Symmetric Encryption Showdown:

AES & DES UNDER UNIFIED KEY CONDITIONS

Diego Acosta Cantu

Cybersecurity Portfolio

Table of Contents

Introduction

As digital threats grow more advanced, it’s become more important than ever to understand how encryption keeps our data safe. Symmetric encryption algorithms, like AES and DES, are core tools for securing information, but they’re not all created equal. By comparing these algorithms, we can better understand their strengths, their limitations, and when it makes sense to use one over the other.

I started this project to get hands-on experience with how encryption actually works in practice. Instead of just reading about AES or DES in a textbook, I wanted to build, test, and compare them myself. This way, I can deepen my understanding of how encryption is used to protect data in the real world, and what makes some methods more secure or efficient than others..

Prerequisites

This document assumes you have a basic understanding of cybersecurity, how encryption works, and some experience with Python. The focus is on comparing and working with symmetric encryption algorithms in practice, rather than diving deep into the underlying cryptographic theory.

Project Objectives

The objectives of this project are as follows:

1. Implement Symmetric Encryption Algorithms: Develop working implementations of Advanced Encryption Standard (AES) and Data Encryption Standard (DES) using Python and the PyCryptodome library to demonstrate real-world encryption processes.

2. Apply Consistent Key Management Practices: Use a shared key generation approach to ensure both algorithms operate under the same key material, enabling an accurate and fair comparison between encryption outputs and performance.

3. Analyze Algorithm Performance and Efficiency: Measure encryption and decryption speed, memory usage, and CPU impact for both AES and DES to evaluate their computational efficiency under similar conditions.

4. Evaluate Security Strengths and Weaknesses: Compare cryptographic characteristics of AES and DES, including key size, block size, resistance to brute-force attacks, and vulnerability to modern cryptanalysis.

5. Document Practical Implications for Real-World Use: Summarize findings and provide guidance on the appropriate use cases for each algorithm, highlighting the risks of using outdated ciphers like DES in modern security environments.

What is Symmetric Encryption?

First of all, we need to understand what symmetric encryption is, and why it matters. Symmetric encryption is a way to protect information by using a single secret key to both encrypt and decrypt data.

It is important because it helps keep sensitive data safe from prying eyes, ensuring that only trusted parties can read it. However, if the key falls into the wrong hands, the entire system can be compromised, so protecting the key is just as important as protecting the data itself.

Why is the Key Size Important?

One of the factors that differentiate each encryption algorithm is the key size, and it is the one of the things that makes the algorithm more strong and secure, but why? It directly impacts encryption strength because it determines the number of possible key combinations an attacker must try in a brute-force attack. A larger key size increases the search space exponentially, making it much harder for an attacker to crack the encryption within a feasible time frame.

In more technical terms, a 56-bit key offers 2^56 possible combinations, while a 256-bit key offers 2^256 combinations, vastly more, which makes brute-forcing practically impossible with current computing power. Choosing the right key size ensures that encryption remains secure against evolving threats.

What is Bit Permutation?

Bit permutation is part of the things that makes encryption algorithms strong and secure. It is a process that rearranges the bits of data in a specific order. It is often used in cryptographic algorithms to increase complexity and make patterns harder to predict. By shuffling the bits, the original data is transformed into a new form, adding another layer of security. This helps prevent attackers from easily analyzing or detecting relationships between the input and output, enhancing the strength of encryption. And how is Bit Permutation achieved? By appliying Diffusiona and Confusion.

What is Difussion and Confussion?

Confusion refers to making the relationship between the plaintext and ciphertext as complex as possible. This hides any patterns in the data.

Diffusion involves spreading the influence of each bit of the plaintext across many bits of the ciphertext, making it harder to identify any correlation between the input and output.

Why Use A Shared Key for Both Algorithms?

We'll use the same base key for both AES and DES to keep things fair when comparing them. Even though they need different key sizes, we'll derive both from the same starting point, like a passphrase. This way, we can focus on how the algorithms themselves perform and what their strengths and weaknesses are..

Script Overview

For this project, the main library I used was ***pycryptodome***, which is a self-contained Python package of low-level cryptographic primitives. Specifically, I used the ***Crypto.Cipher***, ***Crypto.Util.Padding***, and ***Crypto.Random*** modules, since within them one can use the DES, AES, and padding utilities.

On the other hand, I used the ***time*** library to measure how long each algorithm takes to encrypt the message, and the ***psutil*** library to measure RAM memory usage.

I divided the project into five modules: the first two are for each of the algorithm's functions. The other two are where I measured each algorithm's diffusion rate. The last one is main.py, which brings everything together and prints the results so they can be compared.

AES and DES Algorithm Modules

I first started by adding a global variable with the key that both algorithms were going to share, in this case, 32 bytes. Then, within each algorithm's function, I trimmed the base key to match their respective key lengths: 16 bytes for AES and 8 bytes for DES.

A timer is started to measure how long the encryption process takes. I then created a variable to define the cipher object using its key and CBC mode, which, by the way, is more secure than ECB mode. Right after that, I added the variable that encrypts and pads the message using AES.block_size, which ensures the message is properly padded to match AES's required block size.

After creating the cipher object and encrypting the message, the timer stops, and the Initialization Vector (IV) is saved in its own variable for use in the decryption function. Finally, the encrypted ciphertext is printed in a readable format.

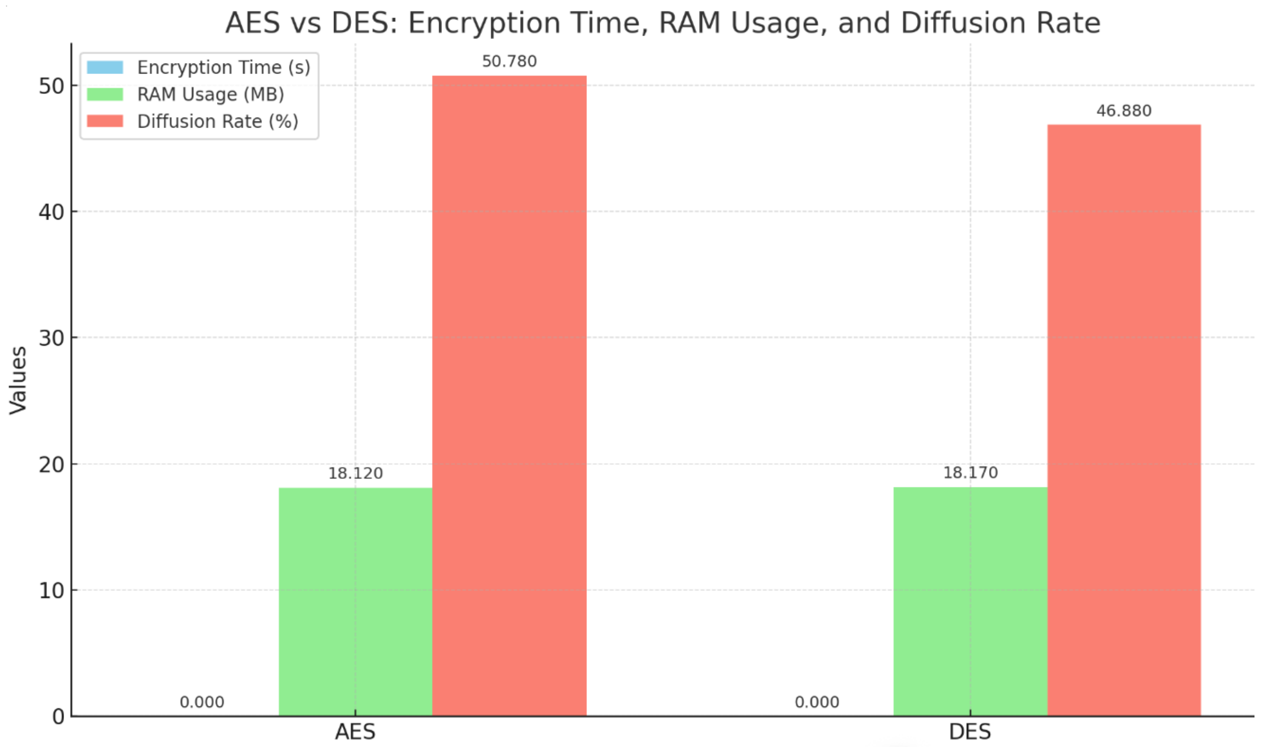
The decryption function is almost identical, I just added the Initialization Vector to the decryption step, along with the base key and block size specification. It prints the decrypted plaintext at the end. I also used the .process() module and process.memory_info().rss to show how much RAM (in MB) the script is using at that moment.

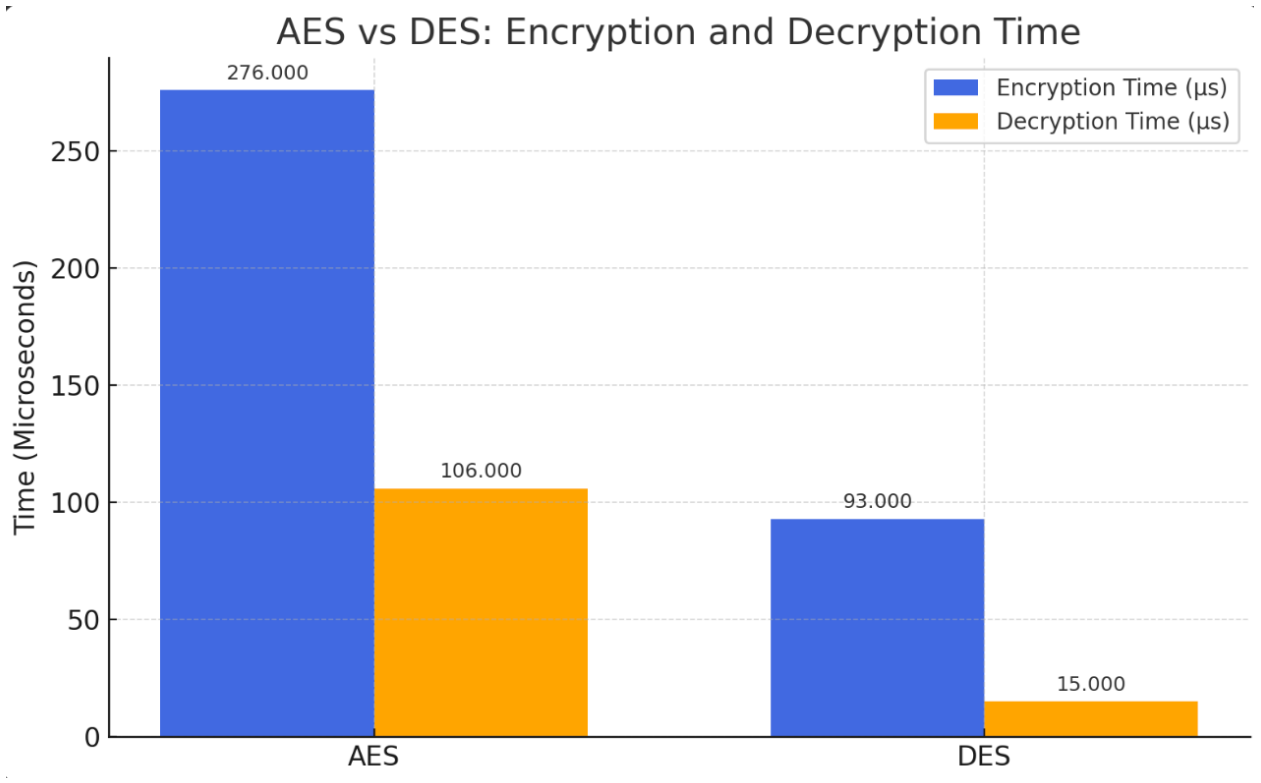
AES and DES Diffusion Rate

For the diffusion rate, I added a function named calculate_diffusion_rate(). This function takes two slightly different messages and uses the same randomly generated 16-byte (or 8-byte) key and 16-byte (or 8-byte) IV for both. It then creates two AES/DES cipher objects using CBC mode and encrypts both messages.

After encryption, I compare the resulting ciphertexts byte by byte using an XOR operation to count how many individual bits differ. This gives a clear measurement of the diffusion rate, expressed as a percentage of differing bits compared to the total number of bits. At the end, it prints the diffusion rate, as well as both ciphertexts in hexadecimal format so I can visually inspect how much the output changes, even with a small input difference.

Result Analysis





When comparing AES and DES, it's clear that each has its strengths. DES is noticeably faster when it comes to both encryption and decryption, which makes it a good choice for systems where speed matters more than security. On the other hand, AES, while a bit slower, provides stronger security thanks to its better diffusion and longer key size. Both use about the same amount of RAM, so memory usage isn't really a deciding factor.

In the end, if someone prefers speed and working in a low-risk environment, DES coukld be an option but the thing is that nowadays, with the technology we have right now it isn't feasable to use DES since it is super insecure and it can be cracked in less than a few seconds, so for anything that requires strong, modern encryption, AES is the better and safer option. Besides,

Conclusion

This project really showed how important encryption is when it comes to keeping data safe. By testing and comparing AES and DES, I got to see how each algorithm handles speed, memory usage, and how well they scramble data (diffusion). DES turned out to be faster, but AES clearly offered stronger security thanks to its more complex structure and better diffusion.

In the real world, that extra security can make a huge difference, whether it’s protecting personal messages or securing sensitive information in banks or hospitals. At the end of the day, knowing how and when to use the right encryption method is key to building safer systems.