Parallel AES Encryption & Decryption

Authors: Syed Shaheer Sabri(24522) & Muhammad Danish Raza(24796)

Institute of business administration

Course Number: PDC - 96002

Contents

[INTRODUCTION 3](#_Toc163571279)

[METHODOLOGY 5](#_Toc163571280)

[Unlocking the Power of Parallelism: Optimizing AES Encryption 7](#_Toc163571281)

[The Essence of Parallel Processing 7](#_Toc163571282)

[Why Parallelism Matters for AES 7](#_Toc163571283)

[Implementation Approaches 8](#_Toc163571284)

[The Role of Developers 8](#_Toc163571285)

[Parallel AES Strategies in Python 9](#_Toc163571286)

[Challenges and Future Directions 9](#_Toc163571287)

[References: 10](#_Toc163571288)

# INTRODUCTION

Information has become such a vital element in our business, research, and everyday life that security of data transmission is paramount. The situation is even more complicated now because of a constantly growing number of devices and related data-transmission ways that make even more hidden ways to steal private intel via the Internet, simply because it becomes more and more intricate to intercept information. Obviously, in such a Catch-22 situation, encryption and decryption processes have become of the utmost importance: only those who know the way how original data were encoded (transformed into some indecipherable form) can decode and change them. Ironically, having more information nearly always equals having more problems: whenever this system becomes bigger and more complicated, it gets more difficult both from the security point of view and in terms of its speed of operation.

This is where our project is trying to make a difference – the whole point of it is to provide more safe and secure data, but in a way that also speeds it up and standardises it. And the big puzzle piece here is something called parallel processing. Where before you were doing anything one piece at a time, you can now do a whole bunch of pieces at once to make things speed up without revealing anything. We study how parallel processing compares with the traditional way of doing things: processing one data item at a time. As a case study, we consider the most widely used encryption method, called the Advanced Encryption Standard or AES. It’s a method that’s secure and fast. With this, we show how to simulate what AES does with parallel processing, in a way that leads to significant speedups. All data exchange of information users and providers, regardless of the industry, require strong cyber security.

Banks need security to protect the financial service industry; governments secure classified data to

ensure safety for their citizens. Data security is a fundamental requirement for any operation.

However, our current methods of using complicated encryption and computation-heavy decryption

procedures cannot keep up with the current volume of data. Furthermore, such sequential processes

are glacial and clunky. Parallel processing may be the answer to these limitations. It could help to solve these issues multiple tasks concurrently, it allows for much faster processing times without sacrificing security. Our

project is about comparing two ways of processing data: parallel processing and traditional serial

processing. We're focusing on the Advanced Encryption Standard (AES), a commonly used encryption

technique that is strong and efficient. We program in Python, which is a great programming language.

In terms of simulating what AES does to encrypt, this sort of work suits it because it can do several things at once like decrypting data.

We’re demonstrating this with our simulations to show that parallel processing can work better for cryptographic things, such as speeding up both encryption and decryption. We hope that our project will demonstrate the potential for parallel processing to fundamentally change how we secure data.

All we have looked at is the AES and a few other parallel processing algorithms, so we acknowledge some of our methods have limitations. Even so, we believe these results can still be a good guide as to how parallel processing can improve the efficiency and safety of data processing.

# METHODOLOGY

**Explanation of The Code**

The process revolves around the standard method of dividing the input data into smaller chunks, processing these chunks in parallel using threads and then combining the results together.

**Data Chunking:**

* + The input data is divided into chunks, ensuring each chunk's size is suitable for AES encryption.
  + The chunk size is calculated by dividing total length of data by the number of threads.

**Parallel Encryption:**

* + Each data chunk is encrypted independently in a separate thread.
  + The encrypt\_chunk function handles the encryption process for each chunk, using AES in CBC mode.
  + The function uses padding to ensure the data fits the AES block size, encrypts the chunk, and then stores the result along with the initialization vector (IV) in a thread-safe queue.
  + The function makes sure that the data fits the AES block size by using padding. It also encrypts the chunk and then stores the initialization vector as well.
  + Locks are used to synchronize the threads.

**Parallel Decryption:**

* + Using separate threads, each encrypted chunk is decrypted in parallel.
  + The decrypt\_chunk function decrypts each chunk, removes the padding, and stores the result in a thread-safe queue.
  + The decrypted chunks are then combined to reconstruct the original data.

**Key Functions and Their Purposes**

*encrypt\_chunk(data\_chunk, key, queue, lock, index):*

* + Encrypts a single chunk of data using AES in CBC mode.
  + Adds the encrypted data and IV to a thread-safe queue.

*decrypt\_chunk(encrypted\_chunk, iv, key, queue, lock, index):*

* + Decrypts a single chunk of encrypted data.
  + Adds the decrypted data to a thread-safe queue.

*parallel\_encrypt(data, key, num\_threads):*

* + Manages the encryption process by dividing the data into chunks and starting threads for parallel encryption.
  + Collects and sorts the encrypted chunks based on their index to maintain order.

*parallel\_decrypt(encrypted\_data, key, num\_threads):*

* + Manages the decryption process by starting threads for parallel decryption of each encrypted chunk.
  + Collects and sorts the decrypted chunks to reconstruct the original data.

**Algorithms and Techniques Used**

The project utilizes several algorithms and techniques to achieve efficient and secure data processing. AES encryption in CBC mode is employed, which requires an initialization vector (IV) for each encryption operation to ensure that identical plaintext blocks produce different ciphertext blocks, enhancing security. Padding and unpadding techniques are used to align data chunks with the AES block size. Multithreading leverages Python's **threading** module to create and manage multiple threads, enabling parallel processing and taking advantage of multiple CPU cores for improved performance. Thread-safe queues and locks from the **queue** and **threading** modules are used to synchronize threads and prevent race conditions when accessing shared resources. Overall, this combination of cryptographic principles and parallel computing techniques facilitates fast and secure encryption and decryption of large datasets.

**Flow Diagram:**

A diagram of a diagram

Description automatically generated

# Unlocking the Power of Parallelism: Optimizing AES Encryption

## If your data is well-protected, then you are lying to yourself. No encryption system except, at this point, AES, has been yet broken Despite its impressive reputation, AES’s current drawback is that it remains a single-core system, and encrypting a large data stream takes time. As the internet continues to expand and every bit and byte suddenly becomes valuable, the need for faster and more efficient encryption becomes more pressing. This is where the technique of parallel processing could be a gamechanger for AE\*S systems.

## T**he Essence of Parallel Processing**

## At bottom, parallel processing represents a computational approach to processing multiple parts of a given task in parallel (ie, at the same time) using multiple processing units (‘CPUs’ or ‘cores’) rather than the traditional approach allowing just one CPU or core. For example, imagine an encrypted mail communication – if multiple builders, not just one bricklayer, could work on your new home, your job would get done significantly faster The complex encryption tasks of AES can be divided into several smaller, decoupled sub-tasks. Each sub-task can then be assigned to a separate, dedicated digital processor for execution in parallel.

## Why Parallelism Matters for AES

AES, while remarkably secure, is computationally intensive. Its encryption and decryption processes involve a series of iterative rounds comprising operations like substitution, permutation, and key mixing. Single-processor AES can be a bottleneck when it comes to high-bandwidth data or applications requiring round-the-clock encryption (think video streaming).

Parallel processing brings several key benefits to AES:

• **Faster throughput**: If you divide the job of encryption or decryption into many tasks that a single processor can run simultaneously, much more data can be processed in a given time period. You can move more bits in a given amount of time (faster file transfers), watch more bits of an encrypted video (smoother streaming), or process even more of those big piles of sensor data (faster analysis).

• **Scalability:** A parallel version of AES can scale to different computational resources. If you have more processors (or cores) available, you can have more cores doing encryption or decryption – and you can ask more of fewer cores as hardware availability changes.

• **Better utilization of resources:** Parallel AES can help deal with the situation when there are many streams of data requiring encryption in a single process, by allowing these tasks to be efficiently distributed across different hardware tools; it prevent wastage of data streams lying idle when processors are committed unfettered to the same function.

## Real-World Parallelization of AES

Studies and practical implementations demonstrate the transformative potential of parallel processing in AES:

• **Cloud Storage**: Shi, Wang, & Sun (2020) using highly parallelised AES algorithms show efficacy in large-scale datasets, querying and encrypting/decrypting the massive cloud data efficiently.

• **Many-Core Processors** Xing et al (2021) developed methods to speed up AES on heterogeneous many-core processors. These are processors with different core designs on the same chip. Splitting and distributing individual operations of AES to different cores substantially improved performance. This is a good example of how a parallel implementation needs to be designed for the target hardware architecture.

# Implementation Approaches

There are several ways to implement parallel processing in AES encryption and decryption:

• **Task-level Parallelism:** Here, the AES algorithm is broken down into more coarse-grained sub-tasks – such as individual encryption rounds or the encryption of independent data blocks, and these tasks are assigned to different processors or cores and executed concurrently.

• **Instruction-level Parallelism:** Instruction-level parallelism focuses on identifying sub-instructions of fine granularity inside the AES algorithm that are independent from each other, in the sense that they can be executed on multiple processing units in parallel. Many of today’s processors have instruction-level parallelism capabilities, using technologies such as SIMD (Single Instruction, Multiple Data).

• **Special hardware**: Over recent decades, hardware such as GPUs (Grpahics Processor Units) and FPGAs (Field-Programmable Gate Arrays) has become increasingly optimised for the task of parallel computation. The most efficient AES circuits can be ported to such chips, potentially yielding substantial further performance benefits.

## The Role of Developers

Resources such as the Secure Programming Cookbook for C and C++ can be of great help to developers in designing effective parallel AES implementations. Some of the critical considerations are as follows:

• **Mode:** AES supports several modes of operation (eg, CBC, CTR), some of which are naturally more parallelizable than others.

• **Programming Models** Libraries such as OpenMP and MPI provide programming abstractions for task distribution and communication between processors.

• **Synchronization:** parallel processes must make sure data is synchronizssed for AES encryption and decryption to work properly.

# Parallel AES Strategies in Python

Python offers two core strategies for parallelizing AES encryption or decryption operations: **multiprocessing** and **threading**. Multiprocessing leverages multiple processes for true parallelism, making it optimal for CPU-intensive AES tasks across multiple cores. In contrast, threading uses lightweight threads within a single process, better suited for situations where AES operations have I/O bottlenecks or require fine-grained parallelism. Be aware that Python's Global Interpreter Lock (GIL) may hinder the performance gains of threading for purely CPU-bound tasks.

## Challenges and Future Directions

While immensely promising, optimizing AES through parallel processing comes with its own set of challenges:

* **Overhead:** Distributing tasks, coordinating communication, and synchronizing processes introduces overhead that can potentially diminish gains.
* **Balancing Complexity:** Striking the right balance between parallelism granularity and the complexity of managing multiple threads is essential.
* **Hardware Diversity** Adapting parallel AES implementations to the range of available processors and architectures requires careful optimization.

# Results

**Runtimes of sequential vs parallel:**

**Encryption:**

**Decryption:**

As shown in the graph above we don’t see the advantages of the parallel implementation on small data lengths however we do see the advantages after the data length is increased.

**Conclusion**

The project successfully demonstrated the effectiveness of using multithreading for parallel AES encryption and decryption. By dividing data into chunks and processing them simultaneously, the project achieved faster encryption and decryption times compared to a sequential approach. The results align well with the project objectives of improving performance and maintaining data security.

**Future Work**

Further research could explore optimizing the chunk size and thread count for even better performance. Additionally, the project could be expanded to support other encryption algorithms and modes, or to include more sophisticated error handling and logging mechanisms. Integrating this approach into real-world applications would also be a valuable next step.

# References

*Advanced Encryption Standard*. (n.d.). Retrieved from WikiPedia: https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard

*Advanced Encryption Standard (AES).* (n.d.). Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

*AES Encryption and Decryption Online*. (n.d.). Retrieved from Devglan: https://www.devglan.com/online-tools/aes-encryption-decryption

Aj Elbirt, W. Y. (2000). An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists.

Alfred J. Menezes, P. C. (1996). *Handbook of Applied Cryptography.* CRC Press.

Ferguson, N. (2010). *Cryptography Engineering: Design Principles and Practical Applications.* Wiley.

Ghada F. Elkabbany, H. K. (2015). *A Design of a Fast Parallel-Pipelined Implementation of AES: Advanced Encryption Standard.*

Jingang Shi, S. W. (2020). *A Parallel AES Encryption Algorithms and Its Application.* Retrieved from researchgate: https://www.researchgate.net/publication/338542389\_A\_Parallel\_AES\_Encryption\_Algorithms\_and\_Its\_Application

Joan Daemen, V. R. (2002). *The Design of Rijndael.* Springer Berlin, Heidelberg.

Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice, 8th edition.* Pearson.

Viega, J. (2003). *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More.* O'Reilly Media.

Yongquan Yang, D. W. (2021, June). *Accelerating DES and AES Algorithms for a Heterogeneous Many-core Processor.* Retrieved from researchgate: https://www.researchgate.net/publication/350937834\_Accelerating\_DES\_and\_AES\_Algorithms\_for\_a\_Heterogeneous\_Many-core\_Processor