device Threads in Warp
d. GPU Device Maximum Thread Dimensions
e. GPU Device Maximum Grid Dimensions

(7) GPU Device Driver Information

a. Error Correcting Code (ECC) Support—Enabled/Disabled
b. GPU Device CUDA Driver Mode—Tesla Compute Cluster(TCC)/Windows Display Driver Model (WDDM).

(B) **Evaluation Result**

While comparing results with existing implementations, the proposed system includes 2 different performance benchmarks, one which compares the performance obtained on different CPUs and GPUs, thus specifying the need to use parallel computing and the second compares compute capability of different GPUs.

The calculation of the time taken for performing operations on the binary data is done using the "**helper_timer**" library offered by NVIDIA. This is achieved using the set of APIs—

a. sdkCreateTimer()—To create a timer pointer of type StopWatchInterface
b. sdkStartTimer()—To start the timer
c. sdkStopTimer()—To stop the timer
d. sdkGetTimerValue()—To get the timer value after the timer is stopped
e. sdkDeleteTimer()—To free the timer pointer.

Figure 6 depicts the program results obtained using 2048 threads. The time required to encrypt and decrypt the images is calculated and displayed in seconds. The number of threads passed to the application is modifiable and is passed as command line arguments to the program.

```
Time To Encrypt Image on NVIDIA GeForce GTX 1650 = 0.002000 seconds

Time To Decrypt Image on NVIDIA GeForce GTX 1650 = 0.005600 seconds
```

Fig. 6 Program execution using 2048 threads

Table 4 CPU and GPU performance comparison

Device	Sample size	Encryption time	Decryption time
AMD Ryzen 5 5600H	800 Kb	0.093	0.149
	3 Mb	0.373	0.588
	7 Mb	0.841	1.312
	50 Mb	6.185	9.847
	100 Mb	11.072	16.378
Intel i7-12700 K	800 Kb	0.071	0.097
	3 Mb	0.274	0.387
	7 Mb	0.598	0.876
	50 Mb	4.319	6.364
	100 Mb	7.542	11.1
Nvidia GeForce GTX 1650	800 Kb	0.0019	0.0053
	3 Mb	0.0018	0.0043
	7 Mb	0.0018	0.0042
	50 Mb	0.0017	0.0039
	100 Mb	0.0016	0.0036
Nvidia GeForce RTX 3060	800 Kb	0.0012	0.0044
	3 Mb	0.0012	0.0036
	7 Mb	0.0011	0.0035
	50 Mb	0.0011	0.0033
	100 Mb	0.0009	0.0029

Table 4 shows the different time values required to perform the encryption and decryption on various CPUs and GPUs.

a. Column 1—Represents the device on which the program is tested.
b. Column 2—Specifies the sample size. Samples are the bitmap images used for testing.
c. Column 3—Time required to encrypt the data, represented in seconds.
d. Column 4—Time required to decrypt the data, represented in seconds.

Figures 7 and 8 portray the time required for encryption and decryption on CPU and GPU for different sample sizes. According to the results specified in Table 4, as the size of input data increases the GPU takes less time to perform AES operations.

Table 5 depicts the different time values required to perform the encryption and decryption on various GPUs with variable threads. First column represents the name of the GPU, second column specifies the number of threads tested on that GPU. The third and fourth columns state the time required for encryption and decryption measured in seconds, respectively. This data is dynamic as the values can change over different runs. But overall, it gives the idea of performance capabilities of different NVIDIA GPUs.

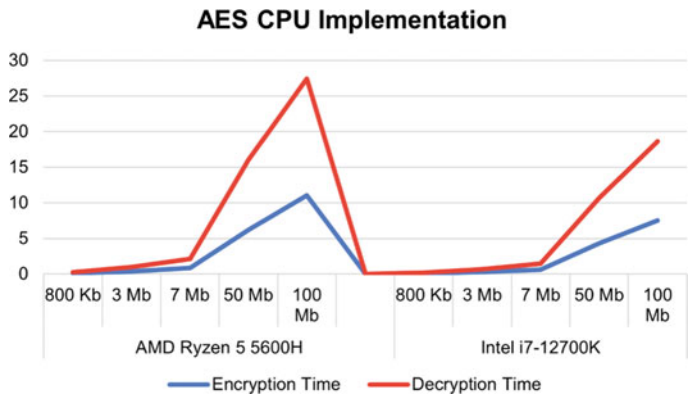

Fig. 7 CPU performance comparison

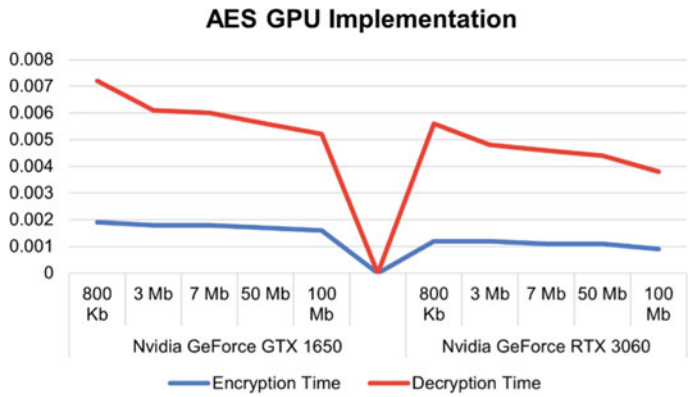

Fig. 8 GPU performance comparison

Table 5 GPU benchmarks

GPU	Threads	Encryption time	Decryption time
MX 450	1024	0.029	0.0284
	2048	0.0022	0.0124
	4096	0.0021	0.0095
GTX 1650	1024	0.0222	0.0122
	2048	0.0024	0.0056
	4096	0.0021	0.0053
GTX 1660 SUPER	1024	0.0145	0.0121
	2048	0.0015	0.0046
	4096	0.0014	0.0045
RTX 3060	1024	0.0115	0.0101
	2048	0.0008	0.003
	4096	0.0007	0.0028

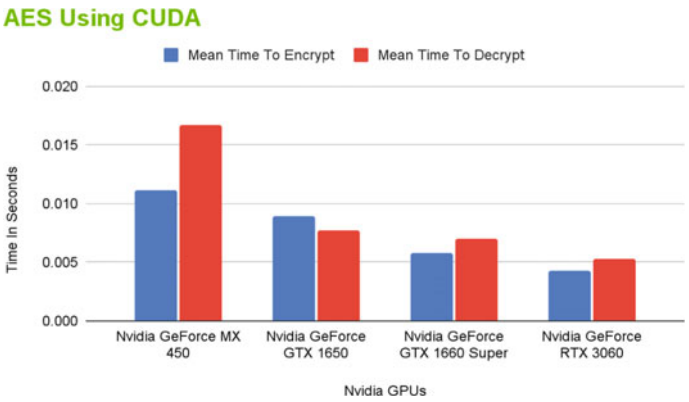

Fig. 9 GPU benchmarks

Figure 9 is used to represent the time and speedup factor by visualizing it. From the results, we can clearly see that, the more powerful the GPU, the less time required to complete the task. Here, the number of threads is also a crucial factor while determining the best GPU. For our testing, NVIDIA RTX 3060 was the best performer.

From the results, we can say that using CUDA extensively saves time and increases the throughput. This can be useful in hash algorithms as well, which can then be implemented in blockchain technology which will compute the hash of the block a lot faster. CUDA can also be used to conserve the amount of energy and power required to maintain the blockchain network. Basically, it will save time, resources and computational cost will be reduced to a great extent.

5 Conclusion

We proposed a method to parallelize the encryption and decryption processes in order to overcome the issue of high resource consumption in the traditional implementation of AES that would run on the CPU. We designed and implemented the AES encryption and decryption algorithm, which works on 128-bit, 192-bit, and 256-bit key sizes, to run on GPUs using CUDA, thereby reducing power consumption and increasing efficiency. This method provided a significant speedup over the CPU, providing high speed. This may change the way traditional resources are used, as these implementations can be used to encrypt binary data in all forms, including images and videos, as well as full disk encryption like Microsoft BitLocker. This process would require extreme fine-tuning to make such implementations a standard for other security techniques.

6 Future Scope

Currently, the proposed system presents a parallel implementation of the AES algorithm that can only be run on NVIDIA GPUs, as the presented research uses CUDA. This limits portability of testing and deploying to infrastructure using AMD or Intel GPUs, may it be integrated or discrete. To overcome this issue, we would need to develop a codebase using OpenCL that would allow us to cover every GPU and CPU device. But there are various parameters that are yet to be considered to optimize the algorithm to make proper and efficient use of the GPU to save energy and still produce similar results.

References

1. Yuan Y, He Z, Gong Z, Qiu W (2014) Acceleration of AES encryption with OpenCL. In: 2014 Ninth Asia joint conference on information security, pp 64–70
2. Jaiswal M, Kumari R, Singh I (2018) Analysis and implementation of parallel AES algorithm based on T-table using CUDA on the multicore GPU. IJCRT 6(1). ISSN: 2320-2882
3. Ma J, Chen X, Xu R, Shi J (2017) Implementation and evaluation of different parallel designs of AES using CUDA. In: 2017 IEEE second international conference on data science in cyberspace (DSC), pp 606–614
4. Abdelrahman AA, Fouad MM, Dahshan H, Mousa AM (2017) High performance CUDA AES implementation: a quantitative performance analysis approach. In: 2017 Computing conference, pp 1077–1085
5. An S, Seo SC (2020) Highly efficient implementation of block ciphers on graphic processing units for massively large data. NATO Adv Sci Inst Ser E Appl Sci 10(11):3711
6. Biryukov A, Großschädl J (2012) Cryptanalysis of the full AES using GPU-like special-purpose hardware. Fund Inform 114(3–4):221–237
7. Li Q, Zhong C, Zhao K, Mei X, Chu X (2012) Implementation and analysis of AES encryption on GPU. In: 2012 IEEE 14th international conference on high performance computing and communication & 2012 IEEE 9th international conference on embedded software and systems, pp 843–848
8. Iwai K, Kurokawa T, Nisikawa N (2010) AES encryption implementation on CUDA GPU and its analysis. In: 2010 First international conference on networking and computing, pp 209–214
9. Mei C, Jiang H, Jenness J (2010) CUDA-based AES parallelization with fine-tuned GPU memory utilization. In: 2010 IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW), pp 1–7
10. Jadhav S, Vanjale SB, Mane PB (2014) Illegal access point detection using clock skews method in wireless LAN. In: 2014 International conference on computing for sustainable global development (INDIACom). https://doi.org/10.1109/indiacom.2014.6828057
11. Stallings W (2022) Cryptography and network security: principles and practice. Pearson.
12. Daemen J, Rijmen V (2013) The design of Rijndael: AES—the advanced encryption standard. Springer Science & Business Media
13. Sanders J, Kandrot E (2010) CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional
14. Wilt N (2020) The Cuda handbook: a comprehensive guide to Gpu Programming. Addison-Wesley Professional
15. Nguyen H, NVIDIA Corporation (2008) GPU gems 3. Addison-Wesley. https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-36-aes-encryption-and-decryption-gpu

16. Tezcan C (2021) Optimization of advanced encryption standard on graphics processing units. IEEE Access 9:67315–67326
17. Sanida T, Sideris A, Dasygenis M (2020) Accelerating the AES algorithm using OpenCL. In: 2020 9th International conference on modern circuits and systems technologies (MOCAST), pp 1–4
18. Bharadwaj B, Saira Banu J, Madiajagan M, Ghalib MR, Castillo O, Shankar A (2021) GPU-accelerated implementation of a genetically optimized image encryption algorithm. Soft Comput 25(22):14413–14428
19. Wang C, Chu X (2019) GPU accelerated AES algorithm. arXiv [cs.DC]. arXiv. http://arxiv.org/abs/1902.05234
20. Inampudi GR, Shyamala K, Ramachandram S (2018) Parallel implementation of cryptographic algorithm: AES using OpenCL on GPUs. In: 2018 2nd International conference on inventive systems and control (ICISC), pp 984–988